Roy Featherstone

Rigid Body Dynamics Algorithms

Roy Featherstone
The Austrailian National University
Canberra, ACT
Austrailia

Preface

The purpose of this book is to present a substantial collection of the most efficient algorithms for calculating rigid-body dynamics, and to explain them in enough detail that the reader can understand how they work, and how to adapt them (or create new algorithms) to suit the reader's needs. The collection includes the following well-known algorithms: the recursive Newton-Euler algorithm, the composite-rigid-body algorithm and the articulated-body algorithm. It also includes algorithms for kinematic loops and floating bases. Each algorithm is derived from first principles, and is presented both as a set of equations and as a pseudocode program, the latter being designed for easy translation into any suitable programming language.

This book also explains some of the mathematical techniques used to formulate the equations of motion for a rigid-body system. In particular, it shows how to express dynamics using six-dimensional (6D) vectors, and it explains the recursive formulations that are the basis of the most efficient algorithms. Other topics include: how to construct a computer model of a rigid-body system; exploiting sparsity in the inertia matrix; the concept of articulated-body inertia; the sources of rounding error in dynamics calculations; and the dynamics of physical contact and impact between rigid bodies.

Rigid-body dynamics has a tendency to become a sea of algebra. However, this is largely the result of using 3D vectors, and it can be remedied by using a 6D vector notation instead. This book uses a notation based on *spatial vectors*, in which the linear and angular aspects of rigid-body motion are combined into a unified set of quantities and equations. The result is typically a four- to six-fold reduction in the volume of algebra. The benefit is also felt in the computer code: shorter, clearer programs that are easier to read, write and debug, but are still just as efficient as code using standard 3D vectors.

This book is intended to be accessible to a wide audience, ranging from senior undergraduates to researchers and professionals. Readers are assumed to have some prior knowledge of rigid-body dynamics, such as might be obtained from an introductory course on dynamics, or from reading the first few chapters of an introductory text. However, no prior knowledge of 6D vectors is required, as this topic is explained from the beginning in Chapter 2. This book does also contain some advanced material, such as might be of interest to dynamics

experts and scholars of 6D vectors. No software is distributed with this book, but readers can obtain source code for most of the algorithms described here from the author's web site.

This text was originally intended to be a second edition of a book entitled *Robot Dynamics Algorithms*, which was published back in 1987; but it quickly became clear that there was enough new material to justify the writing of a whole new book. Compared with its predecessor, the most notable new materials to be found here are: explicit pseudocode descriptions of the algorithms; a chapter on how to model rigid-body systems; algorithms to exploit branch-induced sparsity; an enlarged treatment of kinematic loops and floating-base systems; planar vectors (the planar equivalent of spatial vectors); numerical errors and model sensitivity; and guidance on how to implement spatial-vector arithmetic.

Contents

Chapter 1

Introduction

Rigid-body dynamics is an old subject that has been rejuvenated and transformed by the computer. Today, we can find dynamics calculations in computer games, in animation and virtual-reality software, in simulators, in motion control systems, and in a variety of engineering design and analysis tools. In every case, a computer is calculating the forces, accelerations, and so on, associated with the motion of a rigid-body approximation of a physical system.

The main purpose of this book is to present a collection of efficient algorithms for performing various dynamics calculations on a computer. Each algorithm is described in detail, so that the reader can understand how it works and why it is efficient; and basic concepts are explained, such as recursive formulation, branch-induced sparsity and articulated-body inertia.

Rigid-body dynamics is usually expressed using 3D vectors. However, the subject of this book is dynamics *algorithms*, and this subject is better expressed using 6D vectors. We therefore adopt a 6D notation based on spatial vectors, which is explained in Chapter 2. Compared with 3D vectors, spatial notation greatly reduces the volume of algebra, simplifies the tasks of describing and explaining a dynamics algorithm, and simplifies the process of implementing an algorithm on a computer.

This chapter provides a short introduction to the subject matter of this book. It says a little about dynamics algorithms, a little about spatial vectors, and it explains how the book is organized.

1.1   Dynamics Algorithms

The dynamics of a rigid-body system is described its equation of motion, which specifies the relationship between the forces acting on the system and the accelerations they produce. A dynamics algorithm is a procedure for calculating the numeric values of quantities that are relevant to the dynamics. We will be concerned mainly with algorithms for two particular calculations:

forward dynamics: the calculation of the acceleration response of a given rigid-body system to a given applied force; and

inverse dynamics: the calculation of the force that must be applied to a given rigid-body system in order to produce a given acceleration response.

Forward dynamics is used mainly in simulation. Inverse dynamics has a variety of uses, such as: motion control systems, trajectory planning, mechanical design, and as a component in a forward-dynamics calculation. In Chapter 9 we will consider a third kind of calculation, called hybrid dynamics, in which some of the acceleration and force variables are given and the task is to calculate the rest.

The equation of motion for a rigid-body system can be written in the following canonical form:

$$\boldsymbol{\tau} = \boldsymbol{H}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,. \tag{1.1}$$

In this equation, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are vectors of position, velocity and acceleration variables, respectively, and $\boldsymbol{\tau}$ is a vector of applied forces. $\boldsymbol{H}$ is a matrix of inertia terms, and is written $\boldsymbol{H}(\boldsymbol{q})$ to show that it is a function of $\boldsymbol{q}$. $\boldsymbol{C}$ is a vector of force terms that account for the Coriolis and centrifugal forces, gravity, and any other forces acting on the system other than those in $\boldsymbol{\tau}$. It is written $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ to show that it depends on both $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Together, $\boldsymbol{H}$ and $\boldsymbol{C}$ are the coefficients of the equation of motion, and $\boldsymbol{\tau}$ and $\ddot{\boldsymbol{q}}$ are the variables. Typically, one of the variables is given and the other is unknown.

Although it is customary to write $\boldsymbol{H}(\boldsymbol{q})$ and $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, it would be more accurate to write $\boldsymbol{H}(model, \boldsymbol{q})$ and $\boldsymbol{C}(model, \boldsymbol{q}, \dot{\boldsymbol{q}})$, to indicate that both $\boldsymbol{H}$ and $\boldsymbol{C}$ depend on the particular rigid-body system to which they apply. The symbol *model* refers to a collection of data that describes a particular rigid-body system in terms of its component parts: the number of bodies and joints, the manner in which they are connected together, and the values of every parameter associated with each component (inertia parameters, geometric parameters, and so on). We call this description a *system model* in order to distinguish it from a mathematical model. The difference is this: a system model describes the system itself, whereas a mathematical model describes some aspect of its behaviour. Equation 1.1 is a mathematical model. On a computer, *model* would be a variable of type 'system model', and its value would be a data structure containing the system model of a specific rigid-body system.

Let us encapsulate the forward and inverse dynamics calculations into a pair of functions, FD and ID. These functions satisfy

$$\ddot{\boldsymbol{q}} = \mathrm{FD}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}) \tag{1.2}$$

and

$$\boldsymbol{\tau} = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\,. \tag{1.3}$$

On comparing these equations with Eq. 1.1, it is obvious that FD and ID must evaluate to $\boldsymbol{H}^{-1}(\boldsymbol{\tau} - \boldsymbol{C})$ and $\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C}$, respectively. However, the point of these

equations is that they show clearly the inputs and outputs of each calculation. In particular, they show that the system model is an input in both cases. Thus, there is an expectation that the algorithms implementing FD and ID will work for a class of rigid-body systems, and will use the data in the system model to work out the dynamics of the particular rigid-body system described by that model. We shall use the name *model-based algorithm* to refer to algorithms that work like this.

The big advantage of this approach is that a single piece of computer code can be written, tested, documented, and so on, to calculate the dynamics of any rigid-body system in a broad class. The two main classes of interest are called kinematic trees and closed-loop systems. Roughly speaking, a *kinematic tree* is any rigid-body system that does not contain kinematic loops, and a *closed-loop system* is any rigid-body system that is not a kinematic tree.[1] Calculating the dynamics of a kinematic tree is significantly easier than for a closed-loop system.

Model-based algorithms can be classified according to their two main attributes: what calculation they perform, and what class of system they apply to. The main body of algorithms in this book consists of forward and inverse dynamics algorithms for kinematic trees and closed-loop systems.

1.2 Spatial Vectors

A rigid body in 3D space has six degrees of motion freedom; yet we usually express its dynamics using 3D vectors. Thus, to state the equation of motion for a rigid body, we must actually state two vector equations:

$$\boldsymbol{f} = m\,\boldsymbol{a}_C \qquad \text{and} \qquad \boldsymbol{n}_C = \boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}\,. \tag{1.4}$$

The first expresses the relationship between the force applied to the body and the linear acceleration of its centre of mass. The second expresses the relationship between the moment applied to the body, referred to its centre of mass, and its angular acceleration.

In spatial vector notation, we use 6D vectors that combine the linear and angular aspects of rigid-body motion. Thus, linear and angular acceleration are combined to form a spatial acceleration vector, force and moment are combined to form a spatial force vector, and so on. Using spatial notation, the equation of motion for a rigid body can be written

$$\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}\,, \tag{1.5}$$

where $\boldsymbol{f}$ is the spatial force acting on the body, $\boldsymbol{v}$ and $\boldsymbol{a}$ are the body's spatial velocity and acceleration, and $\boldsymbol{I}$ is the body's spatial inertia tensor. The symbol $\times^*$ denotes a spatial vector cross product. One obvious feature of this equation

---

[1] A precise definition will be given in Chapter 4, including a definition of 'kinematic loop.'

is that there are some name clashes with Eq. 1.4. We can solve this problem by placing hats above the spatial symbols (e.g. $\hat{\boldsymbol{f}}$).

Spatial notation offers many more simplifications than just the equation of motion. For example, if two rigid bodies are joined together to form a single rigid body, then the spatial inertia of the new body is given by the simple formula

$$\boldsymbol{I}_{new} = \boldsymbol{I}_1 + \boldsymbol{I}_2\,,$$

where $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ are the inertias of the two original bodies. This equation replaces three in the 3D vector approach: one to compute the new mass, one to compute the new centre of mass, and one to compute the new rotational inertia about the new centre of mass. The overall effect of all these simplifications is that spatial notation typically cuts the volume of algebra by at least a factor of 4 compared with standard 3D vector notation. With the barrier of algebra out of the way, the analyst is free to state a problem more succinctly, to solve it in fewer steps, and to arrive at a more compact solution.

Another benefit of spatial vectors is that they simplify the process of writing computer code. The resulting code is shorter, and is easier to read, write and debug, but is no less efficient than software using 3D vectors. For example, some programming languages will allow you to implement Eq. 1.5 with a statement like

```
f = I*a + v.cross(I*v);
```

in which the type of each variable has been declared to the compiler. Thus, the compiler knows that `I` and `v` denote a rigid-body inertia and a spatial motion vector, respectively, so it can compile the expression `I*v` into code that calls a routine in the spatial arithmetic library which is designed to perform this particular multiplication efficiently.

1.3   Units and Notation

Angles are assumed to be measured in radians. Apart from that, the equations in this book are independent of the choice of units, and require only that the chosen system of units be consistent. In the few places where units are mentioned explicitly, SI units are used.

The mathematical notation generally follows ISO guidelines. Thus, variables are set in italic; constants and functions are set in roman; and vectors and matrices are set in bold italic. Vectors are denoted by lower-case letters,[2] and matrices by upper-case. A short list of symbols can be found on page 265, and some symbols have entries in the index.

A few details are worth mentioning here. The symbols $\boldsymbol{0}$ and $\boldsymbol{1}$ denote the zero and identity matrices, respectively, and $\boldsymbol{0}$ also denotes the zero vector. The superscript $^{-\mathrm{T}}$ denotes the transpose of the inverse, so $\boldsymbol{A}^{-\mathrm{T}}$ means $(\boldsymbol{A}^{-1})^{\mathrm{T}}$. An

---

[2]The vector $\boldsymbol{C}$ in Eq. 1.1 is the only exception to this rule.

expression like $\boldsymbol{a}\times$ denotes an operator that maps $\boldsymbol{b}$ to $\boldsymbol{a} \times \boldsymbol{b}$. If a quantity is described as an array, then it is a numbered list of things. If λ is an array then element i of λ is written $\lambda(i)$. If a symbol has a leading superscript (e.g. $^A\boldsymbol{v}$) then that superscript identifies a coordinate system. Coordinate systems can also appear in subscripts. Spatial vectors are sometimes marked with a hat, and coordinate vectors with an underline, as explained in Chapter 2.

1.4 Readers' Guide

The contents of this book can be divided into three parts: preparation, main algorithms and additional topics. The preparatory chapters are 2, 3 and 4; and the main algorithms are are described in Chapters 5 to 8. Readers with enough background may find they can skip straight to Chapter 5. Chapters 9 onwards assume a basic understanding of earlier material, but are otherwise self-contained.

Chapter 2 describes spatial vector algebra from first principles; and Chapter 3 explains how to formulate and analyse the equations of motion for a rigid-body system. These are the two most mathematical chapters, and they both cover their subject matter in greater depth than the minimum required for the algorithms that follow. Chapter 4 describes the various components of a system model. Many of the quantities used in later chapters are defined here.

Chapters 5, 6 and 7 describe the three best known algorithms for kinematic trees: the recursive Newton-Euler algorithm for inverse dynamics, and the composite-rigid-body and articulated-body algorithms for forward dynamics, in that order. Chapter 5 also explains the concept of recursive formulation, which is the reason for the efficiency of these algorithms. Chapter 8 then presents several techniques for calculating the dynamics of closed-loop systems. These four chapters are arranged approximately in order of increasing complexity (i.e., the algorithms get progressively more complicated).

Chapters 9, 10 and 11 then tackle a variety of extra topics. Chapter 9 considers several more algorithms, including hybrids of forward and inverse dynamics and algorithms for floating-base systems. Chapter 10 considers the issues of numerical accuracy and computational efficiency; and Chapter 11 examines the dynamics of rigid-body systems that are subject to contacts and impacts.

Readers will find examples dotted unevenly throughout the text. In some cases, they illustrate a concept that is covered in the main text. In other cases, they describe ideas that are more easily explained in the form of an example. Readers will also find that every major algorithm is presented both as a set of equations and as a pseudocode program. In many cases, the two are side-by-side. The pseudocode is intended to be self-explanatory and easy to translate into real computer code.

Many people, including the author, find it helps to have some working source code to hand when studying a new algorithm. No source code is distributed with this book, on the grounds that such software becomes obsolete too quickly.

Readers should instead look for this code on the Web. At the time of writing, the author's web site[3] contained source code for most of the algorithms described here, but readers may also be able to find useful software elsewhere.

1.5   Further Reading

A variety of books on rigid-body dynamics have been published in recent years, although many are now out of print. The following are suitable introductory texts: Amirouche (2006), Moon (1998), Shabana (2001) and Huston (1990). The first three are formal teaching texts with copious examples and homework exercises, while Huston's book is more tutorial in nature. All are designed to be accessible to undergraduates. More advanced texts include Stejskal and Valášek (1996), Roberson and Schwertassek (1988) and Wittenburg (1977). For a different perspective, try Coutinho (2001), which is aimed more at computer games and virtual reality.

Much of the material in this book has its origins in the robotics community. Other books on robot dynamics include Balafoutis and Patel (1991), Lilly (1993), Yamane (2004) and the author's earlier work, Featherstone (1987), which has been superseded by the present volume. Sections and chapters on dynamics can be found in Angeles (2003), Khalil and Dombre (2002), Siciliano and Khatib (2008) and various other robotics books.

Readers with an interest in 6D vectors of various kinds may find the following books and articles of interest: Ball (1900), Brand (1953), von Mises (1924a,b), Murray et al. (1994) and Selig (1996). Also, the treatment of spatial vectors in Featherstone (1987) is a little different from that presented here.

---

[3]The URL is (or was) `http://users.rsise.anu.edu.au/~roy/spatial/`. If that doesn't work, then try searching for 'Roy Featherstone'.

Chapter 2

Spatial Vector Algebra

Spatial vectors are 6D vectors that combine the linear and angular aspects of rigid-body motions and forces. They provide a compact notation for studying rigid-body dynamics, in which a single spatial vector can do the work of two 3D vectors, and a single spatial equation replaces two (or sometimes more) 3D vector equations. Spatial vector notation allows us to develop equations of motion quickly, and to express them succinctly in symbolic form. It also allows us to derive dynamics algorithms quickly, and to express them in a compact form that is easily converted into efficient computer code.

This chapter presents the spatial vector algebra from first principles up to the equation of motion for a single rigid body. It also presents the planar vector algebra, which is an adaptation of spatial vector algebra to rigid bodies that move only in a 2D plane. Methods of implementing spatial vector arithmetic are covered in Appendix A; and the use of spatial vectors to analyse general rigid-body systems is covered in Chapter 3.

2.1   Mathematical Preliminaries

This section reviews briefly some of the mathematical concepts and notations used in spatial vector algebra. More details can be found in books like Greenwood (1988), Selig (1996) and many others.

Vectors and Vector Spaces: Linear algebra defines a vector as being an element of a vector space; and different kinds of vector are elements of different vector spaces. Four vector spaces occur frequently in this book, and have been given special names:

$$
\begin{array}{lll}
\mathsf{R}^n & \text{---} & \text{coordinate vectors} \\
\mathsf{E}^n & \text{---} & \text{Euclidean vectors} \\
\mathsf{M}^n & \text{---} & \text{spatial motion vectors} \\
\mathsf{F}^n & \text{---} & \text{spatial force vectors}
\end{array}
$$

In each case, the superscript indicates the dimension. One more space that occurs frequently is the space of $m \times n$ matrices, which is denoted $\mathsf{R}^{m \times n}$.

A coordinate vector is an n-tuple of real numbers, or, in matrix form, an $n \times 1$ matrix of real numbers (i.e., a column vector). Coordinate vectors typically represent other vectors; and we use the term *abstract vector* to refer to the vector being represented. Euclidean vectors have the special property that a Euclidean inner product is defined on them. This product endows them with the familiar properties of magnitude and direction. The 3D vectors used to describe rigid-body dynamics are Euclidean vectors. Spatial vectors are not Euclidean, but are instead the elements of a pair of vector spaces: one for motion vectors and one for forces. Spatial motion vectors describe attributes of rigid-body motion, such as velocity and acceleration, while spatial force vectors describe force, impulse and momentum. The two spaces M^6 and F^6 are the main topic of this chapter.

Hats and Underlines: When spatial vectors and 3D vectors appear together, we mark some spatial vectors with hats (e.g. $\hat{\boldsymbol{v}}$, $\hat{\boldsymbol{f}}$) in order to avoid name clashes with 3D vectors having the same name. Likewise, we normally make no distinction between coordinate vectors and the abstract vectors they represent, but if a distinction is required then we underline the coordinate vector (e.g. $\underline{\boldsymbol{v}}$ representing $\boldsymbol{v}$). We use these notational devices only where they are needed, and hardly ever outside this chapter.

The Dual of a Vector Space: Let V be a vector space. Its dual, denoted V^*, is a vector space having the same dimension as V, and having the property that a scalar product is defined between it and V (i.e., the scalar product takes one argument from each space). If $\boldsymbol{u} \in V^*$ and $\boldsymbol{v} \in V$ then this scalar product can be written either $\boldsymbol{u} \cdot \boldsymbol{v}$ or $\boldsymbol{v} \cdot \boldsymbol{u}$, the two expressions meaning the same. Duality is a symmetrical relationship: if $U = V^*$ then $V = U^*$.

The notion of duality is relevant to spatial vector algebra because the spaces M^n and F^n are dual (i.e., each is the dual of the other). In particular, a scalar product is defined between motion vectors and force vectors such that if $\boldsymbol{m} \in \mathsf{M}^6$ describes the velocity of a rigid body and $\boldsymbol{f} \in \mathsf{F}^6$ describes the force acting on it, then $\boldsymbol{m} \cdot \boldsymbol{f}$ is the power delivered by the force.

The scalar product between a vector space and its dual is required to be nondegenerate (also called nonsingular). A scalar product is nondegenerate if it has the following property: for any $\boldsymbol{v} \in V$, if $\boldsymbol{v} \neq \boldsymbol{0}$ then there exists at least one vector $\boldsymbol{u} \in V^*$ satisfying $\boldsymbol{v} \cdot \boldsymbol{u} \neq 0$. This property is a sufficient condition to guarantee the existence of a dual basis on V and V^*.

Dual Bases: Suppose we have two vector spaces, U and V, such that $U = V^*$. Let $\mathcal{D} = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n\}$ be a basis on U, and let $\mathcal{E} = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ be a basis on V. If these bases satisfy the following condition, then they form a dual basis

on U and V:

$$\boldsymbol{d}_i \cdot \boldsymbol{e}_j = \left\{ \begin{array}{ll} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{array} \right.$$

This is known as the reciprocity condition. If $\mathcal{D}$ and $\mathcal{E}$ satisfy this condition, then each is the reciprocal of the other. If one basis is given, then the other is determined uniquely by the reciprocity condition. We can use the notation $\mathcal{D}^*$ to denote the reciprocal of $\mathcal{D}$. Note that a dual basis consists of two sets of basis vectors, and covers two vector spaces.

If $U = \mathsf{E}^n$ then it becomes possible to set $V = U$, so that $\mathcal{D}$ and $\mathcal{E}$ are bases on the same vector space. It also becomes possible to find special cases of $\mathcal{D}$ with the property $\mathcal{D} = \mathcal{E}$. Such bases are *orthonormal*. Thus, an orthonormal basis is a special case of a dual basis in which $U = V = \mathsf{E}^n$ and $\mathcal{D} = \mathcal{E}$.

Dual Coordinates: A dual basis defines a dual coordinate system. We will always use dual coordinates for representing spatial vectors; and we will mostly use Plücker coordinates, which are a particularly convenient choice of dual coordinate system. The special property of dual coordinates is that

$$\boldsymbol{u} \cdot \boldsymbol{v} = \underline{\boldsymbol{u}}^{\mathrm{T}} \underline{\boldsymbol{v}} \,,$$

where $\underline{\boldsymbol{u}}$ and $\underline{\boldsymbol{v}}$ are coordinate vectors representing $\boldsymbol{u} \in U$ and $\boldsymbol{v} \in V$ in the dual coordinate system defined by $\mathcal{D}$ and $\mathcal{E}$. An immediate corollary is that the individual elements in $\underline{\boldsymbol{u}}$ and $\underline{\boldsymbol{v}}$ are given by

$$u_i = \boldsymbol{e}_i \cdot \boldsymbol{u} \qquad \text{and} \qquad v_i = \boldsymbol{d}_i \cdot \boldsymbol{v} \,.$$

To perform a coordinate transformation from one dual coordinate system to another, we need two transformation matrices: one to operate on $\underline{\boldsymbol{u}}$, and one to operate on $\underline{\boldsymbol{v}}$. If $\boldsymbol{X}$ is the matrix that performs the desired coordinate transformation on $\underline{\boldsymbol{u}}$, then the matrix that does the same job on $\underline{\boldsymbol{v}}$ is denoted $\boldsymbol{X}^*$, and the two matrices are related by the equation

$$\boldsymbol{X}^* = \boldsymbol{X}^{-\mathrm{T}} \,.$$

This relationship follows from the requirement that $\underline{\boldsymbol{u}}^{\mathrm{T}} \underline{\boldsymbol{v}} = (\boldsymbol{X}\underline{\boldsymbol{u}})^{\mathrm{T}}(\boldsymbol{X}^*\underline{\boldsymbol{v}})$ for all $\underline{\boldsymbol{u}}$ and $\underline{\boldsymbol{v}}$.

Operators $\boldsymbol{a}\cdot$ and $\boldsymbol{a}\times$: It is possible to interpret expressions like $\boldsymbol{a} \cdot \boldsymbol{b}$ and $\boldsymbol{a} \times \boldsymbol{b}$ as being the result of applying an operator, $\boldsymbol{a}\cdot$ or $\boldsymbol{a}\times$, to the operand $\boldsymbol{b}$. Thus, $\boldsymbol{a}\cdot$ is the operator that maps $\boldsymbol{b}$ to the scalar $\boldsymbol{a} \cdot \boldsymbol{b}$, while $\boldsymbol{a}\times$ maps $\boldsymbol{b}$ to $\boldsymbol{a} \times \boldsymbol{b}$. If $\boldsymbol{a}$ is a coordinate vector, then $\boldsymbol{a}\cdot = \boldsymbol{a}^{\mathrm{T}}$, and $\boldsymbol{a}\times$ is a square matrix. Some properties of $\boldsymbol{a}\times$ are listed in Tables 2.1 and 2.3. Table 2.3 also lists properties of the operator $\boldsymbol{a}\times^*$, which is the dual of $\boldsymbol{a}\times$.

Dyads and Dyadics: An expression of the form $\boldsymbol{a}\,\boldsymbol{b}\cdot$ is called a dyad. It is a linear operator that maps vectors to vectors. In particular, the dyad $\boldsymbol{a}\,\boldsymbol{b}\cdot$ will map the vector $\boldsymbol{c}$ to the vector $\boldsymbol{a}\,(\boldsymbol{b}\cdot\boldsymbol{c})$, which is a scalar multiple of $\boldsymbol{a}$. If $\underline{\boldsymbol{a}}$ and $\underline{\boldsymbol{b}}$ are coordinate vectors representing $\boldsymbol{a}$ and $\boldsymbol{b}$, then the dyad $\boldsymbol{a}\,\boldsymbol{b}\cdot$ is represented by the matrix $\underline{\boldsymbol{a}}\,\underline{\boldsymbol{b}}^{\mathrm{T}}$, which has a rank of 1. In the most general case, we can have $\boldsymbol{a}\in U$, $\boldsymbol{b}\in V$ and $\boldsymbol{c}\in W$, where there are no restrictions on the choice of any of these vector spaces, except that a scalar product must be defined between V and W. Such a dyad maps from W to U.

A dyadic (or dyadic tensor) is a general linear mapping of vectors. Any dyadic can be expressed as a sum of r linearly independent dyads:

$$L = \sum_{i=1}^{r} \boldsymbol{a}_i\,\boldsymbol{b}_i\cdot\,,$$

where $\boldsymbol{a}_i\in U$, $\boldsymbol{b}_i\in V$, and $r\le\min(m,n)$, where $m=\dim(U)$ and $n=\dim(V)$. (A set of dyads is linearly independent if the vectors $\boldsymbol{a}_i$ are linearly independent in U, and the vectors $\boldsymbol{b}_i$ are linearly independent in V.) If $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are coordinate vectors, then $\boldsymbol{L}$ can be written

$$L = \sum_{i=1}^{r} \boldsymbol{a}_i\,\boldsymbol{b}_i^{\mathrm{T}}\,,$$

which is an $m\times n$ matrix of rank r. If $m=n=r$ then $\boldsymbol{L}$ is a $1:1$ mapping, and is invertible. Dyadics are used in connection with the spatial inertia tensor.

2.2   Spatial Velocity

Let us work out a formula for the spatial velocity of a rigid body. We start with a rigid body, B, and choose a fixed point, O, which can be located anywhere in space. (See Figure 2.1(a).) Given O, the velocity of B can be specified by a pair of 3D vectors: the linear velocity, $\boldsymbol{v}_O$, of the body-fixed point that currently coincides with O, and an angular velocity vector, $\boldsymbol{\omega}$. The body as a whole can then be regarded as translating with a linear velocity of $\boldsymbol{v}_O$, while simultaneously rotating with an angular velocity of $\boldsymbol{\omega}$ about an axis passing through O.

From this description, we can calculate the velocity of any other point in the body using the formula

$$\boldsymbol{v}_P = \boldsymbol{v}_O + \boldsymbol{\omega}\times\overrightarrow{OP}\,, \tag{2.1}$$

where P is the point of interest, $\boldsymbol{v}_P$ is the velocity of the body-fixed point currently at P, and $\overrightarrow{OP}$ gives the position of P relative to O. This equation shows clearly that there are two contributions to $\boldsymbol{v}_P$: one due to the translation of the whole body by $\boldsymbol{v}_O$, and one due to its rotation by $\boldsymbol{\omega}$ about an axis passing
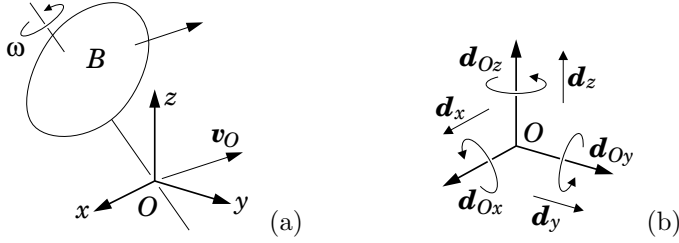
Figure 2.1: The velocity of a rigid body expressed in terms of $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ (a), and the basis vectors for Plücker motion coordinates (b)

through O. Although each individual contribution depends on the position of O, their sum does not (i.e., the dependencies cancel out), so that $\boldsymbol{v}_P$ depends only on the motion of B and the location of P. Note that P, $\boldsymbol{v}_P$ and $\boldsymbol{\omega}$ together describe the same rigid-body motion as O, $\boldsymbol{v}_O$ and $\boldsymbol{\omega}$.

We now introduce a Cartesian coordinate frame, O_{xyz}, with its origin at O. This frame defines three mutually perpendicular directions, x, y and z, and three directed lines, Ox, Oy and Oz, each passing through the point O. This frame also defines an orthonormal basis, $\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\} \subset \mathsf{E}^3$, which we can use to express $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ in terms of their Cartesian coordinates:

$$\boldsymbol{\omega} = \omega_x \boldsymbol{i} + \omega_y \boldsymbol{j} + \omega_z \boldsymbol{k}$$

and

$$\boldsymbol{v}_O = v_{Ox} \boldsymbol{i} + v_{Oy} \boldsymbol{j} + v_{Oz} \boldsymbol{k} \, .$$

Having resolved $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ into their coordinates, we can describe the velocity of B as the sum of six elementary motions: a rotation of magnitude ω_x about the line Ox, a rotation of magnitude ω_y about Oy, a rotation of magnitude ω_z about Oz, and translations of magnitudes v_{Ox}, v_{Oy} and v_{Oz} in the x, y and z directions, respectively.

Our objective is to obtain a spatial velocity vector, $\hat{\boldsymbol{v}} \in \mathsf{M}^6$, that describes the same motion as $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$. We can accomplish this by first defining a basis on M^6, and then working out the coordinates of $\hat{\boldsymbol{v}}$ in that basis. The basis we will use is

$$\mathcal{D}_O = \{\boldsymbol{d}_{Ox}, \boldsymbol{d}_{Oy}, \boldsymbol{d}_{Oz}, \boldsymbol{d}_x, \boldsymbol{d}_y, \boldsymbol{d}_z\} \subset \mathsf{M}^6 \, , \tag{2.2}$$

which is illustrated in Figure 2.1(b). This is called a Plücker basis, and it defines a Plücker coordinate system on M^6. The first three basis vectors, $\boldsymbol{d}_{Ox}$, $\boldsymbol{d}_{Oy}$ and $\boldsymbol{d}_{Oz}$, are unit rotations about the lines Ox, Oy and Oz, respectively, and the rest are unit translations in the x, y and z directions, respectively. On comparing the definitions of these basis vectors with the six elementary motions just described, it follows immediately that

$$\hat{\boldsymbol{v}} = \omega_x \boldsymbol{d}_{Ox} + \omega_y \boldsymbol{d}_{Oy} + \omega_z \boldsymbol{d}_{Oz} + v_{Ox} \boldsymbol{d}_x + v_{Oy} \boldsymbol{d}_y + v_{Oz} \boldsymbol{d}_z \, . \tag{2.3}$$

Thus, the Plücker coordinates of $\hat{v}$ in $\mathcal{D}_O$ are the Cartesian coordinates of $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ in $\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$. Each individual term on the right-hand side depends on the position and orientation of O_{xyz}, but it can be shown that the expression as a whole is invariant. (See Example 2.2.) The coordinate vector that represents $\hat{v}$ in $\mathcal{D}_O$ coordinates can be written

$$\underline{\hat{\boldsymbol{v}}}_O = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_{Ox} \\ v_{Oy} \\ v_{Oz} \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{\omega}} \\ \underline{\boldsymbol{v}}_O \end{bmatrix}, \tag{2.4}$$

where the expression on the far right is a convenient compact notation in which the 6D coordinate vector is expressed as the concatenation of the two 3D coordinate vectors $\underline{\boldsymbol{\omega}} = [\,\omega_x\ \omega_y\ \omega_z\,]^{\mathrm{T}}$ and $\underline{\boldsymbol{v}}_O = [\,v_{Ox}\ v_{Oy}\ v_{Oz}\,]^{\mathrm{T}}$.

Example 2.1 *Equation 2.1 associates a vector $\boldsymbol{v}_P$ with each point P in space. It therefore defines a vector field—the velocity field of body B. We could have written this equation in the form*

$$\boldsymbol{V}(P) = \boldsymbol{v}_O + \boldsymbol{\omega} \times \overrightarrow{OP}, \tag{2.5}$$

where $\boldsymbol{V}$ denotes the vector field, and $\boldsymbol{V}(P)$ is the vector associated by $\boldsymbol{V}$ with the point P. Although O appears in the definition of $\boldsymbol{V}$, the right-hand side is invariant with respect to the location of O, and so the field itself is independent of O. This connection between spatial vectors and vector fields can help one to visualize what a spatial vector really is, and what a spatial-vector expression really means. For example, suppose $\hat{v}_1$ and $\hat{v}_2$ are two spatial velocities, and $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ are the corresponding vector fields. If $\hat{v}_{sum} = \hat{v}_1 + \hat{v}_2$ then the vector field that corresponds to $\hat{v}_{sum}$ has the property $\boldsymbol{V}_{sum}(P) = \boldsymbol{V}_1(P) + \boldsymbol{V}_2(P)$ for all P.

Example 2.2 *Let us investigate the invariance of Eq. 2.3 with respect to the choice of origin. We can do this by defining $\hat{v}'$ to be the spatial velocity obtained by using the point P as origin, and showing that $\hat{v}' = \hat{v}$. The expression for $\hat{v}'$ is therefore*

$$\hat{v}' = \omega_x \boldsymbol{d}_{Px} + \omega_y \boldsymbol{d}_{Py} + \omega_z \boldsymbol{d}_{Pz} + v_{Px} \boldsymbol{d}_x + v_{Py} \boldsymbol{d}_y + v_{Pz} \boldsymbol{d}_z \,,$$

and the task is to equate this expression with Eq. 2.3. To keep things simple, we shall choose P to have the coordinates $(r, 0, 0)$ relative to O_{xyz}, as shown in Figure 2.2. In this case, the linear velocity coordinates are given by Eq. 2.1 as

$$\begin{bmatrix} v_{Px} \\ v_{Py} \\ v_{Pz} \end{bmatrix} = \underline{\boldsymbol{v}}_O + \underline{\boldsymbol{\omega}} \times \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} v_{Ox} \\ v_{Oy} + \omega_z r \\ v_{Oz} - \omega_y r \end{bmatrix} \,;$$
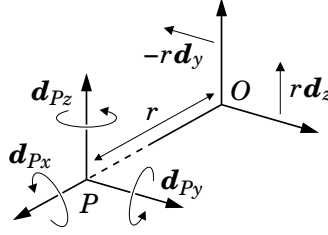
Figure 2.2: Expressing $\boldsymbol{d}_{Py}$ and $\boldsymbol{d}_{Pz}$ at O

and the three basis vectors that depend on P are

$$
\begin{aligned}
\boldsymbol{d}_{Px} &= \boldsymbol{d}_{Ox}\,, \\
\boldsymbol{d}_{Py} &= \boldsymbol{d}_{Oy} + r\boldsymbol{d}_z\,, \\
\boldsymbol{d}_{Pz} &= \boldsymbol{d}_{Oz} - r\boldsymbol{d}_y\,.
\end{aligned}
$$

Combining these equations gives

$$
\begin{aligned}
\hat{\boldsymbol{v}}' &= \omega_x \boldsymbol{d}_{Ox} + \omega_y(\boldsymbol{d}_{Oy} + r\boldsymbol{d}_z) + \omega_z(\boldsymbol{d}_{Oz} - r\boldsymbol{d}_y) \\
&\quad + v_{Ox}\boldsymbol{d}_x + (v_{Oy} + \omega_z r)\boldsymbol{d}_y + (v_{Oz} - \omega_y r)\boldsymbol{d}_z\,,
\end{aligned}
$$

which can be seen to be equal to Eq. 2.3.

Returning to the three basis vectors, we have $\boldsymbol{d}_{Px} = \boldsymbol{d}_{Ox}$ because the two lines Px and Ox are the same. To understand the expression for $\boldsymbol{d}_{Py}$, imagine a rigid body that is rotating with unit angular velocity about Py, so that its spatial velocity is $\boldsymbol{d}_{Py}$. The body-fixed point at O will then have a linear velocity of magnitude r in the z direction, as shown in Figure 2.2; and so the body's velocity can be expressed as the sum of a unit angular velocity about the line Oy and a linear velocity of magnitude r in the z direction (i.e., $\boldsymbol{d}_{Oy} + r\boldsymbol{d}_z$).[1] Repeating this exercise with Pz yields the given expression for $\boldsymbol{d}_{Pz}$.

The analysis presented here is easily extended to the case of a general location for P, and a similar argument can be used to show that Eq. 2.3 is invariant with respect to the orientation of O_{xyz}.

2.3 Spatial Force

Having obtained an expression for spatial velocity, let us now do the same for spatial force. Given an arbitrary point, O, which can be located anywhere in space, the most general force that can act on a rigid body B consists of a linear force, $\boldsymbol{f}$, acting along a line that passes through O, together with a couple, $\boldsymbol{n}_O$,

---

[1]Readers who are puzzled by this are encouraged to invent and solve their own Plücker coordinate exercises. A degree of practice with these coordinates is helpful.
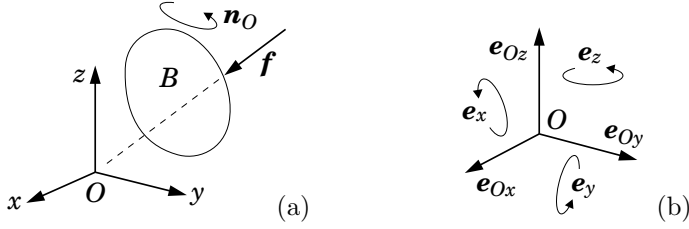
Figure 2.3: The force acting on a rigid body expressed in terms of $\boldsymbol{f}$ and $\boldsymbol{n}_O$ (a), and the basis vectors for Plücker force coordinates (b)

which is equal to the total moment about O. (See Figure 2.3(a).) Once again, we have a representation comprising one arbitrary point and two 3D vectors. $\boldsymbol{f}$ has taken the place of $\boldsymbol{\omega}$, and $\boldsymbol{n}_O$ has taken the place of $\boldsymbol{v}_O$.

Given O, $\boldsymbol{n}_O$ and $\boldsymbol{f}$, we can calculate the total moment about any other point, P, using the formula

$$\boldsymbol{n}_P = \boldsymbol{n}_O + \boldsymbol{f} \times \overrightarrow{OP}. \tag{2.6}$$

This equation is the force-vector analogue of Eq. 2.1, and it shares with that equation the property that the right-hand side is independent of the location of O. The three quantities P, $\boldsymbol{n}_P$ and $\boldsymbol{f}$ together describe the same applied force as O, $\boldsymbol{n}_O$ and $\boldsymbol{f}$.

We now introduce a Cartesian coordinate frame, O_{xyz}, with its origin at O, and use it to define the orthonormal basis $\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\} \in \mathsf{E}^3$ so that $\boldsymbol{n}_O$ and $\boldsymbol{f}$ can be written in terms of their Cartesian coordinates:

$$\boldsymbol{n}_O = n_{Ox}\boldsymbol{i} + n_{Oy}\boldsymbol{j} + n_{Oz}\boldsymbol{k}$$

and

$$\boldsymbol{f} = f_x\boldsymbol{i} + f_y\boldsymbol{j} + f_z\boldsymbol{k}.$$

Having resolved $\boldsymbol{n}_O$ and $\boldsymbol{f}$ into their components, we can describe the force acting on B as the sum of six elementary forces: couples of magnitudes n_{Ox}, n_{Oy} and n_{Oz} in the x, y and z directions, respectively, and linear forces of magnitudes f_x, f_y and f_z acting along the lines Ox, Oy and Oz, respectively.

This coordinate frame also defines the following Plücker basis on F^6:

$$\mathcal{E}_O = \{\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z, \boldsymbol{e}_{Ox}, \boldsymbol{e}_{Oy}, \boldsymbol{e}_{Oz}\} \subset \mathsf{F}^6, \tag{2.7}$$

which is illustrated in Figure 2.3(b). The first three elements are unit couples in the x, y and z directions, and the rest are unit linear forces acting along the lines Ox, Oy and Oz. On comparing the definitions of these basis vectors with the six elementary forces just described, it follows immediately that if $\hat{\boldsymbol{f}} \in \mathsf{F}^6$ is the spatial force vector representing the same force as O, $\boldsymbol{n}_O$ and $\boldsymbol{f}$, then

$$\hat{\boldsymbol{f}} = n_{Ox}\boldsymbol{e}_x + n_{Oy}\boldsymbol{e}_y + n_{Oz}\boldsymbol{e}_z + f_x\boldsymbol{e}_{Ox} + f_y\boldsymbol{e}_{Oy} + f_z\boldsymbol{e}_{Oz}. \tag{2.8}$$

Each individual term on the right-hand side of this equation depends on the position and orientation of O_{xyz}, but the expression as a whole is invariant. Thus, the value of $\hat{f}$ does not depend at all on O_{xyz}, but only on the forces acting on B. The coordinate vector representing $\hat{f}$ in $\mathcal{E}_O$ coordinates is

$$\underline{\hat{f}}_O = \begin{bmatrix} n_{Ox} \\ n_{Oy} \\ n_{Oz} \\ f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} \underline{n}_O \\ \underline{f} \end{bmatrix}, \tag{2.9}$$

where $\underline{n}_O = [n_{Ox}\ n_{Oy}\ n_{Oz}]^{\mathrm{T}}$ and $\underline{f} = [f_x\ f_y\ f_z]^{\mathrm{T}}$.

2.4 Plücker Notation

Plücker coordinates date back to the 19$^{\text{th}}$ century, but the basis vectors are new.[2] They are the coordinates of choice for 6D vectors, partly because they lend themselves to efficient computer implementation, but mainly because they are the most convenient and easy to use.

The notations appearing in Eqs. 2.4 and 2.9 combine Plücker coordinates with notations that are specific to velocity and force. In the case of velocity, we have used two different symbols (v and ω) to represent linear and angular velocity in 3D vector notation, and we have preserved these symbols in the list of Plücker coordinates. The story is the same for forces: we have used different symbols for the 3D linear force and moment vectors, and preserved their names in the list of Plücker coordinates.

In general case, the coordinates would have the same name as the vector. Thus, if $\hat{m}$ and $\hat{f}$ denote generic elements of M^6 and F^6, respectively, then their Plücker coordinates would be

$$\underline{\hat{m}}_O = \begin{bmatrix} m_x \\ m_y \\ m_z \\ m_{Ox} \\ m_{Oy} \\ m_{Oz} \end{bmatrix} = \begin{bmatrix} \underline{m} \\ \underline{m}_O \end{bmatrix} \qquad \text{and} \qquad \underline{\hat{f}}_O = \begin{bmatrix} f_{Ox} \\ f_{Oy} \\ f_{Oz} \\ f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} \underline{f} \\ \underline{f}_O \end{bmatrix}.$$

We shall always list Plücker coordinates in angular-before-linear order; that is, the three angular coordinates will always be listed first. However, readers should be aware that it is perfectly acceptable to list the three linear coordinates first, and they will encounter this alternative ordering in the literature. From a

---

[2]The coordinates appeared in Plücker (1866). The basis vectors appeared in the author's Ph.D. thesis, and again in Featherstone (1987), under the name 'standard basis'. The notation used here is different, but the concept is the same.

mathematical point of view, it makes no difference which order the coordinates are listed. However, from a computational point of view, it does matter because the software is typically written to work with one particular order.

