Multimedia Contents

3. Dynamics

Roy Featherstone, David E. Orin

The dynamic equations of motion provide the relationships between actuation and contact forces acting on robot mechanisms, and the acceleration and motion trajectories that result. Dynamics is important for mechanical design, control, and simulation. A number of algorithms are important in these applications, and include computation of the following: *inverse dynamics*, *forward dynamics*, the *joint-space inertia matrix*, and the *operational-space inertia matrix*. This chapter provides efficient algorithms to perform each of these calculations on a rigid-body model of a robot mechanism. The algorithms are presented in their most general form and are applicable to robot mechanisms with general connectivity, geometry, and joint types. Such mechanisms include fixed-base robots, mobile robots, and parallel robot mechanisms.

In addition to the need for computational efficiency, algorithms should be formulated with a compact set of equations for ease of development and implementation. The use of spatial notation has been very effective in this regard, and is used in presenting the dynamics algorithms. Spatial vector algebra is a concise vector notation for describing rigid-body velocity, acceleration, inertia, etc., using six-dimensional (6-D) vectors and tensors.

The goal of this chapter is to introduce the reader to the subject of robot dynamics and to provide the reader with a rich set of algorithms, in a compact form, that they may apply to their particular robot mechanism. These algorithms are presented in tables for ready access.

3.1 Overview

Robot dynamics provides the relationships between actuation and contact forces, and the acceleration and motion trajectories that result. The dynamic equations of motion provide the basis for a number of computational algorithms that are useful in mechanical design, control, and simulation. A growing area of their use is in computer animation of mobile systems, especially using human and humanoid models. In this Chapter, the fundamental dynamic relationships for robot mechanisms are presented, along with efficient algorithms for the most common computations. Spatial vector notation, a concise representation which makes use of 6-D vectors and tensors, is used in the algorithms.

This chapter presents efficient low-order algorithms for four major computations:

1. Inverse dynamics, in which the required joint actuator torques/forces are computed from a specification of the robot's trajectory (position, velocity, and acceleration),
2. Forward dynamics in which the applied joint actuator torques/forces are specified and the joint accelerations are to be determined,
3. The joint-space inertia matrix, which maps the joint accelerations to the joint torques/forces, and
4. The operational-space inertia matrix, which maps task accelerations to task forces in operational or Cartesian space.

Inverse dynamics is used in feedforward control and trajectory planning. Forward dynamics is required for simulation. The joint-space 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 operational-space inertia matrix is used in control at the task or end-effector level.

3.1.1 Spatial Vector Notation

Section 3.2 presents the spatial vector notation, which is used to express the algorithms in this chapter in a clear and concise manner. It was originally developed by *Featherstone* [3.1] to provide a concise vector notation for describing rigid-body velocity, acceleration, inertia, etc., using 6-D vectors and tensors. Section 3.2 explains the meanings of spatial vectors and operators, and provides a detailed tabulation of the correspondence between spatial and standard three-dimensional (3-D) quantities and operators, so that the algorithms in the later sections can be understood. Formulae for efficient computer implementation of spatial arithmetic are also provided. Effort is taken in the discussion of spatial vectors to distinguish between the coordinate vectors and the quantities they represent. This illuminates some of the important characteristics of spatial vectors.

3.1.2 Canonical Equations

The dynamic equations of motion are provided in Sect. 3.3 in two fundamental forms: the joint-space formulation and the operational-space formulation. The terms in the joint-space formulation have traditionally been derived using a Lagrangian approach in which they are developed independently of any reference coordinate frame. The Lagrange formulation provides a description of the relationship between the joint actuator forces and the motion of the mechanism, and fundamentally operates on the kinetic and potential energy in the system. The resulting joint-space formulation has a number of notable properties that have proven useful for developing control algorithms. The equations to relate the terms in the joint-space and operational-space formulations, along with an impact model, are also provided in this section.

3.1.3 Dynamic Models of Rigid-Body Systems

The algorithms in this chapter are model-based and require a data structure describing a robot mechanism as one of their input arguments. Section 3.4 gives a description of the components of this model: a connectivity graph, link geometry parameters, link inertia parameters, and a set of joint models. The description of the connectivity is general so that it covers both kinematic trees and closed-loop mechanisms. Kinematic trees and the spanning tree for a closed-loop mechanism share a common notation. In order to describe

the link and joint geometry, two coordinate frames are associated with each joint, one each attached to the *predecessor* and *successor* links. These frames are defined to be compatible with the modified Denavit–Hartenberg convention of *Craig* [3.2] for serial mechanisms containing single-degree-of-freedom (DOF) joints, but are nevertheless applicable to general rigid-body systems containing general multi-DOF joints. The relationship between connected links is described using the general joint model of *Roberson* and *Schwertassek* [3.3]. A humanoid robot is given as an example to illustrate the link and joint numbering scheme, as well as the assignment of coordinate frames to describe the links and joints. The example includes a floating base, and revolute, universal, and spherical joints.

3.1.4 Kinematic Trees

The algorithms presented in Sect. 3.5 calculate the inverse dynamics, forward dynamics, joint-space inertia matrix, and operational-space inertia matrix for any robot mechanism that is a kinematic tree. An $O(n)$ algorithm for inverse dynamics is provided, where n is the number of degrees of freedom in the mechanism. It uses a Newton–Euler formulation of the problem, and is based on the very efficient recursive Newton–Euler algorithm (RNEA) of *Luh* et al. [3.4]. Two algorithms are provided for forward dynamics. The first is the $O(n)$ articulated-body algorithm (ABA) which was developed by *Featherstone* [3.1]. The second is the $O(n^2)$ composite-rigid-body algorithm (CRBA), developed by *Walker* and *Orin* [3.5], to compute the joint-space inertia matrix (JSIM). This matrix, together with a vector computed using the RNEA, provide the coefficients of the equation of motion, which can then be solved directly for the accelerations [3.5]. The operational-space inertia matrix (OSIM) is a kind of articulated-body inertia, and two algorithms are given to calculate it. The first uses the basic definition of the OSIM, and the second is a straightforward $O(n)$ algorithm which is based on efficient solution of the forward dynamics problem. The inputs, outputs, model data, and pseudocode for each algorithm are summarized in tables for ready access.

3.1.5 Kinematic Loops

The above algorithms apply only to mechanisms having the connectivity of kinematic trees, including unbranched kinematic chains. A final algorithm is provided in Sect. 3.6 for the forward dynamics of closed-loop systems, including parallel robot mechanisms. The algorithm makes use of the dynamic equations of motion for a spanning tree of the closed-loop system, and supplements these with loop-closure constraint equations. Three different methods are outlined to solve the resulting linear system of equations. Method 2 is particularly useful if $n \gg n^c$, where n^c is the number of constraints due to the loop-closing joints. This method offers the opportunity to use $O(n)$ algorithms on the spanning tree [3.6]. The section ends with an efficient algorithm to compute the loop-closure constraints by transforming them to a single coordinate system. Since the loop-closure constraint equations are applied at the acceleration level, standard *Baumgarte* stabilization [3.7] is used to prevent the accumulation of position and velocity errors in the loop-closure constraints.

The final section in this chapter provides a conclusion and suggestions for further reading. The area of robot dynamics has been, and continues to be, a very rich area of investigation. This section outlines the major contributions that have been made in the area and the work most often cited. Unfortunately, space does not permit us to provide a comprehensive review of the extensive literature in the area.

3.2 Spatial Vector Notation

There is no single standard notation for robot dynamics. The notations currently in use include 3-D vectors, 4×4 matrices, and several types of 6-D vector: screws, motors, Lie algebra elements, and spatial vectors. Six-dimensional vector notations are generally the best, being more compact than 3-D vectors, and more powerful than 4×4 matrices. We therefore use 6-D vectors throughout this chapter. In particular, we shall use the spatial vector algebra described in [3.8]. This section provides a brief summary of spatial vectors. Descriptions of 4×4 matrix notations can be found in [3.2, 9], and descriptions of other 6-D vector notations can be found in [3.10–12].

In this handbook, vectors are usually denoted by bold italic letters (e.g., $\boldsymbol{f}$, $\boldsymbol{v}$). However, to avoid a few name clashes, we shall use upright bold letters to denote spatial vectors (e.g., $\mathbf{f}$, $\mathbf{v}$). Note that this applies only to vectors, not tensors. Also, *in this section only*, we will underline coordinate vectors to distinguish them from the vectors that they represent (e.g., $\underline{\boldsymbol{v}}$ and $\underline{\mathbf{v}}$, representing $\boldsymbol{v}$ and $\mathbf{v}$).

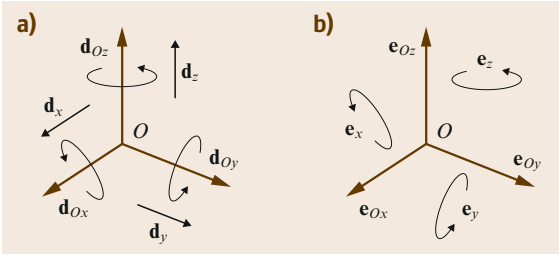

Fig.3.1a,b Plücker basis vectors for motions **(a)** and forces **(b)**

3.2.1 Motion and Force

For mathematical reasons, it is useful to distinguish between those vectors that describe the motions of rigid bodies, and those that describe the forces acting upon them. We therefore place motion vectors in a vector space called M^6, and force vectors in a space called F^6. (The superscripts indicate the dimension.) Motion vectors describe quantities like velocity, acceleration, infinitesimal displacement, and directions of motion freedom; force vectors describe force, momentum, contact normals, and so on.

3.2.2 Basis Vectors

Suppose that $\boldsymbol{v}$ is a 3-D vector, and that $\underline{\boldsymbol{v}} = (v_x, v_y, v_z)^{\mathsf{T}}$ is the Cartesian coordinate vector that represents $\boldsymbol{v}$ in the orthonormal basis $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}\}$. The relationship between $\boldsymbol{v}$ and $\underline{\boldsymbol{v}}$ is then given by the formula

$$\boldsymbol{v} = \hat{\boldsymbol{x}} v_x + \hat{\boldsymbol{y}} v_y + \hat{\boldsymbol{z}} v_z \,.$$

This same idea applies also to spatial vectors, except that we use Plücker coordinates instead of Cartesian coordinates, and a Plücker basis instead of an orthonormal basis.

Plücker coordinates were introduced in Sect. 2.2.6, but the basis vectors are shown in Fig. 3.1. There are 12 basis vectors in total: six for motion vectors and six for forces. Given a Cartesian coordinate frame, O_{xyz}, the Plücker basis vectors are defined as follows: three unit rotations about the directed lines Ox, Oy, and Oz, denoted by $\mathbf{d}_{Ox}$, $\mathbf{d}_{Oy}$, and $\mathbf{d}_{Oz}$, three unit translations in the directions x, y, and z, denoted by $\mathbf{d}_x$, $\mathbf{d}_y$, and $\mathbf{d}_z$, three unit couples about the x, y, and z directions, denoted by $\mathbf{e}_x$, $\mathbf{e}_y$, and $\mathbf{e}_z$, and three unit forces along the lines Ox, Oy, and Oz, denoted by $\mathbf{e}_{Ox}$, $\mathbf{e}_{Oy}$, and $\mathbf{e}_{Oz}$.

3.2.3 Spatial Velocity and Force

Given any point O, the velocity of a rigid body can be described by a pair of 3-D vectors, $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$, which specify the body's angular velocity and the linear velocity of the body-fixed point currently at O. Note that $\boldsymbol{v}_O$ is not the velocity of O itself, but the velocity of the body-fixed point that happens to coincide with O at the current instant.

The velocity of this same rigid body can also be described by a single spatial motion vector, $\mathbf{v} \in \mathsf{M}^6$. To obtain $\mathbf{v}$ from $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$, we first introduce a Cartesian frame, O_{xyz}, with its origin at O. This frame defines a Cartesian coordinate system for $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$, and also a Plücker coordinate system for $\mathbf{v}$. Given these coordinate systems, it can be shown that

$$\begin{aligned}\mathbf{v} = {} & \mathbf{d}_{Ox}\omega_x + \mathbf{d}_{Oy}\omega_y + \mathbf{d}_{Oz}\omega_z + \mathbf{d}_x v_{Ox} \\ & + \mathbf{d}_y v_{Oy} + \mathbf{d}_z v_{Oz} \,, \end{aligned} \tag{3.1}$$

where $\omega_x, \ldots, v_{Oz}$ are the Cartesian coordinates of $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ in O_{xyz}. Thus, the Plücker coordinates of $\mathbf{v}$ are the Cartesian coordinates of $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$. The coordinate vector representing $\mathbf{v}$ in O_{xyz} can be written

$$\underline{\mathbf{v}}_O = \begin{pmatrix} \omega_x \\ \vdots \\ v_{Oz} \end{pmatrix} = \begin{pmatrix} \underline{\boldsymbol{\omega}} \\ \underline{\boldsymbol{v}}_O \end{pmatrix} . \tag{3.2}$$

The notation on the far right of this equation is simply a convenient abbreviation of the list of Plücker coordinates.

The definition of spatial force is very similar. Given any point O, any system of forces acting on a single rigid body is equivalent to a single force $\boldsymbol{f}$ acting on a line passing through O, together with a pure couple, $\boldsymbol{n}_O$, which is the moment of the force system about O. Thus, the two vectors $\boldsymbol{f}$ and $\boldsymbol{n}_O$ describe the force acting on a rigid body in much the same way that $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ describe its velocity. This same force can also be described by a single spatial force vector, $\mathbf{f} \in \mathsf{F}^6$. Introducing the frame O_{xyz}, as before, it can be shown that

$$\begin{aligned}\mathbf{f} = {} & \mathbf{e}_x n_{Ox} + \mathbf{e}_y n_{Oy} + \mathbf{e}_z n_{Oz} + \mathbf{e}_{Ox} f_x \\ & + \mathbf{e}_{Oy} f_y + \mathbf{e}_{Oz} f_z \,, \end{aligned} \tag{3.3}$$

where $n_{Ox}, \ldots, f_z$ are the Cartesian coordinates of $\boldsymbol{n}_O$ and $\boldsymbol{f}$ in O_{xyz}. The coordinate vector representing $\mathbf{f}$ in O_{xyz} can then be written

$$\underline{\mathbf{f}}_O = \begin{pmatrix} n_{Ox} \\ \vdots \\ f_z \end{pmatrix} = \begin{pmatrix} \underline{\boldsymbol{n}}_O \\ \underline{\boldsymbol{f}} \end{pmatrix} . \tag{3.4}$$

Again, these are the Plücker coordinates of $\mathbf{f}$ in O_{xyz}, and the notation on the far right is simply a convenient abbreviation of the list of Plücker coordinates.

3.2.4 Addition and Scalar Multiplication

Spatial vectors behave in the obvious way under addition and scalar multiplication. For example, if $\mathbf{f}_1$ and $\mathbf{f}_2$ both act on the same rigid body, then their resultant is $\mathbf{f}_1 + \mathbf{f}_2$; if two different bodies have velocities of $\mathbf{v}_1$ and $\mathbf{v}_2$, then the velocity of the second body relative to the first is $\mathbf{v}_2 - \mathbf{v}_1$; and if $\mathbf{f}$ denotes a force of 1 N acting along a particular line in space, then $\alpha\,\mathbf{f}$ denotes a force of α N acting along the same line.

3.2.5 Scalar Product

A scalar product is defined between any two spatial vectors, provided that one of them is a motion and the other a force. Given any $\mathbf{m} \in \mathsf{M}^6$ and $\mathbf{f} \in \mathsf{F}^6$, the scalar product can be written either $\mathbf{f}\cdot\mathbf{m}$ or $\mathbf{m}\cdot\mathbf{f}$, and expresses the work done by a force $\mathbf{f}$ acting on a body with motion $\mathbf{m}$. Expressions like $\mathbf{f}\cdot\mathbf{f}$ and $\mathbf{m}\cdot\mathbf{m}$ are not defined. If $\underline{\mathbf{m}}$ and $\underline{\mathbf{f}}$ are coordinate vectors representing $\mathbf{m}$ and $\mathbf{f}$ in the same coordinate system, then

$$\mathbf{m}\cdot\mathbf{f} = \underline{\mathbf{m}}^{\mathrm{T}}\underline{\mathbf{f}}\,. \tag{3.5}$$

3.2.6 Coordinate Transforms

Motion and force vectors obey different transformation rules. Let A and B be two coordinate frames, each defining a coordinate system of the same name; and let $\underline{\mathbf{m}}_A$, $\underline{\mathbf{m}}_B$, $\underline{\mathbf{f}}_A$, and $\underline{\mathbf{f}}_B$ be coordinate vectors representing the spatial vectors $\mathbf{m} \in \mathsf{M}^6$ and $\mathbf{f} \in \mathsf{F}^6$ in A and B coordinates, respectively. The transformation rules are then

$$\underline{\mathbf{m}}_B = {}^B\boldsymbol{X}_A\,\underline{\mathbf{m}}_A \tag{3.6}$$

and

$$\underline{\mathbf{f}}_B = {}^B\boldsymbol{X}_A^F\,\underline{\mathbf{f}}_A\,, \tag{3.7}$$

where ${}^B\boldsymbol{X}_A$ and ${}^B\boldsymbol{X}_A^F$ are the coordinate transformation matrices from A to B for motion and force vectors, respectively. These matrices are related by the identity

$${}^B\boldsymbol{X}_A^F \equiv ({}^B\boldsymbol{X}_A)^{-T} \equiv ({}^A\boldsymbol{X}_B)^{\mathrm{T}}\,. \tag{3.8}$$

Suppose that the position and orientation of frame A relative to frame B is described by a position vector ${}^B\underline{\boldsymbol{p}}_A$ and a 3×3 rotation matrix ${}^B\boldsymbol{R}_A$, as described in Sect. 2.2. The formula for ${}^B\boldsymbol{X}_A$ is then

$$\begin{aligned} {}^B\boldsymbol{X}_A &= \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \boldsymbol{S}({}^B\underline{\boldsymbol{p}}_A) & \mathbf{1} \end{pmatrix} \begin{pmatrix} {}^B\boldsymbol{R}_A & \mathbf{0} \\ \mathbf{0} & {}^B\boldsymbol{R}_A \end{pmatrix} \\ &= \begin{pmatrix} {}^B\boldsymbol{R}_A & \mathbf{0} \\ \boldsymbol{S}({}^B\underline{\boldsymbol{p}}_A){}^B\boldsymbol{R}_A & {}^B\boldsymbol{R}_A \end{pmatrix}, \end{aligned} \tag{3.9}$$

and its inverse is

$${}^A\boldsymbol{X}_B = \begin{pmatrix} {}^A\boldsymbol{R}_B & \mathbf{0} \\ \mathbf{0} & {}^A\boldsymbol{R}_B \end{pmatrix} \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ -\boldsymbol{S}({}^B\underline{\boldsymbol{p}}_A) & \mathbf{1} \end{pmatrix}. \tag{3.10}$$

The quantity $\boldsymbol{S}(\underline{\boldsymbol{p}})$ is the skew-symmetric matrix that satisfies $\boldsymbol{S}(\underline{\boldsymbol{p}})\underline{\boldsymbol{v}} = \underline{\boldsymbol{p}} \times \underline{\boldsymbol{v}}$ for any 3-D vector $\underline{\boldsymbol{v}}$. It is defined by the equation

$$\boldsymbol{S}(\underline{\boldsymbol{p}}) = \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix}. \tag{3.11}$$

3.2.7 Vector Products

There are two vector (cross) products defined on spatial vectors. The first takes two motion-vector arguments, and produces a motion-vector result. It is defined by the formula

$$\begin{aligned} \underline{\mathbf{m}}_1 \times \underline{\mathbf{m}}_2 &= \begin{pmatrix} \underline{\boldsymbol{m}}_1 \\ \underline{\boldsymbol{m}}_{1O} \end{pmatrix} \times \begin{pmatrix} \underline{\boldsymbol{m}}_2 \\ \underline{\boldsymbol{m}}_{2O} \end{pmatrix} \\ &= \begin{pmatrix} \underline{\boldsymbol{m}}_1 \times \underline{\boldsymbol{m}}_2 \\ \underline{\boldsymbol{m}}_1 \times \underline{\boldsymbol{m}}_{2O} + \underline{\boldsymbol{m}}_{1O} \times \underline{\boldsymbol{m}}_2 \end{pmatrix}. \end{aligned} \tag{3.12}$$

The second takes a motion vector as left-hand argument and a force vector as right-hand argument, and produces a force-vector result. It is defined by the formula

$$\begin{aligned} \underline{\mathbf{m}} \times \underline{\mathbf{f}} &= \begin{pmatrix} \underline{\boldsymbol{m}} \\ \underline{\boldsymbol{m}}_O \end{pmatrix} \times \begin{pmatrix} \underline{\boldsymbol{f}}_O \\ \underline{\boldsymbol{f}} \end{pmatrix} \\ &= \begin{pmatrix} \underline{\boldsymbol{m}} \times \underline{\boldsymbol{f}}_O + \underline{\boldsymbol{m}}_O \times \underline{\boldsymbol{f}} \\ \underline{\boldsymbol{m}} \times \underline{\boldsymbol{f}} \end{pmatrix}. \end{aligned} \tag{3.13}$$

These products arise in differentiation formulae.

It is possible to define a spatial cross-product operator, in analogy with (3.11), as follows

$$\boldsymbol{S}(\underline{\mathbf{m}}) = \begin{pmatrix} \boldsymbol{S}(\underline{\boldsymbol{m}}) & \mathbf{0} \\ \boldsymbol{S}(\underline{\boldsymbol{m}}_O) & \boldsymbol{S}(\underline{\boldsymbol{m}}) \end{pmatrix}, \tag{3.14}$$

in which case

$$\underline{\mathbf{m}}_1 \times \underline{\mathbf{m}}_2 = \boldsymbol{S}(\underline{\mathbf{m}}_1)\,\underline{\mathbf{m}}_2\,, \tag{3.15}$$

but

$$\underline{\mathbf{m}} \times \underline{\mathbf{f}} = -\boldsymbol{S}(\underline{\mathbf{m}})^{\mathrm{T}}\underline{\mathbf{f}}\,. \tag{3.16}$$

Observe that $\boldsymbol{S}(\underline{\mathbf{m}})$ maps motion vectors to motion vectors, but $\boldsymbol{S}(\underline{\boldsymbol{m}})^{\mathrm{T}}$ maps force vectors to force vectors.

