Research Subject

Dynamics Computation and Behavior Capture of Human Figures (Nakamura Group)

(1) Goal and summary

Introduction Humanoid has less actuators than its movable degrees of freedom (DOF) which includes the unactuated six DOF of the translation and rotation of the pelvis. Therefore, we may not be able to find a sequence of actuator inputs to achieve a motion generated without considering the dynamics. In addition, it is very difficult to adapt a motion to various situations because common humanoids have more than 20 DOF and practical motion generation techniques are limited to motion capture or numerical optimization.

In order to study brain-like information processing, it is important to measure or compute the sensor information such as vision and somatosensory information, as well as the motion data. Commercial motion capture systems can only capture the motion of subjects which are typically modeled as kinematic chains with similar complexity as humanoids. In addition, it is very difficult to add a new hardware or improve the software of commercial systems.

In this research, we developed the following methods and systems:

- Parallel efficient dynamics computation of human figures: This method not only serves as the basis for motion generation considering dynamics, but also improves the efficiency of the computations of simulating and controlling motions of humanoids.
- Motion generation of human figures considering physical consistency: This method, called dynamics filter, can generated motions that are both physically consistent and human-like by modifying motion capture data.
- Intuitive motion generation using inverse kinematics: This method is capable of generating whole-body motions of human figures by only specifying several fixed links and the trajectory of a link. This is enabled by extending conventional algorithm for inverse kinematics.
- Behavior capture system: We combined our original motion capture system with other sensors including force plate and gaze direction sensor.
- Dynamics computation of musculoskeletal human model: We can compute the somatosensory information by developing the methods for computing the dynamics of human model composed of bones, muscles, and tendons.

The following sections briefly describes the result of our research in the above topics.

Parallel Dynamics Computation of Human Figures The dynamics computation of kinematic chains has been studied by many researchers. However, realtime dynamics computation of human figures is still a challenging problem because they usually contain 20 to 50 DOF. In this research, we developed an algorithm which takes only O(logN) CPU time where N is the number of links.

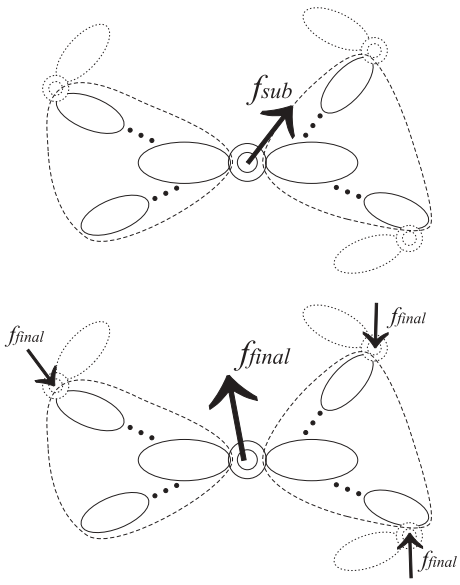

Figure 1: Concept of steps 1.(above) and 2.(below)

Summary of the algorithm The algorithm consists of the following four steps:

1. Assuming that the links are not constrained at all, add a joint one by one and compute the constraint force at each new joint, $\boldsymbol{f}_{sub}$, applying the Principle of Virtual Work (see 1 above). The intermediate chains are called subchains. Note that $\boldsymbol{f}_{sub}$ is the constraint force in the subchain, not in the complete chain.

2. Remove every joint in the reverse order of step 1. and compute the final constraint force $\boldsymbol{f}_{final}$. Completing this step gives all the joint forces.

3. Compute the acceleration of each link.

4. Compute the joint accelerations using those of the neighboring links.

Note that, in steps 1 and 2, adding or removing a joint to assemble or disassemble subchains with no kinematic connections can be processed in parallel. By utilizing parallel computation, the total computation time can be reduced to O(logN) where N is the number of links.

Optimizing the schedule The order of adding and removing joints in steps 1. and 2. is described by a binary as shown in Figure 2(a)–(c). Each node represents the subchain including all the successor joints. Node 3 in Figure 2(b), for example, represents the subchain with joints 1 and 3. The whole chain will be completed by connecting subchains 3 and 2 through joint 4.

