Robot Dynamics: Equations and Algorithms

Roy Featherstone
Department of Computer Science
University of Wales, Aberystwyth
Penglais, Aberystwyth SY23 3DB, Wales, UK

David Orin
Department of Electrical Engineering
Ohio State University
Columbus, OH 43210-1272, USA

Abstract

This paper reviews some of the accomplishments in the field of robot dynamics research, from the development of the recursive Newton-Euler algorithm to the present day. Equations and algorithms are given for the most important dynamics computations, expressed in a common notation to facilitate their presentation and comparison.

1 Introduction

Many contributions have been made in the area of robot dynamics since the earliest work more than two decades ago. In the field of the dynamics of mechanisms, the robotics community has especially focused on the problem of computational efficiency. In fact, many of the most efficient algorithms in dynamics, that are applicable to a wide class of mechanisms, were developed by robotics researchers [23, 33, 10].

While computational efficiency continues to be important for the simulation and control of increasingly complex mechanisms operating at higher speeds, other aspects of the dynamics problem are also important. Algorithms should be formulated with a compact set of equations for ease of development and implementation. On the other hand, there should be a clear relationship between these equations and the recursive set from which the greatest computational efficiency is obtained. The use of spatial notation and spatial operator algebra [11, 29] has been very effective in this regard. Also, it is important to develop algorithms which have applicability to robotic mechanisms with general geometries and joint structures. While the early algorithms [23, 33, 10] were applicable to single, open-chain manipulators with either rotational or prismatic joints, general joint models have since been developed and applied to more complex configurations [11].

The purpose of this paper is to review some of the major contributions in robot dynamics. The equations and algorithms for the most important recursive computations will be presented in a common, concise notation in Section 3. As given, the algorithms are directly applicable to tree-structure mechanisms. The equations for closed-loop systems are then developed in a compact form, and this is followed by a discussion of global analysis techniques. Space does not permit us to include references to all of the important contributions in the field. None-the-less the most cited papers are generally included.

2 Foundational Work in Robot Dynamics

Early efforts in robot dynamics were directed to expressing the equations of motion for robot manipulators, and other single open-chain systems, in the most efficient form. Algorithms were developed for the most common computations for robot analysis, control, and simulation. In this section, emphasis will be placed on outlining the major contributions and the work most often cited. Unfortunately, space does not permit us to provide a comprehensive review of the extensive literature in the area. See some of the early books in the area for additional references [7, 11].

The classic approach to expressing the equations of motion was based on a Lagrangian formulation [18, 31] of the problem. Algorithms developed using Lagrangian dynamics were $O(N^4)$, and had to be adapted for real-time control. Efficient low-order algorithms were sought for three major computations:

1. inverse dynamics in which the required joint actuator torques/forces are computed from a specification of the manipulator's trajectory (position, velocity, and acceleration),

2. forward dynamics in which the applied joint ac-

This is an electronic version of a paper that appeared originally in Proc. IEEE Int. Conf. Robotics & Automation, San Francisco, CA, 2000, pp. 826–834. It is word-for-word identical to the original, but the formatting is slightly different.

tuators torques/forces are specified and the joint accelerations are to be determined, and

3. the manipulator inertia matrix which maps the joint accelerations to the joint forces.

Inverse dynamics is used in feedforward control, and forward dynamics is required for simulation. The inertia (mass) matrix is used in analysis, in feedback control to linearize the dynamics, and is an integral part of many forward dynamics formulations.

The first researchers to develop $O(N)$ algorithms for inverse dynamics for robotics used a Newton-Euler (NE) formulation of the problem. Stepanenko and Vukobratovic [30] developed a recursive NE method for human limb dynamics, and Orin et al. [26] made the recursive method more efficient by referring forces and moments to local link coordinates for real-time control of a leg of a walking machine. Luh, Walker, and Paul [23] developed a very efficient Recursive NE Algorithm (RNEA) by referring most quantities to link coordinates. The RNEA is the most cited. Hollerbach [16] developed an $O(N)$ recursive Lagrangian formulation, but found that it was much less efficient than the RNEA in terms of the number of multiplications and additions/subtractions required in the algorithm. Further gains have been made in efficiency over the years. The results of Balafoutis et al. [3] and He and Goldenberg [15] are representative of those that are up to a factor of 1.7 faster than early implementations of the RNEA (for a 6-DoF robot).

Walker and Orin [33] used the RNEA for inverse dynamics [23] as the basis for efficient algorithms for forward dynamics. Their Method 3, later named the Composite-Rigid-Body Algorithm (CRBA) by Featherstone [11], computed the inertial parameters of composite sets of rigid bodies at the outer end of the manipulator chain. The columns of the inertia matrix were computed very efficiently through successive application of inverse dynamics with the joint velocities set to zero, and the joint accelerations set to zero or a unit vector. Since this implies that only one joint is in motion at a time, the inverse dynamics reduces to a much simplified analysis of a base set of links in static equilibrium and a composite rigid body in motion at the outer end of the chain. Because of the need to solve a linear system of equations whose size grows with N, the algorithm was $O(N^3)$. For small N, the first-order terms dominated the computation so that the result was quite efficient.

The earliest known $O(N)$ algorithm for forward dynamics was developed by Vereshchagin [32]. This algorithm uses a recursive formula to evaluate the Gibbs-Appel form of the equation of motion, and is applicable to unbranched chains with revolute and prismatic joints. The recursive formula was obtained via dynamic programming techniques. This algorithm closely resembles the Articulated-Body Algorithm (ABA), but the paper was way ahead of its time and languished in obscurity for a decade. Later, Armstrong developed an $O(N)$ algorithm for mechanisms with spherical joints [1], and then Featherstone developed the ABA [10]. The first version of this algorithm was applicable to manipulators with single-degree-of-freedom joints, but the second included a general joint model and was faster [11]. In terms of the total number of arithmetic operations required, the ABA was more efficient than the CRBA for $N > 9$ [11]. Also, using similar efficient transformations and link coordinates as Featherstone [11, 10], Brandl et al. [8] made further improvements on the ABA so that it was roughly comparable to the CRBA for $N = 6$. Further gains have been made in efficiency over the years, with McMillan and Orin [24] being representative of those that have reduced the computation (another 15% reduction).

The efficiency of the CRBA was directly related to the efficiency of computing the joint space inertia (mass) matrix [33]. Featherstone [11] used efficient transformations and link coordinates to reduce the computation of the inertia matrix by about 30%. A number of other gains have been made over the years [4, 25] giving an overall improvement of close to a factor of two from that of [33]. Lilly and Orin [22] developed four methods for computation of the inertia matrix. Their Modified Composite Rigid Body Method includes computation of the manipulator Jacobian so that it is very efficient for computing the inertia matrix in operational space.