3.2.8 Differentiation

The derivative of a spatial vector is defined by

$$\frac{\mathrm{d}}{\mathrm{d}x}\mathbf{s}(x) = \lim_{\delta x \to 0} \frac{\mathbf{s}(x+\delta x) - \mathbf{s}(x)}{\delta x}\,, \tag{3.17}$$

where $\mathbf{s}$ here stands for any spatial vector. The derivative is a spatial vector of the same kind (motion or force) as that being differentiated.

The formula for differentiating a spatial coordinate vector in a moving coordinate system is

$$\left(\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s}\right)_A = \frac{\mathrm{d}}{\mathrm{d}t}\underline{\mathbf{s}}_A + \underline{\mathbf{v}}_A \times \underline{\mathbf{s}}_A\,, \tag{3.18}$$

where $\mathbf{s}$ is any spatial vector, $\mathrm{d}\mathbf{s}/\mathrm{d}t$ is the time derivative of $\mathbf{s}$, A is the moving coordinate system, $(\mathrm{d}\mathbf{s}/\mathrm{d}t)_A$ is the coordinate vector that represents $\mathrm{d}\mathbf{s}/\mathrm{d}t$ in A coordinates, $\underline{\mathbf{s}}_A$ is the coordinate vector that represents $\mathbf{s}$ in A coordinates, $\mathrm{d}\underline{\mathbf{s}}_A/\mathrm{d}t$ is the time derivative of $\underline{\mathbf{s}}_A$ (which is the componentwise derivative, since $\underline{\mathbf{s}}_A$ is a coordinate vector), and $\underline{\mathbf{v}}_A$ is the velocity of the A coordinate frame, expressed in A coordinates.

The time derivative of a spatial vector that changes only because it is moving is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{s} = \mathbf{v} \times \mathbf{s}\,, \tag{3.19}$$

where $\mathbf{v}$ is the velocity of $\mathbf{s}$. This formula is useful for differentiating quantities that do not change in their own right, but are attached to moving rigid bodies (e.g., joint axis vectors).

3.2.9 Acceleration

Spatial acceleration is defined as the rate of change of spatial velocity. Unfortunately, this means that spatial acceleration differs from the classical textbook definition of rigid-body acceleration, which we shall call *classical acceleration*. Essentially, the difference can be summarized as follows

$$\underline{\mathbf{a}} = \begin{pmatrix} \underline{\dot{\boldsymbol{\omega}}} \\ \underline{\dot{\boldsymbol{v}}}_O \end{pmatrix} \quad \text{and} \quad \underline{\mathbf{a}}' = \begin{pmatrix} \underline{\dot{\boldsymbol{\omega}}} \\ \underline{\dot{\boldsymbol{v}}}'_O \end{pmatrix}, \tag{3.20}$$

where $\underline{\mathbf{a}}$ is the spatial acceleration, $\underline{\mathbf{a}}'$ is the classical acceleration, $\underline{\dot{\boldsymbol{v}}}_O$ is the derivative of $\underline{\boldsymbol{v}}_O$ taking O to be fixed in space, and $\underline{\dot{\boldsymbol{v}}}'_O$ is the derivative of $\underline{\boldsymbol{v}}_O$ taking O to be fixed in the body. The two accelerations are related by

$$\underline{\mathbf{a}}' = \underline{\mathbf{a}} + \begin{pmatrix} \mathbf{0} \\ \underline{\boldsymbol{\omega}} \times \underline{\boldsymbol{v}}_O \end{pmatrix}. \tag{3.21}$$

If $\boldsymbol{r}$ is a position vector giving the position of the body-fixed point at O relative to any fixed point, then

$$\begin{aligned} \boldsymbol{v}_O &= \dot{\boldsymbol{r}}\,, \\ \dot{\boldsymbol{v}}'_O &= \ddot{\boldsymbol{r}}\,, \\ \dot{\boldsymbol{v}}_O &= \ddot{\boldsymbol{r}} - \boldsymbol{\omega} \times \boldsymbol{v}_O\,. \end{aligned} \tag{3.22}$$

The practical difference is that spatial accelerations are easier to use. For example, if the bodies B_1 and B_2 have velocities of $\mathbf{v}_1$ and $\mathbf{v}_2$, respectively, and $\mathbf{v}_{\mathrm{rel}}$ is the relative velocity of B_2 with respect to B_1, then

$$\mathbf{v}_2 = \mathbf{v}_1 + \mathbf{v}_{\mathrm{rel}}\,.$$

The relationship between their spatial accelerations is obtained simply by differentiating the velocity formula

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{v}_2 = \mathbf{v}_1 + \mathbf{v}_{\mathrm{rel}}) \quad \Rightarrow \quad \mathbf{a}_2 = \mathbf{a}_1 + \mathbf{a}_{\mathrm{rel}}\,.$$

Observe that spatial accelerations are composed by addition, exactly like velocities. There are no Coriolis or centrifugal terms to worry about. This is a significant improvement on the formulae for composing classical accelerations, such as those in [3.2, 13, 14].

3.2.10 Spatial Momentum

Suppose that a rigid body has a mass of m, a center of mass at C, and a rotational inertia of $\bar{\boldsymbol{I}}^{\mathrm{cm}}$ about C (Fig. 3.2). If this body is moving with a spatial velocity of $\underline{\mathbf{v}}_C = (\underline{\boldsymbol{\omega}}^{\mathrm{T}}\,\underline{\boldsymbol{v}}_C^{\mathrm{T}})^{\mathrm{T}}$, then its linear momentum is $\boldsymbol{h} = m\boldsymbol{v}_C$, and its intrinsic angular momentum is $\boldsymbol{h}_C = \bar{\boldsymbol{I}}^{\mathrm{cm}}\boldsymbol{\omega}$. Its moment of momentum about a general point, O, is $\boldsymbol{h}_O = \boldsymbol{h}_C + \boldsymbol{c} \times \boldsymbol{h}$, where $\boldsymbol{c} = \overrightarrow{OC}$. We can assemble these vectors into a spatial momentum vector as follows

$$\underline{\mathbf{h}}_C = \begin{pmatrix} \underline{\boldsymbol{h}}_C \\ \underline{\boldsymbol{h}} \end{pmatrix} = \begin{pmatrix} \bar{\boldsymbol{I}}^{\mathrm{cm}}\underline{\boldsymbol{\omega}} \\ m\underline{\boldsymbol{v}}_C \end{pmatrix} \tag{3.23}$$

and

$$\underline{\mathbf{h}}_O = \begin{pmatrix} \underline{\boldsymbol{h}}_O \\ \underline{\boldsymbol{h}} \end{pmatrix} = \begin{pmatrix} \mathbf{1} & S(\underline{\boldsymbol{c}}) \\ \mathbf{0} & \mathbf{1} \end{pmatrix} \underline{\mathbf{h}}_C\,. \tag{3.24}$$

Spatial momentum is a force vector, and transforms accordingly.

3.2.11 Spatial Inertia

The spatial momentum of a rigid body is the product of its spatial inertia and velocity

$$\mathbf{h} = \boldsymbol{I}\mathbf{v}\,, \tag{3.25}$$

where $\boldsymbol{I}$ is the spatial inertia. Expressed in Plücker coordinates at C, we have

$$\mathbf{h}_C = \boldsymbol{I}_C \mathbf{v}_C \,, \tag{3.26}$$

which implies

$$\boldsymbol{I}_C = \begin{pmatrix} \bar{\boldsymbol{I}}^{\mathrm{cm}} & \mathbf{0} \\ \mathbf{0} & m\mathbf{1} \end{pmatrix} . \tag{3.27}$$

This is the general formula for the spatial inertia of a rigid body expressed at its center of mass. To express it at another point, O, we proceed as follows. From (3.24), (3.26), and (3.27)

$$\begin{aligned} \underline{\mathbf{h}}_O &= \begin{pmatrix} \mathbf{1} & \boldsymbol{S}(\underline{\boldsymbol{c}}) \\ \mathbf{0} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \bar{\boldsymbol{I}}^{\mathrm{cm}} & \mathbf{0} \\ \mathbf{0} & m\mathbf{1} \end{pmatrix} \underline{\mathbf{v}}_C \\ &= \begin{pmatrix} \mathbf{1} & \boldsymbol{S}(\underline{\boldsymbol{c}}) \\ \mathbf{0} & \mathbf{1} \end{pmatrix} \begin{pmatrix} \bar{\boldsymbol{I}}^{\mathrm{cm}} & \mathbf{0} \\ \mathbf{0} & m\mathbf{1} \end{pmatrix} \begin{pmatrix} \mathbf{1} & \mathbf{0} \\ \boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} & \mathbf{1} \end{pmatrix} \underline{\mathbf{v}}_O \\ &= \begin{pmatrix} \bar{\boldsymbol{I}}^{\mathrm{cm}} + m\boldsymbol{S}(\underline{\boldsymbol{c}})\boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} & m\boldsymbol{S}(\underline{\boldsymbol{c}}) \\ m\boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} & m\mathbf{1} \end{pmatrix} \underline{\mathbf{v}}_O \,; \end{aligned}$$

but we also have that $\underline{\mathbf{h}}_O = \boldsymbol{I}_O \underline{\mathbf{v}}_O$, so

$$\boldsymbol{I}_O = \begin{pmatrix} \bar{\boldsymbol{I}}^{\mathrm{cm}} + m\boldsymbol{S}(\underline{\boldsymbol{c}})\boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} & m\boldsymbol{S}(\underline{\boldsymbol{c}}) \\ m\boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} & m\mathbf{1} \end{pmatrix} . \tag{3.28}$$

This equation can also be written

$$\boldsymbol{I}_O = \begin{pmatrix} \bar{\boldsymbol{I}}_O & m\boldsymbol{S}(\underline{\boldsymbol{c}}) \\ m\boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} & m\mathbf{1} \end{pmatrix} , \tag{3.29}$$

where

$$\bar{\boldsymbol{I}}_O = \bar{\boldsymbol{I}}^{\mathrm{cm}} + m\boldsymbol{S}(\underline{\boldsymbol{c}})\boldsymbol{S}(\underline{\boldsymbol{c}})^{\mathrm{T}} \tag{3.30}$$

is the rotational inertia of the rigid body about O.

Spatial inertia matrices are symmetric and positive-definite. In the general case, 21 numbers are required to specify a spatial inertia (e.g., for an articulated-body or operational-space inertia); but a rigid-body inertia needs only 10 parameters: the mass, the coordinates of the center of mass, and the six independent elements of either $\bar{\boldsymbol{I}}^{\mathrm{cm}}$ or $\bar{\boldsymbol{I}}_O$.

The transformation rule for spatial inertias is

$$\boldsymbol{I}_B = {}^B\boldsymbol{X}_A^F \, \boldsymbol{I}_A \, {}^A\boldsymbol{X}_B \,, \tag{3.31}$$

where A and B are any two coordinate systems. In practice, we often need to calculate $\boldsymbol{I}_A$ from $\boldsymbol{I}_B$, given only ${}^B\boldsymbol{X}_A$. The formula for this transformation is

$$\boldsymbol{I}_A = ({}^B\boldsymbol{X}_A)^{\mathrm{T}} \boldsymbol{I}_B \, {}^B\boldsymbol{X}_A \,. \tag{3.32}$$

If two bodies, having inertias $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, are rigidly connected to form a single composite body, then the inertia of the composite, $\boldsymbol{I}_{\mathrm{tot}}$, is the sum of the inertias of its parts

$$\boldsymbol{I}_{\mathrm{tot}} = \boldsymbol{I}_1 + \boldsymbol{I}_2 \,. \tag{3.33}$$

This single equation takes the place of three equations in the traditional 3-D vector approach: one to compute the composite mass, one to compute the composite center of mass, and one to compute the composite rotational inertia. If a rigid body with inertia $\boldsymbol{I}$ is moving with a velocity of $\mathbf{v}$, then its kinetic energy is

$$T = \frac{1}{2} \mathbf{v} \cdot \boldsymbol{I}\mathbf{v} \,. \tag{3.34}$$

If a rigid body, B, is part of a larger system, then it is possible to define an apparent-inertia matrix for B, which describes the relationship between a force acting on B and its resulting acceleration, taking into account the effects of the other bodies in the system. Such quantities are called articulated-body inertias. If B happens to be the end-effector of a robot, then its apparent inertia is called an operational-space inertia.

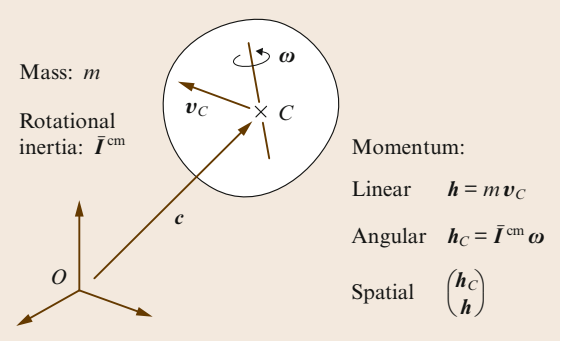

Fig. 3.2 Spatial momentum

3.2.12 Equation of Motion

The spatial equation of motion states that the net force acting on a rigid body equals its rate of change of momentum

$$\mathbf{f} = \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{I}\mathbf{v}) = \boldsymbol{I}\mathbf{a} + \dot{\boldsymbol{I}}\mathbf{v} \,.$$

It can be shown that the expression $\dot{\boldsymbol{I}}\mathbf{v}$ evaluates to $(\mathbf{v} \times \boldsymbol{I}\mathbf{v})$ [3.8, 15], so the equation of motion can be written

$$\mathbf{f} = \boldsymbol{I}\mathbf{a} + \mathbf{v} \times \boldsymbol{I}\mathbf{v} \,. \tag{3.35}$$

This single equation incorporates both Newton's and Euler's equations of motion for a rigid body. To

Table 3.1 Summary of spatial vector notation

Spatial quantities:	
$\mathbf{v}$	Velocity of a rigid body
$\mathbf{a}$	Spatial acceleration of a rigid body ($\mathbf{a} = \dot{\mathbf{v}}$)
$\mathbf{a}'$	Classical description of rigid-body acceleration expressed as a 6-D vector
$\mathbf{f}$	Force acting on a rigid body
$\boldsymbol{I}$	Inertia of a rigid body
$\boldsymbol{X}$	Plücker coordinate transform for motion vectors
$\boldsymbol{X}^F$	Plücker coordinate transform for force vectors ($\boldsymbol{X}^F = \boldsymbol{X}^{-T}$)
${}^B\boldsymbol{X}_A$	Plücker transform from A coordinates to B coordinates
$\mathbf{m}$	A generic motion vector (any element of M^6)
3-D quantities:	
O	Coordinate system origin
$\boldsymbol{r}$	Position of the body-fixed point at O relative to any fixed point in space
$\boldsymbol{\omega}$	Angular velocity of a rigid body
$\boldsymbol{v}_O$	Linear velocity of the body-fixed point at O ($\boldsymbol{v}_O = \dot{\boldsymbol{r}}$)
$\dot{\boldsymbol{\omega}}$	Angular acceleration of a rigid body
$\dot{\boldsymbol{v}}_O$	The derivative of $\boldsymbol{v}_O$ taking O to be fixed in space
$\dot{\boldsymbol{v}}'_O$	The derivative of $\boldsymbol{v}_O$ taking O to be fixed in the body; the classical acceleration of the body-fixed point at O ($\dot{\boldsymbol{v}}'_O = \ddot{\boldsymbol{r}}$)
$\boldsymbol{f}$	Linear force acting on a rigid body, or the resultant of a system of linear forces
$\boldsymbol{n}_O$	Moment about O of a linear force or system of linear forces
m	Mass of a rigid body
$\boldsymbol{c}$	Position of a rigid body's center of mass, measured relative to O
$\boldsymbol{h}$	First moment of mass of a rigid body, $\boldsymbol{h} = m\boldsymbol{c}$; can also denote linear momentum
$\bar{\boldsymbol{I}}^{\mathrm{cm}}$	Moment of inertia about a body's centre of mass
$\bar{\boldsymbol{I}}$	Moment of inertia about O
${}^B\boldsymbol{R}_A$	Orthonormal rotation matrix transforming from A coordinates to B coordinates
${}^A\boldsymbol{p}_B$	Location of the origin of B coordinates relative to the origin of A coordinates, expressed in A coordinates

verify this, we can recover them as follows. Expressing (3.35) at the body's center of mass, and using (3.16), (3.14), and (3.22), we have

$$\begin{aligned}\begin{pmatrix}\underline{\boldsymbol{n}}_C\\ \underline{\boldsymbol{f}}\end{pmatrix} &= \begin{pmatrix}\bar{\boldsymbol{I}}^{\mathrm{cm}} & \mathbf{0}\\ \mathbf{0} & m\mathbf{1}\end{pmatrix}\begin{pmatrix}\dot{\underline{\boldsymbol{\omega}}}\\ \dot{\underline{\boldsymbol{v}}}_C\end{pmatrix} - \begin{pmatrix}S(\underline{\boldsymbol{\omega}})^{\mathrm{T}} & S(\underline{\boldsymbol{v}}_C)^{\mathrm{T}}\\ \mathbf{0} & S(\underline{\boldsymbol{\omega}})^{\mathrm{T}}\end{pmatrix}\begin{pmatrix}\bar{\boldsymbol{I}}^{\mathrm{cm}}\underline{\boldsymbol{\omega}}\\ m\underline{\boldsymbol{v}}_C\end{pmatrix}\\ &= \begin{pmatrix}\bar{\boldsymbol{I}}^{\mathrm{cm}} & \mathbf{0}\\ \mathbf{0} & m\mathbf{1}\end{pmatrix}\begin{pmatrix}\dot{\underline{\boldsymbol{\omega}}}\\ \ddot{\underline{\boldsymbol{c}}} - \underline{\boldsymbol{\omega}}\times\underline{\boldsymbol{v}}_C\end{pmatrix} + \begin{pmatrix}\underline{\boldsymbol{\omega}}\times\bar{\boldsymbol{I}}^{\mathrm{cm}}\underline{\boldsymbol{\omega}}\\ m\underline{\boldsymbol{\omega}}\times\underline{\boldsymbol{v}}_C\end{pmatrix}\\ &= \begin{pmatrix}\bar{\boldsymbol{I}}^{\mathrm{cm}}\dot{\underline{\boldsymbol{\omega}}} + \underline{\boldsymbol{\omega}}\times\bar{\boldsymbol{I}}^{\mathrm{cm}}\underline{\boldsymbol{\omega}}\\ m\ddot{\underline{\boldsymbol{c}}}\end{pmatrix}.\end{aligned} \tag{3.36}$$

Table 3.1 (continued)

Equations
$\mathbf{v} = \begin{pmatrix}\boldsymbol{\omega}\\ \boldsymbol{v}_O\end{pmatrix} \quad \mathbf{a} = \begin{pmatrix}\dot{\boldsymbol{\omega}}\\ \dot{\boldsymbol{v}}_O\end{pmatrix} = \begin{pmatrix}\dot{\boldsymbol{\omega}}\\ \ddot{\boldsymbol{r}} - \boldsymbol{\omega}\times\dot{\boldsymbol{r}}\end{pmatrix}$
$\mathbf{f} = \begin{pmatrix}\boldsymbol{n}_O\\ \boldsymbol{f}\end{pmatrix} \quad \mathbf{a}' = \begin{pmatrix}\dot{\boldsymbol{\omega}}\\ \dot{\boldsymbol{v}}'_O\end{pmatrix} = \begin{pmatrix}\dot{\boldsymbol{\omega}}\\ \ddot{\boldsymbol{r}}\end{pmatrix} = \mathbf{a} + \begin{pmatrix}\mathbf{0}\\ \boldsymbol{\omega}\times\boldsymbol{v}_O\end{pmatrix}$
$\boldsymbol{I} = \begin{pmatrix}\bar{\boldsymbol{I}} & S(\boldsymbol{h})\\ S(\boldsymbol{h})^{\mathrm{T}} & m\mathbf{1}\end{pmatrix} = \begin{pmatrix}\bar{\boldsymbol{I}}^{\mathrm{cm}} + mS(\boldsymbol{c})S(\boldsymbol{c})^{\mathrm{T}} & mS(\boldsymbol{c})\\ mS(\boldsymbol{c})^{\mathrm{T}} & m\mathbf{1}\end{pmatrix}$
${}^B\boldsymbol{X}_A = \begin{pmatrix}{}^B\boldsymbol{R}_A & \mathbf{0}\\ {}^B\boldsymbol{R}_A S({}^A\boldsymbol{p}_B)^{\mathrm{T}} & {}^B\boldsymbol{R}_A\end{pmatrix} = \begin{pmatrix}{}^B\boldsymbol{R}_A & \mathbf{0}\\ S({}^B\boldsymbol{p}_A)\,{}^B\boldsymbol{R}_A & {}^B\boldsymbol{R}_A\end{pmatrix}$
$\mathbf{v}\cdot\mathbf{f} = \mathbf{f}\cdot\mathbf{v} = \mathbf{v}^{\mathrm{T}}\mathbf{f} = \boldsymbol{\omega}\cdot\boldsymbol{n}_O + \boldsymbol{v}_O\cdot\boldsymbol{f}$
$\mathbf{v}\times\mathbf{m} = \begin{pmatrix}\boldsymbol{\omega}\times\boldsymbol{m}\\ \boldsymbol{v}_O\times\boldsymbol{m} + \boldsymbol{\omega}\times\boldsymbol{m}_O\end{pmatrix} = \begin{pmatrix}S(\boldsymbol{\omega}) & \mathbf{0}\\ S(\boldsymbol{v}_O) & S(\boldsymbol{\omega})\end{pmatrix}\begin{pmatrix}\boldsymbol{m}\\ \boldsymbol{m}_O\end{pmatrix}$
$\mathbf{v}\times\mathbf{f} = \begin{pmatrix}\boldsymbol{\omega}\times\boldsymbol{n}_O + \boldsymbol{v}_O\times\boldsymbol{f}\\ \boldsymbol{\omega}\times\boldsymbol{f}\end{pmatrix} = \begin{pmatrix}S(\boldsymbol{\omega}) & S(\boldsymbol{v}_O)\\ \mathbf{0} & S(\boldsymbol{\omega})\end{pmatrix}\begin{pmatrix}\boldsymbol{n}_O\\ \boldsymbol{f}\end{pmatrix}$

