Proceedings of the 2000 IEEE
International Conference on Robotics & Automation
San Francisco, CA • April 2000

Operational Space Dynamics: Efficient Algorithms for Modeling and Control of Branching Mechanisms

Kyong-Sok Chang Oussama Khatib

Robotics Laboratory, Computer Science Department
Stanford University, Stanford, CA 94305, U.S.A.
{kcchang, khatib}@cs.stanford.edu

Abstract

This paper discusses intuitive and efficient ways to model and control the dynamics of highly redundant branching mechanisms using the operational space formulation. As the complexity of mechanisms increases, their modeling and control become increasingly difficult. The operational space formulation provides a natural framework for these problems since its basic structure provides dynamic decoupling among multiple tasks and posture behavior. Efficient recursive algorithms are presented for the computation of the operational space dynamics of branching mechanisms with multiple operational points. The application of these algorithms results in a significant increase in the interactivity and usability of dynamic control of complex branching mechanisms. The experimental results are presented using real-time dynamic simulation.

1 Introduction

With advances in computational and mechanical hardware technology, a growing body of interest has emerged in the area of real-time dynamic simulation and control of branching mechanisms - systems of tree-like topology (Figure 1) such as a humanoid robot (Figure 2). Application of such interest requires fast algorithms to provide intuitive, systematic, and efficient ways to model and control the dynamics of these complex systems.

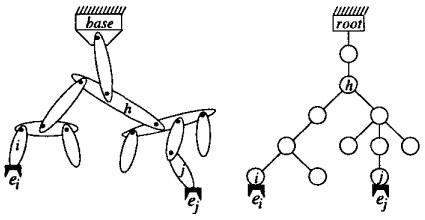

Figure 1: Branching robot with tree-like topology

The operational space formulation [6, 14] is an approach for dynamic modeling and control of a complex branching redundant mechanism at its task or end-effector level. This formulation is particularly useful for dealing with simultaneous tasks involving multiple end-effectors since its basic structure provides dynamic decoupling among the end-effectors' tasks (task behavior) and the complex internal dynamics in their associated null space (posture behavior). As an example, a humanoid robot can be controlled by directly specifying tasks evolving its feet, hands, and head while using the associated null space to achieve some desired posture behavior, e.g. balancing. This paper presents efficient recursive algorithms, using the operational space formulation, to model and solve, at the task/posture level, the dynamics problems of highly redundant articulated branching mechanisms (n links) with multiple (m) operational points. Task/posture behavior modeling under the operational space formulation and a new computationally optimized operational space control structure are developed to provide dynamically accounted operational and null space control vectors in section 2. Section 3 introduces a modified spatial notation to model complex robot kinematics and dynamics in an intuitive and systematic way. Using these control structure and notation, efficient recursive algorithms are presented in section 4. These algorithms provide dynamic control of intuitive task-level commands and posture behavior and achieve the overall complexity of $O(nm + m^3)$. Since m is a small constant in practice, these algorithms significantly increase the interactivity and usability of dynamic control of complex branching mechanisms. Finally, real-time simulation results with a humanoid robot ($n=24, m=2$) in Figure 2 are presented.

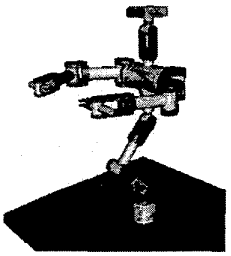

Figure 2: Robot

2 Operational Space Formulation

The *operational space formulation* [6, 14] provides dynamic modeling and control directly in the operational space by projecting the joint space dynamics into the operational space and its associated null space. The general joint space dynamic equations of motion for an n-link N-degree-of-freedom open-chain robotic mechanism with m operational points may be written in the following form [15]:

$$\mathbf{A}\,\ddot{\mathbf{q}} + \mathbf{b} + \mathbf{g} = \boldsymbol{\tau} \tag{1}$$

where $\boldsymbol{\tau}$ is the $N \times 1$ total control torque, $\ddot{\mathbf{q}}$ is the $N \times 1$ joint acceleration vector, $\mathbf{A}$ is the $N \times N$ joint space inertia matrix, $\mathbf{b}$ is the $N \times 1$ joint space Coriolis and centrifugal force vector, and $\mathbf{g}$ is the $N \times 1$ joint space gravitational force vector.
The generalized torque/force relationship [6, 14] provides the decomposition of the total control torque, $\boldsymbol{\tau}$ in Equation (1) into two dynamically decoupled control torque vectors: the torque corresponding to the task behavior command vector and the torque that only affects posture behavior in the null space:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{task} + \boldsymbol{\tau}_{posture} \tag{2}$$

Notice that this decomposition provides a natural framework for modeling and control of task/posture behavior of complex branching mechanisms.

