애니메이션 공간에서의 제어를 통한 동작 생성

박 지 헌[†] · 박 성 헌[††]

요 약

본 논문은 관절을 가진 물체를 애니메이션하기 위하여 좌표계 변환을 사용한 애니메이션 기법을 다룬다. 이 방법은 관절 체로 표현된 공간과 사용자가 정의한 애니메이션 공간과의 변환을 통하여 이루어지는데, 애니메이션 공간은 아주 일반적인 공간으로, 사용자의 애니메이션 목적에 따라 다양한 함수로 구성되는 공간이다. 제약 식은 애니메이션 공간을 설정할 때 포함시킬 수 있고, 애니메이션 공간은 매우 일반적이어서 closed loop, open loop등 거의 모든 종류의 애니메이션 문제를 표현, 해결할 수 있다. 설정된 다양한 애니메이션 함수들은 ,예를 들어, 동작 재활용에 있어서 추가의 동작제어를 할 경우 활용이 가능하다. 이를 보여주기 위하여 closed loop 및 open loop 동작을 예로 보여주며, open loop에서는 일반 동작과 달리 제약을 가진 관절체의 특정 부위에 대한 직접적인 제어가 가능함을 보였다. 또한 여러 다른 애니메이션 방법들을 비교하기 위하여 여러 다른 방법으로 구현된 벌레들과, 뛰어서 벌레를 밟는 사람에 대한 영화를 만들었다.

Motion Control on Animation Space

Ji-Hun Park[†] · Sung-Hun Park[††]

ABSTRACT

This paper presents a new methodology for specification and control of the motion of an articulated rigid body for the purposes of animation by coordinate transformations. The approach is to formulate the problem as a coordinate transformation from the joint space of the body to a user-defined animation space which is chosen for convenience in constraining the motion. Constraints are applied to the resulting coordinate transformation equations. It is sufficiently general so that it can be applied to all common types of control problems, including closed loop as well as open loop mechanisms. We also provided a new approach to simulate a closed loop mechanism, which is using animation space transformation technique. The method is formulated in detail and is demonstrated by animating the motion of an inchworm.

키워드 : 애니메이션(Animation), 동작 제어(Motion Control), 좌표계 변환(Coordinate Transformation), Kinematics

1. Introduction

It is by now well known that producing realistic animated motion of natural objects requires modeling the dynamics which determine key aspects of that motion [5, 26, 27]. Controlling very complex motions such as walking animals is a particularly challenging task. Ideally, an animation system allows the animator to directly specify the macroscopic features of the motion such as the intermediate goal configurations of a moving animal. The direct specification of the kinematic features of the walking motion itself, however, is not at all desirable, since it is essentially impossible to achieve realism this way. Instead, a dynamic model of the animal is built which determines these details. It is desirable that such models be specified in terms of parameters which allow effective and intuitive control by the animator of the relevant features of the motion.

Dynamic models of articulated bodies typically have linear or tree-structured topologies. In either case, they are *open loop* mechanisms. That is, they can be modeled as links connected by joints, where the links at the ends of the structure have a free, unconstrained end. However, creatures moving on the ground cannot always be modeled as open loop mechanisms since the free ends of the links are often constrained by the forces involved when they touch the ground. In this case, a *closed-loop* model of the moving

※ 본 연구는 한국과학재단 목적기초연구(2000-2-30400-011-1) 지원으로 수행되었음.
† 정 회 원 : 홍익대학교 컴퓨터 공학과 교수
†† 정 회 원 : 명지대학교 지식정보학부 교수
논문접수 : 2000년 11월 13일, 심사완료 : 2001년 5월 11일

creature is needed. Closed-loop models occur in many other situations besides locomotion. A human holding a cup or lifting a barbell also requires a closed-loop model.

Given a model, we still need to determine values for the dynamic parameters such as the appropriate torques on the joints of the body for a forward dynamic simulation which will give proper motion for a particular specified animation. There are two basic approaches to determining these parameters. One is to use heuristic algorithms which generate torque values for each motion increment and then verify that they cause the proper effects. Another is to use inverse dynamics to specify constraints on the dynamic parameters which will cause the specified motion to occur.

Either type of model can be specified in terms of the *generalized coordinate system* or *joint space* of the model itself. This space has a dimension for each degree of freedom (DOF) of each of the joints in the model. The forces exerted at each joint determine the motion of the mechanism in its working space, which is normally Cartesian space with six degrees of freedom in the case of a single open kinematic loop. We can think of the working space as the space of kinematic parameters input to the system, for which we must compute the forces in joint space needed to achieve the specified result using inverse dynamics. These dynamic parameters can be applied using a forward dynamic simulator to generate the resulting motion.

We can specify the relationship between joint space and working space as a coordinate transformation from n-dimensional joint space to 6 dimensional Cartesian space. Since for most articulated creatures n is much larger than 6, a number of unconstrained degrees of freedom are left in the transformation. If in such cases a particular spatial configuration of the articulated body has been specified, there will be a large number of solutions to the transformation equations which produce this result. This redundancy makes motion control very difficult [22].

The usual approach to this problem is to add constraint functions, and if the system remains underdetermined, to use optimization techniques to find the required joint angle velocities for given end effector positions and orientations. From this information the joint angle values are obtained by integrating. This approach does not lead to an easy interface for animator specified parameters to be input to the system.

Any minimal set of coordinates to specify a motion is called a set of generalized coordinates [3] ; generalized refers to the fact that they need not be only positional or only angular coordinates. We extended generalized coordinates to define *animation space*. By *animation space* we mean that we allow any function, not only position or angle, meaningful for our motion control to be modeled as a coordinate dimension. Our approach to the problem is to replace working space with an arbitrary user-designed *animation space* with up to n dimensions. Typically, this user-designed space will contain working space as a subspace, with additional dimensions defined to represent features of the system which are to be constrained by input from the animation specification. We refer to techniques based on transformations between generalized coordinate systems as *animation space transformation* methods. Based on this generalized coordinate transformation and differential relationships between joint space and animation space, we introduce a dynamic formulation which is based on the General Principle of D'Alembert and virtual work. This approach gives a well-structured matrix (closed) form for the control equations. This formulation is particularly useful in the analysis of dynamic system models. We have used this dynamic formulation for several physical simulations including an open loop serial chain (figure 3), an inchworm (figure 5) and a pin pointed dropping chain.