Khatib [20] developed an operational-space formulation of robot dynamics, in which the equations are expressed in the same coordinate system that is used to command the robot: Cartesian coordinates and orientation of the end-effector. This formulation has been particularly successful in hybrid motion/force control and related applications [21].

Rodriguez [28] recognized the parallels between the concepts and techniques of Kalman filtering and the forward dynamics problem, and developed the spatial operator algebra framework for the study of multibody dynamics. This enabled him and others at JPL [29] to develop alternative factorizations of the mass matrix to derive the ABA. Jain [17] used the spatial operator algebra framework to provide a unified formulation for manipulator dynamics. From this, he was able to compare the various $O(N^3)$, $O(N^2)$, and $O(N)$ algorithms that had been previously published. Ascher,

Pai, and Cloutier [2] used the spatial operator framework to unify the derivation of both the CRBA and the ABA, as two elimination methods to solve the same linear system. They also showed that the ABA is more accurate than the CRBA.

The above works are all concerned with rigid-body dynamics, and are therefore applicable whenever a robot mechanism can be adequately modelled by a rigid-body system. Some forms of non-rigid behaviour, like compliance in the joint bearings, are relatively easy to incorporate into a rigid-body model; but elastic links are more complicated. This problem was addressed by Book [6], who developed an efficient, recursive Lagrangian formulation (using 4×4 matrices) of both inverse and forward dynamics for serial chains with flexible links. A general modal formulation of elastic displacement was used.

3 Equations and Algorithms

This section presents the main algorithms and associated equations. For the sake of brevity, equations are written in spatial notation; but readers who are not familiar with this notation should still be able to follow the material.

Spatial vectors are 6×1 vectors containing both the linear and angular components of physical quantities like velocity, acceleration and force. For technical reasons, they are separated into two vector spaces: motion-type vectors in M^6 and force-type vectors in F^6. Tensor quantities, like inertia, are represented using 6×6 matrices. The underlying algebra is a dual system of vector spaces. A brief description of the current version of spatial algebra can be found in the appendix of [12], and a detailed description of the previous version in [11].

3.1 The Recursive Newton-Euler Algorithm

A general robot mechanism with tree structure can be modelled by a set of N movable links (rigid bodies), numbered $1 \ldots N$, a fixed base link, numbered 0, and a set of N joints that connect between the links so that joint i connects from link $\lambda(i)$ to link i, where $\lambda(i)$ is the link number of the parent of link i in the tree, taking the base link as the root node. The numbers are chosen so that $\lambda(i) < i$. In the special case of an unbranched kinematic chain, $\lambda(i) = i-1$ and the links and joints are numbered consecutively from the base to the tip.

If we let $\mathbf{v}_i$ be the velocity of link i, and $\mathbf{v}_i^J$ be the velocity across joint i then

$$\mathbf{v}_i^J = \mathbf{v}_i - \mathbf{v}_{\lambda(i)} \,. \tag{1}$$

The joint velocity can also be described in the form

$$\mathbf{v}_i^J = \mathbf{h}_i \, \dot{\mathbf{q}}_i \,, \tag{2}$$

where $\mathbf{h}_i$ is a $6 \times d_i$ matrix spanning the motion freedom subspace of joint i, $\dot{\mathbf{q}}_i$ is a $d_i \times 1$ vector of joint velocity variables, and d_i is the degree of freedom (DoF) of joint i. In the special case of a 1-DoF joint, $\mathbf{h}_i$ is a vector describing the joint's axis of motion.

Combining Eqs. 1 and 2 produces

$$\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{h}_i \, \dot{\mathbf{q}}_i \,, \tag{3}$$

which is the standard recursive formula for computing link velocities. The equivalent formula for accelerations is just the time-derivative of Eq. 3:

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \dot{\mathbf{h}}_i \, \dot{\mathbf{q}}_i + \mathbf{h}_i \, \ddot{\mathbf{q}}_i \,, \tag{4}$$

where $\mathbf{a}_i$ is the acceleration of link i and $\ddot{\mathbf{q}}_i$ is a vector of joint acceleration variables.[1] For revolute and prismatic joints, and many other special cases,

$$\dot{\mathbf{h}}_i = \mathbf{v}_i \times \mathbf{h}_i \,.$$

Given the velocity and acceleration of the base, $\mathbf{v}_0$ and $\mathbf{a}_0$, these formulas calculate the velocity and acceleration of each link in turn, working outward from the base to the terminal links. A typical algorithm looks like this:

for $i = 1$ **to** N **do**
  $\mathbf{v}_i = \mathbf{v}_{\lambda(i)} + \mathbf{h}_i \, \dot{\mathbf{q}}_i$;
  $\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \dot{\mathbf{h}}_i \, \dot{\mathbf{q}}_i + \mathbf{h}_i \, \ddot{\mathbf{q}}_i$
end

The property $\lambda(i) < i$ ensures that $\mathbf{v}_{\lambda(i)}$ is calculated before $\mathbf{v}_i$.

The equation of motion for link i is

$$\mathbf{f}_i + \mathbf{f}_i^x = \mathbf{I}_i \, \mathbf{a}_i + \mathbf{v}_i \times \mathbf{I}_i \, \mathbf{v}_i \,, \tag{5}$$

where $\mathbf{I}_i$ is the spatial inertia of link i (a 6×6 matrix), $\mathbf{f}_i$ is the net force applied to link i through the joints, and $\mathbf{f}_i^x$ is the sum of all other forces acting on link i. This equation combines Newton's and Euler's

[1] This equation uses spatial accelerations. The majority of published works use standard linear and angular accelerations instead, either separately (*e.g.* [23]) or in a 6-D notation (*e.g.* spatial operator algebra). The difference is explained in [11, §2.7]. Papers that use non-spatial accelerations have different expressions in Eqs. 4 and 5.

equations for the linear and angular motion of a rigid body.

$\mathbf{f}_i^x$ may include contributions from springs, dampers, force fields, contact with the environment, and so on; but its value is assumed to be known, or at least to be calculable from known quantities (in which case the calculation of $\mathbf{f}_i^x$ is considered to be a separate problem). $\mathbf{f}_i^x$ may also include the effect of gravity on link i; but the special case of a uniform gravitational field can be simulated most efficiently by imparting a fictitious acceleration to the base: if $\mathbf{g}$ is the gravitational acceleration vector then add $-\mathbf{g}$ to $\mathbf{a}_0$ [23].