2.1 Task Behavior

$\boldsymbol{\tau}_{task}$ in Equation (2) is the torque corresponding to the computed task behavior command vector, $\mathbf{f}_e$, acting in the operational space:

$$\boldsymbol{\tau}_{task} = \mathbf{J}_e^T\,\mathbf{f}_e \tag{3}$$

where $\mathbf{f}_e$ is a $6m \times 1$ vertically concatenated vector of the 6×1 force of each of m end-effectors and $\mathbf{J}_e$ is the $6m \times N$ vertically concatenated matrix of the $6 \times N$ Jacobian matrix of each of m end-effectors:

$$\mathbf{f}_e = \begin{bmatrix} \mathbf{f}_{e_1} \\ \vdots \\ \mathbf{f}_{e_m} \end{bmatrix} \quad \text{and} \quad \mathbf{J}_e = \begin{bmatrix} \mathbf{J}_{e_1} \\ \vdots \\ \mathbf{J}_{e_m} \end{bmatrix} \tag{4}$$

Since dynamic control of the task behavior in the operational space is desired, $\mathbf{f}_e$ can be computed using the operational space (task behavior) command vector for a linearized unit-mass system, $\mathbf{a}_e$, and the dynamics obtained by projecting the joint space dynamics into its operational space:

$$\boldsymbol{\Lambda}_e\,\mathbf{a}_e + \boldsymbol{\mu}_e + \boldsymbol{\rho}_e = \mathbf{f}_e \tag{5}$$

where the operational space inertia matrix, $\boldsymbol{\Lambda}_e$, is an $6m \times 6m$ symmetric positive definite matrix [6, 14]:

$$\boldsymbol{\Lambda}_e^{-1} = \mathbf{J}_e\,\mathbf{A}^{-1}\,\mathbf{J}_e^T \tag{6}$$

and $\boldsymbol{\rho}_e$ and $\boldsymbol{\mu}_e$ are the operational space gravitational force and Coriolis and centrifugal force vectors, respectively and defined as:

$$\boldsymbol{\rho}_e = \bar{\mathbf{J}}_e^T\,\mathbf{g} \quad \text{and} \quad \boldsymbol{\mu}_e = \bar{\mathbf{J}}_e^T\,\mathbf{b} - \boldsymbol{\Lambda}_e\,\mathbf{h}_e$$

$$\mathbf{h}_e = \frac{d}{dt}(\mathbf{J}_e)\,\dot{\mathbf{q}} \tag{7}$$

where the bias acceleration vector, $\mathbf{h}_e$, is the product of the absolute derivative of the Jacobian matrix and the $N \times 1$ joint velocity vector, $\dot{\mathbf{q}}$. $\bar{\mathbf{J}}_e$ is the dynamically consistent generalized inverse of the Jacobian matrix $\mathbf{J}_e$ (4) and has been proven to be unique [6, 14]:

$$\bar{\mathbf{J}}_e^T = \boldsymbol{\Lambda}_e\,\mathbf{J}_e\,\mathbf{A}^{-1} \tag{8}$$

2.2 Posture Behavior

$\boldsymbol{\tau}_{task}$ in Equation (2) is the torque that only affects posture behavior in the null space given any arbitrary null space (posture behavior) command vector, $\boldsymbol{\tau}_{null}$:

$$\boldsymbol{\tau}_{posture} = \mathbf{N}_e^T\,\boldsymbol{\tau}_{null} \tag{9}$$

where $\mathbf{N}_e$ is the dynamically consistent null space projection matrix that maps $\boldsymbol{\tau}_{null}$ to the appropriate control torque:

$$\mathbf{N}_e = \mathbf{1}_N - \bar{\mathbf{J}}_e\,\mathbf{J}_e \tag{10}$$

where $\mathbf{1}_N$ is an $N \times N$ identity matrix.
Dynamic consistency is the essential property for task behavior to maintain its responsiveness and to be dynamically decoupled from posture behavior since it guarantees not to produce any coupling acceleration in the operational space given any $\boldsymbol{\tau}_{null}$. In Figure 3, the robot was commanded to keep the position of both hands constant (task behavior) while rocking its torso back and forth in the null space (posture behavior). Notice that dynamic consistency enables task behavior and posture behavior to be specified independently of each other, providing an intuitive control of complex systems.

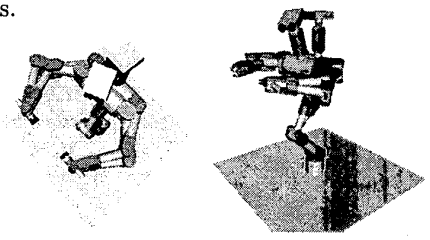

Figure 3: Posture behavior

2.3 Optimized Control Structure

The most (computationally) expensive terms in the operational space control structure (2) are the ones involving the explicit $O(n^3)$ inversion operations of the joint space inertia matrix, $\mathbf{A}$. Therefore, eliminating the explicit usage of $\mathbf{A}^{-1}$ is an essential requirement for the real-time efficiency.