2.5   Line Vectors and Free Vectors

A line vector is a quantity that is characterized by a directed line and a magnitude. A pure rotation of a rigid body is a line vector, and so is a linear force acting on a rigid body. A free vector is a quantity that can be characterized by a magnitude and a direction. Pure translations of a rigid body are free vectors, and so are pure couples. A line vector can be specified by five numbers, and a free vector by three. A line vector can also be specified by a free vector and any one point on the line.

Let $\hat{s}$ be any spatial vector, motion or force, and let s and s_O be the two 3D coordinate vectors that supply the Plücker coordinates of $\hat{s}$. Here are some basic facts about line vectors and free vectors.

- If $s = \mathbf{0}$ then $\hat{s}$ is a free vector.

- If $s \cdot s_O = 0$ then $\hat{s}$ is a line vector. The direction of the line is given by s, and the line itself is the set of points P that satisfy $\overrightarrow{OP} \times s = s_O$.

- Any spatial vector can be expressed as the sum of a line vector and a free vector. If the line vector must pass through a given point, then the expression is unique.

- Any spatial vector, other than a free vector, can be expressed uniquely as the sum of a line vector and a parallel free vector. The expression (for a motion vector) is

$$\begin{bmatrix} s \\ s_O - hs \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ hs \end{bmatrix} \qquad \text{where} \qquad h = \frac{s \cdot s_O}{s \cdot s} . \qquad (2.10)$$

This last result implies that any spatial vector, other than a free vector, can be described uniquely by a directed line, a linear magnitude and an angular magnitude. Free vectors can also be described in this manner, but the description is not unique, as only the direction of the line matters.

According to screw theory, the most general motion of a rigid body consists of a rotation about a line in space together with a translation along it (i.e., a helical or screwing motion). Such a motion is called a *twist*. Likewise, the most general force that can act on a rigid body consists of a linear force acting along a line in space, together with a couple acting about it (so the force and couple are parallel). Such a force is called a *wrench*. In both cases, the line itself is called the screw axis, and the ratio of the free-vector magnitude to the line-vector magnitude is called the pitch. Pure free vectors are treated as twists and wrenches of infinite pitch, and do not have a definite screw axis. The elements of M^6 and F^6 can be regarded as twists and wrenches, respectively.

Magnitude

Spatial vectors are not Euclidean, and therefore do not have magnitudes in the Euclidean sense. However, there are four sets of spatial vectors for which a meaningful magnitude can be defined: rotations, translations, linear forces and couples (i.e., all the line vectors and free vectors in M^6 and F^6). Magnitudes are only comparable within each set; so the magnitude of a rotation, for example, can be compared with that of another rotation, but not with a translation or a force.

A vector having unit magnitude can be called a unit vector. However, the meaning of 'unit magnitude' does depend on the chosen system of units. For example, a unit rotational velocity is one radian per chosen time unit, and a unit translational velocity is one length unit per time unit. Thus, the Plücker basis vectors, which are all unit vectors, depend not only on the position and velocity of the coordinate frame, but also on the chosen system of units.

2.6 Scalar Product

A scalar product is defined on spatial vectors, in which one argument must be a motion vector, the other must be a force vector, and the result is a scalar representing energy, power or some similar quantity. Given $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$, the scalar product can be written either $\boldsymbol{m} \cdot \boldsymbol{f}$ or $\boldsymbol{f} \cdot \boldsymbol{m}$, both meaning the same, but the expressions $\boldsymbol{m} \cdot \boldsymbol{m}$ and $\boldsymbol{f} \cdot \boldsymbol{f}$ are not defined (i.e., there is no inner product on either M^6 or F^6). If $\boldsymbol{f}$ is the force acting on a rigid body, and $\boldsymbol{m}$ is the velocity of that body, then $\boldsymbol{m} \cdot \boldsymbol{f}$ is the power delivered by $\boldsymbol{f}$.

The scalar product creates a connection between M^6 and F^6 which is formally known as a duality relationship: each space is the dual of the other. The practical consequences of this property for spatial vector algebra are as follows:

1. we use dual coordinates on M^6 and F^6;

2. force and motion vectors obey different coordinate transformation rules; and

3. there are two cross-product operators: one for motion vectors and one for forces (see §2.9).

A dual coordinate system on M^6 and F^6 is any coordinate system formed by two bases, $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_6\} \subset \mathsf{M}^6$ and $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_6\} \subset \mathsf{F}^6$, in which the basis vectors satisfy the condition

$$\boldsymbol{d}_i \cdot \boldsymbol{e}_j = \left\{ \begin{array}{ll} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{array} \right. \tag{2.11}$$

From the definitions of $\mathcal{D}_O$ and $\mathcal{E}_O$ in Eqs. 2.2 and 2.7, it can be seen that these bases form a dual coordinate system. (Treat $\boldsymbol{d}_{Ox}$ as $\boldsymbol{d}_1$, $\boldsymbol{d}_{Oy}$ as $\boldsymbol{d}_2$, and so on.)

The most important property of a dual coordinate system is that if $\underline{\boldsymbol{m}}$ and $\underline{\boldsymbol{f}}$ represent $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$ in a dual coordinate system, then

$$\boldsymbol{m} \cdot \boldsymbol{f} = \underline{\boldsymbol{m}}^{\mathrm{T}} \underline{\boldsymbol{f}} \,. \tag{2.12}$$

An immediate consequence of this property is that we need different coordinate transformation matrices for motions and forces. If the matrix $\boldsymbol{X}$ performs a coordinate transformation on motion vectors, and $\boldsymbol{X}^*$ performs the same transformation on force vectors, then the two are related by

$$\boldsymbol{X}^* = \boldsymbol{X}^{-\mathrm{T}} \,. \tag{2.13}$$

This equation follows from the requirement that $\underline{\boldsymbol{m}}^{\mathrm{T}} \underline{\boldsymbol{f}} = (\boldsymbol{X}\underline{\boldsymbol{m}})^{\mathrm{T}} (\boldsymbol{X}^* \underline{\boldsymbol{f}})$ for all $\underline{\boldsymbol{m}}$ and $\underline{\boldsymbol{f}}$.

2.7   Using Spatial Vectors

Spatial vectors are a tool for expressing and analysing the dynamics of rigid bodies and rigid-body systems. This section presents a list of rules for using spatial vectors, which describe the connections between spatial vectors and the physical phenomena they represent. The last few entries preview some rules for quantities that we have not yet met: acceleration, momentum and inertia. The list is not complete.

usage: The elements of M^6 describe aspects of rigid-body motion, such as: velocity, acceleration, infinitesimal displacement, and directions of motion freedom and constraint. The elements of F^6 describe the forces acting on a rigid body, and quantities related to force such as: momentum, impulse, and directions of force freedom and constraint.

uniqueness: There is a $1:1$ mapping between the elements of M^6 and the set of all possible motions of a rigid body. Likewise, there is a $1:1$ mapping between the elements of F^6 and the set of all possible forces acting on a rigid body (i.e., the set of all wrenches).

relative velocity: If bodies B_1 and B_2 have velocities $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, respectively, then the relative velocity of B_2 with respect to B_1 is $\boldsymbol{v}_{rel} = \boldsymbol{v}_2 - \boldsymbol{v}_1$.

rigid connection: If two bodies are connected together rigidly, then their velocities are the same.

summation of forces: If forces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ both act on the same rigid body, then they are equivalent to a single force, $\boldsymbol{f}_{tot}$, given by $\boldsymbol{f}_{tot} = \boldsymbol{f}_1 + \boldsymbol{f}_2$.

action and reaction: If body B_1 exerts a force of $\boldsymbol{f}$ on body B_2, then B_2 exerts a force of $-\boldsymbol{f}$ on B_1 (Newton's 3rd law).

scalar product: If a force $\boldsymbol{f}$ acts on a rigid body having a velocity $\boldsymbol{v}$, then the power delivered by the force is $\boldsymbol{f} \cdot \boldsymbol{v}$.

scaling: A force of $\alpha\boldsymbol{f}$ acting on a body with velocity $\beta\boldsymbol{v}$ delivers $\alpha\beta$ times as much power as a force of $\boldsymbol{f}$ acting on a body with velocity $\boldsymbol{v}$. A body moving at a constant velocity of $\boldsymbol{v}$ makes the same movement in α units of time as a body moving with velocity $\alpha\boldsymbol{v}$ in one unit of time.

acceleration: Spatial acceleration is the rate of change of spatial velocity. Spatial accelerations are true vectors, and follow the same summation rule as velocities. For example, if $\boldsymbol{v}_{rel} = \boldsymbol{v}_2 - \boldsymbol{v}_1$ then $\boldsymbol{a}_{rel} = \boldsymbol{a}_2 - \boldsymbol{a}_1$.

summation of inertias: If bodies B_1 and B_2 have inertias of $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, and they are connected together to form a single composite rigid body, then the inertia of the composite is $\boldsymbol{I}_1 + \boldsymbol{I}_2$.

momentum: If a rigid body has a spatial inertia of $\boldsymbol{I}$ and a velocity of $\boldsymbol{v}$, then its momentum is $\boldsymbol{I}\boldsymbol{v}$.

equation of motion: The rate of change of a body's momentum equals the force acting on it; that is, $\boldsymbol{f} = \mathrm{d}/\mathrm{d}t(\boldsymbol{I}\boldsymbol{v})$. If we perform the differentiation, then we get the formula $\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}$.
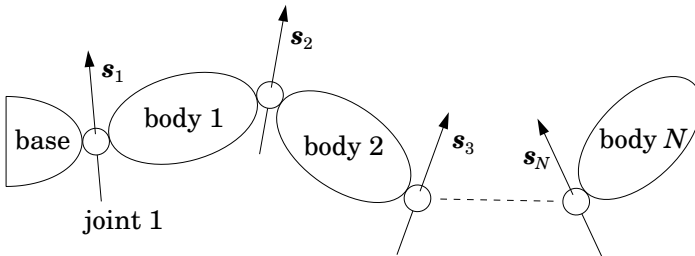
Figure 2.4: Kinematic chain

Example 2.3 *Figure 2.4 shows a collection of rigid bodies joined together in a chain, with one end joined to a fixed base. A system like this is typical of an industrial robot arm, and is called a kinematic chain. There are a total of N moving bodies and N joints in this system, and the bodies are connected together such that joint i connects from body $i - 1$ to body i. (The base is body 0.) We say that a joint connects 'from' one body 'to' another so that we can define the velocity across the joint as being the velocity of the 'to' body relative to the 'from' body. Thus, if $\boldsymbol{v}_{\mathrm{J}i}$ is the velocity across joint i, and $\boldsymbol{v}_i$ is the velocity of body i, then*

$$\boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{v}_i - \boldsymbol{v}_{i-1} \,. \tag{2.14}$$

We shall assume that each joint allows only a single degree of motion freedom, so that $\boldsymbol{v}_{\mathrm{J}i}$ can be described as a scalar multiple of a single motion vector. Specifically,

$$\boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{s}_i \, \dot{q}_i \,, \tag{2.15}$$

where $\boldsymbol{s}_i$ and $\dot{q}_i$ are the axis vector and velocity variable for joint i. If joint i is revolute, then $\boldsymbol{s}_i$ will be a unit rotation vector so that $\boldsymbol{v}_{\mathrm{J}i}$ will be a unit rotational velocity when $\dot{q}_i = 1$. Likewise, if joint i is prismatic then $\boldsymbol{s}_i$ will be a unit translation vector.

Let us work out the velocity of each body in the system. From Eqs. 2.14 and 2.15 we have

$$\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + \boldsymbol{s}_i \, \dot{q}_i \,. \qquad (\boldsymbol{v}_0 = \boldsymbol{0}) \tag{2.16}$$

This equation lets us calculate the velocity of each body in turn, starting at the base and working out towards the free end of the chain. Another expression for the velocity is

$$\boldsymbol{v}_i = \sum_{j=1}^{i} \boldsymbol{s}_j \, \dot{q}_j \,. \tag{2.17}$$

From a computational point of view, Eq. 2.16 is more efficient than Eq. 2.17 because it reuses the results of previous calculations. Yet another expression for $\boldsymbol{v}_i$ is

$$\boldsymbol{v}_i = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{s}_2 & \cdots & \boldsymbol{s}_i & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \vdots \\ \dot{q}_N \end{bmatrix} = \boldsymbol{J}_i \, \dot{\boldsymbol{q}} \,. \tag{2.18}$$

In this equation, $\boldsymbol{J}_i$ is a $6 \times N$ matrix called the body Jacobian for body i, and $\dot{\boldsymbol{q}}$ is the joint-space velocity vector.

2.8   Coordinate Transforms

The coordinate transform from A to B coordinates for a motion vector is usually written ${}^{B}\boldsymbol{X}_A$, and the same transform for a force vector is ${}^{B}\boldsymbol{X}_A^{*}$. The two are related by the equation ${}^{B}\boldsymbol{X}_A^{*} = {}^{B}\boldsymbol{X}_A^{-\mathrm{T}}$ (cf. Eq. 2.13). In this section, we shall develop the formulae for coordinate transforms between Plücker coordinate systems.

Let A and B denote two Plücker coordinate systems. Each is defined by the position and orientation of a Cartesian frame, and each frame also defines a Cartesian coordinate system. For convenience, we shall use the same names for all three entities; so frame A defines both Plücker coordinate system A and Cartesian coordinate system A, and similarly for B. The formula for ${}^{B}\boldsymbol{X}_A$ depends only on the position and orientation of frame B relative to frame A, so it can be expressed as the product of two simpler transforms: one depending on the relative position of frame B and the other on its relative orientation.

Rotation

Let A and B be two Cartesian frames with a common origin at O. Given O, any motion vector $\hat{m} \in M^6$ can be represented by two 3D vectors, m and m_O, and the Plücker coordinates of $\hat{m}$ will be the Cartesian coordinates of m and m_O. Therefore, let $^A\hat{m}$, ^Am, ^Am_O, $^B\hat{m}$, ^Bm and ^Bm_O be the coordinate vectors representing $\hat{m}$, m and m_O in A and B coordinates, respectively, and let E be the 3×3 rotation matrix that transforms 3D vectors from A to B coordinates (i.e., $E = {}^BE_A$). We now have

$$^B\hat{m} = \begin{bmatrix} ^Bm \\ ^Bm_O \end{bmatrix} = \begin{bmatrix} E\,^Am \\ E\,^Am_O \end{bmatrix} = \begin{bmatrix} E & 0 \\ 0 & E \end{bmatrix} {}^A\hat{m} \,,$$

so

$$^BX_A = \begin{bmatrix} E & 0 \\ 0 & E \end{bmatrix} . \tag{2.19}$$

The equivalent transform for a force vector is exactly the same:

$$^BX_A^* = {}^BX_A^{-T} = \begin{bmatrix} E & 0 \\ 0 & E \end{bmatrix} . \tag{2.20}$$

Translation

Let O and P be two points in space, and let there be a Cartesian frame located at each point, the two frames having the same orientation. Any motion vector $\hat{m} \in M^6$ can be expressed at O by the 3D vectors m and m_O, and at P by the vectors m and m_P, where $m_P = m_O - \overrightarrow{OP} \times m$ (as per Eq. 2.1). In both cases, the Plücker coordinates of $\hat{m}$ are the Cartesian coordinates of the appropriate pair of 3D vectors, so we have

$$\hat{m}_P = \begin{bmatrix} m \\ m_P \end{bmatrix} = \begin{bmatrix} m \\ m_O - r \times m \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -r\times & 1 \end{bmatrix} \hat{m}_O \,,$$

where $r = \overrightarrow{OP}$. Therefore

$$^PX_O = \begin{bmatrix} 1 & 0 \\ -r\times & 1 \end{bmatrix} . \tag{2.21}$$

The equivalent transform for a force vector is

$$^PX_O^* = {}^PX_O^{-T} = \begin{bmatrix} 1 & -r\times \\ 0 & 1 \end{bmatrix} . \tag{2.22}$$

An expression of the form $a\times$ (pronounced 'a-cross') denotes an operator that maps b to $a \times b$. If a is a 3D coordinate vector, then $a\times$ is the 3×3 matrix given by the formula

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} . \tag{2.23}$$

$$\boldsymbol{a}\times = -\boldsymbol{a}\times^{\mathrm{T}}$$
$$(\lambda\,\boldsymbol{a})\times = \lambda\,(\boldsymbol{a}\times) \qquad\qquad (\lambda \text{ is a scalar})$$
$$(\boldsymbol{a}+\boldsymbol{b})\times = \boldsymbol{a}\times + \boldsymbol{b}\times$$
$$(\boldsymbol{E}\,\boldsymbol{a})\times = \boldsymbol{E}\,\boldsymbol{a}\times \boldsymbol{E}^{-1} \qquad (\boldsymbol{E} \text{ is orthonormal})$$
$$(\boldsymbol{a}\times)\,\boldsymbol{b} = -(\boldsymbol{b}\times)\,\boldsymbol{a}$$
$$(\boldsymbol{a}\times\boldsymbol{b})\boldsymbol{\cdot} = \boldsymbol{a}\boldsymbol{\cdot}\,\boldsymbol{b}\times \qquad\qquad ((\boldsymbol{a}\times\boldsymbol{b})^{\mathrm{T}} = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}\times)$$
$$(\boldsymbol{a}\times\boldsymbol{b})\times = \boldsymbol{a}\times\boldsymbol{b}\times - \boldsymbol{b}\times\boldsymbol{a}\times$$
$$(\boldsymbol{a}\times\boldsymbol{b})\times = \boldsymbol{b}\,\boldsymbol{a}^{\mathrm{T}} - \boldsymbol{a}\,\boldsymbol{b}^{\mathrm{T}}$$
$$\boldsymbol{a}\times\boldsymbol{b}\times = \boldsymbol{b}\,\boldsymbol{a}^{\mathrm{T}} - (\boldsymbol{a}\boldsymbol{\cdot}\boldsymbol{b})\,\mathbf{1}_{3\times3}$$

Table 2.1: Properties of the Euclidean cross operator

This operator has several mathematical properties, which are listed in Table 2.1. We shall interpret expressions of the form $\boldsymbol{a}\times\boldsymbol{b}\times\boldsymbol{c}$ as meaning $(\boldsymbol{a}\times)(\boldsymbol{b}\times)\boldsymbol{c}$, which is the same as $\boldsymbol{a}\times(\boldsymbol{b}\times\boldsymbol{c})$. Any other interpretation requires parentheses. Likewise, the expressions $\boldsymbol{a}+\boldsymbol{b}\times\boldsymbol{c}$ and $\boldsymbol{E}\,\boldsymbol{a}\times$ mean $\boldsymbol{a}+(\boldsymbol{b}\times\boldsymbol{c})$ and $\boldsymbol{E}(\boldsymbol{a}\times)$, respectively, and any other interpretation requires parentheses.

General Transforms

Let A and B be Cartesian frames with origins at O and P, respectively; let $\boldsymbol{r}$ be the coordinate vector expressing $\overrightarrow{OP}$ in A coordinates; and let $\boldsymbol{E}$ be the rotation matrix that transforms 3D vectors from A to B coordinates. The Plücker transform from A to B coordinates is then the product of a translation by $\boldsymbol{r}$ and a rotation by $\boldsymbol{E}$. The formula for a motion-vector transform is

$$^{B}\boldsymbol{X}_A = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \boldsymbol{0} \\ -\boldsymbol{r}\times & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\,\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix}; \qquad (2.24)$$

the equivalent transform for a force vector is

$$^{B}\boldsymbol{X}_A^* = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \mathbf{1} & -\boldsymbol{r}\times \\ \boldsymbol{0} & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E} & -\boldsymbol{E}\,\boldsymbol{r}\times \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix}; \qquad (2.25)$$

and the inverses of these two transforms are

$$^{A}\boldsymbol{X}_B = \begin{bmatrix} \mathbf{1} & \boldsymbol{0} \\ \boldsymbol{r}\times & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{r}\times\boldsymbol{E}^{\mathrm{T}} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} \qquad (2.26)$$

and

$$^{A}\boldsymbol{X}_B^* = \begin{bmatrix} \mathbf{1} & \boldsymbol{r}\times \\ \boldsymbol{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{r}\times\boldsymbol{E}^{\mathrm{T}} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix}. \qquad (2.27)$$

For comparison, the formula for the 4×4 homogeneous coordinate transformation matrix from A to B (homogeneous) coordinates is

$$^{B}\boldsymbol{T}_A = \begin{bmatrix} \boldsymbol{E} & -\boldsymbol{E}\boldsymbol{r} \\ \boldsymbol{0} & 1 \end{bmatrix}. \qquad (2.28)$$

$$\text{rx}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} \qquad \text{ry}(\theta) = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} \qquad \text{rz}(\theta) = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$(\, c = \cos(\theta)\,, \;\; s = \sin(\theta)\,)$

$$\text{rot}(\boldsymbol{E}) = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \qquad \text{xlt}(\boldsymbol{r}) = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ -\boldsymbol{r}\times & \boldsymbol{1} \end{bmatrix}$$

$$\text{rotx}(\theta) = \text{rot}(\text{rx}(\theta))$$
$$\text{roty}(\theta) = \text{rot}(\text{ry}(\theta))$$
$$\text{rotz}(\theta) = \text{rot}(\text{rz}(\theta))$$

Table 2.2: Functions for rotations and Plücker transforms

To facilitate the description of Plücker transforms, we will occasionally use the functions listed in Table 2.2. In terms of these functions, Eq. 2.24 could have been written $^B\boldsymbol{X}_A = \text{rot}(\boldsymbol{E})\,\text{xlt}(\boldsymbol{r})$, and so on.

Finally, note that we have adopted the following convention for describing coordinate transforms: the formula for $^B\boldsymbol{X}_A$ (or for $^B\boldsymbol{E}_A$) is associated with a description of how you would have to move frame A so as to make it coincide with frame B. For example, the expression 'rotx(θ)' reads like an abbreviated version of the statement 'rotate about the x axis by an amount θ', and its value is the coordinate transform $^B\boldsymbol{X}_A$ for the case where frame B is rotated relative to frame A by an amount θ about their common x axis. Readers should bear in mind that some authors use a different convention.

2.9 Spatial Cross Products

Let $\boldsymbol{r} \in \mathsf{E}^3$ be a Euclidean vector that is rotating with an angular velocity of $\boldsymbol{\omega}$, but is not otherwise changing. The derivative of this vector is given by the well-known formula $\dot{\boldsymbol{r}} = \boldsymbol{\omega} \times \boldsymbol{r}$. In this context, $\boldsymbol{\omega}\times$ is acting like a differentiation operator: it is mapping $\boldsymbol{r}$ to $\dot{\boldsymbol{r}}$.

It turns out that this idea can be extended to spatial vectors, but we need to have two operators: one for motions and one for forces. Therefore, let $\hat{\boldsymbol{m}} \in \mathsf{M}^6$ and $\hat{\boldsymbol{f}} \in \mathsf{F}^6$ be two spatial vectors that are moving with a velocity of $\hat{\boldsymbol{v}} \in \mathsf{M}^6$ (e.g. because they are fixed in a rigid body having that velocity), but are not otherwise changing. We now define two new operators, $\times$ and $\times^*$, which are required to satisfy the following equations:

$$\dot{\hat{\boldsymbol{m}}} = \hat{\boldsymbol{v}} \times \hat{\boldsymbol{m}} \tag{2.29}$$

and

$$\dot{\hat{\boldsymbol{f}}} = \hat{\boldsymbol{v}} \times^* \hat{\boldsymbol{f}}\,. \tag{2.30}$$

The operator $\times^*$ can be regarded as the dual of $\times$. The relationship between these two operators is similar to that between the coordinate transform matrices $\boldsymbol{X}$ and $\boldsymbol{X}^*$.

$\times$	d_{Ox}	d_{Oy}	d_{Oz}	d_x	d_y	d_z
d_{Ox}	0	d_{Oz}	$-d_{Oy}$	0	d_z	$-d_y$
d_{Oy}	$-d_{Oz}$	0	d_{Ox}	$-d_z$	0	d_x
d_{Oz}	d_{Oy}	$-d_{Ox}$	0	d_y	$-d_x$	0
d_x	0	d_z	$-d_y$	0	0	0
d_y	$-d_z$	0	d_x	0	0	0
d_z	d_y	$-d_x$	0	0	0	0

$\times^*$	e_x	e_y	e_z	e_{Ox}	e_{Oy}	e_{Oz}
d_{Ox}	0	e_z	$-e_y$	0	e_{Oz}	$-e_{Oy}$
d_{Oy}	$-e_z$	0	e_x	$-e_{Oz}$	0	e_{Ox}
d_{Oz}	e_y	$-e_x$	0	e_{Oy}	$-e_{Ox}$	0
d_x	0	0	0	0	e_z	$-e_y$
d_y	0	0	0	$-e_z$	0	e_x
d_z	0	0	0	e_y	$-e_x$	0
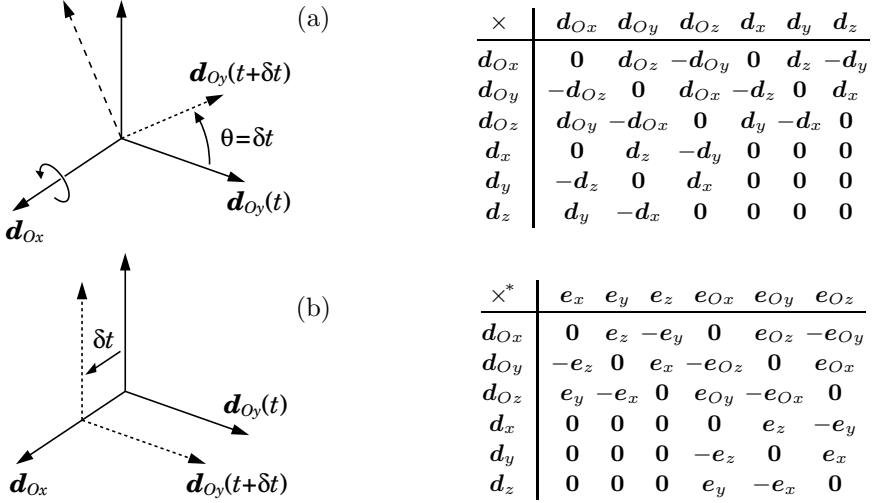
Figure 2.5: Spatial cross-product tables

These operators can be defined explicitly by means of the multiplication tables shown in Figure 2.5. Each entry in these tables is the time derivative of the basis vector listed on the top row, assuming it has a velocity equal to the basis vector listed in the left-hand column. For example, Figure 2.5(a) shows a coordinate frame that is moving with a velocity equal to d_{Ox}, which means that it is rotating with unit angular velocity about the line Ox. Over a period of δt time units, this motion has no effect on the line Ox, but it causes Oy to rotate by δt radians in the y–z plane. So we have $d_{Ox}(t + \delta t) = d_{Ox}(t)$, but $d_{Oy}(t + \delta t) = d_{Oy}(t) + \delta t\, d_{Oz}$. The time derivatives are therefore

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\, d_{Ox} &= \lim_{\delta t \to 0} \frac{d_{Ox}(t + \delta t) - d_{Ox}(t)}{\delta t} \\ &= \lim_{\delta t \to 0} \frac{d_{Ox}(t) - d_{Ox}(t)}{\delta t} = 0 \end{aligned}$$

and

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\, d_{Oy} &= \lim_{\delta t \to 0} \frac{d_{Oy}(t + \delta t) - d_{Oy}(t)}{\delta t} \\ &= \lim_{\delta t \to 0} \frac{d_{Oy}(t) + \delta t\, d_{Oz} - d_{Oy}(t)}{\delta t} = d_{Oz}\,. \end{aligned}$$

This accounts for the first two entries on the first line in the first table. Similarly, Figure 2.5(b) shows a coordinate frame that is moving with a velocity equal to d_x, which means that it is translating with unit linear velocity in the x direction. Over a period of δt time units, this motion has no effect on the line Ox, but it causes Oy to shift by δt length units in the x direction. We therefore have

$$\boldsymbol{v}\times^* = -\boldsymbol{v}\times^{\mathrm{T}}$$

$$(\lambda\,\boldsymbol{v})\times = \lambda\,(\boldsymbol{v}\times) \qquad\qquad (\lambda\,\boldsymbol{v})\times^* = \lambda\,(\boldsymbol{v}\times^*) \qquad\qquad (\lambda \text{ is a scalar})$$

$$(\boldsymbol{u}+\boldsymbol{v})\times = \boldsymbol{u}\times + \boldsymbol{v}\times \qquad (\boldsymbol{u}+\boldsymbol{v})\times^* = \boldsymbol{u}\times^* + \boldsymbol{v}\times^*$$

$$(\boldsymbol{X}\boldsymbol{v})\times = \boldsymbol{X}\,\boldsymbol{v}\times\,\boldsymbol{X}^{-1} \qquad (\boldsymbol{X}\boldsymbol{v})\times^* = \boldsymbol{X}^*\,\boldsymbol{v}\times^*\,(\boldsymbol{X}^*)^{-1} \qquad (\text{Plücker transform})$$

$$\boldsymbol{u}\times\boldsymbol{v} = -\boldsymbol{v}\times\boldsymbol{u}$$

$$(\boldsymbol{u}\times\boldsymbol{v})\cdot = -\boldsymbol{v}\cdot\boldsymbol{u}\times^* \qquad\qquad (\boldsymbol{u}\times^*\boldsymbol{f})\cdot = -\boldsymbol{f}\cdot\boldsymbol{u}\times$$

$$(\boldsymbol{u}\times\boldsymbol{v})\times = \boldsymbol{u}\times\,\boldsymbol{v}\times - \boldsymbol{v}\times\,\boldsymbol{u}\times \qquad (\boldsymbol{u}\times\boldsymbol{v})\times^* = \boldsymbol{u}\times^*\,\boldsymbol{v}\times^* - \boldsymbol{v}\times^*\,\boldsymbol{u}\times^*$$

Table 2.3: Properties of the spatial cross operators

$\boldsymbol{d}_{Ox}(t+\delta t) = \boldsymbol{d}_{Ox}(t)$, implying that $\dot{\boldsymbol{d}}_{Ox} = \boldsymbol{0}$, but $\boldsymbol{d}_{Oy}(t+\delta t) = \boldsymbol{d}_{Oy}(t) + \delta t\,\boldsymbol{d}_z$, implying that $\dot{\boldsymbol{d}}_{Oy} = \boldsymbol{d}_z$. This accounts for the first two entries on the fourth line in the first table. All other entries are calculated in a similar manner.

Using the tables in Figure 2.5, we can deduce that the 6×6 matrices representing the operators $\hat{\boldsymbol{v}}\times$ and $\hat{\boldsymbol{v}}\times^*$ in Plücker coordinates are

$$\hat{\boldsymbol{v}}_O\times = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix}\times = \begin{bmatrix} \boldsymbol{\omega}\times & \boldsymbol{0} \\ \boldsymbol{v}_O\times & \boldsymbol{\omega}\times \end{bmatrix} \tag{2.31}$$

and

$$\hat{\boldsymbol{v}}_O\times^* = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix}\times^* = \begin{bmatrix} \boldsymbol{\omega}\times & \boldsymbol{v}_O\times \\ \boldsymbol{0} & \boldsymbol{\omega}\times \end{bmatrix} = -(\hat{\boldsymbol{v}}_O\times)^{\mathrm{T}}\,, \tag{2.32}$$

and that the spatial cross products of two coordinate vectors are

$$\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} \times \begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{m}_O \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega}\times\boldsymbol{m} \\ \boldsymbol{\omega}\times\boldsymbol{m}_O + \boldsymbol{v}_O\times\boldsymbol{m} \end{bmatrix} \tag{2.33}$$

and

$$\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} \times^* \begin{bmatrix} \boldsymbol{f}_O \\ \boldsymbol{f} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega}\times\boldsymbol{f}_O + \boldsymbol{v}_O\times\boldsymbol{f} \\ \boldsymbol{\omega}\times\boldsymbol{f} \end{bmatrix}\,. \tag{2.34}$$

Various properties of these operators are listed in Table 2.3.

2.10   Differentiation

Vectors are differentiated in the same way as scalars. If $\boldsymbol{v}(x)$ is a differentiable vector function of a scalar variable x, then its derivative is given by the formula

$$\frac{\mathrm{d}}{\mathrm{d}x}\boldsymbol{v}(x) = \lim_{\delta x\to 0} \frac{\boldsymbol{v}(x+\delta x) - \boldsymbol{v}(x)}{\delta x}\,. \tag{2.35}$$

This formula applies to all vectors. In particular, it applies both to abstract vectors and to coordinate vectors. The derivative of a coordinate vector is therefore

$$\frac{\mathrm{d}}{\mathrm{d}x}(\boldsymbol{u} \cdot \boldsymbol{v}) = \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}x} \cdot \boldsymbol{v} + \boldsymbol{u} \cdot \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}x} \qquad\qquad \frac{\mathrm{d}}{\mathrm{d}x}(\boldsymbol{u}\cdot) = (\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}x})\cdot$$

$$\frac{\mathrm{d}}{\mathrm{d}x}(\boldsymbol{u} \times \boldsymbol{v}) = \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}x} \times \boldsymbol{v} + \boldsymbol{u} \times \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}x} \qquad\qquad \frac{\mathrm{d}}{\mathrm{d}x}(\boldsymbol{u}\times) = (\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}x})\times$$

$$\frac{\mathrm{d}}{\mathrm{d}x}(\boldsymbol{u}\,\boldsymbol{v}\cdot) = \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}x}\,\boldsymbol{v}\cdot + \boldsymbol{u}\,(\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}x})\cdot$$

$$\frac{\mathrm{d}}{\mathrm{d}x}(\lambda\,\boldsymbol{v}) = \frac{\mathrm{d}\lambda}{\mathrm{d}x}\,\boldsymbol{v} + \lambda\,\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}x} \qquad\qquad\qquad (\lambda \text{ scalar})$$

$$\frac{\mathrm{d}}{\mathrm{d}x}(\boldsymbol{L}\,\boldsymbol{v}) = \frac{\mathrm{d}\boldsymbol{L}}{\mathrm{d}x}\,\boldsymbol{v} + \boldsymbol{L}\,\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}x} \qquad\qquad (\boldsymbol{L} \text{ matrix or dyadic})$$

Table 2.4: Differentiation formulae for vector expressions

its componentwise derivative, as can be seen from the following equation:

$$\frac{\mathrm{d}}{\mathrm{d}x} \begin{bmatrix} v_1(x) \\ \vdots \\ v_n(x) \end{bmatrix} = \lim_{\delta x \to 0} \frac{1}{\delta x} \begin{bmatrix} v_1(x + \delta x) - v_1(x) \\ \vdots \\ v_n(x + \delta x) - v_n(x) \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}v_1}{\mathrm{d}x} \\ \vdots \\ \frac{\mathrm{d}v_n}{\mathrm{d}x} \end{bmatrix}.$$

The formula also applies to matrices and dyadics. We will be concerned mostly with time derivatives, which we will usually write using dot notation; that is, $\mathrm{d}\boldsymbol{v}/\mathrm{d}t = \dot{\boldsymbol{v}}$, $\mathrm{d}\dot{\boldsymbol{v}}/\mathrm{d}t = \ddot{\boldsymbol{v}}$, and so on. The derivative of a vector is a vector of the same kind; so the derivative of a motion vector is a motion vector, the derivative of a force vector is a force vector, and so on. Differentiation formulae for various vector expressions are shown in Table 2.4.

Differentiation in Moving Coordinates

The problem of differentiation in moving coordinates can be stated as follows: given the coordinate vector that represents a vector $\boldsymbol{v}$, find the coordinate vector that represents $\dot{\boldsymbol{v}}$. If the basis vectors are constants, then the solution is simply the componentwise derivative of the coordinate vector; otherwise, a formula is required that takes into account the motion of the basis vectors.

Let A be the Plücker coordinate system associated with a Cartesian frame that is moving with a spatial velocity of $\boldsymbol{v}_A$. Suppose we have two coordinate vectors, $^A\boldsymbol{m}$ and $^A\boldsymbol{f}$, which represent the spatial vectors $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$, respectively, and we wish to find the coordinate vectors that represent $\dot{\boldsymbol{m}}$ and $\dot{\boldsymbol{f}}$ in A coordinates. These vectors are given by the formulae

$$^A\!\left(\frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}t}\right) = \frac{\mathrm{d}^A\boldsymbol{m}}{\mathrm{d}t} + {}^A\boldsymbol{v}_A \times {}^A\boldsymbol{m} \qquad\qquad (2.36)$$

and

$$^A\!\left(\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}t}\right) = \frac{\mathrm{d}^A\boldsymbol{f}}{\mathrm{d}t} + {}^A\boldsymbol{v}_A \times^* {}^A\boldsymbol{f}\,. \qquad\qquad (2.37)$$

The expressions on the left-hand sides denote the coordinate vectors representing $\dot{\boldsymbol{m}}$ and $\dot{\boldsymbol{f}}$ in A coordinates, and the expressions $\mathrm{d}^A\boldsymbol{m}/\mathrm{d}t$ and $\mathrm{d}^A\boldsymbol{f}/\mathrm{d}t$ denote

the componentwise derivatives of $^A\boldsymbol{m}$ and $^A\boldsymbol{f}$, respectively. $^A\boldsymbol{v}_A$ is the coordinate vector representing $\boldsymbol{v}_A$ in A coordinates. For comparison, the well-known formula for the derivative of a 3D Euclidean vector in rotating coordinates, expressed in the same notation, is

$$^A\!\left(\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t}\right) = \frac{\mathrm{d}^A\boldsymbol{u}}{\mathrm{d}t} + {}^A\boldsymbol{\omega}_A \times {}^A\boldsymbol{u} \,. \tag{2.38}$$

In this equation, $\boldsymbol{u} \in \mathsf{E}^3$ is the vector being differentiated, and $\boldsymbol{\omega}_A$ is the angular velocity of the coordinate frame.

Note: In formulating the dynamics algorithms that appear in subsequent chapters, we generally avoid performing differentiation in moving coordinates by formulating the algorithms in stationary coordinate systems that happen to coincide with the moving ones at the current instant.

Dot and Ring Notation

It is slightly risky to mix dot notation with coordinate vectors because it is not obvious whether a symbol like $^A\dot{\boldsymbol{m}}$ means $^A(\mathrm{d}\boldsymbol{m}/\mathrm{d}t)$ or $\mathrm{d}(^A\boldsymbol{m})/\mathrm{d}t$. Nevertheless, we shall take this risk, and make the following definitions: for any abstract vector $\boldsymbol{v}$, and any coordinate vector $^A\boldsymbol{v}$ representing $\boldsymbol{v}$ in A coordinates,

$$^A\dot{\boldsymbol{v}} = {}^A\!\left(\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\right) \tag{2.39}$$

and

$$^A\mathring{\boldsymbol{v}} = \frac{\mathrm{d}^A\boldsymbol{v}}{\mathrm{d}t} \,. \tag{2.40}$$

With this notation, Eqs. 2.36 and 2.37 can be written

$$^A\dot{\boldsymbol{m}} = {}^A\mathring{\boldsymbol{m}} + {}^A\boldsymbol{v}_A \times {}^A\boldsymbol{m} \tag{2.41}$$

and

$$^A\dot{\boldsymbol{f}} = {}^A\mathring{\boldsymbol{f}} + {}^A\boldsymbol{v}_A \times^* {}^A\boldsymbol{f} \,. \tag{2.42}$$

Furthermore, if the identity of the coordinate system is clear from the context, then we need not specify it explicitly. The equations can then be written

$$\dot{\boldsymbol{m}} = \mathring{\boldsymbol{m}} + \boldsymbol{v}_A \times \boldsymbol{m} \tag{2.43}$$

and

$$\dot{\boldsymbol{f}} = \mathring{\boldsymbol{f}} + \boldsymbol{v}_A \times^* \boldsymbol{f} \,. \tag{2.44}$$

The symbols $\mathring{\boldsymbol{m}}$ and $\mathring{\boldsymbol{f}}$ can be regarded as the apparent rates of change of $\boldsymbol{m}$ and $\boldsymbol{f}$ as viewed by an observer having a velocity of $\boldsymbol{v}_A$.