If we define $\mathbf{f}_i^J$ to be the force transmitted from link $\lambda(i)$ to link i through joint i, then

$$\mathbf{f}_i = \mathbf{f}_i^J - \sum_{j \in \mu(i)} \mathbf{f}_j^J \tag{6}$$

where $\mu(i)$ is the set of children of link i:

$$\mu(i) = \{\, j \mid \lambda(j) = i \,\}\,.$$

Combining Eqs. 5 and 6 produces

$$\mathbf{f}_i^J = \mathbf{I}_i\,\mathbf{a}_i + \mathbf{v}_i \times \mathbf{I}_i\,\mathbf{v}_i - \mathbf{f}_i^x + \sum_{j \in \mu(i)} \mathbf{f}_j^J\,, \tag{7}$$

which is a recursive formula for calculating joint forces, starting at the terminal links and working towards the base. A typical implementation of this formula looks like this:

for $i = 1$ **to** N **do**
  $\mathbf{f}_i^J = \mathbf{I}_i\,\mathbf{a}_i + \mathbf{v}_i \times \mathbf{I}_i\,\mathbf{v}_i - \mathbf{f}_i^x$
end
for $i = N$ **to** 1 **do**
  if $\lambda(i) \neq 0$ **then** $\mathbf{f}_{\lambda(i)}^J = \mathbf{f}_{\lambda(i)}^J + \mathbf{f}_i^J$
end

The final step is to extract the $d_i \times 1$ vector of joint force variables, $\boldsymbol{\tau}_i$, from the spatial vector of joint forces. This is done by

$$\boldsymbol{\tau}_i = \mathbf{h}_i^T\,\mathbf{f}_i^J\,. \tag{8}$$

Equations 3, 4, 7 and 8 together form the RNEA for a general tree-structure mechanism.

For maximum efficiency, the equations should be evaluated in link coordinates. If we assign a coordinate system to each link, and represent each spatial vector in the coordinates of the link to which it refers, then Eqs. 3, 4 and 7 need to include coordinate transforms at appropriate places in the expressions. The modified equations are

$$\mathbf{v}_i = {}^i\mathbf{X}_{\lambda(i)}^M\,\mathbf{v}_{\lambda(i)} + \mathbf{h}_i\,\dot{\mathbf{q}}_i\,,$$

$$\mathbf{a}_i = {}^i\mathbf{X}_{\lambda(i)}^M\,\mathbf{a}_{\lambda(i)} + \dot{\mathbf{h}}_i\,\dot{\mathbf{q}}_i + \mathbf{h}_i\,\ddot{\mathbf{q}}_i$$

and

$$\mathbf{f}_i^J = \mathbf{I}_i\,\mathbf{a}_i + \mathbf{v}_i \times \mathbf{I}_i\,\mathbf{v}_i + \sum_{j \in \mu(i)} {}^i\mathbf{X}_j^F\,\mathbf{f}_j^J\,,$$

where ${}^i\mathbf{X}_{\lambda(i)}^M$ and ${}^i\mathbf{X}_j^F$ are coordinate transformation matrices for motion-type and force-type vectors, respectively.

The complete Newton-Euler algorithm, in link coordinates, looks like this:

for $i = 1$ **to** N **do**
  $\mathbf{v}_i = {}^i\mathbf{X}_{\lambda(i)}^M\,\mathbf{v}_{\lambda(i)} + \mathbf{h}_i\,\dot{\mathbf{q}}_i$;
  $\mathbf{a}_i = {}^i\mathbf{X}_{\lambda(i)}^M\,\mathbf{a}_{\lambda(i)} + \dot{\mathbf{h}}_i\,\dot{\mathbf{q}}_i + \mathbf{h}_i\,\ddot{\mathbf{q}}_i$;
  $\mathbf{f}_i^J = \mathbf{I}_i\,\mathbf{a}_i + \mathbf{v}_i \times \mathbf{I}_i\,\mathbf{v}_i - \mathbf{f}_i^x$
end
for $i = N$ **to** 1 **do**
  $\boldsymbol{\tau}_i = \mathbf{h}_i^T\,\mathbf{f}_i^J$;
  if $\lambda(i) \neq 0$ **then** $\mathbf{f}_{\lambda(i)}^J = \mathbf{f}_{\lambda(i)}^J + {}^{\lambda(i)}\mathbf{X}_i^F\,\mathbf{f}_i^J$
end

3.2 The Composite-Rigid-Body Algorithm

As explained in [33, 11], if we express the equation of motion of a tree-structure rigid-body system in the form

$$\mathbf{M}\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau}\,,$$

where $\ddot{\mathbf{q}} \in \mathsf{M}^n$ is the vector of generalized accelerations, $\boldsymbol{\tau} \in \mathsf{F}^n$ is the vector of generalized forces, $\mathbf{M} : \mathsf{M}^n \mapsto \mathsf{F}^n$ is the system mass matrix (joint-space inertia matrix), and $\mathbf{C} \in \mathsf{F}^n$ contains all of the acceleration-independent terms, then

$$\begin{aligned} \mathbf{M}\,\ddot{\mathbf{q}} &= \mathbf{D}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) - \mathbf{D}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{0}) \\ &= \mathbf{D}(\mathbf{q}, \mathbf{0}, \ddot{\mathbf{q}}) - \mathbf{D}(\mathbf{q}, \mathbf{0}, \mathbf{0})\,, \end{aligned}$$

where $\mathbf{D}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is an inverse dynamics calculation function. The velocity argument can be set to zero because the velocity terms cancel. (Gravity and $\mathbf{f}^x$ terms also cancel.) This equation immediately gives us a simple algorithm for calculating $\mathbf{M}$:

$$\mathbf{M}\,\boldsymbol{\delta}_i = \mathbf{D}(\mathbf{q}, \mathbf{0}, \boldsymbol{\delta}_i) - \mathbf{D}(\mathbf{q}, \mathbf{0}, \mathbf{0})\,, \qquad i = 1 \ldots n\,,$$

where $\boldsymbol{\delta}_i$ is an $n \times 1$ vector with a 1 in row i and zeros elsewhere. The expression $\mathbf{M}\,\boldsymbol{\delta}_i$ is column i of $\mathbf{M}$. This algorithm is method 1 in [33].

We may infer from this equation the following physical interpretation of $\mathbf{M}$: in a system where all of the velocities and acceleration-independent forces are zero,

column i of $\mathbf{M}$ is the generalized force vector that causes an acceleration of $\boldsymbol{\delta}_i$.