Compact computer representations			
Mathematical object	**Size**	**Computer representation**	**Size**
$\begin{pmatrix}\boldsymbol{\omega}\\ \boldsymbol{v}_O\end{pmatrix}$	6×1	$(\boldsymbol{\omega}\,;\,\boldsymbol{v}_O)$	3+3
$\begin{pmatrix}\boldsymbol{n}_O\\ \boldsymbol{f}\end{pmatrix}$	6×1	$(\boldsymbol{n}_O\,;\,\boldsymbol{f})$	3+3
$\begin{pmatrix}\bar{\boldsymbol{I}} & S(\boldsymbol{h})\\ S(\boldsymbol{h})^{\mathrm{T}} & m\mathbf{1}\end{pmatrix}$	6×6	$(m\,;\,\boldsymbol{h}\,;\,\bar{\boldsymbol{I}})$	1+3+9
$\begin{pmatrix}\boldsymbol{R} & \mathbf{0}\\ \boldsymbol{R}S(\boldsymbol{p})^{\mathrm{T}} & \boldsymbol{R}\end{pmatrix}$	6×6	$(\boldsymbol{R}\,;\,\boldsymbol{p})$	9+3

Efficient spatial arithmetic formulae	
Expression	**Computed value**
$\boldsymbol{X}\,\mathbf{v}$	$(\boldsymbol{R}\,\boldsymbol{\omega}\,;\,\boldsymbol{R}(\boldsymbol{v}_O - \boldsymbol{p}\times\boldsymbol{\omega}))$
$\boldsymbol{X}^F\,\mathbf{f}$	$(\boldsymbol{R}(\boldsymbol{n}_O - \boldsymbol{p}\times\boldsymbol{f})\,;\,\boldsymbol{R}\boldsymbol{f})$
$\boldsymbol{X}^{-1}$	$(\boldsymbol{R}^{\mathrm{T}}\,;\,-\boldsymbol{R}\boldsymbol{p})$
$\boldsymbol{X}^{-1}\,\mathbf{v}$	$(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{\omega}\,;\,\boldsymbol{R}^{\mathrm{T}}\boldsymbol{v}_O + \boldsymbol{p}\times\boldsymbol{R}^{\mathrm{T}}\boldsymbol{\omega})$
$(\boldsymbol{X}^F)^{-1}\,\mathbf{f}$	$(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{n}_O + \boldsymbol{p}\times\boldsymbol{R}^{\mathrm{T}}\boldsymbol{f}\,;\,\boldsymbol{R}^{\mathrm{T}}\boldsymbol{f})$
$\boldsymbol{X}_1\boldsymbol{X}_2$	$(\boldsymbol{R}_1\boldsymbol{R}_2\,;\,\boldsymbol{p}_2 + \boldsymbol{R}_2^{\mathrm{T}}\boldsymbol{p}_1)$
$\boldsymbol{I}_1 + \boldsymbol{I}_2$	$(m_1 + m_2\,;\,\boldsymbol{h}_1 + \boldsymbol{h}_2\,;\,\bar{\boldsymbol{I}}_1 + \bar{\boldsymbol{I}}_2)$
$\boldsymbol{I}\,\mathbf{v}$	$(\bar{\boldsymbol{I}}\boldsymbol{\omega} + \boldsymbol{h}\times\boldsymbol{v}_O\,;\,m\,\boldsymbol{v}_O - \boldsymbol{h}\times\boldsymbol{\omega})$
$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{X}$	$(m\,;\,\boldsymbol{R}^{\mathrm{T}}\boldsymbol{h} + m\boldsymbol{p}\,;\,\boldsymbol{R}^{\mathrm{T}}\bar{\boldsymbol{I}}\boldsymbol{R} - S(\boldsymbol{p})S(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{h}) - S(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{h} + m\boldsymbol{p})S(\boldsymbol{p}))$

For meaning of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{X}$ see (3.32)

3.2.13 Computer Implementation

The easiest way to implement spatial vector arithmetic on a computer is to start with an existing matrix arithmetic tool, like MATLAB and write (or download from the Web) routines to do the following:

1. Calculate $\boldsymbol{S}(\mathbf{m})$ from $\mathbf{m}$ according to (3.14).
2. Compose $\boldsymbol{X}$ from $\boldsymbol{R}$ and $\boldsymbol{p}$ according to (3.9).
3. Compose $\boldsymbol{I}$ from m, $\boldsymbol{c}$, and $\bar{\boldsymbol{I}}^{\mathrm{cm}}$ according to (3.28).

All other spatial arithmetic operations can be performed using standard matrix arithmetic routines. However, some additional routines could usefully be added to this list, such as:

- Routines to calculate $\boldsymbol{R}$ from various other representations of rotation.
- Routines to convert between spatial and 4×4 matrix quantities.

This is the recommended approach whenever human productivity is more important than computational efficiency. A software package along these lines can be found at [3.16].

If greater efficiency is required, then a more elaborate spatial arithmetic library must be used, in which

1. A dedicated data structure is defined for each kind of spatial quantity, and
2. A suite of calculation routines are provided, each implementing a spatial arithmetic operation by means of an efficient formula.

Some examples of suitable data structures and efficient formulae are shown in Table 3.1. Observe that the suggested data structures for rigid-body inertias and Plücker transforms contain only a third as many numbers as the 6×6 matrices they represent. The efficient arithmetic formulae listed in this table offer cost savings ranging from a factor of 1.5 to a factor of 6 relative to the use of general 6×6 and 6×1 matrix arithmetic. Even more efficient formulae can be found in [3.17].

3.2.14 Summary

Spatial vectors are 6-D vectors that combine the linear and angular aspects of rigid-body motion, resulting in a compact notation that is very suitable for describing dynamics algorithms. To avoid a few name clashes with 3-D vectors, we have used bold upright letters to denote spatial vectors, while tensors are still denoted by italics. In the sections that follow, upright letters will be used to denote both spatial vectors and vectors that are concatenations of other vectors, like $\dot{\mathbf{q}}$.

Table 3.1 presents a summary of the spatial quantities and operators introduced in this section, together with the formulae that define them in terms of 3-D quantities and operators. It also presents data structures and formulae for efficient computer implementation of spatial arithmetic. This table should be read in conjunction with Tables 2.5 and 2.6, which show how to compute the orientation, position, and spatial velocities for a variety of joint types. Note that ${}^{j}\boldsymbol{R}_{i}$ and ${}^{j}\boldsymbol{p}_{i}$ in these tables correspond to ${}^{B}\boldsymbol{R}_{A}^{T}$ and ${}^{A}\boldsymbol{p}_{B}$, respectively, when read in conjunction with Table 3.1.

3.3 Canonical Equations

The equations of motion of a robot mechanism are usually presented in one of two canonical forms: the joint-space formulation,

$$\boldsymbol{H}(\mathbf{q})\ddot{\mathbf{q}}+\boldsymbol{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}+\boldsymbol{\tau}_{\mathrm{g}}(\mathbf{q})=\boldsymbol{\tau}\ , \tag{3.37}$$

or the operational-space formulation,

$$\boldsymbol{\Lambda}(\boldsymbol{x})\dot{\mathbf{v}}+\boldsymbol{\mu}(\boldsymbol{x},\mathbf{v})+\boldsymbol{\rho}(\boldsymbol{x})=\mathbf{f}\,. \tag{3.38}$$

These equations show the functional dependencies explicitly: $\boldsymbol{H}$ is a function of $\mathbf{q}$, $\boldsymbol{\Lambda}$ is a function of $\boldsymbol{x}$, and so on. Once these dependencies are understood, they are usually omitted. In (3.38), $\boldsymbol{x}$ is a vector of operational-space coordinates, while $\mathbf{v}$ and $\mathbf{f}$ are spatial vectors denoting the velocity of the end-effector and the external force acting on it. If the robot is redundant, then the coefficients of this equation must be defined as functions of $\mathbf{q}$ and $\dot{\mathbf{q}}$ rather than $\boldsymbol{x}$ and $\mathbf{v}$.

These two equations are further explained below, along with a description of the Lagrange formulation of (3.37), and the impulsive equations of motion for impact.

3.3.1 Joint-Space Formulation

The symbols $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\boldsymbol{\tau}$ denote n-dimensional vectors of joint position, velocity, acceleration and force variables, respectively, where n is the number of degrees of motion freedom of the robot mechanism. $\boldsymbol{H}$ is an $n\times n$ symmetric, positive-definite matrix, and is called the generalized, or joint-space, inertia matrix (JSIM). $\boldsymbol{C}$ is an $n\times n$ matrix such that $\boldsymbol{C}\,\dot{\mathbf{q}}$ is the vector of Coriolis and centrifugal terms (collectively known as *velocity product* terms); and $\boldsymbol{\tau}_{\mathrm{g}}$ is the vector of gravity terms. More terms can be added to this equation, as required, to account for other dynamical effects (e.g., viscous friction). The effects of a force $\mathbf{f}$ exerted on the mechanism at the end-effector can be accounted for by adding the term $\boldsymbol{J}^{\mathrm{T}}\mathbf{f}$ to the right side of (3.37), where $\boldsymbol{J}$ is the Jacobian of the end-effector (Sect. 2.8.1).

$\mathbf{q}$ specifies the coordinates of a point in the mechanism's configuration space. If the mechanism is a kinematic tree (Sect. 3.4), then $\mathbf{q}$ contains every joint variable in the mechanism, otherwise it contains only an independent subset. The elements of $\mathbf{q}$ are generalized coordinates. Likewise, the elements of $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\boldsymbol{\tau}$ are generalized velocities, accelerations, and forces.

3.3.2 Lagrange Formulation

Various methods exist for deriving the terms in (3.37). The two that are most commonly used in robotics are the Newton–Euler formulation and the Lagrange formulation. The former works directly with Newton's and Euler's equations for a rigid body, which are contained within the spatial equation of motion, (3.35). This formulation is especially amenable to the development of efficient recursive algorithms for dynamics computations, such as those described in Sects. 3.5 and 3.6.

The Lagrange formulation proceeds via the Lagrangian of the robot mechanism,

$$L = T - U \,, \tag{3.39}$$

where T and U are the total kinetic and potential energy, respectively, of the mechanism. The kinetic energy is given by

$$T = \frac{1}{2}\dot{\mathbf{q}}^{\mathrm{T}}\boldsymbol{H}\dot{\mathbf{q}} \,. \tag{3.40}$$

The dynamic equations of motion can then be developed using Lagrange's equation for each generalized coordinate

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i \,. \tag{3.41}$$

The resulting equation can be written in scalar form

$$\sum_{j=1}^{n} H_{ij}\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} C_{ijk}\dot{q}_j\dot{q}_k + \tau_{gi} = \tau_i \,, \tag{3.42}$$

which shows the structure of the velocity-product terms. C_{ijk} are known as Christoffel symbols of the first type, and are given by

$$C_{ijk} = \frac{1}{2}\left(\frac{\partial H_{ij}}{\partial q_k} + \frac{\partial H_{ik}}{\partial q_j} - \frac{\partial H_{jk}}{\partial q_i}\right) \,. \tag{3.43}$$

They are functions of only the position variables, q_i. The elements of $\boldsymbol{C}$ in (3.37) can be defined as

$$C_{ij} = \sum_{k=1}^{n} C_{ijk}\dot{q}_k \,. \tag{3.44}$$

However, $\boldsymbol{C}$ is not unique, and other definitions are possible.

With the choice of $\boldsymbol{C}$ given in (3.44), it is possible to show that the matrix $\boldsymbol{N}$, given by

$$\boldsymbol{N}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\boldsymbol{H}}(\mathbf{q}) - 2\boldsymbol{C}(\mathbf{q}, \dot{\mathbf{q}}) \,, \tag{3.45}$$

is skew-symmetric [3.18]. Thus, for any $n \times 1$ vector $\boldsymbol{\alpha}$,

$$\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{N}(\mathbf{q}, \dot{\mathbf{q}})\,\boldsymbol{\alpha} = 0 \,. \tag{3.46}$$

This property is quite useful in control, especially when considering $\boldsymbol{\alpha} = \dot{\mathbf{q}}$, which gives

$$\dot{\mathbf{q}}^{\mathrm{T}}\boldsymbol{N}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} = 0 \,. \tag{3.47}$$

By applying the principle of conservation of energy, it can be shown that (3.47) holds for any choice of the matrix $\boldsymbol{C}$ [3.18, 19].

3.3.3 Operational-Space Formulation

In (3.38), $\boldsymbol{x}$ is a 6-D vector of operational-space coordinates giving the position and orientation of the robot's end-effector; $\mathbf{v}$ is the velocity of the end-effector; and $\mathbf{f}$ is the force exerted on the end-effector. $\boldsymbol{x}$ is typically a list of Cartesian coordinates, and Euler angles or quaternion components, and is related to $\mathbf{v}$ via a differential equation of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{E}(\boldsymbol{x})\mathbf{v} \,. \tag{3.48}$$

$\boldsymbol{\Lambda}$ is the operational-space inertia matrix, which is the apparent inertia of the end-effector taking into account the effect of the rest of the robot's mechanism (i. e., it is an articulated-body inertia). $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ are vectors of velocity-product and gravity terms, respectively.

Operational space (also known as task space) is the space in which high-level motion and force commands are issued and executed. The operational-space formulation is therefore particularly useful in the context of motion and force control systems (Sects. 8.2 and 9.2). Equation (3.38) can be generalized to operational spaces with dimensions other than six, and to operational spaces that incorporate the motions of more than one end-effector [3.20].

The terms in (3.37) and (3.38) are related by the following formulae

$$\mathbf{v} = \boldsymbol{J}\dot{\mathbf{q}} \,, \tag{3.49}$$

$$\dot{\mathbf{v}} = \boldsymbol{J}\ddot{\mathbf{q}} + \dot{\boldsymbol{J}}\dot{\mathbf{q}} \,, \tag{3.50}$$

$$\boldsymbol{\tau} = \boldsymbol{J}^{\mathrm{T}}\mathbf{f} \,, \tag{3.51}$$

$$\boldsymbol{\Lambda} = (\boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{J}^{\mathrm{T}})^{-1} \,, \tag{3.52}$$

$$\boldsymbol{\mu} = \boldsymbol{\Lambda}(\boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{C}\dot{\mathbf{q}} - \dot{\boldsymbol{J}}\dot{\mathbf{q}}) \,, \tag{3.53}$$

and

$$\boldsymbol{\rho} = \boldsymbol{\Lambda}\boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{\tau}_{\mathrm{g}} \,. \tag{3.54}$$

These equations assume that $m \leq n$ (m is the dimension of operational-space coordinates), and that the Jacobian $\boldsymbol{J}$ has full rank. More details can be found in [3.21].

3.3.4 Impact Model

If a robot strikes a rigid body in its environment, then an impulsive force arises at the moment of impact and causes a step change in the robot's velocity. Let us assume that the impact occurs between the end effector and a rigid body in the environment, and that a spatial impulse of $\mathbf{f}'$ is exerted on the end-effector. This impulse causes a step change of $\Delta\mathbf{v}$ in the end-effector's velocity; and the two are related by the operational-space equation of impulsive motion [3.22],

$$\boldsymbol{\Lambda}\Delta\mathbf{v} = \mathbf{f}' \,. \tag{3.55}$$

In joint space, the equation of impulsive motion for a robot mechanism is

$$\boldsymbol{H}\Delta\dot{\mathbf{q}} = \boldsymbol{\tau}' \,, \tag{3.56}$$

where $\boldsymbol{\tau}'$ and $\Delta\dot{\mathbf{q}}$ denote the joint-space impulse and velocity change, respectively. In the case of a collision involving the robot's end-effector, we have

$$\boldsymbol{\tau}' = \boldsymbol{J}^{\mathrm{T}}\mathbf{f}' \tag{3.57}$$

and

$$\Delta\mathbf{v} = \boldsymbol{J}\Delta\dot{\mathbf{q}} \,, \tag{3.58}$$

which follow from (3.51) and (3.49). Equations (3.55)–(3.57) imply that

$$\Delta\dot{\mathbf{q}} = \bar{\boldsymbol{J}}\Delta\mathbf{v} \,, \tag{3.59}$$

where $\bar{\boldsymbol{J}}$ is the inertia-weighted pseudoinverse of $\boldsymbol{J}$ and is given by

$$\bar{\boldsymbol{J}} = \boldsymbol{H}^{-1}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Lambda} \,. \tag{3.60}$$

$\bar{\boldsymbol{J}}$ is also known as the dynamically consistent generalized inverse of the Jacobian matrix [3.21]. Note that the expression $\boldsymbol{\Lambda}\boldsymbol{J}\boldsymbol{H}^{-1}$, which appears in (3.53) and (3.54), is equal to $\bar{\boldsymbol{J}}^{\mathrm{T}}$ since $\boldsymbol{H}$ and $\boldsymbol{\Lambda}$ are both symmetric. Although we have introduced $\bar{\boldsymbol{J}}$ in the context of impulsive dynamics, it is more typically used in normal (i. e., non-impulsive) dynamics equations.

3.4 Dynamic Models of Rigid-Body Systems

A basic rigid-body model of a robot mechanism has four components: a connectivity graph, link and joint geometry parameters, link inertia parameters, and a set of joint models. To this model, one can add various force-producing elements, such as springs, dampers, joint friction, actuators, and drives. The actuators and drives, in particular, may have quite elaborate dynamic models of their own. It is also possible to add extra motion freedoms to model elasticity in the joint bearings or links (Chap. 11). This section describes a basic model. More on this topic can be found in books such as [3.3, 8, 23].

3.4.1 Connectivity

A connectivity graph is an undirected graph in which each node represents a rigid body and each arc represents a joint. The graph must be connected; and exactly one node represents a fixed base or reference frame. If the graph represents a mobile robot (i. e., a robot that is not connected to a fixed base), then it is necessary to introduce a fictitious 6-DOF joint between the fixed base and any one body in the mobile robot. The chosen body is then known as a *floating base*. If a single graph is to represent a collection of mobile robots, then each robot has its own floating base, and each floating base has its own 6-DOF joint. Note that a 6-DOF joint imposes no constraints on the two bodies it connects, so the introduction of a 6-DOF joint alters the connectivity of the graph without altering the physical properties of the system it represents.

In graph-theory terminology, a loop is an arc that connects a node to itself, and a cycle is a closed path that does not traverse any arc more than once. In the connectivity graph of a robot mechanism, loops are not allowed, and cycles are called kinematic loops. A mechanism that contains kinematic loops is called a closed-loop mechanism; and a mechanism that does not is called an open-loop mechanism or a kinematic tree. Every closed-loop mechanism has a spanning tree, which defines an open-loop mechanism, and every joint that is not in the spanning tree is called a loop-closing joint. The joints in the tree are called tree joints.

The fixed base serves as the root node of a kinematic tree, and the root node of any spanning tree on a closed-loop mechanism. A kinematic tree is said to be branched if at least one node has at least two children, and unbranched otherwise. An unbranched kinematic

tree is also called a kinematic chain, and a branched tree can be called a branched kinematic chain. A typical industrial robot arm, without a gripper, is a kinematic chain, while a typical humanoid robot is a kinematic tree with a floating base.

In a system containing N_B moving bodies and N_J joints, where N_J includes the 6-DOF joints mentioned above, the bodies and joints are numbered as follows. First, the fixed base is numbered body 0. The other bodies are then numbered from 1 to N_B in any order such that each body has a higher number than its parent. If the system contains kinematic loops then one must first choose a spanning tree, and commit to that choice, since the identity of a body's parent is determined by the spanning tree. This style of numbering is called a *regular numbering scheme*.

Having numbered the bodies, we number the tree joints from 1 to N_B such that joint i connects body i to its parent. The loop-closing joints, if any, are then numbered from N_B+1 to N_J in any order. Each loop-closing joint k closes one independent kinematic loop, and we number the loops from 1 to N_L (where $N_L = N_J - N_B$ is the number of independent loops) such that loop l is the one closed by joint $k = N_B + l$. Kinematic loop l is the unique cycle in the graph that traverses joint k, but does not traverse any other loop-closing joint.

For an unbranched kinematic tree, these rules produce a unique numbering in which the bodies are numbered consecutively from base to tip, and the joints are numbered such that joint i connects bodies i and $i-1$. In all other cases, regular numberings are not unique.

Although the connectivity graph is undirected, it is necessary to assign a direction to each joint for the purpose of defining joint velocity and force. This is necessary for both tree joints and loop-closing joints. Specifically, a joint is said to connect from one body to another. We may call them the predecessor $p(i)$ and successor $s(i)$ for joint i, respectively. Joint velocity is then defined as the velocity of the successor relative to the predecessor; and joint force is defined as a force acting on the successor. It is standard practice (but not a necessity) for all tree joints to connect from the parent to the child.

The connectivity of a kinematic tree, or the spanning tree on a closed-loop mechanism, is described by an N_B-element array of parent body numbers, where the i-th element $p(i)$ is the parent of body i. Note that the parent $p(i)$ for body i is also the predecessor $p(i)$ for joint i, and thus the common notation. Many algorithms rely on the property $p(i) < i$ to perform their calculations in the correct order. The set of all body numbers for the children of body i, $c(i)$, is also useful in many recursive algorithms.