Example 2.4 *Let A and B be two Cartesian frames having velocities of $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, respectively, and let $^B\boldsymbol{X}_A$ be the Plücker coordinate transform from A to B coordinates. We can work out the time derivative of $^B\boldsymbol{X}_A$ using these four equations, which follow from Eqs. 2.39 to 2.41:*

$$
\begin{aligned}
{}^B\dot{\boldsymbol{m}} &= {}^B\boldsymbol{X}_A \, {}^A\dot{\boldsymbol{m}} \\
{}^B\mathring{\boldsymbol{m}} &= \frac{\mathrm{d}}{\mathrm{d}t}\left({}^B\boldsymbol{X}_A \, {}^A\boldsymbol{m}\right) = \left(\frac{\mathrm{d}}{\mathrm{d}t}\,{}^B\boldsymbol{X}_A\right){}^A\boldsymbol{m} + {}^B\boldsymbol{X}_A \, {}^A\mathring{\boldsymbol{m}} \\
{}^A\dot{\boldsymbol{m}} &= {}^A\mathring{\boldsymbol{m}} + {}^A\boldsymbol{v}_A \times {}^A\boldsymbol{m} \\
{}^B\dot{\boldsymbol{m}} &= {}^B\mathring{\boldsymbol{m}} + {}^B\boldsymbol{v}_B \times {}^B\boldsymbol{m} \,.
\end{aligned}
$$

The derivation proceeds as follows:

$$
\begin{aligned}
\left(\frac{\mathrm{d}}{\mathrm{d}t}\,{}^B\boldsymbol{X}_A\right){}^A\boldsymbol{m} &= {}^B\mathring{\boldsymbol{m}} - {}^B\boldsymbol{X}_A \, {}^A\mathring{\boldsymbol{m}} \\
&= {}^B\dot{\boldsymbol{m}} - {}^B\boldsymbol{v}_B \times {}^B\boldsymbol{m} - {}^B\boldsymbol{X}_A({}^A\dot{\boldsymbol{m}} - {}^A\boldsymbol{v}_A \times {}^A\boldsymbol{m}) \\
&= {}^B\dot{\boldsymbol{m}} - {}^B\boldsymbol{X}_A \, {}^A\dot{\boldsymbol{m}} - {}^B\boldsymbol{v}_B \times {}^B\boldsymbol{X}_A \, {}^A\boldsymbol{m} + {}^B\boldsymbol{X}_A \, {}^A\boldsymbol{v}_A \times {}^A\boldsymbol{m} \\
&= ({}^B\boldsymbol{X}_A \, {}^A\boldsymbol{v}_A \times - \, {}^B\boldsymbol{v}_B \times {}^B\boldsymbol{X}_A)\,{}^A\boldsymbol{m} \,.
\end{aligned}
$$

As this must be true for all $^A\boldsymbol{m}$, we have

$$
\frac{\mathrm{d}}{\mathrm{d}t}\,{}^B\boldsymbol{X}_A = {}^B(\boldsymbol{v}_A - \boldsymbol{v}_B)\times \,{}^B\boldsymbol{X}_A \,. \tag{2.45}
$$

2.11   Acceleration

We define the *spatial acceleration* of a rigid body to be the time derivative of its spatial velocity. Thus, if a body has a spatial velocity of $[\boldsymbol{\omega}^{\mathrm{T}} \; \boldsymbol{v}_O^{\mathrm{T}}]^{\mathrm{T}}$ expressed at O, then its spatial acceleration is

$$
\hat{\boldsymbol{a}}_O = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\boldsymbol{\omega}\\\boldsymbol{v}_O\end{bmatrix} = \lim_{\delta t \to 0}\frac{1}{\delta t}\begin{bmatrix}\boldsymbol{\omega}(t+\delta t) - \boldsymbol{\omega}(t)\\\boldsymbol{v}_O(t+\delta t) - \boldsymbol{v}_O(t)\end{bmatrix} = \begin{bmatrix}\dot{\boldsymbol{\omega}}\\\dot{\boldsymbol{v}}_O\end{bmatrix}. \tag{2.46}
$$

This definition may sound obvious, but it contains a surprise for those who are already expert in the traditional method of describing rigid-body acceleration. Consider the body shown in Figure 2.6, which is rotating at a constant angular velocity about a fixed line in space. This body has a constant spatial velocity, and must therefore have zero spatial acceleration; yet almost every point in the body is travelling in a circular path, and is therefore accelerating. This paradox is easily resolved. First, one must realise that spatial acceleration is a property of the body as a whole, rather than a property of individual body-fixed points. Second, one must realise that $\boldsymbol{v}_O$ is not the velocity of one particular body-fixed point, but a measure of the flow of points through O. Thus, $\dot{\boldsymbol{v}}_O$ is not the acceleration of an individual point, but a measure of the rate of change of flow. If the flow is constant, then $\dot{\boldsymbol{v}}_O$ will be zero even if each individual point is accelerating.
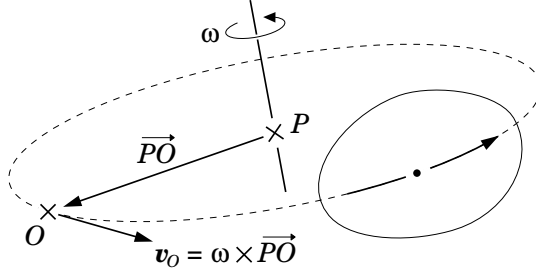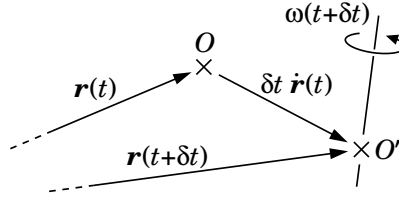
Figure 2.6: Acceleration of a uniformly rotating body

Figure 2.7: Computing $\dot{\boldsymbol{v}}_O$ from $\ddot{\boldsymbol{r}}$

Let us compare spatial acceleration with the traditional method of describing acceleration. To this end, we introduce two more quantities: a second point, O', which is the body-fixed point that coincides with O at time t, and a vector, $\boldsymbol{r}$, which gives the position of O' as a function of time. Thus, $\dot{\boldsymbol{r}}$ is the velocity of O' and $\ddot{\boldsymbol{r}}$ is its acceleration. In the standard textbooks on classical mechanics, the acceleration of a rigid body would be specified by the two 3D vectors $\dot{\boldsymbol{\omega}}$ and $\ddot{\boldsymbol{r}}$; so let us examine how $\dot{\boldsymbol{v}}_O$ differs from $\ddot{\boldsymbol{r}}$. From Eq. 2.46 we have

$$\dot{\boldsymbol{v}}_O = \lim_{\delta t \to 0} \frac{\boldsymbol{v}_O(t + \delta t) - \boldsymbol{v}_O(t)}{\delta t}\,.$$

Now, $\boldsymbol{v}_O(t) = \dot{\boldsymbol{r}}(t)$ because O' coincides with O at time t, but $\boldsymbol{v}_O(t + \delta t)$ is not equal to $\dot{\boldsymbol{r}}(t + \delta t)$ because O' has moved away from O by an amount $\delta t\, \dot{\boldsymbol{r}}(t)$ (see Figure 2.7), so $\boldsymbol{v}_O(t + \delta t)$ is given instead by

$$\boldsymbol{v}_O(t + \delta t) = \dot{\boldsymbol{r}}(t + \delta t) - \boldsymbol{\omega} \times \delta t\, \dot{\boldsymbol{r}}(t)\,.$$

Combining these equations gives

$$\begin{aligned}
\dot{\boldsymbol{v}}_O &= \lim_{\delta t \to 0} \frac{\dot{\boldsymbol{r}}(t + \delta t) - \dot{\boldsymbol{r}}(t) - \boldsymbol{\omega} \times \delta t\, \dot{\boldsymbol{r}}(t)}{\delta t} \\
&= \ddot{\boldsymbol{r}} - \boldsymbol{\omega} \times \dot{\boldsymbol{r}}\,,
\end{aligned} \tag{2.47}$$

which implies that

$$\hat{\boldsymbol{a}}_O = \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\boldsymbol{r}} - \boldsymbol{\omega} \times \dot{\boldsymbol{r}} \end{bmatrix}\,. \tag{2.48}$$

For comparison, we now define a new 6D vector,

$$\hat{\boldsymbol{a}}_O' = \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\boldsymbol{r}} \end{bmatrix}, \tag{2.49}$$

which we shall call the *classical acceleration* of a rigid body, expressed at O. The name reflects the fact that this vector is composed of the two 3D vectors that are used in the classical description of rigid-body acceleration. Physically, the classical acceleration is the apparent derivative of spatial velocity in a Plücker coordinate system having its origin at O', rather than O, meaning that the coordinate frame has a pure linear velocity of $\dot{\boldsymbol{r}}$. Thus, according to Eq. 2.43, $\hat{\boldsymbol{a}}_O$ and $\hat{\boldsymbol{a}}_O'$ are related by

$$\hat{\boldsymbol{a}}_O = \hat{\boldsymbol{a}}_O' + \begin{bmatrix} \boldsymbol{0} \\ \dot{\boldsymbol{r}} \end{bmatrix} \times \hat{\boldsymbol{v}}_O, \tag{2.50}$$

which is easily verified from Eqs. 2.48 and 2.49.

The big advantage of spatial acceleration is that it is a true vector, and it obeys the same combination and coordinate transformation rules as velocity. For example, if $\hat{\boldsymbol{v}}_1$ and $\hat{\boldsymbol{v}}_2$ denote the spatial velocities of bodies B_1 and B_2, respectively, and $\hat{\boldsymbol{v}}_{rel}$ is the relative velocity of B_2 with respect to B_1, then we already know that

$$\hat{\boldsymbol{v}}_{rel} = \hat{\boldsymbol{v}}_2 - \hat{\boldsymbol{v}}_1\,.$$

The equivalent formula for spatial accelerations is obtained immediately by differentiating the velocity formula:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{v}}_{rel} = \frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{v}}_2 - \frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{v}}_1 \quad \Rightarrow \quad \hat{\boldsymbol{a}}_{rel} = \hat{\boldsymbol{a}}_2 - \hat{\boldsymbol{a}}_1\,. \tag{2.51}$$

Thus, spatial accelerations are composed simply by adding them together, just like velocities. This compares favourably with the traditional formulae for composing accelerations, which involve the use of Coriolis terms.

Example 2.5 *Example 2.3 presented several formulae for the velocities of the bodies in a kinematic chain. Let us now obtain the corresponding formulae for their accelerations. First, we define $\boldsymbol{a}_{\mathrm{J}i}$ to be the acceleration across joint i and $\boldsymbol{a}_i$ to be the acceleration of body i; so $\boldsymbol{a}_{\mathrm{J}i} = \dot{\boldsymbol{v}}_{\mathrm{J}i}$ and $\boldsymbol{a}_i = \dot{\boldsymbol{v}}_i$. As spatial accelerations are additive, we immediately have*

$$\boldsymbol{a}_{\mathrm{J}i} = \boldsymbol{a}_i - \boldsymbol{a}_{i-1}\,, \tag{2.52}$$

which is the time derivative of Eq. 2.14. Next, the derivative of Eq. 2.15 is

$$\boldsymbol{a}_{\mathrm{J}i} = \boldsymbol{s}_i\,\ddot{q}_i + \dot{\boldsymbol{s}}_i\,\dot{q}_i\,, \tag{2.53}$$

where $\ddot{q}_i$ is the acceleration variable for joint i. Given that $\boldsymbol{s}_i$ represents a joint motion axis that is fixed in body i (and also in body $i-1$), we have

$$\dot{\boldsymbol{s}}_i = \boldsymbol{v}_i \times \boldsymbol{s}_i\,. \tag{2.54}$$

Combining these three equations gives

$$\boldsymbol{a}_i = \boldsymbol{a}_{i-1} + \boldsymbol{s}_i\,\ddot{q}_i + \boldsymbol{v}_i \times \boldsymbol{s}_i\,\dot{q}_i\,, \tag{2.55}$$

which is the derivative of Eq. 2.16. Next, the derivative of Eq. 2.17 is

$$\begin{aligned}
\boldsymbol{a}_i &= \sum_{j=1}^{i} \boldsymbol{s}_j\,\ddot{q}_j + \boldsymbol{v}_j \times \boldsymbol{s}_j\,\dot{q}_j \\
&= \sum_{j=1}^{i} \boldsymbol{s}_j\,\ddot{q}_j + \sum_{j=1}^{i}\sum_{k=1}^{j-1} \boldsymbol{s}_k \times \boldsymbol{s}_j\,\dot{q}_j\,\dot{q}_k\,.
\end{aligned} \tag{2.56}$$

Finally, the derivative of Eq. 2.18 is

$$\boldsymbol{a}_i = \boldsymbol{J}_i\,\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i\,\dot{\boldsymbol{q}}\,, \tag{2.57}$$

where $\ddot{\boldsymbol{q}}$ is the joint-space acceleration vector and $\dot{\boldsymbol{J}}_i = [\dot{\boldsymbol{s}}_1\ \dot{\boldsymbol{s}}_2\ \cdots\ \dot{\boldsymbol{s}}_i\ \boldsymbol{0}\ \cdots\ \boldsymbol{0}]$.

2.12 Momentum

Figure 2.8 shows a moving rigid body whose centre of mass coincides with the fixed point C at the current instant. This body has a spatial velocity of $\hat{\boldsymbol{v}}$, which is expressed at C by the two 3D vectors $\boldsymbol{\omega}$ and $\boldsymbol{v}_C$. If the body has a mass of m, and a rotational inertia of $\bar{\boldsymbol{I}}_C$ about its centre of mass, then its momentum is described by the two 3D vectors $\boldsymbol{h} = m\,\boldsymbol{v}_C$ and $\boldsymbol{h}_C = \bar{\boldsymbol{I}}_C\,\boldsymbol{\omega}$, where $\boldsymbol{h}$ specifies the magnitude and direction of the body's linear momentum, and $\boldsymbol{h}_C$ specifies the body's intrinsic angular momentum. Linear momentum is actually a line vector (like linear force), and its line of action passes through the body's centre of mass. The body's moment of momentum about any given point, O, is the sum of its intrinsic angular momentum and the moment of its linear momentum about the given point:

$$\boldsymbol{h}_O = \boldsymbol{h}_C + \overrightarrow{OC} \times \boldsymbol{h}\,, \tag{2.58}$$
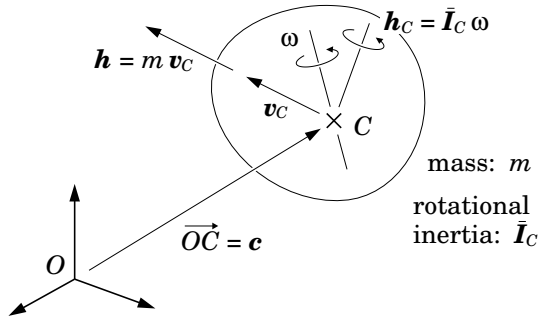

Figure 2.8: Spatial momentum

which is essentially the same formula as Eq. 2.6.

Let $\hat{\boldsymbol{h}}$ denote the spatial momentum of this rigid body, and let $\hat{\boldsymbol{h}}_C$ and $\hat{\boldsymbol{h}}_O$ be the coordinate vectors that represent $\hat{\boldsymbol{h}}$ in Plücker coordinate systems with origins at C and O, respectively. These coordinate vectors are given by

$$\hat{\boldsymbol{h}}_C = \begin{bmatrix} \boldsymbol{h}_C \\ \boldsymbol{h} \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{I}}_C\,\boldsymbol{\omega} \\ m\,\boldsymbol{v}_C \end{bmatrix} \tag{2.59}$$

and

$$\hat{\boldsymbol{h}}_O = \begin{bmatrix} \boldsymbol{h}_O \\ \boldsymbol{h} \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{I}}_C\,\boldsymbol{\omega} + \overrightarrow{OC} \times m\,\boldsymbol{v}_C \\ m\,\boldsymbol{v}_C \end{bmatrix} = \begin{bmatrix} 1 & \overrightarrow{OC}\times \\ 0 & 1 \end{bmatrix} \hat{\boldsymbol{h}}_C\,. \tag{2.60}$$

As spatial momentum vectors are elements of F^6, they have the same algebraic properties as other spatial force vectors. In particular, they transform according to the formula ${}^B\hat{\boldsymbol{h}} = {}^B\boldsymbol{X}_A^*\,{}^A\hat{\boldsymbol{h}}$, and a scalar product is defined between momenta and elements of M^6. The kinetic energy of the body in Figure 2.8 is $\frac{1}{2}\,\hat{\boldsymbol{h}} \cdot \hat{\boldsymbol{v}}$.

2.13   Inertia

The spatial inertia tensor of a rigid body defines the relationship between its velocity and momentum. Spatial inertia is therefore a mapping from M^6 to F^6. If a rigid body has a spatial velocity of $\boldsymbol{v}$ and a spatial inertia of $\boldsymbol{I}$, then its momentum is given by the formula

$$\boldsymbol{h} = \boldsymbol{I}\,\boldsymbol{v}\,. \tag{2.61}$$

If $\boldsymbol{h}$ and $\boldsymbol{v}$ are coordinate vectors, then $\boldsymbol{I}$ is a 6×6 matrix.

Let $\boldsymbol{I}_O$ and $\boldsymbol{I}_C$ be the matrices representing the spatial inertia of a given rigid body in Plücker coordinate systems having origins at O and C, respectively, where O is an arbitrary point and C coincides with the body's centre of mass. From Eq. 2.59 we have

$$\boldsymbol{h}_C = \begin{bmatrix} \bar{\boldsymbol{I}}_C & \mathbf{0} \\ \mathbf{0} & m\,\mathbf{1} \end{bmatrix} \boldsymbol{v}_C\,,$$

which implies that

$$\boldsymbol{I}_C = \begin{bmatrix} \bar{\boldsymbol{I}}_C & \mathbf{0} \\ \mathbf{0} & m\,\mathbf{1} \end{bmatrix}\,. \tag{2.62}$$

The inertia matrix will take this special form whenever the origin coincides with the centre of mass. From Eq. 2.60 we have

$$\begin{aligned} \boldsymbol{h}_O &= \begin{bmatrix} 1 & \boldsymbol{c}\times \\ 0 & 1 \end{bmatrix} \boldsymbol{I}_C\,\boldsymbol{v}_C \\ &= \begin{bmatrix} 1 & \boldsymbol{c}\times \\ 0 & 1 \end{bmatrix} \boldsymbol{I}_C \begin{bmatrix} 1 & 0 \\ \boldsymbol{c}\times^{\mathrm{T}} & 1 \end{bmatrix} \boldsymbol{v}_O\,, \end{aligned}$$

where $\boldsymbol{c} = \overrightarrow{OC}$, which implies that

$$\boldsymbol{I}_O = \begin{bmatrix} \bar{\boldsymbol{I}}_C + m\,\boldsymbol{c}\times\boldsymbol{c}\times^{\mathrm{T}} & m\,\boldsymbol{c}\times \\ m\,\boldsymbol{c}\times^{\mathrm{T}} & m\,\boldsymbol{1} \end{bmatrix}. \tag{2.63}$$

This is the general form of the inertia matrix in Plücker coordinates. The quantity $\bar{\boldsymbol{I}}_C + m\,\boldsymbol{c}\times\boldsymbol{c}\times^{\mathrm{T}}$ is the rotational inertia of the rigid body about O. As $\bar{\boldsymbol{I}}_C$ is a symmetric matrix, it follows that both $\boldsymbol{I}_C$ and $\boldsymbol{I}_O$ are symmetric. Furthermore, if $m > 0$ and $\bar{\boldsymbol{I}}_C$ is positive definite, then both $\boldsymbol{I}_C$ and $\boldsymbol{I}_O$ are positive definite.

Dyadic Representation of Inertia

An expression of the form $\boldsymbol{f}\boldsymbol{g}\cdot$, where $\boldsymbol{f}, \boldsymbol{g} \in \mathsf{F}^6$, is a dyad that maps motion vectors to force vectors. Specifically, it maps a motion vector $\boldsymbol{m} \in \mathsf{M}^6$ to the force vector $\boldsymbol{f}(\boldsymbol{g} \cdot \boldsymbol{m})$, which is a scalar multiple of $\boldsymbol{f}$. This dyad can be represented in dual coordinates by a 6×6 matrix of the form $\underline{\boldsymbol{f}}\,\underline{\boldsymbol{g}}^{\mathrm{T}}$, where $\underline{\boldsymbol{f}}$ and $\underline{\boldsymbol{g}}$ are the coordinate vectors representing $\boldsymbol{f}$ and $\boldsymbol{g}$ in the chosen coordinate system. If $\boldsymbol{f} = \boldsymbol{g}$ then the dyad is said to be symmetric, and its matrix representation is also symmetric.

Spatial inertia is a symmetric dyadic tensor that maps from M^6 to F^6. It can therefore be expressed as the sum of six symmetric dyads as follows:

$$\boldsymbol{I} = \sum_{i=1}^{6} \boldsymbol{g}_i\,\boldsymbol{g}_i\cdot \qquad (\boldsymbol{g}_i \in \mathsf{F}^6). \tag{2.64}$$

Furthermore, a body's spatial inertia is a quantity that is fixed in the body: it does not vary except as a function of the body's motion. It is therefore possible to express $\boldsymbol{I}$ as a sum of dyads in which the individual vectors $\boldsymbol{g}_i$ are all fixed in the body to which they refer. Given this expression, we can calculate the time derivative of a spatial inertia as follows:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{I} &= \sum_{i=1}^{6} (\dot{\boldsymbol{g}}_i\,\boldsymbol{g}_i\cdot + \boldsymbol{g}_i\,\dot{\boldsymbol{g}}_i\cdot) \\
&= \sum_{i=1}^{6} (\boldsymbol{v}\times^*\boldsymbol{g}_i)\,\boldsymbol{g}_i\cdot + \sum_{i=1}^{6}\boldsymbol{g}_i\,(\boldsymbol{v}\times^*\boldsymbol{g}_i)\cdot \\
&= \boldsymbol{v}\times^*\sum_{i=1}^{6}\boldsymbol{g}_i\,\boldsymbol{g}_i\cdot - \left(\sum_{i=1}^{6}\boldsymbol{g}_i\,\boldsymbol{g}_i\cdot\right)\boldsymbol{v}\times \\
&= \boldsymbol{v}\times^*\boldsymbol{I} - \boldsymbol{I}\,\boldsymbol{v}\times, \tag{2.65}
\end{aligned}$$

where $\boldsymbol{v}$ is the body's velocity. In obtaining this result, we have used Eq. 2.30 and formulae from Tables 2.3 and 2.4.

mapping	e.g.	dyad form	coordinate transform formula	type
$\mathsf{M}^n \mapsto \mathsf{M}^n$	$\boldsymbol{v}\times$	$\boldsymbol{m}\,\boldsymbol{f}\cdot$	${}^B\boldsymbol{X}_A\,(\cdots)\,{}^A\boldsymbol{X}_B$	similarity
$\mathsf{F}^n \mapsto \mathsf{F}^n$	$\boldsymbol{v}\times^*$	$\boldsymbol{f}\,\boldsymbol{m}\cdot$	${}^B\boldsymbol{X}_A^*\,(\cdots)\,{}^A\boldsymbol{X}_B^*$	similarity
$\mathsf{M}^n \mapsto \mathsf{F}^n$	$\boldsymbol{I}$	$\boldsymbol{f}\,\boldsymbol{f}\cdot$	${}^B\boldsymbol{X}_A^*\,(\cdots)\,{}^A\boldsymbol{X}_B$	congruence
$\mathsf{F}^n \mapsto \mathsf{M}^n$	$\boldsymbol{\Phi}$	$\boldsymbol{m}\,\boldsymbol{m}\cdot$	${}^B\boldsymbol{X}_A\,(\cdots)\,{}^A\boldsymbol{X}_B^*$	congruence

Table 2.5: Properties of dyadics

Another property that can be deduced from Eq. 2.64 is the coordinate transformation rule for spatial inertias. Let ${}^A\boldsymbol{I}$ and ${}^B\boldsymbol{I}$ be the matrices representing a given spatial inertia in A and B coordinates, respectively. We therefore have

$$ {}^A\boldsymbol{I} = \sum_{i=1}^{6} {}^A\boldsymbol{g}_i\,{}^A\boldsymbol{g}_i^{\mathrm{T}} \qquad \text{and} \qquad {}^B\boldsymbol{I} = \sum_{i=1}^{6} {}^B\boldsymbol{g}_i\,{}^B\boldsymbol{g}_i^{\mathrm{T}} \,, $$

where ${}^A\boldsymbol{g}_i$ and ${}^B\boldsymbol{g}_i$ are the coordinate vectors representing $\boldsymbol{g}_i$ in A and B coordinates, respectively. Now, ${}^B\boldsymbol{g}_i = {}^B\boldsymbol{X}_A^*\,{}^A\boldsymbol{g}_i$, so we have

$$ \begin{aligned} {}^B\boldsymbol{I} &= \sum_{i=1}^{6} {}^B\boldsymbol{X}_A^*\,{}^A\boldsymbol{g}_i\,{}^A\boldsymbol{g}_i^{\mathrm{T}}\,({}^B\boldsymbol{X}_A^*)^{\mathrm{T}} \\ &= {}^B\boldsymbol{X}_A^*\,{}^A\boldsymbol{I}\,{}^A\boldsymbol{X}_B \,. \end{aligned} \tag{2.66} $$

Spatial Dyadics

Spatial inertia is a mapping from M^6 to F^6; but there are three more mappings we can define involving these two spaces, making a total of four kinds of spatial dyadic tensor. Some of their basic properties are listed in Table 2.5. The symbol $\boldsymbol{\Phi}$ in the examples column denotes an inverse inertia (i.e., $\boldsymbol{\Phi} = \boldsymbol{I}^{-1}$). The column marked 'dyad form' shows the pattern of motion and force vectors in the dyads for each kind of tensor. If we think of a dyadic as mapping from the 'From' space to the 'To' space, then the first vector in the dyad must be an element of the 'To' space, and the second vector must be an element in the space that is dual to the 'From' space. The coordinate transformation formulae are each labelled as either a similarity transform or a congruence transform. A similarity transform preserves properties such as rank and eigenvalues, but not symmetry or positive definiteness. A congruence transform preserves properties like rank, symmetry and positive definiteness, but not eigenvalues.

Properties of Spatial Inertia

We have already covered coordinate transforms (Eq. 2.66), the time derivative (Eq. 2.65), the general form of a spatial inertia in Plücker coordinates (Eq. 2.63)

and the fact that inertia maps velocity to momentum (Eq. 2.61). Here are a few more properties.

- Ten parameters are required to define a spatial rigid-body inertia. More general kinds of inertia, such as the articulated-body inertias in Chapter 7, require up to 21 parameters, which is the maximum number of independent values in a symmetric 6×6 matrix.

- The inertia of a rigid body that is composed of multiple parts is the sum of the inertias of its parts.

- The kinetic energy of rigid body having a spatial velocity of $\boldsymbol{v}$ and a spatial inertia of $\boldsymbol{I}$ is

$$T = \frac{1}{2}\,\boldsymbol{v} \cdot \boldsymbol{I}\,\boldsymbol{v}\,. \tag{2.67}$$

2.14   Equation of Motion

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

$$\boldsymbol{f} = \frac{\mathrm{d}}{\mathrm{d}t}\,(\boldsymbol{I}\boldsymbol{v}) \; = \; \boldsymbol{I}\boldsymbol{a} + (\boldsymbol{v}\times^{*}\boldsymbol{I} - \boldsymbol{I}\boldsymbol{v}\times)\,\boldsymbol{v}$$
$$= \; \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v}\times^{*}\boldsymbol{I}\boldsymbol{v}\,. \tag{2.68}$$

We will often write the equation of motion in the form of an inhomogeneous linear equation

$$\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p}\,, \tag{2.69}$$

in which $\boldsymbol{I}$ and $\boldsymbol{p}$ are the coefficients and $\boldsymbol{f}$ and $\boldsymbol{a}$ are the variables. In typical use, the coefficients are assumed to be known, and one or both of the variables are unknown. $\boldsymbol{p}$ is called the bias force, and it is simply the value that $\boldsymbol{f}$ must take in order to produce zero acceleration. If $\boldsymbol{f}$ is the net force acting on the body, then $\boldsymbol{p} = \boldsymbol{v}\times^{*}\boldsymbol{I}\boldsymbol{v}$.

The $\boldsymbol{f}$ in Eq. 2.68 is always the net force acting on the body, but Eq. 2.69 affords the possibility to transfer some known components of the net force from $\boldsymbol{f}$ to $\boldsymbol{p}$. For example, suppose the net force, $\boldsymbol{f}_{net}$, is the sum of an unknown force, $\boldsymbol{f}_{u}$, and a known gravitational force, $\boldsymbol{f}_{g}$. Equation 2.68 would then be

$$\boldsymbol{f}_{net} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v}\times^{*}\boldsymbol{I}\boldsymbol{v}\,;$$

but we have the option of choosing $\boldsymbol{f}_{u}$ as the left-hand quantity in Eq. 2.69, in which case the equation would be

$$\boldsymbol{f}_{u} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p}\,,$$

and $\boldsymbol{p}$ would be defined as

$$\boldsymbol{p} = \boldsymbol{v}\times^{*}\boldsymbol{I}\boldsymbol{v} - \boldsymbol{f}_{g}\,.$$

Example 2.6 *The traditional 3D equations of motion can be recovered from Eq. 2.68 by expanding the spatial quantities into their 3D components. To keep the equations simple, we place the origin at the centre of mass. Equation 2.68 then expands to*

$$
\begin{bmatrix} \boldsymbol{n}_C \\ \boldsymbol{f} \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{I}}_C & \boldsymbol{0} \\ \boldsymbol{0} & m\,\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\boldsymbol{c}} - \boldsymbol{\omega} \times \boldsymbol{v}_C \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega}\times & \boldsymbol{v}_C\times \\ \boldsymbol{0} & \boldsymbol{\omega}\times \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{I}}_C & \boldsymbol{0} \\ \boldsymbol{0} & m\,\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_C \end{bmatrix}
$$

$$
= \begin{bmatrix} \bar{\boldsymbol{I}}_C\,\dot{\boldsymbol{\omega}} \\ m\,\ddot{\boldsymbol{c}} - m\,\boldsymbol{\omega} \times \boldsymbol{v}_C \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega} \times \bar{\boldsymbol{I}}_C\,\boldsymbol{\omega} + m\,\boldsymbol{v}_C \times \boldsymbol{v}_C \\ m\,\boldsymbol{\omega} \times \boldsymbol{v}_C \end{bmatrix}
$$

$$
= \begin{bmatrix} \bar{\boldsymbol{I}}_C\,\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \bar{\boldsymbol{I}}_C\,\boldsymbol{\omega} \\ m\,\ddot{\boldsymbol{c}} \end{bmatrix}.
$$

(We have used Eq. 2.48 to express the body's spatial acceleration in terms of $\dot{\boldsymbol{\omega}}$, $\ddot{\boldsymbol{c}}$ and $\boldsymbol{v}_C\ (= \dot{\boldsymbol{c}})$.) Thus, we have recovered Newton's equation for the motion of the centre of mass,

$$
\boldsymbol{f} = m\,\ddot{\boldsymbol{c}}\,, \tag{2.70}
$$

and also Euler's equation for the rotational motion of the body about its centre of mass,

$$
\boldsymbol{n}_C = \bar{\boldsymbol{I}}_C\,\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \bar{\boldsymbol{I}}_C\,\boldsymbol{\omega}\,. \tag{2.71}
$$

2.15 Inverse Inertia

It is sometimes useful to write the equation of motion in the form

$$
\boldsymbol{a} = \boldsymbol{\Phi}\boldsymbol{f} + \boldsymbol{b}\,, \tag{2.72}
$$

where $\boldsymbol{\Phi}$ is the body's inverse inertia and $\boldsymbol{b}$ is its bias acceleration, which is the acceleration that the body would have if $\boldsymbol{f} = \boldsymbol{0}$. If this equation is the inverse of Eq. 2.69, then $\boldsymbol{\Phi} = \boldsymbol{I}^{-1}$ and $\boldsymbol{b} = -\boldsymbol{\Phi}\boldsymbol{p}$. The advantage of Eq. 2.72 over Eq. 2.69 is that it can be applied to a constrained rigid body.

Inverse inertia is a symmetric dyadic tensor that maps from F^6 to M^6. Some of its properties are already listed in Table 2.5. Another useful property is that the range of an inverse inertia is the subspace of M^6 within which the body it applies to is free to move. This, in turn, implies that the rank of an inverse inertia equals the degree of motion freedom of the body to which it applies. Thus, if the body has fewer than six degrees of freedom, then $\boldsymbol{\Phi}$ will be singular, but $\boldsymbol{I}$ will not exist. The formulae for the inverse inertia of an unconstrained rigid body, expressed in Plücker coordinates at C (the centre of mass) and O (a general point) are

$$
\boldsymbol{\Phi}_C = \begin{bmatrix} \bar{\boldsymbol{I}}_C^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \frac{1}{m}\boldsymbol{1} \end{bmatrix} \tag{2.73}
$$

and

$$
\boldsymbol{\Phi}_O = \begin{bmatrix} \bar{\boldsymbol{I}}_C^{-1} & \bar{\boldsymbol{I}}_C^{-1}\boldsymbol{c}\times^{\mathrm{T}} \\ \boldsymbol{c}\times \bar{\boldsymbol{I}}_C^{-1} & \frac{1}{m}\boldsymbol{1} + \boldsymbol{c}\times \bar{\boldsymbol{I}}_C^{-1}\boldsymbol{c}\times^{\mathrm{T}} \end{bmatrix}. \tag{2.74}
$$

These are just the inverses of Eqs. 2.62 and 2.63.

2.16 Planar Vectors

If a rigid body is constrained to move parallel to a given plane, then it becomes, in effect, a 'planar' rigid body. If every body in a rigid-body system is constrained to move parallel to the same plane, then it is a planar rigid-body system. Many practical rigid-body systems are planar.

Spatial vectors can easily describe the motions of planar rigid bodies, since planar motion is a subset of spatial motion. However, it can be advantageous to define a stripped-down version of spatial vector algebra specifically for planar rigid bodies—a planar vector algebra. A planar rigid body has three degrees of motion freedom, so planar vectors are elements of the vector spaces M^3 and F^3.

The easiest way to derive planar vector algebra is to start with spatial vector algebra and restrict it to the x–y plane. Essentially, this means removing the first, second and sixth basis vectors and their corresponding coordinates. Thus:

$$
\left.\begin{matrix}
\boldsymbol{d}_{Ox} \\
\boldsymbol{d}_{Oy} \\
\boldsymbol{d}_{Oz} \\
\boldsymbol{d}_x \\
\boldsymbol{d}_y \\
\boldsymbol{d}_z
\end{matrix}\right\} \Rightarrow \begin{matrix} \boldsymbol{d}_O \\ \boldsymbol{d}_x \\ \boldsymbol{d}_y \end{matrix}
\qquad
\left.\begin{matrix}
\boldsymbol{e}_x \\
\boldsymbol{e}_y \\
\boldsymbol{e}_z \\
\boldsymbol{e}_{Ox} \\
\boldsymbol{e}_{Oy} \\
\boldsymbol{e}_{Oz}
\end{matrix}\right\} \Rightarrow \begin{matrix} \boldsymbol{e} \\ \boldsymbol{e}_{Ox} \\ \boldsymbol{e}_{Oy} \end{matrix}
\qquad
\left.\begin{matrix}
\omega_x \\
\omega_y \\
\omega_z \\
v_{Ox} \\
v_{Oy} \\
v_{Oz}
\end{matrix}\right\} \Rightarrow \begin{matrix} \omega \\ v_{Ox} \\ v_{Oy} \end{matrix}
\qquad
\left.\begin{matrix}
n_{Ox} \\
n_{Oy} \\
n_{Oz} \\
f_x \\
f_y \\
f_z
\end{matrix}\right\} \Rightarrow \begin{matrix} n_O \\ f_x \\ f_y \end{matrix}
$$

Note that we have removed the z subscript from the angular basis vectors and coordinates.

The planar versions of the spatial cross operator, coordinate transform and inertia matrix can all be obtained by extracting the appropriate 3×3 submatrix from the 6×6 matrix. For example, the planar cross operator matrix is obtained as follows:

$$
\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} \times = \begin{bmatrix} \boldsymbol{\omega}\times & \boldsymbol{0} \\ \boldsymbol{v}_O\times & \boldsymbol{\omega}\times \end{bmatrix} = \begin{bmatrix}
0 & -\omega_z & \omega_y & 0 & 0 & 0 \\
\omega_z & 0 & -\omega_x & 0 & 0 & 0 \\
-\omega_y & \omega_x & 0 & 0 & 0 & 0 \\
0 & -v_{Oz} & v_{Oy} & 0 & -\omega_z & \omega_y \\
v_{Oz} & 0 & -v_{Ox} & \omega_z & 0 & -\omega_x \\
-v_{Oy} & v_{Ox} & 0 & -\omega_y & \omega_x & 0
\end{bmatrix},
$$

so

$$
\begin{bmatrix} \omega \\ v_{Ox} \\ v_{Oy} \end{bmatrix} \times = \begin{bmatrix}
0 & 0 & 0 \\
v_{Oy} & 0 & -\omega \\
-v_{Ox} & \omega & 0
\end{bmatrix} \tag{2.75}
$$

and

$$
\begin{bmatrix} \omega \\ v_{Ox} \\ v_{Oy} \end{bmatrix} \times \begin{bmatrix} m \\ m_{Ox} \\ m_{Oy} \end{bmatrix} = \begin{bmatrix}
0 \\
v_{Oy}\,m - \omega\,m_{Oy} \\
\omega\,m_{Ox} - v_{Ox}\,m
\end{bmatrix}. \tag{2.76}
$$

In like manner, we can deduce that the formula for a planar coordinate transformation matrix corresponding to a shift of origin from $(0, 0)$ to (x, y), followed by a rotation of the coordinate frame about its new origin by an angle of θ, is

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -y & 1 & 0 \\ x & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sx - cy & c & s \\ cx + sy & -s & c \end{bmatrix}, \qquad (2.77)$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. Note that planar vectors inherit the properties $\boldsymbol{v}\times^* = -(\boldsymbol{v}\times)^{\mathrm{T}}$ and $\boldsymbol{X}^* = \boldsymbol{X}^{-\mathrm{T}}$ from their spatial-vector counterparts. And finally, the formula for the inertia matrix of a planar rigid body is

$$\boldsymbol{I} = \begin{bmatrix} I_C + m\,(c_x^2 + c_y^2) & -m\,c_y & m\,c_x \\ -m\,c_y & m & 0 \\ m\,c_x & 0 & m \end{bmatrix} = \begin{bmatrix} I_O & -m\,c_y & m\,c_x \\ -m\,c_y & m & 0 \\ m\,c_x & 0 & m \end{bmatrix}, \quad (2.78)$$

where m is the mass, c_x and c_y are the coordinates of the centre of mass, and I_C and I_O are the rotational inertias of the body about its centre of mass and coordinate origin, respectively. A total of four parameters are needed to define a planar rigid-body inertia.

2.17   Further Reading

Spatial vectors resemble the twists and wrenches of screw theory, the real-number motors and the elements of the Lie algebra se(3) and its dual, se*(3). Screw theory was developed in the 19th century, and predates the modern concept of a vector. The main work on screw theory is Ball (1900), but more modern treatments can be found in Angeles (2003), Hunt (1978) and Selig (1996). Motors come in two varieties: 6D vectors over the field of real numbers and 3D vectors over the ring of dual numbers. The former are described in von Mises (1924a,b) and the latter in Brand (1953). The main difference between spatial vectors and motors is that all motors reside in a single vector space. In this respect, the spatial vectors in Featherstone (1987) closely resemble motor algebra. Formally, a Lie algebra is the tangent space to a Lie group at the identity. The spaces M^6 and M^3 correspond to the Lie algebras se(3) and se(2), and F^6 and F^3 correspond to their duals, se*(3) and se*(2). se(3) and se(2) are the algebras associated with the special Euclidean groups SE(3) and SE(2), which are the sets of every possible displacement of a rigid body in a 3D or 2D Euclidean space. Lie algebras are covered in Murray et al. (1994) and Selig (1996). Other articles that may be of interest include Jain (1991); Park et al. (1995); Plücker (1866); Rodriguez et al. (1991); Woo and Freudenstein (1970). There are many more, of course—the literature on 6D vectors of various kinds is quite extensive.

Chapter 3

Dynamics of Rigid Body Systems

The purpose of a dynamics algorithm is to evaluate numerically an equation of motion; but, before we can evaluate it, we must first know what that equation is. This chapter introduces the equations of motion for a general rigid-body system, and presents a variety of methods for constructing them from simpler equations. These methods form the basis for various algorithms that appear in later chapters. However, the focus of this chapter is mainly on the mathematics of rigid-body dynamics, rather than on particular algorithms.

A rigid-body system is a collection of rigid bodies that may be connected together by joints, and that may be acted upon by various forces. The effect of a joint is to impose a motion constraint on the two bodies it connects: relative motions are allowed in some directions but not in others. To be more precise, the effect of a joint is to introduce a constraint force into the system. This force has the special property that we do not know its value, but we do know what effect it has on the system—it takes whatever value it needs to take in order that the resulting motion complies with the motion constraint. A large part of this chapter is concerned with methods for describing motion constraints, and for applying them to rigid-body equations of motion.

Sections 3.1 and 3.2 provide a high-level introduction to the subject: Section 3.1 presents the equations of motion in their various standard forms, and Section 3.2 explains how they are constructed from equations describing the subsystems, individual rigid bodies and motion constraints. The next three sections cover the topic of motion constraint in more detail: Section 3.3 shows how to express them as vector subspaces; Section 3.4 presents a standard classification of the kinds of constraint that can arise in a rigid-body system; and Section 3.5 presents a detailed model of joint constraints. (We define a joint to be any kinematic constraint on the relative motion of two bodies.) Finally, Sections 3.6 and 3.7 show how to obtain explicit equations of motion for individual

rigid bodies, and for collections of rigid bodies, that are connected together by joints. These last two sections illustrate the use of spatial vectors in formulating equations of motion.

3.1   Equations of Motion

The equation of motion for a general rigid-body system can usually be written in the following canonical form:

$$\boldsymbol{H}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau}\,. \qquad (3.1)$$

In this equation, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are vectors of generalized position, velocity and acceleration variables, respectively, and $\boldsymbol{\tau}$ is a vector of generalized forces. $\boldsymbol{H}$ is the generalized inertia matrix, and it is written $\boldsymbol{H}(\boldsymbol{q})$ to show that it depends on $\boldsymbol{q}$. Likewise, $\boldsymbol{C}$ is the generalized bias force, and it is written $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ to show that it depends on both $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Once these dependencies are understood, the shorter symbols $\boldsymbol{H}$ and $\boldsymbol{C}$ are used. The bias force is simply the value of $\boldsymbol{\tau}$ that will produce zero acceleration. It accounts for the Coriolis and centrifugal forces, gravity, and any other forces acting on the system other than those in $\boldsymbol{\tau}$. Together, $\boldsymbol{H}$ and $\boldsymbol{C}$ are the coefficients of the equation of motion, and $\boldsymbol{\tau}$ and $\ddot{\boldsymbol{q}}$ are the variables. In addition to its role in the equation of motion, $\boldsymbol{H}$ has another important property: the kinetic energy of the system is

$$T = \frac{1}{2}\,\dot{\boldsymbol{q}}^{\mathrm{T}}\boldsymbol{H}\,\dot{\boldsymbol{q}}\,. \qquad (3.2)$$

Every quantity in Eq. 3.1 is either a coordinate vector or a matrix. It is useful to consider, at least briefly, what kinds of object these quantities represent. To this end, we let the symbols M^n and F^n denote the spaces of n-dimensional motion and force vectors, respectively, where n is the degree of freedom of the system described by Eq. 3.1. We can now say that $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ represent elements of M^n, whereas $\boldsymbol{\tau}$ and $\boldsymbol{C}$ represent elements of F^n, and $\boldsymbol{H}$ is a mapping from M^n to F^n. However, $\boldsymbol{q}$ does not represent a vector, but instead represents a point in the system's configuration space, which is an n-dimensional manifold.

By definition, a set of generalized force variables cannot be chosen at random, but must be chosen so that $\boldsymbol{\tau}^{\mathrm{T}}\dot{\boldsymbol{q}}$ is the power delivered by $\boldsymbol{\tau}$ to the system. This implies that a system of generalized coordinates on M^n and F^n is actually a system of dual coordinates. This, in turn, implies that generalized motion and force vectors share many of the algebraic properties of spatial vectors, as described in Chapter 2. The one big exception is that a cross-product operator is not defined on generalized motions and forces.

Strictly speaking, $\boldsymbol{H}$ and $\boldsymbol{C}$ do not depend only on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, but also on the rigid-body system itself. Thus, it would be more accurate to write $\boldsymbol{H}(model, \boldsymbol{q})$ and $\boldsymbol{C}(model, \boldsymbol{q}, \dot{\boldsymbol{q}})$, where the symbol *model* refers to a collection of data that describes a particular rigid-body system in terms of its component parts: the

number of bodies and joints, the manner in which they are connected together, the values of the bodies' inertia parameters, and so on. We call this kind of description a *system model*, and it is the subject of Chapter 4. For now, it is enough to know that such a model exists, and that quantities like $\boldsymbol{H}$ and $\boldsymbol{C}$ depend on it.

Our eventual aim is to be able to calculate the numeric values of one or more of the quantities appearing in Eq. 3.1. In particular, we have an interest in the following two calculations: *forward dynamics*, which is the calculation of $\ddot{\boldsymbol{q}}$ given $\boldsymbol{\tau}$, and *inverse dynamics*, which is the calculation of $\boldsymbol{\tau}$ given $\ddot{\boldsymbol{q}}$. We can express these calculations succinctly as follows:

$$\ddot{\boldsymbol{q}} = \mathrm{FD}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}) \tag{3.3}$$

and

$$\boldsymbol{\tau} = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\,, \tag{3.4}$$

where FD and ID are functions denoting the forward and inverse dynamics calculations, respectively. On comparing these equations with Eq. 3.1, it is clear that FD and ID must evaluate to $\boldsymbol{H}^{-1}(\boldsymbol{\tau} - \boldsymbol{C})$ and $\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C}$, respectively. However, the algorithms that implement them need not necessarily work by calculating $\boldsymbol{C}$ or $\boldsymbol{H}$ or $\boldsymbol{H}^{-1}$. Thus, the useful property of Eqs. 3.3 and 3.4 is that they show clearly the inputs and outputs of each calculation, but do not imply any particular method of calculation.

Equation 3.1 assumes that the position variables are the integrals of the velocity variables. This is not always the case. For example, if the system contains nonholonomic constraints (see §3.4) then it will have more position variables than velocity variables. Alternatively, certain kinds of motion are better described by a redundant set of position variables, which again leads to there being more position variables than velocity variables. Even if the number of position and velocity variables is the same, it may occasionally be useful for the latter to be something other than the derivatives of the former. Whatever the reason, we can adapt Eq. 3.1 simply by replacing $\dot{\boldsymbol{q}}$ with an alternative vector of velocity variables, $\boldsymbol{\alpha}$, to get

$$\boldsymbol{H}(\boldsymbol{q})\,\dot{\boldsymbol{\alpha}} + \boldsymbol{C}(\boldsymbol{q}, \boldsymbol{\alpha}) = \boldsymbol{\tau}\,. \tag{3.5}$$

A corresponding modification is then made to Eqs. 3.3 and 3.4. For simulation purposes, we still need to know the value of $\dot{\boldsymbol{q}}$, so that it can be integrated to obtain $\boldsymbol{q}$. Equation 3.5 must therefore be supplemented with an equation to calculate $\dot{\boldsymbol{q}}$ from $\boldsymbol{q}$ and $\boldsymbol{\alpha}$. This equation can be written

$$\dot{\boldsymbol{q}} = \boldsymbol{Q}(\boldsymbol{q})\,\boldsymbol{\alpha}\,, \tag{3.6}$$

where $\boldsymbol{Q}$ is a matrix that depends on $\boldsymbol{q}$ and the system model. Alternatively, it can be written

$$\dot{\boldsymbol{q}} = \mathrm{qdfn}(model, \boldsymbol{q}, \boldsymbol{\alpha})\,, \tag{3.7}$$

where qdfn denotes the calculation of $\dot{q}$ from q and α. Equation 3.6 reveals the mathematical structure of the relationship between q, $\dot{q}$ and α by showing that $\dot{q}$ depends linearly on α; while Eq. 3.7 shows the inputs and outputs of the calculation without implying any particular calculation method.