For a given binary tree, step 1. is executed from the leaf nodes to the root, while step 2. is executed from the root to the leaves. A binary tree gives an intuitive idea of the parallelism and the total computational cost of the schedule as follows:

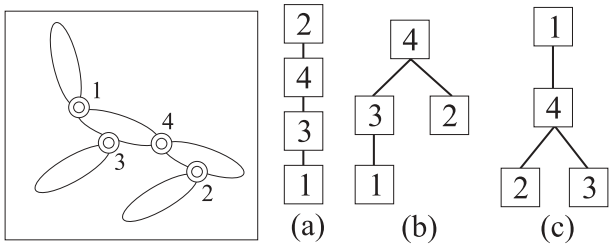

Figure 2: A binary tree to describe the schedule

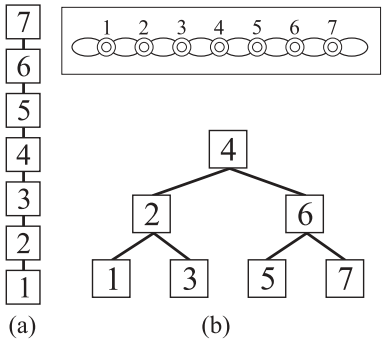

Figure 3: Binary trees for a 7-joint serial chain optimized for (a) serial computation, (b) parallel computation

- The width of the tree indicates the total computational cost. Optimizing the tree in terms of the total computational cost yields the tree shown in Figure 2(a) which has the least number of branches. This is the optimal schedule for serial computation.

- The height of the tree indicates the computation time when enough number of processors exist, or in other words, the parallelism of the schedule. The trees in Figure 2(b) and (c) takes approximately the same time when they are executed on two processors. The total computational cost, on the other hand, is larger than the schedule in Figure 2(a).

Figure 2 shows two trees optimized for (a) serial computation and (b) parallel computation on four processors.

Simulation examples We implemented the algorithm on an 8-node cluster. Each node has a PentiumIII 1GHz processor and connected to the network via Myrinet. Table 1 shows the computation time for serial chains with 8 to 32 links on 1 to 8 processes.

Figure 4 shows the motion obtained by simulating the motion of a 40 DOF human figure, which initially holds the bar by both hands and later releases the right hand. The computation time for serial computation was 5.1 ms without connection, 6.3 ms with one connection, and 7.2 ms with two connections. Figure 5 shows the schedule optimized for serial computation when only one of the hands holds the bar. This schedule still has large parallelism because of the branched structure of the human figure.

Table 1: Computation time for serial chains (ms)

links	8	16	32
1 procs	1.31	2.75	6.08
2 procs	0.984	1.87	3.93
4 procs	0.897	1.70	3.39
8 procs	—	1.57	2.90
4 procs*	—	1.58	3.16

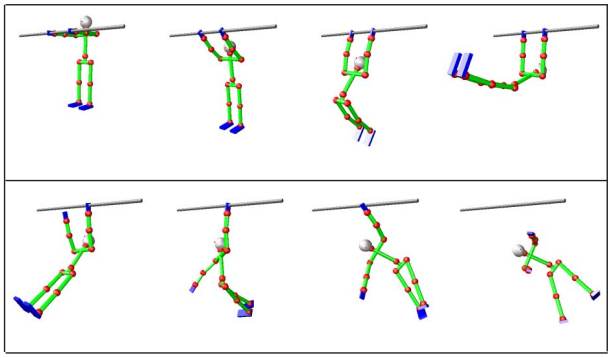

Figure 4: Simulation of a 40 DOF human figure

Motion generation of human figures considering physical consistency

Dynamics filter The dynamics filter is a filter that converts a physically inconsistent motion into a consistent one by slightly modifying the original. A captured motion is physically consistent for the human subject, but may be inconsistent for a humanoid. Kinematically editing several motion sequences is also likely to yield physically inconsistent motions. We would be able to reuse existing motion data if we could modify physically inconsistent motions such that the new motion becomes consistent.

Previous work on the dynamics filter tends to use global optimization techniques which do not allow interactions with the user. In real environment, however, the motion has to be generated in real time rather than planned in advance. The dynamics filter we developed only uses efficient local optimization, and therefore allows realtime input of the reference motion to maintain high interactivity.

Equation of motion of constrained systems Humanoids are often subject to external constraints from the environment. Their equation of motion is described as follows:

$$\begin{pmatrix} \boldsymbol{A} & -\hat{\boldsymbol{H}}_C^T & -\boldsymbol{H}_J^T \\ \boldsymbol{H}_C & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \ddot{\boldsymbol{\theta}}_G \\ \boldsymbol{\tau}_C \\ \boldsymbol{\tau}_J \end{pmatrix} = \begin{pmatrix} -\boldsymbol{b} \\ -\dot{\boldsymbol{H}}_C \dot{\boldsymbol{\theta}}_G \end{pmatrix} \tag{1}$$