In this subsection, a new computationally optimized operational space control structure is developed by completely eliminating the explicit computation of the joint space inertia matrix and its inverse. This elimination can be achieved by the combination of the dynamically accounted task/posture behavior command vectors, resulting in a dynamic control structure optimized for the computational requirement.

As in the case of the dynamic control in the operational space, since dynamic control in the null space is desired, the posture behavior command vector, τ_{null}, should account for the dynamic effects by inertial, Coriolis, centrifugal, and gravitational forces, resulting the computed posture behavior command vector:

$$\tau_{null} = \mathbf{A}\,\ddot{\mathbf{q}}_{null} + \mathbf{b} + \mathbf{g} \tag{11}$$

where $\ddot{\mathbf{q}}_{null}$ is the null space (posture behavior) command vector for a linearized unit-mass system.

Combining Equations (2), (5), (7), (10), and (11), the total control torque can be decomposed into two control torques: one with the terms related to $\mathbf{\Lambda}_e$ and the other with the terms related to $\mathbf{A}$:

$$\tau = \mathbf{J}_e^T\,\mathbf{f}_e' + \tau_{null} \tag{12}$$

$$\mathbf{f}_e' = \mathbf{\Lambda}_e\,(\mathbf{a}_e - \mathbf{h}_e - \mathbf{J}_e\,\ddot{\mathbf{q}}_{null}) \tag{13}$$

This control structure reveals physical insight about the dynamic consistency of the null space projection matrix: the extra accelerations in the operational space generated by the dynamic coupling effect of the computed posture behavior command vector, τ_{null}, are being offset by $\mathbf{J}_e\,\ddot{\mathbf{q}}_{null}$ in Equation (13).

It should be emphasized that the posture behavior command vector for a linearized unit-mass system, $\ddot{\mathbf{q}}_{null}$ in Equations (11) and (13), is not guaranteed to be achieved since not all null space motion is possible without affecting the motion of the operational space points. Instead, this control structure provides the closest solution to achieve $\ddot{\mathbf{q}}_{null}$ without generating any coupling acceleration effect in the task behavior in its operational space while minimizing the instantaneous kinetic energy in the system. This behavior is an example of the *dynamic consistency.*

Given the task and posture behavior command vectors, $\mathbf{a}_e$ and $\ddot{\mathbf{q}}_{null}$, the unknown quantities in Equations (12) and (13) are $\mathbf{J}_e$, $\mathbf{h}_e$, $\mathbf{\Lambda}_e$, and τ_{null}. Section 4 presents efficient recursive algorithms to compute these quantities. The next section introduces the notation used to derive these algorithms to produce the total control torque, τ in Equation (12).

3 Spatial Notation and Quantities

3.1 Spatial Notation

Spatial notation has been widely used in the modeling of kinematics and dynamics of complex robotic mechanisms [4, 13, 8, 9, 10, 12]. The modified spatial notation and quantities developed in this section combines various versions of existing spatial notations and conventional vector notations [5, 3, 7] in order to utilize the results from various researchers in a unified way. In this modified spatial notation, each quantity incorporates the appropriate linear (placed in upper or upper-left corners) and angular (placed in lower or lower-right corners) components and results in a concise form (6×1 vector or 6×6 matrix).

For example, a spatial velocity, $\mathbf{v}_i$, and a spatial force, $\mathbf{f}_i$, of link i are defined as:

$$\mathbf{v}_i = \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} \quad \text{and} \quad \mathbf{f}_i = \begin{bmatrix} f_i \\ \mathcal{N}_i \end{bmatrix}$$

where v_i, ω_i, f_i, and $\mathcal{N}_i$, are 3×1 linear velocity, angular velocity, force, and moment vectors expressed in frame i, respectively. Also, the spatial inertia matrix of link i in frame c_i, $\mathbf{I}_{c_i}$, is a 6×6 symmetric positive definite matrix and defined as:

$$\mathbf{I}_{c_i} = \begin{bmatrix} M_i & \mathbf{0} \\ \mathbf{0} & I_{c_i} \end{bmatrix}, \quad M_i = m_i\,\mathbf{1}_3 \tag{14}$$

where $\mathbf{1}_3$ is a 3×3 identity matrix, m_i is the mass of link i, and I_{c_i} is the 3×3 inertia tensor matrix of link i in frame c_i. The origin of frame c_i is located at the center of mass of link i shown in Figure 4.

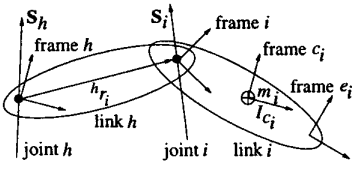

Figure 4: Basic notation

Note that, in Figure 4, the origin of each link frame is located at the joint and any variable without the reference frame number (front superscript) is expressed in its own frame. Also, if link i is a leaf (outermost) link, end-effector frame e_i is located at the tip (operational point) of link i (see Figure 1).