If every joint has one DoF then there is a $1:1$ correspondence between joint numbers and column index numbers; and we may interpret $\boldsymbol{\delta}_i$ as a unit acceleration of joint i, and each individual element $\mathbf{M}_{ji}$ of $\mathbf{M}$ as the force required at joint j to produce the acceleration $\boldsymbol{\delta}_i$.

If the system contains multi-DoF joints then we treat $\mathbf{M}$ as a block matrix—a matrix whose elements are themselves matrices—and arrange for block-column i to be the group of d_i real columns that correspond to the d_i acceleration variables of joint i. Similar treatment is given to the rows of $\mathbf{M}$ and to composite vectors such as $\ddot{\mathbf{q}}$ and $\boldsymbol{\tau}$. This allows us to use the same notation for single- and multi-DoF joints.

The CRBA is a fast algorithm for calculating $\mathbf{M}$, and it starts with the following observation: if the zero-velocity system mentioned above is given an acceleration of $\boldsymbol{\delta}_i$ then the whole subtree rooted at link i is behaving like a single (composite) rigid body, with acceleration equal to $\mathbf{h}_i$, and the rest of the system is in static equilibrium. The elements of (block) column i of $\mathbf{M}$ can therefore be calculated according to the following algorithm.

1. Let C_i be the composite rigid body comprising all of the links in the subtree rooted at link i, and let $\mathbf{I}_i^C$ be the inertia of C_i.

2. Let $\mathbf{f}_i^C$ be the force required to impart an acceleration of $\mathbf{h}_i$ to C_i. This force is transmitted to C_i through joint i; but since link $\lambda(i)$ is in static equilibrium, meaning that there is no net force acting on it, the force transmitted through joint $\lambda(i)$ must be the same as the force through joint i. By the same argument, every joint on the path between the base and link i transmits $\mathbf{f}_i^C$.

3. Each element $\mathbf{M}_{ji}$, where j is a joint on the path between the base and link i, is given by $\mathbf{M}_{ji} = \mathbf{h}_j^T\,\mathbf{f}_i^C$ (cf. Eq. 8).

4. Each element $\mathbf{M}_{ji}$, where $j < i$ but is not on the path between the base and link i, is zero.

5. Elements $\mathbf{M}_{ji}$, where $j > i$, are given by $\mathbf{M}_{ji} = \mathbf{M}_{ij}^T$ because $\mathbf{M}$ is symmetric.

The inertia of a composite rigid body is simply the sum of the inertias of its component parts, so

$$\mathbf{I}_i^C = \sum_{j\in\nu(i)} \mathbf{I}_j\,,$$

where $\nu(i)$ is the set of link numbers of every link in the subtree; but it is more efficient to use the recursive formula

$$\mathbf{I}_i^C = \mathbf{I}_i + \sum_{j\in\mu(i)} \mathbf{I}_j^C\,. \tag{9}$$

The force exerted on C_i is

$$\mathbf{f}_i^C = \mathbf{I}_i^C\,\mathbf{h}_i\,,$$

and the force transmitted through joint j $(j \le i)$ is

$$\mathbf{f}_i^C(j) = \begin{cases} \mathbf{f}_i^C & \text{if } i \in \nu(j) \\ \mathbf{0} & \text{otherwise;} \end{cases}$$

so the complete expression for $\mathbf{M}$ is

$$\mathbf{M}_{ji} = \begin{cases} \mathbf{h}_j^T\,\mathbf{I}_i^C\,\mathbf{h}_i & \text{if } i \in \nu(j) \\ \mathbf{h}_j^T\,\mathbf{I}_j^C\,\mathbf{h}_i & \text{if } j \in \nu(i) \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{10}$$

Equations 9 and 10 express the CRBA in absolute coordinates. This algorithm does run faster in link coordinates (unless n is large [11]), and a typical implementation might look like this:

$\mathbf{M} = \mathbf{0}$;
for $i = 1$ **to** N **do** $\mathbf{I}_i^C = \mathbf{I}_i$;
for $i = N$ **to** 1 **do**
  $\mathbf{f}_i^C(i) = \mathbf{I}_i^C\,\mathbf{h}_i$;
  for each $j \in \nu(i)$ **do** $\mathbf{M}_{ij} = \mathbf{h}_i^T\,\mathbf{f}_j^C(i)$;
  if $\lambda(i) \neq 0$ **then**
    $\mathbf{I}_{\lambda(i)}^C = \mathbf{I}_{\lambda(i)}^C + {}^{\lambda(i)}\mathbf{X}_i^F\,\mathbf{I}_i^C\,{}^i\mathbf{X}_{\lambda(i)}^M$;
    for each $j \in \nu(i)$ **do**
      $\mathbf{f}_j^C(\lambda(i)) = {}^{\lambda(i)}\mathbf{X}_i^F\,\mathbf{f}_j^C(i)$
    end
end

3.3 The Articulated-Body Algorithm

The starting point of the ABA is the observation that the accelerations of bodies in a rigid-body system are always linear functions of the applied forces. If the motion of a rigid-body system is disturbed by the application of an external test force, $\mathbf{f}$, to any one body in the system, then the acceleration of that body can be expressed as a linear equation

$$\mathbf{a} = \boldsymbol{\Phi}\,\mathbf{f} + \mathbf{b}\,,$$

where $\mathbf{a}$ is the body's acceleration, $\mathbf{b}$ is the acceleration that it would have if $\mathbf{f}$ were zero, and $\boldsymbol{\Phi} : \mathsf{F}^6 \mapsto \mathsf{M}^6$ is a 6×6 matrix. If the motion of the chosen body

is unconstrained then $\mathbf{\Phi}$ is nonsingular, and the whole equation can be inverted to read

$$\mathbf{f} = \mathbf{I}^A \, \mathbf{a} + \mathbf{p}^A \,, \tag{11}$$

where $\mathbf{I}^A = \mathbf{\Phi}^{-1}$ and $\mathbf{p}^A = -\mathbf{I}^A \, \mathbf{b}$.

In articulated-body terminology, the whole system is called an articulated body, the one body that experiences the test force is called a handle, $\mathbf{I}^A$ is the articulated-body inertia (ABI) of the handle, and $\mathbf{p}^A$ is the corresponding bias force, which is the force required to bring the handle's acceleration to zero.

The point of this exercise is that Eq. 11 applies to any body in any system, no matter how many other bodies there are in that system, provided only that the chosen body has full motion freedom. This is the secret of the $O(N)$ complexity of the ABA.