The connectivity data for kinematic loops may be described in a variety of ways. A representation that facilitates use in recursive algorithms includes the following conventions. Loop-closing joint k joins bodies $p(k)$ (the predecessor) and $s(k)$ (the successor). The set $LR(i)$ for body i gives the numbers of the loops for which body i is the root. Using the property $p(i) < i$ for bodies in the spanning tree, the root of a loop is chosen as the body with the lowest number. In addition, the set $LB(i)$ for body i gives the numbers of the loops to which body i belongs but is not the root.

An example of a closed-loop system is given in Fig. 3.3. The system consists of a humanoid mechanism with topologically varying contacts, with the environment and within the mechanism, which form closed loops. The system has $N_B = 16$ moving bodies and $N_J = 19$ joints with $N_L = N_J - N_B = 3$ loops. The main body (1) is considered to be a *floating base* for this mobile robot system. It is connected to the fixed base (0) through a fictitious 6-DOF joint (1). To complete the example, the loop-closing joint and the body numbers $p(k)$ and $s(k)$ as well as the root body for each loop are given in Table 3.2. The body-based sets $c(i)$ and

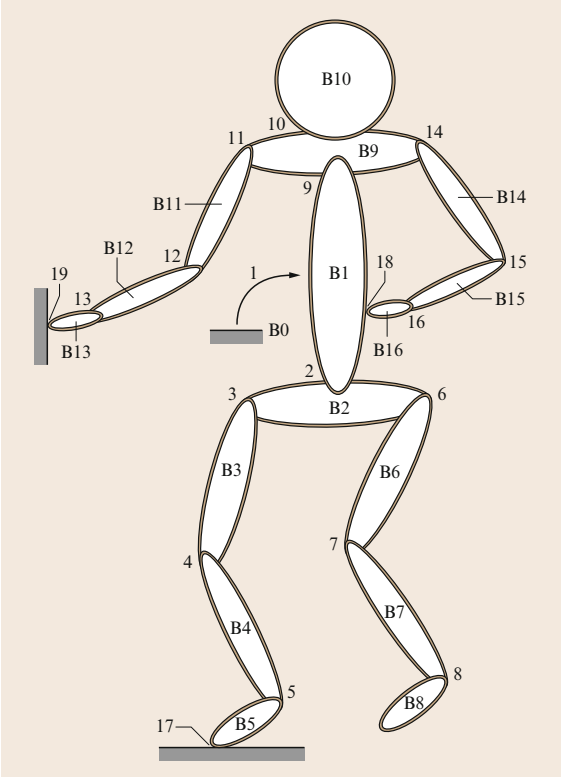

Fig. 3.3 Humanoid robot example. Note: to distinguish between body numbers and joint numbers in this figure, body numbers are preceded by a B for clarity

$LB(i)$ are given in Table 3.3. Note that $LR(0) = \{1, 3\}$ and $LR(1) = \{2\}$ and all other LR sets are null for this example.

3.4.2 Link Geometry

When two bodies are connected by a joint, a complete description of the connection consists of a description of the joint itself, and the locations of two coordinate frames, one in each body, which specify where in each body the joint is located. If there are N_J joints in the system, then there are a total of $2N_J$ joint-attachment frames. One half of these frames are identified with the numbers 1 to N_J, and the remainder with the labels $J1$ to JN_J. Each joint i connects from frame Ji to frame i.

For joints 1 to N_B (i. e., the tree joints), frame i is rigidly attached to body i. For joints $N_B + 1$ to N_J, frame k for loop-closing joint k will be rigidly attached to body $s(k)$. The second coordinate frame Ji is attached to the predecessor $p(i)$ for each joint i, whether it is a tree joint or a loop-closing joint. Coordinate frame Ji provides a base frame for joint i in that the joint rotation and/or translation is defined relative to this frame.

Figure 3.4 shows the coordinate frames and transforms associated with each joint in the system. The overall transform from frame $p(i)$ coordinates to frame i coordinates for a tree joint is given by

$$ {}^{i}\boldsymbol{X}_{p(i)} = {}^{i}\boldsymbol{X}_{Ji}\,{}^{Ji}\boldsymbol{X}_{p(i)} = \boldsymbol{X}_J(i)\boldsymbol{X}_L(i) \,. \tag{3.61} $$

The transform $\boldsymbol{X}_L(i)$ is a fixed link transform which sets the base frame Ji of joint i relative to $p(i)$. It may be used to transform spatial motion vectors from $p(i)$ to Ji coordinates. The transform $\boldsymbol{X}_J(i)$ is a variable joint transform which completes the transformation across joint i from Ji to i coordinates.

Similarly, the overall transform from frame $p(k)$ coordinates to frame k coordinates for a loop-closing joint is given by

$$ {}^{k}\boldsymbol{X}_{p(k)} = {}^{k}\boldsymbol{X}_{Jk}\,{}^{Jk}\boldsymbol{X}_{p(k)} = \boldsymbol{X}_J(k)\,\boldsymbol{X}_{L1}(k) \,. \tag{3.62} $$

An additional transform $\boldsymbol{X}_{L2}(k)$ is defined from frame $s(k)$ coordinates to frame k coordinates and is given by

$$ \boldsymbol{X}_{L2}(k) = {}^{k}\boldsymbol{X}_{s(k)} \,. \tag{3.63} $$

Link and joint geometry data can be specified in a variety of different ways. The most common method is to use *Denavit–Hartenberg* parameters [3.24]. However, standard Denavit–Hartenberg parameters are not completely general, and are insufficient for describing the geometry for a branched kinematic tree, or for a mechanism containing certain kinds of multi-DOF joints. A modified form of Denavit–Hartenberg parameters [3.2] is used for single-DOF joints in this Handbook (Sect. 2.4). The parameters have been extended for branched kinematic trees [3.23] and closed-loop mechanisms.

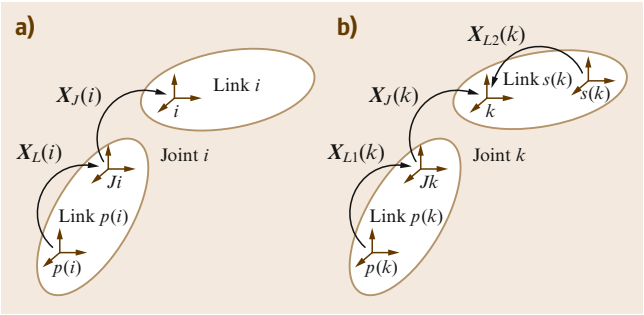

Fig.3.4a,b Coordinate frames and transforms associated with **(a)** a tree joint and **(b)** a loop-closing joint

Table 3.2 Loop-closing joints and roots of loops for the humanoid example

Loop l	Loop-closing joint k	$p(k)$	$s(k)$	Root
1	17	0	5	0
2	18	16	1	1
3	19	0	13	0

Table 3.3 Body-based sets for the humanoid example

Body i	$c(i)$	$LB(i)$	Body i	$c(i)$	$LB(i)$
0	1		9	10, 11, 14	2, 3
1	2, 9	1, 3	10		
2	3, 6	1	11	12	3
3	4	1	12	13	3
4	5	1	13		3
5		1	14	15	2
6	7		15	16	2
7	8		16		2
8					

3.4.3 Link Inertias

The link inertia data consists of the masses, positions of centers of mass, and rotational inertias of each link in the mechanism. The inertia parameters for link i are expressed in coordinate frame i, and are therefore constants.

3.4.4 Joint Models

The relationship between connected links is described using the general joint model of *Roberson* and *Schwertassek* [3.3]. For a kinematic tree or spanning tree on

a closed-loop mechanism, an $n_i \times 1$ vector, $\dot{\boldsymbol{q}}_i$, relates the velocity of link i to the velocity of its parent, link $p(i)$, where n_i is the number of degrees of freedom at the joint connecting the two links. For a loop-closing joint in a closed-loop mechanism, the relationship is between the velocity of link $s(i)$ (the successor) and the velocity of link $p(i)$ (the predecessor). In either case, the relationship is between the velocity of coordinate frames i and Ji.

Let $\mathbf{v}_{\text{rel}}$ and $\mathbf{a}_{\text{rel}}$ denote the velocity and acceleration across joint i, that is, the velocity and acceleration of link $s(i)$ relative to $p(i)$. The free modes of the joint are represented by the $6 \times n_i$ matrix $\boldsymbol{\Phi}_i$, such that $\mathbf{v}_{\text{rel}}$ and $\mathbf{a}_{\text{rel}}$ are given as follows

$$\mathbf{v}_{\text{rel}} = \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i \tag{3.64}$$

and

$$\mathbf{a}_{\text{rel}} = \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i \,, \tag{3.65}$$

where $\boldsymbol{\Phi}_i$ and $\dot{\boldsymbol{\Phi}}_i$ depend on the type of joint [3.3]. The matrix $\boldsymbol{\Phi}_i$ has full column rank, so we can define a complementary matrix, $\boldsymbol{\Phi}_i^{\text{c}}$, such that the 6×6 matrix $\left(\boldsymbol{\Phi}_i \boldsymbol{\Phi}_i^{\text{c}}\right)$ is invertible. We can regard the columns of this matrix as forming a basis on M^6 such that the first n_i basis vectors define the directions in which motion is allowed, and the remaining $6 - n_i = n_i^{\text{c}}$ vectors define directions in which motion is not allowed. Thus, $\boldsymbol{\Phi}_i^{\text{c}}$ represents the constrained modes of joint i.

The force transmitted across joint i from its predecessor to its successor, $\mathbf{f}_i$, is given as follows

$$\mathbf{f}_i = \left(\boldsymbol{\Psi}_i \boldsymbol{\Psi}_i^{\text{c}}\right) \begin{pmatrix} \boldsymbol{\tau}_i \\ \boldsymbol{\lambda}_i \end{pmatrix} , \tag{3.66}$$

where $\boldsymbol{\tau}_i$ is the $n_i \times 1$ vector of applied forces along the free modes, $\boldsymbol{\lambda}_i$ is the $(6 - n_i) \times 1$ vector of constraint forces, and $\boldsymbol{\Psi}_i$ and $\boldsymbol{\Psi}_i^{\text{c}}$ are computed as follows

$$\left(\boldsymbol{\Psi}_i \boldsymbol{\Psi}_i^{\text{c}}\right) = \left(\boldsymbol{\Phi}_i \boldsymbol{\Phi}_i^{\text{c}}\right)^{-T} . \tag{3.67}$$

For most common joint types, it is possible to choose $\boldsymbol{\Phi}_i$ and $\boldsymbol{\Phi}_i^{\text{c}}$ such that the matrix $(\boldsymbol{\Phi}_i \boldsymbol{\Phi}_i^{\text{c}})$ is numerically orthonormal, so that $(\boldsymbol{\Psi}_i \boldsymbol{\Psi}_i^{\text{c}})$ is numerically equal to $(\boldsymbol{\Phi}_i \boldsymbol{\Phi}_i^{\text{c}})$. Note that (3.67) implies the following relationships: $(\boldsymbol{\Psi}_i)^{\text{T}} \boldsymbol{\Phi}_i = \mathbf{1}_{n_i \times n_i}$, $(\boldsymbol{\Psi}_i)^{\text{T}} \boldsymbol{\Phi}_i^{\text{c}} = \mathbf{0}_{n_i \times (6-n_i)}$, $(\boldsymbol{\Psi}_i^{\text{c}})^{\text{T}} \boldsymbol{\Phi}_i = \mathbf{0}_{(6-n_i) \times n_i}$, and $(\boldsymbol{\Psi}_i^{\text{c}})^{\text{T}} \boldsymbol{\Phi}_i^{\text{c}} = \mathbf{1}_{(6-n_i) \times (6-n_i)}$. When applied to (3.66), the following useful relationship results

$$\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^{\text{T}} \mathbf{f}_i \,. \tag{3.68}$$

The value of $\dot{\boldsymbol{\Phi}}_i$ in (3.65) depends on the type of joint. The general formula is

$$\dot{\boldsymbol{\Phi}}_i = \mathring{\boldsymbol{\Phi}}_i + \mathbf{v}_i \times \boldsymbol{\Phi}_i \,, \tag{3.69}$$

where $\mathbf{v}_i$ is the velocity of link i, and $\mathring{\boldsymbol{\Phi}}_i$ is the apparent derivative of $\boldsymbol{\Phi}_i$, as seen by an observer moving with link i, and is given by

$$\mathring{\boldsymbol{\Phi}}_i = \frac{\partial \boldsymbol{\Phi}_i}{\partial \boldsymbol{q}_i} \dot{\boldsymbol{q}}_i \,. \tag{3.70}$$

For most common joint types, $\mathring{\boldsymbol{\Phi}}_i = \mathbf{0}$.

Single-DOF joints ($n_i = 1$) are especially straightforward to work with when using the Denavit–Hartenberg convention. Motion is chosen along (prismatic) or about (revolute) the $\hat{\mathbf{z}}_i$ coordinate axis. In this case, $\boldsymbol{\Phi}_i = (0\,0\,0\,0\,0\,1)^{\text{T}}$ for a prismatic joint and $\boldsymbol{\Phi}_i = (0\,0\,1\,0\,0\,0)^{\text{T}}$ for a revolute joint. Also, $\mathring{\boldsymbol{\Phi}}_i = \mathbf{0}$.

The fictitious 6-DOF joint for a floating base for a mobile robot is also handled relatively easily. For this case, $\boldsymbol{\Phi}_i = \mathbf{1}$ (6×6 identity matrix) and $\mathring{\boldsymbol{\Phi}}_i = \mathbf{0}$.

The revolute joint and floating-base joint, as well as the universal joint ($n_i = 2$) and spherical joint ($n_i = 3$) are illustrated in the example in the next section. For additional details on joint kinematics, see Sect. 2.3.

3.4.5 Example System

In order to illustrate the conventions used for the link and joint models, coordinate frames are attached to the first five links (bodies) and fixed base of the humanoid robot as shown in Fig. 3.5. Note that frame Ji is attached to link $p(i) = i - 1$ for each of the five joints. For this example, the origin of frame $J1$ is set coincident with the origin of frame 0, and the origins of frames $J2$, $J3$, $J4$, and $J5$ are coincident with the origins of frames 2, 3, 4, and 5, respectively.

Note that $J1$ could be set at any position/orientation on the fixed base (B0) to permit the most convenient

Table 3.4 Number of degrees of freedom (n_i), fixed rotation $\left({}^{Ji}\boldsymbol{R}_{p(i)}\right)$ and position $\left({}^{p(i)}\boldsymbol{p}_{Ji}\right)$ from frame $p(i)$ to base frame Ji for joint i of the example system. Note that $2l_i$ is the nominal length of link i along its long axis

Joint	n_i	${}^{Ji}\boldsymbol{R}_{p(i)}$	${}^{p(i)}\boldsymbol{p}_{Ji}$
1	6	$\mathbf{1}_{3\times3}$	$\mathbf{0}_{3\times1}$
2	1	$\mathbf{1}_{3\times3}$	$\begin{pmatrix} 0 \\ 0 \\ -l_1 \end{pmatrix}$
3	3	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -l_2 \\ 0 \end{pmatrix}$
4	1	$\mathbf{1}_{3\times3}$	$\begin{pmatrix} 0 \\ 2l_3 \\ 0 \end{pmatrix}$
5	2	$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 2l_4 \\ 0 \end{pmatrix}$

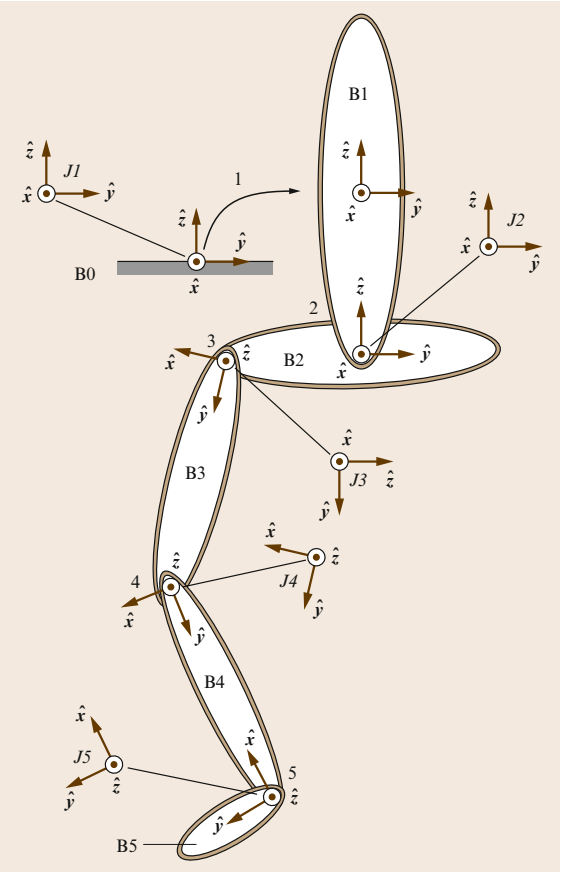

Fig. 3.5 Coordinate frames for the first five links and joints of the humanoid robot example

representation for the motion of the floating base (B1) relative to the fixed base. Also, the origin of $J2$ could be set anywhere along $\hat{\boldsymbol{z}}_2$.

The number of degrees of freedom, and fixed rotation and position of the base frames Ji for each joint of the example system, are given in Table 3.4. The rotation ${}^{Ji}\boldsymbol{R}_{p(i)}$ transforms 3-D vectors in $p(i)$ coordinates to Ji coordinates. The position ${}^{p(i)}\boldsymbol{p}_{Ji}$ is the vector giving the position of the origin O_{Ji} relative to $O_{p(i)}$, expressed in $p(i)$ coordinates. The spatial transform $\boldsymbol{X}_L(i) = {}^{Ji}\boldsymbol{X}_{p(i)}$ may be composed from these 3-D quantities through the equation for ${}^{B}\boldsymbol{X}_A$ in Table 3.1. The humanoid has a floating base, the torso, a revolute joint between the torso and pelvis (about $\hat{\boldsymbol{z}}_2$), a spherical joint at the hip, a revolute joint at the knee, and a universal joint at the ankle. As shown in Fig. 3.5, the leg is slightly bent and the foot is turned out to the side ($\approx 90^\circ$ rotation about $\hat{\boldsymbol{y}}_3$ at the hip).

The free modes, velocity variables, and position variables for all of the joint types in the humanoid are given in Tables 2.5 and 2.6. The expressions for ${}^{j}\boldsymbol{R}_i$ and ${}^{j}\boldsymbol{p}_i$ in these tables give ${}^{i}\boldsymbol{R}_{Ji}^{\mathrm{T}}$ and ${}^{Ji}\boldsymbol{p}_i$, respectively, through which the joint transform $\boldsymbol{X}_J(i) = {}^{i}\boldsymbol{X}_{Ji}$ may be composed. Revolute joints follow the Denavit–Hartenberg convention with rotation about the $\hat{\boldsymbol{z}}_i$ axis. The ankle has a pitch rotation of α_5 about the $\hat{\boldsymbol{z}}_{J5}$ axis followed by a roll rotation of β_5 about the $\hat{\boldsymbol{y}}_5$ axis (see the Z–Y–X Euler angle definitions in Table 2.1). The hip is modeled as a ball-and-socket, spherical joint. To avoid the singularities that are associated with Euler angles, the quaternion $\boldsymbol{\epsilon}_i$ may be used to represent the orientation at the hip. The relationship between the quaternion rate $\dot{\boldsymbol{\epsilon}}_i$ and the relative rotation rate $\boldsymbol{\omega}_{i\,\mathrm{rel}}$ is given in (2.8) of the Handbook.

The floating base uses the position of the torso ${}^{0}\boldsymbol{p}_1$ and quaternion $\boldsymbol{\epsilon}_1$ for its position and orientation state variables, respectively. The position of the torso may be computed by integrating the velocity for the link, as expressed in fixed base coordinates: ${}^{0}\boldsymbol{v}_1 = {}^{0}\boldsymbol{R}_1\boldsymbol{v}_1$, where $\boldsymbol{v}_1$ is the velocity of the torso in moving coordinates.

Note that $\mathring{\boldsymbol{\Phi}}_i = 0$ for all joints except the universal joint. Since the components of $\hat{\boldsymbol{z}}_{J5}$ in link 5 coordinates vary with β_5, $\hat{\boldsymbol{z}}_{J5} \neq 0$. See Sect. 2.3 of the Handbook for further details of the joint kinematics.

3.5 Kinematic Trees

The dynamics of a kinematic tree is simpler, and easier to calculate, than the dynamics of a closed-loop mechanism. Indeed, many algorithms for closed-loop mechanisms work by first calculating the dynamics of a spanning tree, and then subjecting it to the loop-closure constraints.

This section describes the following dynamics algorithms for kinematic trees: the recursive Newton–Euler algorithm (RNEA) for inverse dynamics, the articulated-body algorithm (ABA) for forward dynamics, the composite-rigid-body algorithm (CRBA) for calculating the joint-space inertia matrix (JSIM), and two algorithms to calculate the operational-space inertia matrix (OSIM). Implementations of the first three can be found in [3.16].

3.5.1 The Recursive Newton–Euler Algorithm

This is an $O(n)$ algorithm for calculating the inverse dynamics of a fixed-base kinematic tree, and is based on the very efficient RNEA of *Luh* et al. [3.4]. A floating-

base version can be found in [3.8, 15]. Given the joint position and velocity variables, this algorithm calculates the applied joint torque/force variables required to produce a given set of joint accelerations.

The link velocities and accelerations are first computed through an outward recursion from the fixed base to the leaf links of the tree. The required forces on each link are computed using the Newton–Euler equations (3.35) during this recursion. A second, inward recursion uses the force balance equations at each link to compute the spatial force across each joint and the value of each joint torque/force variable. The key step for computational efficiency is to refer most quantities to local link coordinates. Also, the effects of gravity on each link are efficiently included in the equations by accelerating the base of the mechanism upward.