The remainder of the paper is organized as follows. In section 2 we review related work and in section 3 we introduce our approach. In section 4 we provide motion equations for animation of linear chain topologies. Section 4.1 contains a new formulation of the geometry of the serial structure of an articulated body. In section 4.2 we describe the *animation space transformation technique*, and in section 4.3 we introduce a dynamic formulation which is easy to use for analysis purposes. In section 4.4 we show three ways to simulate closed loop systems, and in section 5 we describe specific dynamic simulation results.

2. Related Work

Much work has been done on the dynamic simulation of moving creatures or articulated bodies in the past several years. The most common approaches model a real creature as an articulated body consisting of joints connected by links, possibly also including springs and dampers [27, 26]. In [21], a specialized spring and damper body for modeling the sliding motions of snakes and worms was developed.

The motions of these models are determined in two basic ways. Forward dynamics based systems provide realistic simulations of the motion of figures but are difficult for an animator to control since the specification of unknown joint torques is required. Lagrangian [5], Gibbs-Appell [25], Armstrong's Newton-Euler [2] and Featherstone's [20] formulations of the dynamics have been used in such systems. Iterative approaches, particularly the latter two, provide the greatest computational efficiency among extant simulation methods.

The basic problem in forward dynamics systems is finding a set of torque functions to control the body. This requires either a heuristic approach or an inverse dynamic (constraint based) approach. In [16], a system allowing geometric constraints on the joints and specification of either accelerations or torques on the joints for open loop systems is described. Witkin and Kass [28] give a method which solves geometric constraint equations on the joints along with constraints on the control forces for the entire span of the simulation at once [19]. discusses an iterative method to find a path satisfying kinematic and dynamic constraints.

In [17], a method based on D'Alembert's principle and virtual work which allows inverse dynamic solutions for both open and closed loop systems was developed. An articulated body is defined in terms of a generalized coordinate system with a dimension for each DOF of each joint in the figure. Constraint equations involving one or more of these DOFs can be specified. Lagrangian multipliers for each kinematic constraint equation are used to represent the unknown forces required to satisfy the constraints. Other techniques for handling closed loop systems have been developed for robotics. Freeman [9, 11] has worked on two dimensional serial chains and Stewart platforms. His approach is to add selected joint positions and angles to the working space to form a new space with the same dimensionality as joint space. A constrained coordinate transformation from generalized coordinates (joint space) to this target space provides the solution to torques required.

Much of the recent research in motion control has been devoted to developing various kinds of editing tools to produce a convincing motion from prerecorded motion clips. To reuse motion captured data, animators often adapt them to a different character, i.e., retargeting a motion from one character to another [14], or to a different environment to compensate for geometric variations [29]. Animators also combine two motion clips in such a way that the end of one motion is seamlessly connected to the start of the other. We survey the motion editing techniques developed so far. Non-interactive motion editing involves motion editing with constraints and large number of global controls (variables) for parameter optimization.

A common approach in editing and reuse of existing motion is to provide interactive animation tools for motion editing, with the goal of capturing the style of the existing motion, while editing the content. Gleicher [13] provides a low-level interactive motion editing tool that searches for a new motion that meets some new constraints while minimizing the distance to the old motion. A related optimization method is also used to adapt a motion to new characters [14]. Lee et al [18]. provide an interactive multi-resolution motion editor for fast, fine scale control of the motion.

Gleicher [13] suggested a method for editing a pre-existing motion such that it meets new needs yet preserves as much of the original quality as possible. Their approach enables the user to interactively position characters using direct manipulation. They used spacetime constraints which consider the entire motion simultaneously. These methods enable the user to specify constraints over the whole motion and use a solver to compute the best motion that meets these requirements. Like more traditional keyframe and inverse kinematics methods, the user makes adjustments to an animated character with direct manipulation, for example pulling on a character's hand to reposition it. But, to achieve these new positions, the animation system makes adjustments that attempt to preserve the original motion.

Gleicher [5, 14] suggested a technique for retargeting motion that is the problem of adapting an animated motion from one character to another. Their focus is on adapting the motion of one articulated figure to another figure with identical structure but different segment lengths. Their method creates adaptations that preserve desirable qualities of the original motion. They identify specific features of the motion as constraints that must be maintained. A spacetime constraints solver computes an adapted motion that re-establishes these constraints while preserving the frequency characteristics of the original signal. Constraints are used to identify features of the original motion that must be present in the retargeted result. Specification of these constraints typically involves only a small amount of work in comparison with the tasks of creating the characters and motions. Constraints are generally defined once for each motion, and this one set of constraints is used for any retargettings. Since there are typically many possible motions that satisfy the constraints, they use an objective function to select the best choice.

Lee et al [18]. suggested a technique for adapting existing motion of a human-like character to have the desired features that are specified by a set of constraints. This problem can be typically formulated as a spacetime constraint problem. Their approach combines a hierarchical curve fitting technique with inverse kinematics. They employed the multi-level B-spline fitting technique. Using the kinematics solver, they can adjust the configuration of an articulated figure to meet the constraints in each frame. Through the fitting technique, the motion displacement of every joint at each constrained frame is interpolated and thus smoothly propagated to frames. They are able to adaptively add motion details to satisfy the constraints within a specified tolerance by adopting a multi-level B-spline representation. The performance of their system is further enhanced by the new inverse kinematics solver.

3. Our Approach

Our approach is an extension of the generalized coordinate technique [10]. We relax restrictions on *generalized* working space to define an *animation space* which is allowed to have any number of DOF less than or equal to that of joint space and to allow any function of the kinematic parameters to form a dimension in animation space. These newly added dimensions represent kinematic constraints imposed on the dynamic model by the animation system in which it is embedded. Their values are specified as inputs to the dynamic model, which then solves for the torques required to satisfy them. Our methodology is general enough to handle open and closed loop mechanisms and most other common types of control problems.