The general joint model, $\mathbf{S}_i$, is a $6 \times n_i$ matrix with full column rank, n_i, when joint i has n_i degrees of freedom ($n_i \leq 6$) [4, 9]. Its columns (unit vectors) make up a basis for the motion space of joint i. Notice that this matrix is constant since it is expressed in its own frame. For example, if joint i is a prismatic joint along y-axis and joint j is a spherical joint, their corresponding general joint models are:

$$\mathbf{S}_i = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T \quad \text{and} \quad \mathbf{S}_j = \begin{bmatrix} \mathbf{0} & \mathbf{1}_3 \end{bmatrix}^T$$

The 6×6 spatial transformation matrix, ${}^h_i\mathbf{X}$, transforms a spatial quantity from frame i to frame h:

$${}^h_i\mathbf{X} = \begin{bmatrix} {}^h_iR & \mathbf{0} \\ \widehat{{}^hr_i}\,{}^h_iR & {}^h_iR \end{bmatrix} \tag{15}$$

where ${}^h_iR$ is the 3×3 rotation matrix and $\widehat{{}^hr_i}$ is the cross-product operator (3×3 skew-symmetric matrix) associated with ${}^hr_i$, the 3×1 position vector from the origin of frame h to the origin of frame i expressed in frame h shown in Figure 4.
For example, by spatially transforming $\mathbf{I}_{c_i}$ (14) from frame c_i to frame i, the 6×6 spatial inertia matrix of link i in frame i can be computed as:

$$\mathbf{I}_i = {}^i_{c_i}\mathbf{X}\,\mathbf{I}_{c_i}\,{}^i_{c_i}\mathbf{X}^T \tag{16}$$

where $\mathbf{I}_i$ is a symmetric positive definite matrix since Equation (16) encapsulates the spatial counterpart of the similarity transformation and the parallel axis theorem [5, 3] for $\mathbf{I}_{c_i}$.

3.2 Spatial Quantities

The spatial velocity of link i is can be recursively computed in terms of the spatial velocity of its parent link and its joint velocity:

$$\mathbf{v}_i = {}^h_i\mathbf{X}^T\,\mathbf{v}_h + \mathbf{S}_i\,\dot{q}_i \;,\; (\mathbf{v}_{root} = \mathbf{0}) \tag{17}$$

Similarly, the spatial acceleration can be recursively computed in terms of the spatial acceleration of its parent link, its joint acceleration, and the cross-product of velocity vectors:

$$\mathbf{a}_i = {}^h_i\mathbf{X}^T\,\mathbf{a}_h + \mathbf{S}_i\,\ddot{q}_i + \mathbf{c}_i \;,\; (\mathbf{a}_{root} = \mathbf{0}) \tag{18}$$

$$\mathbf{c}_i = \boldsymbol{\omega}_i \times \mathbf{v}_i - {}^h_i\mathbf{X}^T(\boldsymbol{\omega}_h \times \mathbf{v}_h) + \mathbf{v}_i \times \mathbf{S}_i\,\dot{q}_i \tag{19}$$

where $\boldsymbol{\omega}_i$ is the spatial angular velocity of link i, $\boldsymbol{\omega}_i = \begin{bmatrix} 0 & \omega_i^T \end{bmatrix}^T$ and $\times$ is the spatial analogue of the cross-product operator. The spatial cross-product operator associated with a spatial vector $\mathbf{d} = [a^T\,b^T]^T$ is defined as:

$$\mathbf{d} \times = \begin{bmatrix} a \\ b \end{bmatrix} \times = \begin{bmatrix} \widehat{b} & \widehat{a} \\ \mathbf{0} & \widehat{b} \end{bmatrix}$$

The articulated-body inertia matrix of link h, $\mathbf{I}^A_h$, relates the spatial force and acceleration of a link, taking into account the dynamics of the rest of the articulated body [4, 13, 9] and defined as:

$$\mathbf{I}^A_h = \mathbf{I}_h + \sum_i \left[{}^h_i\mathbf{L}\,\mathbf{I}^A_i\,{}^h_i\mathbf{L}^T \right] \;,\; (\mathbf{I}^A_{leaf} = \mathbf{I}_{leaf}) \tag{20}$$

where $\mathbf{I}_h$ is the spatial inertia matrix (16) of link h and i represents the index of each child link of link h. The force propagator, ${}^h_i\mathbf{L}$, propagates a spatial force from link i to its parent link h across the actuated joint i in a dynamically consistent manner [9] similar to the null space projection matrix, $\mathbf{N}_e$ in Equation (10) and defined as:

$${}^h_i\mathbf{L} = {}^h_i\mathbf{X}\left[\mathbf{1}_6 - \mathbf{S}_i\,\overline{\mathbf{S}}_i\right]^T \tag{21}$$

$$\overline{\mathbf{S}}_i = \mathbf{D}_i^{-1}\,\mathbf{S}_i^T\,\mathbf{I}_i^A \quad , \quad \mathbf{D}_i = \mathbf{S}_i^T\,\mathbf{I}_i^A\,\mathbf{S}_i$$

where $\mathbf{1}_6$ is a 6×6 identity matrix, $\overline{\mathbf{S}}_i$ is the dynamically consistent generalized inverse of $\mathbf{S}_i$ weighted by the corresponding inertia matrix similar to $\overline{\mathbf{J}}_e$ in Equation (8), and $\mathbf{D}_i$ is the $n_i \times n_i$ full rank matrix projecting $\mathbf{I}^A_i$ onto the motion space of joint i with n_i degrees of freedom.

4 Efficient Recursive Algorithms

This section describes efficient recursive algorithms for the computation of the operational space dynamics of n-link branching mechanisms with m operational points. These algorithms are for the computations of $\mathbf{J}_e$, $\mathbf{h}_e$, $\boldsymbol{\Lambda}_e$, and $\boldsymbol{\tau}_{null}$ from the computationally optimized control structure (12) presented in section 2.3. The proposed algorithms are shown to be of $O(nm + m^3)$ overall complexity.

4.1 Jacobian Matrix

The $6 \times N$ Jacobian matrix, $\mathbf{J}_i$, relates the $N \times 1$ joint velocity vector, $\dot{\mathbf{q}}$, to the spatial velocity of link i:

$$\mathbf{v}_i = \mathbf{J}_i\,\dot{\mathbf{q}} \tag{22}$$

Using the spatial transformation matrix, the recursive equation (17) of the spatial velocity can be converted to a summation form:

$$\mathbf{v}_i = \sum_k \left[{}^k_i\mathbf{X}^T\,\mathbf{S}_k\,\dot{q}_k \right] \tag{23}$$

where k is the indices of the links in the path from link i to the root link. Then, from Equations (22) and (23), $\mathbf{J}_i$ can be written as:

$$\mathbf{J}_i = \begin{bmatrix} \cdots & \jmath_k & \cdots \end{bmatrix} \tag{24}$$

$$\jmath_k = \begin{cases} {}^k_i\mathbf{X}^T\,\mathbf{S}_k & \text{if } k \text{ is an ancestor of } i \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{25}$$

Now, since the spatial velocity of the operational point associated with the leaf link i is $\mathbf{v}_{e_i} = {}^{i}_{e_i}\mathbf{X}^T \mathbf{v}_i$, its associated Jacobian matrix can be written as:

$$\mathbf{J}_{e_i} = {}^{i}_{e_i}\mathbf{X}^T \mathbf{J}_i$$

4.2 Bias Acceleration Vector

Using Equations (7) and (22), the 6×1 bias acceleration vector, $\mathbf{h}_i$ of link i can be given as:

$$\mathbf{h}_i = \frac{d}{dt}(\mathbf{J})_i \dot{\mathbf{q}} = \mathbf{a}_i - \mathbf{J}_i \ddot{\mathbf{q}} \qquad (26)$$

An efficient method is to compute $\mathbf{h}_i$ recursively without using the absolute derivative of the Jacobian matrix explicitly. The recursive equation of the spatial acceleration (18) can be converted to a summation form:

$$\mathbf{a}_i = \sum_k \left[{}^{k}_{i}\mathbf{X}^T \mathbf{S}_k \ddot{q}_k \right] + \sum_k \left[{}^{k}_{i}\mathbf{X}^T \mathbf{c}_k \right] \qquad (27)$$

where k is the indices of the links in the path from link i to the root link. Then, from Equations (24), (25), (26) and (27), $\mathbf{h}_i$ can be written as:

$$\mathbf{h}_i = \sum_k \left[{}^{k}_{i}\mathbf{X}^T \mathbf{c}_k \right]$$

where k is the indices of the links in the path from link i to the root link. The corresponding recursive equation is:

$$\mathbf{h}_i = {}^{h}_{i}\mathbf{X}^T \mathbf{h}_h + \mathbf{c}_i \ , \ (\mathbf{h}_{root} = \mathbf{0}) \qquad (28)$$