The calculation proceeds in four steps, as follows, with two steps in each of the two recursions.

Step 1

Calculate the velocity and acceleration of each link in turn, starting with the known velocity and acceleration of the fixed base, and working towards the tips (i. e., the leaf nodes in the connectivity graph).

The velocity of each link in a kinematic tree is given by the recursive formula

$$\mathbf{v}_i = \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i \,, \quad (\mathbf{v}_0 = \mathbf{0}) \,, \tag{3.71}$$

where $\mathbf{v}_i$ is the velocity of link i, $\boldsymbol{\Phi}_i$ is the motion matrix of joint i, and $\dot{\boldsymbol{q}}_i$ is the vector of joint velocity variables for joint i.

The equivalent formula for accelerations is obtained by differentiating (3.71), giving

$$\mathbf{a}_i = \mathbf{a}_{p(i)} + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i \,, \quad (\mathbf{a}_0 = \mathbf{0}) \,, \tag{3.72}$$

where $\mathbf{a}_i$ is the acceleration of link i, and $\ddot{\boldsymbol{q}}_i$ is the vector of joint acceleration variables.

The effect of a uniform gravitational field on the mechanism can be simulated by initializing $\mathbf{a}_0$ to $-\mathbf{a}_g$ instead of zero, where $\mathbf{a}_g$ is the gravitational acceleration vector. In this case, $\mathbf{a}_i$ is not the true acceleration of link i, but the sum of its true acceleration and $-\mathbf{a}_g$.

Step 2

Calculate the equation of motion for each link. This step computes the forces required to cause the accelerations calculated in step 1. The equation of motion for link i is

$$\mathbf{f}_i^{\mathrm{a}} = \boldsymbol{I}_i \mathbf{a}_i + \mathbf{v}_i \times \boldsymbol{I}_i \mathbf{v}_i \,, \tag{3.73}$$

where $\boldsymbol{I}_i$ is the spatial inertia of link i, and $\mathbf{f}_i^{\mathrm{a}}$ is the net force acting on link i.

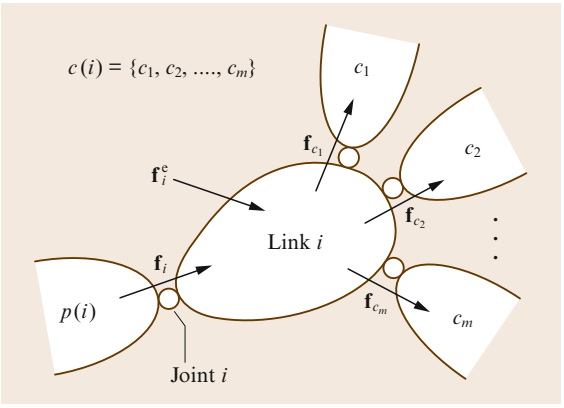

Fig. 3.6 Forces acting on link i

Step 3

Calculate the spatial force across each joint. Referring to Fig. 3.6, the net force acting on link i is

$$\mathbf{f}_i^{\mathrm{a}} = \mathbf{f}_i^{\mathrm{e}} + \mathbf{f}_i - \sum_{j \in c(i)} \mathbf{f}_j \,,$$

where $\mathbf{f}_i$ is the force transmitted across joint i, $\mathbf{f}_i^{\mathrm{e}}$ is the sum of all relevant external forces acting on link i, and $c(i)$ is the set of children of link i. Rearranging this equation gives the following recursive formula for calculating the joint forces

$$\mathbf{f}_i = \mathbf{f}_i^{\mathrm{a}} - \mathbf{f}_i^{\mathrm{e}} + \sum_{j \in c(i)} \mathbf{f}_j \,, \tag{3.74}$$

where i iterates from N_B to 1.

$\mathbf{f}_i^{\mathrm{e}}$ 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.

If gravity has not been simulated by a fictitious base acceleration, then the gravitational force acting on link i must be included in $\mathbf{f}_i^{\mathrm{e}}$.

Step 4

Calculate the joint force variables, $\boldsymbol{\tau}_i$. By definition, they are given by the equation

$$\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^{\mathrm{T}} \mathbf{f}_i \,. \tag{3.75}$$

Coordinate-Free Algorithm

Equations (3.71)–(3.75) imply the algorithm shown in Algorithm 3.1, which is the coordinate-free version of the RNEA. This is the simplest form of the algorithm, and it is suitable for mathematical analysis and related purposes. However, it is not suitable for numerical

computation because a numerical version must use coordinate vectors.

Algorithm 3.1 Coordinate-free recursive Newton–Euler algorithm (RNEA) for inverse dynamics

$\mathbf{v}_0 = \mathbf{0}$
$\mathbf{a}_0 = -\mathbf{a}_g$
for $i = 1$ **to** N_B **do**
  $\mathbf{v}_i = \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i$
  $\mathbf{a}_i = \mathbf{a}_{p(i)} + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i$
  $\mathbf{f}_i = \boldsymbol{I}_i \mathbf{a}_i + \mathbf{v}_i \times \boldsymbol{I}_i \mathbf{v}_i - \mathbf{f}_i^e$
end for
for $i = N_B$ **to** 1 **do**
  $\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^T \mathbf{f}_i$
  if $p(i) \neq 0$ **then**
    $\mathbf{f}_{p(i)} = \mathbf{f}_{p(i)} + \mathbf{f}_i$
  end if
end for

Link-Coordinates Algorithm

In general, we say that an algorithm is implemented in link coordinates if a coordinate system is defined for each link, and the calculations pertaining to link i are performed in the coordinate system associated with link i. The alternative is to implement the algorithm in absolute coordinates, in which case all calculations are performed in a single coordinate system, typically that of the base link. In practice, the RNEA is more computationally efficient when implemented in link coordinates, and the same is true of most other dynamics algorithms.

To convert the RNEA to link coordinates, we first examine the equations to see which ones involve quantities from more than one link. Equations (3.73) and (3.75) each involve quantities pertaining to link i only, and therefore need no modification. Such equations are said to be local to link i. The remaining equations involve quantities from more than one link, and therefore require the insertion of coordinate transformation matrices. The modified versions of (3.71), (3.72), and (3.74) are

$$\mathbf{v}_i = {}^i\boldsymbol{X}_{p(i)}\, \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i \,, \tag{3.76}$$

$$\mathbf{a}_i = {}^i\boldsymbol{X}_{p(i)}\, \mathbf{a}_{p(i)} + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i \,, \tag{3.77}$$

and

$$\mathbf{f}_i = \mathbf{f}_i^a - {}^i\boldsymbol{X}_0^F\, {}^0\mathbf{f}_i^e + \sum_{j \in c(i)} {}^i\boldsymbol{X}_j^F\, \mathbf{f}_j \,. \tag{3.78}$$

Equation (3.78) assumes that external forces are expressed in absolute (i. e., link 0) coordinates.

Algorithm 3.2 Recursive Newton–Euler algorithm using spatial vectors

inputs: $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, *model* , ${}^0\mathbf{f}_i^e$
output: $\boldsymbol{\tau}$
model data : N_B, jtype(i), $p(i)$, $\boldsymbol{X}_L(i)$, $\boldsymbol{I}_i$

$\mathbf{v}_0 = \mathbf{0}$
$\mathbf{a}_0 = -\mathbf{a}_g$
for $i = 1$ **to** N_B **do**
  $\boldsymbol{X}_J(i) = \text{xjcalc}(\text{jtype}(i), \boldsymbol{q}_i)$
  ${}^i\boldsymbol{X}_{p(i)} = \boldsymbol{X}_J(i)\, \boldsymbol{X}_L(i)$
  if $p(i) \neq 0$ **then**
    ${}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{p(i)}\, {}^{p(i)}\boldsymbol{X}_0$
  end if
  $\boldsymbol{\Phi}_i = \text{pcalc}(\text{jtype}(i), \boldsymbol{q}_i)$
  $\mathring{\boldsymbol{\Phi}}_i = \text{pdcalc}(\text{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
  $\mathbf{v}_i = {}^i\boldsymbol{X}_{p(i)}\, \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i$
  $\boldsymbol{\zeta}_i = \mathring{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i + \mathbf{v}_i \times \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i$
  $\mathbf{a}_i = {}^i\boldsymbol{X}_{p(i)}\, \mathbf{a}_{p(i)} + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \boldsymbol{\zeta}_i$
  $\mathbf{f}_i = \boldsymbol{I}_i \mathbf{a}_i + \mathbf{v}_i \times \boldsymbol{I}_i \mathbf{v}_i - {}^i\boldsymbol{X}_0^{-T}\, {}^0\mathbf{f}_i^e$
end for
for $i = N_B$ **to** 1 **do**
  $\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^T \mathbf{f}_i$
  if $p(i) \neq 0$ **then**
    $\mathbf{f}_{p(i)} = \mathbf{f}_{p(i)} + {}^i\boldsymbol{X}_{p(i)}^T\, \mathbf{f}_i$
  end if
end for

The complete algorithm is shown in Algorithm 3.2. The function `jtype` returns the type code for joint i; the function `xjcalc` calculates the joint transformation matrix for the specified type of joint; and the functions `pcalc` and `pdcalc` calculate $\boldsymbol{\Phi}_i$ and $\mathring{\boldsymbol{\Phi}}_i$. The formulae used by these functions for a variety of joint types can be found in Tables 2.5 and 2.6, bearing in mind that the rotation matrices that must be used are the transposes of those listed in these tables. In the general case, both `pcalc` and `pdcalc` are needed. However, for most common joint types, $\boldsymbol{\Phi}_i$ is a known constant in link coordinates, and $\mathring{\boldsymbol{\Phi}}_i$ is therefore zero. If it is known in advance that all joints will have this property, then the algorithm can be simplified accordingly. The quantities $\boldsymbol{I}_i$ and $\boldsymbol{X}_L(i)$ are known constants in link coordinates, and are part of the data structure describing the robot mechanism.

The last assignment in the first loop initializes each $\mathbf{f}_i$ to the expression $\mathbf{f}_i^a - {}^i\boldsymbol{X}_0^F\, {}^0\mathbf{f}_i^e$ (using the identity ${}^i\boldsymbol{X}_0^F = {}^i\boldsymbol{X}_0^{-T}$). The summation on the right-hand side of (3.78) is then performed in the second loop. This algorithm includes code to calculate ${}^i\boldsymbol{X}_0$, which is used to transform the external forces to link coordinates. If there are no external forces, then this code can be omitted. If there is only a single external force (e.g., a force

at the end-effector of a robot arm) then this code can be replaced with code that transforms the external force vector successively from one link coordinate system to the next, using ${}^{i}\boldsymbol{X}_{p(i)}$.

Note: although the phrase *link coordinates* suggests that we are using moving coordinate frames, the algorithm is in fact implemented in stationary coordinates that happen to coincide with the moving coordinates at the current instant.

3-D Vector RNEA

The original version of the RNEA was developed and expressed using 3-D vectors (e.g. [3.2, 4]). Algorithm 3.3 shows a special case of this algorithm, in which the joints are assumed to be revolute, and the joint axes are assumed to coincide with the z axes of the link coordinate systems. (Without these assumptions, the equations would be a lot longer.) It also assumes that the external forces are zero.

Algorithm 3.3 Recursive Newton–Euler algorithm in 3-D vectors, for revolute joints only

inputs: $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, *model*
output: $\boldsymbol{\tau}$
model data : N_B, $p(i)$, $\boldsymbol{R}_L(i)$, ${}^{p(i)}\boldsymbol{p}_i$, m_i, $\boldsymbol{c}_i$, $\bar{\boldsymbol{I}}_i^{\mathrm{cm}}$

$\boldsymbol{\omega}_0 = \mathbf{0}$
$\dot{\boldsymbol{\omega}}_0 = \mathbf{0}$
$\dot{\boldsymbol{v}}'_0 = -\dot{\boldsymbol{v}}'_{\mathrm{g}}$
for $i = 1$ **to** N_B **do**
  ${}^{i}\boldsymbol{R}_{p(i)} = \mathrm{rotz}(\boldsymbol{q}_i)\,\boldsymbol{R}_L(i)$
  $\boldsymbol{\omega}_i = {}^{i}\boldsymbol{R}_{p(i)}\,\boldsymbol{\omega}_{p(i)} + \hat{\boldsymbol{z}}_i\dot{q}_i$
  $\dot{\boldsymbol{\omega}}_i = {}^{i}\boldsymbol{R}_{p(i)}\,\dot{\boldsymbol{\omega}}_{p(i)} + \left({}^{i}\boldsymbol{R}_{p(i)}\,\boldsymbol{\omega}_{p(i)}\right) \times \hat{\boldsymbol{z}}_i\dot{q}_i + \hat{\boldsymbol{z}}_i\ddot{q}_i$
  $\dot{\boldsymbol{v}}'_i = {}^{i}\boldsymbol{R}_{p(i)}\left(\dot{\boldsymbol{v}}'_{p(i)} + \dot{\boldsymbol{\omega}}_{p(i)} \times {}^{p(i)}\boldsymbol{p}_i + \boldsymbol{\omega}_{p(i)} \times \boldsymbol{\omega}_{p(i)} \times {}^{p(i)}\boldsymbol{p}_i\right)$
  $\boldsymbol{f}_i = m_i(\dot{\boldsymbol{v}}'_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{c}_i + \boldsymbol{\omega}_i \times \boldsymbol{\omega}_i \times \boldsymbol{c}_i)$
  $\boldsymbol{n}_i = \bar{\boldsymbol{I}}_i^{\mathrm{cm}}\dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times \bar{\boldsymbol{I}}_i^{\mathrm{cm}}\boldsymbol{\omega}_i + \boldsymbol{c}_i \times \boldsymbol{f}_i$
end for
for $i = N_B$ **to** 1 **do**
  $\tau_i = \hat{\boldsymbol{z}}_i^{\mathrm{T}}\boldsymbol{n}_i$
  if $p(i) \neq 0$ **then**
    $\boldsymbol{f}_{p(i)} = \boldsymbol{f}_{p(i)} + {}^{i}\boldsymbol{R}_{p(i)}^{\mathrm{T}}\boldsymbol{f}_i$
    $\boldsymbol{n}_{p(i)} = \boldsymbol{n}_{p(i)} + {}^{i}\boldsymbol{R}_{p(i)}^{\mathrm{T}}\boldsymbol{n}_i + {}^{p(i)}\boldsymbol{p}_i \times {}^{i}\boldsymbol{R}_{p(i)}^{\mathrm{T}}\boldsymbol{f}_i$
  end if
end for

In this algorithm, $\dot{\boldsymbol{v}}'_{\mathrm{g}}$ is the linear acceleration due to gravity, expressed in base (link 0) coordinates; `rotz` computes the transpose of the matrix shown in (2.2); $\boldsymbol{R}_L(i)$ is the rotational component of $\boldsymbol{X}_L(i)$; ${}^{i}\boldsymbol{R}_{p(i)}$ is the rotational component of ${}^{i}\boldsymbol{X}_{p(i)}$; `pcalc` and `pdcalc` are not used because $\boldsymbol{\Phi}_i$ is the known constant $(\hat{\boldsymbol{z}}^{\mathrm{T}}\ \mathbf{0}^{\mathrm{T}})^{\mathrm{T}}$; $\dot{\boldsymbol{v}}'_i$ is the linear acceleration of the origin of link i coordinates (O_i), and is the linear component of the classical acceleration of link i; ${}^{p(i)}\boldsymbol{p}_i$ is the position of O_i relative to $O_{p(i)}$ expressed in $p(i)$ coordinates; and m_i, $\boldsymbol{c}_i$, and $\bar{\boldsymbol{I}}_i^{\mathrm{cm}}$ are the inertia parameters of link i. (See Table 3.1 for the equations relating these 3-D quantities to the corresponding spatial quantities.)

At first sight, the 3-D vector algorithm looks significantly different from the spatial vector algorithm. Nevertheless, it can be obtained directly from the spatial vector algorithm simply by expanding the spatial vectors to their 3-D components, restricting the joint type to revolute, converting spatial accelerations to classical accelerations (i. e., replacing each instance of $\dot{\boldsymbol{v}}_i$ with $\dot{\boldsymbol{v}}'_i - \boldsymbol{\omega}_i \times \boldsymbol{v}_i$ as per (3.22)), and applying some 3-D vector identities to bring the equations into the form shown in the table. The conversion from spatial to classical acceleration has one interesting side-effect: $\boldsymbol{v}_i$ cancels out of the equation of motion, and therefore does not need to be calculated. As a result, the 3-D version of the algorithm has a slight speed advantage over the spatial version.

3.5.2 The Articulated-Body Algorithm

The ABA is an $O(N_B)$ algorithm for calculating the forward dynamics of a kinematic tree. However, under normal circumstances, $O(N_B) = O(n)$, so we shall refer to it as an $O(n)$ algorithm. The ABA was developed by *Featherstone* [3.1] and is an example of a constraint-propagation algorithm. Given the joint position, velocity, and applied torque/force variables, this algorithm calculates the joint accelerations. With the joint accelerations determined, numerical integration may be used to provide a simulation of the mechanism.

The key concept in the ABA is illustrated in Fig. 3.7. The subtree rooted at link i interacts with the rest of the kinematic tree only through a force $\mathbf{f}_i$ that is transmitted across joint i. Suppose we break the tree at this point, and consider only the motion of the subtree subject to an unknown force, $\mathbf{f}_i$, acting on link i. It is possible to show that the acceleration of link i is related to the applied force according to the equation

$$\mathbf{f}_i = \boldsymbol{I}_i^A\mathbf{a}_i + \mathbf{p}_i^A\ , \tag{3.79}$$

where $\boldsymbol{I}_i^A$ is called the articulated-body inertia of link i in the subtree (which we can now call an articulated body), and $\mathbf{p}_i^A$ is the associated bias force, which is the force required to produce zero acceleration in link i. Note that $\mathbf{p}_i^A$ depends on the velocities of the individual bodies in the articulated body. Equation (3.79) takes into account the complete dynamics of the subtree. Thus, if we happened to know the correct value of $\mathbf{f}_i$, then (3.79)

would immediately give us the correct acceleration of link i.

The reason we are interested in the quantities $\boldsymbol{I}_i^A$ and $\mathbf{p}_i^A$ is that they allow us to calculate $\ddot{\boldsymbol{q}}_i$ from $\mathbf{a}_{p(i)}$, which in turn allows us to calculate $\mathbf{a}_i$, which then allows us to calculate more joint accelerations, and so on. Combining (3.79) with (3.75) and (3.72) gives

$$\boldsymbol{\tau}_i = \boldsymbol{\Phi}_i^{\mathrm{T}} \mathbf{f}_i = \boldsymbol{\Phi}_i^{\mathrm{T}} \left(\boldsymbol{I}_i^A (\mathbf{a}_{p(i)} + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i) + \mathbf{p}_i^A \right) ,$$

which can be solved for $\ddot{\boldsymbol{q}}_i$ to give

$$\ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i \left(\boldsymbol{u}_i - \boldsymbol{U}_i^{\mathrm{T}} \mathbf{a}_{p(i)} \right) , \tag{3.80}$$

where

$$\begin{aligned} \boldsymbol{U}_i &= \boldsymbol{I}_i^A \boldsymbol{\Phi}_i , \\ \boldsymbol{D}_i &= \left(\boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{U}_i \right)^{-1} = \left(\boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{I}_i^A \boldsymbol{\Phi}_i \right)^{-1} , \\ \boldsymbol{u}_i &= \boldsymbol{\tau}_i - \boldsymbol{U}_i^{\mathrm{T}} \boldsymbol{\zeta}_i - \boldsymbol{\Phi}_i^{\mathrm{T}} \mathbf{p}_i^A \end{aligned}$$

and

$$\boldsymbol{\zeta}_i = \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i = \mathring{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i + \mathbf{v}_i \times \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i .$$

$\mathbf{a}_i$ can then be calculated via (3.72).

It turns out that the articulated-body inertias and bias forces can be calculated efficiently via the recursive formulae

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \sum_{j \in c(i)} \left(\boldsymbol{I}_j^A - \boldsymbol{U}_j \boldsymbol{D}_j \boldsymbol{U}_j^{\mathrm{T}} \right) \tag{3.81}$$

and

$$\mathbf{p}_i^A = \mathbf{p}_i + \sum_{j \in c(i)} \left(\mathbf{p}_j^A + \boldsymbol{I}_j^A \boldsymbol{\zeta}_j + \boldsymbol{U}_j \boldsymbol{D}_j \boldsymbol{u}_j \right) , \tag{3.82}$$

where

$$\mathbf{p}_i = \mathbf{v}_i \times \boldsymbol{I}_i \mathbf{v}_i - \mathbf{f}_i^{\mathrm{e}} .$$

These formulae are obtained by examining the relationship between $\mathbf{f}_i$ and $\mathbf{a}_i$ in Fig. 3.7 and assuming that $\boldsymbol{I}_j^A$ and $\mathbf{p}_j^A$ are already known for every $j \in c(i)$. For more details see [3.1, 8, 15, 25].

The complete algorithm is shown in Algorithm 3.4.