Still on the subject of simulation, most simulators expect to be given a set of first-order differential equations. We can accommodate this by defining a state variable, $x = [q^{\mathrm{T}} \ \alpha^{\mathrm{T}}]^{\mathrm{T}}$, and expressing the forward dynamics in state-space form:

$$\dot{x} = \begin{bmatrix} \dot{q} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} Q\,\alpha \\ H^{-1}(\tau - C) \end{bmatrix} . \tag{3.8}$$

If FD_x denotes the state-space forward dynamics calculation function, then

$$\dot{x} = \mathrm{FD}_x(model, x, \tau) = \begin{bmatrix} \mathrm{qdfn}(model, q, \alpha) \\ \mathrm{FD}(model, q, \alpha, \tau) \end{bmatrix} . \tag{3.9}$$

In subsequent sections and chapters, we will work mostly with systems that can be described by Eq. 3.1, rather than the more general systems described by Eq. 3.5. This is because most of the results we shall obtain for the simpler kind of system can be adapted in a straightforward manner to the more general kind. Most dynamics algorithms are indifferent to the distinction between Eqs. 3.1 and 3.5, except at the level of programming details.

Finally, there is one last possibility to mention. It is sometimes impractical or undesirable to identify and work with an explicit set of independent velocity variables. In this case, one still uses Eq. 3.1 or 3.5 as the equation of motion; but now $\dot{q}$ and $\ddot{q}$ (or α and $\dot{\alpha}$) are subject to a set of motion constraints, and so the equation of motion must be supplemented with a second equation to describe these constraints. This topic is covered in the next section.

3.2   Constructing Equations of Motion

The equations of motion for a rigid-body system are obtained as the end result of a sequence of mathematical operations. The starting point is a collection of equations of motion for individual rigid bodies and/or subsystems; and the two main operations that we perform on them are:

- collecting equations together to form the equation of a bigger subsystem, and

- applying additional motion constraints.

By performing these operations in a particular order, we define a particular procedure for obtaining the desired equation of motion, hence also a particular algorithm (or class of algorithms) for a computer to calculate the numeric value of that equation. Some of the most common procedures are as follows.

1. Collect all of the bodies (i.e., collect their equations into a single equation), and then apply all of the constraints.

2. Obtain the equation of motion for the spanning tree, and then apply all remaining constraints.

3. Starting with a single body or subsystem do: add one more body or subsystem; apply constraints pertaining to that body or subsystem; repeat.

4. Combine subsystems in pairs; apply constraints separately within each subsystem; repeat.

Method 1 is typical of many simple dynamics algorithms. It is easy to understand, but it produces large, sparse matrices. Algorithms that take this approach must use sparse matrix techniques, or they will be too slow compared with other algorithms. An example of this method appears in Section 3.7. Method 2 is standard for closed-loop systems, and is covered in detail in Chapter 8. The spanning tree of a closed-loop system is defined in Chapter 4 to be a kinematic tree that accounts for all of the bodies in the original system but only a subset of the motion constraints. The point of method 2 is that there are very efficient algorithms available for calculating the dynamics of the spanning tree. Method 3 is approximately how the articulated-body algorithm works, except that it uses articulated-body equations instead of complete equations of motion. This algorithm is described in Chapter 7. Finally, method 4 is used by divide-and-conquer algorithms (Featherstone, 1999a,b). Algorithms that work like this are suitable for implementation on parallel computers.

Collecting Equations of Motion

Suppose we have a set of N rigid-body subsystems, such that system i has the equation of motion $\boldsymbol{H}_i\ddot{\boldsymbol{q}}_i + \boldsymbol{C}_i = \boldsymbol{\tau}_i$. If we wish to treat them as a single system, then the equation of motion for that system is

$$
\begin{bmatrix}
\boldsymbol{H}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\
\boldsymbol{0} & \boldsymbol{H}_2 & \cdots & \boldsymbol{0} \\
\vdots & \vdots & \ddots & \vdots \\
\boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{H}_N
\end{bmatrix}
\begin{bmatrix}
\ddot{\boldsymbol{q}}_1 \\
\ddot{\boldsymbol{q}}_2 \\
\vdots \\
\ddot{\boldsymbol{q}}_N
\end{bmatrix}
+
\begin{bmatrix}
\boldsymbol{C}_1 \\
\boldsymbol{C}_2 \\
\vdots \\
\boldsymbol{C}_N
\end{bmatrix}
=
\begin{bmatrix}
\boldsymbol{\tau}_1 \\
\boldsymbol{\tau}_2 \\
\vdots \\
\boldsymbol{\tau}_N
\end{bmatrix}. \tag{3.10}
$$

Note that a single rigid body is also a rigid-body subsystem. Thus, we could have started with a set of N rigid bodies, or even an arbitrary mixture of individual bodies and other systems. If subsystem i happens to be a single rigid body, then we can use its spatial equation of motion, $\boldsymbol{I}_i\boldsymbol{a}_i + \boldsymbol{p}_i = \boldsymbol{f}_i$, in place of the generalized-coordinates equation of motion (i.e., we use $\boldsymbol{I}_i$ in place of $\boldsymbol{H}_i$, etc.). The reason why this works is because spatial motion and force vectors are special cases of generalized motion and force vectors; that is, M^6 and F^6 are special cases of M^n and F^n, and Plücker coordinates on M^6 and F^6 are a special case of generalized coordinates on M^n and F^n. Thus, we can mix spatial velocities freely with generalized velocities, and similarly for accelerations and forces.

Motion Constraints

Motion constraints are algebraic constraints on the motion variables of a system. They can be expressed in two different ways, as follows:

$$\begin{array}{cccc} & \text{position} & \text{velocity} & \text{acceleration} \\ \text{implicit:} & \boldsymbol{\phi}(\boldsymbol{q}) = \boldsymbol{0} & \boldsymbol{K}\dot{\boldsymbol{q}} = \boldsymbol{0} & \boldsymbol{K}\ddot{\boldsymbol{q}} = \boldsymbol{k} \\ \text{explicit:} & \boldsymbol{q} = \boldsymbol{\gamma}(\boldsymbol{y}) & \dot{\boldsymbol{q}} = \boldsymbol{G}\dot{\boldsymbol{y}} & \ddot{\boldsymbol{q}} = \boldsymbol{G}\ddot{\boldsymbol{y}} + \boldsymbol{g} \end{array} \qquad (3.11)$$

where

$$\boldsymbol{K} = \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{q}}, \qquad \boldsymbol{k} = -\dot{\boldsymbol{K}}\dot{\boldsymbol{q}}, \qquad \boldsymbol{G} = \frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{y}} \quad \text{and} \quad \boldsymbol{g} = \dot{\boldsymbol{G}}\dot{\boldsymbol{y}}.$$

(These are not the most general equations. The general case is covered in §3.4.) If $\boldsymbol{\phi}$ and $\boldsymbol{\gamma}$ both describe the same constraint, then

$$\boldsymbol{\phi} \circ \boldsymbol{\gamma} = \boldsymbol{0}, \qquad \boldsymbol{K}\boldsymbol{G} = \boldsymbol{0} \quad \text{and} \quad \boldsymbol{K}\boldsymbol{g} = \boldsymbol{k}. \qquad (3.12)$$

Motion constraints are caused by constraint forces. If an unconstrained (or less constrained) rigid-body system has the equation of motion

$$\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} = \boldsymbol{\tau}, \qquad (3.13)$$

and this system is to be subjected to a motion constraint, then the equation of motion for the constrained system is

$$\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} = \boldsymbol{\tau} + \boldsymbol{\tau}_c, \qquad (3.14)$$

where $\boldsymbol{\tau}_c$ is the constraint force. This force is unknown, but it has one important property that allows us either to calculate its value, or to eliminate it from the equation, as desired:

> The constraint force delivers zero power along every direction of velocity freedom that is compatible with the motion constraints. (Jourdain's principle of virtual power.)

In effect, this statement says that $\boldsymbol{\tau}_c$ must satisfy $\boldsymbol{\tau}_c \cdot \dot{\boldsymbol{q}} = 0$, not just for one value of $\dot{\boldsymbol{q}}$, but for every value that is allowed by the constraint. If the system is subject to an implicit motion constraint, then it can be shown that $\boldsymbol{\tau}_c$ takes the form

$$\boldsymbol{\tau}_c = \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\lambda}, \qquad (3.15)$$

where $\boldsymbol{\lambda}$ is a vector of unknown force variables; and if the system is subject to an explicit motion constraint, then $\boldsymbol{\tau}_c$ has the property

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}_c = \boldsymbol{0}. \qquad (3.16)$$

These equations are easily verified. If $\boldsymbol{\tau}_c = \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\lambda}$ then $\boldsymbol{\tau}_c \cdot \dot{\boldsymbol{q}} = \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{K}\dot{\boldsymbol{q}}$, and this expression will be zero for every $\dot{\boldsymbol{q}}$ satisfying $\boldsymbol{K}\dot{\boldsymbol{q}} = \boldsymbol{0}$. Likewise, if $\dot{\boldsymbol{q}} = \boldsymbol{G}\dot{\boldsymbol{y}}$ then $\dot{\boldsymbol{q}} \cdot \boldsymbol{\tau}_c = \dot{\boldsymbol{y}}^{\mathrm{T}}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}_c$, and this expression will be zero for all $\dot{\boldsymbol{y}}$ if $\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}_c = \boldsymbol{0}$. The elements of $\boldsymbol{\lambda}$ can be regarded as a set of Lagrange multipliers.

Applying Implicit Motion Constraints

To apply an implicit motion constraint to a rigid-body system, we simply substitute Eq. 3.15 into Eq. 3.14, and combine the result with the implicit acceleration constraint equation in Eq. 3.11. The result is

$$\begin{bmatrix} H & K^{\mathrm{T}} \\ K & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} \tau - C \\ k \end{bmatrix}. \tag{3.17}$$

This is a set of $n + n_c$ equations in $n + n_c$ unknowns, where $n = \dim(\ddot{q})$ is the degree of freedom of the unconstrained system and $n_c = \dim(\lambda) = \dim(\phi)$ is the number of constraints imposed upon it. It can be shown that the rank of the coefficient matrix is $n + n_{ic}$, where $n_{ic} = \operatorname{rank}(K)$ is the number of independent constraints imposed on the system. Thus, if $n_{ic} = n_c$ then the coefficient matrix is nonsingular, and Eq. 3.17 can be solved for both $\ddot{q}$ and λ. However, if $n_{ic} < n_c$ then it is still possible to solve Eq. 3.17 for $\ddot{q}$, but there will be $n_c - n_{ic}$ degrees of indeterminacy in λ. A system in which $n_{ic} < n_c$ is said to be *overconstrained*. Ideally, we would always have $n_{ic} = n_c$, but it is not always possible to guarantee this in practice.

Applying Explicit Motion Constraints

To apply an explicit motion constraint to a rigid-body system, we combine Eq. 3.14 with Eq. 3.16 and the explicit acceleration constraint equation in Eq. 3.11. The resulting equation is

$$\begin{bmatrix} H & -1 & 0 \\ -1 & 0 & G \\ 0 & G^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \tau_c \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \tau - C \\ -g \\ 0 \end{bmatrix}. \tag{3.18}$$

Applying one step of Gaussian elimination to this equation produces

$$\begin{bmatrix} H & -1 & 0 \\ 0 & H^{-1} & G \\ 0 & G^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \tau_c \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} \tau - C \\ H^{-1}(\tau - C) - g \\ 0 \end{bmatrix},$$

from which we can extract

$$\begin{bmatrix} H^{-1} & G \\ G^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \tau_c \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} H^{-1}(\tau - C) - g \\ 0 \end{bmatrix}; \tag{3.19}$$

and applying Gaussian elimination to this equation produces

$$\begin{bmatrix} H^{-1} & G \\ 0 & -G^{\mathrm{T}}HG \end{bmatrix} \begin{bmatrix} \tau_c \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} H^{-1}(\tau - C) - g \\ -G^{\mathrm{T}}(\tau - C - Hg) \end{bmatrix},$$

from which we get

$$G^{\mathrm{T}}HG\,\ddot{y} = G^{\mathrm{T}}(\tau - C - Hg). \tag{3.20}$$

Finally, if we make the substitution $\boldsymbol{u} = \boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}$, where $\boldsymbol{u}$ is a new set of generalized forces, then we get

$$\boldsymbol{H}_G\,\ddot{\boldsymbol{y}} + \boldsymbol{C}_G = \boldsymbol{u}\,, \tag{3.21}$$

where

$$\boldsymbol{H}_G = \boldsymbol{G}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{G} \quad \text{and} \quad \boldsymbol{C}_G = \boldsymbol{G}^{\mathrm{T}}(\boldsymbol{C} + \boldsymbol{H}\boldsymbol{g})\,.$$

Thus, we have obtained a new equation of motion for the constrained system that has the same algebraic form as the equation of motion for the unconstrained system in Eq. 3.13.

A few remarks are in order at this point. First, Eq. 3.19 can be regarded as the dual of Eq. 3.17 in the sense that the motion and force variables have changed places. However, 3.17 is more useful than 3.19 in practice. Second, Eq. 3.20 can be obtained directly from Eq. 3.14 by first premultiplying it by $\boldsymbol{G}^{\mathrm{T}}$, which eliminates $\boldsymbol{\tau}_c$, and then substituting for $\ddot{\boldsymbol{q}}$ using the explicit acceleration constraint in Eq. 3.11. This method is sometimes called the *projection method* on the grounds that the first step amounts to a projection of the equation of motion onto the range space of $\boldsymbol{G}^{\mathrm{T}}$. Third, $\boldsymbol{H}_G$ inherits from $\boldsymbol{H}$ the properties of symmetry and positive definiteness. It also has the property that the kinetic energy of the constrained system is

$$T = \frac{1}{2}\,\dot{\boldsymbol{y}}^{\mathrm{T}}\boldsymbol{H}_G\,\dot{\boldsymbol{y}}\,. \tag{3.22}$$

Finally, we can show that $\boldsymbol{u}\cdot\dot{\boldsymbol{y}}$ is the power delivered to the constrained system as follows:

$$\boldsymbol{u}\cdot\dot{\boldsymbol{y}} = (\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau})^{\mathrm{T}}\dot{\boldsymbol{y}} = \boldsymbol{\tau}^{\mathrm{T}}\boldsymbol{G}\dot{\boldsymbol{y}} = \boldsymbol{\tau}\cdot\dot{\boldsymbol{q}}\,. \tag{3.23}$$

Thus, $\boldsymbol{u}$ does indeed satisfy the definition of a vector of generalized forces.

3.3   Vector Subspaces

Vector subspaces are the natural mathematical tool for describing and analysing kinematic constraints because they capture the essential aspects of a constraint. For example, the system in Eq. 3.13 is free to move in M^n because $\dot{\boldsymbol{q}}$ is an element of M^n and it is unconstrained. However, the system in Eq. 3.14 is constrained to move only in the subspace $S \subset \mathsf{M}^n$. If the constraint has been described implicitly, then $S = \mathrm{null}(\boldsymbol{K})$; and if it has been described explicitly, then $S = \mathrm{range}(\boldsymbol{G})$. If $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ both describe the same constraint, then the one thing they must have in common is that $\mathrm{null}(\boldsymbol{K}_1) = \mathrm{null}(\boldsymbol{K}_2) = S$. Likewise, if $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ describe the same constraint, then the one thing they must have in common is that $\mathrm{range}(\boldsymbol{G}_1) = \mathrm{range}(\boldsymbol{G}_2) = S$. Thus, it is the subspace S, rather than any one particular matrix, that captures the essence of the constraint. This section therefore reviews some basic material on vector subspaces and how they can be used.

Basics

A subspace of a vector space is a subset that is also a vector space. Thus, a subspace is any subset that is closed with respect to the operations of addition and scalar multiplication. Let V be a vector space, and let $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots\}$ be any subset of V. The quantity span(Y) is the set of all linear combinations of the elements of Y, and is therefore a subspace. It follows that any subset that satisfies $Y = $ span(Y) is a subspace. If S is a subspace of V, and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots\}$ is a basis on S, then $S = $ span(Y) and dim$(S) = |Y|$ (i.e., the dimension of S equals the number of elements in Y). A zero-dimensional subspace is a set containing only the zero vector. Given that a subspace is itself a vector space, it follows that a subspace may have subspaces, and so on.

There is no special symbol for 'is a subspace of'. Instead, we use the symbol for a subset. Thus, we write $Y \subseteq V$ to indicate that Y is a subset of V, and $S \subseteq V$ to indicate that S is a subspace. One must read the surrounding text to discover which meaning is intended. The expression $S \subset V$ indicates that S is a proper subspace of V, which implies that dim$(S) < $ dim(V).

Suppose that $S \subseteq V$, and that dim$(S) = m$ and dim$(V) = n$. It follows that m must lie in the range $0 \le m \le n$. If $m = 0$ then $S = \{\boldsymbol{0}\}$; and if $m = n$ then $S = V$; but if m takes any other value, then there are infinitely many subspaces having that dimension. The number of parameters required to identify uniquely an m-dimensional subspace of an n-dimensional vector space is $m(n - m)$.

Matrix Representation

The simplest way to represent a subspace is as the range or null space of a matrix. We will mostly use the former. Let S be an m-dimensional subspace of the n-dimensional vector space V, and let $\mathcal{S} = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$ be a basis on S. Any vector $\boldsymbol{v} \in S$ can then be expressed in the form $\boldsymbol{v} = \sum_{i=1}^{m} \boldsymbol{s}_i \alpha_i$, where α_i are the coordinates of $\boldsymbol{v}$ in $\mathcal{S}$. If $\boldsymbol{s}_i$ are themselves coordinate vectors, expressed in a basis on V, then they can be assembled into an $n \times m$ matrix, $\boldsymbol{S} = [\boldsymbol{s}_1 \ \cdots \ \boldsymbol{s}_m]$, which has the property that range$(\boldsymbol{S}) = S$. $\boldsymbol{v}$ can then be expressed in the form $\boldsymbol{v} = \boldsymbol{S}\boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = [\alpha_1 \ \cdots \ \alpha_m]^\mathrm{T}$.

The matrix $\boldsymbol{S}$ defines both a subspace and a basis. In fact, of the mn numbers contained in $\boldsymbol{S}$, we can say that $m(n - m)$ define the subspace and m^2 define the basis. If $\mathcal{S}' = \{\boldsymbol{s}'_1, \ldots, \boldsymbol{s}'_m\}$ is another basis on S, then the matrix $\boldsymbol{S}' = [\boldsymbol{s}'_1 \ \cdots \ \boldsymbol{s}'_m]$ defines the same subspace as $\boldsymbol{S}$, but a different basis. In this case, $\boldsymbol{S}'$ can be expressed in the form $\boldsymbol{S}' = \boldsymbol{S}\boldsymbol{A}$, where $\boldsymbol{A}$ is the coordinate transform from $\mathcal{S}'$ to $\mathcal{S}$ coordinates. (The elements of $\boldsymbol{A}$ satisfy $\boldsymbol{s}'_j = \sum_{i=1}^{m} \boldsymbol{s}_i A_{ij}$.) Thus, any matrix $\boldsymbol{S}\boldsymbol{A}$, where $\boldsymbol{A}$ is invertible, describes the same subspace as $\boldsymbol{S}$.

Decomposition of Vectors

One of the most basic operations that can be performed on a vector is to decompose it into its components in two subspaces. Given a vector $\boldsymbol{v} \in V$ and

subspaces $S_1, S_2 \subseteq V$, the idea is to express $\boldsymbol{v}$ in the form $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$, where $\boldsymbol{v}_1 \in S_1$ and $\boldsymbol{v}_2 \in S_2$. This decomposition is possible if $\boldsymbol{v} \in \text{span}(S_1 \cup S_2)$, and it is unique if $S_1 \cap S_2 = \{\boldsymbol{0}\}$ (i.e., S_1 and S_2 have no nonzero element in common). If we want the decomposition to be both possible and unique for any $\boldsymbol{v}$, then S_1 and S_2 must satisfy

$$S_1 \cap S_2 = \{\boldsymbol{0}\} \qquad \text{and} \qquad \dim(S_1) + \dim(S_2) = \dim(V). \tag{3.24}$$

(These conditions imply $\text{span}(S_1 \cup S_2) = V$.) If S_1 and S_2 satisfy Eq. 3.24 then we can say that V is the *direct sum* of S_1 and S_2, which is written

$$V = S_1 \oplus S_2. \tag{3.25}$$

This equation is simply a short-hand way of saying that any element of V can be decomposed uniquely into the sum of an element of S_1 and an element of S_2. Essentially, Eqs. 3.24 and 3.25 say the same thing.

Let us now perform the decomposition. We are given a vector $\boldsymbol{v} \in V$ and two matrices $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ representing subspaces that satisfy Eq. 3.25; and the task is to find $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ that satisfy

$$\boldsymbol{v} = \boldsymbol{S}_1\, \boldsymbol{\alpha}_1 + \boldsymbol{S}_2\, \boldsymbol{\alpha}_2 = [\boldsymbol{S}_1 \ \ \boldsymbol{S}_2] \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{bmatrix}. \tag{3.26}$$

This problem can be solved using the following result: if $S_1 \oplus S_2 = V$ then $[\boldsymbol{S}_1 \ \boldsymbol{S}_2]$ defines a basis on V, and is therefore a nonsingular matrix. Given that $[\boldsymbol{S}_1 \ \boldsymbol{S}_2]^{-1}$ exists, the solution to Eq. 3.26 is simply

$$\begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{bmatrix} = [\boldsymbol{S}_1 \ \ \boldsymbol{S}_2]^{-1} \boldsymbol{v}. \tag{3.27}$$

This formula extends in the obvious way to the task of decomposing a vector into components in more than two subspaces.

Orthogonal Complements

If two Euclidean vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$, satisfy the equation $\boldsymbol{u} \cdot \boldsymbol{v} = 0$, then they are said to be orthogonal. If Y_1 and Y_2 are subsets of E^n, and every element of Y_1 is orthogonal to every element of Y_2, then the two subsets are said to be orthogonal, which is written $Y_1 \perp Y_2$. The set of all vectors that are orthogonal to a given subset, $Y \subseteq \mathsf{E}^n$, is called its orthogonal complement, and is written $Y^\perp$. An orthogonal complement is always a subspace. Some basic properties of orthogonal subsets are listed in Table 3.1.

This form of orthogonality is based on the Euclidean inner product, and is therefore applicable only to Euclidean vectors. However, it is possible to define other forms of orthogonality, which are based on other scalar products. In particular, if $\boldsymbol{u} \in U$ and $\boldsymbol{v} \in V$, where $U = V^*$, then a scalar product is defined between $\boldsymbol{u}$ and $\boldsymbol{v}$, and it is possible to say that if $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{0}$ then they

Subsets:	Subspaces:
$Y_1 \perp Y_2 \Rightarrow Y_2 \perp Y_1$	$\dim(S) + \dim(S^\perp) = \dim(U)$
$Y_1 \perp Y_2 \Rightarrow \mathrm{span}(Y_1) \perp Y_2$	$S_1 \perp S_2 \Leftrightarrow \boldsymbol{S}_1^\mathrm{T} \boldsymbol{S}_2 = \boldsymbol{0}$
$Y_1 \perp Y_2 \wedge Z \subseteq Y_1 \Rightarrow Z \perp Y_2$	$S_1 \oplus S_2^\perp = U \Leftrightarrow (\boldsymbol{S}_1^\mathrm{T} \boldsymbol{S}_2)^{-1}$ exists
$Y_1 \perp Y_2 \Rightarrow Y_1 \subseteq Y_2^\perp$	$S_1 \oplus S_2^\perp = U \Leftrightarrow S_1^\perp \oplus S_2 = V$
$Y^\perp = (\mathrm{span}(Y))^\perp$	**Euclidean Subspaces:**
$(Y^\perp)^\perp = \mathrm{span}(Y)$	$S \oplus S^\perp = \mathsf{E}^n$

Table 3.1: Properties of orthogonal subsets. In the general case, $S, S_1, Y, Y_1 \subseteq U$ and $S_2, Y_2 \subseteq V$ where $U = V^*$. In the Euclidean case, $S, \ldots, Y_2 \subseteq \mathsf{E}^n$

are orthogonal. By extension, if Y_1 and Y_2 are subsets of U and V, respectively, and every element of Y_1 is orthogonal to every element of Y_2, then $Y_1 \perp Y_2$. Likewise, the set of all elements of V that are orthogonal to $Y \subseteq U$ is the orthogonal complement of Y.

Observe that the Euclidean and dual forms of orthogonality have different and incompatible mathematical properties. In particular, the relationship $Y_1 \perp Y_2$ can only be true in the Euclidean sense if Y_1 and Y_2 are subsets of the same vector space, and it can only be true in the dual sense if they are subsets of different vector spaces.[1] Orthogonal complements have gained a degree of notoriety in the robotics literature, owing to the mistaken practice of treating 6D vectors as if they were Euclidean, and using the Euclidean form of orthogonal complement instead of the dual form. The matter is discussed in Duffy (1990). Some authors have sought to distance themselves from this mistake by using alternative names like 'natural orthogonal complement' or 'reciprocal complement' for the dual form of the orthogonal complement.

Orthogonal complements play the following role in rigid-body dynamics: if a system is initially free to move in M^n, and a motion constraint restricts the motion to a subspace $S \subseteq \mathsf{M}^n$, then the constraint force that imposes this constraint on the system is an element of $S^\perp \subseteq \mathsf{F}^n$.

Example 3.1 *Suppose we have been given a motion subspace, $S \subseteq \mathsf{M}^6$, and a spatial inertia, $\boldsymbol{I} : \mathsf{M}^6 \mapsto \mathsf{F}^6$, which is positive definite. With these quantities, we can define two subspaces, $T_a = \boldsymbol{I}S$ and $T_c = S^\perp$, which have the property $T_a \oplus T_c = \mathsf{F}^6$. (The expression $\boldsymbol{I}S$ is the image of S under the mapping $\boldsymbol{I}$, i.e., the set $\{\boldsymbol{I}\boldsymbol{s} \mid \boldsymbol{s} \in S\}$.) We can prove this property as follows. According to Eq. 3.24, we must show that $\dim(T_a) + \dim(T_c) = 6$, and that $T_a \cap T_c$ does not contain a nonzero element. The first part is easy: $\dim(T_a) = \dim(S)$ and $\dim(T_c) = 6 - \dim(S)$. To prove the second part, we suppose that $\boldsymbol{t} \neq 0 \in (T_a \cap T_c)$ and prove a contradiction: if $\boldsymbol{t} \in T_a$ then there exists a nonzero $\boldsymbol{s} \in S$ such that $\boldsymbol{t} = \boldsymbol{I}\boldsymbol{s}$; but if $\boldsymbol{t} \in T_c$ then $\boldsymbol{s}^\mathrm{T}\boldsymbol{t} = 0$, implying $\boldsymbol{s}^\mathrm{T}\boldsymbol{I}\boldsymbol{s} = 0$; but this is*

---

[1]One way around this is to declare that a Euclidean vector space is self-dual.

impossible because $\boldsymbol{I}$ is positive definite.

Having established that $T_a \oplus T_c = \mathsf{F}^6$, we can decompose any $\boldsymbol{f} \in \mathsf{F}^6$ uniquely into components $\boldsymbol{f}_a \in T_a$ and $\boldsymbol{f}_c \in T_c$. Given any matrix $\boldsymbol{S}$ spanning S, we can obtain $\boldsymbol{f}_a$ and $\boldsymbol{f}_c$ as follows:

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{f} = \boldsymbol{S}^{\mathrm{T}}\boldsymbol{f}_a + \boldsymbol{S}^{\mathrm{T}}\boldsymbol{f}_c = \boldsymbol{S}^{\mathrm{T}}\boldsymbol{f}_a \,,$$

but $\boldsymbol{f}_a = \boldsymbol{IS}\boldsymbol{\alpha}$ for some $\boldsymbol{\alpha}$, so

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{f} = \boldsymbol{S}^{\mathrm{T}}\boldsymbol{IS}\boldsymbol{\alpha} \,,$$

which implies

$$\boldsymbol{\alpha} = (\boldsymbol{S}^{\mathrm{T}}\boldsymbol{IS})^{-1}\boldsymbol{S}^{\mathrm{T}}\boldsymbol{f} \,,$$

hence

$$\boldsymbol{f}_a = \boldsymbol{IS}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{IS})^{-1}\boldsymbol{S}^{\mathrm{T}}\boldsymbol{f} \,. \tag{3.28}$$

$\boldsymbol{f}_c$ *then follows simply from* $\boldsymbol{f}_c = \boldsymbol{f} - \boldsymbol{f}_a$. *The expression in Eq. 3.28 clearly must be independent of the particular choice of* $\boldsymbol{S}$. *We can prove this by replacing each instance of* $\boldsymbol{S}$ *in this equation with* $\boldsymbol{SA}$, *where* $\boldsymbol{A}$ *is any invertible matrix of the correct size, and showing that the '$\boldsymbol{A}$'s cancel out.*

The physical interpretation of this decomposition is as follows. If $\boldsymbol{I}$ is the inertia of a rigid body that is constrained to move in the subspace S, and $\boldsymbol{f}$ is a force applied to that body, then $\boldsymbol{f}_a$ is the component of $\boldsymbol{f}$ that causes acceleration, and $\boldsymbol{f}_c$ is the component that is opposed by the constraint force (i.e., the constraint force will be $-\boldsymbol{f}_c$). If the body is at rest, and is not subject to any forces other than $\boldsymbol{f}$ and the motion constraint force, then its acceleration will be

$$\boldsymbol{a} = \boldsymbol{I}^{-1}\boldsymbol{f}_a = \boldsymbol{S}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{IS})^{-1}\boldsymbol{S}^{\mathrm{T}}\boldsymbol{f} \,.$$

The expression $\boldsymbol{S}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{IS})^{-1}\boldsymbol{S}^{\mathrm{T}}$ *is the apparent inverse inertia of the constrained rigid body (cf. Eq. 3.54).*

3.4   Classification of Constraints

Kinematic constraints can be classified according to the nature of the constraint they impose on a rigid-body system. In every case, the constraint is described by an algebraic equation in the motion variables of the system, but the form of the equation varies. A standard classification hierarchy is shown in Figure 3.1.

The first distinction is made between equality and inequality constraints. The former arise from permanent physical contact between pairs of rigid bodies; and the latter arise when bodies are free both to make contact and to separate. Inequality constraints can be used to describe phenomena such as collision, bouncing and loss of contact. They are the subject of Chapter 11, and will not be considered further here.

The next distinction is between holonomic and nonholonomic constraints. The former are constraints on the position variables, and arise typically from
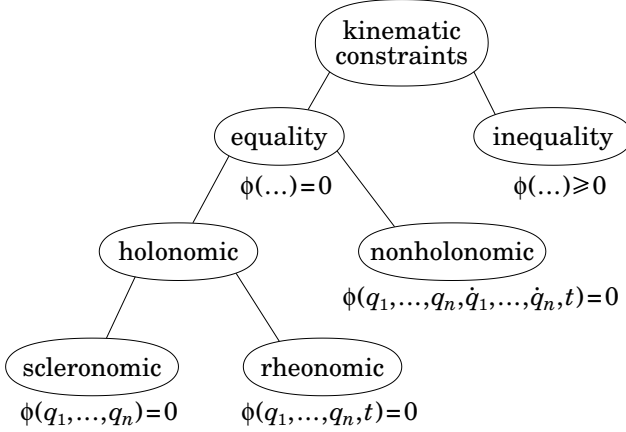
Figure 3.1: Classification of kinematic constraints

sliding contact. The latter are constraints on the velocity variables, and arise typically from rolling contact. A nonholonomic constraint function, ϕ_{nh}, is linear in the velocity variables, and can therefore be expressed in the form

$$\phi_{nh}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = \phi_{nh}^0 + \sum_{i=1}^{n} \phi_{nh}^i \, \dot{q}_i \,, \qquad (3.29)$$

where the coefficients ϕ_{nh}^i are functions of the position variables and time only. For comparison, the time derivative of a holonomic constraint function, ϕ_h, is

$$\frac{\mathrm{d}}{\mathrm{d}t} \phi_h(\boldsymbol{q}, t) = \frac{\partial \phi_h}{\partial t} + \sum_{i=1}^{n} \frac{\partial \phi_h}{\partial q_i} \, \dot{q}_i \,, \qquad (3.30)$$

which has the same algebraic form. Thus, at the velocity and acceleration levels, there is no difference between holonomic and nonholonomic constraints. The big difference between these two equations is that Eq. 3.29 is *nonintegrable*; that is, ϕ_{nh} is not the derivative of any function, or else it would simply be a holonomic constraint expressed via its derivative.[2] The consequence of this property is that a system containing nonholonomic constraints has more positional degrees of freedom than velocity degrees of freedom, and therefore needs more position variables than velocity variables. If a rigid-body system contains at least one nonholonomic constraint, then it is called a nonholonomic system; otherwise, it is called a holonomic system. Thus, Eq. 3.1 can describe only a holonomic system, but Eq. 3.5 can describe both holonomic and nonholonomic systems.

The final distinction is between a scleronomic constraint, which is a function of the position variables only, and a rheonomic constraint, which is a function

---

[2]Given that a constraint equation holds at all times, rather than only at a single instant, the two equations $\dot{\phi} = 0$ and $\phi = 0$ describe the same constraint.
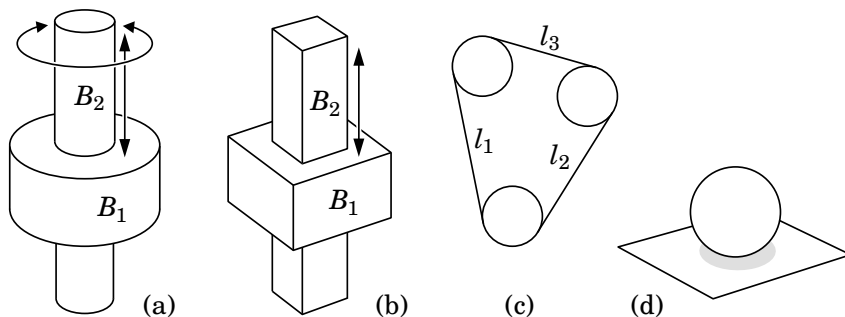
Figure 3.2: Examples of kinematic constraints

also of time. Scleronomic constraints arise from sliding contact between a sur-
face fixed in one body and a surface fixed in another. They are both the simplest
and the most common kind of constraint in a typical mechanical system. Rheo-
nomic constraints can be regarded as scleronomic constraints in which one or
more sliding freedoms have been forced to vary as a prescribed function of time.
The role of rheonomic constraints is to introduce kinematic excitations into a
system; that is, prescribed motions.

Some examples of kinematic constraints are shown in Figure 3.2. Diagrams
(a) and (b) show a cylindrical joint and a prismatic joint, respectively. Both are
scleronomic constraints. The cylindrical joint allows body B_2 to slide and rotate
relative to B_1, whereas the prismatic joint allows only a relative sliding motion.
Thus, the cylindrical joint allows two degrees of motion freedom, while the
prismatic joint allows only one. If we were to make one of the cylindrical joint's
freedoms a prescribed function of time, then it would become a one-degree-of-
freedom rheonomic constraint. If we were to do the same to the prismatic joint,
then it would become a zero-degree-of-freedom rheonomic constraint (which is
actually quite useful).

A kinematic constraint is nearly always a relationship between two bodies.
However, it is possible to devise constraints that involve more than two bodies,
and that cannot be reduced to a set of two-body constraints. An example is
shown in Figure 3.2(c). This diagram shows a 2-D rigid-body system in which
three identical circular bodies are constrained to lie within the perimeter of a
massless, zero-diameter, inelastic string. Assuming the string stays taut, the
three bodies are free to move in any way that satisfies the equation $l_1 + l_2 +
l_3 + 2\pi r = p$, where p and r are the perimeter of the string and the radius of a
body, respectively.

Figure 3.2(d) shows a sphere resting on a planar surface. If this sphere is
constrained to roll without slipping, then it has three degrees of instantaneous
motion freedom: it can roll in two directions, and it can spin about the contact
point. However, by a suitable sequence of manoeuvres, it is possible for the
sphere to arrive at any point on the plane, with any orientation. It therefore

has five degrees of position freedom. This is the characteristic feature of a nonholonomic constraint. If the sphere were free to slide, then it would be a scleronomic constraint.

It is clear from the shapes of the bodies in Figure 3.2(a) and (b) that they cannot be pulled apart; but there is nothing that physically prevents the string in 3.2(c) from going slack, or the sphere in 3.2(d) from losing contact with the plane. If we know enough about the forces that will be acting on these systems to be sure that such an event will not occur, then it is possible to model these constraints as equality constraints; otherwise, they must be modelled as inequality constraints.

3.5 Joint Constraints

We define a joint to be a kinematic constraint between two bodies. Thus, a joint can impose anywhere from zero to six constraints on the relative velocity of a pair of rigid bodies. We call the two bodies the *predecessor* and the *successor*, and we say that the joint connects *from* the predecessor *to* the successor. The joint velocity, $\boldsymbol{v}_{\mathrm{J}}$, is then defined to be the velocity of the successor relative to the predecessor:

$$\boldsymbol{v}_{\mathrm{J}} = \boldsymbol{v}_s - \boldsymbol{v}_p \,, \tag{3.31}$$

where $\boldsymbol{v}_p$ and $\boldsymbol{v}_s$ are the velocities of the predecessor and successor bodies, respectively. The general form of the joint constraint is

$$\boldsymbol{v}_{\mathrm{J}} = \boldsymbol{S}(\boldsymbol{q},t)\,\dot{\boldsymbol{q}} + \boldsymbol{\sigma}(\boldsymbol{q},t) \,, \tag{3.32}$$

where $\boldsymbol{S}$ is the joint's motion subspace matrix, $\boldsymbol{\sigma}$ is the bias velocity, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are the joint's position and velocity variables, and t is time. If the position variables are not the integrals of the velocity variables, then one must use a different symbol either for $\dot{\boldsymbol{q}}$ or for $\boldsymbol{q}$. The bias velocity is the value that $\boldsymbol{v}_{\mathrm{J}}$ takes when $\dot{\boldsymbol{q}} = \boldsymbol{0}$, and it has a nonzero value only if the joint depends explicitly on time (i.e., if it is a rheonomic or time-dependent nonholonomic constraint). If the joint does not depend on time, then Eq. 3.32 simplifies to

$$\boldsymbol{v}_{\mathrm{J}} = \boldsymbol{S}(\boldsymbol{q})\,\dot{\boldsymbol{q}} \,. \tag{3.33}$$

The effect of Eq. 3.32 is to confine the velocity $\boldsymbol{v}_{\mathrm{J}} - \boldsymbol{\sigma}$ to the subspace $S \subseteq \mathsf{M}^6$. If the joint allows n_f degrees of relative motion freedom, then $\dim(S) = n_f$ and $\boldsymbol{S}$ is a $6 \times n_f$ matrix.[3] The number of constraints imposed by the joint is $n_c = 6 - n_f$, and the constraint force must lie in the n_c-dimensional subspace $S^\perp \subseteq \mathsf{F}^6$.

---

[3]If $n_f = 0$ then $\boldsymbol{S}$ is a 6×0 matrix. Such quantities follow the basic rules of matrix algebra, plus one extra rule: the product of an $m \times 0$ matrix with a $0 \times n$ matrix is an $m \times n$ matrix of zeros. Thus, the product of the 6×0 matrix $\boldsymbol{S}$ with the 0×1 vector $\dot{\boldsymbol{q}}$ is a 6×1 vector of zeros.

Let $\boldsymbol{f}_{\mathrm{J}}$ be the force transmitted across the joint. We define this to be the force transmitted from the predecessor to the successor; so the force acting on the successor is $\boldsymbol{f}_{\mathrm{J}}$ and the force acting on the predecessor is $-\boldsymbol{f}_{\mathrm{J}}$. This force can be regarded as the sum of an active force and a constraint force. The former comes from actuators, springs, dampers, or any other force element acting at the joint (or between the two bodies), and the latter is the force that imposes the motion constraint. At this point, we have a choice. The constraint force must be an element of the subspace $S^{\perp}$, but there are two ways to handle the active force: it can be treated either as a known spatial force, or as a generalized force mapped onto a subspace of F^6. We shall choose the latter here.

Let T and T_a be the constraint and active force subspaces, respectively. By definition, $T = S^{\perp}$, but T_a can be any subspace that satisfies $T \oplus T_a = \mathsf{F}^6$. We can now express $\boldsymbol{f}_{\mathrm{J}}$ in the form

$$f_{\mathrm{J}} = \boldsymbol{T}_a \boldsymbol{\tau} + \boldsymbol{T}\boldsymbol{\lambda}, \tag{3.34}$$

where $\boldsymbol{T}_a\boldsymbol{\tau}$ is the active force, $\boldsymbol{T}\boldsymbol{\lambda}$ is the constraint force, $\boldsymbol{T}_a$ and $\boldsymbol{T}$ satisfy

$$\boldsymbol{T}_a^{\mathrm{T}} \boldsymbol{S} = \boldsymbol{1} \tag{3.35}$$

and

$$\boldsymbol{T}^{\mathrm{T}} \boldsymbol{S} = \boldsymbol{0}, \tag{3.36}$$

$\boldsymbol{\tau}$ is the generalized force vector corresponding to $\dot{\boldsymbol{q}}$, and $\boldsymbol{\lambda}$ is a vector of constraint force variables. Given the properties of $\boldsymbol{T}_a$ and $\boldsymbol{T}$, we can immediately deduce that

$$\boldsymbol{S}^{\mathrm{T}} \boldsymbol{f}_{\mathrm{J}} = \boldsymbol{\tau} \tag{3.37}$$

and

$$\boldsymbol{T}^{\mathrm{T}} \boldsymbol{v}_{\mathrm{J}} = \boldsymbol{T}^{\mathrm{T}} \boldsymbol{\sigma}. \tag{3.38}$$

Equation 3.37 is the formula for obtaining the generalized force at a joint from the spatial force transmitted across it; and Eq. 3.38 is the implicit form of the velocity constraint equation.

The justification for Eq. 3.35 is as follows. The total power delivered by the joint is $\boldsymbol{f}_{\mathrm{J}} \cdot \boldsymbol{v}_{\mathrm{J}}$, but this comes from two independent power sources: $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$. The power attributable to $\boldsymbol{\sigma}$ is $\boldsymbol{f}_{\mathrm{J}} \cdot \boldsymbol{\sigma}$, so the power attributable to $\boldsymbol{\tau}$ must be $\boldsymbol{f}_{\mathrm{J}} \cdot \boldsymbol{S}\dot{\boldsymbol{q}}$. Now, this second power must also be equal to $\boldsymbol{\tau} \cdot \dot{\boldsymbol{q}}$ because they are the generalized force and velocity vectors at the joint, so we have the following power balance equation:

$$\boldsymbol{\tau} \cdot \dot{\boldsymbol{q}} = \boldsymbol{f}_{\mathrm{J}} \cdot \boldsymbol{S}\dot{\boldsymbol{q}} = \boldsymbol{\tau}^{\mathrm{T}} \boldsymbol{T}_a^{\mathrm{T}} \boldsymbol{S}\, \dot{\boldsymbol{q}}; \tag{3.39}$$

but this equation must be true for all $\boldsymbol{\tau}$ and $\dot{\boldsymbol{q}}$, so we must have $\boldsymbol{T}_a^{\mathrm{T}} \boldsymbol{S} = \boldsymbol{1}$.