Now, since the spatial acceleration of the operational point associated with the leaf link i is $\mathbf{a}_{e_i} = {}^{i}_{e_i}\mathbf{X}^T \mathbf{a}_i$, its associated $\mathbf{h}_{e_i}$ can be written as:

$$\mathbf{h}_{e_i} = {}^{i}_{e_i}\mathbf{X}^T \mathbf{h}_i$$

4.3 Operational Space Inertia Matrix

The inverse of the operational space inertia matrix, $\mathbf{\Lambda}_e^{-1}$, relates the forces at the end-effectors to the accelerations at the end-effectors:

$$\mathbf{a}_e = \mathbf{\Lambda}_e^{-1} \mathbf{f}_e \qquad (29)$$

Note that since $\mathbf{\Lambda}_e$ is a function of configuration only (6), $\dot{\mathbf{q}} = \mathbf{0}$ can be assumed for the analysis of $\mathbf{\Lambda}_e$ without loss of generality. Also, since $\mathbf{\Lambda}_e^{-1}$ is symmetric, it can be expressed in terms of its 6×6 block matrix components as:

$$\mathbf{\Lambda}_e^{-1} = \begin{bmatrix} \mathbf{\Lambda}^{-1}_{e_1,e_1} & \cdots & \mathbf{\Lambda}^{-1}_{e_1,e_m} \\ \vdots & \ddots & \vdots \\ \mathbf{\Lambda}^{-T}_{e_1,e_m} & \cdots & \mathbf{\Lambda}^{-1}_{e_m,e_m} \end{bmatrix} \qquad (30)$$

In this subsection, we will briefly discuss the $O(nm + m^3)$ recursive algorithm to compute $\mathbf{\Lambda}_e$ presented in [2]. This algorithm was developed by separately analyzing the inertial effects of the block diagonal matrices, $\mathbf{\Lambda}^{-1}_{e_i,e_i}$, and of the block off-diagonal matrices, $\mathbf{\Lambda}^{-1}_{e_i,e_j}$ $(i \neq j)$, in Equation (30).

Since $\mathbf{\Lambda}^{-1}_{e_i,e_i}$ is the inertial quantity that would occur if link i is the only leaf link with an end-effector, $\mathbf{\Lambda}^{-1}_{e_i,e_i}$ can be computed using an extension of the Force Propagation Method, an $O(n)$ recursive algorithm to compute the 6×6 inverse operational space inertia matrix of a single serial-chain mechanism [13, 8, 9]. $\mathbf{\Lambda}^{-1}_{e_i,e_j}$ $(i \neq j)$ may be regarded as cross-coupling inertial quantities that are a measure of the inertia coupling to the i^{th} end-effector from the force of the j^{th} end-effector via the *nearest common ancestor*[1] of leaf links i and j. From this physical property, we can compute $\mathbf{\Lambda}^{-1}_{e_i,e_j}$ as an inertial quantity which propagates the spatial forces from the j^{th} end-effector to link h (the nearest common ancestor of leaf links i and j) and then propagates the resulting spatial accelerations of link h to the i^{th} end-effector.

Table 1 summarize this efficient $O(nm+m^3)$ algorithm to recursively compute the operational space inertia matrix $\mathbf{\Lambda}_e$ for branching robotic mechanisms without any explicit computation of $\mathbf{A}^{-1}$ [2]. Note that although most processing occurs along the paths from the root link to the leaf links with end-effectors, the effects of the other links enter through the articulated-body inertias (see Equation (20)) of the links in the paths.

Figure 5 illustrates the recursion processes in Table 1 for the branching robot shown in Figure 1. Arrows indicate the direction of the recursion. Also, there is no computation required among the nodes connected by dotted lines.

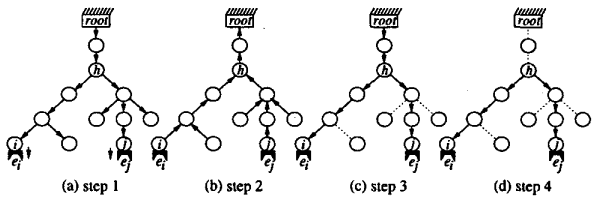

Figure 5: Recursion processes of steps in Table 1: (a) outward recursion for ${}^{h_i}_{i}\mathbf{X}$, (b) inward recursion for ${}^{h_i}_{i}\mathbf{L}$, (c) outward recursion for $\mathbf{\Omega}_{i,i}$, and (d) outward recursion for $\mathbf{\Omega}_{i,j}$ $(i \neq j)$

[1] The *nearest common ancestor* of links i and j is the first common link in two paths; one from link i to the root link and the other from link j to the root link. For example, in Figure 1, link h is the nearest common ancestor of leaf links i and j.

Table 1: $O(mn+m^3)$ recursive algorithm for $\mathbf{\Lambda}_e$.