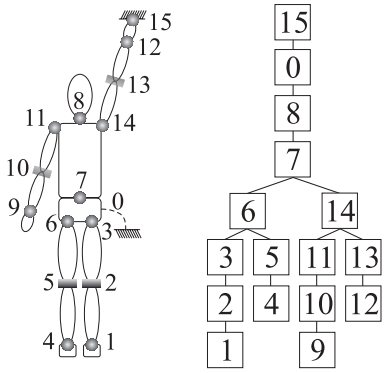

Figure 5: The schedule optimized for the human figure in Figure 4

where

$\boldsymbol{A}$: joint-space inertia matrix
$\boldsymbol{b}$: gravity, centrifugal, and Coriolis forces
$\boldsymbol{\theta}_G$: generalized coordinates
$\boldsymbol{\tau}_J$: joint torques
$\boldsymbol{\tau}_C$: constraint forces
$\boldsymbol{H}_J = \partial \boldsymbol{\theta}_J / \partial \boldsymbol{\theta}_G$
$\boldsymbol{H}_C = \partial \boldsymbol{\theta}_C / \partial \boldsymbol{\theta}_G$
$\boldsymbol{\theta}_J$: joint values of the actuated joints
$\boldsymbol{\theta}_C$: position and orientation of the constraint

If we know the joint torques $\boldsymbol{\tau}_J$, Eq.(1) has a unique solution and we can compute the generalized accelerations for dynamics simulation. For the dynamics filter, in contrast, we treat the equation as a redundant equation in generalized accelerations $\ddot{\boldsymbol{\theta}}_G$, joint torques $\boldsymbol{\tau}_J$, and constraint forces $\boldsymbol{\tau}_C$ whose solutions include all the physically feasible motions.

Implementation of the dynamics filter The accelerations in the original motion data $\ddot{\boldsymbol{\theta}}_G^d$ do not always satisfy Eq.(1). We introduce the error term $\Delta\ddot{\boldsymbol{\theta}}_G$ so that the modified acceleration $\ddot{\boldsymbol{\theta}}_G = \ddot{\boldsymbol{\theta}}_G^d + \Delta\ddot{\boldsymbol{\theta}}_G$ satisfies Eq.(1). $\Delta\ddot{\boldsymbol{\theta}}_G$ is computed by the following equation:

$$\begin{pmatrix} \boldsymbol{A} & -\hat{\boldsymbol{H}}_C^T & -\boldsymbol{H}_J^T \\ \boldsymbol{H}_C & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \Delta\ddot{\boldsymbol{\theta}}_G \\ \boldsymbol{\tau}_C \\ \boldsymbol{\tau}_J \end{pmatrix} = \begin{pmatrix} -\boldsymbol{A}\ddot{\boldsymbol{\theta}}_G^d - \boldsymbol{b} \\ -\boldsymbol{H}_C\ddot{\boldsymbol{\theta}}_G^d - \dot{\boldsymbol{H}}_C\dot{\boldsymbol{\theta}}_G \end{pmatrix} \tag{2}$$

The dynamics filter consists of the following two components:

(A) determining $\ddot{\boldsymbol{\theta}}_G^d$ We need a feedback controller to correct the tracking error due to the modification of the acceleration. We apply the feedback controller used in resolved acceleration control [1] to all the joints including the 6 DOF of the body joint, to compute the desired acceleration $\ddot{\boldsymbol{\theta}}_G^d$. We also apply a skyhook controller which feedback the

Cartesian position of the head to keep the balance of the whole body. The skyhook controller modifies $\ddot{\boldsymbol{\theta}}_G^d$ such that the acceleration of the head equals to its desired acceleration computed from the current and desired position and orientation of the head link.

(B) optimization We first compute the solution space of the following equation using weighted pseudo inverse:

$$\left(\begin{array}{ccc} \Delta\ddot{\boldsymbol{\theta}}_G^T & \boldsymbol{\tau}_C^T & \boldsymbol{\tau}_J^T \end{array} \right)^T = \boldsymbol{W}^\sharp \boldsymbol{u} + \boldsymbol{V}\boldsymbol{y} \tag{3}$$

where $\boldsymbol{W}$ is the coefficient matrix of left-hand-side of Eq.(2), $\boldsymbol{u}$ is the vector in the right-hand-side, $\boldsymbol{V} = \boldsymbol{E} - \boldsymbol{W}^\sharp\boldsymbol{W}$, and $\boldsymbol{y}$ is an arbitrary vector. We can adjust the relative magnitude of the acceleration change and joint torques by the weight for computing the weighted pseudo inverse. Next, we reduce the error term $\Delta\ddot{\boldsymbol{\theta}}_G^T$ by using $\boldsymbol{y}$. Let us focus on the rows related to $\Delta\ddot{\boldsymbol{\theta}}_G^T$ of Eq.(3):

$$\Delta\ddot{\boldsymbol{\theta}}_G = \Delta\ddot{\boldsymbol{\theta}}_G^0 + \boldsymbol{V}_G\boldsymbol{y} \tag{4}$$

where $\Delta\ddot{\boldsymbol{\theta}}_G^0$ is the error term for $\boldsymbol{y} = \boldsymbol{O}$. $\boldsymbol{y}$ is computed by solving the following equation:

$$\boldsymbol{V}_G\boldsymbol{y} = -\Delta\ddot{\boldsymbol{\theta}}_G^0 \tag{5}$$

which is then substituted to Eq.(3) to compute the modified acceleration.