The joint's acceleration constraint equations are obtained directly from Eqs. 3.32 and 3.38 by differentiation. The derivative of Eq. 3.32 is

$$
\begin{aligned}
\boldsymbol{a}_\mathrm{J} &= \boldsymbol{S}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{S}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{\sigma}} \\
&= \boldsymbol{S}\ddot{\boldsymbol{q}} + (\mathring{\boldsymbol{S}} + \boldsymbol{v}_s \times \boldsymbol{S})\dot{\boldsymbol{q}} + (\mathring{\boldsymbol{\sigma}} + \boldsymbol{v}_s \times \boldsymbol{\sigma}) \\
&= \boldsymbol{S}\ddot{\boldsymbol{q}} + \mathring{\boldsymbol{S}}\dot{\boldsymbol{q}} + \mathring{\boldsymbol{\sigma}} + \boldsymbol{v}_s \times (\boldsymbol{S}\dot{\boldsymbol{q}} + \boldsymbol{\sigma}) \\
&= \boldsymbol{S}\ddot{\boldsymbol{q}} + \boldsymbol{c}_\mathrm{J} + \boldsymbol{v}_s \times \boldsymbol{v}_\mathrm{J} \,,
\end{aligned}
\tag{3.40}
$$

where $\mathring{\boldsymbol{S}}$ and $\mathring{\boldsymbol{\sigma}}$ are the apparent derivatives of $\boldsymbol{S}$ and $\boldsymbol{\sigma}$ in a coordinate system that is moving with the successor body, and

$$
\boldsymbol{c}_\mathrm{J} = \mathring{\boldsymbol{S}}\dot{\boldsymbol{q}} + \mathring{\boldsymbol{\sigma}} \,.
\tag{3.41}
$$

(Actually, $\boldsymbol{c}_\mathrm{J} = \mathring{\boldsymbol{v}}_\mathrm{J}$.) If $\boldsymbol{S}$ and $\boldsymbol{\sigma}$ are functions of $\boldsymbol{q}$ and t, as shown in Eq. 3.32, then the general formulae for $\mathring{\boldsymbol{S}}$ and $\mathring{\boldsymbol{\sigma}}$ are

$$
\mathring{\boldsymbol{S}} = \frac{\partial \boldsymbol{S}}{\partial t} + \sum_{i=1}^{n_p} \frac{\partial \boldsymbol{S}}{\partial q_i} \dot{q}_i
\tag{3.42}
$$

and

$$
\mathring{\boldsymbol{\sigma}} = \frac{\partial \boldsymbol{\sigma}}{\partial t} + \sum_{i=1}^{n_p} \frac{\partial \boldsymbol{\sigma}}{\partial q_i} \dot{q}_i \,,
\tag{3.43}
$$

where n_p is the number of joint position variables.

Before moving on, readers should bear in mind that the relatively complicated equations above apply to a general case that is almost never needed in practice. This is partly because most common joint types have the property that $\mathring{\boldsymbol{S}} = \boldsymbol{0}$ and $\boldsymbol{\sigma} = \boldsymbol{0}$ (implying $\boldsymbol{c}_\mathrm{J} = \boldsymbol{0}$), and partly because on the rare occasions when one encounters an explicit time dependency in a joint (i.e., a joint for which $\boldsymbol{\sigma} \neq \boldsymbol{0}$), it is almost always possible to implement the time-dependent term via the hybrid dynamics algorithms of Chapter 9 rather than using $\boldsymbol{\sigma}$.

Finally, the implicit acceleration constraint equation can be obtained either by differentiating Eq. 3.38 or by multiplying Eq. 3.40 by $\boldsymbol{T}^\mathrm{T}$. Both give the same result, but the latter is easier:

$$
\begin{aligned}
\boldsymbol{T}^\mathrm{T} \boldsymbol{a}_\mathrm{J} &= \boldsymbol{T}^\mathrm{T}(\boldsymbol{S}\ddot{\boldsymbol{q}} + \boldsymbol{c}_\mathrm{J} + \boldsymbol{v}_s \times \boldsymbol{v}_\mathrm{J}) \\
&= \boldsymbol{T}^\mathrm{T}(\boldsymbol{c}_\mathrm{J} + \boldsymbol{v}_s \times \boldsymbol{v}_\mathrm{J}) \,.
\end{aligned}
\tag{3.44}
$$

Example 3.2 *Let us examine how to define an active force subspace. Figure 3.3 shows a system comprising a moving body, B, that is connected to a fixed base via a revolute joint, and a motor, M, that drives this joint via a pinion and gear wheel. The x and y axes of a coordinate frame are also shown, the z axis*
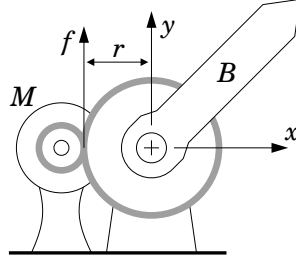
Figure 3.3: Defining an active force subspace

being aligned with the joint's rotation axis. Expressed in these coordinates, the joint's motion and constraint-force subspace matrices are

$$
\boldsymbol{S} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.
$$

Strictly speaking, these are not the only possible values for $\boldsymbol{S}$ and $\boldsymbol{T}$, since any two matrices that span the correct subspaces would do.

The active force is transmitted to B at the point where the two gears touch, and it is a linear force of magnitude f in the y direction. The coordinates of the contact point are $(-r, 0, 0)$, where r is the radius of the gear wheel, so the Plücker coordinates of the active spatial force are $\boldsymbol{f} = [0 \ \ 0 \ \ -rf \ \ 0 \ \ f \ \ 0]^{\mathrm{T}}$. Now, $\boldsymbol{\tau}$ is just a scalar in this example, so if $\boldsymbol{f} = \boldsymbol{T}_a \boldsymbol{\tau}$ then $\boldsymbol{T}_a$ is just a scalar multiple of $\boldsymbol{f}$. From Eq. 3.35 we have that $\boldsymbol{T}_a$ must satisfy $\boldsymbol{T}_a^{\mathrm{T}} \boldsymbol{S} = \boldsymbol{1}$, so the solution is simply

$$
\boldsymbol{T}_a = \boldsymbol{f}/(\boldsymbol{f}^{\mathrm{T}} \boldsymbol{S}) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -1/r \\ 0 \end{bmatrix}.
$$

Observe that both $\boldsymbol{T}_a$ and range($\boldsymbol{T}_a$) depend on r. This dependency has no effect on the equation of motion, but it does affect the value of the constraint force. Physically, the constraint force comes from the joint's bearings. Thus, if the task is to simulate the motion of the system, then r is irrelevant; but if the task is to select appropriate bearings for the joint, then the value of r must be taken into account.
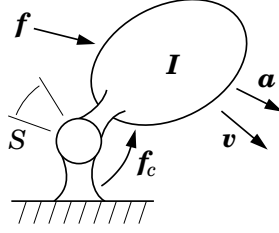
Figure 3.4: A constrained rigid body

3.6 Dynamics of a Constrained Rigid Body

Consider the rigid body shown in Figure 3.4. This body is connected to a fixed base by a joint having a motion subspace of $S \subseteq \mathsf{M}^6$. The joint imposes n_c constraints on the motion of the body, leaving it with n_f degrees of motion freedom, where $n_f = \dim(S)$ and $n_c = 6 - n_f = \dim(S^\perp)$. Three forces act on this body: an applied force, $\boldsymbol{f}$, a constraint force, $\boldsymbol{f}_c$, and a gravity force, $\boldsymbol{f}_g$, which is not shown. The equation of motion for this body is therefore

$$\boldsymbol{f} + \boldsymbol{f}_c + \boldsymbol{f}_g = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v} \,,$$

where $\boldsymbol{I}$, $\boldsymbol{a}$ and $\boldsymbol{v}$ are the body's inertia, acceleration and velocity, respectively. We are not particularly interested in the individual terms $\boldsymbol{f}_g$ and $\boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}$, so we collect them into a single bias-force term, $\boldsymbol{p} = \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v} - \boldsymbol{f}_g$. The equation of motion for the constrained rigid body is then

$$\boldsymbol{f} + \boldsymbol{f}_c = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p} \,, \tag{3.45}$$

subject to the constraints

$$\boldsymbol{v} \in S \qquad \text{and} \qquad \boldsymbol{f}_c \in S^\perp \,. \tag{3.46}$$

We seek a solution to this system of equations in which the acceleration is given as a function of the applied force. Note that $\boldsymbol{f}_c$ is an unknown quantity, and must be evaluated and/or eliminated as part of the solution process.

Method 1

Let us introduce a $6 \times n_f$ matrix, $\boldsymbol{S}$, that spans the subspace S. With this matrix, the constraint equations in Eq. 3.46 can be written in the form

$$\boldsymbol{v} = \boldsymbol{S}\,\dot{\boldsymbol{q}} \tag{3.47}$$

and

$$\boldsymbol{S}^\mathrm{T}\boldsymbol{f}_c = \boldsymbol{0} \,; \tag{3.48}$$

and the acceleration constraint is

$$a = S\,\ddot{q} + \dot{S}\,\dot{q}\,,\qquad(3.49)$$

where $\dot{q}$ and $\ddot{q}$ are vectors of joint velocity and acceleration variables, respectively. Equation 3.48 implies that we can eliminate f_c from Eq. 3.45 by multiplying both sides by S^{T}, giving

$$S^{\mathrm{T}}f = S^{\mathrm{T}}(I\,a + p)\,.\qquad(3.50)$$

Substituting Eq. 3.49 into this equation produces

$$S^{\mathrm{T}}f = S^{\mathrm{T}}(I\,(S\,\ddot{q} + \dot{S}\,\dot{q}) + p)\,,$$

which can be solved for $\ddot{q}$ as follows:

$$\ddot{q} = (S^{\mathrm{T}}IS)^{-1}S^{\mathrm{T}}(f - I\,\dot{S}\,\dot{q} - p)\,.\qquad(3.51)$$

The expression $S^{\mathrm{T}}IS$ is an $n_f \times n_f$ symmetric, positive-definite matrix, and is therefore guaranteed to be invertible. Having found an expression for $\ddot{q}$, the final step is to substitute Eq. 3.51 back into Eq. 3.49, giving

$$a = S\,(S^{\mathrm{T}}IS)^{-1}S^{\mathrm{T}}(f - I\,\dot{S}\,\dot{q} - p) + \dot{S}\,\dot{q}\,.\qquad(3.52)$$

This equation can be written in the form

$$a = \Phi\,f + b\,,\qquad(3.53)$$

where

$$\Phi = S\,(S^{\mathrm{T}}IS)^{-1}S^{\mathrm{T}}\qquad(3.54)$$

and

$$b = \dot{S}\,\dot{q} - \Phi(I\,\dot{S}\,\dot{q} + p)\,.\qquad(3.55)$$

Φ is the constrained body's apparent inverse inertia, and b is its bias acceleration, which is the acceleration it would have if $f = 0$. The apparent inverse inertia describes how the acceleration would change in response to a change in the applied force. It is a symmetric, positive-semidefinite matrix with the following properties: range(Φ) = S, null(Φ) = $S^{\perp}$ and rank(Φ) = n_f.

Method 2

Another way to solve the system of equations in Eqs. 3.45 and 3.46 is to introduce a $6 \times n_c$ matrix, T, that spans $S^{\perp}$. With this matrix, the constraint equations can be written in the form

$$T^{\mathrm{T}}v = 0\qquad(3.56)$$

and

$$f_c = T\,\lambda\,,\qquad(3.57)$$

where $\boldsymbol{\lambda}$ is a vector of constraint force variables, and the acceleration constraint is

$$\boldsymbol{T}^{\mathrm{T}}\boldsymbol{a} + \dot{\boldsymbol{T}}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{0}\,. \tag{3.58}$$

Substituting Eq. 3.57 into Eq. 3.45 gives

$$\boldsymbol{f} + \boldsymbol{T}\,\boldsymbol{\lambda} = \boldsymbol{I}\,\boldsymbol{a} + \boldsymbol{p}\,. \tag{3.59}$$

Equations 3.58 and 3.59 can then be combined into a single equation,

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{T} \\ \boldsymbol{T}^{\mathrm{T}} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} - \boldsymbol{p} \\ -\dot{\boldsymbol{T}}^{\mathrm{T}}\boldsymbol{v} \end{bmatrix}, \tag{3.60}$$

which can be solved for $\boldsymbol{a}$ and $\boldsymbol{\lambda}$. The coefficient matrix in this equation is symmetric and nonsingular, but not positive definite.

Method 3

A third possibility is to derive an equation of motion expressed in generalized coordinates. The vectors $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ can be used as the generalized velocity and acceleration variables of the constrained rigid body, so Eq. 3.51 already gives the generalized acceleration as a function of the applied spatial force, $\boldsymbol{f}$. Let $\boldsymbol{\tau}$ denote the generalized force acting on the body. For $\boldsymbol{\tau}$ to be equivalent to $\boldsymbol{f}$, both forces must deliver the same power for any velocity in S. The power delivered by $\boldsymbol{\tau}$ is $\boldsymbol{\tau}^{\mathrm{T}}\dot{\boldsymbol{q}}$, and that delivered by $\boldsymbol{f}$ is $\boldsymbol{f}^{\mathrm{T}}\boldsymbol{v}$, so we have

$$\boldsymbol{\tau}^{\mathrm{T}}\dot{\boldsymbol{q}} = \boldsymbol{f}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{f}^{\mathrm{T}}\boldsymbol{S}\,\dot{\boldsymbol{q}}\,;$$

but this equation must hold for all $\dot{\boldsymbol{q}}$, so

$$\boldsymbol{\tau} = \boldsymbol{S}^{\mathrm{T}}\boldsymbol{f}\,. \tag{3.61}$$

Substituting this equation into Eq. 3.51 gives

$$\ddot{\boldsymbol{q}} = (\boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{S})^{-1}(\boldsymbol{\tau} - \boldsymbol{S}^{\mathrm{T}}(\boldsymbol{I}\,\dot{\boldsymbol{S}}\,\dot{\boldsymbol{q}} + \boldsymbol{p}))\,,$$

which is the equation of motion for the constrained rigid body, expressed in generalized coordinates. It can be written in the standard form

$$\boldsymbol{H}\,\ddot{\boldsymbol{q}} + \boldsymbol{C} = \boldsymbol{\tau}\,, \tag{3.62}$$

where

$$\boldsymbol{H} = \boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{S} \tag{3.63}$$

is the generalized inertia matrix and

$$\boldsymbol{C} = \boldsymbol{S}^{\mathrm{T}}(\boldsymbol{I}\,\dot{\boldsymbol{S}}\,\dot{\boldsymbol{q}} + \boldsymbol{p}) \tag{3.64}$$

is the generalized bias force.

Equation 3.61 implies a small loss of information in going from $\boldsymbol{f}$ to $\boldsymbol{\tau}$. Infinitely many different values of $\boldsymbol{f}$ map to the same value of $\boldsymbol{\tau}$, and therefore produce the same acceleration, but each produces a different value of $\boldsymbol{f}_c$. If only $\boldsymbol{\tau}$ is known, then it is possible to work out the value of the sum $\boldsymbol{f} + \boldsymbol{f}_c$, but not the values of the individual vectors. If the objective is motion simulation, then the lost information is irrelevant; but if the objective is to design a mechanical system, then the lost information is relevant to tasks like calculating the dynamic load forces on the joint bearings.

3.7   Dynamics of a Multibody System

We conclude this chapter by developing an equation of motion for a system containing multiple bodies and joints. This analysis shows the connection between the joint-related material in Sections 3.5 and 3.6 and the generalized constraints in Section 3.2, by showing how the latter can be constructed from the former.

Suppose we have a rigid-body system containing the following elements: a fixed base, which we regard as body number zero; N_B mobile rigid bodies, which are numbered 1 to N_B; and N_J joints, which are numbered 1 to N_J. For each joint j, we define $p(j)$ and $s(j)$ to be the body numbers of the predecessor and successor bodies of that joint. This set of $2N_J$ numbers defines the *connectivity* of the system. For the analysis to follow, we make no restrictions on the connectivity.

In formulating the equation of motion, we shall follow Method 2 in Section 3.6, but with extra steps where necessary because of the greater complexity of the current system.

Step 1: Body Equations of Motion

We assume that the only forces acting on body i are the force of gravity, $\boldsymbol{f}_{gi}$, and forces from the joints that are connected to it. The equation of motion for body i can then be written

$$\boldsymbol{f}_i = \boldsymbol{I}_i \, \boldsymbol{a}_i + \boldsymbol{p}_i \,,$$

where $\boldsymbol{f}_i$ is the sum of the joint forces acting on body i, and $\boldsymbol{p}_i = \boldsymbol{v}_i \times^* \boldsymbol{I}_i \boldsymbol{v}_i - \boldsymbol{f}_{gi}$ is the bias force. The equations for all N_B bodies can be assembled into a single equation,

$$\begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \vdots \\ \boldsymbol{f}_{N_B} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{I}_{N_B} \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_{N_B} \end{bmatrix} + \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \vdots \\ \boldsymbol{p}_{N_B} \end{bmatrix}. \tag{3.65}$$

We can express this equation more compactly as

$$\boldsymbol{f} = \boldsymbol{I} \, \boldsymbol{a} + \boldsymbol{p} \,, \tag{3.66}$$

where $\boldsymbol{f}$, $\boldsymbol{a}$ and $\boldsymbol{p}$ are $6N_B$-dimensional vectors, and $\boldsymbol{I}$ is a $6N_B \times 6N_B$ block-diagonal matrix. For completeness, we also define $\boldsymbol{v} = [\boldsymbol{v}_1^{\mathrm{T}}\ \boldsymbol{v}_2^{\mathrm{T}}\ \cdots\ \boldsymbol{v}_{N_B}^{\mathrm{T}}]^{\mathrm{T}}$, where $\boldsymbol{v}_i$ is the velocity of body i, as this vector will be needed in the joint constraint equation.

Step 2: Joint Motion from Body Motion

The vector $\boldsymbol{a}$ in Eq. 3.66 is a vector of body accelerations; but the constraints imposed by the joints are expressed in terms of joint velocities and accelerations. Therefore, we need a formula to express joint motion in terms of body motion. Let $\boldsymbol{v}_{\mathrm{J}j}$ and $\boldsymbol{a}_{\mathrm{J}j}$ denote the velocity and acceleration, respectively, across joint j; and let $\boldsymbol{v}_{\mathrm{J}}$ and $\boldsymbol{a}_{\mathrm{J}}$ be the $6N_J$-dimensional vectors formed by concatenating $\boldsymbol{v}_{\mathrm{J}1} \cdots \boldsymbol{v}_{\mathrm{J}N_J}$ and $\boldsymbol{a}_{\mathrm{J}1} \cdots \boldsymbol{a}_{\mathrm{J}N_J}$, respectively. By definition, $\boldsymbol{v}_{\mathrm{J}j}$ and $\boldsymbol{a}_{\mathrm{J}j}$ are the velocity and acceleration of the joint's successor body relative to its predecessor; so

$$\boldsymbol{v}_{\mathrm{J}j} = \boldsymbol{v}_{s(j)} - \boldsymbol{v}_{p(j)}$$

and

$$\boldsymbol{a}_{\mathrm{J}j} = \boldsymbol{a}_{s(j)} - \boldsymbol{a}_{p(j)}\,.$$

These equations can be collected together and expressed as follows:

$$\boldsymbol{v}_{\mathrm{J}} = \begin{bmatrix} \boldsymbol{v}_{\mathrm{J}1} \\ \vdots \\ \boldsymbol{v}_{\mathrm{J}N_J} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{s(1)} - \boldsymbol{v}_{p(1)} \\ \vdots \\ \boldsymbol{v}_{s(N_J)} - \boldsymbol{v}_{p(N_J)} \end{bmatrix} = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{v} \tag{3.67}$$

and

$$\boldsymbol{a}_{\mathrm{J}} = \begin{bmatrix} \boldsymbol{a}_{\mathrm{J}1} \\ \vdots \\ \boldsymbol{a}_{\mathrm{J}N_J} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_{s(1)} - \boldsymbol{a}_{p(1)} \\ \vdots \\ \boldsymbol{a}_{s(N_J)} - \boldsymbol{a}_{p(N_J)} \end{bmatrix} = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{a}\,, \tag{3.68}$$

where

$$\boldsymbol{P}_{ij} = \begin{cases} \mathbf{1}_{6\times6} & \text{if } i = s(j) \\ -\mathbf{1}_{6\times6} & \text{if } i = p(j) \\ \mathbf{0}_{6\times6} & \text{otherwise.} \end{cases} \tag{3.69}$$

$\boldsymbol{P}$ is therefore a $6N_B \times 6N_J$ matrix, which is organized as an $N_B \times N_J$ block matrix of 6×6 blocks, and it maps the vectors of collected body velocities and accelerations ($\boldsymbol{v}$ and $\boldsymbol{a}$) to the vectors of collected joint velocities and accelerations ($\boldsymbol{v}_{\mathrm{J}}$ and $\boldsymbol{a}_{\mathrm{J}}$). Each row[4] in $\boldsymbol{P}$ corresponds to a body, and each column to a joint. (Observe that we are using i as a body-number index and j as a joint-number index.) Each column contains at most two nonzero blocks: a $\mathbf{1}$ in the row corresponding to the joint's successor body and a $-\mathbf{1}$ in the row corresponding to its predecessor. If a joint connects between two moving bodies, then it will have exactly two nonzero blocks in its column, as just described.

---

[4]Strictly speaking, we mean *block row*, and similarly for columns.

However, if it connects to or from the fixed base (body 0), then its column will contain only one nonzero block: a $\mathbf{1}$ or $-\mathbf{1}$, as appropriate, in the row for the moving body it connects to/from. Each row i will contain a $\mathbf{1}$ for each joint that has body i as its successor, and a $-\mathbf{1}$ for each joint that has body i as its predecessor.

Step 3: Body Forces from Joint Forces

Let $\boldsymbol{f}_{\mathrm{J}j}$ be the force transmitted across joint j, and let $\boldsymbol{f}_{\mathrm{J}}$ be the $6N_{\mathrm{J}}$-dimensional vector formed by concatenating the vectors $\boldsymbol{f}_{\mathrm{J}1} \cdots \boldsymbol{f}_{\mathrm{J}N_{\mathrm{J}}}$. By definition, $\boldsymbol{f}_{\mathrm{J}j}$ is the force transmitted from body $p(j)$ to body $s(j)$ through joint j. If we were to define the sets $s^{-1}(i)$ and $p^{-1}(i)$ to be the sets of all joints having body i as their successor or predecessor, respectively, then the vector $\boldsymbol{f}_i$ (which was defined earlier to be the sum of all joint forces acting on body i) could be expressed in the following manner:

$$\boldsymbol{f}_i = \sum_{j \in s^{-1}(i)} \boldsymbol{f}_{\mathrm{J}j} - \sum_{j \in p^{-1}(i)} \boldsymbol{f}_{\mathrm{J}j}.$$

However, on comparing this expression with the description of row i of $\boldsymbol{P}$, it is clear that we could also express $\boldsymbol{f}_i$ as follows:

$$\boldsymbol{f}_i = \sum_{j=1}^{N_{\mathrm{J}}} \boldsymbol{P}_{ij} \boldsymbol{f}_{\mathrm{J}j}.$$

Therefore

$$\boldsymbol{f} = \boldsymbol{P} \boldsymbol{f}_{\mathrm{J}}. \tag{3.70}$$

Step 4: Motion Constraints

Let $\boldsymbol{T}_j$ be the $6 \times n_{cj}$ matrix that spans the subspace $S_j^{\perp}$, where S_j is the motion subspace for joint j and n_{cj} is the number of constraints it imposes. The velocity constraint equation for joint j can then be written

$$\boldsymbol{T}_j^{\mathrm{T}} \boldsymbol{v}_{\mathrm{J}j} = \boldsymbol{0},$$

and the acceleration constraint is

$$\boldsymbol{T}_j^{\mathrm{T}} \boldsymbol{a}_{\mathrm{J}j} + \dot{\boldsymbol{T}}_j^{\mathrm{T}} \boldsymbol{v}_{\mathrm{J}j} = \boldsymbol{0}.$$

Collecting together the acceleration constraints gives

$$\boldsymbol{T}^{\mathrm{T}} \boldsymbol{a}_{\mathrm{J}} + \dot{\boldsymbol{T}}^{\mathrm{T}} \boldsymbol{v}_{\mathrm{J}} = \boldsymbol{0}, \tag{3.71}$$

where $\boldsymbol{T}$ is the $6N_{\mathrm{J}} \times n_c$ block-diagonal matrix having $\boldsymbol{T}_j$ in its jth diagonal block. n_c is the total number of constraints imposed by the joints, and is given by $n_c = \sum_{j=1}^{N_{\mathrm{J}}} n_{cj}$. Combining Eq. 3.71 with Eqs. 3.67 and 3.68 yields the following constraint on the body accelerations:

$$\boldsymbol{T}^{\mathrm{T}} \boldsymbol{P}^{\mathrm{T}} \boldsymbol{a} + \dot{\boldsymbol{T}}^{\mathrm{T}} \boldsymbol{P}^{\mathrm{T}} \boldsymbol{v} = \boldsymbol{0}. \tag{3.72}$$

Step 5: Constraint Forces

$\boldsymbol{f}_{Jj}$ is the sum of an active force and a constraint force, and it can be expressed in the form

$$\boldsymbol{f}_{Jj} = \boldsymbol{T}_{aj}\boldsymbol{\tau}_j + \boldsymbol{T}_j\boldsymbol{\lambda}_j$$

(see Eq. 3.34). In this equation, $\boldsymbol{T}_{aj}$ is the $6 \times n_{fj}$ matrix that spans the active-force subspace for joint j, $\boldsymbol{\tau}_j$ is the n_{fj}-dimensional vector of generalized forces at joint j, $\boldsymbol{\lambda}_j$ is the n_{cj}-dimensional vector of constraint-force variables, and $n_{fj} = 6 - n_{cj}$ is the number of motion freedoms allowed by joint j. Collecting the equations for every joint, we have

$$\boldsymbol{f}_J = \boldsymbol{T}_a\boldsymbol{\tau} + \boldsymbol{T}\boldsymbol{\lambda}, \tag{3.73}$$

where $\boldsymbol{\tau}$ and $\boldsymbol{\lambda}$ are the vectors formed by concatenating $\boldsymbol{\tau}_1 \cdots \boldsymbol{\tau}_{N_J}$ and $\boldsymbol{\lambda}_1 \cdots \boldsymbol{\lambda}_{N_J}$, respectively, and $\boldsymbol{T}_a$ is the block-diagonal matrix having $\boldsymbol{T}_{aj}$ on its jth diagonal block.

Step 6: Final Equation

Substituting Eqs. 3.70 and 3.73 into 3.66 produces the following equation of motion:

$$\boldsymbol{P}(\boldsymbol{T}_a\boldsymbol{\tau} + \boldsymbol{T}\boldsymbol{\lambda}) = \boldsymbol{I}\,\boldsymbol{a} + \boldsymbol{p}\,;$$

and combining this with Eq. 3.72 produces

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{P}\boldsymbol{T} \\ \boldsymbol{T}^{\text{T}}\boldsymbol{P}^{\text{T}} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}\boldsymbol{T}_a\boldsymbol{\tau} - \boldsymbol{p} \\ -\dot{\boldsymbol{T}}^{\text{T}}\boldsymbol{P}^{\text{T}}\boldsymbol{v} \end{bmatrix}. \tag{3.74}$$

This is the equation of motion for the original rigid-body system. The two unknowns are $\boldsymbol{a}$ and $\boldsymbol{\lambda}$, and the equation can be solved uniquely for $\boldsymbol{a}$. If the coefficient matrix is nonsingular, then $\boldsymbol{\lambda}$ is also unique. On comparing the equations here with those in Section 3.2, it can be seen that $\boldsymbol{T}^{\text{T}}\boldsymbol{P}^{\text{T}}$ and $-\dot{\boldsymbol{T}}^{\text{T}}\boldsymbol{P}^{\text{T}}\boldsymbol{v}$ here are equivalent to $\boldsymbol{K}$ and $\boldsymbol{k}$ in Eq. 3.11, that $\boldsymbol{I}$, $\boldsymbol{a}$ and $\boldsymbol{p}$ here are equivalent to $\boldsymbol{H}$, $\ddot{\boldsymbol{q}}$ and $\boldsymbol{C}$ in Eq. 3.14, that $\boldsymbol{P}\boldsymbol{T}_a\boldsymbol{\tau}$ and $\boldsymbol{P}\boldsymbol{T}\boldsymbol{\lambda}$ are equivalent to $\boldsymbol{\tau}$ and $\boldsymbol{\tau}_c$ in Eq. 3.14, and that Eq. 3.74 as a whole is equivalent to Eq. 3.17.

Discussion

Equation 3.74 is a viable basis for a dynamics algorithm, but there are several details that would need to be resolved in the process of designing the algorithm. For example, the velocity and acceleration variables are clearly the Plücker coordinates of the velocity and acceleration vectors of the individual bodies, but in what coordinate system? No mention has been made of the position variables, or of how to calculate $\boldsymbol{I}$, $\boldsymbol{T}$, $\boldsymbol{T}_a\boldsymbol{\tau}$ or $\dot{\boldsymbol{T}}^{\text{T}}\boldsymbol{v}$. Another issue is how to solve Eq. 3.74. The coefficient matrix is large, but it is also sparse, and using a sparse factorization procedure would greatly improve the efficiency. However, if the constraints in Eq. 3.72 are not all linearly independent, then the coefficient

matrix will be singular, in which case it will be necessary to use a solution procedure that works with singular matrices. Finally, there is the problem of constraint stabilization: as Eq. 3.72 specifies the motion constraints at the acceleration level, it is possible for errors to accumulate in the position and velocity constraints, due to numerical rounding and truncation errors during simulation. This problem is discussed in Section 8.3.

Chapter 4

Modelling Rigid Body Systems

A rigid-body system is an assembly of component parts. Those components include rigid bodies, joints and various force elements. The joints are responsible for the kinematic constraints in the system, so we interpret the term 'joint' broadly to mean any possible kinematic relationship between a pair of rigid bodies. In this chapter, we look at ways of describing a rigid-body system in terms of its component parts. In particular, we seek to describe systems consisting solely of bodies and joints. Such a description can be called a *system model*, on the grounds that it describes the system itself, rather than some aspect of its behaviour.

The role of the system model can be understood as follows. Suppose we wish to calculate the dynamics of a given rigid-body system on a computer. There are basically two ways to proceed:

1. work out the symbolic equations of motion of the given system, and translate these equations into computer code; or

2. construct a system model describing the given system, and supply it as an argument to either

    (a) a model-based dynamics calculation routine, or
    (b) a model-based automatic equation (and code) generator.

The second approach is preferable, partly because it is much easier to construct and modify a system model than to work out manually the equations of motion, and partly because a model-based routine can be thoroughly debugged and optimized in anticipation of being used a large number of times. The dynamics algorithms in this book are model-based, and expect to be given a system model along the lines described in this chapter. Approach 2(b) is considered in Section 10.4.
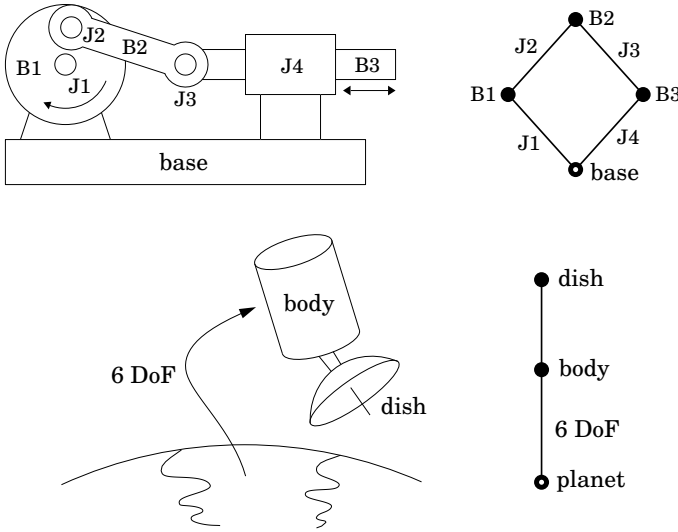
Figure 4.1: Connectivity graphs of two rigid-body systems

4.1   Connectivity

A system model must contain the following data: a description of the component parts, and a description of how they are connected together. The latter is referred to as the system's topology, or connectivity, and it can be expressed in the form of a *connectivity graph* having the following properties:

1. The nodes represent bodies.

2. The arcs represent joints.

3. Exactly one node represents a fixed body, which is known as the base. All other nodes represent moving bodies.

4. The graph is undirected.

5. The graph is connected (i.e., a path exists between any two nodes).

The term 'moving body' includes any body that is capable of moving, whether or not it actually moves. Thus, a body that has been immobilized by kinematic constraints would count as a moving body. From here on, the term and 'graph' should be taken to mean 'undirected graph', unless indicated otherwise.

Some examples of connectivity graphs are shown in Figure 4.1. The first is a simple planar mechanism called a crank-slider linkage. It consists of three moving bodies (B1 to B3), a fixed base, three revolute joints (J1 to J3) and one prismatic joint (J4). Rotation of body B1 (the crank) causes B3 to slide back and forth relative to the base. In the connectivity graph, each node represents

one body, and each arc one joint. It is usually necessary to label at least some of the nodes and arcs, so that the correspondence between the graph and the original mechanism is clear. In addition, it is sometimes useful to make the base node stand out from the others, for easy visual identification. We accomplish this by marking the base node with a white dot. Note that the bodies in a mechanism are usually called *links*.

The second example is a satellite in orbit. The satellite itself consists of two moving bodies and one joint. However, we cannot construct a connectivity graph directly from this two-body system, as rule 3 requires that one node must represent a fixed base. We therefore add a third body—in this case, the planet below—to serve as a fixed base. Having added a third body, rule 5 requires that we also add a second joint, in order to connect the planet to one of the other two bodies. As there is no physical connection between the satellite and the planet, we use a special joint that allows six degrees of motion freedom—a 6-DoF joint. This joint imposes no kinematic constraints, and therefore has no physical effect on the system. Its role is to define additional joint variables, so that the position and velocity variables for the system as a whole can be drawn from the position and velocity variables of the joints it contains. In this case, the satellite has a total of $6 + n_s$ degrees of freedom, where n_s is the degree of freedom allowed by the joint on the satellite. We therefore need a total of $6 + n_s$ velocity variables, and that is exactly the number defined by the two joints.

A body that is connected to a fixed base via a 6-DoF joint is called a *floating base*. In this example, the satellite's body was chosen as the floating base, but we could have chosen the dish instead.

4.1.1 Kinematic Trees and Loops

A graph is a topological tree if there exists exactly one path between any two nodes in the graph. If the connectivity graph of a rigid-body system is a topological tree, then we call the system itself a *kinematic tree*. The satellite in Figure 4.1 is a kinematic tree, but the crank-slider linkage is not. Kinematic trees have special properties that make it easy to calculate their dynamics efficiently.

Let G be any connectivity graph. A *spanning tree* of G, denoted G_t, is a subgraph of G containing all of the nodes in G, together with any subset of the arcs in G such that G_t is a topological tree. Every connectivity graph has at least one spanning tree. If G itself is a tree, then $G_t = G$; otherwise, G_t is a proper subgraph of G and is not unique. Figure 4.2 shows some examples of spanning trees.

A topological tree with n nodes has $n - 1$ arcs. Therefore, if G contains n_n nodes and n_a arcs, then G_t will contain n_n nodes and $n_n - 1$ arcs, implying that there will be $n_a - n_n + 1$ arcs that are in G but not in G_t. These non-tree arcs are called *chords*. Although the number of chords in G is fixed, the set of arcs that happen to be chords will vary with the choice of G_t. This variation is illustrated in Figure 4.2. (We distinguish chords from tree arcs by drawing them with dashed lines.)
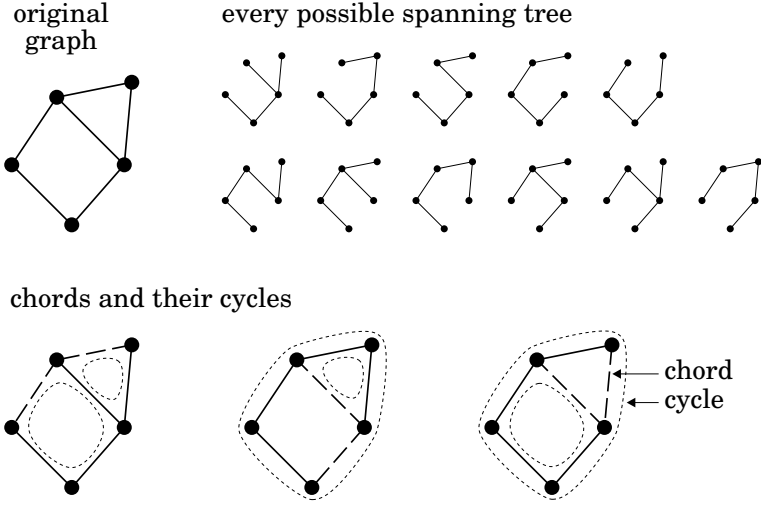
Figure 4.2: Spanning trees, chords and cycles

A cycle is a closed path in a graph. There are no cycles in a tree, so every cycle in a graph must traverse at least one chord. We are particularly interested in those cycles that traverse exactly one chord. There is always exactly one such cycle for each chord in the graph, but its route depends on the choice of spanning tree. For example, the graph in Figure 4.2 has two chords and three cycles. For any choice of spanning tree, two of these cycles are one-chord cycles, and the third (which is not shown) traverses both chords. The path of a one-chord cycle consists of the one chord it traverses, together with the unique path in the spanning tree between the two nodes connected by the chord.

Each cycle in a connectivity graph defines a *kinematic loop*. However, we are not interested in them all, but only in the *independent kinematic loops*. This is the minimal set of loops that must be taken into account when formulating the equations of motion. The importance of the one-chord cycles is that they identify the set of independent kinematic loops. If we describe a rigid-body system as having n kinematic loops, then what we really mean is that it has n independent kinematic loops.

Consider a rigid-body system comprising a fixed base, N_B moving bodies and N_J joints, the latter including any 6-DoF joints that were added to ensure that the connectivity graph was connected. Let G be the connectivity graph of this system, and let G_t be a spanning tree of G. Both graphs will contain $N_B + 1$ nodes. However, G will contain N_J arcs, whereas G_t will contain only N_B, so the number of chords in the system is $N_J - N_B$. The number of independent kinematic loops in the system, N_L, equals the number of chords in G, and is therefore given by the formula

$$N_L = N_J - N_B \,. \tag{4.1}$$

Applying this formula to the crank-slider linkage in Figure 4.1 reveals that it has one kinematic loop.

If a system has kinematic loops, then it is called a *closed-loop system*, otherwise it is a kinematic tree. The joints associated with the chords are known as *loop joints*, or loop-closing joints, and those that are part of the spanning tree are *tree joints*. A rigid-body system therefore has N_B tree joints and N_L loop joints.

4.1.2 Numbering

Bodies and joints can be identified in various ways, but the most convenient way is to number them. We shall use a numbering scheme that is well suited to describing and implementing dynamics algorithms. According to this scheme, the bodies are numbered from 0 to N_B, the joints from 1 to N_J, and the numbers are assigned according to the following rules.

1. Choose a spanning tree, G_t.

2. Assign the number 0 to the node representing the fixed base, and define this node to be the root node of G_t.

3. Number the remaining nodes from 1 to N_B in any order such that each node has a higher number than its parent in G_t.

4. Number the arcs in G_t from 1 to N_B such that arc i connects between node i and its parent.

5. Number all remaining arcs from $N_B + 1$ to N_J in any order.

6. Each body gets the same number as its node, and each joint gets the same number as its arc.

This scheme is called *regular numbering*. If these rules are applied to an unbranched kinematic tree (a tree in which no node has more than one child), then the bodies and joints are numbered consecutively from the base, as shown in Figure 4.3. In all other cases, the numbering is not unique. For example, Figure 4.4 shows every possible numbering of the crank-slider linkage in Figure 4.1. These eight possibilities arise from four possible spanning trees. Two trees are unbranched, and therefore have only one numbering. These two appear in the top left and bottom right graphs in the figure. The other two are branched at the root node, and they have three numberings each.

4.1.3 Joint Polarity

Each joint connects between two bodies. We identify one of these bodies as the joint's *predecessor*, and the other as the joint's *successor*. The joint itself is said to connect *from* its predecessor *to* its successor. The purpose of this terminology
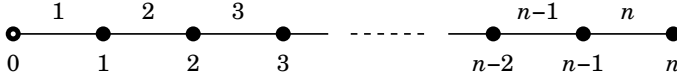
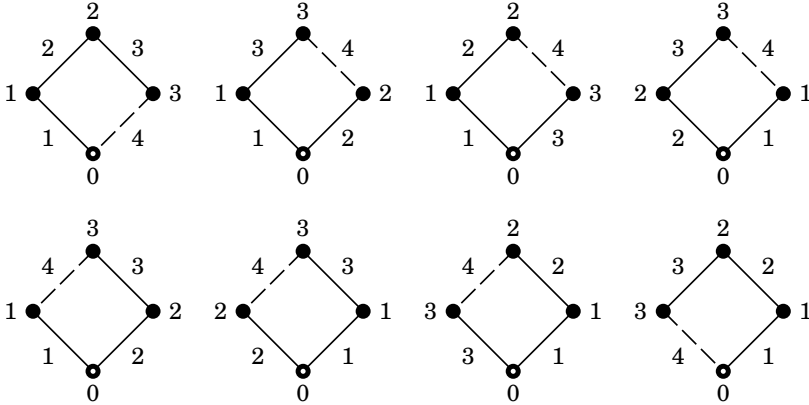Figure 4.3: Unique numbering of an unbranched kinematic tree

Figure 4.4: Every possible numbering of the crank-slider linkage in Figure 4.1

is twofold: to define the relationship between joint and body variables, and to specify which way round the joint is connected in the system. For example, the velocity across a joint is defined to be the velocity of the successor relative to the predecessor, and the force transmitted across a joint is defined to be the force transmitted from the predecessor to the successor.

The identification of which way round a joint is connected between two bodies is called its polarity. We can identify the polarity of a joint in a connectivity graph by placing an arrow on the arc such that it points from the predecessor to the successor. Examples of this are shown in Figures 4.5 and 4.6. As a matter of convenience, we adopt the following convention on the polarity of tree joints: the normal polarity for a tree joint is to connect from the parent to the child. All tree joints are assumed to have this polarity unless indicated otherwise.

Some joints are physically symmetrical, meaning that the motion of the successor relative to the predecessor is qualitatively the same as the motion of the predecessor relative to the successor. Joints that do not have this property are asymmetrical. Switching the polarity of a symmetrical joint does not make a material difference to a rigid-body system, but switching the polarity of an asymmetrical joint does. For example, if two bodies are connected by a ball-and-socket joint, which is symmetrical, then it does not really matter which body has the ball and which the socket—if we were to switch them round, then the motion would still be the same. In contrast, if the two bodies are connected by a rack-and-pinion joint, which is asymmetrical, then it does matter which body has the rack and which the pinion. Thus, the polarity of an asymmetrical
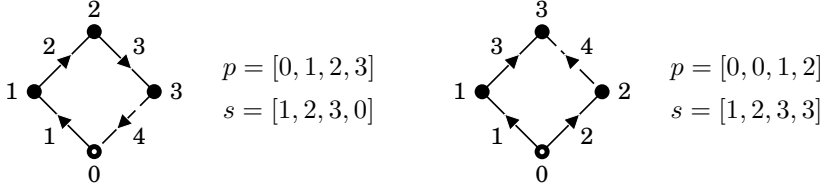
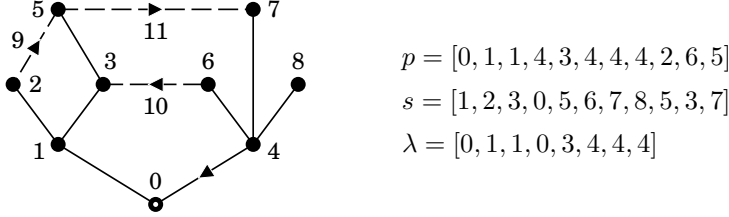Figure 4.5: Joint predecessor and successor arrays

Figure 4.6: Predecessor, successor and parent arrays

joint is determined by the physical system, but the polarity of a symmetrical joint is not, leaving us free to choose its polarity in the system model.

4.1.4   Representation

There are many ways to represent a connectivity graph in a computer. One of the simplest is to record the joint predecessor and successor body numbers in a pair of arrays, p and s, such that $p(i)$ and $s(i)$ are the predecessor and successor body numbers, respectively, of joint i. This method is illustrated in Figure 4.5, which reproduces two of the connectivity graphs from Figure 4.4, adding arrows to show the polarity of each joint.