The main step in the ABA is to calculate an ABI and bias force for each link in turn, choosing the link itself as the handle and the subtree rooted at that link as the articulated body. This can be accomplished by a computation that starts at the terminal links and works toward the base, using the following recursive formulae [11]:

$$\mathbf{I}_i^A = \mathbf{I}_i + \sum_{j \in \mu(i)} \left(\mathbf{I}_j^A - \mathbf{I}_j^A \, \mathbf{h}_j \left(\mathbf{h}_j^T \mathbf{I}_j^A \, \mathbf{h}_j \right)^{-1} \mathbf{h}_j^T \, \mathbf{I}_j^A \right), \tag{12}$$

$$\mathbf{p}_i^A = \mathbf{p}_i + \sum_{j \in \mu(i)} \left(\mathbf{p}_j^\alpha + \mathbf{I}_j^A \, \mathbf{h}_j \left(\mathbf{h}_j^T \mathbf{I}_j^A \, \mathbf{h}_j \right)^{-1} \left(\boldsymbol{\tau}_i - \mathbf{h}_j^T \mathbf{p}_j^\alpha \right) \right), \tag{13}$$

where

$$\mathbf{p}_i = \mathbf{v}_i \times \mathbf{I}_i \, \mathbf{v}_i - \mathbf{f}_i^x \,,$$
$$\mathbf{p}_j^\alpha = \mathbf{p}_j^A + \mathbf{I}_j^A \, \dot{\mathbf{h}}_j \, \dot{\mathbf{q}}_j \,.$$

Having computed these quantities, the next step is to calculate the joint accelerations. Assuming that $\mathbf{a}_{\lambda(i)}$ has already been computed, we can construct the following system of equations in the three unknowns $\mathbf{a}_i$, $\mathbf{f}_i^J$ and $\ddot{\mathbf{q}}_i$:

$$\mathbf{a}_i = \mathbf{a}_{\lambda(i)} + \dot{\mathbf{h}}_i \, \dot{\mathbf{q}}_i + \mathbf{h}_i \, \ddot{\mathbf{q}}_i \,,$$
$$\mathbf{f}_i^J = \mathbf{I}_i^A \, \mathbf{a}_i + \mathbf{p}_i^A \,,$$
$$\boldsymbol{\tau}_i = \mathbf{h}_i^T \, \mathbf{f}_i^J \,.$$

These equations can be solved for $\ddot{\mathbf{q}}_i$ to give

$$\ddot{\mathbf{q}}_i = \left(\mathbf{h}_i^T \mathbf{I}_i^A \, \mathbf{h}_i \right)^{-1} \left(\boldsymbol{\tau}_i - \mathbf{h}_i^T (\mathbf{I}_i^A \, \mathbf{a}_{\lambda(i)} + \mathbf{p}_i^\alpha) \right), \tag{14}$$

which can in turn be used to evaluate $\mathbf{a}_i$, and so on.

Equations 3, 12, 13, 14 and 4 together describe the ABA in absolute coordinates. This algorithm runs faster in link coordinates. The procedure for converting the ABA to link coordinates is the same as for the RNEA.

3.4 Closed-Loop Systems

Systems with closed kinematic loops are much more complicated than tree-structure systems, and need a lengthy explanation to do them justice; so we refer readers to the lengthy explanations in [27, 34].

The main problem is that the joint variables are no longer independent, since they are subject to (potentially very complicated) loop-closure constraints. It is usually impractical to identify an independent set of position variables, so most algorithms work with a larger, non-independent set, and take measures to enforce the constraints that exist among them.

The most common method for dealing with kinematic loops is to extract a spanning tree from the connectivity graph, compute the equations of motion for the spanning tree, and then add loop-closure forces that mimic the effects of the kinematic loops. A spanning tree is a subgraph that contains all of the nodes and a subset of the arcs that were in the original graph, and has tree connectivity. The corresponding tree-structure mechanism contains all of the bodies but only a subset of the joints that were in the original, and the loop-closure forces mimic the effects of the omitted joints.

Let us express the equation of motion of a closed-loop system as follows:

$$\mathbf{M} \, \ddot{\mathbf{q}} + \mathbf{C} = \boldsymbol{\tau} + \boldsymbol{\tau}_c \,, \tag{15}$$

where $\mathbf{M}$ and $\mathbf{C}$ refer to the spanning tree, and are computed using kinematic-tree methods, $\ddot{\mathbf{q}}$ and $\boldsymbol{\tau}$ are composite vectors containing the tree-joint acceleration and force variables, respectively, and $\boldsymbol{\tau}_c$ is the vector of tree-joint forces that mimics the effects of the missing joints. The kinematic loops impose the following constraints on the motion of the tree, and hence on the values of the tree-joint variables:[2]

$$\boldsymbol{\phi}(\mathbf{q}) = \mathbf{0} \,. \tag{16}$$

Equations 15 and 16 together provide a complete description of the dynamics, but not in a convenient form. The usual next step is to differentiate Eq. 16 twice to get an equation involving accelerations:[3]

$$\ddot{\boldsymbol{\phi}} = \boldsymbol{\phi}' \, \ddot{\mathbf{q}} + \dot{\boldsymbol{\phi}}' \, \dot{\mathbf{q}} = \mathbf{0} \,, \tag{17}$$

where $\boldsymbol{\phi}' = \partial \boldsymbol{\phi} / \partial \mathbf{q}$. In general, $\boldsymbol{\phi}'$ can be rank-deficient, and the rank can vary as a function of $\mathbf{q}$;

[2] This equation assumes scleronomic constraints. The processing of general holonomic and nonholonomic constraints is described in [27, 34].

[3] In practice, one must include constraint stabilization terms, since the differential equation $\ddot{\boldsymbol{\phi}} = \mathbf{0}$ is not numerically stable [27, 34].

so let us also define a full-rank version of Eq. 17:

$$\mathbf{L}\,\ddot{\mathbf{q}} = \mathbf{c}\,, \tag{18}$$

where $\mathbf{L}$ is an $n_c \times n$ matrix comprising n_c linearly-independent rows of $\boldsymbol{\phi}'$, $\mathbf{c}$ contains the corresponding elements of $-\dot{\boldsymbol{\phi}}'\,\dot{\mathbf{q}}$, and n_c is the rank of $\boldsymbol{\phi}'$.

It can be shown that the loop-closure force can always be expressed in the form

$$\boldsymbol{\tau}_c = \mathbf{L}^T\,\boldsymbol{\lambda} + \boldsymbol{\tau}_a\,, \tag{19}$$

where $\boldsymbol{\lambda}$ is an $n_c \times 1$ vector of unknown constraint-force variables (or Lagrange multipliers), and $\boldsymbol{\tau}_a$ accounts for any active forces in the loop-closing joints. If all of these joints are passive then $\boldsymbol{\tau}_a = \mathbf{0}$.