We will deal with highly redundant systems. In a redundant system whose degree of freedom is slightly greater than that of animation space, we can usually control motion by adding constraints using Lagrangian multipliers. But because the articulated body is highly redundant, we can not control the *shape* of the body by Lagrangian multipliers alone. We use Lagrangian multipliers in calculating pseudo inverse matrices, but we also extended animation space to actively control the shape of the body.

Our Animation Space Transformation technique can be used in retargetting motion. We set a number of interest points of an animated character. The points can be functions representing either positions or orientations, for example. These functions make a space what we call "Animation Space." We can extend our concept a little bit by allowing any meaningful function for motion retargetting. Say we have a set of source data of a source articulated rigid body. We need to transform/retarget source motion to a target body we desire. Target structure should be same, but we can allow different link segment length, different DOF, and different type of joints. The only resort for us to retarget motion seems using positions and orientations of the source body. The corresponding points of the target should be assigned. The set of functions related with points of the target makes a space called Animation Space. The motion transformation from joint space to animation space is what we call "Animation Space Transformation," because joint space is also considered a subset of Animation Space.

4. Motion Control Equations

4.1 Coordinate Transformations and Differential Relationships

If we are trying to represent a coordinate in terms of another coordinate, we need to derive a set of equations relating one coordinate to another. In robotics, this usually means transforming from the generalized coordinate space of the joints, where each dimension represents one DOF of a joint, to Cartesian space, which is the normal type of working space. Thus each object can be represented by a set of positions and orientations. In special circumstances, the working space can be spherical or cylindrical [3]. Freeman and Tesar [9, 12] allow the working space to be a generalized coordinate system consisting of Cartesian space positions and orientations and joint angles. This generalized system is required to have the same dimension as joint space.

Usually the coordinate transformation from joint space to Cartesian space is aimed to give output in Cartesian space when the motion control is done in joint space. We may interpret this in reverse, where if we want some output in Cartesian space, we calculate the corresponding effective change in joint space. We extend the idea of generalized coordinate transformations by including *implicit* output functions [10]. By implicit, we mean that the meaning of the function is usually not as obvious as other position or orientation functions. For example if we want to fix joint i, then the function can be $(\phi_i - a)^4$ where a is joint angle. Note that there are many ways of controlling the constraint. We can use any arbitrary function in terms of joint angles and positions and orientations in the ordinary working space to control the motion of the articulated body. That is to say, we create a new *animation* space whose dimensions are defined by these functions. So our animation space can include motion constraint space as well as positions and orientations which are for motion control.

Let $\vec{u}$ denote a vector in the created m dimensional animation space and $\vec{\phi}$ a vector in joint space having n degrees of freedom. Let us work on a general function, f_i, $(i=1,2,3)$, for animation space. Then

$$\vec{u} = \begin{pmatrix} f_1(\phi_1, \phi_2, \cdots, \phi_n) \\ f_2(\phi_1, \phi_2, \cdots, \phi_n) \\ \vdots \\ f_m(\phi_1, \phi_2, \cdots, \phi_n) \end{pmatrix}$$

$$\begin{aligned} \dot{f}_i &= \frac{\partial f_1}{\partial \phi_1}\frac{d\phi_1}{dt} + \frac{\partial f_1}{\partial \phi_2}\frac{d\phi_2}{dt} + \cdots \\ &+ \frac{\partial f_i}{\partial \phi_n}\frac{d\phi_n}{dt} \\ &= g_1^{f_i}\dot{\phi}_1 + g_2^{f_i}\dot{\phi}_2 + \cdots + g_u^{f_i}\dot{\phi}_n \end{aligned}$$

$$\begin{aligned} \dot{\vec{u}} &= \begin{pmatrix} g_1^{f_1} \\ g_1^{f_2} \\ \vdots \\ g_1^{f_m} \end{pmatrix} \dot{\vec{\phi}} + \begin{pmatrix} g_2^{f_1} \\ g_2^{f_2} \\ \vdots \\ g_2^{f_m} \end{pmatrix} \dot{\vec{\phi}} + \cdots \begin{pmatrix} g_n^{f_1} \\ g_n^{f_2} \\ \vdots \\ g_n^{f_m} \end{pmatrix} \dot{\vec{\phi}} \\ &= g_1^u\dot{\phi}_1 + g_2^u\dot{\phi}_2 + \cdots + g_n^u\dot{\phi}_n \end{aligned}$$

$$= [G_\phi^u]\dot{\vec{\phi}} \tag{1}$$

where G is a gradient or Jacobian tensor. The tensor $[G_\phi^u]$ consists of

$$\begin{aligned} [G_\phi^u] &= \left[\frac{\partial \vec{u}}{\partial \phi_1} \frac{\partial \vec{u}}{\partial \phi_2} \cdots \frac{\partial \vec{u}}{\partial \phi_n}\right] \\ &= [g_1^u g_2^u \cdots g_n^u]. \end{aligned} \tag{2}$$

g_i^u denotes the effect in $\vec{u}$ space of a change in joint i.