Together, p and s provide a complete description of a connectivity graph, from which other useful quantities can be calculated. The first of these is the *parent array*, λ, which identifies the parent of each body in the spanning tree: $\lambda(i)$ is the parent of body i. According to rules 3 and 4 of the numbering scheme, tree joint i connects between bodies i and $\lambda(i)$, and $\lambda(i) < i$, so

$$\lambda(i) = \min(p(i), s(i)). \qquad (1 \le i \le N_B) \qquad (4.2)$$

The parent arrays for the two graphs in Figure 4.5 are $\lambda = [0, 1, 2]$ and $\lambda = [0, 0, 1]$; and a larger example is shown in Figure 4.6.

If a tree joint satisfies $s(i) = i$, then it connects from the parent to the child, and is said to be in the forward direction in the tree. Otherwise, it connects from the child to the parent, and is said to be in the reverse direction. The tree joints in Figure 4.5 are all in the forward direction. (The arrows point away from the root.) The tree joints in Figure 4.6 are also all in the forward direction, except for joint 4. The tree-joint numbers and forward-pointing arrows have been

omitted from this diagram: the former because they can be deduced from the numbering scheme, and the latter because we regard 'forward' as the standard direction for a tree joint.

Three more quantities provide useful information about the connectivity of a kinematic tree: for any body i except the base, $\kappa(i)$ is the set of all tree joints on the path between body i and the base; and for any body i, including the base, $\mu(i)$ is the set of children of body i, and $\nu(i)$ is the set of bodies in the subtree starting at body i. They are called the support, child and subtree sets, respectively. (A tree joint is said to support a body if it lies on the path between that body and the base.) $\nu(i)$ can also be regarded as the set of all bodies supported by joint i. For the spanning tree in Figure 4.6, the support, child and subtree sets are as follows:

$$
\begin{array}{lll}
\kappa(1) = \{1\} & \mu(0) = \{1,4\} & \nu(0) = \{0,1,2,\ldots,8\} \\
\kappa(2) = \{1,2\} & \mu(1) = \{2,3\} & \nu(1) = \{1,2,3,5\} \\
\kappa(3) = \{1,3\} & \mu(2) = \{\} & \nu(2) = \{2\} \\
\;\;\vdots & \;\;\vdots & \;\;\vdots \\
\kappa(8) = \{4,8\} & \mu(8) = \{\} & \nu(8) = \{8\}
\end{array}
$$

κ, λ, μ and ν obviously have a lot of simple properties that follow from their definitions. For example, $\mu(i) \subseteq \nu(i)$, $j \in \kappa(i) \Rightarrow i \in \nu(j)$, and so on. One not-so-obvious property that turns out to be quite useful is the identity

$$
\sum_{i=1}^{N_B} \sum_{j \in \kappa(i)} (\cdots) = \sum_{j=1}^{N_B} \sum_{i \in \nu(j)} (\cdots). \tag{4.3}
$$

The left-hand side is a summation over all i,j pairs in which joint j supports body i, but the right-hand side is also a summation over all i,j pairs in which joint j supports body i, and so the two are the same.

Example 4.1 *The role of λ in dynamics algorithms is illustrated by the following simple example. Suppose we wish to calculate the velocity of every body in a rigid-body system, given the velocities across the tree joints. Let $\boldsymbol{v}_i$ and $\boldsymbol{v}_{\mathrm{J}i}$ be the velocity of body i and the velocity across joint i, respectively, the latter being the velocity of the child relative to the parent (i.e., the joints are in the forward direction). Now, the velocity of body i can be defined recursively as the sum of its parent's velocity and the velocity across the joint connecting it to its parent; so the formula for calculating the body velocities is*

$$
\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i}. \qquad (\boldsymbol{v}_0 = \boldsymbol{0})
$$

The algorithm for calculating these velocities is therefore

$$
\begin{aligned}
&\boldsymbol{v}_0 = \boldsymbol{0} \\
&\textbf{for } i = 1 \textbf{ to } N_B \textbf{ do} \\
&\qquad \boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i} \\
&\textbf{end}
\end{aligned}
$$

The property $\lambda(i) < i$ ensures that $\boldsymbol{v}_{\lambda(i)}$ is always calculated before it is used. Another way to express the body velocities is

$$\boldsymbol{v}_i = \sum_{j \in \kappa(i)} \boldsymbol{v}_{\mathrm{J}j} \,. \tag{4.4}$$

Example 4.2 *Example 2.3 introduced a matrix called the* body Jacobian, *which gives the spatial velocity of an individual body as a function of the joint velocity variables. If $\boldsymbol{J}_i$ is the body Jacobian for body i, then*

$$\boldsymbol{v}_i = \boldsymbol{J}_i \, \dot{\boldsymbol{q}} \,.$$

Equation 2.18 gave a formula for $\boldsymbol{J}_i$ that is valid for the special case of an unbranched kinematic chain, so let us now obtain the general formula. Let the velocity across joint j be $\boldsymbol{S}_j \dot{\boldsymbol{q}}_j$, and let ϵ_{ij} be defined as follows:

$$\epsilon_{ij} = \begin{cases} 1 & \text{if } j \in \kappa(i) \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

So ϵ_{ij} is nonzero if joint j supports body i. The velocity of body i can now be expressed as follows:

$$\boldsymbol{v}_i = \sum_{j \in \kappa(i)} \boldsymbol{S}_j \dot{\boldsymbol{q}}_j = \sum_{j=1}^{N_B} \epsilon_{ij} \boldsymbol{S}_j \dot{\boldsymbol{q}}_j = \begin{bmatrix} \epsilon_{i1} \boldsymbol{S}_1 & \cdots & \epsilon_{iN_B} \boldsymbol{S}_{N_B} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}}_1 \\ \vdots \\ \dot{\boldsymbol{q}}_{N_B} \end{bmatrix} \,.$$

Thus, the general formula for the body Jacobian is

$$\boldsymbol{J}_i = \begin{bmatrix} \epsilon_{i1} \boldsymbol{S}_1 & \epsilon_{i2} \boldsymbol{S}_2 & \cdots & \epsilon_{iN_B} \boldsymbol{S}_{N_B} \end{bmatrix} \,. \tag{4.6}$$

4.2 Geometry

If two bodies are connected by a joint, then their relative motion is constrained. A complete description of the constraint requires knowledge of two things: the motion allowed by the joint, and the location of the joint relative to each body. The former is described by a joint model. The latter is described by the system's geometry.

A geometric model of a rigid-body system is a description of where the connections are. In particular, it is a description of the relative position of each joint on each rigid body. Figure 4.7 shows a geometric model for a rigid-body system comprising a fixed base, B_0, three moving bodies, B_1 to B_3, three tree joints, J_1 to J_3, and one loop joint, J_4. The dashed lines in this figure indicate only the connectivity of each joint, not its exact physical location.

Each moving body has a coordinate frame embedded in it, which defines a local coordinate system for that body. These frames are labelled F_1 to F_3,
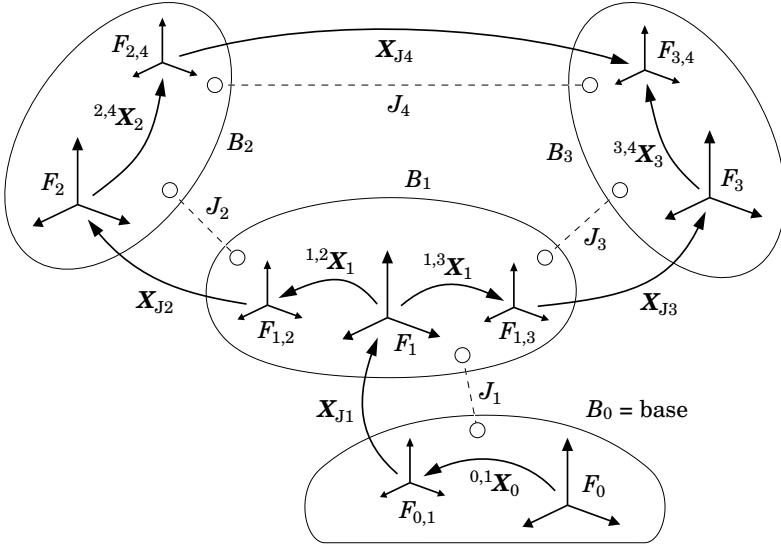
Figure 4.7: Geometric model of a rigid-body system

and the coordinate systems they define are known as *body coordinates*, or *link coordinates*. There is also a frame F_0, embedded in the base, which defines an absolute, reference, or base coordinate system for the whole rigid-body system. The position of body B_i is defined as the position of F_i relative to F_0, and it can be represented by the coordinate transform ${}^iX_0$.

The figure also shows several frames with names of the form $F_{i,j}$ where i refers to a body and j to a joint. These frames identify where the joints are located on each body. Each tree joint i connects from frame $F_{\lambda(i),i}$ to frame F_i (or vice versa if the joint is in the reverse direction), and each loop joint i connects from $F_{p(i),i}$ to $F_{s(i),i}$. There are therefore $N_B + 2N_L$ locating frames in total: one for each tree joint and two for each loop joint. The position of frame $F_{i,j}$ relative to F_i is given by the transform ${}^{i,j}X_i$, which is a constant. These transforms, or a set of data from which they can be calculated, define the geometric model of a rigid-body system.

For convenience, we define an array of tree transforms, X_T, such that $X_T(i) = {}^{\lambda(i),i}X_{\lambda(i)}$. These are the locating transforms for the tree joints, and this array provides a complete description of the geometry of the spanning tree. We also define the arrays X_P and X_S, each indexed from 1 to N_L, such that $X_P(i-N_B) = {}^{p(i),i}X_{p(i)}$ and $X_S(i-N_B) = {}^{s(i),i}X_{s(i)}$ for each loop joint i. These are the locating transforms for the loop joints.

Finally, each joint defines a joint transform, X_{Ji}, which is a function of the joint's position variables. These transforms are discussed in Section 4.4. As shown in the figure, X_{Ji} is equal to ${}^iX_{\lambda(i),i}$ if i is a tree joint, and ${}^{s(i),i}X_{p(i),i}$ if i is a loop joint.

Example 4.3 *Suppose we wish to calculate the position of each body in a kinematic tree; that is, we wish to calculate $^i\boldsymbol{X}_0$ for each body. The algorithm to accomplish this is*

```
for i = 1 to N_B do
    X_J = jcalc(jtype(i), q_i)
    ^iX_λ(i) = X_J X_T(i)
    if λ(i) ≠ 0 then
        ^iX_0 = ^iX_λ(i) ^λ(i)X_0
    end
end
```

In this code fragment, $\boldsymbol{X}_\mathrm{J}$ is a local variable, jtype(i) is a data structure describing joint i, and jcalc is a function that calculates a joint transform from a joint description and a vector of joint position variables. jtype(i) and jcalc are covered in Section 4.4.

4.3   Denavit-Hartenberg Parameters

In general, six parameters are needed to specify the position of one coordinate frame relative to another. However, for certain kinds of rigid-body system, it is possible to locate the body coordinate frames in such a way that fewer parameters are needed. One such method, originally described by Denavit and Hartenberg (Denavit and Hartenberg, 1955), needs only four parameters per joint. It exploits the fact that some of the most common joint types can be characterized by a line in space, and that only four parameters are needed to locate a line.

Figure 4.8 shows how the Denavit-Hartenberg (DH) method is applied to an unbranched kinematic tree representing a robot arm or similar device. The important geometrical features of this system are:

- a base coordinate frame,
- n joint axes, and
- an end-effector coordinate frame, which is embedded in the final link.

The end-effector frame is needed for the correct execution of positioning commands by the robot, but is not needed for dynamics calculations. The DH coordinate frames are placed according to the following rules.

1. Axes z_0 and z_{n+1} are aligned with the z axes of the base and end-effector coordinate frames, respectively.

2. Axes z_1 to z_n are aligned with the n joint axes such that z_i is aligned with the axis of joint i. (If joint i happens to be prismatic, then z_i can be any axis with the correct direction; but it is sensible to choose an axis that intersects with z_{i-1}.)
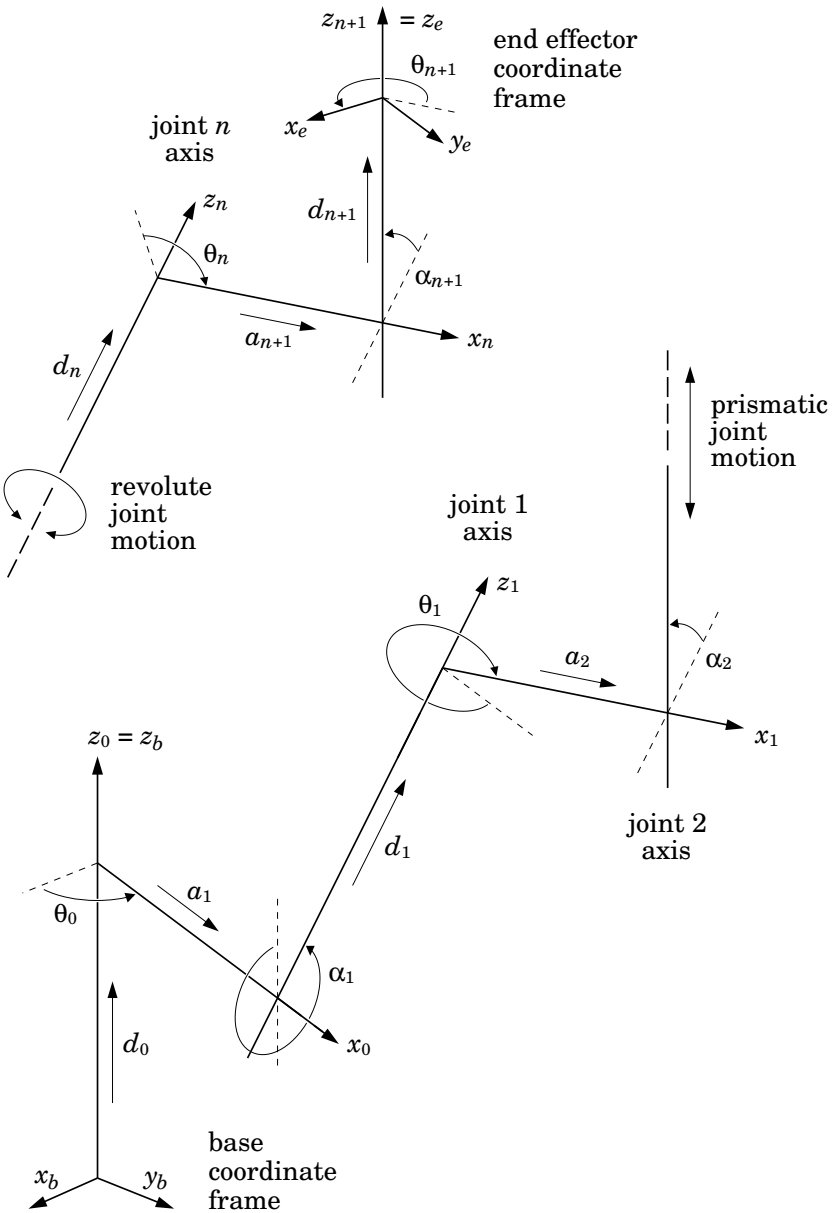
Figure 4.8: Denavit-Hartenberg Coordinate Frames and Parameters

3. Axis x_i is the common perpendicular between z_i and z_{i+1}, directed from z_i to z_{i+1}. If z_i and z_{i+1} intersect, then there is a 180° ambiguity in the direction of x_i. If z_i and z_{i+1} are parallel, then the location of the common perpendicular is indeterminate, and it is reasonable to choose one that intersects with x_{i-1}.

4. The body coordinate frame F_i is defined by the axes x_i, y_i and z_i, where y_i is derived from x_i and z_i to make a right-handed coordinate frame.

Having placed the coordinate frames, their relative locations are described by the following *DH parameters*:

d_i the distance from x_{i-1} to x_i measured along z_i.

θ_i the angle from x_{i-1} to x_i measured about z_i.

a_i the distance from z_{i-1} to z_i measured along x_{i-1}. The definition of x_{i-1} implies $a_i \geq 0$.

α_i the angle from z_{i-1} to z_i measured about x_{i-1}.

All of the parameters a_i and α_i are constants, but some of the parameters d_i and θ_i serve as joint variables. Specifically, if joint i is revolute, then θ_i is its joint variable; if joint i is prismatic, then d_i is its joint variable; if joint i is cylindrical, then both θ_i and d_i are joint variables; and if joint i is a screw joint with a pitch of h_i, then θ_i is the joint variable, and the distance from x_{i-1} to x_i is defined to be $d_i + h_i \theta_i$. If every joint is revolute, then the DH parameters define the following geometric model for the unbranched tree in Figure 4.8:

$$i = 1: \quad \boldsymbol{X}_{\mathrm{T}}(1) = \mathrm{xlt}([a_1 \ 0 \ d_1]^{\mathrm{T}}) \ \mathrm{rotx}(\alpha_1) \ \mathrm{rotz}(\theta_0) \ \mathrm{xlt}([0 \ 0 \ d_0]^{\mathrm{T}})$$

$$i > 1: \quad \boldsymbol{X}_{\mathrm{T}}(i) = \mathrm{xlt}([a_i \ 0 \ d_i]^{\mathrm{T}}) \ \mathrm{rotx}(\alpha_i)$$

(See Table 2.2 for definitions of of the translation and rotation functions. Also, see §A.4 in the appendix for screw transforms.)

It requires a total of $4n+6$ DH parameters to describe a system with n joints, of which at least n will be joint variables rather than parameters. However, the last four parameters serve only to locate the end-effector frame, and can be omitted if this information is not needed. Up to five more parameters can be omitted if the base frame can be chosen to be aligned with z_1.

In its basic form, the Denavit-Hartenberg method is applicable to any rigid-body system in which every joint can be characterized by a joint axis and the topology is either an unbranched kinematic tree or a single closed kinematic loop. However, the method can be adapted to a broader class of systems, as described in Khalil and Dombre (2002). DH parameters are described in various textbooks, such as Angeles (2003) and Craig (2005). The notational details tend to vary from one author to the next, especially the numbering of x_i, a_i and α_i. The notation used here follows the Springer Handbook of Robotics (Siciliano and Khatib, 2008).
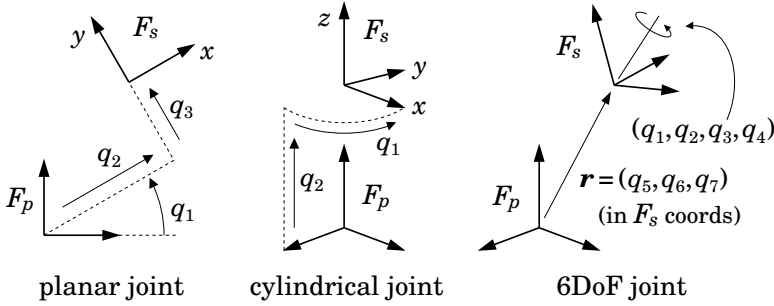
Figure 4.9: Geometrical interpretation of some joint position variables

4.4   Joint Models

A joint can be regarded as a motion constraint between two Cartesian frames—one embedded in the joint's successor body and the other in its predecessor. We shall call these frames F_s and F_p, respectively. The purpose of a joint model is to provide the information required to calculate the motion of F_s relative to F_p as a function of the joint variables.

The kind of motion allowed by a joint depends on its joint type. For example, a revolute joint allows pure rotation about a single axis, whereas a spherical joint allows arbitrary rotation about a specific point. We therefore need one joint model for each type of joint. The contents of a joint model are a collection of data and formulae for calculating specific quantities, such as: the joint transform (^sX_p), the joint velocity (v_J) the joint's motion subspace (S), and so on. Formulae for several common joint types are shown in Table 4.1, and the geometries of some of these joints are shown in Figure 4.9.

Let us examine one model in more detail. The first entry in Table 4.1 describes a revolute joint that allows F_s to rotate relative to F_p about their common z axis, and defines the joint variable to be the angle of rotation. The two columns marked E and r describe the rotational and translational components of the joint transform, and the transform itself is given by the equation $^sX_p = \mathrm{rot}(E)\,\mathrm{xlt}(r)$. (See Table 2.2 for the definitions of rot, xlt and rz.) Observe that F_s coincides with F_p when the joint variable (q_1) is zero. We apply this convention as far as possible to all joint models. Another convention we apply to all joint models is that S, T, and related quantities, are expressed in F_s coordinates. In the case of a revolute joint this does not really matter, since $^sS = {}^pS$, but it does matter for other joints.

In a well-designed dynamics simulator, there would likely be a *joint model library*, containing one model for each type of joint supported by the software. A system model would then contain a *joint type descriptor* for each joint in the system. We shall use the expression jtype(i) to denote the joint type descriptor of joint i. In the simplest case, a joint type descriptor is just a type code identifying a particular model in the library. For example, the type code 'R'

Joint Type	Joint Transform ($^s\boldsymbol{X}_p$)		Motion Subsp. ($\boldsymbol{S}$)	Constraint Force Subspace ($\boldsymbol{T}$)
	$\boldsymbol{E}$	$\boldsymbol{r}$		
Revolute (hinge joint)	$\mathrm{rz}(q_1)$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Prismatic (sliding joint)	$\boldsymbol{1}_{3\times3}$	$\begin{bmatrix} 0 \\ 0 \\ q_1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
Helical (screw joint) with pitch h	$\mathrm{rz}(q_1)$	$\begin{bmatrix} 0 \\ 0 \\ h\,q_1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ h \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -h \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Cylindrical	$\mathrm{rz}(q_1)$	$\begin{bmatrix} 0 \\ 0 \\ q_2 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
Planar (see Fig. 4.9)	$\mathrm{rz}(q_1)$	$\begin{bmatrix} c_1 q_2 - s_1 q_3 \\ s_1 q_2 + c_1 q_3 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Spherical (ball & socket)	$\begin{bmatrix} \text{see} \\ \text{Eq. 4.12} \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
6-DoF Joint (free motion)	$\begin{bmatrix} \text{see} \\ \text{Eq. 4.12} \end{bmatrix}$	$\boldsymbol{E}^{-1}\begin{bmatrix} q_5 \\ q_6 \\ q_7 \end{bmatrix}$	$\boldsymbol{1}_{6\times6}$	

Notes: $c_1 = \cos(q_1)$, $s_1 = \sin(q_1)$, and $^s\boldsymbol{X}_p = \mathrm{rot}(\boldsymbol{E})\ \mathrm{xlt}(\boldsymbol{r})$ (see Table 2.2).

Table 4.1: Joint Model Formulae

could identify a revolute joint and 'P' a prismatic one. However, some joints are characterized by both a type and a set of parameters. For example, a helical joint is characterized both by the fact that it is helical and a parameter that specifies the pitch of the helix. Each individual helical joint in a rigid-body system can have a different pitch; so parameters like this are a property of individual joints, rather than the joint model. To accommodate joint parameters, we define a joint type descriptor to be a data structure containing both a type code and the set of parameter values (if any) required for a joint of that type.

Joint Model Calculation Routine

Subsequent chapters describe several dynamics algorithms, all of which make use of joint model data. To keep the algorithm descriptions simple, we assume the existence of a single function, called jcalc, that calculates all requested joint-related data in a single call. A specimen call to this function looks like this:

$$[\boldsymbol{X}_\mathrm{J}, \boldsymbol{S}, \boldsymbol{v}_\mathrm{J}, \boldsymbol{c}_\mathrm{J}] = \mathrm{jcalc}(type, \boldsymbol{q}, \boldsymbol{\alpha}) \qquad \text{(explicit joint constraint)}$$

$$[\boldsymbol{\delta}, \boldsymbol{T}] = \mathrm{jcalc}(type, {}^s\boldsymbol{X}_p) \qquad \text{(implicit joint constraint)}$$

$$[\dot{\boldsymbol{q}}] = \mathrm{jcalc}(type, \boldsymbol{q}, \boldsymbol{\alpha}) \qquad \text{(for qdfn (Eq. 3.7))}$$

In each case, the expression on the left is a list of requested return values, and the argument *type* refers to a joint type descriptor (e.g. jtype(i)). The use of jcalc to calculate implicit joint constraints is covered in Chapter 8, and its use to calculate $\dot{\boldsymbol{q}}$ is covered later in this chapter. In the algorithm listings, we generally assume that $\boldsymbol{\alpha} = \dot{\boldsymbol{q}}$, and use $\dot{\boldsymbol{q}}$ in place of $\boldsymbol{\alpha}$ in the argument lists. If a joint depends explicitly on time, then time must be added as a final argument in the list. Note that $\boldsymbol{v}_\mathrm{J}$, $\boldsymbol{c}_\mathrm{J}$, $\boldsymbol{S}$ and $\boldsymbol{T}$ are all expressed in F_s coordinates. Expressions for $\boldsymbol{v}_\mathrm{J}$ and $\boldsymbol{c}_\mathrm{J}$ are given in Section 3.5.

Example 4.4 *Example 4.3 showed how to calculate the coordinate transform ${}^i\boldsymbol{X}_0$ for each body in a kinematic tree. We now extend this example to calculate both ${}^i\boldsymbol{X}_0$ and ${}^0\boldsymbol{v}_i$ (the velocity of body i expressed in base coordinates). The algorithm to accomplish this is*

$$
\begin{aligned}
&{}^0\boldsymbol{v}_0 = \boldsymbol{0} \\
&\textbf{for } i = 1 \textbf{ to } N_B \textbf{ do} \\
&\qquad [\boldsymbol{X}_\mathrm{J}, \boldsymbol{v}_\mathrm{J}] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i) \\
&\qquad {}^i\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_\mathrm{J}\,\boldsymbol{X}_\mathrm{T}(i) \\
&\qquad \textbf{if } \lambda(i) \neq 0 \textbf{ then} \\
&\qquad\qquad {}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{X}_0 \\
&\qquad \textbf{end} \\
&\qquad {}^0\boldsymbol{v}_i = {}^0\boldsymbol{v}_{\lambda(i)} + {}^0\boldsymbol{X}_i\boldsymbol{v}_\mathrm{J} \\
&\textbf{end}
\end{aligned}
$$

This code fragment contains a new feature: it calculates $^iX_0$, but then uses ^0X_i to transform v_J. The explanation for this practice can be found in Appendix A, where it is shown how to calculate $X^{-1}v$ directly from X and v without having to calculate X^{-1} first. It is also possible to calculate X^*f and $(X^*)^{-1}f$ directly from X and f without having to calculate X^* or $(X^*)^{-1}$ first. We exploit this fact in several dynamics algorithm listings.

Example 4.5 *A peculiar feature of both the planar joint and the 6-DoF joint in Table 4.1 is that the translational joint variables express the translation in F_s coordinates rather than F_p coordinates. The reason for doing this is to make the motion subspace matrix a simple constant, so that $\mathring{S} = 0$, $c_J = 0$, and the simple form of S allows various optimizations to be applied to the dynamics algorithms. However, there is potentially a big disadvantage in this arrangement: if the successor body is far distant from the predecessor, and starts to spin, then the spinning will cause large rates of change in the position variables. (Just think of a tumbling satellite or aircraft.) One way to get the best of both worlds is to use a set of position variables in which the translations are expressed in F_p coordinates, together with a set of velocity variables in which the linear velocity is expressed in F_s coordinates.*

Let $^pq_1$, $^pq_2$ and $^pq_3$ be a set of position variables for a planar joint, such that $^pq_2$ and $^pq_3$ are translations in the x and y directions of F_p; and let $^sq_1$, $^sq_2$ and $^sq_3$ be a second set of position variables for the same planar joint, such that $^sq_2$ and $^sq_3$ are translations in the x and y directions of F_s. This second set of variables is the set used in Table 4.1 and shown in Figure 4.9. The relationship between these two sets is

$$\begin{bmatrix} ^pq_1 \\ ^pq_2 \\ ^pq_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & -s_1 \\ 0 & s_1 & c_1 \end{bmatrix} \begin{bmatrix} ^sq_1 \\ ^sq_2 \\ ^sq_3 \end{bmatrix}.$$

We wish to use ^pq_i as the position variables, but $^s\dot{q}_i$ as the velocity variables. To cater for this, we need a formula to calculate the derivatives of the position variables from the position and velocity variables. Such a formula is obtained by differentiating the above equation, to get

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -q_3 & c_1 & -s_1 \\ q_2 & s_1 & c_1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix},$$

where we have dropped the superscript p from ^pq_i and $^p\dot{q}_i$, and substituted α_i for $^s\dot{q}_i$. This is the calculation that jcalc must perform if it is called upon to calculate $\dot{q}$ from q and α for this particular model of the planar joint.

Example 4.6 *Let us construct a joint model for the rack-and-pinion joint shown in Figure 4.10. This joint allows a pinion that is part of the successor body to roll along a rack that is part of the predecessor body. The rack*
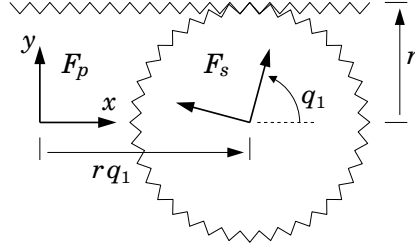
Figure 4.10: Rack-and-pinion joint

is parallel to the x axis of F_p, and the pinion is centred on the z axis of F_s. The pinion has a radius of r, which is a parameter of the joint model. The joint allows only a single degree of motion freedom, so there is only one joint variable, denoted q_1, and it is chosen to be the rotation angle of the pinion.

If F_p and F_s coincide when $q_1 = 0$, then the transform from F_p to F_s consists of a translation by rq_1 in the x direction followed by a rotation of q_1 about the z axis. The joint transform is therefore

$$X_J = {}^sX_p = \begin{bmatrix} E & 0 \\ 0 & E \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -p\times & 1 \end{bmatrix},$$

where

$$E = \mathrm{rz}(q_1) = \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad p = \begin{bmatrix} rq_1 \\ 0 \\ 0 \end{bmatrix}.$$

The next step is to find formulae for v_J and S. We begin by expressing the joint velocity as a function of the joint's position and velocity variables. Relative to F_p, the pinion is rotating with an angular velocity of $\omega = [0\ 0\ \dot{q}_1]^T$ about an axis passing through p, and it is translating with a linear velocity of $\dot{p} = [r\dot{q}_1\ 0\ 0]^T$. This equates to a spatial velocity, expressed in F_p coordinates, of

$$ {}^pv_J = \begin{bmatrix} \omega \\ p \times \omega \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{p} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{q}_1 \end{bmatrix} \\ \begin{bmatrix} rq_1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \dot{q}_1 \end{bmatrix} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ r\dot{q}_1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dot{q}_1 \\ r\dot{q}_1 \\ -rq_1\dot{q}_1 \\ 0 \end{bmatrix}. $$

As there is only a single joint variable, the motion subspace matrix is given by

the formula $\boldsymbol{S} = \boldsymbol{v}_J/\dot{q}_1$. Therefore

$$\boldsymbol{v}_J = {}^s\boldsymbol{X}_p\, {}^p\boldsymbol{v} = \begin{bmatrix} 0 \\ 0 \\ \dot{q}_1 \\ rc_1\dot{q}_1 \\ -rs_1\dot{q}_1 \\ 0 \end{bmatrix} \qquad and \qquad \boldsymbol{S} = \frac{\boldsymbol{v}_J}{\dot{q}_1} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ rc_1 \\ -rs_1 \\ 0 \end{bmatrix}.$$

Finally, as $\boldsymbol{S}$ is not a constant, we need a formula for the velocity-product term $\boldsymbol{c}_J$. This is given by

$$\boldsymbol{c}_J = \mathring{\boldsymbol{S}}\,\dot{q}_1 = \frac{\partial \boldsymbol{S}}{\partial q_1}\,\dot{q}_1^2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -rs_1\dot{q}_1^2 \\ -rc_1\dot{q}_1^2 \\ 0 \end{bmatrix}.$$

(See Eq. 3.42 for $\mathring{\boldsymbol{S}}$.) Observe that the rack-and-pinion joint simplifies to a revolute joint if $r = 0$.

Polarity Reversal

As explained in Section 4.1.4, a joint in a kinematic tree is said to be in the forward direction if the predecessor is the parent of the successor, and in the reverse direction otherwise. If a joint is symmetrical, then its polarity can be chosen at the time the system model is being defined so that it is in the forward direction; but if it is asymmetrical, then its polarity is determined by the rigid-body system being modelled. Thus, it is not possible to guarantee that every tree joint is in the forward direction.

The simplest way to deal with a reverse-direction joint is to convert it to a forward-direction joint. We can do this as follows. First, add a boolean variable to the joint descriptor data structure that takes the value *true* if the joint is a tree joint in the reverse direction, and takes the value *false* otherwise. Given this extra piece of information, jcalc can be modified to simulate a joint of the opposite polarity whenever this boolean variable is true. In effect, jcalc reverses the roles of the predecessor and successor. To accomplish this task, jcalc merely has to modify some of the calculated values before they are returned. For example, instead of returning the matrix ${}^s\boldsymbol{X}_p$ that has been calculated according to the formula for the given joint type, it returns the inverse of this matrix; and, instead of returning the calculated value of $\boldsymbol{v}_J$, it returns $-{}^p\boldsymbol{X}_s\boldsymbol{v}_J$. A complete set of modified return values is shown in Table 4.2. The advantage of this approach is that it allows the dynamics algorithms to assume that all tree joints are in the forward direction. This, in turn, makes the algorithms simpler and easier to implement.

normal return value	simulating opposite polarity
${}^s\boldsymbol{X}_p$	${}^p\boldsymbol{X}_s$
$\boldsymbol{S}$	$-{}^p\boldsymbol{X}_s\boldsymbol{S}$
$\boldsymbol{T}$	$-{}^p\boldsymbol{X}_s\boldsymbol{T}$
$\boldsymbol{v}_{\mathrm{J}}$	$-{}^p\boldsymbol{X}_s\boldsymbol{v}_{\mathrm{J}}$
$\boldsymbol{c}_{\mathrm{J}}$	$-{}^p\boldsymbol{X}_s\boldsymbol{c}_{\mathrm{J}}$
$\dot{\boldsymbol{q}}$	$\dot{\boldsymbol{q}}$

Table 4.2: How jcalc simulates an opposite-polarity joint

4.5   Spherical Motion

There are many ways to represent a general 3D rotation (e.g. see Rooney, 1977), but the two main choices are Euler angles and Euler parameters. The latter are also known as unit quaternions. Euler angles have the advantage that the position variables are a set of three independent angles, and the velocity variables are their derivatives. This makes them relatively easy to implement. Their disadvantage is that they suffer from singularities: as the orientation approaches a singularity, the representation becomes increasingly ill-conditioned, and eventually breaks down altogether. The main advantage of Euler parameters is that they are free of singularities; but their disadvantage is that they are a set of four non-independent position variables, which implies that the velocity variables cannot be the derivatives of the position variables. Software that uses Euler parameters must be written to allow the number of position variables to be different from the number of velocity variables.

Euler Angles

We use the term 'Euler angles' in both a broad sense and a narrow sense. In the broad sense, it refers to any method for representing a general rotation by means of a sequence of three rotations about specified coordinate axes. In the narrow sense, it refers to the special case in which the first and third rotations take place about the z axis, and the second rotation takes place about either the x axis or the y axis.

Every version of Euler angles suffers from singularities. They occur whenever the first and third axes are aligned. If the first and third rotations use the same coordinate axis, then the singularities occur when the second angle is either $0°$ or $180°$; otherwise, they occur when the second angle is $\pm 90°$. In the vicinity of a singularity, small changes in the rotation being represented can induce large changes in the first and third angles. A joint model based on Euler angles is not valid at a singularity.

Let us construct a spherical joint model using yaw, pitch and roll angles. These are a sequence of rotations about the z, y and x axes, in that order.

Thus, the correct orientation of F_s relative to F_p is obtained by first placing F_s coincident with F_p, then rotating it about its z axis by an amount q_1 (yaw), then rotating it about its y axis by an amount q_2 (pitch), and then rotating it about its x axis by an amount q_3 (roll). The formula for $\boldsymbol{E}$ is therefore

$$
\begin{aligned}
\boldsymbol{E} &= \mathrm{rx}(q_3)\ \mathrm{ry}(q_2)\ \mathrm{rz}(q_1) \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_3 & s_3 \\ 0 & -s_3 & c_3 \end{bmatrix} \begin{bmatrix} c_2 & 0 & -s_2 \\ 0 & 1 & 0 \\ s_2 & 0 & c_2 \end{bmatrix} \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_1 c_2 & s_1 c_2 & -s_2 \\ c_1 s_2 s_3 - s_1 c_3 & s_1 s_2 s_3 + c_1 c_3 & c_2 s_3 \\ c_1 s_2 c_3 + s_1 s_3 & s_1 s_2 c_3 - c_1 s_3 & c_2 c_3 \end{bmatrix}.
\end{aligned} \tag{4.7}
$$

The formula for $\boldsymbol{S}$ can be obtained by expressing the joint velocity as a function of $\dot{\boldsymbol{q}}$, and using $\boldsymbol{S} = \partial \boldsymbol{v}_{\mathrm{J}}/\partial \dot{\boldsymbol{q}}$. We shall skip the details, and just present the final result:

$$
\boldsymbol{S} = \begin{bmatrix} -s_2 & 0 & 1 \\ c_2 s_3 & c_3 & 0 \\ c_2 c_3 & -s_3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{4.8}
$$

The columns of $\boldsymbol{S}$ contain the coordinates in F_s of the axes about which the rotations take place: the first column is the z axis of F_p; the second is where the y axis was after the first rotation; and the third is where the x axis was after the first two rotations. As $\boldsymbol{S}$ is not a constant, we also need a formula for $\boldsymbol{c}_{\mathrm{J}}$. This is given by

$$
\begin{aligned}
\boldsymbol{c}_{\mathrm{J}} = \mathring{\boldsymbol{S}}\,\dot{\boldsymbol{q}} &= \begin{bmatrix} -c_2 \dot{q}_2 & 0 & 0 \\ -s_2 s_3 \dot{q}_2 + c_2 c_3 \dot{q}_3 & -s_3 \dot{q}_3 & 0 \\ -s_2 c_3 \dot{q}_2 - c_2 s_3 \dot{q}_3 & -c_3 \dot{q}_3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \\
&= \begin{bmatrix} -c_2 \dot{q}_1 \dot{q}_2 \\ -s_2 s_3 \dot{q}_1 \dot{q}_2 + c_2 c_3 \dot{q}_1 \dot{q}_3 - s_3 \dot{q}_2 \dot{q}_3 \\ -s_2 c_3 \dot{q}_1 \dot{q}_2 - c_2 s_3 \dot{q}_1 \dot{q}_3 - c_3 \dot{q}_2 \dot{q}_3 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{4.9}
\end{aligned}
$$

Equations 4.7 and 4.8 replace the formulae for $\boldsymbol{E}$ and $\boldsymbol{S}$ in Table 4.1, but the formula for $\boldsymbol{T}$ remains unchanged. This is therefore an example of a joint model in which $\boldsymbol{S}$ varies while $\boldsymbol{T}$ remains constant. Such a thing is possible because range($\boldsymbol{S}$) is a constant, and $\boldsymbol{T}$ is required only to satisfy range($\boldsymbol{T}$) = range($\boldsymbol{S}$)$^{\perp}$.

Euler Parameters

A general rotation can also be described by means of a unit vector and an
angle. The unit vector specifies the axis of rotation, and the angle specifies the
magnitude. Euler parameters are closely related to this representation. If $\boldsymbol{u}$
and θ are the unit vector and angle, then the Euler parameters describing the
same rotation are

$$
\begin{aligned}
p_0 &= \cos(\theta/2) \\
p_1 &= \sin(\theta/2)u_x \\
p_2 &= \sin(\theta/2)u_y \\
p_3 &= \sin(\theta/2)u_z \,.
\end{aligned}
\tag{4.10}
$$

They are not independent, but must satisfy the equation

$$
p_0^2 + p_1^2 + p_2^2 + p_3^2 = 1 \,.
\tag{4.11}
$$

A spherical joint must have three independent velocity variables. There-
fore, if we wish to use the four non-independent Euler parameters as position
variables, then the velocity variables cannot be the derivatives of the position
variables. The best choice for the velocity variables is to use the Cartesian
coordinates in F_s of the joint's angular velocity vector, $\boldsymbol{\omega}$. This choice results
in $\boldsymbol{S}$ taking the simple, constant value shown in Table 4.1. We therefore need
only a formula for $\boldsymbol{E}$ and a formula for the derivatives of the position variables.
These are

$$
\boldsymbol{E} = 2 \begin{bmatrix} p_0^2 + p_1^2 - 1/2 & p_1 p_2 + p_0 p_3 & p_1 p_3 - p_0 p_2 \\ p_1 p_2 - p_0 p_3 & p_0^2 + p_2^2 - 1/2 & p_2 p_3 + p_0 p_1 \\ p_1 p_3 + p_0 p_2 & p_2 p_3 - p_0 p_1 & p_0^2 + p_3^2 - 1/2 \end{bmatrix}
\tag{4.12}
$$

and

$$
\begin{bmatrix} \dot{p}_0 \\ \dot{p}_1 \\ \dot{p}_2 \\ \dot{p}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -p_1 & -p_2 & -p_3 \\ p_0 & -p_3 & p_2 \\ p_3 & p_0 & -p_1 \\ -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \,.
\tag{4.13}
$$

A spherical joint model based on these equations is numerically superior to one
based on Eqs. 4.7, 4.8 and 4.9. However, it does place a burden on the sur-
rounding software of having to cope with a joint model in which the number
of position variables differs from the number of velocity variables. Three more
complications are: the values of the position variables must be obtained by in-
tegrating Eq. 4.13 rather than directly integrating the velocity variables; a zero
rotation is defined by a nonzero set of Euler parameters; and it is necessary to
renormalize Euler parameters frequently during the integration process. (Trun-
cation errors during integration cause the parameters to fail to satisfy Eq. 4.11
exactly, and this causes a linear scaling error affecting the whole subtree sup-
ported by the joint. Ideally, they should be renormalized after every integration
step.)

data for a kinematic tree	extra data for a closed-loop system
N_B	N_L
$\lambda(1) \; \cdots \; \lambda(N_B)$	$p(N_B + 1) \; \cdots \; p(N_B + N_L)$
jtype$(1) \; \cdots \;$ jtype(N_B)	$s(N_B + 1) \; \cdots \; s(N_B + N_L)$
$\boldsymbol{X}_{\mathrm{T}}(1) \; \cdots \; \boldsymbol{X}_{\mathrm{T}}(N_B)$	jtype$(N_B + 1) \; \cdots \;$ jtype$(N_B + N_L)$
$\boldsymbol{I}_1 \; \cdots \; \boldsymbol{I}_{N_B}$	$\boldsymbol{X}_{\mathrm{P}}(1) \; \cdots \; \boldsymbol{X}_{\mathrm{P}}(N_L)$
	$\boldsymbol{X}_{\mathrm{S}}(1) \; \cdots \; \boldsymbol{X}_{\mathrm{S}}(N_L)$

Table 4.3: Complete system model

4.6 A Complete System Model

Table 4.3 presents a complete system model for a rigid-body system. For a kinematic tree, the required data are: N_B, the parent array, an array of joint type descriptors, the tree transform array, and an array of rigid-body inertias. The inertias are expressed in body coordinates, in which they are constants. For a closed-loop system, the required extra data are: N_L and, for each loop joint, the predecessor and successor body numbers, the joint type descriptor, and the predecessor and successor transforms.

In the simplest case, a software package might cater for a restricted set of joint types, such as revolute and prismatic joints only. In this case, jtype(i) could simply be a numeric type code, and it would be practical to hard-wire the appropriate values for $\boldsymbol{X}_{\mathrm{J}}$, $\boldsymbol{v}_{\mathrm{J}}$ and $\boldsymbol{S}$ directly into the source code for jcalc, or even directly into the source code of a dynamics algorithm. Another simplification concerns access into vectors of joint variables, such as $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. In the general case, $\boldsymbol{q}_i$ is a subvector of $\boldsymbol{q}$, and its location within $\boldsymbol{q}$ depends on the sizes of all of the preceding subvectors. Thus, the code that accesses and updates $\boldsymbol{q}_i$ should have an array of offsets, one per joint, indicating which element of $\boldsymbol{q}$ is the first element of $\boldsymbol{q}_i$. However, if every joint has exactly one degree of freedom, as is the case for revolute and prismatic joints, then $\boldsymbol{q}_i$ is simply element i of $\boldsymbol{q}$.