Algorithm 3.4 Articulated-body algorithm for forward dynamics

inputs: $\mathbf{q}$, $\dot{\mathbf{q}}$, $\boldsymbol{\tau}$, *model*, ${}^0\mathbf{f}_i^{\mathrm{e}}$
output: $\ddot{\mathbf{q}}$
model data: N_B, jtype(i), $p(i)$, $\boldsymbol{X}_L(i)$, $\boldsymbol{I}_i$

$\mathbf{v}_0 = \mathbf{0}$
$\mathbf{a}_0 = -\mathbf{a}_{\mathrm{g}}$
for $i = 1$ **to** N_B **do**
  $\boldsymbol{X}_J(i) = \mathrm{xjcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i)$
  ${}^i\boldsymbol{X}_{p(i)} = \boldsymbol{X}_J(i)\, \boldsymbol{X}_L(i)$
  if $p(i) \neq 0$ **then**
    ${}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{p(i)}\, {}^{p(i)}\boldsymbol{X}_0$
  end if
  $\boldsymbol{\Phi}_i = \mathrm{pcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i)$
  $\mathring{\boldsymbol{\Phi}}_i = \mathrm{pdcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
  $\mathbf{v}_i = {}^i\boldsymbol{X}_{p(i)}\, \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i$
  $\boldsymbol{\zeta}_i = \mathring{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i + \mathbf{v}_i \times \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_i$
  $\boldsymbol{I}_i^A = \boldsymbol{I}_i$
  $\mathbf{p}_i^A = \mathbf{v}_i \times \boldsymbol{I}_i \mathbf{v}_i - {}^i\boldsymbol{X}_0^{-T}\, {}^0\mathbf{f}_i^{\mathrm{e}}$
end for
for $i = N_B$ **to** 1 **do**
  $\boldsymbol{U}_i = \boldsymbol{I}_i^A \boldsymbol{\Phi}_i$
  $\boldsymbol{D}_i = \left(\boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{U}_i \right)^{-1}$
  $\boldsymbol{u}_i = \boldsymbol{\tau}_i - \boldsymbol{U}_i^{\mathrm{T}} \boldsymbol{\zeta}_i - \boldsymbol{\Phi}_i^{\mathrm{T}} \mathbf{p}_i^A$
  if $p(i) \neq 0$ **then**
    $\boldsymbol{I}_{p(i)}^A = \boldsymbol{I}_{p(i)}^A + {}^i\boldsymbol{X}_{p(i)}^{\mathrm{T}} \left(\boldsymbol{I}_i^A - \boldsymbol{U}_i \boldsymbol{D}_i \boldsymbol{U}_i^{\mathrm{T}} \right) {}^i\boldsymbol{X}_{p(i)}$
    $\mathbf{p}_{p(i)}^A = \mathbf{p}_{p(i)}^A + {}^i\boldsymbol{X}_{p(i)}^{\mathrm{T}} \left(\mathbf{p}_i^A + \boldsymbol{I}_i^A \boldsymbol{\zeta}_i + \boldsymbol{U}_i \boldsymbol{D}_i \boldsymbol{u}_i \right)$
  end if
end for
for $i = 1$ **to** N_B **do**
  $\mathbf{a}_i = {}^i\boldsymbol{X}_{p(i)}\, \mathbf{a}_{p(i)}$
  $\ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i \left(\boldsymbol{u}_i - \boldsymbol{U}_i^{\mathrm{T}} \mathbf{a}_i \right)$
  $\mathbf{a}_i = \mathbf{a}_i + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_i + \boldsymbol{\zeta}_i$
end for

It is expressed in link coordinates, as per the RNEA in Algorithm 3.2. It makes a total of three passes through the kinematic tree. The first pass iterates from the base out to the tips; it calculates the link velocities using (3.76), the velocity-product term $\boldsymbol{\zeta}_i = \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_i$, and it initializes the variables $\boldsymbol{I}_i^A$ and $\mathbf{p}_i^A$ to the values $\boldsymbol{I}_i$ and $\mathbf{p}_i (= \mathbf{v}_i \times \boldsymbol{I}_i \mathbf{v}_i - {}^i\boldsymbol{X}_0^{F}\, {}^0\mathbf{f}_i^{\mathrm{e}})$, respectively. The second pass iterates from the tips back to the base; it calculates the articulated-body inertia and bias force for each link using (3.81) and (3.82). The third pass iterates from the base to the tips; it calculates the link and joint accelerations using (3.80) and (3.77).

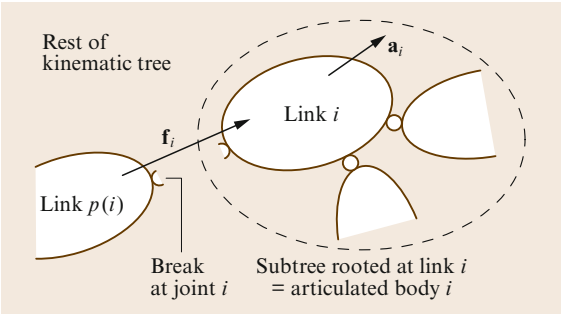

Fig. 3.7 Definition of articulated body i

3.5.3 The Composite-Rigid-Body Algorithm

The CRBA is an algorithm for calculating the joint-space inertia matrix (JSIM) of a kinematic tree. The most common use for the CRBA is as part of a forward dynamics algorithm. It first appeared as method 3 in [3.5].

Forward dynamics, in joint space, is the task of calculating $\ddot{\mathbf{q}}$ from $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\boldsymbol{\tau}$. Starting from (3.37), the most obvious way to proceed is to calculate $\boldsymbol{H}$ and $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$, and then solve the linear equation

$$\boldsymbol{H}\ddot{\mathbf{q}} = \boldsymbol{\tau} - (\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}) \tag{3.83}$$

for $\ddot{\mathbf{q}}$. If the mechanism is a kinematic tree, then $\boldsymbol{H}$ and $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$ can be computed in $O(n^2)$ and $O(n)$ operations, respectively, and (3.83) can be solved in $O(n^3)$ operations. Algorithms that take this approach are therefore known collectively as $O(n^3)$ algorithms. However, this figure of $O(n^3)$ should be regarded as the worst-case complexity, since the actual complexity depends on the amount of branching in the tree [3.26]. Furthermore, even in the worst case, the n^3 term has a small coefficient, and does not dominate until approximately $n = 60$.

$\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$ can be calculated using an inverse dynamics algorithm. If $ID(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is the result of an inverse dynamics calculation with arguments $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$, then

$$ID(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \boldsymbol{\tau} = \boldsymbol{H}\ddot{\mathbf{q}} + \boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}} ,$$

so

$$\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}} = ID(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{0}) . \tag{3.84}$$

Thus, the value of $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$ for a kinematic tree can be calculated efficiently using the RNEA with $\ddot{\mathbf{q}} = \mathbf{0}$.

The key concept in the CRBA is to note that the JSIM only depends on the joint positions, and not their rates. The CRBA makes the simplifying assumption that the rate at each joint is zero. By also assuming that gravity is zero, $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$ is eliminated from (3.83). Furthermore, for a revolute joint, a unit of joint acceleration applied at the j-th joint produces the j-th column of the JSIM. This partitions the mechanism into two composite rigid bodies connected by the j-th joint, and simplifies the dynamics considerably. This concept has been generalized so that the CRBA may be applied to any joint type within a kinematic tree structure.

It can be shown that the general form of the JSIM for a kinematic tree is

$$\boldsymbol{H}_{ij} = \begin{cases} \boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{I}_i^{\mathrm{C}} \boldsymbol{\Phi}_j & \text{if } i \in c^*(j) \\ \boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{I}_j^{\mathrm{C}} \boldsymbol{\Phi}_j & \text{if } j \in c^*(i) \\ \mathbf{0} & \text{otherwise,} \end{cases} \tag{3.85}$$

where $c^*(i)$ is the set of links in the subtree rooted at link i, including link i itself, and

$$\boldsymbol{I}_i^{\mathrm{C}} = \sum_{j \in c^*(i)} \boldsymbol{I}_j . \tag{3.86}$$

See [3.8, 15]. In fact, $\boldsymbol{I}_i^{\mathrm{C}}$ is the inertia of the composite rigid body formed by the rigid assembly of all the links in $c^*(i)$, and this is where the algorithm gets its name.

Equations (3.85) and (3.86) are the basis of the algorithm shown in Algorithm 3.5, which is the CRBA in link coordinates.

Algorithm 3.5 Composite-rigid-body algorithm for calculating the JSIM

inputs: *model* , *RNEA partial results*
output: $\boldsymbol{H}$
model data : $N_B, p(i), \boldsymbol{I}_i$
RNEA data : $\boldsymbol{\Phi}_i, {}^i\boldsymbol{X}_{p(i)}$

$\boldsymbol{H} = \mathbf{0}$
for $i = 1$ **to** N_B **do**
  $\boldsymbol{I}_i^{\mathrm{C}} = \boldsymbol{I}_i$
end for
for $i = N_B$ **to** 1 **do**
  $\boldsymbol{F} = \boldsymbol{I}_i^{\mathrm{C}} \boldsymbol{\Phi}_i$
  $\boldsymbol{H}_{ii} = \boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{F}$
  if $p(i) \neq 0$ **then**
    $\boldsymbol{I}_{p(i)}^{\mathrm{C}} = \boldsymbol{I}_{p(i)}^{\mathrm{C}} + {}^i\boldsymbol{X}_{p(i)}^{\mathrm{T}} \boldsymbol{I}_i^{\mathrm{C}} {}^i\boldsymbol{X}_{p(i)}$
  end if
  $j = i$
  while $p(j) \neq 0$ **do**
    $\boldsymbol{F} = {}^j\boldsymbol{X}_{p(j)}^{\mathrm{T}} \boldsymbol{F}$
    $j = p(j)$
    $\boldsymbol{H}_{ij} = \boldsymbol{F}^{\mathrm{T}} \boldsymbol{\Phi}_j$
    $\boldsymbol{H}_{ji} = \boldsymbol{H}_{ij}^{\mathrm{T}}$
  end while
end for

This algorithm assumes that the matrices ${}^i\boldsymbol{X}_{p(i)}$ and $\boldsymbol{\Phi}_i$ have already been calculated, e.g., during the calculation of $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$. If this is not the case, then the relevant lines from Algorithm 3.2 can be inserted into the first loop. The matrix $\boldsymbol{F}$ is a local variable. The first step, $\boldsymbol{H} = \mathbf{0}$, can be omitted if there are no branches in the tree.

Having calculated $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$ and $\boldsymbol{H}$, the final step is to solve (3.83) for $\ddot{\mathbf{q}}$. This can be done using a standard Cholesky or $\boldsymbol{LDL}^{\mathrm{T}}$ factorization. Note that $\boldsymbol{H}$ can be highly ill-conditioned [3.27], reflecting an underlying ill-conditioning of the kinematic tree itself, so it is recommended to use double-precision arithmetic for every step in the forward dynamics calculation. (This advice applies also to the ABA.)

Exploiting Sparsity

Equation (3.85) implies that some elements of $\boldsymbol{H}$ will automatically be zero if there are branches in the kinematic tree. An example of this effect is shown in Fig. 3.8. Observe that nearly half of the elements are zero. It is possible to exploit this sparsity using the factorization algorithms described in [3.26]. Depending on the amount of branching in the tree, the sparse algorithms can run many times faster than the standard algorithms.

3.5.4 Operational-Space Inertia Matrix

Two different algorithms are presented to calculate the OSIM. The first is an $O(n^3)$ algorithm that uses the basic definition of the OSIM along with efficient factorization of the JSIM. The second is an $O(n)$ algorithm that is based on efficient solution of the forward dynamics problem.

Algorithm Using Basic Definition

If a robot has a relatively small number of freedoms (e.g., six), then the most efficient method for calculating the OSIM is via (3.52). The procedure is as follows:

1. Calculate $\boldsymbol{H}$ via the CRBA.
2. Factorize $\boldsymbol{H}$ into $\boldsymbol{H} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$ (Cholesky factorization).
3. Use back-substitution to calculate $\boldsymbol{Y} = \boldsymbol{L}^{-1}\boldsymbol{J}^{\mathrm{T}}$.
4. $\boldsymbol{\Lambda}^{-1} = \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$.
5. Factorize $\boldsymbol{\Lambda}^{-1}$ (optional).

The final step is only possible if the end-effector has a full six DOFs, and is only necessary if the application requires $\boldsymbol{\Lambda}$ rather than $\boldsymbol{\Lambda}^{-1}$. In the second step, an $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}}$ factorization can be used instead of Cholesky, or one can use one of the efficient factorizations described in [3.26] for branched kinematic trees.

The other terms in (3.38) can be calculated via (3.53) and (3.54). In particular, (3.38) can be rewritten in the form

$$\dot{\mathbf{v}} + \boldsymbol{\Lambda}^{-1}(\boldsymbol{x})[\boldsymbol{\mu}(\boldsymbol{x}, \mathbf{v}) + \boldsymbol{\rho}(\boldsymbol{x})] = \boldsymbol{\Lambda}^{-1}(\boldsymbol{x})\,\mathbf{f}\,, \qquad (3.87)$$

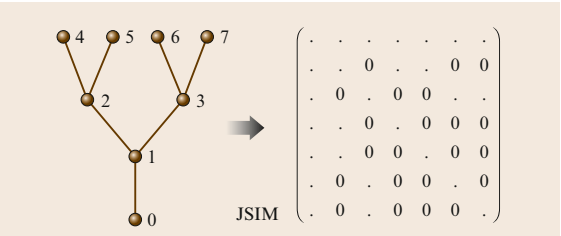

Fig. 3.8 Branch-induced sparsity: branches in the kinematic tree cause certain elements in the JSIM to be zero

and the quantity $\boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu} + \boldsymbol{\rho})$ can be calculated from the formula

$$\boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu} + \boldsymbol{\rho}) = \boldsymbol{J}\boldsymbol{H}^{-1}(\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}) - \dot{\boldsymbol{J}}\dot{\mathbf{q}}\,. \qquad (3.88)$$

The term $\dot{\boldsymbol{J}}\dot{\mathbf{q}}$ is the velocity-product acceleration of the end-effector (3.50). It is calculated as a by-product of calculating $\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}$ via the RNEA (3.84). Specifically, $\dot{\boldsymbol{J}}\dot{\mathbf{q}} = \mathbf{a}_{\mathrm{ee}} - \mathbf{a}_0$, where $\mathbf{a}_{\mathrm{ee}}$ is the calculated acceleration of the end-effector (expressed in the same coordinates as $\dot{\mathbf{v}}$) and $\mathbf{a}_0$ is the acceleration of the base ($-\mathbf{a}_{\mathrm{g}}$).

$O(n)$ Algorithm

For a sufficiently large value of n, it becomes more efficient to use an $O(n)$ algorithm. Several such algorithms can be found in [3.28–30]. In this section, a more straightforward algorithm is given, which is based on an $O(n)$ calculation of the joint-space forward dynamics problem, e.g., via the ABA. It is a variation of the unit force method [3.29] and computes the inverse of the OSIM.

Starting with (3.87), observe that $\boldsymbol{\Lambda}^{-1}$ is a function of position only, and certain terms in the dynamic equations can be neglected without affecting its value. Specifically, if the joint rates, $\dot{\mathbf{q}}$, joint forces, $\boldsymbol{\tau}$, and gravitational forces are all set to zero, the value of $\boldsymbol{\Lambda}$ will remain unchanged. Under these conditions,

$$\dot{\mathbf{v}} = \boldsymbol{\Lambda}^{-1}\mathbf{f}\,. \qquad (3.89)$$

Let us define $\hat{\mathbf{e}}_i$ to be a 6-D coordinate vector with a 1 in the i-th coordinate and zeros elsewhere. If we set $\mathbf{f} = \hat{\mathbf{e}}_i$ in (3.89), then $\dot{\mathbf{v}}$ will equal column i of $\boldsymbol{\Lambda}^{-1}$. Let us also define the function $FD(i, j, \mathbf{q}, \dot{\mathbf{q}}, \mathbf{a}_0, \boldsymbol{\tau}, \mathbf{f})$, which performs a forward-dynamics calculation and returns the true acceleration of link i (i. e., $\mathbf{a}_i - \mathbf{a}_0$), expressed in the same coordinates as $\mathbf{f}$ (typically the base coordinates). The arguments $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\boldsymbol{\tau}$ set the values of the joint position, velocity, and force variables, while j and $\mathbf{f}$ specify that an external force of $\mathbf{f}$ is to be applied to link j. The argument $\mathbf{a}_0$ specifies a fictitious base acceleration to include gravitational effects, and is set to either $\mathbf{0}$ or $-\mathbf{a}_{\mathrm{g}}$.

With these definitions, we have

$$(\boldsymbol{\Lambda}^{-1})^i = FD(ee, ee, \mathbf{q}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \hat{\mathbf{e}}_i) \qquad (3.90)$$

and

$$\boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu} + \boldsymbol{\rho}) = -FD(ee, ee, \mathbf{q}, \dot{\mathbf{q}}, -\mathbf{a}_{\mathrm{g}}, \boldsymbol{\tau}, \mathbf{0})\,, \qquad (3.91)$$

where $(\boldsymbol{\Lambda}^{-1})^i$ is column i of $\boldsymbol{\Lambda}^{-1}$, and ee is the body number of the end-effector. It therefore follows that the coefficients of (3.87) can be calculated using the algorithm shown in Algorithm 3.6. This algorithm is $O(n)$.

Algorithm 3.6 Algorithm to compute the inverse of the operational-space inertia matrix and other terms

for $j = 1$ **to** 6 **do**
  $\dot{\mathbf{v}}^j = FD(ee, ee, \mathbf{q}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \hat{\mathbf{e}}_j)$
end for
$\boldsymbol{\Lambda}^{-1} = \left[\dot{\mathbf{v}}^1\ \dot{\mathbf{v}}^2\ \cdots\ \dot{\mathbf{v}}^6\right]$
$\boldsymbol{\Lambda}^{-1}(\boldsymbol{\mu} + \boldsymbol{\rho}) = -FD(ee, ee, \mathbf{q}, \dot{\mathbf{q}}, -\mathbf{a}_g, \boldsymbol{\tau}, \mathbf{0})$

The efficiency of the algorithm may be increased significantly when computing $\boldsymbol{\Lambda}^{-1}$ by noting that: (1) $\mathbf{v}_i$, $\boldsymbol{\zeta}_i$, and $\boldsymbol{\tau}_i$ in the ABA calculation (Algorithm 3.4) may be set to zero, and (2) $\boldsymbol{I}_i^A$ and the quantities that depend upon it ($\boldsymbol{U}_i$ and $\boldsymbol{D}_i$) need only be computed once, since they do not vary with the applied force. Also, note that the algorithm may be applied to multiple end-effectors by modifying *FD* to accept a list of end-effector body numbers in its first argument, and return a composite vector containing the accelerations of all the specified bodies. Algorithm 3.6 is then enclosed in a `for` loop that controls the second argument to *FD*, and iterates over all of the end-effector body numbers [3.20].

However, for the case of branched mechanisms with multiple end-effectors, several published algorithms achieve even better efficiency, and should be used instead [3.20, 29, 31–33]. The best achievable complexity is $O(n + md + m^2)$, where m is the number of end-effectors and d is the depth of the system's connectivity tree [3.33]; but the fastest algorithm for a typical humanoid exploits branch-induced sparsity [3.32, 33].

3.6 Kinematic Loops

All of the algorithms in the last section were for kinematic trees. In this section, a final algorithm is provided for the forward dynamics of closed-loop systems. The algorithm supplements the dynamic equations of motion for a spanning tree of the closed-loop system with the loop-closure constraint equations. Three different methods are given to solve the resulting linear system of equations. An efficient algorithm is given to compute the loop-closure constraints.

Systems with closed kinematic loops exhibit more complicated dynamics than kinematic trees. For example:

1. The degree of motion freedom of a kinematic tree is fixed, but that of a closed-loop system can vary.
2. The degree of instantaneous motion freedom is always the same as the degree of finite motion freedom in a kinematic tree, but they can be different in a closed-loop system.
3. Every force in a kinematic tree can be determined, but some forces in a closed-loop system can be indeterminate. This occurs whenever a closed-loop system is overconstrained.

Two examples of these phenomena are shown in Fig. 3.9. The mechanism in Fig. 3.9a has no finite motion freedom, but it has two degrees of infinitesimal motion freedom. The mechanism in Fig. 3.9b has one degree of freedom when $\theta \neq 0$, but if $\theta = 0$ then the two arms, A and B, are able to move independently, and the mechanism has two degrees of freedom. Moreover, at the boundary between these two motion regimes, the mechanism has three degrees of infinitesimal motion freedom. Both these mechanisms are planar, and are therefore overconstrained. As a result, the out-of-plane components of the joint constraint forces are indeterminate. This kind of indeterminacy has no effect on the motions of these mechanisms, but it does complicate the calculation of their dynamics.

3.6.1 Formulation of Closed-Loop Algorithm

A closed-loop system can be modeled as a spanning tree subject to a set of loop-closure constraint forces. If

$$\boldsymbol{H}\ddot{\mathbf{q}} + \boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_g = \boldsymbol{\tau}$$

is the equation of motion of the spanning tree on its own, then the equation of motion for the closed-loop system is

$$\boldsymbol{H}\ddot{\mathbf{q}} + \boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_g = \boldsymbol{\tau} + \boldsymbol{\tau}^a + \boldsymbol{\tau}^c\,, \tag{3.92}$$

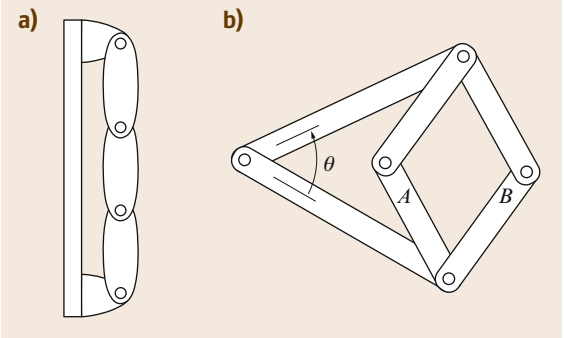

Fig.3.9a,b Pathological closed-loop systems. **(a)** Example of varying motion freedom, **(b)** Example with differing finite and infinitesimal motion freedoms

where $\boldsymbol{\tau}^{\mathrm{a}}$ and $\boldsymbol{\tau}^{\mathrm{c}}$ are vectors of loop-closure active and constraint forces, respectively, expressed in the generalized force coordinates of the spanning tree. $\boldsymbol{\tau}^{\mathrm{a}}$ is a known quantity, and $\boldsymbol{\tau}^{\mathrm{c}}$ is unknown. $\boldsymbol{\tau}^{\mathrm{a}}$ comes from the force elements acting at the loop-closing joints (springs, dampers and actuators). If there are no such force elements, then $\boldsymbol{\tau}^{\mathrm{a}} = \mathbf{0}$.