Figure 6 shows an example of generated motion, where we used the capture data of walking motion on a horizontal floor as the reference and put the human figure on a down slope. The dynamics filter can generate a physically feasible walking motion on a different environment.

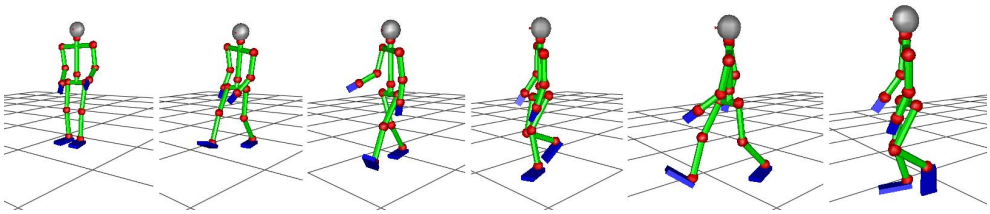

Figure 6: A walking motion on a down slope generated by the dynamics filter

Intuitive interface for generating whole-body motions Inverse kinematics is frequently used in robot motion planning as well as CG animation to compute the joint angles that realizes the given end-effector position. Most of the existing methods, however, are limited to 6 DOF arms or only consider a single arm or leg. It is still difficult to generate whole-body motions of human figures. In this research, we extended the conventional inverse kinematics algorithm to handle the whole-body simultaneously and enabled the whole-body motion generation by only one pin-and-drag operation. We also included joint motion range and desired joint angle constraints to yield natural motions without tweaking the parameters.

Summary of the algorithm The proposed algorithm takes the trajectory of the drag link and computes a whole-body motion that satisfies the following constraints:

1. fix the positions of any number of pinned links,
2. every joint do not exceed its motion range, and
3. the joint value of every joint becomes as close as possible to the desired value.

The trajectory of the drag link can be specified by any device such as mouse or joystick. As shown in the next section, the markers measured by motion capture system can be used as multiple drag links.

The same algorithm can also be used for realtime editing of existing motions by varying the positions of the pinned links instead of using fixed values. In walking motion, for example, we can modify the arm motion by dragging the hand link with the both feet pinned.

Equations The algorithm is composed of the following four steps:

1. Compute the Jacobian matrix of the position of the drag link with respect to the joint angles, $\boldsymbol{J}_P$, and obtain the joint angles that realizes the given drag link velocity $\dot{\boldsymbol{r}}_P^{ref}$ by

$$\dot{\boldsymbol{\theta}} = \boldsymbol{J}_P^{\sharp}\dot{\boldsymbol{r}}_P^{ref} + (\boldsymbol{E} - \boldsymbol{J}_P^{\sharp}\boldsymbol{J}_P)\boldsymbol{y} \tag{6}$$

   where $\boldsymbol{J}_P^{\sharp}$ is the weighted pseudo inverse of $\boldsymbol{J}_P$, $\boldsymbol{E}$ is the identity matrix, and $\boldsymbol{y}$ is an arbitrary vector. Eq.(6) is simplified as

$$\dot{\boldsymbol{\theta}} = \dot{\boldsymbol{\theta}}_0 + \boldsymbol{W}\boldsymbol{y}. \tag{7}$$

2. Compute the desired velocity of the constraints $\dot{\boldsymbol{p}}_{aux}^d$ from the current and ideal status. For a pinned link, for example, the desired velocity $\dot{\boldsymbol{r}}_{Fi}^d$ is computed from the current position $\boldsymbol{r}_{Fi}$ and the pinned position $\boldsymbol{r}_{Fi}^{ref}$ as

$$\dot{\boldsymbol{r}}_{Fi}^d = \boldsymbol{K}_{Fi}(\boldsymbol{r}_{Fi}^{ref} - \boldsymbol{r}_{Fi}) \tag{8}$$

   where $\boldsymbol{K}_{Fi}$ is a positive-definite gain matrix. If there are joints out of their motion range, we also set a desire velocity for each of them such that the joint value returns to its motion range.