Up to now we have discussed first order differential relationships. But we also frequently need second order differential relationships. An acceleration vector of $\vec{u}$ space, $\ddot{\vec{u}}$, can be expressed in terms of a gradient tensor and Hessian tensor.

$$\begin{aligned} \ddot{f}_i &= g_1^{f_i}\ddot{\phi}_1 + g_2^{f_i}\ddot{\phi}_2 + \cdots + g_n^{f_i}\ddot{\phi}_n \\ &+ \dot{g}_1^{f_i}\dot{\phi}_1 + \dot{g}_2^{f_i}\dot{\phi}_2 + \cdots + \dot{g}_n^{f_i}\dot{\phi}_n \end{aligned}$$

$$\begin{aligned} \dot{g}_j^{f_i} &= \frac{\partial g_j^{f_i}}{\partial \phi_1}\frac{d\phi_1}{dt} + \frac{\partial g_j^{f_i}}{\partial \phi_2}\frac{d\phi_2}{dt} + \cdots \\ &+ \frac{\partial g_j^{f_i}}{\partial \phi_n}\frac{d\phi_n}{dt} \\ &= h_{j1}^{f_i}\dot{\phi}_1 + h_{j2}^{f_i}\dot{\phi}_2 + \cdots + h_{jn}^{f_i}\dot{\phi}_n \end{aligned}$$

$$\begin{aligned} \ddot{f}_i &= \begin{pmatrix} g_1^{f_i} \\ g_2^{f_i} \\ \vdots \\ g_n^{f_i} \end{pmatrix} \ddot{\vec{\phi}} \\ &+ \dot{\vec{\phi}}^T \begin{pmatrix} h_{11}^{f_i} & h_{12}^{f_i} & \cdots & h_{1n}^{f_i} \\ h_{21}^{f_i} & h_{22}^{f_i} & \cdots & h_{2n}^{f_i} \\ \vdots & \vdots & \vdots & \vdots \\ h_{n1}^{f_i} & h_{n2}^{f_i} & \cdots & h_{nn}^{f_i} \end{pmatrix} \dot{\vec{\phi}} \end{aligned}$$

$$\begin{aligned} \ddot{\vec{u}} &= [g_1^u g_2^u \cdots g_n^u]\ddot{\vec{\phi}} + \begin{pmatrix} \dot{\vec{\phi}}^T[H_{\phi\phi}^{f_1}]\dot{\vec{\phi}} \\ \dot{\vec{\phi}}^T[H_{\phi\phi}^{f_1}]\dot{\vec{\phi}} \\ \vdots \\ \dot{\vec{\phi}}^T[H_{\phi\phi}^{f_1}]\dot{\vec{\phi}} \end{pmatrix} \\ &= [G_\phi^u]\ddot{\vec{\phi}} + \dot{\vec{\phi}}^T[H_{\phi\phi}^u]\dot{\vec{\phi}} \end{aligned} \tag{3}$$

Let us assume $\vec{u}$ space is defined by a set of m functions f_1^i, f_2^i, $\cdots$ f_m^i of the ϕ_i. Then the G and H tensors are given by

$$[G_\phi^u] = \begin{pmatrix} \frac{\partial f_1}{\partial \phi_1} & \frac{\partial f_1}{\partial \phi_2} & \cdots & \frac{\partial f_1}{\partial \phi_n} \\ \frac{\partial f_2}{\partial \phi_1} & \frac{\partial f_2}{\partial \phi_2} & \cdots & \frac{\partial f_2}{\partial \phi_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial \phi_1} & \frac{\partial f_m}{\partial \phi_2} & \cdots & \frac{\partial f_m}{\partial \phi_n} \end{pmatrix} \tag{4}$$

$$[H^{f_i}_{\phi\phi}] = \begin{pmatrix} \frac{\partial^2 f_i}{\partial\phi_1\partial\phi_1} & \frac{\partial^2 f_i}{\partial\phi_1\partial\phi_2} & \cdots & \frac{\partial^2 f_i}{\partial\phi_1\partial\phi_n} \\ \frac{\partial^2 f_i}{\partial\phi_2\partial\phi_1} & \frac{\partial^2 f_i}{\partial\phi_2\partial\phi_2} & \cdots & \frac{\partial^2 f_i}{\partial\phi_2\partial\phi_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f_i}{\partial\phi_n\partial\phi_1} & \frac{\partial^2 f_i}{\partial\phi_n\partial\phi_2} & \cdots & \frac{\partial^2 f_i}{\partial\phi_n\partial\phi_n} \end{pmatrix} \quad (5)$$

Here f can be interpreted to be a very general function. The position and orientation functions of an end-effector are well-known cases of f, which are commonly used in robotics. We can introduce additional functions as long as the total does not exceed the number of degrees of freedom of the articulated body.

The derivation of the differential relations between the joint space and animation space dimensions is based on [9, 12]. These relations describe the effective change of every position (including the base of local coordinates) of a serial kinematic chain as a result of a change in a joint.

Now we have both direct and differential relationships between $\vec{u}$ space and $\vec{\phi}$ space. For example, geometry (position and orientation) is a common direct relationship. For control, we can assign some values to the vectors $\dot{\vec{u}}$ and $\ddot{\vec{u}}$ in the newly created space. By the constraints thus imposed on $\dot{\vec{u}}$ and $\ddot{\vec{u}}$, the articulated body is constrained in motion. The simplest case is assigning the value zero. If we assign zero to the velocity and acceleration functions of the end-effector position, we convert an open loop chain mechanism to a closed loop.

Usually the dimension of $\vec{u}$ will be smaller than that of joint space. In this case, we need to apply optimization techniques, as is commonly done in robotics [3, 22]. In our simulation, we used a joint velocity minimization technique.

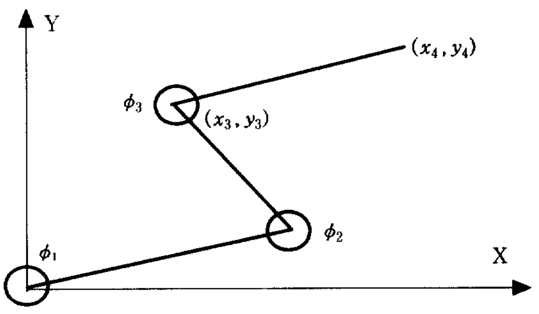
4.2 Basic Dynamic Equations

Using the gradient and Hessian tensors we have derived, we can formulate dynamics equations for a dynamic animation. This section is heavily due to [9, 12, 24]. The dynamic equation is based on the Generalized Principle of D'Alembert and virtual work, that is to say, the derivation of the dynamic equation is based on the virtual work of inertial loads. We can use other techniques based on inertial power or Lagrangian methods to derive exactly the same set of equations. Be ware that the dynamic equation is based on three dimensional space. In order to derive dynamic equations we need to derive kinematic differential equations in terms of positions and orientations. If we have external forces / torques, then we also have to compute differential relations for each corresponding positions.