We can now combine Eqs. 15, 18 and 19 to produce

$$\begin{bmatrix} \mathbf{M} & \mathbf{L}^T \\ \mathbf{L} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau} - \mathbf{C} + \boldsymbol{\tau}_a \\ \mathbf{c} \end{bmatrix}. \tag{20}$$

This equation, or something similar, appears at some point in most closed-loop dynamics formulations.

The next step is to solve Eq. 20, or its equivalent, for $\ddot{\mathbf{q}}$. The three main approaches are

1. solve Eq. 20 directly for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$,
2. solve for $\boldsymbol{\lambda}$ first, and then use the result to solve for $\ddot{\mathbf{q}}$, or
3. solve Eq. 18 (or Eq. 17) for $\ddot{\mathbf{q}}$, substitute the result into Eq. 15, eliminate the unknown constraint forces, and solve for the remaining unknowns.

Method 1 is the simplest, but generally also the least efficient. As the size of the system matrix is $(n+n_c)\times(n+n_c)$, this method is $O((n+n_c)^3)$.

Method 2 is particularly useful if $n \gg n_c$, and offers the opportunity to use $O(n)$ algorithms on the spanning tree [5]. From Eq 20,

$$\mathbf{L}\,\mathbf{M}^{-1}\,\mathbf{L}^T\,\boldsymbol{\lambda} = \mathbf{c} - \mathbf{L}\,\mathbf{M}^{-1}\,(\boldsymbol{\tau} - \mathbf{C} + \boldsymbol{\tau}_a)\,. \tag{21}$$

This equation can be formulated in $O(n\,n_c^2)$ operations via $O(n)$ algorithms, and solved in $O(n_c^3)$. Once $\boldsymbol{\lambda}$ is known, $\boldsymbol{\tau}_c$ can be calculated in $O(n\,n_c)$ operations, and Eq. 15 solved by an $O(n)$ algorithm; so the total complexity is $O(n\,n_c^2 + n_c^3)$.

Method 3 is useful if $n-n_c$ is small and/or n_c has to be determined at run time. A special version of Gaussian elimination (or similar procedure), equipped with a numerical rank test and designed to solve underdetermined systems, is applied directly to Eq. 17 to get

$$\ddot{\mathbf{q}} = \mathbf{K}\,\mathbf{y} + \ddot{\mathbf{q}}_0\,,$$

where $\mathbf{y}$ is a vector of $n - n_c$ unknowns (typically a linearly-independent subset of the elements of $\ddot{\mathbf{q}}$), $\ddot{\mathbf{q}}_0$ is any particular solution to Eq. 17, and $\mathbf{K}$ is an $n \times (n - n_c)$ matrix with the property $\mathbf{L}\,\mathbf{K} = \mathbf{0}$. Substituting this expression for $\ddot{\mathbf{q}}$ into Eq. 15, and premultiplying both sides by $\mathbf{K}^T$ to eliminate the $\mathbf{L}^T\,\boldsymbol{\lambda}$ component of $\boldsymbol{\tau}_c$, produces

$$\mathbf{K}^T\mathbf{M}\,\mathbf{K}\,\mathbf{y} = \mathbf{K}^T(\boldsymbol{\tau} - \mathbf{C} + \boldsymbol{\tau}_a - \mathbf{M}\,\ddot{\mathbf{q}}_0)\,. \tag{22}$$

This method also has cubic complexity, but it can be the most efficient if $n - n_c$ is small. It is also reported to be more stable than Method 1 [9].

3.5 Global Analysis Techniques

It is possible to express the equation of motion of a system of N independent rigid bodies in the form

$$\mathbf{f} = \mathbf{I}\,\mathbf{a} + \mathbf{v} \times \mathbf{I}\,\mathbf{v}\,, \tag{23}$$

where

$$\mathbf{f} = [\mathbf{f}_1^T, \ldots, \mathbf{f}_N^T]^T \in \mathsf{F}^{6N},$$
$$\mathbf{a} = [\mathbf{a}_1^T, \ldots, \mathbf{a}_N^T]^T \in \mathsf{M}^{6N},$$
$$\mathbf{I} = \mathrm{diag}(\mathbf{I}_1, \ldots, \mathbf{I}_N) : \mathsf{M}^{6N} \mapsto \mathsf{F}^{6N},$$

and so on. Composite vectors and matrices like these are the starting point of the spatial operator algebra developed by Rodriguez et al. [28, 29, 17]. We give them the spatial-algebra treatment by introducing the two vector spaces M^{6N} and F^{6N}, which are Cartesian products of N lots of M^6 and F^6, respectively.

Suppose that the above system is subjected to kinematic constraints that confine the (instantaneous) velocity to an n-dimensional subspace $S \subset \mathsf{M}^{6N}$. It follows that the acceleration is constrained by

$$\mathbf{a} - \mathbf{a}_0 \in S \subset \mathsf{M}^{6N},$$

where $\mathbf{a}_0$ is a vector of (known) velocity-product terms. If $\mathbf{S}$ is a $6N \times n$ matrix with the property $\mathrm{Range}(\mathbf{S}) = S$, then

$$\mathbf{a} = \mathbf{a}_0 + \mathbf{S}\,\boldsymbol{\alpha}\,, \tag{24}$$

where $\boldsymbol{\alpha}$ is a vector of unknowns.

These kinematic constraints will impose certain forces on the system. Let us define the total applied force as $\mathbf{f} = \mathbf{f}_a + \mathbf{f}_c$, where $\mathbf{f}_a$ is a vector of known active forces and $\mathbf{f}_c$ a vector of unknown constraint forces. The equation of motion of the constrained system is therefore

$$\mathbf{f}_a + \mathbf{f}_c = \mathbf{I}\,\mathbf{a} + \mathbf{v} \times \mathbf{I}\,\mathbf{v}\,. \tag{25}$$

By the principle of virtual work, $\mathbf{S}^T \mathbf{f}_c = \mathbf{0}$; so if we substitute for $\mathbf{a}$ using Eq. 24 and premultiply the result by $\mathbf{S}^T$ then we get

$$\mathbf{S}^T \mathbf{I}\, \mathbf{S}\, \boldsymbol{\alpha} = \mathbf{S}^T (\mathbf{f}_a - \mathbf{I}\, \mathbf{a}_0 - \mathbf{v} \times \mathbf{I}\, \mathbf{v}) \,. \qquad (26)$$

This equation has the same structure as Eq. 22, because they both use the same method for applying constraints and differ only in the system they start with: Eq. 22 starts with a kinematic tree, and Eq. 26 with a system of unconstrained rigid bodies. The matrices in Eq. 26 are bigger than those in Eq. 22, but they have a sparse structure that is useful for describing low-order algorithms.