3. Compute the Jacobian matrix of the constraints with respect to the joint values, $\boldsymbol{J}_{aux}$, which is defined as

$$\dot{\boldsymbol{p}}_{aux} = \boldsymbol{J}_{aux}\dot{\boldsymbol{\theta}}. \tag{9}$$

   For a pinned link, for example, we include the Jacobian matrix of its position with respect to the joint angles.

4. Compute the joint velocity described as Eq.(6) that realizes the desired velocity $\dot{\boldsymbol{p}}^d_{aux}$ as much as possible. First we substitute Eq.(7) into Eq.(9) to yield

$$\dot{\boldsymbol{p}}_{aux} = \boldsymbol{J}_{aux}\dot{\boldsymbol{\theta}}_0 + \boldsymbol{J}_{aux}\boldsymbol{W}\boldsymbol{y}. \tag{10}$$

Eq.(10) is written as

$$\boldsymbol{S}\boldsymbol{y} = \Delta\dot{\boldsymbol{p}}_{aux} \tag{11}$$

where $\boldsymbol{S} = \boldsymbol{J}_{aux}\boldsymbol{W}$ and $\Delta\dot{\boldsymbol{p}}_{aux} = \dot{\boldsymbol{p}}^d_{aux} - \boldsymbol{J}_{aux}\dot{\boldsymbol{\theta}}_0$. We solve this equation by the singularity-robust inverse (SR-inverse) [2] as

$$\boldsymbol{y} = \boldsymbol{S}^*\Delta\dot{\boldsymbol{p}}_{aux} \tag{12}$$

where $\boldsymbol{S}^*$ denotes the SR-inverse of $\boldsymbol{S}$. Normal inverse is not sufficient here because generally $\boldsymbol{S}$ is not a full-rank matrix. Finally, the joint velocity is computed by substituting $\boldsymbol{y}$ into Eq.(7).

The proposed algorithm has the following properties:

- A single pin-and-drag operation moves the whole body.
- Any link can be pinned or dragged, even if it is not an end effector.
- Any number of links can be pinned.
- The pinned and drag links can be switched at any time.
- We never encounter unnaturally large joint velocities even when the constraints conflict with each others thanks to the SR-inverse.
- We can adjust the movability of the joints and the priorities of the constraints by changing the weights for computing the weighted pseudo inverse.
- The motion is generated in real time on normal PC.

Example The algorithm is implemented as the computational engine of Animanium^{TM} developed by Sega Corporation for creating CG animation. The software package is equipped with the interface for switching pinned and drag links and setting the parameters.

Figure 7 shows the motion generated by a single drag operation on the right hand while the toes, heels, and the left hand are pinned.

Behavior capture system The behavior capture system was developed by connecting various measurement tools with our original motion capture system.

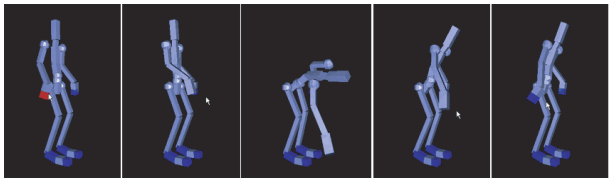

Figure 7: An example of motion generated by a single pin-and-drag operation

Table 2: Cameras used in the motion capture system

name	number	resolution	rate (frame/sec)
Adimec1000	10	1000×1000	30–50
Dalsa	8	512×512	252

Motion capture system We adopted passive optical motion capture considering its expandability and potential speed. The monochrome cameras detect the positions of the retro-reflective markers attached to the subject, which are then reconstructed to compute their Cartesian positions.

The system overview is shown in Figure 8. Each camera sends the image data to a PC which processes the image to detect the markers in parallel. The reconstruction PC receives the 2D position data from all camera PC's and computes the Cartesian positions of the markers. The PC's are connected via Myrinet to minimize the overhead due to inter-process communications. The user can select appropriate cameras listed in Table 2 according to the velocity and required precision of the target motion.

Labeling is the largest technical problem in passive optical motion capture. Labeling is the process to identify the correspondence between the detected markers and the given marker set. We developed an algorithm utilizing asymmetric marker placements which is capable to label the markers in real time. The motion in Figure 9 was capture in real time using this technique.

We also apply the inverse kinematics algorithm described in the previous section to compute the joint angles from marker positions. Our flexible algorithm enables the conversion from capture data with any number of markers to joint data of any model.

Behavior capture system The behavior capture system was integrated by connecting the following devices with the motion capture system:

- force plate to measure the force between the subject and the floor,
- eye-mark recorder to detect the direction of the gaze,
- hybrid magnetic-ultrasonic motion tracker (IS600) to measure the 3D position and orientation of a link, and
- electro-myograph (EMG) to measure the muscle activity.