If the objective is to develop a general-purpose software package, then it makes more sense to create a library of joint models, and have jcalc access this library. One particularly useful kind of joint is a user-defined joint. This facility could be provided by making the library extensible. Alternatively, one could simply require that one of the 'parameters' of a user-defined joint is the address of a user-supplied jcalc for that joint, and the built-in jcalc could be programmed to pass control to the user-supplied function.

Chapter 5

Inverse Dynamics

Inverse dynamics is the problem of finding the forces required to produce a given acceleration in a rigid-body system. Inverse dynamics calculations are used in motion control systems, trajectory design and optimization (e.g. for robots and animated figures), in mechanical design, and as a component in some forward dynamics algorithms. This last application is covered in Chapter 6. The inverse dynamics calculation can be summarized by the equation

$$\boldsymbol{\tau} = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}),$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ are vectors of generalized position, velocity, acceleration and force variables, respectively, and $model$ is a system model of a particular rigid-body system.

This chapter presents an algorithm to calculate the inverse dynamics of a kinematic tree. An algorithm for closed-loop systems appears in Chapter 8. The inverse dynamics of a kinematic tree is particularly easy to calculate, which is why we start with this problem. The algorithm is called the *recursive Newton-Euler algorithm*, and it is the simplest, most efficient algorithm for the job. It has a computational complexity of $O(n)$, which means that the amount of calculation grows linearly with the number of bodies, or joint variables, in the tree. The algorithm is presented first in its basic form; then the implementation details are added; and then the spatial-vector version is compared with the original 3D vector version. This chapter also covers the topic of algorithm complexity, and explains why recursive algorithms are so efficient.

5.1   Algorithm Complexity

We define the computational cost of an algorithm to be its *operations count*— the number of arithmetic operations performed in the course of executing the algorithm. If an algorithm involves real-number arithmetic, like a dynamics algorithm, then it is customary to count only the floating-point operations.

Thus, integer operations, like those required to calculate an array index or increment a loop variable, get ignored.

The *computational complexity* of an algorithm is a measure of how the cost scales with the size of the problem. We use 'big O' notation to specify this quantity. If an algorithm is described as having $O(n^p)$ complexity, where n is the size of the problem, then the cost grows in proportion to n^p as $n \to \infty$. To be more precise, an algorithm is said to have $O(f(n))$ complexity if there exist positive constants C_{min}, C_{max} and n_{min} such that the computational cost is at least $C_{min}f(n)$, but no more than $C_{max}f(n)$, for all $n > n_{min}$.

The expression $O(f(n))$ can be interpreted as the set of all functions that grow in proportion to $f(n)$ as $n \to \infty$. Thus, a statement like "Algorithm X is $O(f(n))$" means that the cost of X, expressed as a function of n, is an element of the set $O(f(n))$. An expression like $O(f(n)) = O(g(n))$ is then a set equality, and implies $g(n) \in O(f(n))$ and vice versa. When quoting complexities, we select the simplest function in the set to serve as the argument to O. For example, if the exact cost of an algorithm is given by the polynomial $a_0 + a_1 n + \cdots + a_p n^p$, then the simplest function that grows in proportion to this polynomial is n^p, and so we quote the complexity as $O(n^p)$. In general, only the fastest-growing component of f matters. If f converges to a constant then we say that f is $O(1)$.

An algorithm's complexity figure depends on how we measure the size of the problem. For example, the complexity of matrix inversion is said to be $O(n^3)$ because the cost of inverting an $n \times n$ matrix grows in proportion to n^3 as $n \to \infty$. This figure relies on n being the dimension of the matrix. If n were instead the number of elements in the matrix, then the complexity figure would be $O(n^{1.5})$. More generally, if m and n are two different measures of problem size, and $m = g(n)$, then $O(f(m)) = O(f(g(n)))$.

For a dynamics algorithm, the size of the problem is the size of the rigid-body system. If the system is a kinematic tree, then there are two reasonable measures of system size: the number of bodies, N_B, or the number of degrees of freedom, n. If every joint in the tree has at least one degree of freedom, then these two numbers satisfy $N_B \le n \le 6N_B$, which implies that

$$O(N_B^p) = O(n^p)\,. \tag{5.1}$$

There is therefore no need to distinguish between N_B and n when quoting complexities.

5.2   Recurrence Relations

Modern dynamics algorithms are recursive, and they are orders of magnitude faster than their non-recursive predecessors. They are called recursive because they make use of recurrence relations to calculate sequences of intermediate results. It is the use of recurrence relations that accounts for their efficiency; so let us take a moment to examine how these recurrence relations work.

A *recurrence relation* is a formula that defines the next element in a sequence in terms of previous elements. This formula, together with a set of initial values, provides a recursive definition of the sequence. For example, the sequence of Fibonacci numbers, 0, 1, 1, 2, 3, 5, 8, 13, ..., is defined recursively by the recurrence relation $F_i = F_{i-1} + F_{i-2}$ and the initial values $F_0 = 0$ and $F_1 = 1$. In a rigid-body system, if the bodies are numbered according to the rules in Section 4.1.2, then the sequence of body velocities, v_0, v_1, v_2, ..., is defined recursively by the recurrence relation $v_i = v_{\lambda(i)} + S_i \dot{q}_i$ and a given initial value for v_0.

Let us examine the task of calculating body velocities, both with and without the use of recurrence relations. To keep things simple, let us assume that the system is an unbranched kinematic tree with 1-DoF joints. This means that $\lambda(i) = i - 1$, and that the velocity across joint i is the product of the joint axis vector, s_i, with the scalar joint velocity variable $\dot{q}_i$. The recurrence relation for body velocities is then

$$v_i = v_{i-1} + s_i \dot{q}_i , \qquad (5.2)$$

with the initial condition $v_0 = 0$; and the non-recursive formula for body velocities is

$$v_i = \sum_{j=1}^{i} s_j \dot{q}_j . \qquad (5.3)$$

Let m and a denote the cost of a scalar-vector multiplication and a vector addition, respectively. The cost of one application of Eq. 5.2 is then $m + a$, and the cost of Eq. 5.3 is $i\,m + (i-1)a$. The cost of calculating the first n body velocities via Eq. 5.2 is therefore $n(m + a)$, and the cost via Eq. 5.3 is $\frac{1}{2}(n(n+1)m + n(n-1)a)$. Thus, the use of Eq. 5.2 results in an efficient $O(n)$ calculation, while Eq. 5.3 leads to an inefficient $O(n^2)$ calculation.

The source of the inefficiency is not hard to see. If we perform the task using Eq. 5.3, then we end up calculating

$$v_1 = s_1 \dot{q}_1$$
$$v_2 = s_1 \dot{q}_1 + s_2 \dot{q}_2$$
$$\vdots$$
$$v_n = s_1 \dot{q}_1 + s_2 \dot{q}_2 + \cdots + s_n \dot{q}_n .$$

The product $s_1 \dot{q}_1$ is calculated n times over, when it only needed to be calculated once. Likewise, the product $s_2 \dot{q}_2$ is calculated $n-1$ times, the sum $s_1 \dot{q}_1 + s_2 \dot{q}_2$ is calculated $n-1$ times, and so on. Recurrence relations therefore offer a systematic way to avoid unnecessary repetition in calculating sequences of related quantities, and this is the secret of their efficiency.

The computational advantage of recurrence relations can be even greater when the calculation is more complicated. For example, the recurrence relation for body accelerations in the unbranched chain is

$$a_i = a_{i-1} + s_i \ddot{q}_i + v_i \times s_i \dot{q}_i , \qquad (5.4)$$

which is just the time-derivative of Eq. 5.2, and the non-recursive formula is

$$\boldsymbol{a}_i = \sum_{j=1}^{i} \boldsymbol{s}_j \, \ddot{q}_j + \sum_{j=1}^{i} \sum_{k=1}^{j-1} (\boldsymbol{s}_k \, \dot{q}_k) \times (\boldsymbol{s}_j \, \dot{q}_j) \, . \tag{5.5}$$

(In both cases, we have assumed that $\dot{\boldsymbol{s}}_i = \boldsymbol{v}_i \times \boldsymbol{s}_i$.) Calculating the accelerations via Eq. 5.4 is an $O(n)$ computation; but doing it via Eq. 5.5 is an $O(n^3)$ computation.

Equations 5.3 and 5.5 express the body velocities and accelerations directly in terms of their elementary constituents—joint axis vectors and joint velocity and acceleration variables. Equations like this are said to be in closed form. The equation of motion of a rigid-body system, expressed in closed form, is

$$\tau_i = \sum_{j=1}^{n} H_{ij} \, \ddot{q}_i + \sum_{j=1}^{n} \sum_{k=1}^{n} C_{ijk} \, \dot{q}_j \, \dot{q}_k + g_i \, , \tag{5.6}$$

where H_{ij} are the elements of the generalized inertia matrix, C_{ijk} are the coefficients of the centrifugal and Coriolis forces, and g_i are the gravity terms. Early, non-recursive algorithms calculated the dynamics via equations like this, and their computational complexities were typically $O(n^4)$. Recursive algorithms are far more efficient, and they have computational complexities as low as $O(n)$. When the recursive Newton-Euler algorithm first appeared, it was found to be about a hundred times faster than its non-recursive predecessor, the Uicker/Kahn algorithm, for a 6-DoF robot arm (Hollerbach, 1980). Prior to the invention of efficient recursive algorithms, closed-form equations were the standard way of expressing and evaluating equations of motion.

5.3   The Recursive Newton-Euler Algorithm

The inverse dynamics of a general kinematic tree can be calculated in three steps as follows:

1. Calculate the velocity and acceleration of each body in the tree.

2. Calculate the forces required to produce these accelerations.

3. Calculate the forces transmitted across the joints from the forces acting on the bodies.

These are the steps of the recursive Newton-Euler algorithm. Let us now work out the equations for each step, assuming that the tree is modelled by the quantities $\lambda(i)$, $\mu(i)$, $\boldsymbol{S}_i$ and $\boldsymbol{I}_i$, as explained in Chapter 4.
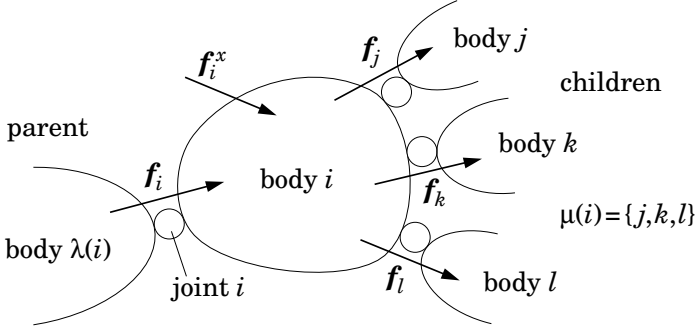
Figure 5.1: Forces acting on body i

Step 1: Let $\boldsymbol{v}_i$ and $\boldsymbol{a}_i$ be the velocity and acceleration of body i. The velocity of any body in a kinematic tree can be defined recursively as the sum of the velocity of its parent and the velocity across the joint connecting it to its parent. Thus,

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i\,. \qquad (\boldsymbol{v}_0 = \boldsymbol{0}) \tag{5.7}$$

(Initial values are shown in brackets.) The recurrence relation for accelerations is obtained by differentiating this equation, resulting in

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i\,. \qquad (\boldsymbol{a}_0 = \boldsymbol{0}) \tag{5.8}$$

Successive iterations of these two formulae, with i ranging from 1 to N_B, will calculate the velocity and acceleration of every body in the tree.

Step 2: Let $\boldsymbol{f}_i^{B}$ be the net force acting on body i. This force is related to the acceleration of body i by the equation of motion

$$\boldsymbol{f}_i^{B} = \boldsymbol{I}_i\,\boldsymbol{a}_i + \boldsymbol{v}_i \times^{*} \boldsymbol{I}_i\,\boldsymbol{v}_i\,. \tag{5.9}$$

Step 3: Referring to Figure 5.1, let $\boldsymbol{f}_i$ be the force transmitted from body $\lambda(i)$ to body i across joint i, and let $\boldsymbol{f}_i^{x}$ be the external force (if any) acting on body i. External forces can model a variety of environmental influences: force fields, physical contacts, and so on. They are regarded as inputs to the algorithm; that is, their values are assumed to be known. The net force on body i is then

$$\boldsymbol{f}_i^{B} = \boldsymbol{f}_i + \boldsymbol{f}_i^{x} - \sum_{j \in \mu(i)} \boldsymbol{f}_j\,,$$

which can be rearranged to give a recurrence relation for the joint forces:

$$\boldsymbol{f}_i = \boldsymbol{f}_i^{B} - \boldsymbol{f}_i^{x} + \sum_{j \in \mu(i)} \boldsymbol{f}_j\,. \tag{5.10}$$

Successive iterations of this formula, with i ranging from N_B down to 1, will calculate the spatial force across every joint in the tree. No initial values are required for this formula, as $\mu(i)$ is the empty set for any body with no children.

Having computed the spatial force across each joint, it remains only to calculate the generalized forces at the joints, which are given by

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{f}_i \qquad\qquad (5.11)$$

(cf. Eq. 3.37). Equations 5.7 to 5.11 describe the recursive Newton Euler algorithm in its simplest form.

External forces can be used to model a variety of environmental influences, including gravitational forces. However, a uniform gravitational field can be modelled more efficiently as a fictitious acceleration of the base. To do this, we simply replace the initial value for Eq. 5.8 with $\boldsymbol{a}_0 = -\boldsymbol{a}_g$, where $\boldsymbol{a}_g$ is the gravitational acceleration vector. If this trick is used, then the calculated values of $\boldsymbol{a}_i$ and $\boldsymbol{f}_i^B$ are no longer the true acceleration and net force for body i, but are offset by the gravitational acceleration and force vectors, respectively. (The gravitational force on body i is $\boldsymbol{I}_i \, \boldsymbol{a}_g$.)

Algorithm Features

Equations 5.7 and 5.8 involve a propagation of data from the root to the leaves; Eq. 5.10 involves a propagation of data from the leaves to the root; and Eqs. 5.9 and 5.11 describe calculations that are local to each body. The recursive Newton-Euler algorithm is therefore a two-pass algorithm: it makes an outward pass through the tree, during which the velocities and accelerations are calculated, and then an inward pass, during which the joint forces are calculated. The calculation of body forces can be assigned to either the first or the second pass. The former is generally the better choice.

This algorithm calculates more that just $\boldsymbol{\tau}_i$. Other quantities, like $\boldsymbol{v}_i$ and $\boldsymbol{f}_i$, can sometimes be useful. For example, one of the tasks of a machine designer is to select joint bearings of sufficient strength to withstand the dynamic reaction forces during operation. These forces can be calculated from $\boldsymbol{f}_i$.

By setting some inputs to zero, one can calculate specific components of the dynamics. For example, if the external force and joint velocity and acceleration terms are set to zero, but the fictitious base acceleration is retained, then the algorithm calculates only the gravity compensation terms—the terms that statically balance the force of gravity. Such terms are used in robot control systems. Alternatively, if we set gravity, acceleration and external force terms all to zero, but keep the joint velocities, then the algorithm calculates only the Coriolis and centrifugal terms. If efficiency is a concern, then a stripped-down version of the algorithm can be used to perform these specialized calculations. For example, if the objective is to calculate the gravity-compensation terms

only, then the algorithm simplifies to

$$f_i = -I_i \, a_g + \sum_{j \in \mu(i)} f_j \tag{5.12}$$

$$\tau_i = S_i^{\mathrm{T}} f_i \,. \tag{5.13}$$

To calculate the actual gravity terms, replace $-I_i \, a_g$ with $I_i \, a_g$.

Equations in Body Coordinates

Equations 5.7 to 5.11 express the algorithm without reference to any particular coordinate system. In practice, the algorithm works best if the calculations pertaining to body i are performed in body i coordinates (as defined in §4.2). This is the body-coordinates (or link-coordinates) version of the algorithm.

Let us transform the equations of the recursive Newton-Euler algorithm into body coordinates. The quantities v_i, a_i, S_i, I_i, f_i and f_i^B are now all considered to be expressed in body i coordinates. Note that the system model already supplies S_i and I_i in these coordinates. Expressed in body coordinates, Eqs. 5.7 and 5.8 become

$$v_i = {}^iX_{\lambda(i)} \, v_{\lambda(i)} + S_i \, \dot{q}_i \qquad (v_0 = 0) \tag{5.14}$$

and

$$a_i = {}^iX_{\lambda(i)} \, a_{\lambda(i)} + S_i \, \ddot{q}_i + \mathring{S}_i \, \dot{q}_i + v_i \times S_i \, \dot{q}_i \,. \qquad (a_0 = -a_g) \tag{5.15}$$

Compared with the original equations, the following changes have been made: a coordinate transform has been inserted to transform $v_{\lambda(i)}$ and $a_{\lambda(i)}$ from body $\lambda(i)$ coordinates to body i coordinates; and the matrix $\dot{S}_i$ has been replaced with $\mathring{S}_i + v_i \times S_i$, where $\mathring{S}_i$ is the apparent derivative of S_i in body coordinates. We have also taken the opportunity to replace the original initial acceleration value with $a_0 = -a_g$ to simulate the effect of gravity.

The joint calculation routine, jcalc, is able to supply the quantities S_i, $v_{\mathrm{J}i} = S_i \, \dot{q}_i$ and $c_{\mathrm{J}i} = \mathring{S}_i \, \dot{q}_i$, all in body i coordinates. We therefore modify Eqs. 5.14 and 5.15 as follows, in order to exploit the available data:

$$v_{\mathrm{J}i} = S_i \, \dot{q}_i \tag{5.16}$$

$$c_{\mathrm{J}i} = \mathring{S}_i \, \dot{q}_i \tag{5.17}$$

$$v_i = {}^iX_{\lambda(i)} \, v_{\lambda(i)} + v_{\mathrm{J}i} \qquad (v_0 = 0) \tag{5.18}$$

$$a_i = {}^iX_{\lambda(i)} \, a_{\lambda(i)} + S_i \, \ddot{q}_i + c_{\mathrm{J}i} + v_i \times v_{\mathrm{J}i} \,. \qquad (a_0 = -a_g) \tag{5.19}$$

Equations 5.9 and 5.11 remain unchanged in the body-coordinates version, but Eq. 5.10 becomes

$$f_i = f_i^B - {}^iX_0^* \, f_i^x + \sum_{j \in \mu(i)} {}^iX_j^* \, f_j \,. \tag{5.20}$$

Basic Equations:

$$\boldsymbol{v}_0 = \boldsymbol{0}$$

$$\boldsymbol{a}_0 = -\boldsymbol{a}_g$$

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i$$

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i$$

$$\boldsymbol{f}_i^B = \boldsymbol{I}_i\,\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i$$

$$\boldsymbol{f}_i = \boldsymbol{f}_i^B - \boldsymbol{f}_i^x + \textstyle\sum_{j\in\mu(i)} \boldsymbol{f}_j$$

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{f}_i$$

Equations in Body Coordinates:

$$\boldsymbol{v}_0 = \boldsymbol{0}$$

$$\boldsymbol{a}_0 = -\boldsymbol{a}_g$$

$$\boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i$$

$$\boldsymbol{c}_{\mathrm{J}i} = \mathring{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i$$

$$\boldsymbol{v}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i}$$

$$\boldsymbol{a}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i + \boldsymbol{c}_{\mathrm{J}i} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}i}$$

$$\boldsymbol{f}_i^B = \boldsymbol{I}_i\,\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i$$

$$\boldsymbol{f}_i = \boldsymbol{f}_i^B - {}^{i}\boldsymbol{X}_0^*\,\boldsymbol{f}_i^x + \textstyle\sum_{j\in\mu(i)} {}^{i}\boldsymbol{X}_j^*\,\boldsymbol{f}_j$$

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{f}_i$$

Algorithm:

$$\boldsymbol{v}_0 = \boldsymbol{0}$$

$$\boldsymbol{a}_0 = -\boldsymbol{a}_g$$

for $i = 1$ **to** N_B **do**

    $[\boldsymbol{X}_{\mathrm{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\mathrm{J}}, \boldsymbol{c}_{\mathrm{J}}] =$

               $\mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$

    ${}^{i}\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}}\,\boldsymbol{X}_{\mathrm{T}}(i)$

    if $\lambda(i) \neq 0$ **then**

        ${}^{i}\boldsymbol{X}_0 = {}^{i}\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{X}_0$

    end

    $\boldsymbol{v}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}}$

    $\boldsymbol{a}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$

               $+ \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}}$

    $\boldsymbol{f}_i = \boldsymbol{I}_i\,\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i - {}^{i}\boldsymbol{X}_0^*\,\boldsymbol{f}_i^x$

end

for $i = N_B$ **to** 1 **do**

    $\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{f}_i$

    if $\lambda(i) \neq 0$ **then**

        $\boldsymbol{f}_{\lambda(i)} = \boldsymbol{f}_{\lambda(i)} + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{f}_i$

    end

end

Table 5.1: The recursive Newton-Euler equations and algorithm

In this equation, we have assumed that the external forces emanate from outside the system, and have therefore assumed that they are expressed in absolute (i.e., body 0) coordinates. Equations 5.16 to 5.20, together with 5.9 and 5.11, are the equations of the body-coordinates version of the recursive Newton-Euler algorithm.

Algorithm Details

Table 5.1 shows the pseudocode for the body-coordinates version of the algorithm, together with the equations for both the basic version and the body-coordinates version. Bear in mind that the symbols $\boldsymbol{v}_i$, $\boldsymbol{a}_i$, etc. in the body-coordinates version are expressed in body coordinates, whereas the same symbols in the basic version represent either abstract vectors or coordinate vectors in an unspecified common coordinate system.

    The two-pass structure of the algorithm is evident from the two **for** loops in the pseudocode. The symbols $\boldsymbol{X}_{\mathrm{J}}$, $\boldsymbol{v}_{\mathrm{J}}$ and $\boldsymbol{c}_{\mathrm{J}}$ are local variables that take new values on each iteration of the first loop. If the tree contains joints for which the velocity variables are not the derivatives of the position variables, then

simply replace the symbols $\dot{\boldsymbol{q}}_i$ and $\ddot{\boldsymbol{q}}_i$ with $\boldsymbol{\alpha}_i$ and $\dot{\boldsymbol{\alpha}}_i$, respectively. Likewise, if the tree contains any joints that depend explicitly on time, then jcalc will automatically calculate $\boldsymbol{v}_{\mathrm{J}}$ and $\boldsymbol{c}_{\mathrm{J}}$ using Eqs. 3.32 and 3.41 instead of 5.16 and 5.17; and the only change in the pseudocode is to pass the current time as a fourth argument to jcalc. The only place where ${}^i\boldsymbol{X}_0$ is used is to transform the external forces from base to body coordinates. If there are no external forces, then the **if** statement that calculates these transforms can be omitted.

In implementing Eq. 5.20, we have transformed it from a calculation using $\mu(i)$ to a calculation using $\lambda(i)$. A literal translation of Eq. 5.20 into pseudocode looks like this:

> **for** $i = N_B$ **to** 1 **do**
> $\boldsymbol{f}_i = \boldsymbol{f}_i^B - {}^i\boldsymbol{X}_0^* \, \boldsymbol{f}_i^x$
>   **for each** j **in** $\mu(i)$ **do**
> $\boldsymbol{f}_i = \boldsymbol{f}_i + {}^i\boldsymbol{X}_j^* \, \boldsymbol{f}_j$
>   **end**
> **end**

However, the same calculation can be performed in a different order, but with the same end result, by the code fragment

> **for** $i = 1$ **to** N_B **do**
> $\boldsymbol{f}_i = \boldsymbol{f}_i^B - {}^i\boldsymbol{X}_0^* \, \boldsymbol{f}_i^x$
> **end**
> **for** $i = N_B$ **to** 1 **do**
>   **if** $\lambda(i) \neq 0$ **then**
> $\boldsymbol{f}_{\lambda(i)} = \boldsymbol{f}_{\lambda(i)} + {}^{\lambda(i)}\boldsymbol{X}_i^* \, \boldsymbol{f}_i$
>   **end**
> **end**

In merging this code fragment with the rest of the algorithm, the body of the first loop becomes the last line in the body of the first loop in the algorithm.

5.4 The Original Version

The recursive Newton-Euler algorithm was first described in a paper by Luh et al. (1980a), which was republished in Brady et al. (1982). This original algorithm was formulated using 3D vectors, and appears superficially to be quite different from the one described in Section 5.3. However, the two algorithms are actually almost the same, the only significant difference being that one uses spatial accelerations while the other uses classical accelerations (see §2.11).

Table 5.2 shows a special case of the original algorithm in which the joints are all revolute. One of the drawbacks of using 3D vectors to express rigid-body dynamics is that the equations are sensitive to joint type. The algorithm

Original Equations:

$$\begin{aligned}
\boldsymbol{\omega}_i &= {}^i\boldsymbol{E}_{i-1}\,\boldsymbol{\omega}_{i-1} + \boldsymbol{z}\,\dot{q}_i \\
\boldsymbol{v}_i &= {}^i\boldsymbol{E}_{i-1}\,(\boldsymbol{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{r}_i) \\
\dot{\boldsymbol{\omega}}_i &= {}^i\boldsymbol{E}_{i-1}\,\dot{\boldsymbol{\omega}}_{i-1} + \boldsymbol{z}\,\ddot{q}_i + ({}^i\boldsymbol{E}_{i-1}\,\boldsymbol{\omega}_{i-1}) \times \boldsymbol{z}\,\dot{q}_i \\
\dot{\boldsymbol{v}}_i &= {}^i\boldsymbol{E}_{i-1}\,(\dot{\boldsymbol{v}}_{i-1} + \dot{\boldsymbol{\omega}}_{i-1} \times {}^{i-1}\boldsymbol{r}_i + \boldsymbol{\omega}_{i-1} \times \boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{r}_i) \\
\boldsymbol{F}_i &= m_i\,(\dot{\boldsymbol{v}}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{c}_i + \boldsymbol{\omega}_i \times \boldsymbol{\omega}_i \times \boldsymbol{c}_i) \\
\boldsymbol{N}_i &= \boldsymbol{I}_i^{cm}\,\dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times \boldsymbol{I}_i^{cm}\,\boldsymbol{\omega}_i \\
\boldsymbol{f}_i &= \boldsymbol{F}_i + {}^i\boldsymbol{E}_{i+1}\,\boldsymbol{f}_{i+1} \\
\boldsymbol{n}_i &= \boldsymbol{N}_i + \boldsymbol{c}_i \times \boldsymbol{F}_i + {}^i\boldsymbol{E}_{i+1}\boldsymbol{n}_{i+1} + {}^i\boldsymbol{r}_{i+1} \times {}^i\boldsymbol{E}_{i+1}\,\boldsymbol{f}_{i+1} \\
\tau_i &= \boldsymbol{z}^{\mathrm{T}}\boldsymbol{n}_i
\end{aligned}$$

Symbols:

$\boldsymbol{\omega}_i,\ \dot{\boldsymbol{\omega}}_i$	angular velocity and acceleration of body i
$\boldsymbol{v}_i,\ \dot{\boldsymbol{v}}_i$	linear velocity and acceleration of the origin of F_i (body i coordinate frame)
$\boldsymbol{F}_i,\ \boldsymbol{N}_i$	net force and moment on body i, expressed at its centre of mass
$\boldsymbol{f}_i,\ \boldsymbol{n}_i$	force and moment exerted on body i through joint i
$m_i,\ \boldsymbol{c}_i,\ \boldsymbol{I}_i^{cm}$	mass, centre of mass, and rotational inertia about centre of mass for body i in F_i coordinates
${}^i\boldsymbol{E}_{i-1}$	3×3 orthogonal rotation matrix transforming from F_{i-1} to F_i coordinates
${}^{i-1}\boldsymbol{r}_i$	position of origin of F_i relative to F_{i-1}, expressed in F_{i-1} coordinates
$\dot{q}_i,\ \ddot{q}_i,\ \tau_i$	joint i velocity, acceleration and force variables
$\boldsymbol{z}$	direction of joint i axis in F_i coordinates

Correspondence with version in Table 5.1:

$$\hat{\boldsymbol{v}}_i = \begin{bmatrix} \boldsymbol{\omega}_i \\ \boldsymbol{v}_i \end{bmatrix} \qquad \hat{\boldsymbol{a}}_i = \begin{bmatrix} \dot{\boldsymbol{\omega}}_i \\ \dot{\boldsymbol{v}}_i - \boldsymbol{\omega}_i \times \boldsymbol{v}_i \end{bmatrix} \qquad \hat{\boldsymbol{f}}_i^B = \begin{bmatrix} \boldsymbol{N}_i + \boldsymbol{c}_i \times \boldsymbol{F}_i \\ \boldsymbol{F}_i \end{bmatrix} \qquad \hat{\boldsymbol{f}}_i = \begin{bmatrix} \boldsymbol{n}_i \\ \boldsymbol{f}_i \end{bmatrix}$$

$$^i\boldsymbol{X}_{i-1} = \begin{bmatrix} {}^i\boldsymbol{E}_{i-1} & \boldsymbol{0} \\ -{}^i\boldsymbol{E}_{i-1}\,{}^{i-1}\boldsymbol{r}_i \times & {}^i\boldsymbol{E}_{i-1} \end{bmatrix} \qquad \hat{\boldsymbol{I}}_i = \begin{bmatrix} \boldsymbol{I}_i^{cm} - m_i\,\boldsymbol{c}_i \times \boldsymbol{c}_i \times & m_i\,\boldsymbol{c}_i \times \\ -m_i\,\boldsymbol{c}_i \times & m_i\,\boldsymbol{1} \end{bmatrix}$$

$$\boldsymbol{S}_i = \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{0} \end{bmatrix} \qquad \boldsymbol{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \lambda(i) = i - 1 \qquad \mu(i) = \{i + 1\}$$

Table 5.2: The original recursive Newton-Euler equations

described in Luh et al. (1980a) was formulated for both revolute and prismatic joints, and it is full of equations of the form

$$variable = \begin{cases} one\ expression & \text{if joint is revolute} \\ another\ expression & \text{if joint is prismatic.} \end{cases}$$

The choice of 3D vector symbols in this table is a compromise between the notation used in this book and that used in other books that list this algorithm. The equations in this table are written for an unbranched kinematic tree, in line with how this algorithm is described elsewhere, but it is a simple matter to modify them to work on a branched tree.

Table 5.2 also shows the correspondence between the 3D vectors in the original algorithm and the spatial vectors in the algorithm shown in Table 5.1. There are only two places where the 3D vectors do not match the corresponding 3D component of a spatial vector: linear acceleration and net body moment. The latter is a trivial difference that is immediately rectified when N_i is used to calculate n_i. It is caused by N_i being calculated at the centre of mass instead of at the body coordinate system's origin. The former is a non-trivial difference, as it affects the quantities propagated from one body to the next.[1]

One interesting feature of the original algorithm is that the linear velocity is never used in any subsequent expression, and therefore need not be calculated. This is a special property that arises from using classical accelerations. If the velocity calculation is omitted, then this algorithm is slightly more efficient than the spatial version. However, the difference is only a few percent, and is easily dwarfed by other efficiency-related matters, such as the choice of programming language, the computer hardware, and so on.

5.5   Additional Notes

The algorithm presented in Luh et al. (1980a) is the one that we usually regard as *the* recursive Newton-Euler algorithm. However, the idea of combining the Newton-Euler equations of motion with a recursive evaluation strategy predates this algorithm by a few years. In particular, one can see early forms of recursive Newton-Euler algorithms in Stepanenko and Vukobratovic (1976) and Orin et al. (1979). The algorithm has since been refined and optimized, and a few of these optimizations are covered in Chapter 10. Two of the fastest published versions can be found in He and Goldenberg (1989) and Balafoutis et al. (1988), the latter also appearing in Balafoutis and Patel (1991). At the time of

---

[1]According to Section 2.11, classical acceleration is the apparent derivative of spatial velocity in a coordinate system having a pure linear velocity equal to the velocity of the body-fixed point at the origin. In light of this, one can interpret the difference between the two algorithms as follows: the algorithm in Table 5.1 expresses the equations in stationary coordinate systems that coincide with the body coordinate frames at the current instant; whereas the algorithm in Table 5.2 expresses the equations in non-rotating coordinate systems that coincide with the body coordinate frames at the current instant, and that have the same linear velocities as the origins of the body coordinate frames.

its invention, Lagrangian dynamics was more popular than Newton-Euler; and this prompted Hollerbach to devise a recursive Lagrangian algorithm, and to compare the computational complexities of various recursive and nonrecursive algorithms (Hollerbach, 1980). This is the work that contrasted the $O(n^4)$ complexity of the nonrecursive approach, as typified by the Uicker/Kahn algorithm (Kahn and Roth, 1971), with the $O(n)$ complexity of the recursive algorithms.

The impetus for this kind of research within the robotics community at that time was to develop an inverse dynamics algorithm that would be fast enough for use in real-time control of robot motion. An early example of this can be found in Luh et al. (1980b), although the operational-space formulation eventually became more popular (Khatib, 1987). The relatively slow speed of computers at the time led some researchers to investigate parallel-computer implementations (Lathrop, 1985), while others investigated symbolic optimization techniques (Murray and Neuman, 1984; Neuman and Murray, 1987). One interesting new direction is the development of efficient algorithms to calculate the partial derivative of inverse dynamics with respect to a design parameter. An example of this can be found in Fang and Pollard (2003).

Chapter 6

Forward Dynamics — Inertia Matrix Methods

Forward dynamics is the problem of finding the acceleration of a rigid-body system in response to given applied forces. It is used mainly for simulation; and it is sometimes called 'direct dynamics', or simply 'dynamics'. In this chapter and the next, we examine the forward dynamics of kinematic trees. The dynamics of closed-loop systems is covered in Chapter 8.

There are two main approaches to solving the forward dynamics problem for a kinematic tree:

1. form an equation of motion for the whole system, and solve it for the acceleration variables; or

2. propagate constraints from one body to the next in such a way that the accelerations can be calculated one joint at a time.

We examine the first approach in this chapter, and the second in Chapter 7. The first approach involves calculating the elements of an $n \times n$ joint-space inertia matrix, and then factorizing this matrix in order to solve a set of n linear equations for the acceleration variables. Such algorithms have $O(n^3)$ complexity in the worst case, and are sometimes referred to collectively as $O(n^3)$ algorithms. However, it would be more accurate to describe them as $O(nd^2)$ algorithms, where d is the depth of the tree. The second approach involves making a fixed number of passes through the tree, each pass performing a fixed number of calculations per body. Algorithms that take this approach have $O(n)$ complexity, and are therefore the best choice for systems containing a large number of bodies. However, the $O(nd^2)$ algorithms can match or slightly exceed the speed of the $O(n)$ algorithms on trees with only a few bodies, or that are branched enough to have a small depth. The $O(nd^2)$ algorithms are also an important component in closed-loop dynamics algorithms. The efficiency trade-off between these two approaches is discussed in Chapter 10.

This chapter commences with some basic material on how to solve the forward dynamics problem using the joint-space inertia matrix. Then Section 6.2 presents the composite-rigid-body algorithm, which is the fastest algorithm for calculating this matrix; and Section 6.3 presents a second derivation that reflects how the algorithm was originally discovered and described. The next two sections describe the phenomenon of branch-induced sparsity, and present two factorization algorithms to exploit it. This sparsity is the reason why the cost of calculating the joint-space inertia matrix is only $O(nd)$, and the cost of factorizing it only $O(nd^2)$. Finally, Section 6.6 presents some background material on the forward dynamics problem.

6.1 The Joint-Space Inertia Matrix

The equation of motion for a kinematic tree can be expressed in matrix form as

$$\boldsymbol{\tau} = \boldsymbol{H}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{f}^x)\,, \tag{6.1}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ are vectors of joint position, velocity, acceleration and force variables, $\boldsymbol{f}^x$ is a vector of external forces (if any), $\boldsymbol{H}$ is the *joint-space inertia matrix*, and $\boldsymbol{C}$ is the joint-space bias force. $\boldsymbol{H}$ and $\boldsymbol{C}$ are the coefficients of the equation of motion, while $\boldsymbol{\tau}$ and $\ddot{\boldsymbol{q}}$ are the variables. This equation shows $\boldsymbol{H}$ as a function of $\boldsymbol{q}$, and $\boldsymbol{C}$ as a function of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\boldsymbol{f}^x$. Once these functional dependencies are understood, they are usually omitted. The most useful piece of information they convey is the fact that $\boldsymbol{H}$ depends only on the position variables. If the system is at rest, and there are no forces acting on it other than $\boldsymbol{\tau}$, then $\boldsymbol{C} = \boldsymbol{0}$ and the equation of motion simplifies to $\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}}$.

In the forward dynamics problem, $\boldsymbol{\tau}$ is given, and the task is to calculate $\ddot{\boldsymbol{q}}$. The methods described in this chapter solve the problem by the following procedure:

1. calculate $\boldsymbol{C}$,
2. calculate $\boldsymbol{H}$,
3. solve $\boldsymbol{H}\ddot{\boldsymbol{q}} = \boldsymbol{\tau} - \boldsymbol{C}$ for $\ddot{\boldsymbol{q}}$.

These three steps have computational complexities of $O(n)$, $O(nd)$ and $O(nd^2)$, respectively, where d is the depth of the tree. Given that d is bounded by $d \leq n$, the worst-case complexities for steps 2 and 3 are $O(n^2)$ and $O(n^3)$. At the other extreme, if d is subject to a fixed upper limit (which can happen in practice) then the complexity of all three steps is $O(n)$. As explained below, step 1 can be accomplished easily by an inverse dynamics algorithm, so this chapter is concerned mostly with steps 2 and 3.

$\boldsymbol{H}$ is a symmetric, positive-definite, $n \times n$ matrix that is organised into an $N_B \times N_B$ array of submatrices. (Recall that $N_B = N_J$ for a tree.) If n_i is the degree of freedom of joint i, then $\boldsymbol{H}_{ij}$ is the $n_i \times n_j$ submatrix of $\boldsymbol{H}$ occupying block-row i and block-column j. If $p_i = \sum_{k=1}^{i} n_k$ then block-row i consists of

rows $p_{i-1} + 1$ to p_i inclusive, and block-column j consists of columns $p_{j-1} + 1$ to p_j inclusive. Physically, $\boldsymbol{H}_{ij}$ relates the acceleration at joint j to the force at joint i. If every joint in the tree has only one degree of freedom, then $n = N_B$, and $\boldsymbol{H}_{ij}$ is the 1×1 matrix on row i and column j of $\boldsymbol{H}$.

Suppose we have an inverse dynamics calculation function, ID, like the one described in Chapter 5. This function performs the calculation

$$\boldsymbol{\tau} = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{f}^x) \,.$$

On comparing this expression with Eq. 6.1, it follows immediately that

$$\boldsymbol{C} = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{0}, \boldsymbol{f}^x) \tag{6.2}$$

and

$$\boldsymbol{H}\ddot{\boldsymbol{q}} = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{f}^x) - \boldsymbol{C} \,. \tag{6.3}$$

Thus, the value of $\boldsymbol{C}$ can be obtained with a single call to ID, and the product of $\boldsymbol{H}$ with any desired vector is the difference between two calls to ID. Let us define a *differential inverse dynamics* function, ID_δ, which calculates the change in joint force corresponding to a change in desired acceleration. This function is given by

$$\mathrm{ID}_\delta(model, \boldsymbol{q}, \ddot{\boldsymbol{q}}) = \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{f}^x) - \mathrm{ID}(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{0}, \boldsymbol{f}^x) \,. \tag{6.4}$$

With this function, we can calculate column α of $\boldsymbol{H}$ as follows:

$$\boldsymbol{H}\boldsymbol{\delta}_\alpha = \mathrm{ID}_\delta(model, \dot{\boldsymbol{q}}, \boldsymbol{\delta}_\alpha) \,,$$

where $\boldsymbol{\delta}_\alpha$ is an n-dimensional coordinate vector having a 1 in its αth element and zeros elsewhere. By repeating this calculation with α taking every value from 1 to n, we can calculate the whole of $\boldsymbol{H}$ one column at a time. This idea is the basis of methods 1 and 2 described in Walker and Orin (1982).

There is a great deal of cancellation of terms on the right-hand side of Eq. 6.4. In particular, every term depending on either $\dot{\boldsymbol{q}}$ or $\boldsymbol{f}^x$ cancels out, which is why these vectors have been omitted from the argument list of ID_δ. It is therefore possible to formulate a greatly simplified version of the recursive Newton-Euler algorithm specifically to calculate ID_δ. The equations and pseudocode for this algorithm are shown in Table 6.1. The pseudocode omits the calculation of $\boldsymbol{S}_i$ and ${}^i\boldsymbol{X}_{\lambda(i)}$ because they are assumed to be available as a by-product of calculating $\boldsymbol{C}$.

The algorithm in Table 6.1 has the advantage of being simple and straightforward. However, it is not the most efficient way to calculate $\boldsymbol{H}$. The fastest algorithm for that job is the composite-rigid-body algorithm, which is the subject of the next section.

Equations:	Algorithm:
$\boldsymbol{a}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i \qquad (\boldsymbol{a}_0 = \boldsymbol{0})$	$\boldsymbol{a}_0 = \boldsymbol{0}$
	for $i = 1$ **to** N_B **do**
$\boldsymbol{f}_i = \boldsymbol{I}_i\,\boldsymbol{a}_i + \displaystyle\sum_{j\in\mu(i)} {}^i\boldsymbol{X}_j^*\,\boldsymbol{f}_j$	$\qquad \boldsymbol{a}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$
	$\qquad \boldsymbol{f}_i = \boldsymbol{I}_i\,\boldsymbol{a}_i$
$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{f}_i$	**end**
	for $i = N_B$ **to** 1 **do**
	$\qquad \boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{f}_i$
	$\qquad$ **if** $\lambda(i) \neq 0$ **then**
	$\qquad\qquad \boldsymbol{f}_{\lambda(i)} = \boldsymbol{f}_{\lambda(i)} + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{f}_i$
	$\qquad$ **end**
	end

Table 6.1: Equations and algorithm to calculate $\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}}$

6.2   The Composite-Rigid-Body Algorithm

The kinetic energy of a kinematic tree is the sum of the kinetic energies of its bodies. Thus,

$$T = \frac{1}{2}\sum_{k=1}^{N_B} \boldsymbol{v}_k^{\mathrm{T}}\,\boldsymbol{I}_k\,\boldsymbol{v}_k\,. \tag{6.5}$$

The velocity of body k is given by

$$\boldsymbol{v}_k = \sum_{i\in\kappa(k)} \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i\,, \tag{6.6}$$

where $\kappa(k)$ is the set of joints that support body k; that is, the set of joints on the path between body k and the base. (See §4.1.4 for descriptions of $\kappa(i)$, $\nu(i)$ and $\mu(i)$.) Substituting Eq. 6.6 into 6.5 gives

$$T = \frac{1}{2}\sum_{k=1}^{N_B}\sum_{i\in\kappa(k)}\sum_{j\in\kappa(k)} \dot{\boldsymbol{q}}_i^{\mathrm{T}}\,\boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{I}_k\,\boldsymbol{S}_j\,\dot{\boldsymbol{q}}_j\,. \tag{6.7}$$

Now, this equation is a sum over all i, j, k triples in which both joint i and joint j support body k. It can therefore be rearranged into

$$T = \frac{1}{2}\sum_{i=1}^{N_B}\sum_{j=1}^{N_B}\sum_{k\in\nu(i)\cap\nu(j)} \dot{\boldsymbol{q}}_i^{\mathrm{T}}\,\boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{I}_k\,\boldsymbol{S}_j\,\dot{\boldsymbol{q}}_j\,, \tag{6.8}$$

in which $\nu(i) \cap \nu(j)$ is the intersection of the set of bodies supported by joint i and the set of bodies supported by joint j. Equation 6.8 is therefore a sum

over all i, j, k triples in which body k is supported both by joint i and by joint j, which is the same sum as in Eq. 6.7. The value of $\nu(i) \cap \nu(j)$ is given by

$$\nu(i) \cap \nu(j) = \begin{cases} \nu(i) & \text{if } i \in \nu(j) \\ \nu(j) & \text{if } j \in \nu(i) \\ \emptyset & \text{otherwise} \end{cases} \qquad (6.9)$$

where $\emptyset$ denotes the empty set.