4.3 Closed Loop Mechanisms

There are three methods for simulating the closed loop mechanism : soft constraints, hard constraints and simulation by *animation space* transformation technique.

4.3.1 Simulation by the Extended Generalized Coordinate Transformation Technique



(Figure 1) Example of Serial Chain

By this method closed loop mechanisms are easily handled by including corresponding constraint equations at animation space and by doing motion control appropriately. Consider (Figure 1). Notice that this model is the same with a human sitting on a chair. If we want to simulate the human, then we need to make an animation space. Because this example is highly constrained we need to be careful. If there are lots of redundancy, it may be easier to decide an animation space. Let us make animation space and its control as follows. $\vec{u} = (x_3\ y_3\ \phi_3)^T$, $\dot{\vec{u}} = (0\ 0\ \dot{\phi}_3)^T$, $\ddot{\vec{u}} = (0\ 0\ \ddot{\phi}_3)^T$ so that we can constrain $\dot{x}_3 = \ddot{x}_3 = \dot{y}_3 = \ddot{y}_3 = 0$ to fix (x_3, y_3). By differential kinematic equations derived so far, we can calculate corresponding joint velocity and acceleration values. Then with this, we do inverse dynamic simulation. We still need the simulation if there are external forces acting on a body except the forces caused by hip constraints. By this way, we can simulate human sitting on a chair. Because this example is really simple, the animation becomes straight forward.

4.3.2 Soft Constraints Technique

Soft constraints method models external forces to constraint the motion by using stiff springs and dampers. Consider (Figure 1). In order to fix a hip of a human body which corresponds to (x_3, y_3), we need to use vertical and horizontal direction stiff spring and dampers. Dampers should work opposite direction than that of springs to restrain wild action. We do forward dynamic simulation in terms of time by including external forces caused by springs and dampers. During simulation, the hip is moving a very small amount which causes springs and dampers reaction forces. Due to the reaction forces, the hip moves to opposite direction. But by the small amount of displacement, the hip get opposite direction forces due to springs and dampers. But this approach is only effective for forward dynamic simulation (so can not be applied in this paper approach) and is very expensive in terms of computation time because we need to make the integration time step very small especially wild motion is involved. For some tip, even this simple simulation is not easy because we need to carefully select spring and damper coefficients, and integration time step, and the most difficult part is the control of the body. We used a weak variant of an optimal control method to get a best set of controls for the human movement. Also a good model of human foot is very important if the simulation(animation) includes human jumping or walking.

4.3.3 Hard Constraints Technique

In this method, we use kinematic constraints to derive the resulting constrained closed loop mechanism. In terms of this approach the motion equation usually becomes messy. Our dynamic model carries enough information that we can model hard constraints in a closed form equation and is shown in this subsection. Other dynamic formulation, for example recursive algorithms [8], usually do not compute enough differential kinematic relations and can not include hard constraints straight forwardly because their equation is not in a closed form. We have to derive kinematic constraints for a set of joint angles and then has to simplify corresponding dynamic equation.

Consider the closed loop mechanism (Figure 1) where the hip of an open loop mechanism is fixed on the chair. Then the velocity and acceleration of the hip are both zero. If we want to derive dynamic equation using the recursive algorithm [8] then we need to derive differential kinematic equation for hip position. Then as a result, we know constraints such as velocity and acceleration of joints 1 and 2 have some relation (zero). By applying these kinematic constraint equations to dynamic equations and simplifying them, we get new dynamic equations for closed loop mechanism. Joint torque 1 and 2 means reaction forces acting at an ankle and knee in this case. But you can see that this process is pretty messy if DOF of a mechanism becomes higher and higher. But the method presented in this paper is pretty neat. The method uses *animation space* transformations which is used in deriving closed loop dynamic equations.

Let us make an animation space for the closed loop mechanism the same with subsection 4.3.1. in order to include hip constraints. We then derive differential kinematic equations $[G^u_\phi]$ and $[H^u_{\phi\phi}]$ and derive the dynamic equation in terms of animation space. Then we get $\vec{T}_u{}^{I}$. The meaning of $\vec{T}_{x_3}{}^{I}$ and $\vec{T}_{y_3}{}^{I}$ becomes hip reaction forces in horizontal and vertical direction, and $\vec{T}_{\phi_3}{}^{I}$ is the same with $\vec{T}_{\phi_3}{}^{I}$ which is derived in joint space $\vec{\phi}$. Then we discard $\vec{T}_{x_3}{}^{I}$ and $\vec{T}_{y_3}{}^{I}$, and use only $\vec{T}_{\phi_3}{}^{I}$ for forward dynamic simulation. Note that we are very flexible in selecting an animation space. We can include any function if meaningful for our animation and use it to constrain the motion. The *constraint* do not mean fixed constraint in geometry. It can be any thing.

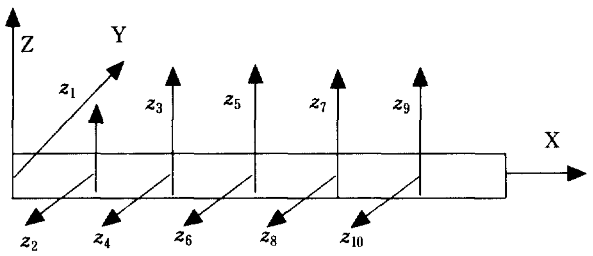
Now we shall work in general case. Basically the open loop case and closed loop case can be handled in the same way in terms of our *animation space* transformation technique. The only difference is the constraints assigned as inputs to the system. If the dimensions of $\vec{\phi}$ and $\vec{u}$ are the same, the problem is easy. Usually the dimension of $\vec{u}$ is smaller than that of $\vec{\phi}$, so we need to use optimization techniques [3, 7, 22] and need to use different approaches to solve the closed loop equation. We have tried several optimization techniques, including joint velocity optimization with weights and weighted joint torque optimization. Among these weighted joint velocity optimization gave the best results in our simulations. Also we can easily adjust joint weights to modify the resulting motion.