The program to obtain the data from each device, including the motion capture system, is implemented as a CORBA server, so that the user can connect to the devices via the

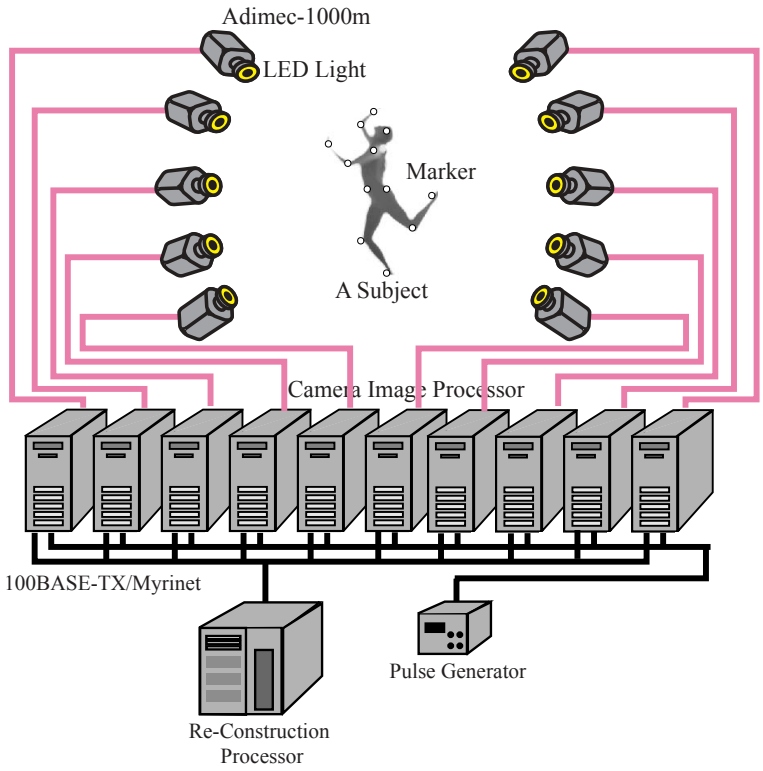

Figure 8: Overview of the motion capture system

network by a client program. Figure 10 visualizes the result of simultaneous measurement using the force plate, the eye-mark recorder, IS600, and the motion capture system.

Dynamics computation of musculoskeletal human model

Musculoskeletal human model We constructed a musculoskeletal human model shown in Figure 11. This model is composed of 51 bone groups, 366 muscles, 91 tendons, and 34 ligaments, and has 155 DOF. Each muscle, tendon, or ligament is modeled as a wire with a linear actuator. Branched muscles such as M. Biceps Bracii are modeled using virtual links as shown in Figure 12.

Inverse dynamics Inverse dynamics can compute the forces to drive the muscles to realize a captured motion sequence as well as the somatosensory information such as the forces applied to the bones or the tendons. The inverse dynamics of musculoskeletal models is formulated as follows:

$$\boldsymbol{\tau} = \boldsymbol{J}^T \boldsymbol{f} \tag{13}$$

where $\boldsymbol{\tau}$ is the joint torques computed by the inverse dynamics computation of kinematic chains, $\boldsymbol{f}$ is the muscle force vector, $\boldsymbol{J}$ is the Jacobian matrix of the muscle length with respect to the joint angles. In most cases, Eq.(13) is a redundant equation with infinite number of solutions because the number of muscles is greater than the DOF. We

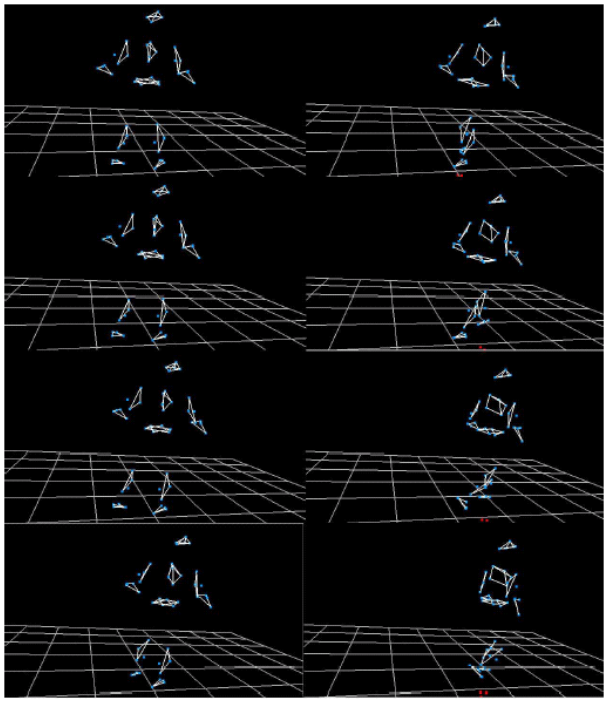

Figure 9: Realtime motion capture using asymmetric marker placements

developed two algorithms to solve Eq.(13) using linear programming and quadratic programming that minimizes the total muscle force. Figure 13 visualizes the result of inverse dynamics computation for a kick motion. The color of muscles changes from yellow to red as their force increase.