If we also introduce a subspace $T \subset \mathsf{F}^{6N}$ to describe the space of all possible constraint forces arising from the motion constraints, and a $6N \times (6N - n)$ matrix $\mathbf{T}$ satisfying $\mathrm{Range}(\mathbf{T}) = T$, then

$$\mathbf{f}_c = \mathbf{T}\, \boldsymbol{\beta} \,, \qquad \mathbf{T}^T \mathbf{S} = \mathbf{0} \,, \qquad \mathbf{T}^T \mathbf{a} = \mathbf{T}^T \mathbf{a}_0 \,.$$

Combining these equations with Eq. 25 produces

$$\begin{bmatrix} \mathbf{I} & \mathbf{T} \\ \mathbf{T}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ -\boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_a - \mathbf{v} \times \mathbf{I}\, \mathbf{v} \\ \mathbf{T}^T \mathbf{a}_0 \end{bmatrix}, \qquad (27)$$

which is obviously analogous to Eq. 20; and solving for $\boldsymbol{\beta}$ produces

$$\mathbf{T}^T \mathbf{I}^{-1}\, \mathbf{T}\, \boldsymbol{\beta} = \mathbf{T}^T \mathbf{a}_0 - \mathbf{T}^T \mathbf{I}^{-1} \left(\mathbf{f}_a - \mathbf{v} \times \mathbf{I}\, \mathbf{v}\right), \qquad (28)$$

which is analogous to Eq. 21.

Suppose that the constraints embodied in Eq. 24 are due to a set of joints that connect the bodies together into a kinematic tree, as described in Section 3.1. In this case, Eq. 24 is a global expression of the individual joint constraints

$$\mathbf{a}_i - \mathbf{a}_{\lambda(i)} = \mathbf{h}_i\, \ddot{\mathbf{q}}_i + \dot{\mathbf{h}}_i\, \dot{\mathbf{q}}_i \,, \quad i = 1 \ldots N \,.$$

This equation can be expressed in global form as

$$\mathbf{P}\, \mathbf{a} = \mathbf{H}\, \ddot{\mathbf{q}} + \dot{\mathbf{H}}\, \dot{\mathbf{q}} \,, \qquad (29)$$

where $\mathbf{H} = \mathrm{diag}(\mathbf{h}_i)$, and $\mathbf{P}$ is the incidence matrix,[4] defined as follows:

$$\mathbf{P}_{ij} = \begin{cases} \mathbf{1}_{6\times 6} & : \; j = i \\ -\mathbf{1}_{6\times 6} & : \; j = \lambda(i) \\ \mathbf{0}_{6\times 6} & : \; \text{otherwise.} \end{cases}$$

This matrix is sparse, lower-triangular and (trivially) invertible. Its inverse corresponds to the matrix ϕ^* in [29].

[4] The elements of a standard incidence matrix are the scalars $+1$, -1 and 0, not 6×6 matrices. See [27, 34]. (Their incidence matrix S corresponds to $-\mathbf{P}^T$.)

Putting Eq. 29 into the same form as Eq. 24, we get $\mathbf{S} = \mathbf{P}^{-1}\, \mathbf{H}$ and $\boldsymbol{\alpha} = \ddot{\mathbf{q}}$. Substituting these expressions into Eq. 26 produces the following expression for the system mass matrix:

$$\mathbf{M} = \mathbf{H}^T\, \mathbf{P}^{-T}\, \mathbf{I}\, \mathbf{P}^{-1}\, \mathbf{H} \,,$$

which is the Newton-Euler factorization described in [29]. The operator interpretation of this equation leads directly to the RNEA, and via an operator inversion formula to the ABA.

Another $O(N)$ forward-dynamics algorithm for kinematic trees can be obtained directly from Eq. 27 by exploiting its special sparse structure: it has only $O(N)$ non-zero elements, and it has a perfect elimination order [5].

For the special case of an unbranched chain, the coefficient matrix in Eq. 28 is block-tridiagonal. This means that it can be solved in $O(\log(N))$ time on a parallel computer with $O(N)$ processors; and this is the key step in the Constraint-Force Algorithm [14, 13].

4 Conclusion

This paper has reviewed some of the major contributions made in the field of robot dynamics. The main algorithms and associated equations have been given using a common, concise notation. The minimization of computational costs has been key in developing new formulations and algorithms for control and simulation of robotic mechanisms. Roboticists may be credited with developing the most computationally-efficient, low-order algorithms for inverse dynamics, forward dynamics, and the inertia matrix.

Space did not permit us to include extensive discussion of a number of other topics that have been investigated in robot dynamics. These include automatic generation and simplification of symbolic equations of motion, algorithms for parallel computers, and applications to particular classes of systems. For a particular robot configuration, the equations of motion when expressed in symbolic form usually involve the least computation to implement. The use of Kane's equations [19] may be of value in this regard, although their use may also involve considerable manipulation of the basic equations to reduce them to the simplest form. Through the use of parallel algorithms, researchers have been able to reduce the computation of $O(N)$ algorithms to $O(\log N)$ on $O(N)$ processors. The application of robot dynamics to multilegged vehicles, biomechanical systems, spacecraft, flexible structures, and underwater robotic vehicles has extended many

of the basic algorithms to the needs of the particular area. Hopefully, as more complex systems and applications are developed, research in robot dynamics will help meet the expanding needs in performance.

5 Acknowledgments

Support for this work was provided in part by Grant No. IIS-9907121 from the National Science Foundation to The Ohio State University.

6 References

[1] W. W. Armstrong, "Recursive Solution to the Equations of Motion of an n-Link Manipulator," in *Proc. of 5th World Congress on Theory of Machines and Mechanisms*, (Montreal), pp. 1343–1346, July 1979.

[2] U. M. Ascher, D. K. Pai and B. P. Cloutier, "Forward Dynamics, Elimination Methods, and Formulation Stiffness in Robot Simulation," *Int. J. Rob. Research*, vol. 16, no. 6, pp. 749–758, 1997.

[3] C. A. Balafoutis, R. V. Patel, and P. Misra, "Efficient Modeling and Computation of Manipulator Dynamics Using Orthogonal Cartesian Tensors," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 665–676, December 1988.