1. **Outward Recursion:** Compute ${}^{h_i}_{i}\mathbf{X}$ (15)
2. **Inward Recursion:** Compute ${}^{h_i}_{i}\mathbf{L}$ (21)
3. **Outward Recursion:** Compute the block diagonal matrices starting with $\mathbf{\Omega}_{root,root} = \mathbf{0}$:

$$\mathbf{\Omega}_{i,i} = \mathbf{S}_i \mathbf{D}_i^{-1} \mathbf{S}_i^T + {}^{h_i}_{i}\mathbf{L}^T \mathbf{\Omega}_{h_i,h_i} {}^{h_i}_{i}\mathbf{L}$$

4. **Outward Recursion:** Compute the block off-diagonal matrices with nearest common ancestor h of links i and j:

$$\mathbf{\Omega}_{i,j} = \begin{cases} \text{return} & \text{if } i = j = h \\ {}^{h_i}_{i}\mathbf{L}^T \mathbf{\Omega}_{j,h_i}^T & \text{else if } j = h \\ \mathbf{\Omega}_{i,h_j} {}^{h_j}_{j}\mathbf{L} & \text{otherwise} \end{cases}$$

5. **Spatial Transformation:** Compute $\mathbf{\Lambda}^{-1}_{e_i,e_j}$ from $\mathbf{\Omega}_{i,j}$ of leaf links with end-effectors:

$$\mathbf{\Lambda}^{-1}_{e_i,e_j} = {}^{i}_{e_i}\mathbf{X}^T \mathbf{\Omega}_{i,j} {}^{j}_{e_j}\mathbf{X}$$

6. **Matrix Inversion:** Compute the extended operational space inertia matrix, $\mathbf{\Lambda}_e$, by inverting $\mathbf{\Lambda}_e^{-1}$ (30).

Table 2: $O(n)$ recursive Newton-Euler method

1. **Given:** The joint space command vector ($\ddot{q}_i$) and the external force ($\mathbf{f}_i^{ext}$)
2. **Outward Recursion:** Compute the spatial net force and gravitational force:

$$\begin{aligned} \mathbf{f}_i^{net} &= {}^{i}_{c_i}\mathbf{X}\, \mathbf{f}_{c_i}^{net} = \mathbf{I}_i\, \mathbf{a}_i + \mathbf{p}_i \\ \mathbf{p}_i &= {}^{i}_{c_i}\mathbf{X}\, (\omega_{c_i} \times \mathbf{I}_{c_i}\, \mathbf{v}_{c_i}) - \mathbf{I}_i\, (\omega_i \times \mathbf{v}_i) \end{aligned}$$

3. **Inward Recursion:** Compute the spatial total force and joint force:

$$\begin{aligned} \mathbf{f}_{h_i} &= \mathbf{f}_{h_i}^{net} - \mathbf{f}_{h_i}^{ext} + \sum_i \left[{}^{h_i}_{i}\mathbf{X}\, \mathbf{f}_i \right] \\ &\quad \left(\mathbf{f}_{leaf} = \mathbf{f}_{leaf}^{net} - \mathbf{f}_{leaf}^{ext} \right) \\ \tau_{h_i} &= \mathbf{S}_{h_i}^T\, \mathbf{f}_{h_i} \end{aligned}$$

4.4 Recursive Newton-Euler Method

Computing τ_{null} in Equation (11), is the classical joint space inverse dynamics problem that can be solved by the $O(n)$ recursive Newton-Euler method [11, 4], the most efficient currently known general method for calculating joint space inverse dynamics. Table 2 summarizes the $O(n)$ recursive Newton-Euler method using the modified spatial notation presented in section 3.

4.5 Computational Complexity

The Jacobian matrix in section 4.1, can be computed in $O(nm)$ since each of m $\mathbf{J}_{e_i}$ requires one inward recursion involving at most n links and the complexity of a recursion step between a child and its parent links is $O(1)$. In section 4.2, since all $\mathbf{h}_i$ can be computed in $O(n)$ using the recursive equation (28) which requires only one outward recursion, $\mathbf{h}_e$ can be computed in $O(n)$. For the computation of the operational inertia matrix, $\mathbf{\Lambda}_e$, an efficient $O(nm+m^3)$ algorithm from [2] is used in section 4.3. Also, since the recursive Newton-Euler method in section 4.4 takes $O(n)$, τ_{null} in equation (11) can be computed in $O(n)$.

Therefore, using the algorithms presented in this section, $\mathbf{J}_e$, $\mathbf{\Lambda}_e$, $\mathbf{h}_e$, and τ_{null} can be computed in $O(nm+m^3)$. Then, the total control torque vector, τ in Equation (12) requires $O(nm+m^3)$ computational complexity using the optimized dynamic control structure (12) presented in section 2.3. Note that since m can be considered as a small constant, $m = O(1)$, for any realistic robotic mechanism in practice, we obtain the linear running time of $O(n)$ for this algorithm. Thus, as the number of links in a mechanism increases, the proposed algorithm performs significantly better than the existing symbolic method [14] which still requires $O(n^3)$ inversion operations of $\mathbf{A}^{-1}$ in Equations (6) and (8).