Mapping from human figure to musculoskeletal model Musculoskeletal human model is a complex system with many number of DOF. It is therefore very difficult to generate and control its motion. If we could find a way to map motions of humanoid to those of musculoskeletal model, it would be possible to apply the techniques for humanoid to musculoskeletal models.

We describe the mapping function by polynomials as follows:

$$\boldsymbol{\phi} = \boldsymbol{M}\boldsymbol{f}(\boldsymbol{\theta}) \tag{14}$$

$$\boldsymbol{\theta} = \boldsymbol{N}\boldsymbol{g}(\boldsymbol{\phi}) \tag{15}$$

where $\boldsymbol{\phi}$ is the joint angle vector of the musculoskeletal model, $\boldsymbol{\theta}$ is the joint angle vector of the human figure, and $\boldsymbol{f}(\boldsymbol{\theta})$ and $\boldsymbol{g}(\boldsymbol{\phi})$ are the terms of the polynomial functions of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively. $\boldsymbol{M}$ and $\boldsymbol{N}$ are the mapping matrix between the two models. We compute the mapping matrices from several sample configurations using pseudo inverse based on singular value decomposition. We can modify the degree of adjustment by changing the number of singular values for computing the pseudo inverse. Figure 14 shows several pairs of postures generated using the mapping function. The left three postures were used to compute the mapping function, while the right three were not. The number indicates the number of singular values used for the pseudo inverse. Although increasing the number

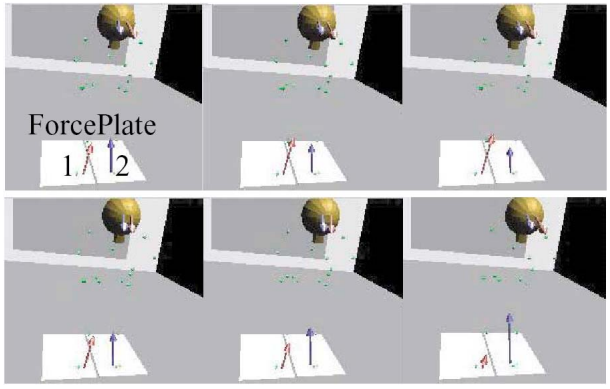

Figure 10: An example of simultaneous measurement using behavior capture system

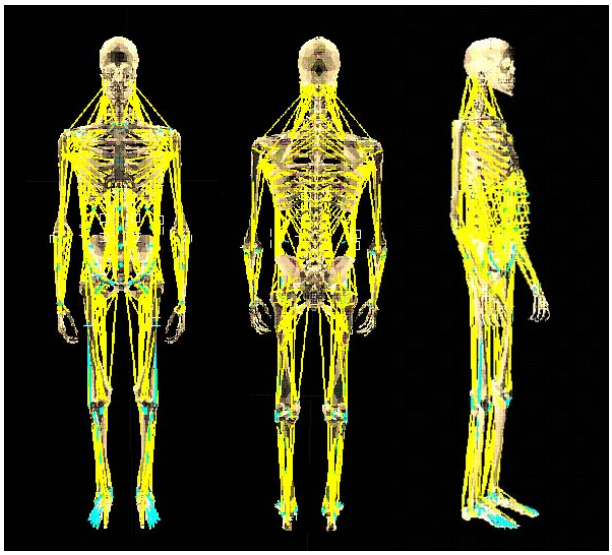

Figure 11: Musculoskeletal human model

of singular values generally improves the approximation, it is not good for postures not included in the samples because smaller singular values may represent the feature of the sample postures rather than the structural difference.

(2) Results and their importance

In this research, we developed the following methods and systems:

- Parallel efficient dynamics computation of human figures: This method not only serves as the basis for motion generation considering dynamics, but also improves the efficiency of the computations of simulating and controlling motions of humanoids.