[4] C. A. Balafoutis and R. V. Patel, "Efficient Computation of Manipulator Inertia Matrices and the Direct Dynamics Problem," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 1313–1321, Sept/Oct 1989.

[5] D. Baraff, "Linear-Time Dynamics using Lagrange Multipliers," Proc. SIGGRAPH '96, pp. 137–146, New Orleans, August 1996.

[6] W. J. Book, "Recursive Lagrangian Dynamics of Flexible Manipulator Arms," *Int. J. Robotics Research*, vol. 3, no. 3, pp. 87–101, 1984.

[7] M. Brady, J. M. Hollerbach, T. L. Johnson, T. Lozano-Perez, and M. T. Mason, *Robot Motion: Planning and Control*. Cambridge, MA: The MIT Press, 1982.

[8] H. Brandl, R. Johanni, and M. Otter, "A Very Efficient Algorithm for the Simulation of Robots and Similar Multibody Systems Without Inversion of the Mass Matrix," in *Proc. of IFAC/IFIP/IMACS International Symposium on Theory of Robots*, (Vienna), December 1986.

[9] R. E. Ellis and S. L. Ricker, "Two Numerical Issues in Simulating Constrained Robot Dynamics," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 24, no. 1, pp. 19–27, 1994.

[10] R. Featherstone, "The Calculation of Robot Dynamics using Articulated-Body Inertias," *Int. J. Robotics Research*, vol. 2, no. 1, pp. 13–30, 1983.

[11] R. Featherstone, *Robot Dynamics Algorithms*, Boston/Dordrecht/Lancaster: Kluwer Academic Publishers, 1987.

[12] R. Featherstone, "A Divide-and-Conquer Articulated-Body Algorithm for Parallel $O(\log(n))$ Calculation of Rigid-Body Dynamics. Part 1: Basic Algorithm," *Int. J. Robotics Research*, vol. 18, no. 9, pp. 867–875, 1999.

[13] R. Featherstone and A. Fijany, "A Technique for Analyzing Constrained Rigid-Body Systems and Its Application to the Constraint Force Algorithm," *IEEE Trans. Robotics & Automation*, vol. 15, no. 6, pp. 1140–4, 1999.

[14] A. Fijany, I. Sharf and G. M. T. D'Eleuterio, "Parallel $O(\log N)$ Algorithms for Computation of Manipulator Forward Dynamics," *IEEE Trans. Robotics & Automation*, vol. 11, no. 3, pp. 389–400, June 1995.

[15] X. He and A. A. Goldenberg, "An Algorithm for Efficient Computation of Dynamics of Robotic Manipulators," in *Proc. of Fourth International Conference on Advanced Robotics*, (Columbus, OH), pp. 175–188, June 1989.

[16] J. M. Hollerbach, "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-10, no. 11, pp. 730–736, 1980.

[17] A. Jain, "Unified Formulation of Dynamics for Serial Rigid Multibody Systems," *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 3, pp. 531–542, 1991.

[18] M. E. Kahn and B. Roth, "The Near Minimum-time Control of Open-loop Articulated Kinematic Chains," *Journal of Dynamic Systems, Measurement, and Control*, vol. 93, pp. 164–172, 1971.

[19] T. R. Kane and D. A. Levinson, "The Use of Kane's Dynamical Equations in Robotics," *Int. J. Robotics Research*, vol. 2, no. 3, pp. 3–21, 1983.

[20] O. Khatib, "A Unified Approach to Motion and Force Control of Robot Manipulators: The Operational Space Formulation," *IEEE J. Robotics & Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[21] O. Khatib, "Inertial Properties in Robotic Manipulation: An Object-Level Framework," *Int. J. Robotics Research*, vol. 14, no. 1, pp. 19–36, 1995.

[22] K. W. Lilly and D. E. Orin, "Alternate Formulations for the Manipulator Inertia Matrix," *International Journal of Robotics Research*, vol. 10, pp. 64–74, February 1991.

[23] J. Y. S. Luh, M. W. Walker and R. P. C. Paul, "On-Line Computational Scheme for Mechanical Manipulators," *Trans. ASME, J. Dynamic Systems, Measurement & Control*, vol. 102, no. 2, pp. 69–76, 1980.

[24] S. McMillan and D. E. Orin, "Efficient Computation of Articulated-Body Inertias Using Successive Axial Screws," *IEEE Trans. on Robotics and Automation*, vol. 11, pp. 606–611, 1995.

[25] S. McMillan and D. E. Orin, "Forward Dynamics of Multilegged Vehicles Using the Composite Rigid Body Method," in *Proc. of IEEE International Conference on Robotics and Automation*, (Leuven, Belgium), pp. 464–470, May 1998.

[26] D. E. Orin, R. B. McGhee, M. Vukobratovic, and G. Hartoch, "Kinematic and Kinetic Analysis of Open-chain Linkages Utilizing Newton-Euler Methods," *Mathematical Biosciences*, vol. 43, pp. 107–130, February 1979.

[27] R. E. Roberson and R. Schwertassek, *Dynamics of Multibody Systems*, Berlin/Heidelberg/New York: Springer-Verlag, 1988.

[28] G. Rodriguez, "Kalman Filtering, Smoothing, and Recursive Robot Arm Forward and Inverse Dynamics," *IEEE Journal on Robotics and Automation*, vol. RA-3, no. 6, pp. 624–639, 1987.

[29] G. Rodriguez, A. Jain and K. Kreutz-Delgado, "A Spatial Operator Algebra for Manipulator Modelling and Control," *Int. J. Robotics Research*, vol. 10, no. 4, pp. 371–381, 1991.

[30] Y. Stepanenko and M. Vukobratovic, "Dynamics of Articulated Open-chain Active Mechanisms," *Math. Biosciences*, vol. 28, pp. 137–170, 1976.

[31] J. J. Uicker, "Dynamic Force Analysis of Spatial Linkages," *Transactions of the ASME Journal of Applied Mechanics*, vol. 34, pp. 418–424, 1967.

[32] A. F. Vereshchagin, "Computer Simulation of the Dynamics of Complicated Mechanisms of Robot Manipulators," *Engineering Cybernetics*, no. 6, pp. 65–70, 1974.

[33] M. W. Walker and D. E. Orin, "Efficient Dynamic Computer Simulation of Robotic Mechanisms," *Trans. ASME, J. Dynamic Systems, Measurement & Control*, vol. 104, pp. 205–211, 1982.

[34] J. Wittenburg, *Dynamics of Systems of Rigid Bodies*, Stuttgart: B. G. Teubner, 1977.