The loop-closure constraints restrict the motion of the spanning tree. At the acceleration level, these constraints can be expressed in the form of a linear equation,

$$\boldsymbol{L}\ddot{\mathbf{q}} = \mathbf{l}\,, \tag{3.93}$$

where $\boldsymbol{L}$ is an $n^{\mathrm{c}} \times n$ matrix. n^{c} is the number of constraints due to the loop-closing joints, and is given by the formula

$$n^{\mathrm{c}} = \sum_{k=N_B+1}^{N_J} n_k^{\mathrm{c}}\,, \tag{3.94}$$

where n_k^{c} is the number of constraints imposed by joint k. If $\mathrm{rank}(\boldsymbol{L}) < n^{\mathrm{c}}$ then the loop-closure constraints are linearly dependent, and the closed-loop mechanism is overconstrained. The mobility of a closed-loop system (i. e., its degree of motion freedom) is given by the formula

$$\text{mobility} = n - \mathrm{rank}(\boldsymbol{L})\,. \tag{3.95}$$

Given a constraint equation in the form of (3.93), it follows that the constraint forces can be expressed in the form

$$\boldsymbol{\tau}^{\mathrm{c}} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{\lambda}\,, \tag{3.96}$$

where $\boldsymbol{\lambda} = \left(\boldsymbol{\lambda}_{N_B+1}^{\mathrm{T}} \cdots \boldsymbol{\lambda}_{N_J}^{\mathrm{T}}\right)^{\mathrm{T}}$ is an $n^{\mathrm{c}} \times 1$ vector of unknown constraint-force variables (or Lagrange multipliers). If the mechanism is overconstrained, then $\boldsymbol{L}^{\mathrm{T}}$ will have a null space, and the component of $\boldsymbol{\lambda}$ lying in this null space will be indeterminate.

It is often possible to identify redundant constraints in advance. For example, if a kinematic loop is known to be planar, then the out-of-plane loop-closure constraints are redundant. In these circumstances, it is advantageous to remove the corresponding rows of $\boldsymbol{L}$ and elements of $\mathbf{l}$ and $\boldsymbol{\lambda}$. The removed elements of $\boldsymbol{\lambda}$ can be assigned a value of zero.

Combining (3.92), (3.93), and (3.96) produces the following equation of motion for a closed-loop system

$$\begin{pmatrix} \boldsymbol{H} & \boldsymbol{L}^{\mathrm{T}} \\ \boldsymbol{L} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}} \\ -\boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\tau} + \boldsymbol{\tau}^{\mathrm{a}} - (\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}) \\ \mathbf{l} \end{pmatrix}. \tag{3.97}$$

The system matrix is symmetric, but indefinite. If $\boldsymbol{L}$ has full rank, then the system matrix will be nonsingular, otherwise it will be singular, and one or more elements of $\boldsymbol{\lambda}$ will be indeterminate.

Equation (3.97) can be solved in any of the following ways:

1. Solve it directly for $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$.
2. Solve for $\boldsymbol{\lambda}$ first, and then use the result to solve for $\ddot{\mathbf{q}}$.
3. Solve (3.93) for $\ddot{\mathbf{q}}$, substitute the result into (3.92), eliminate the unknown constraint forces, and solve for the remaining unknowns.

Method 1 is the simplest, but generally also the least efficient. This method is appropriate when the system matrix is nonsingular. As the size of the system matrix is $(n + n^{\mathrm{c}}) \times (n + n^{\mathrm{c}})$, this method is $O((n + n^{\mathrm{c}})^3)$.

Method 2 is particularly useful if $n \gg n^{\mathrm{c}}$, and offers the opportunity to use $O(n)$ algorithms on the spanning tree [3.6]. From (3.97),

$$\boldsymbol{L}\boldsymbol{H}^{-1}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{l} - \boldsymbol{L}\boldsymbol{H}^{-1}\left[\boldsymbol{\tau} + \boldsymbol{\tau}^{\mathrm{a}} - (\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}})\right]\,. \tag{3.98}$$

This equation can be formulated in $O(n\,(n^{\mathrm{c}})^2)$ operations via $O(n)$ algorithms, and solved in $O((n^{\mathrm{c}})^3)$. Once $\boldsymbol{\lambda}$ is known, $\boldsymbol{\tau}^{\mathrm{c}}$ can be calculated via (3.96) in $O(n\,n^{\mathrm{c}})$ operations, and (3.92) solved by an $O(n)$ algorithm; so the total complexity is $O(n\,(n^{\mathrm{c}})^2 + (n^{\mathrm{c}})^3)$. If $\boldsymbol{L}$ is rank deficient, then $\boldsymbol{L}\boldsymbol{H}^{-1}\boldsymbol{L}^{\mathrm{T}}$ will be singular; but it is still a positive-semidefinite matrix, and presents a slightly easier factorization problem than a singular instance of the indefinite system matrix in (3.97).

Method 3 is useful if $n - n^{\mathrm{c}}$ is small, or if $\boldsymbol{L}$ is expected to be rank deficient. Equation (3.93) is solved using a special version of Gaussian elimination (or similar procedure), which is equipped with a numerical rank test, and which is designed to solve underdetermined systems. The solution is an equation of the form

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

where $\ddot{\mathbf{q}}_0$ is any particular solution to (3.93), $\boldsymbol{K}$ is an $n \times (n - \mathrm{rank}(\boldsymbol{L}))$ matrix with the property $\boldsymbol{L}\boldsymbol{K} = \mathbf{0}$, and $\boldsymbol{y}$ is a vector of $n - \mathrm{rank}(\boldsymbol{L})$ unknowns. (Typically, $\boldsymbol{y}$ is a linearly independent subset of the elements of $\ddot{\mathbf{q}}$.) Substituting this expression for $\ddot{\mathbf{q}}$ into (3.92), and premultiplying both sides by $\boldsymbol{K}^{\mathrm{T}}$ to eliminate $\boldsymbol{\tau}^{\mathrm{c}}$, produces

$$\boldsymbol{K}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{K}\boldsymbol{y} = \boldsymbol{K}^{\mathrm{T}}\left(\boldsymbol{\tau} + \boldsymbol{\tau}^{\mathrm{a}} - (\boldsymbol{C}\dot{\mathbf{q}} + \boldsymbol{\tau}_{\mathrm{g}}) - \boldsymbol{H}\ddot{\mathbf{q}}_0\right)\,. \tag{3.99}$$

This method also has cubic complexity, but it can be the most efficient if $n - n^{\mathrm{c}}$ is small. It is also reported to be more stable than method 1 [3.34].

3.6.2 Closed-Loop Algorithm

Algorithms for calculating $\boldsymbol{H}$ and $\boldsymbol{C}\dot{\mathbf{q}}+\boldsymbol{\tau}_{\mathrm{g}}$ can be found in Sect. 3.5.3 and 3.5.1, respectively, which just leaves $\boldsymbol{L}$, $\mathbf{l}$ and $\boldsymbol{\tau}^{\mathrm{a}}$. To keep things simple, we will assume that all loop-closing joints are zero-DOF joints.

There is no loss of generality with this assumption: one simply breaks open the loops by cutting links instead of joints (Fig. 3.10). However, there may be some loss of efficiency. With this assumption, we only need to calculate $\boldsymbol{L}$ and $\mathbf{l}$, since $\boldsymbol{\tau}^{\mathrm{a}}=\mathbf{0}$.

Loop Constraints

In the general case, the velocity constraint equation for loop k is

$$\left(\boldsymbol{\Psi}_k^{\mathrm{c}}\right)^{\mathrm{T}}\left(\mathbf{v}_{s(k)}-\mathbf{v}_{p(k)}\right)=\mathbf{0}\,, \tag{3.100}$$

and the acceleration constraint is

$$\left(\boldsymbol{\Psi}_k^{\mathrm{c}}\right)^{\mathrm{T}}\left(\mathbf{a}_{s(k)}-\mathbf{a}_{p(k)}\right)+\left(\dot{\boldsymbol{\Psi}}_k^{\mathrm{c}}\right)^{\mathrm{T}}\left(\mathbf{v}_{s(k)}-\mathbf{v}_{p(k)}\right)=\mathbf{0}\,. \tag{3.101}$$

However, if every loop-closing joint has zero DOF, then these equations simplify to

$$\mathbf{v}_{s(k)}-\mathbf{v}_{p(k)}=\mathbf{0} \tag{3.102}$$

and

$$\mathbf{a}_{s(k)}-\mathbf{a}_{p(k)}=\mathbf{0}\,. \tag{3.103}$$

Let us define a loop Jacobian, $\boldsymbol{J}_k$, with the property that

$$\mathbf{v}_{s(k)}-\mathbf{v}_{p(k)}=\boldsymbol{J}_k\dot{\mathbf{q}}\,. \tag{3.104}$$

$\boldsymbol{J}_k$ is a $6\times n$ matrix defined by the formula

$$\boldsymbol{J}_k=\left(e_{1k}\boldsymbol{\Phi}_1\;\cdots\;e_{N_Bk}\boldsymbol{\Phi}_{N_B}\right)\,, \tag{3.105}$$

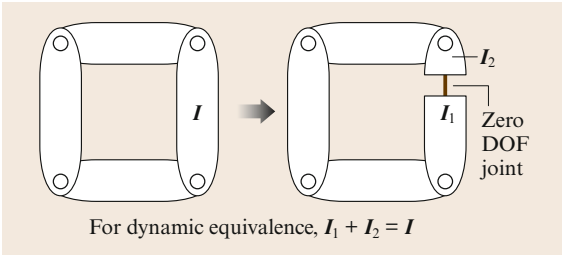

Fig. 3.10 Inserting a zero-DOF joint in preparation for cutting the loop open at that joint

where

$$e_{ik}=\begin{cases}+1 & \text{if } s(k)\in c^*(i) \text{ and } p(k)\notin c^*(i)\,,\\ -1 & \text{if } p(k)\in c^*(i) \text{ and } s(k)\notin c^*(i)\,,\\ 0 & \text{otherwise.}\end{cases}$$

In other words, $e_{ik}=+1$ if joint i lies on the path to $s(k)$ but not the path to $p(k)$; $e_{ik}=-1$ if joint i lies on the path to $p(k)$ but not the path to $s(k)$; and $e_{ik}=0$ if joint i lies on both paths or on neither.

The loop acceleration constraint can now be written

$$\begin{aligned}\mathbf{0}&=\mathbf{a}_{s(k)}-\mathbf{a}_{p(k)}\\&=\boldsymbol{J}_k\ddot{\mathbf{q}}+\dot{\boldsymbol{J}}_k\dot{\mathbf{q}}\\&=\boldsymbol{J}_k\ddot{\mathbf{q}}+\mathbf{a}^{\mathrm{vp}}_{s(k)}-\mathbf{a}^{\mathrm{vp}}_{p(k)}\,,\end{aligned} \tag{3.106}$$

where $\mathbf{a}_i^{\mathrm{vp}}$ is the velocity-product acceleration of link i, which is the acceleration it would have if $\ddot{\mathbf{q}}$ were zero. The velocity-product acceleration of every link is calculated during the calculation of the vector $\boldsymbol{C}\dot{\mathbf{q}}+\boldsymbol{\tau}_{\mathrm{g}}$ (3.84). If the RNEA is used to calculate $\boldsymbol{C}\dot{\mathbf{q}}+\boldsymbol{\tau}_{\mathrm{g}}$, then $\mathbf{a}_i^{\mathrm{vp}}$ will be the value of $\mathbf{a}_i$ calculated by the RNEA with its acceleration argument set to zero.

The matrices $\boldsymbol{L}$ and $\mathbf{l}$ can now be expressed as follows

$$\boldsymbol{L}=\begin{pmatrix}\boldsymbol{L}_{N_B+1}\\ \vdots\\ \boldsymbol{L}_{N_J}\end{pmatrix}\quad\text{and}\quad\mathbf{l}=\begin{pmatrix}\boldsymbol{l}_{N_B+1}\\ \vdots\\ \boldsymbol{l}_{N_J}\end{pmatrix}, \tag{3.107}$$

where

$$\boldsymbol{L}_k=\boldsymbol{J}_k \tag{3.108}$$

and

$$\boldsymbol{l}_k=\mathbf{a}^{\mathrm{vp}}_{p(k)}-\mathbf{a}^{\mathrm{vp}}_{s(k)}\,. \tag{3.109}$$

Constraint Stabilization

In practice, it is necessary to stabilize loop-closure constraints, or they will simply fly apart during simulation because of numerical integration errors. The standard technique is due to *Baumgarte* [3.3, 7, 35], and consists of replacing each constraint equation of the form

$$a_{\mathrm{e}}=0\,,$$

with one of the form

$$a_{\mathrm{e}}+K_{\mathrm{v}}v_{\mathrm{e}}+K_{\mathrm{p}}p_{\mathrm{e}}=0\,,$$

where a_{e}, v_{e}, and p_{e} are the acceleration, velocity, and position errors, respectively, and K_{v} and K_{p} are positive constants. Typically, one chooses a time constant,

t_c, according to how quickly one wants the position and velocity errors to decay. K_p and K_v are then given by the formulae $K_v = 2/t_c$ and $K_p = 1/t_c^2$. However, there is no good rule for choosing t_c. If t_c is too long, then loop-constraint errors accumulate faster than they decay; if t_c is too short, then the equations of motion become excessively stiff, causing a loss of numerical integration accuracy. A reasonable value for a large, slow industrial robot is $t_c = 0.1$, while a smaller, faster robot might need $t_c = 0.01$.

To incorporate stabilization terms into the loop constraint equation, we replace (3.109) with

$$\boldsymbol{l}_k = \mathbf{a}^{\mathrm{vp}}_{p(k)} - \mathbf{a}^{\mathrm{vp}}_{s(k)} - K_v(\mathbf{v}_{s(k)} - \mathbf{v}_{p(k)}) - K_p \mathbf{p}_{ek} \ , \quad (3.110)$$

where $\mathbf{p}_{ek}$ is a vector representing the position error in loop k. In absolute coordinates (i. e., link 0 coordinates), $\mathbf{p}_{ek}$ is given by

$$\mathbf{p}_{ek} = \mathrm{x\_to\_vec}({}^0\boldsymbol{X}_{p(k)}\, \boldsymbol{X}_{L1}^{-1}(k)\, \boldsymbol{X}_{L2}(k)\, {}^{s(k)}\boldsymbol{X}_0) \ , \quad (3.111)$$

where the $\boldsymbol{X}_{L1}(k)$ and $\boldsymbol{X}_{L2}(k)$ transforms are defined in (3.62) and (3.63), and shown in Fig. 3.4 for joint k, and $\mathrm{x\_to\_vec}\left({}^B\boldsymbol{X}_A\right)$ computes a vector approximating the displacement from frame A to frame B, assuming this displacement to be infinitesimal. x_to_vec can be defined as

$$\mathrm{x\_to\_vec}(\boldsymbol{X}) = \frac{1}{2}\begin{pmatrix} X_{23} - X_{32} \\ X_{31} - X_{13} \\ X_{12} - X_{21} \\ X_{53} - X_{62} \\ X_{61} - X_{43} \\ X_{42} - X_{51} \end{pmatrix} . \quad (3.112)$$

Algorithm

Algorithm 3.7 shows an algorithm for calculating $\boldsymbol{L}$ and $\mathbf{l}$ for the special case when all the loop-closing joints have 0-DOF. It combines simplicity with good performance by transforming every quantity that is needed to formulate the loop-closure constraints into a single coordinate system, in this case absolute (link 0) coordinates, so that no further transforms are needed.

The first loop calculates the transforms from absolute to link coordinates, and uses them to transform $\boldsymbol{\Phi}_i$ to absolute coordinates. Only the $\boldsymbol{\Phi}_i$ that are needed in the loop-closure constraints are transformed.

The second loop calculates the nonzero elements of $\boldsymbol{L}$ (which can be sparse), according to (3.105). The inner `while` loop terminates on the root of the loop, which is the highest-numbered common ancestor of links $p(k)$ and $s(k)$. It could be the fixed base if they have no other common ancestor. The second loop ends with the calculation of $\mathbf{l}$, in absolute coordinates, according to (3.110).

Algorithm 3.7 Algorithm to calculate loop-closure constraints

inputs: *model* , *RNEA partial results*
outputs: $\boldsymbol{L}$, $\mathbf{l}$
model data : N_B, $p(i)$, N_J, $p(k)$, $s(k)$, $LB(i)$, $\boldsymbol{X}_{L1}(k)$, $\boldsymbol{X}_{L2}(k)$, K_p, K_v
RNEA data : $\boldsymbol{\Phi}_i$, ${}^i\boldsymbol{X}_{p(i)}$, $\mathbf{v}_{p(k)}$, $\mathbf{v}_{s(k)}$, $\mathbf{a}^{\mathrm{vp}}_{p(k)}$, $\mathbf{a}^{\mathrm{vp}}_{s(k)}$

for $i = 1$ **to** N_B **do**
  if $p(i) \neq 0$ **then**
    ${}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{p(i)}\, {}^{p(i)}\boldsymbol{X}_0$
  end if
  if $LB(i) \neq$ null **then**
    ${}^0\boldsymbol{\Phi}_i = {}^i\boldsymbol{X}_0^{-1}\, \boldsymbol{\Phi}_i$
  end if
end for
$\boldsymbol{L} = \mathbf{0}$
for $k = N_B + 1$ **to** N_J **do**
  $i = p(k)$
  $j = s(k)$
  while $i \neq j$ **do**
    if $i > j$ **then**
      $\boldsymbol{L}_{k,i} = -{}^0\boldsymbol{\Phi}_i$
      $i = p(i)$
    else
      $\boldsymbol{L}_{k,j} = {}^0\boldsymbol{\Phi}_j$
      $j = p(j)$
    end if
  end while
  $\mathbf{a}_e = {}^{s(k)}\boldsymbol{X}_0^{-1}\, \mathbf{a}^{\mathrm{vp}}_{s(k)} - {}^{p(k)}\boldsymbol{X}_0^{-1}\, \mathbf{a}^{\mathrm{vp}}_{p(k)}$
  $\mathbf{v}_e = {}^{s(k)}\boldsymbol{X}_0^{-1}\, \mathbf{v}_{s(k)} - {}^{p(k)}\boldsymbol{X}_0^{-1}\, \mathbf{v}_{p(k)}$
  $\mathbf{p}_e = \mathrm{x\_to\_vec}\left({}^{p(k)}\boldsymbol{X}_0^{-1}\, \boldsymbol{X}_{L1}^{-1}(k)\, \boldsymbol{X}_{L2}(k)\, {}^{s(k)}\boldsymbol{X}_0\right)$
  $\boldsymbol{l}_k = -\mathbf{a}_e - K_v\, \mathbf{v}_e - K_p\, \mathbf{p}_e$
end for

3.7 Conclusions and Further Reading

This chapter has presented the fundamentals of rigid-body dynamics as they apply to robot mechanisms. It has covered the following topics: the spatial vector algebra, which provides a concise notation for describing and implementing dynamics equations and algorithms; the canonical equations of motion that are most fre-

quently used in robotics; how to construct a dynamic model of a robot; and several efficient model-based algorithms for calculating inverse dynamics, forward dynamics, and the joint-space and operational-space inertia matrices.

There are many topics in dynamics that have not been mentioned in this chapter, but can be found in later chapters of this handbook. The dynamics of robots with elastic links and joints is covered in Chap. 11; the problem of identifying the parameters of a dynamic model is covered in Chap. 6; the dynamics of physical contact between a robot and the objects in its environment is described in Chap. 37; and the dynamics of robots with floating bases is described in Chap. 55.

We conclude this chapter by noting that a brief history of robot dynamics can be found in [3.36], and that a more extensive treatment of robot dynamics can be found in books such as [3.8, 10, 15, 29, 37–40]. Finally, some suggestions for further reading are listed below.

3.7.1 Multibody Dynamics

Robot dynamics can be regarded as a subset (or a specific application) of the broader discipline of multibody dynamics. Books on multibody dynamics include [3.3, 14, 35, 41–46]. Of course, multibody dynamics is, in turn, a subset of classical mechanics; and the mathematical foundations of the subject can be found in any good book on classical mechanics, such as [3.13].

3.7.2 Alternative Representations

This chapter has used spatial vectors to express the equations of motion. There are various alternatives to the use of spatial vectors: other kinds of 6-D vector, 3-D vectors, 4×4 matrices, and the spatial operator algebra. All 6-D vector formalisms are similar, but are not exactly the same. The main alternatives to spatial vectors are: screws [3.10–12], motors [3.47], Lie algebras [3.12, 48], and ad hoc notations. (An ad hoc notation is one in which 3-D vectors are grouped into pairs for the purpose of reducing the volume of algebra.) Three-dimensional vectors are the formalism used in most classical mechanics and multibody texts, and are also a precursor to 6-D vector and 4×4 matrix formalisms. 4×4 matrices are popular in robotics because they are very useful for kinematics. However, they are not so useful for dynamics. 4×4 matrix formulations of dynamics can be found in [3.37, 49, 50]. The spatial operator algebra was developed at the Jet Propulsion Laboratory (JPL) by Rodriguez, Jain, and others. It uses $6N$-dimensional vectors and $6N \times 6N$ matrices, the latter regarded as linear operators. Examples of this notation can be found in [3.38, 51–53].

3.7.3 Alternative Formulations

This chapter used a vectorial formulation of the equations of motion that is usually called the Newtonian or Newton–Euler formulation. The main alternative is the Lagrangian formulation, in which the equations of motion are obtained via Lagrange's equation. Examples of the Lagrangian formulation can be found in [3.9, 10, 18, 54, 55]. *Kane*'s method has also been applied in robotics [3.56, 57].