- Motion generation of human figures considering physical consistency: This method, called dynamics filter, can generated motions that are both physically consistent and

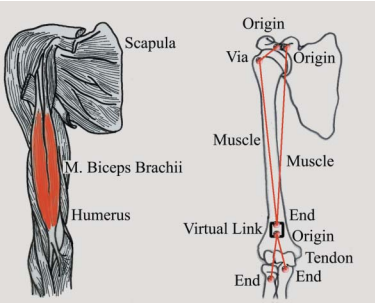

Figure 12: Model of M. Biceps Bracii with a virtual link

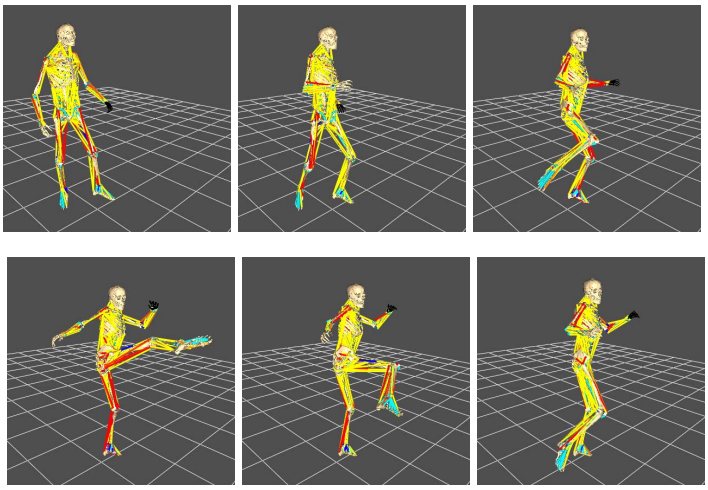

Figure 13: Inverse dynamics computation of musculoskeletal model

human-like by modifying motion capture data.

- Intuitive motion generation using inverse kinematics: This method is capable of generating whole-body motions of human figures by only specifying several fixed links and the trajectory of a link. This is enabled by extending conventional algorithm for inverse kinematics.

- Behavior capture system: We combined our original motion capture system with other sensors including force plate and gaze direction sensor.

- Dynamics computation of musculoskeletal human model: We can compute the somatosensory information by developing the methods for computing the dynamics of human model composed of bones, muscles, and tendons.

The techniques for dynamics simulation, motion generation, and motion / sensory information measurement for human figures developed in research would serve as the basis for humanoid and cognitive science. In fact, some of these techniques have been adopted by other research groups in the project. Some are also applied to software packages for humanoid simulators and CG animation.

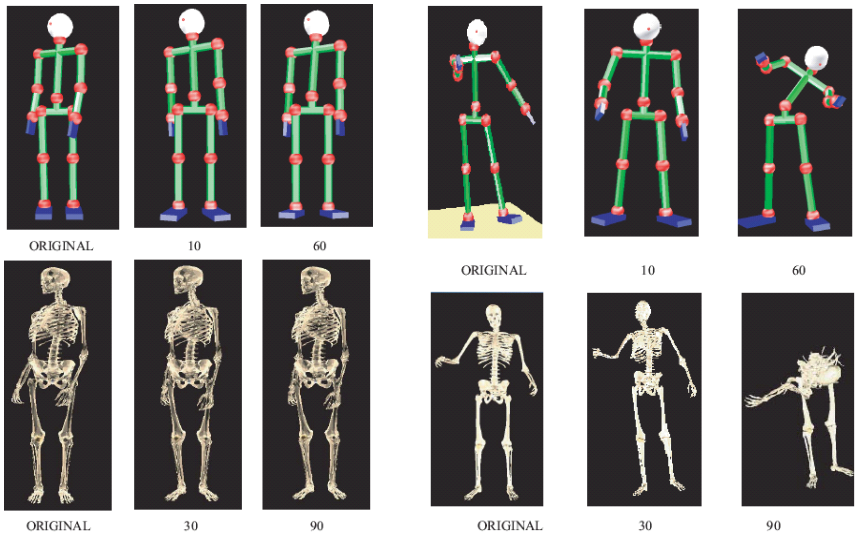

Figure 14: Mapping from human figure to musculoskeletal model

The extension to musculoskeletal human model would have applications beyond humanoids. Potential applications include investigation of human motion control mechanism using somatosensory information, development of new human-robot interface, and applications to medical and sport science.

References

[1] J.Y.S. Luh, M.W. Walker, and R.P.C. Paul: Resolved Acceleration Control of Mechanical Manipulators, IEEE Transactions on Automatic Control, vol.25, no.3, pp.468–474, 1980.

[2] Y. Nakamura and H. Hanafusa: Inverse Kinematics Solutions with Singularity Robustness for Robot Manipulator Control, Journal of Dynamic Systems, Measurement, and Control, vol.108, pp.163–171, 1986.