Let $f : \Phi \rightarrow U$ and $\vec{u} \in U$, $\vec{\phi} \in \Phi$. $\vec{u} = (f_1 f_2 \cdots f_m)^T$, $\vec{\phi} = (\phi_1 \phi_2 \cdots \phi_n)^T$. Then $[G^u_\phi]$ and $[H^u_{\phi\phi}]$ are derived from the differential relationships. If f_i is a linear combination of

ϕ_i in $\vec{\phi}$, then $[H^{f_i}_{\phi\phi}]$ becomes zero.

In order to solve in terms of $\vec{\phi}$, we need to decide redundant and nonredundant joint sets. Then we solve equations in terms of non-redundant joints by eliminating redundant joints. We then solve for the unknown end-effector forces and eliminate the forces from the equation. Thus we can do forward dynamic simulation in terms of non-redundant joints. The angular values of redundant joints can be calculated from non-redundant joints.

4.3.4 Mechanism used for Inchworm Animation



(Figure 2) Segmented Inchworm

Now we will introduce the specific mechanism used in inchworm modeling. The mechanism is very important because some redundant mechanisms produce many singular configurations. We used two DOF joints at each links because these joints give fewer singular configurations than the three degree of freedom ball joints. The mechanism used for modeling an inchworm is shown in (Figure 2). The inchworm has a very flexible body with a great many degrees of freedom. We map the inchworm body to a finite segmented articulated body. Before creating a $\vec{u}$ space, we need to check the degrees of freedom of the mechanism so that we may decide the dimension for our space. The open loop case is pretty easy, and the closed loop case is determined by Grubler's equation [15]. Let us assume one DOF at each joint, which means we assume there are two joints at each segment. Then the DOF for the closed loop is determined by the equation *closed loop* $d.o.f = 6(n-1)-5j$ where n is the number of one DOF links [15]. From this, we can decide the upper limit on the dimension of $\vec{u}$ space.

5. Dynamic Simulation

Using the motion equations developed above, we have done a dynamic simulation of an open loop serial chain, an inchworm and a pin-pointed dropping chain. The pin-pointed dropping chain is a famous simple example which we used to check the correct operation of our system. For inverse dynamic simulations, we used ***animation space*** transformation method to get differential kinematic equations for the control of the motion.

The inchworm example given in Section 4.2 illustrates the basic steps of our method of motion control. These steps are :

① Define a model of the object to be controlled for animation.
② Formulate the equations for motion control in the *animation space* transformation basis.
③ Solve the inverse dynamics problem to obtain torque functions.
④ Execute the forward dynamics simulation of motion, both open loop and closed loop steps, using the information contained in the torque functions.

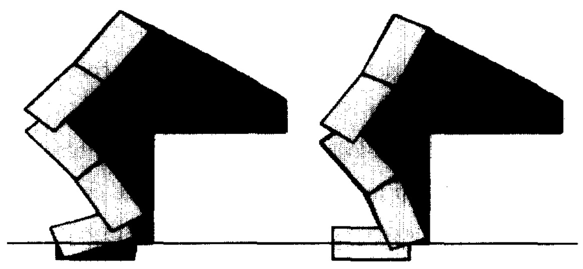
Note that we still need forward dynamic simulation because we need to in clude external forces/torques during simulation.

The core of dynamic simulation is as follows :

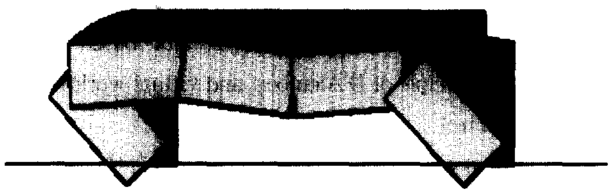
a. Calculate kinematic information with current $\vec{\phi}$
b. Determine $\vec{\dot{u}}$ and $\vec{\ddot{u}}$.
c. Calculate $\vec{\dot{\phi}}$, $\vec{\ddot{\phi}}$ using $\vec{\dot{u}}$, $\vec{\ddot{u}}$.
d. Calculate torque set by inverse dynamics.
e. Do forward dynamic simulation.
f. Obtain a new $\vec{\phi}$, and go to step a.

Note that the motion control is applied at $\vec{\dot{u}}$ and $\vec{\ddot{u}}$ starting from the initial configuration.

5.1 Dynamic Simulation of Open Loop Constrained Mechanism



(Figure 3) Normal and Constrained Motion


(Figure 4) Closed Loop Mechanism

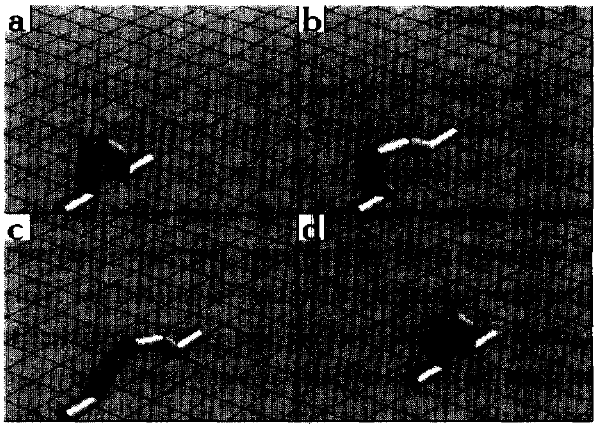
A dynamic simulation on a simple open loop mechanism is shown in (Figure 3). The mechanism has ten DOF and the control is only position based. So we have seven redundant DOF in this example. The redundancy is solved, as was discussed previously, with a joint velocity optimization technique used in robotics [3, 7, 22]. This optimization technique uses a Lagrange multipliers, which adds artificial constraints by reducing the null space. But we stress that this technique alone cannot control the shape of the body. This is why we use *animation space* transformations for our animation. The left simulation is normal movement, but the right simulation is constrained such that the first segment is fixed. As can be seen from (Figure 3), the segment is completely disabled. But when a position is given which cannot be reached with the constrained mechanism, the fixed segment eventually moves to reach it. This is a kind of inverse kinematic problem which was given unreachable position as an input. This is a very simple control example of our *animation space* transformation technique. We set $f_i = \phi_2^2$ and controlled by $\dot{f}_i = \ddot{f}_i = 0$.