5 Experimental Results

Using the proposed algorithms, we were able to perform the computation of the operational space dynamics for the branching robotic mechanism in Figure 2 in less than 1.2 msec using a PC with a 266 MHz Pentium II running under the QNX real-time operating system. This branching robot has an operational point at each of its 2 end-effectors and 24 links connected by one-degree-of-freedom joints.

This result implies that the proposed algorithms enable highly redundant articulated robotic mechanisms such as a humanoid mechanism with multiple operational points to be controlled directly at the task/posture level with a high servo rate in a low-cost hardware environment while providing dynamic decoupling among the end-effectors' task behavior and the complex internal posture behavior dynamics in the associated null space, resulting intuitive task/posture behavior specifications for users.

In order to show the effectiveness of the proposed algorithms, we also have controlled this robot under the operational space formulation using the proposed algorithms. In the real-time dynamic simulation environment developed in our laboratory [1], we have achieved

a servo rate of 300 Hz for dynamic control and simulation of this robot with the setup above.
Figure 6 shows the motion sequence when this robot was commanded to touch the floor with its left end-effector and maintain the position and orientation of its right end-effector constant. In addition, the robot was being advised to keep its posture the same as the initial configuration. Notice that the robot had to adjust its advised posture behavior in the null space without producing any coupling acceleration at both end-effectors in order not to violate the task behavior. This was done automatically without any additional commands.

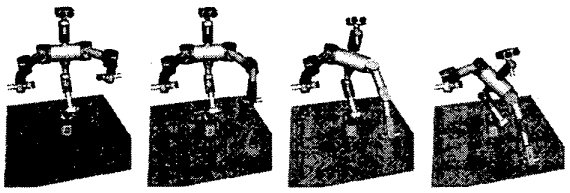

Figure 6: Motion Sequence

6 Conclusion

We have presented efficient recursive algorithms, using the operational space formulation, to model and solve, at the task/posture behavior level, the dynamics problems of highly redundant articulated branching mechanisms (n links) with multiple (m) operational points. A computationally optimized operational space control structure is developed to provide dynamically accounted operational space and null space command vectors, while eliminating any explicit computation of joint space inertia matrix and its inverse. Using this control structure, efficient recursive algorithms are presented in order to provide dynamic control of intuitive task-level commands and posture behavior. The proposed algorithms achieve the overall complexity of $O(nm + m^3)$. Since m is a small constant in practice, the application of these algorithms results in a significant increase in the interactivity and usability of dynamic control of complex branching mechanisms. The real-time simulation results with a humanoid robot ($n = 24$, $m = 2$) illustrate the efficiency of these algorithms for intuitive task/posture behavior control.

Acknowledgments

Many thanks to Oliver Brock, Herman Bruyninckx, Robert Holmberg, Oscar Madrigal, Diego Ruspini, and H.F. Machiel Van der Loos for their comments and support during the preparation of this manuscript.

References

[1] K.-S. Chang. *Robotics Library.* Computer Science Robotics Laboratory, Stanford University, Stanford, California, U.S.A., 2nd edition, 1998. A dynamic control and simulation library.

[2] K.-S. Chang and O. Khatib. Efficient algorithm for extended operational space inertia matrix. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 350–355, Kyongju, Korea, October 1999.

[3] J. J. Craig. *Introduction to Robotics.* Addison-Wesley Publishing Company, second edition, 1989.

[4] R. Featherstone. *Robot Dynamics Algorithms.* Kluwer Academic Publishers, 1987.

[5] D. T. Greenwood. *Principles of Dynamics.* Prentice-Hall Inc., second edition, 1988.

[6] O. Khatib. A unified approach to motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, RA-3(1):43–53, February 1987.

[7] O. Khatib. The impact of redundancy on the dynamic performance of robots. *Laboratory Robotics and Automation*, 8:37–48, 1996.

[8] K. Kreutz-Delgado, A. Jain, and G. Rodriguez. Recursive formulation of operational space control. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1750–1753, April 1991.

[9] K. W. Lilly. *Efficient Dynamic Simulation of Robotic Mechanisms.* Kluwer Academic Publishers, 1992.

[10] K. W. Lilly and D. E. Orin. Efficient O(n) recursive computation of the operational space inertia matrix. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(5):1384–1391, September/October 1993.

[11] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. On-line computational scheme for mechanical manipulators. *Transactions of ASME Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, June 1980.

[12] B. V. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, University of California at Berkeley, Berkeley, California, U.S.A., December 1996.

[13] G. Rodriguez, K. Kreutz, and A. Jain. A spatial operator algebra for manipulator modeling and control. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1374–1379, May 1989.

[14] J. Russakow, O. Khatib, and S. M. Rock. Extended operational space formulation for serial-to-parallel chain (branching) manipulators. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1056–1061, Nagoya, Japan, May 1995.

[15] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Transactions of ASME Journal of Dynamic Systems, Measurement, and Control*, 104(3):205–211, September 1982.