3.7.4 Efficiency

Because of the need for real-time implementation, especially in control, the robotics community has focused on the problem of computational efficiency. For *inverse dynamics*, the $O(n)$ recursive Newton–Euler algorithm (RNEA) of *Luh* et al. [3.4] remains the most important algorithm. Further improvements to the algorithm are given in [3.58, 59]. For *forward dynamics*, the two algorithms presented in this chapter remain the most important for computational considerations: the $O(n)$ articulated-body algorithm (ABA) developed by *Featherstone* [3.1] and the $O(n^3)$ algorithm based on the composite-rigid-body algorithm (CRBA) of *Walker* and *Orin* [3.5]. Improvements were made in the ABA over the years [3.15, 17, 25] so that it was more efficient than the CRBA-based algorithm for decreasingly smaller values of n. However, more recent application of the CRBA to branched kinematic trees [3.26] and robotic systems with motion-controlled appendages [3.60] continue to show the viability of the CRBA approach.

For the *joint-space inertia matrix*, the CRBA [3.5] is the most important algorithm. A number of improvements and modifications have been made over the years to increase its computational efficiency [3.15, 61–63]. For the *operational-space inertia matrix*, efficient $O(n)$ algorithms have been developed [3.28–30] and applied to increasingly complex systems [3.20, 31, 33]. The exploitation of branch-induced sparsity also results in an efficient algorithm [3.32].

3.7.5 Accuracy

Concerns can arise over the numerical accuracy of a dynamics algorithm, the accuracy of a simulation (i. e., numerical integration accuracy), or the accuracy of a dynamic model. The numerical accuracy of dynamics algorithms has received relatively little attention compared with efficiency. The RNEA, CRBA, and ABA have all been tested for accuracy on a large variety of rigid-body systems, but the same cannot be said of most other algorithms. Rigid-body systems are often ill-conditioned, in the sense that a small change

in the applied force (or a model parameter) can produce a large change in the resulting acceleration. This phenomenon was studied by *Featherstone* [3.27], who discovered that the ill-conditioning gets worse with increasing body count, and that it can grow in proportion to $O(n^4)$ in the worst case. Other publications on this topic include [3.8, 34, 64, 65].

3.7.6 Software Packages

A number of software packages have been developed to provide dynamic simulation capabilities for multibody systems, and in particular, robotic systems. Several have been written in MATLAB for ease of integration with other analysis, control, and simulation programs. Many packages are open source, and some are offered at a relatively low cost to the user. They differ in their capabilities in a variety of ways including: speed, topologies and joint models supported, accuracy, underlying dynamic formulation and associated order of complexity, user interface, graphics support, numerical integration routines, integration with other code, application support, and cost. Among those commonly cited are: Adams [3.66], Autolev [3.67], Bullet [3.68], DART [3.69], DynaMechs [3.70], Gazebo [3.71], Open Dynamics Engine [3.72], Robotics Studio [3.73], Robotics Toolbox [3.74], Robotran [3.75, 76], SD/FAST [3.77], Simbody [3.78], SimMechanics [3.79], SYMORO [3.80, 81] and Webots [3.82].

3.7.7 Symbolic Simplification

The technique of symbolic simplification takes a general-purpose dynamics algorithm, and applies it symbolically to a specific dynamic model. The result is a list of assignment statements detailing what the algorithm would have done if it had been executed for real. This list is then inspected and pruned of all unnecessary calculations, and the remainder is output to a text file in the form of computer source code. This code can run as much as ten times faster than the original general-purpose algorithm, but it is specific to one dynamic model. Both Autolev [3.67] and SD/FAST [3.77] use this technique. Other publications on symbolic simplification for dynamics include [3.76, 80, 81, 83–88].

3.7.8 Algorithms for Parallel Computers

In order to speed up the common dynamics computations, a number of algorithms have been developed for parallel and pipelined computers. For *inverse dynamics*, early work focused on speeding up the $O(n)$ RNEA on up to n processors [3.89, 90] while subsequent work resulted in $O(\log_2 n)$ algorithms [3.91, 92]. For the $O(n^2)$ CRBA to compute the *joint-space inertia matrix*, early work resulted in $O(\log_2 n)$ algorithms for n processors to compute the composite-rigid-body inertias and diagonal elements of the matrix [3.93, 94]. Subsequent work resulted in $O(\log_2 n)$ algorithms for $O(n^2)$ processors to compute the entire matrix [3.95, 96]. For *forward dynamics*, speedup was obtained for a multiple manipulator system on a parallel/pipelined supercomputer [3.97]. The first $O(\log_2 n)$ algorithm for n processors was developed for an unbranched serial chain [3.98]. More recent work has focused on $O(\log_2 n)$ algorithms for more complex structures [3.65, 99, 100].

3.7.9 Topologically-Varying Systems

There are many robot mechanisms whose topology varies over time because of a change of contact conditions, especially with the environment. In legged vehicles, use of a compliant ground-contact model to compute the contact forces reduced the closed-loop structure to a tree structure [3.101]. However, for cases in which the contacts are very stiff, numerical integration problems may result. In more recent work [3.40, 102] in which hard contact constraints are assumed, an efficient method was used to reduce the large number of coordinate variables from that which may be necessary in general-purpose motion analysis systems [3.43]. Also, they were able to automatically identify the variables as the structure varied and developed a method for computing the velocity boundary conditions after configuration changes [3.40, 102].

References

3.1 R. Featherstone: The calculation of robot dynamics using articulated-body inertias, Int. J. Robotics Res. **2**(1), 13–30 (1983)

3.2 J.J. Craig: *Introduction to Robotics: Mechanics and Control*, 3rd edn. (Prentice Hall, Upper Saddle River 2005)

3.3 R.E. Roberson, R. Schwertassek: *Dynamics of Multibody Systems* (Springer, Berlin, Heidelberg 1988)

3.4 J.Y.S. Luh, M.W. Walker, R.P.C. Paul: On-line computational scheme for mechanical manipulators, Trans. ASME J. Dyn. Syst. Meas. Control **102**(2), 69–76 (1980)

3.5 M.W. Walker, D.E. Orin: Efficient dynamic computer simulation of robotic mechanisms, Trans. ASME J. Dyn. Syst. Meas. Control **104**, 205–211 (1982)

3.6 D. Baraff: Linear-time dynamics using lagrange multipliers, Proc. 23rd Annu. Conf. Comp. Graph. Interact. Tech., New Orleans (1996) pp. 137–146

3.7 J. Baumgarte: Stabilization of constraints and integrals of motion in dynamical systems, Comput. Methods Appl. Mech. Eng. **1**, 1–16 (1972)

3.8 R. Featherstone: *Rigid Body Dynamics Algorithms* (Springer, New York 2008)

3.9 R.M. Murray, Z. Li, S.S. Sastry: *A Mathematical Introduction to Robotic Manipulation* (CRC, Boca Raton 1994)

3.10 J. Angeles: *Fundamentals of Robotic Mechanical Systems*, 2nd edn. (Springer, New York 2003)

3.11 R.S. Ball: *A Treatise on the Theory of Screws* (Cambridge Univ. Press, London 1900), Republished (1998)

3.12 J.M. Selig: *Geometrical Methods in Robotics* (Springer, New York 1996)

3.13 D.T. Greenwood: *Principles of Dynamics* (Prentice-Hall, Englewood Cliffs 1988)

3.14 F.C. Moon: *Applied Dynamics* (Wiley, New York 1998)

3.15 R. Featherstone: *Robot Dynamics Algorithms* (Kluwer, Boston 1987)

3.16 R. Featherstone: *Spatial v2*, http://royfeatherstone.org/spatial/v2 (2012)

3.17 S. McMillan, D.E. Orin: Efficient computation of articulated-body inertias using successive axial screws, IEEE Trans. Robotics Autom. **11**, 606–611 (1995)

3.18 L. Sciavicco, B. Siciliano: *Modeling and Control of Robot Manipulators*, 2nd edn. (Springer, London 2000)

3.19 J. Slotine, W. Li: On the adaptive control of robot manipulators, Int. J. Robotics Res. **6**(3), 49–59 (1987)

3.20 K.S. Chang, O. Khatib: Operational space dynamics: Efficient algorithms for modeling and control of branching mechanisms, Proc. IEEE Int. Conf. Robotics Autom., San Francisco (2000) pp. 850–856

3.21 O. Khatib: A unified approach to motion and force control of robot manipulators: The operational space formulation, IEEE J. Robotics Autom. **3**(1), 43–53 (1987)

3.22 Y.F. Zheng, H. Hemami: Mathematical modeling of a robot collision with its environment, J. Robotics Syst. **2**(3), 289–307 (1985)

3.23 W. Khalil, E. Dombre: *Modeling, Identification and Control of Robots* (Kogan Page Sci., London 2002)

3.24 J. Denavit, R.S. Hartenberg: A kinematic notation for lower-pair mechanisms based on matrices, J. Appl. Mech. **22**, 215–221 (1955)

3.25 H. Brandl, R. Johanni, M. Otter: A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix, Proc. IFAC/IFIP/IMACS Int. Symp. Theory Robots, Vienna (1986)

3.26 R. Featherstone: Efficient factorization of the joint space inertia matrix for branched kinematic trees, Int. J. Robotics Res. **24**(6), 487–500 (2005)

3.27 R. Featherstone: An empirical study of the joint space inertia matrix, Int. J. Robotics Res. **23**(9), 859–871 (2004)

3.28 K. Kreutz-Delgado, A. Jain, G. Rodriguez: Recursive formulation of operational space control, Proc. IEEE Int. Conf. Robotics Autom., Sacramento (1991) pp. 1750–1753

3.29 K.W. Lilly: *Efficient Dynamic Simulation of Robotic Mechanisms* (Kluwer, Boston 1993)

3.30 K.W. Lilly, D.E. Orin: Efficient O(N) recursive computation of the operational space inertia matrix, IEEE Trans. Syst. Man Cybern. **23**(5), 1384–1391 (1993)

3.31 G. Rodriguez, A. Jain, K. Kreutz-Delgado: Spatial operator algebra for multibody system dynamics, J. Astronaut. Sci. **40**(1), 27–50 (1992)

3.32 R. Featherstone: Exploiting sparsity in operational-space dynamics, Int. J. Robotics Res. **29**(10), 1353–1368 (2010)

3.33 P. Wensing, R. Featherstone, D.E. Orin: A reduced-order recursive algorithm for the computation of the operational-space inertia matrix, Proc. IEEE Int. Conf. Robotics Autom., St. Paul (2012) pp. 4911–4917

3.34 R.E. Ellis, S.L. Ricker: Two numerical issues in simulating constrained robot dynamics, IEEE Trans. Syst. Man Cybern. **24**(1), 19–27 (1994)

3.35 J. Wittenburg: *Dynamics of Systems of Rigid Bodies* (Teubner, Stuttgart 1977)

3.36 R. Featherstone, D.E. Orin: Robot dynamics: Equations and algorithms, Proc. IEEE Int. Conf. Robotics Autom., San Francisco (2000) pp. 826–834

3.37 C.A. Balafoutis, R.V. Patel: *Dynamic Analysis of Robot Manipulators: A Cartesian Tensor Approach* (Kluwer, Boston 1991)

3.38 A. Jain: *Robot and Multibody Dynamics: Analysis and Algorithms* (Springer, New York 2011)

3.39 L.W. Tsai: *Robot Analysis and Design: The Mechanics of Serial and Parallel Manipulators* (Wiley, New York 1999)

3.40 K. Yamane: *Simulating and Generating Motions of Human Figures* (Springer, Berlin, Heidelberg 2004)

3.41 F.M.L. Amirouche: *Fundamentals of Multibody Dynamics: Theory and Applications* (Birkhäuser, Boston 2006)

3.42 M.G. Coutinho: *Dynamic Simulations of Multibody Systems* (Springer, New York 2001)

3.43 E.J. Haug: *Computer Aided Kinematics and Dynamics of Mechanical Systems* (Allyn and Bacon, Boston 1989)

3.44 R.L. Huston: *Multibody Dynamics* (Butterworths, Boston 1990)

3.45 A.A. Shabana: *Computational Dynamics*, 2nd edn. (Wiley, New York 2001)

3.46 V. Stejskal, M. Valášek: *Kinematics and Dynamics of Machinery* (Marcel Dekker, New York 1996)

3.47 L. Brand: *Vector and Tensor Analysis*, 4th edn. (Wiley/Chapman Hall, New York/London 1953)

3.48 F.C. Park, J.E. Bobrow, S.R. Ploen: A lie group formulation of robot dynamics, Int. J. Robotics Res. **14**(6), 609–618 (1995)

3.49 M.E. Kahn, B. Roth: The near minimum-time control of open-loop articulated kinematic chains, J. Dyn. Syst. Meas. Control **93**, 164–172 (1971)

3.50 J.J. Uicker: Dynamic force analysis of spatial linkages, Trans. ASME J. Appl. Mech. **34**, 418–424 (1967)
3.51 A. Jain: Unified formulation of dynamics for serial rigid multibody systems, J. Guid. Control Dyn. **14**(3), 531–542 (1991)
3.52 G. Rodriguez: Kalman filtering, smoothing, and recursive robot arm forward and inverse dynamics, IEEE J. Robotics Autom. **RA-3**(6), 624–639 (1987)
3.53 G. Rodriguez, A. Jain, K. Kreutz-Delgado: A spatial operator algebra for manipulator modelling and control, Int. J. Robotics Res. **10**(4), 371–381 (1991)
3.54 J.M. Hollerbach: A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity, IEEE Trans. Syst. Man Cybern. **SMC-10**(11), 730–736 (1980)
3.55 M.W. Spong, S. Hutchinson, M. Vidyasagar: *Robot Modeling and Control* (Wiley, Hoboken 2006)
3.56 K.W. Buffinton: Kane's Method in Robotics. In: *Robotics and Automation Handbook*, ed. by T.R. Kurfess (CRC, Boca Raton 2005), 6-1–6-31
3.57 T.R. Kane, D.A. Levinson: The use of kane's dynamical equations in robotics, Int. J. Robotics Res. **2**(3), 3–21 (1983)
3.58 C.A. Balafoutis, R.V. Patel, P. Misra: Efficient modeling and computation of manipulator dynamics using orthogonal cartesian tensors, IEEE J. Robotics Autom. **4**, 665–676 (1988)
3.59 X. He, A.A. Goldenberg: An algorithm for efficient computation of dynamics of robotic manipulators, Proc. 4th Int. Conf. Adv. Robotics, Columbus (1989) pp. 175–188
3.60 W. Hu, D.W. Marhefka, D.E. Orin: Hybrid kinematic and dynamic simulation of running machines, IEEE Trans. Robotics **21**(3), 490–497 (2005)
3.61 C.A. Balafoutis, R.V. Patel: Efficient computation of manipulator inertia matrices and the direct dynamics problem, IEEE Trans. Syst. Man Cybern. **19**, 1313–1321 (1989)
3.62 K.W. Lilly, D.E. Orin: Alternate formulations for the manipulator inertia matrix, Int. J. Robotics Res. **10**, 64–74 (1991)
3.63 S. McMillan, D.E. Orin: Forward dynamics of multilegged vehicles using the composite rigid body method, Proc. IEEE Int. Conf. Robotics Autom. (1998) pp. 464–470
3.64 U.M. Ascher, D.K. Pai, B.P. Cloutier: Forward dynamics: Elimination methods, and formulation stiffness in robot simulation, Int. J. Robotics Res. **16**(6), 749–758 (1997)
3.65 R. Featherstone: A divide-and-conquer articulated-body algorithm for parallel $O(\log(n))$ calculation of rigid-body dynamics. Part 2: Trees, loops and accuracy, Int. J. Robotics Res. **18**(9), 876–892 (1999)
3.66 MSC Software Corporation: Adams, http://www.mscsoftware.com/
3.67 T. Kane, D. Levinson: *Autolev user's manual* (OnLine Dynamics Inc., Sunnyvale 2005)
3.68 Real-Time Physics Simulation: Bullet, http://bulletphysics.org/wordpress (2015)
3.69 Georgia Tech Graphics Lab and Humanoid Robotics Lab: DART, http://dartsim.github.io (2011)
3.70 S. McMillan, D.E. Orin, R.B. McGhee: DynaMechs: An object oriented software package for efficient dynamic simulation of underwater robotic vehicles. In: *Underwater Robotic Vehicles: Design and Control*, ed. by J. Yuh (TSI Press, Albuquerque 1995) pp. 73–98
3.71 Open Source Robotics Foundation: Gazebo, http://gazebosim.org (2002)
3.72 R. Smith: Open Dynamics Engine User Guide, http://opende.sourceforge.net (2006)
3.73 Microsoft Corporation: Robotics Developer Studio, http://www.microsoft.com/robotics (2010)
3.74 P.I. Corke: A robotics toolbox for MATLAB, IEEE Robotics Autom. Mag. **3**(1), 24–32 (1996)
3.75 Robotran: http://www.robotran.be (Center for Research in Mechatronics, Université catholique de Louvain 2015)
3.76 J.C. Samin, P. Fisette: *Symbolic Modeling of Multibody Systems* (Kluwer, Dordrecht 2003)
3.77 M.G. Hollars, D.E. Rosenthal, M.A. Sherman: *SD/FAST User's Manual* (Symbolic Dynamics Inc., Mountain View 1994)
3.78 M. Sherman, P. Eastman: Simbody, https://simtk.org/home/simbody (2015)
3.79 G.D. Wood, D.C. Kennedy: *Simulating Mechanical Systems in Simulink with SimMechanics* (MathWorks Inc., Natick 2003)
3.80 W. Khalil, D. Creusot: SYMORO+: A system for the symbolic modeling of robots, Robotica **15**, 153–161 (1997)
3.81 W. Khalil, A. Vijayalingam, B. Khomutenko, I. Mukhanov, P. Lemoine, G. Ecorchard: OpenSYMORO: An open-source software package for symbolic modelling of robots, Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatron. (2014) pp. 126–1211
3.82 Cyberbotics Ltd.: *Webots User Guide*, http://www.cyberbotics.com (2015)
3.83 I.C. Brown, P.J. Larcombe: A survey of customised computer algebra programs for multibody dynamic modelling. In: *Symbolic Methods in Control System Analysis and Design*, ed. by N. Munro (Inst. Electr. Eng., London 1999) pp. 53–77
3.84 J.J. Murray, C.P. Neuman: ARM: An algebraic robot dynamic modeling program, Proc. IEEE Int. Conf. Robotics Autom., Atlanta (1984) pp. 103–114
3.85 J.J. Murray, C.P. Neuman: Organizing customized robot dynamic algorithms for efficient numerical evaluation, IEEE Trans. Syst. Man Cybern. **18**(1), 115–125 (1988)
3.86 F.C. Park, J. Choi, S.R. Ploen: Symbolic formulation of closed chain dynamics in independent coordinates, Mech. Mach. Theory **34**, 731–751 (1999)
3.87 M. Vukobratovic, N. Kircanski: *Real-Time Dynamics of Manipulation Robots* (Springer, New York 1985)
3.88 J. Wittenburg, U. Wolz: Mesa Verde: A symbolic program for nonlinear articulated-rigid-body dynamics, ASME Des. Eng. Div. Conf., Cincinnati (1985) pp. 1–8, ASME Paper No. 85-DET-151
3.89 J.Y.S. Luh, C.S. Lin: Scheduling of parallel computation for a computer-controlled mechanical manipulator, IEEE Trans. Syst. Man Cybern. **12**(2), 214–234 (1982)

3.90 D.E. Orin: Pipelined approach to inverse plant plus jacobian control of robot manipulators, Proc. IEEE Int. Conf. Robotics Autom., Atlanta (1984) pp. 169–175

3.91 R.H. Lathrop: Parallelism in manipulator dynamics, Int. J. Robotics Res. **4**(2), 80–102 (1985)

3.92 C.S.G. Lee, P.R. Chang: Efficient parallel algorithm for robot inverse dynamics computation, IEEE Trans. Syst. Man Cybern. **16**(4), 532–542 (1986)

3.93 M. Amin-Javaheri, D.E. Orin: Systolic architectures for the manipulator inertia matrix, IEEE Trans. Syst. Man Cybern. **18**(6), 939–951 (1988)

3.94 C.S.G. Lee, P.R. Chang: Efficient parallel algorithms for robot forward dynamics computation, IEEE Trans. Syst. Man Cybern. **18**(2), 238–251 (1988)

3.95 M. Amin-Javaheri, D.E. Orin: Parallel algorithms for computation of the manipulator inertia matrix, Int. J. Robotics Res. **10**(2), 162–170 (1991)

3.96 A. Fijany, A.K. Bejczy: A class of parallel algorithms for computation of the manipulator inertia matrix, IEEE Trans. Robotics Autom. **5**(5), 600–615 (1989)

3.97 S. McMillan, P. Sadayappan, D.E. Orin: Parallel dynamic simulation of multiple manipulator systems: Temporal versus spatial methods, IEEE Trans. Syst. Man Cybern. **24**(7), 982–990 (1994)

3.98 A. Fijany, I. Sharf, G.M.T. D'Eleuterio: Parallel $O(\log N)$ algorithms for computation of manipulator forward dynamics, IEEE Trans. Robotics Autom. **11**(3), 389–400 (1995)

3.99 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 Res. **18**(9), 867–875 (1999)

3.100 R. Featherstone, A. Fijany: A technique for analyzing constrained rigid-body systems and its application to the constraint force algorithm, IEEE Trans. Robotics Autom. **15**(6), 1140–1144 (1999)

3.101 P.S. Freeman, D.E. Orin: Efficient dynamic simulation of a quadruped using a decoupled tree-structured approach, Int. J. Robotics Res. **10**, 619–627 (1991)

3.102 Y. Nakamura, K. Yamane: Dynamics computation of structure-varying kinematic chains and its application to human figures, IEEE Trans. Robotics Autom. **16**(2), 124–134 (2000)