5.2 Dynamic Simulation of an Inchworm

The main purpose of using inverse dynamics is to get a set of torque functions which can be used for forward dynamic simulation. An inchworm has legs only at front and rear. It lacks middle pairs of legs, so it moves by extending the front part of the body while the rear grasps the ground, and then pulling the rear forward while the front grasps the ground [1, 4]. When an inchworm extends the front part, it can be modeled as an open loop mechanism. When it pulls the rear forward, it can be modeled as a closed loop mechanism. In the case of the closed loop, there is only a single direction sliding motion at the rear part of the worm.

The inchworm model controls its own motion, subject to the intermediate goal position constraints provided by the animator. Motion planning is done by the worm to find the best body positions, favorable torques etc. For this we used a heuristic which checks a finite number of potential directions to move at each moment and selects the best move among them. (The easiest way to animate inchworm is let end of an inchworm to reach a point on a ground. That is, there is a controller which checks the distance between the next intermediate goal and the tip of an inchworm. And it determines $\dot{\vec{u}}$ and $\ddot{\vec{u}}$. Then using the *animation space* transformation, we get $\vec{\phi}$, $\dot{\vec{\phi}}$ and $\ddot{\vec{\phi}}$. By inverse dynamics, we get a set of joint torque values and do forward dynamic simulation. If there are any external forces caused by another figures during the simulation, the original planned motion is corrupted. So the motion planner modifies its original motion planning at the next time instance such that it could satisfy the next intermediate goal. For this animation, animator just need to give a set of intermediate goals as inputs. Because it can be simply asked to reach the goal position, the movement continues. In this process, we can calculate slope ($\vec{u}$) of displacement from current position to the goal position. $\ddot{\vec{u}}$ also can be calculated by finite difference. In terms of pulling rear part of the inchworm, we may constrain it to stay on the ground.)

Because we simulate an articulated body with high DOF, it is possible to fall into near singular configurations. We tried several methods for handling this including singular value decomposition (SVD) [23] and the damping technique [6]. Some optimization schemes cannot be used with the singular value decomposition method, so we



(Figure 5) Dynamic Simulation of an

incorporated damping factors in calculating the pseudo-inverse Jacobian matrix. The damping approach involves adding some values at places needed for calculating the inverse. These adjustments can cause small errors, which we have not found to be significant. The damping technique has proven quite effective in handling singular configurations.

To change steps, we need to change the base of the worm from one end to the other. We need only two DOF at each segment for motion, but the position of a segment is specified with three DOF. Since there is a segment length constraint between joints, we can change steps with only two degrees of freedom. (Figure 5) shows several frames from the dynamic simulation of an inchworm. A videotape of an "inchworm race" has been produced to better illustrate this simulation. In the simulation, we used two additional constraint functions to control the shape of the body. These functions are of the form $f(\vec{\phi}) = (\phi_i - \alpha)^2$ where α is a function of time.

For numerical integration, we tried Runge-Kutta fourth order adaptive size control and the Stoer-Bulrish method [23]. Both methods seem to have the same computing power. We used the Stoer-Bulrish method for these results. For a normal inchworm we assumed 0.005m and 0.001Kg for each segment. The system is implemented in C. The core of the dynamic routines is about 4000 lines. The computation time varies depending on the number of degrees of freedom. The inchworm simulation takes about three seconds on IBM RS/6000 320 for one step of the inverse dynamic computation. We store only key frame data and display after the whole computation.

6. Conclusion

In this paper, we have provided a new technique for inverse dynamic simulation of both closed and open loop systems. The technique is based on an extension of generalized coordinate transformations, which allows the use of an animation space with arbitrary functions of kinematic parameters defining its dimensions in place of the normal Cartesian workspace. This provides a flexible technique for designing the control interface for many types of articulated body models. Using this technique, we have given three approaches for the analysis of a closed loop, and we have used the constraint control technique on both closed and open loop mechanisms. We used a method based on the generalized principle of D'Alembert and virtual work for our dynamic control.

The complexity of our algorithm is $O(n^3)$ (with some optimization of the tensor multiplication), which is expensive compared to the linear complexity of some iterative dynamic formulations [3]. But usually linear complexity formulations, including Newton-Euler, are not well-suited to the design of control interfaces. Our dynamic formulation is good for the analysis of mechanisms, and we can use linear complexity algorithms after the initial analysis for the forward dynamic simulation of the models. A particularly attractive structural feature of this formulation is that the equations for models with a given branching topology are quite similar. This makes it very easy to derive equations for new models once some experience with the method is obtained. Our initial experience with this technique indicates that it is a very promising method for the development and analysis of open and closed loop dynamic models as well as for inverse dynamic control. Also the *animation space* transformation technique is very useful for motion control in kinematic level only. Because most animation still depends on kinematic level motion control, *we can easily transform / constrain the resulting motion given an input motion.* Further research experience will be required to understand how to determine good functions for generating the desired motion and how well the approach adapts to more complex modeling situations. The next step is to go from moving inchworms to retargetting walking humans, insects, octopi, Martians, etc. Since our animation space dimensions are not simple geometric quantities, it can be difficult to get a set of control functions by intuition. As the dimension of $\vec{u}$ space grows, these problems become more difficult.

We have also experimented with the *finite difference method of two point boundary value problem* of Witkin & Kass [28] and a human jumping animation using musculo-tendon-skeletal dynamics and weak variant of an *Optimal Control* technique. *Finite difference method of two point boundary value problem* involves making a continuous time system completely discrete getting a set of nonlinear equations. An index function is used to organize these nonlinear equations so that they have some physical meaning. This approach achieves computational efficiency by giving up accuracy in modeling the physical world. Optimal control finds the best motion which satisfies all boundary conditions. This technique is very computation intensive because it does forward dynamic simulation while finding the best motion trajectory. We have produced a worm-racing movie generated using the [28]'s method to provide a comparison with the same animation using our technique. The low quality compact movie can be downloaded from the site *http : //user.chollian.net/~shark 102/anim.avi*.

References

[1] Americana. "The Encylopedia Americana, chapter Measuring Worm," Grolier Inc., 1990.

[2] W. Armstrong, M. Green, and R. Lake. "Near-real-time control of human figure models," IEEE Computer Graphics and Applications, pp.52-61, June 1987.

[3] H. Asada and J. Slotin. "Robot Analysis and Control," John Wiley and Sons, New York, NY, 1985.

[4] Britannica. "The New Encylopedia Britannica, chapter Measuring Worm," Encylopedia Britannica Inc., 1990.

[5] A. Bruderlin and T. Calvert. "Goal-directed, dynamic animation of human walking," Computer Graphics, Vol.23, No.3, pp.233-242, July 1989.

[6] S. Chan and P. Lawrence. "General Inverse Kinematics with the Error Damped Pseudoinverse," In IEEE CH2555-1, 1988.

[7] K. Cleary. Decision "Making Software for Redundant Manipulators," PhD thesis, University of Texas at Austin, 1990.

[8] J. Craig. "Introduction to Robotics Mechanics and Control," Addison-Wesley, Reading, MA, 1986.

[9] R. Freeman. "Kinematic and Dynamic Modeling, Analysis and Control of Robotic Mechanisms," PhD thesis, University of Florida, 1985.

[10] R. Freeman. "Discussions with Dr. Freeman," 1990.

[11] R. Freeman and D. Tesar. "Dynamic Modeling of Serial and Parallel Mechanisms / Robotic Systems : Part II," Applications. In Trends and Developments in Mechanisms, Machines and Robotics, 20th Biennial Mechanisms Conference, 1988.

[12] R. Freeman and D. Tesar. "Dynamic Modeling of Serial and Parallel Mechanisms / Robotic Systems : Part I," Methodology. In Trends and Developments in Mechanisms, Machines and Robotics, 20th Biennial Mechanisms Conference, 1988.

[13] M. Gleicher. "Motion Editing with Spacetime Constraints," In Interactive 3D Graphics, pp.139-148, 1997.

[14] M. Gleicher. "Retargetting motion to new characters," In Proceedings of SIGGRAPH, pp.33-42, August 1998.

[15] K. Hunt. "Kinematic Geometry of Mechanisms," Cambridge University Press, Cambridge, 1978.

[16] P. Isaacs and M. Cohen. "Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics," Computer Graphics, Vol.21, No.4, pp.215-224, July 1987.

[17] P. Isaacs and M. Cohen. "Mixed methods for complex kinematic constraints in dynamic figure animation," Visual Computer, pp.296-305, April 1988.

[18] J. Lee and S. Shin. "A hierarchical approach to interactive motion editing for human-like figures," In Proceedings of SIGGRAPH, pp.39-48, August 1999.

[19] P. Lee, S. Wei, J. Zhao, and N. Badler. "Strength guided motion," Computer Graphics, Vol.24, No.4, pp.253-263, August 1990.

[20] M. McKenna and D. Zeltzer. "Dynamic simulation of autonomous legged locomotion," Computer Graphics, Vol.24, No.4, pp.29-38, August 1990.

[21] G. S. Miller. "The motion dynamics of snakes and worms," Computer Graphics, Vol.22, No.4, pp.169-178, August 1988.

[22] D. N. Nenchev. "Redundancy resolution through local optimization : A review," Journal of Robotic Systems, Vol.6, No.6, pp.769-799, 1989.

[23] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. "Numerical Recipes," Cambridge University Press, Cambridge, England, 1986.

[24] M. Thomas and D. Tesar. "Dynamic modeling of serial manipulator arms," Transactions of the ASME, Vol.104, pp.218-228, Sep. 1982.

[25] J. Wilhelms. "Using dynamic analysis for realistic animation of articulated bodies," IEEE Computer Graphics and Applications, Vol.7, No.6, pp.12-27, June 1987.

[26] J. Wilhelms. "Dynamic animation : Interaction and control," Visual Computer, Dec. 1988.

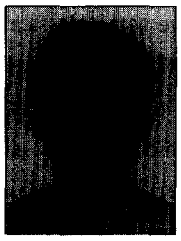
[27] J. Wilhelms. "Making them Move, chapter Dynamic Experiences," Morgan Kaufmann Publishers, 1991.

[28] A. Witkin and M. Kass. "Spacetime constraints," Computer Graphics, Vol.22, No.4, pp.159-168, August 1988.

[29] A. Witkin and Z. Popovic. "Motion warping," In Proceedings of SIGGRAPH, pp.105-108, August 1995.

박 지 헌

e-mail : jhpark@hongik.ac.kr
1983년 서울대 공대 기계설계학과(학사)
1985년 한국과학기술원 전산학과(석사)
1990년 University of Texas at Austin 전산학과(석사)
1994년 University of Texas at Austin 전산학과(박사)
1983년~1986년 한국전자통신연구원
1986년~1993년 부산외국어대학 컴퓨터 공학과 조교수
1994년~현재 홍익대학교 컴퓨터 공학과 부교수
관심분야 : 컴퓨터 그래픽스



박 성 헌

e-mail : shpark@nuri.net
1979년 서울대 공대 금속학과(학사)
1981년 한국과학기술원 경영과학(석사)
1988년 Purdue University 전산학과(석사)
1989년 Purdue University 경영정보학박사
1981년~1984년 울산공과대학교 산업공학과 전임강사
1990년~1996년 성균관대학교 경영학과 부교수
1996년~현재 명지대학교 지식정보학부 부교수
관심분야 : 정보통신기술의 응용 및 전자상거래