Now, one of the properties of the joint-space inertia matrix is that the kinetic energy of the tree is given by the expression

$$T = \frac{1}{2} \dot{\boldsymbol{q}}^{\mathrm{T}} \boldsymbol{H} \dot{\boldsymbol{q}} = \frac{1}{2} \sum_{i=1}^{N_B} \sum_{j=1}^{N_B} \dot{\boldsymbol{q}}_i^{\mathrm{T}} \boldsymbol{H}_{ij} \, \dot{\boldsymbol{q}}_j \, . \qquad (6.10)$$

On comparing this equation with Eq. 6.8, it can be seen that

$$\boldsymbol{H}_{ij} = \sum_{k \in \nu(i) \cap \nu(j)} \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_k \, \boldsymbol{S}_j \, . \qquad (6.11)$$

Let us now introduce a new quantity, $\boldsymbol{I}_i^c$, which is the inertia of the subtree rooted at body i, treated as a single composite rigid body. (This is where the algorithm gets its name.) $\boldsymbol{I}_i^c$ is given by

$$\boldsymbol{I}_i^c = \sum_{j \in \nu(i)} \boldsymbol{I}_j \, ; \qquad (6.12)$$

and it can be computed recursively using the formula

$$\boldsymbol{I}_i^c = \boldsymbol{I}_i + \sum_{j \in \mu(i)} \boldsymbol{I}_j^c \, . \qquad (6.13)$$

Combining Eqs. 6.11, 6.12 and 6.9 produces the following formula for the joint-space inertia matrix:

$$\boldsymbol{H}_{ij} = \begin{cases} \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^c \boldsymbol{S}_j & \text{if } i \in \nu(j) \\ \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_j^c \boldsymbol{S}_j & \text{if } j \in \nu(i) \\ \boldsymbol{0} & \text{otherwise.} \end{cases} \qquad (6.14)$$

Equations 6.13 and 6.14 are the basic equations of the composite-rigid-body algorithm for a general kinematic tree. For the special case of an unbranched tree, these equations simplify to

$$\boldsymbol{I}_i^c = \boldsymbol{I}_i + \boldsymbol{I}_{i+1}^c \qquad (6.15)$$

and

$$\boldsymbol{H}_{ij} = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_{\max(i,j)}^c \, \boldsymbol{S}_j \, . \qquad (6.16)$$

Equations in Body Coordinates

Equation 6.13 can be expressed in body coordinates as

$$\boldsymbol{I}_i^{\mathrm{c}} = \boldsymbol{I}_i + \sum_{j \in \mu(i)} {}^i\boldsymbol{X}_j^* \, \boldsymbol{I}_j^{\mathrm{c}} \, {}^j\boldsymbol{X}_i \,. \tag{6.17}$$

In principle, Eq. 6.14 can be expressed in body coordinates as

$$\boldsymbol{H}_{ij} = \begin{cases} \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^{\mathrm{c}} \, {}^i\boldsymbol{X}_j \, \boldsymbol{S}_j & \text{if } i \in \nu(j) \\ \boldsymbol{S}_i^{\mathrm{T}} \, {}^i\boldsymbol{X}_j^* \boldsymbol{I}_j^{\mathrm{c}} \, \boldsymbol{S}_j & \text{if } j \in \nu(i) \\ \boldsymbol{0} & \text{otherwise.} \end{cases} \tag{6.18}$$

However, to devise an efficient calculation scheme for this equation, it is necessary to define a new quantity, ${}^j\boldsymbol{F}_i$, which is the value of $\boldsymbol{I}_i^{\mathrm{c}}\boldsymbol{S}_i$ expressed in body j coordinates; that is, ${}^j\boldsymbol{F}_i = {}^j\boldsymbol{X}_i^* \boldsymbol{I}_i^{\mathrm{c}}\boldsymbol{S}_i$. To calculate $\boldsymbol{H}$, we need one instance of ${}^j\boldsymbol{F}_i$ for every i, j pair satisfying $j \in \kappa(i)$. They can be calculated recursively using the formula

$$^{\lambda(j)}\boldsymbol{F}_i = {}^{\lambda(j)}\boldsymbol{X}_j^* \, {}^j\boldsymbol{F}_i \,, \qquad ({}^i\boldsymbol{F}_i = \boldsymbol{I}_i^{\mathrm{c}} \, \boldsymbol{S}_i) \tag{6.19}$$

with i taking every value in the range 1 to N_B, while j progresses (separately for each i) through the values i, $\lambda(i)$, $\lambda(\lambda(i))$, and so on, back to the base. The joint-space inertia matrix is then given by

$$\boldsymbol{H}_{ij} = \begin{cases} {}^j\boldsymbol{F}_i^{\mathrm{T}} \boldsymbol{S}_j & \text{if } i \in \nu(j) \\ \boldsymbol{H}_{ji}^{\mathrm{T}} & \text{if } j \in \nu(i) \\ \boldsymbol{0} & \text{otherwise.} \end{cases} \tag{6.20}$$

Equations 6.17, 6.19 and 6.20 are the equations of the composite rigid body algorithm in body coordinates.

Algorithm Details

Table 6.2 shows the pseudocode for the body-coordinates version of the algorithm, together with both the basic equations and the equations for the body-coordinates algorithm. Bear in mind that the symbols $\boldsymbol{I}_i$, $\boldsymbol{I}_i^{\mathrm{c}}$ and $\boldsymbol{S}_i$ in the basic version represent either abstract tensors or tensors expressed in an unspecified common coordinate system, whereas the same symbols in the body-coordinates equations represent quantities that are expressed in the coordinate system of body i. The pseudocode does not include code to calculate $\boldsymbol{S}_i$ or ${}^i\boldsymbol{X}_{\lambda(i)}$ because these quantities are assumed to be available as a by-product of applying the recursive Newton-Euler algorithm to calculate $\boldsymbol{C}$ in Eq. 6.1.

In implementing Eq. 6.17, we have transformed it from a calculation using $\mu(i)$ to a calculation using $\lambda(i)$. A literal translation of Eq. 6.17 into pseudocode looks like this:

Basic Equations:

$$I_i^c = I_i + \sum_{j \in \mu(i)} I_j^c$$

$$H_{ij} = \begin{cases} S_i^T I_i^c S_j & \text{if } i \in \nu(j) \\ S_i^T I_j^c S_j & \text{if } j \in \nu(i) \\ 0 & \text{otherwise} \end{cases}$$

Equations for Body-Coordinates Algorithm:

$$I_i^c = I_i + \sum_{j \in \mu(i)} {}^iX_j^* I_j^c \, {}^jX_i$$

$${}^{\lambda(j)}F_i = {}^{\lambda(j)}X_j^* \, {}^jF_i \qquad ({}^iF_i = I_i^c S_i)$$

$$H_{ij} = \begin{cases} {}^jF_i^T S_j & \text{if } i \in \nu(j) \\ H_{ji}^T & \text{if } j \in \nu(i) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

$$H = 0$$
for $i = 1$ **to** N_B **do**
$\qquad I_i^c = I_i$
end
for $i = N_B$ **to** 1 **do**
$\qquad$ **if** $\lambda(i) \neq 0$ **then**
$\qquad\qquad I_{\lambda(i)}^c = I_{\lambda(i)}^c + {}^{\lambda(i)}X_i^* I_i^c \, {}^iX_{\lambda(i)}$
$\qquad$ **end**
$\qquad F = I_i^c S_i$
$\qquad H_{ii} = S_i^T F$
$\qquad j = i$
$\qquad$ **while** $\lambda(j) \neq 0$ **do**
$\qquad\qquad F = {}^{\lambda(j)}X_j^* F$
$\qquad\qquad j = \lambda(j)$
$\qquad\qquad H_{ij} = F^T S_j$
$\qquad\qquad H_{ji} = H_{ij}^T$
$\qquad$ **end**
end

Table 6.2: The composite rigid body equations and algorithm

```
for i = N_B to 1 do
    I_i^c = I_i
    for each j in μ(i) do
        I_i^c = I_i^c + ^iX_j^* I_j^c ^jX_i
    end
end
```

However, the same calculation can be performed in a different order, but with the same end result, using the following code fragment (cf. the implementation of Eq. 5.20 in §5.3):

```
for i = 1 to N_B do
    I_i^c = I_i
end
for i = N_B to 1 do
    if λ(i) ≠ 0 then
        I_λ(i)^c = I_λ(i)^c + ^λ(i)X_i^* I_i^c ^iX_λ(i)
    end
end
```

The algorithm starts with two preparatory steps: setting H to zero and setting each I_i^c to I_i. The first step is included to remind the reader that
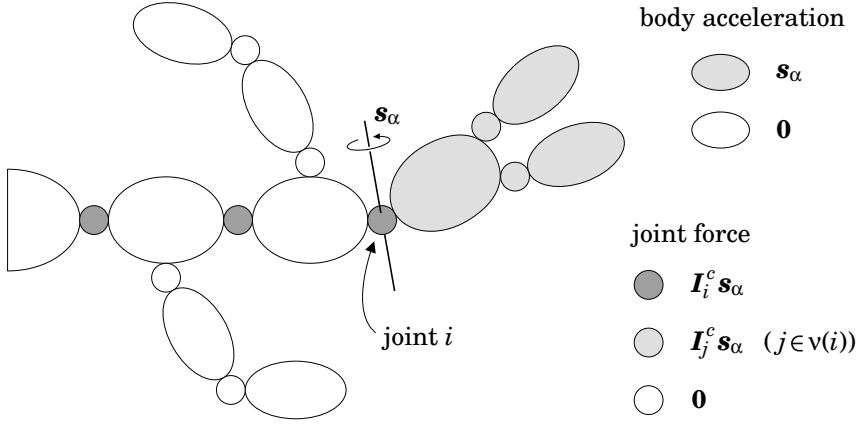
Figure 6.1: Kinematic tree undergoing an acceleration of $\ddot{\boldsymbol{q}} = \boldsymbol{\delta}_\alpha$

the rest of the code will initialize only the nonzero submatrices in $\boldsymbol{H}$. This step may or may not be necessary, depending on how $\boldsymbol{H}$ is to be used. The main loop then implements the calculations for Eqs. 6.17, 6.19 and 6.20. $\boldsymbol{F}$ is a local variable that is initially set to $\boldsymbol{I}_i^c \boldsymbol{S}_i$ (and therefore equal to ${}^i\boldsymbol{F}_i$ in the equations). The **while** loop transforms this variable successively through the coordinate systems of every body j on the path from body i to the base (i.e., every $j \in \kappa(i)$), as it calculates the submatrices $\boldsymbol{H}_{ij}$ and $\boldsymbol{H}_{ji}$. Note that setting $\boldsymbol{H}_{ji}$ will only be necessary if the upper triangle is to be accessed.

If d is the depth of the kinematic tree, then the **while** loop will never execute more than $d-1$ times for any value of i. It therefore follows that the computational cost of the composite-rigid-body algorithm cannot be more than $O(nd)$, and that the number of nonzero elements in $\boldsymbol{H}$ is at most $O(nd)$. A detailed cost analysis appears in Section 10.3.2.

6.3   A Physical Interpretation

This section presents an alternative derivation of the composite-rigid-body algorithm that is closer to the original version, the purpose being to show some of the physical insights that played a role in the discovery of this algorithm.

As mentioned earlier, if the kinematic tree is at rest, and there are no forces acting on it other than $\boldsymbol{\tau}$, then the equation of motion simplifies to $\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}}$. If we set $\ddot{\boldsymbol{q}} = \boldsymbol{\delta}_\alpha$ under these conditions, then $\boldsymbol{\tau}$ will be the αth column of $\boldsymbol{H}$. Figure 6.1 shows what is happening in the tree when $\ddot{\boldsymbol{q}} = \boldsymbol{\delta}_\alpha$. The αth variable in $\ddot{\boldsymbol{q}}$ must belong to a joint. We therefore identify the joint that owns this variable, and call it joint i. Likewise, we identify the column in $\boldsymbol{S}_i$ that corresponds to the αth variable, and call it $\boldsymbol{s}_\alpha$. ($\boldsymbol{s}_\alpha$ is a spatial motion vector.) The motion of the tree can now be stated as follows: the acceleration across joint

i is s_α, and the acceleration across every other joint is zero. As a consequence, the acceleration of every body in the subtree supported by joint i is s_α, and every other body has an acceleration of zero. In other words,

$$a_j = \begin{cases} s_\alpha & \text{if } j \in \nu(i) \\ 0 & \text{otherwise.} \end{cases} \tag{6.21}$$

Let f_j^B and f_j be the net force on body j and the total force transmitted across joint j, respectively. As there are no velocity terms, f_j^B is given by

$$f_j^B = I_j\, a_j = \begin{cases} I_j\, s_\alpha & \text{if } j \in \nu(i) \\ 0 & \text{otherwise.} \end{cases} \tag{6.22}$$

Now, joint j is the only connection between the subtree $\nu(j)$ and the rest of the system. As there are no forces acting on the bodies other than those transmitted across the joints, it follows that f_j must be the net force acting on subtree $\nu(j)$. However, the net force on a subtree is also the sum of the net forces on the bodies in the subtree, so we have

$$f_j = \sum_{k \in \nu(j)} f_k^B. \tag{6.23}$$

Combining these two equations gives

$$f_j = \begin{cases} \sum_{k \in \nu(j)} I_k\, s_\alpha & \text{if } j \in \nu(i) \\ \sum_{k \in \nu(i)} I_k\, s_\alpha & \text{if } j \in \kappa(i) \\ 0 & \text{otherwise,} \end{cases}$$

which simplifies to

$$f_j = \begin{cases} I_j^c s_\alpha & \text{if } j \in \nu(i) \\ I_i^c s_\alpha & \text{if } j \in \kappa(i) \\ 0 & \text{otherwise} \end{cases} \tag{6.24}$$

in view of Eq. 6.12. Finally, the joint force variables are given by

$$\tau_j = S_j^T f_j = \begin{cases} S_j^T I_j^c s_\alpha & \text{if } j \in \nu(i) \\ S_j^T I_i^c s_\alpha & \text{if } j \in \kappa(i) \\ 0 & \text{otherwise.} \end{cases} \tag{6.25}$$

These are the subvectors of column α of H.

The second case in Eq. 6.24 reveals the two key facts that led to the discovery of the composite-rigid-body algorithm: that f_i is the product of a joint motion vector with a composite-rigid-body inertia, and that $f_j = f_i$ for all $j \in \kappa(i)$ (the dark-grey joints in Figure 6.1). The algorithm therefore consists of:

1. calculating the composite-rigid-body inertias via Eq. 6.17;

2. calculating $f_i = I_i^c\, s_\alpha$ in body i coordinates;
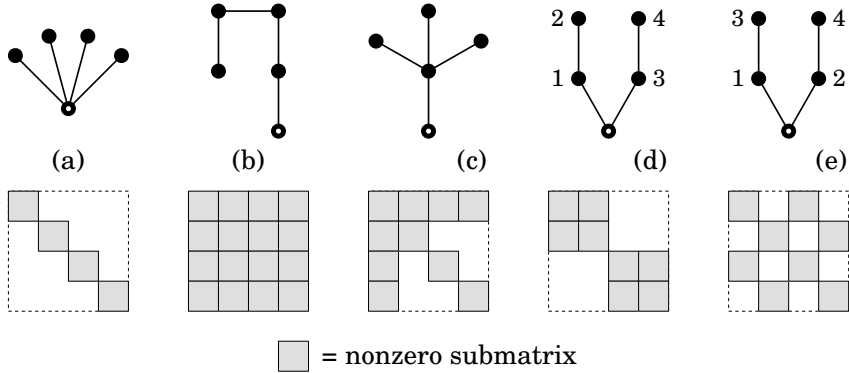
Figure 6.2: Examples of branch-induced sparsity

3. transforming $\boldsymbol{f}_i$ to the coordinate system of each body in $\kappa(i)$; and

4. calculating the elements of $\boldsymbol{H}$ in the portion of column α above the main diagonal, and copying them to the portion of row α below the main diagonal, the two being the same because of symmetry.

The only difference between this algorithm and the one in Section 6.2 is that the latter processes the columns in blocks, rather than individually, one block being all of the columns pertaining to a single joint. If joint i has n_i degrees of freedom, then the quantity ${}^i\boldsymbol{F}_i$ in Eq. 6.19 is simply a $6 \times n_i$ matrix whose columns are the forces $\boldsymbol{f}_i = \boldsymbol{I}_i^c \boldsymbol{s}_\alpha$ for each α belonging to joint i.

6.4   Branch-Induced Sparsity

Equation 6.14 implies that some elements of $\boldsymbol{H}$ will automatically be zero simply because of the connectivity. According to this equation, $\boldsymbol{H}_{ij}$ will be zero whenever i is neither an ancestor of j, nor a descendant, nor equal to j. This can happen only if i and j are on different branches of the tree, so we call it branch-induced sparsity.

Figure 6.2 shows some simple connectivity graphs and the sparsity patterns they produce. Example (a) shows an extreme case in which every body is connected directly to the base, resulting in the greatest possible amount of branch-induced sparsity. In this case, every off-diagonal submatrix is zero. More generally, $\boldsymbol{H}$ will always be block diagonal (or a symmetric permutation thereof) if the base has more than one child. Example (b) shows the opposite extreme: the tree has no branches, and so there is no branch-induced sparsity. Example (c) shows a more typical sparsity pattern; and examples (d) and (e) illustrate the effect of the numbering scheme: two different numberings of the same graph produce matrices that are symmetric permutations of each other.
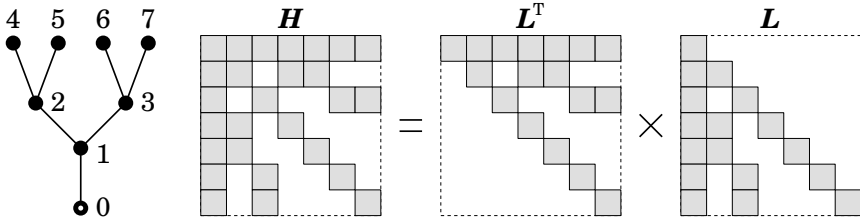
Figure 6.3: Preservation of sparsity in the factorization $\boldsymbol{H} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$

In principle, these two matrices have the same underlying sparsity pattern, and the choice of numbering does not affect our ability to exploit it.

It is often the case that branch-induced sparsity can affect a substantial proportion of the elements of $\boldsymbol{H}$. We can get a rough estimate of its beneficial effect using the following rule of thumb: if $\boldsymbol{H}$ has a density of ρ, then the cost of calculating it is roughly ρ times the cost of calculating a dense $\boldsymbol{H}$ of the same size, and the cost of factorizing it is roughly ρ^2 times the cost of factorizing a dense matrix of the same size. ρ is defined to be the proportion of elements in $\boldsymbol{H}$ that are not branch-induced zeros, so $0 < \rho \leq 1$. It is not unusual to encounter densities of around 0.5, and densities close to zero are possible. Overall, the effect of branch-induced sparsity is to make inertia-matrix methods more efficient on branched kinematic trees than they are on unbranched trees of the same size.

The composite-rigid-body algorithm automatically exploits branch-induced sparsity, simply by calculating only the nonzero submatrices of $\boldsymbol{H}$. However, if we were to try and factorize the resulting matrix using a standard factorization algorithm, then it would treat the matrix as dense, and perform $O(n^3)$ arithmetic operations. Therefore, in order to fully exploit the sparsity, we need a factorization algorithm for matrices containing branch-induced sparsity. This turns out to be an easy problem to solve, the solution being to factorize $\boldsymbol{H}$ into either $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ or $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{DL}$, and design the factorization algorithm to skip over the branch-induced zeros.

In the sparse matrix literature, the factorization $\boldsymbol{H} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ would be described as a reordered Cholesky factorization, meaning that it is equivalent to performing a standard Cholesky factorization on a permutation of the original matrix (George and Liu, 1981). Likewise, the factorization $\boldsymbol{H} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{DL}$ would be described as a reordered $\boldsymbol{LDL}^{\mathrm{T}}$ factorization. This implies that the $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ and $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{DL}$ factorizations have the same numerical properties as the standard Cholesky and $\boldsymbol{LDL}^{\mathrm{T}}$ factorizations.

The special property of an $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ or $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{DL}$ factorization, when applied to a matrix containing branch-induced sparsity, is that the factorization proceeds without *fill-in*. In other words, every branch-induced zero element in the matrix remains zero throughout the factorization process. (This is proved in Featherstone (2005).) A factorization with this property is said to be optimal (Duff

LTL factorization:

for $k = n$ **to** 1 **do**
   $H_{kk} = \sqrt{H_{kk}}$
   $i = \lambda(k)$
   while $i \neq 0$ **do**
      $H_{ki} = H_{ki}/H_{kk}$
      $i = \lambda(i)$
   end
   $i = \lambda(k)$
   while $i \neq 0$ **do**
      $j = i$
      while $j \neq 0$ **do**
         $H_{ij} = H_{ij} - H_{ki}\,H_{kj}$
         $j = \lambda(j)$
      end
      $i = \lambda(i)$
   end
end

LTDL factorization:

for $k = n$ **to** 1 **do**
   $i = \lambda(k)$
   while $i \neq 0$ **do**
      $a = H_{ki}/H_{kk}$
      $j = i$
      while $j \neq 0$ **do**
         $H_{ij} = H_{ij} - a\,H_{kj}$
         $j = \lambda(j)$
      end
      $H_{ki} = a$
      $i = \lambda(i)$
   end
end

Table 6.3: $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ and $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$ factorization algorithms

et al., 1986). One immediate consequence is that the sparsity pattern of $\boldsymbol{H}$ is preserved in the resulting factors, which are therefore also sparse. Figure 6.3 illustrates this effect, showing both the original pattern and the pattern in the resulting triangular factors. This sparsity can be exploited during back-substitution. A second consequence of the no-fill-in property is that the branch-induced zeros can be ignored completely during the factorization process.

6.5   Sparse Factorization Algorithms

Table 6.3 presents the pseudocode for the sparse $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ and $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$ factorization algorithms. These algorithms expect to be given two inputs: an $n \times n$ matrix $\boldsymbol{H}$ and an n-element integer array λ. $\boldsymbol{H}$ is expected to be symmetric and positive definite, and the elements of λ are expected to satisfy $0 \leq \lambda(i) < i$. In addition, $\boldsymbol{H}$ is expected to have the following sparsity pattern, which is optimally exploited by the algorithms: *On each row k of $\boldsymbol{H}$, the nonzero elements below the main diagonal appear only in columns $\lambda(k)$, $\lambda(\lambda(k))$, and so on.* This rule identifies the set of elements that the algorithms will treat as being nonzero. All other elements are assumed to be zero, and are ignored. If $\lambda(k) = k - 1$ for all k, then the algorithms treat every element as nonzero, in which case they perform a dense-matrix $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ or $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$ factorization.

    Figure 6.4 shows how the algorithms work. They both operate *in situ* on the given matrix, and never access any element above the main diagonal. The $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$

LTL

1. $H_{kk} = \sqrt{H_{kk}}$

2. $H_{ki} = H_{ki}/H_{kk}$

3. $H_{ij} = H_{ij} - H_{ki}\,H_{kj}$

LTDL

1. do nothing

2a. $H'_{ki} = H_{ki}/H_{kk}$

3. $H_{ij} = H_{ij} - H'_{ki}\,H_{kj}$

2b. $H_{ki} = H'_{ki}$
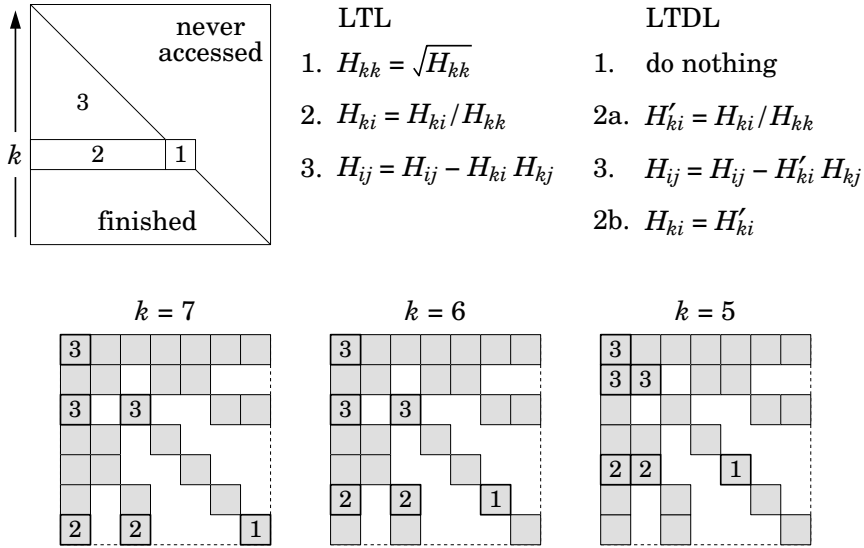
Figure 6.4: Illustration of the factorization process

algorithm computes $\boldsymbol{L}$ and returns it in the lower triangle of $\boldsymbol{H}$. The $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$ algorithm computes $\boldsymbol{D}$ and $\boldsymbol{L}$, returns $\boldsymbol{D}$ on the main diagonal, and returns the off-diagonal elements of $\boldsymbol{L}$ below it. This works because the algorithm computes a unit lower-triangular matrix, so its diagonal elements are known to have the value 1, and therefore do not need to be returned.

Each algorithm has an outer loop that visits each row in turn, working from n back to 1. At any stage in the factorization process, rows $k+1$ to n are finished, and contain rows $k+1$ to n of the returned factors. Rows 1 to k can be divided into three areas, as shown in the diagram. Area 1 consists of just the element H_{kk}; area 2 consists of elements 1 to $k-1$ of row k; and area 3 consists of the triangular region from rows 1 to $k-1$. A different calculation takes place in each area, as shown in the figure. The pseudocode for the $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ factorization performs the area-2 and area-3 calculations in two separate loops; but the pseudocode for the $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}$ factorization interleaves these calculations in such a way that it never needs to remember more than one H'_{ki} value at any point. The current remembered value is stored in the local variable a.

The inner loops are designed to iterate only over the values $\lambda(k)$, $\lambda(\lambda(k))$, and so on. This is where the sparsity is exploited. In effect, the algorithms know where the zeros are, and simply skip over them. Figure 6.4 illustrates the cost reduction by showing the first three steps in the factorization of the matrix $\boldsymbol{H}$ from Figure 6.3. At $k=7$, the algorithms perform two lots of the area-2 calculation and three lots of the area-3 calculation; and similarly at $k=6$ and $k=5$. This is far fewer than the number of calculations that would have been necessary if there had been no zeros.

```
for i = 1 to n do
    λ'(i) = i − 1
end
map(0) = 0
for i = 1 to N_B do
    map(i) = map(i − 1) + n_i
end
for i = 1 to N_B do
    λ'(map(i − 1) + 1) = map(λ(i))
end
```
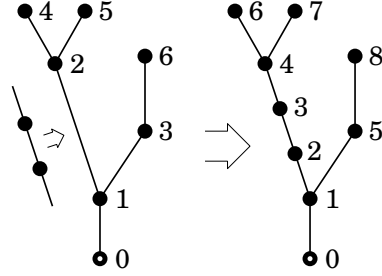
Table 6.4: Algorithm for calculating the expanded parent array, and illustration of the expanded connectivity graph

Expanding the Parent Array

The parent array for a kinematic tree contains N_B elements, but the factorization algorithms require an array with n elements. If $n = N_B$ then no action is required. Otherwise, an expanded parent array must be constructed for use in the factorization process.

Table 6.4 presents an algorithm to do this, and an illustration of what the algorithm does. Essentially, we start with the original connectivity graph, and replace each multi-freedom joint with a chain of single-freedom joints. This produces an expanded graph in which every arc represents a single joint variable. This new graph is renumbered so that arc i represents the ith variable in $\ddot{\boldsymbol{q}}$; and the expanded parent array is computed from this graph. The illustration shows the original and expanded connectivity graphs for a kinematic tree in which joint 2 has three degrees of freedom, but every other joint has only one. Arc 2 is therefore replaced by a chain of three arcs, and the new tree is renumbered so that each arc number equals the index of its variable in $\ddot{\boldsymbol{q}}$. As the variables for joint 2 occupy the second, third and fourth places in $\ddot{\boldsymbol{q}}$, the new arcs must be numbered 2, 3 and 4. The parent array for the original graph is $(0, 1, 1, 2, 2, 3)$; and the parent array for the expanded graph is $(0, 1, 2, 3, 1, 4, 4, 5)$.

The algorithm in Table 6.4 calculates an expanded parent array, λ', directly from the original. The first step is to initialize λ' to the array $(0, 1, 2, \ldots, n-1)$. This is a clever trick that correctly initializes $n - N_B$ of the elements in λ', leaving only N_B elements to be computed from λ. The second loop constructs a mapping array such that $map(i) = \sum_{j=1}^{i} n_j$, where n_j is the degree of freedom of joint j. (The variables of joint i appear in $\ddot{\boldsymbol{q}}$ at locations $map(i − 1) + 1$ to $map(i)$ inclusive.) The third loop performs two mappings rolled into one: the value stored in $\lambda(i)$ is converted to $map(\lambda(i))$, which is the correct value to be inserted into λ'; and this value is inserted into λ' at location $map(i − 1) + 1$, which is the correct place to put it. For the example shown in the table, the mapping array is $(0, 1, 4, 5, 6, 7, 8)$ (indexed from zero); the two entries in λ'

$\underline{\boldsymbol{y} = \boldsymbol{L}\,\boldsymbol{x}}$

for $i = n$ **to** 1 **do**
   $y_i = L_{ii}\,x_i$
   $j = \lambda(i)$
   while $j \neq 0$ **do**
      $y_i = y_i + L_{ij}\,x_j$
      $j = \lambda(j)$
   end
end

$\underline{\boldsymbol{y} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}}$

for $i = 1$ **to** n **do**
   $y_i = L_{ii}\,x_i$
   $j = \lambda(i)$
   while $j \neq 0$ **do**
      $y_j = y_j + L_{ij}\,x_i$
      $j = \lambda(j)$
   end
end

$\underline{\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}}$

for $i = 1$ **to** n **do**
   $y_i = H_{ii}\,x_i$
end
for $i = n$ **to** 1 **do**
   $j = \lambda(i)$
   while $j \neq 0$ **do**
      $y_i = y_i + H_{ij}\,x_j$
      $y_j = y_j + H_{ij}\,x_i$
      $j = \lambda(j)$
   end
end

$\underline{\boldsymbol{x} = \boldsymbol{L}^{-1}\boldsymbol{x}}$

for $i = 1$ **to** n **do**
   $j = \lambda(i)$
   while $j \neq 0$ **do**
      $x_i = x_i - L_{ij}\,x_j$
      $j = \lambda(j)$
   end
   $x_i = x_i/L_{ii}$
end

$\underline{\boldsymbol{x} = \boldsymbol{L}^{-\mathrm{T}}\boldsymbol{x}}$

for $i = n$ **to** 1 **do**
   $x_i = x_i/L_{ii}$
   $j = \lambda(i)$
   while $j \neq 0$ **do**
      $x_j = x_j - L_{ij}\,x_i$
      $j = \lambda(j)$
   end
end

Table 6.5: Multiplication and back-substitution algorithms

that are correctly initialized in the first loop are $\lambda'(3)$ and $\lambda'(4)$; the elements $\lambda(1) \cdots \lambda(3)$ are mapped unaltered to $\lambda'(1)$, $\lambda'(2)$ and $\lambda'(5)$; and $\lambda(4) \cdots \lambda(6)$ are incremented by 2 and mapped to $\lambda'(6) \cdots \lambda'(8)$.

Having computed the expanded parent array once, it should be stored in the system model for future use by the factorization algorithms.

Back-Substitution

The sparsity pattern in $\boldsymbol{H}$ appears also in $\boldsymbol{L}$; so it is possible to exploit the sparsity in $\boldsymbol{L}$ using the same techniques as for $\boldsymbol{H}$. Table 6.5 presents algorithms to calculate the quantities $\boldsymbol{L}\boldsymbol{x}$, $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}$, $\boldsymbol{L}^{-1}\boldsymbol{x}$ and $\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{x}$ for a general vector $\boldsymbol{x}$, given only λ, $\boldsymbol{L}$ and $\boldsymbol{x}$. Back-substitution makes use of $\boldsymbol{L}^{-1}\boldsymbol{x}$ and $\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{x}$. The table also presents an algorithm to calculate $\boldsymbol{H}\boldsymbol{x}$. In every case, the sparsity is exploited by having the inner loop iterate over only the nonzero elements of $\boldsymbol{L}$ or $\boldsymbol{H}$. The algorithms assume that $\boldsymbol{L}$ is a general lower-triangular factor, such as that produced by the $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ factorization. To modify them to work with a unit lower-triangular factor, simply replace L_{ii} with 1.

The algorithms that calculate $\boldsymbol{L}\boldsymbol{x}$ $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}$ and $\boldsymbol{H}\boldsymbol{x}$ place the result in a separate vector, $\boldsymbol{y}$, and leave $\boldsymbol{x}$ unaltered. The $\boldsymbol{L}\boldsymbol{x}$ algorithm allows $\boldsymbol{y}$ to be the same vector as $\boldsymbol{x}$, in which case it works *in situ*; but the other two require $\boldsymbol{y} \neq \boldsymbol{x}$. The algorithms that calculate $\boldsymbol{L}^{-1}\boldsymbol{x}$ and $\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{x}$ both work *in situ*.

	LTL	LTDL
factorization	$n\sqrt{\ } + D_1\mathsf{d} + D_2(\mathsf{m}+\mathsf{a})$	$D_1\mathsf{d} + D_2(\mathsf{m}+\mathsf{a})$
back-substitution	$2n\mathsf{d} + 2D_1(\mathsf{m}+\mathsf{a})$	$n\mathsf{d} + 2D_1(\mathsf{m}+\mathsf{a})$
$\boldsymbol{L}\boldsymbol{x}$, $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}$	$n\mathsf{m} + D_1(\mathsf{m}+\mathsf{a})$	$D_1(\mathsf{m}+\mathsf{a})$
$\boldsymbol{L}^{-1}\boldsymbol{x}$, $\boldsymbol{L}^{-\mathrm{T}}\boldsymbol{x}$	$n\mathsf{d} + D_1(\mathsf{m}+\mathsf{a})$	$D_1(\mathsf{m}+\mathsf{a})$
$\boldsymbol{H}\boldsymbol{x}$	$n\mathsf{m} + 2D_1(\mathsf{m}+\mathsf{a})$	

Table 6.6: Computational cost of sparse algorithms

To implement $\boldsymbol{y} = \boldsymbol{L}^{-1}\boldsymbol{x}$ or $\boldsymbol{y} = \boldsymbol{L}^{-\mathrm{T}}\boldsymbol{x}$, simply copy $\boldsymbol{x}$ to $\boldsymbol{y}$ and apply the algorithm to $\boldsymbol{y}$.

Computational Cost

Table 6.6 presents computational cost figures for each algorithm. The symbols m, a, d and $\sqrt{\ }$ denote the cost of a floating-point multiplication, addition, division and square-root calculation, respectively. The quantities D_1 and D_2 depend on the sparsity pattern in $\boldsymbol{H}$, and are given by

$$D_1 = \sum_{k=1}^{n} (d_k - 1) \tag{6.26}$$

and

$$D_2 = \sum_{k=1}^{n} \frac{d_k(d_k - 1)}{2}, \tag{6.27}$$

where d_k is the number of joints on the path between body k and the base (i.e., $d_k = |\kappa(k)|$), and can be regarded as the depth of body k in the tree. The expressions $d_k - 1$ and $d_k(d_k - 1)/2$ count the number of times that area-2 and area-3 calculations are performed during the processing of row k; so D_1 and D_2 count the total number of area-2 and area-3 calculations, respectively. D_1 is also the total number of nonzero elements below the main diagonal. Thus, the number of nonzero elements in $\boldsymbol{H}$ is $n + 2D_1$. If there is no branch-induced sparsity, then D_1 and D_2 take their maximum possible values, which are

$$D_1 = (n^2 - n)/2 \qquad \text{and} \qquad D_2 = (n^3 - n)/6. \tag{6.28}$$

Plugging these values into the formulae in Table 6.6 produces cost figures that are identical to those of the standard Cholesky and $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{T}}$ factorization algorithms. If d denotes the depth of the tree, defined as the depth of the deepest body, then D_1 and D_2 are bounded by

$$D_1 \leq n(d - 1) \qquad \text{and} \qquad D_2 \leq nd(d - 1)/2. \tag{6.29}$$

These bounds show that the factorization process is at worst $O(nd^2)$.

6.6 Additional Notes

One of the earliest examples of a dynamics algorithm is due to Uicker (1965, 1967). It combined the 4×4 matrix representation of kinematics developed by Denavit and Hartenberg (1955) with a 4×4 pseudoinertia matrix, and used Lagrangian techniques to obtain the equation of motion for a closed-loop mechanism. At about the same time, Hooker and Margulies (1965) used vectorial (Newton-Euler) dynamics and augmented bodies to obtain the equations of motion for a free-floating kinematic tree representing a satellite. By the mid 1970s, there were already several computer programs available to calculate the forward dynamics of various kinds of rigid-body system; and an excellent review of these early works can be found in Paul (1975). One early idea that proved to be particularly successful is the sparse-matrix formulation used in the program ADAMS (Orlandea et al., 1977). The earliest textbook on computational rigid-body dynamics appeared at this time (Wittenburg, 1977). It presented a detailed method for constructing computer-oriented models of general rigid-body systems, and for calculating their dynamics.

The composite-rigid-body algorithm first appeared as method 3 in Walker and Orin (1982). It provided a new method for calculating the joint-space inertia matrix of a kinematic tree, which was substantially faster than previous methods. More efficient versions of this algorithm, and some variations on the basic idea, appeared subsequently in Featherstone (1984, 1987); Balafoutis and Patel (1989, 1991); Lilly and Orin (1991); Lilly (1993); McMillan and Orin (1998). However, the phenomenon of branch-induced sparsity was not recognized until Featherstone (2005).

The first example of a propagation method for forward dynamics appeared in Vereshchagin (1974). In this paper, the dynamics was formulated using Gauss' principle of least constraint, and the resulting minimization solved via dynamic programming. The algorithm in this paper is practically the same as the articulated-body algorithm, but its significance was not recognized until after the articulated-body algorithm had been published. The next example of a propagation algorithm appeared in Armstrong (1979), followed by the preliminary version of the articulated-body algorithm in Featherstone (1983), and then the complete version in Featherstone (1984, 1987). More efficient versions then appeared in Brandl et al. (1988); McMillan and Orin (1995); McMillan et al. (1995). A few other algorithms have turned out to be practically the same as the articulated-body algorithm; for example, the forward dynamics algorithm of Rodriguez et al. (Rodriguez, 1987; Rodriguez et al., 1991) and the $O(n)$ algorithm in Rosenthal (1990).

Superficially, it would seem that the inertia-matrix methods and propagation methods are very different. However, two interesting connections have been found. Drawing on Kalman filter theory, Rodriguez et al. present two factorizations of the joint-space inertia matrix: one implies the recursive Newton-Euler algorithm, while the other has an inverse that implies the articulated-body algorithm (Rodriguez, 1987; Rodriguez et al., 1991). The other connection was

found by Ascher et al. (1997). Starting with an equation similar in form to Eq. 3.18, they show that the joint-space inertia matrix can be obtained by applying Gaussian elimination to one permutation of the coefficient matrix, while the articulated-body algorithm arises from applying Gaussian elimination to another permutation of this matrix.

Strictly speaking, inertia-matrix and propagation methods are only two out of three possible strategies for calculating the forward dynamics of a kinematic tree. The third possibility is to assemble the equations of motion of the individual bodies, together with the equations of constraint, to form a single large matrix equation (as shown in Chapter 3, for example). This strategy is very useful for closed-loop systems, but it is not competitive with inertia-matrix and propagation methods when applied to kinematic trees. An example of this class of algorithm can be found in Baraff (1996).

Chapter 7

Forward Dynamics — Propagation Methods

The forward dynamics problem presents us with two sets of unknowns: the joint accelerations and the joint constraint forces. It is usually not possible to solve for any of these unknowns locally at any one body; but it is possible to formulate equations that they must satisfy. Propagation methods work by calculating the coefficients of such equations locally, and propagating them to neighbouring bodies, until we eventually reach a point where we *can* solve the dynamics locally at one particular body. Having the solution at this one body, it becomes possible to solve the dynamics at neighbouring bodies, and so on, until the whole problem is solved. In principle, the process is akin to solving a linear equation by first triangularizing the coefficient matrix, and then solving it by back-substitution. Indeed, some propagation algorithms have been formulated via this analogy (e.g. see Ascher et al., 1997; Rosenthal, 1990; Saha, 1997). Propagation algorithms are more complicated than inertia-matrix algorithms, but they have a computational complexity of $O(N_B)$, which is the theoretical minimum for solving the forward dynamics problem.

The articulated-body algorithm is the prime example of a propagation algorithm, and it is the fastest for calculating the forward dynamics of a kinematic tree. Most of this chapter is devoted to a description of this algorithm and its attendant concepts, like articulated-body inertia. The final two sections take a broader view of the subject, and present some of the techniques and equations that form the basis of other algorithms.

7.1 Articulated-Body Inertia

Articulated-body inertia is the inertia that a body appears to have when it is part of a rigid-body system. Consider the rigid body B shown in diagram (a) of Figure 7.1. The relationship between the applied force, $\boldsymbol{f}$, and the resulting
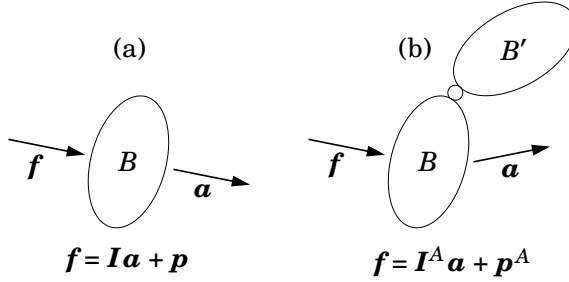
Figure 7.1: Comparing a rigid body (a) with an articulated body (b)

acceleration, $\boldsymbol{a}$, is given by the body's equation of motion:

$$\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p}. \tag{7.1}$$

The coefficients $\boldsymbol{I}$ and $\boldsymbol{p}$ in this equation are the rigid-body inertia and bias force, respectively, of body B. Now consider the two-body system shown in diagram (b), in which a second body, B', has been connected to B via a joint. The effect of this second body is to alter the acceleration response of B, so that the relationship between $\boldsymbol{f}$ and $\boldsymbol{a}$ is now given by

$$\boldsymbol{f} = \boldsymbol{I}^A\,\boldsymbol{a} + \boldsymbol{p}^A. \tag{7.2}$$

This is an *articulated-body equation of motion*. It has the same algebraic form as Eq. 7.1, but different coefficients. The quantities $\boldsymbol{I}^A$ and $\boldsymbol{p}^A$ in this equation are the articulated-body inertia and bias force of body B in the two-body system comprising B, B' and the connecting joint.

An equation like 7.2 describes the acceleration response of a single rigid body, taking into account the dynamic effects of a specific set of other bodies and joints. We call this body a *handle*, and we call the system containing it an *articulated body*. Thus, the handle is the body to which $\boldsymbol{I}^A$ and $\boldsymbol{p}^A$ refer; and the articulated body is the rigid-body system, or subsystem, whose dynamic effects are included in $\boldsymbol{I}^A$ and $\boldsymbol{p}^A$. In Figure 7.1(b), the handle is body B and the articulated body is the whole two-body system.

Basic Properties

Articulated-body inertias share the following properties with rigid-body inertias: they are symmetric, positive-definite matrices; they are mappings from M^6 to F^6; and they obey the same coordinate transformation rule. However, articulated-body inertias are mappings from acceleration to force, not velocity to momentum, so expressions like $\boldsymbol{I}^A\boldsymbol{v}$ and $\frac{1}{2}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{I}^A\boldsymbol{v}$, where $\boldsymbol{v}$ denotes a velocity, do not make sense. An articulated body may consist of just a single rigid body, in which case the articulated-body and rigid-body inertias and bias forces are the same. This provides a starting point for recursive calculation methods.
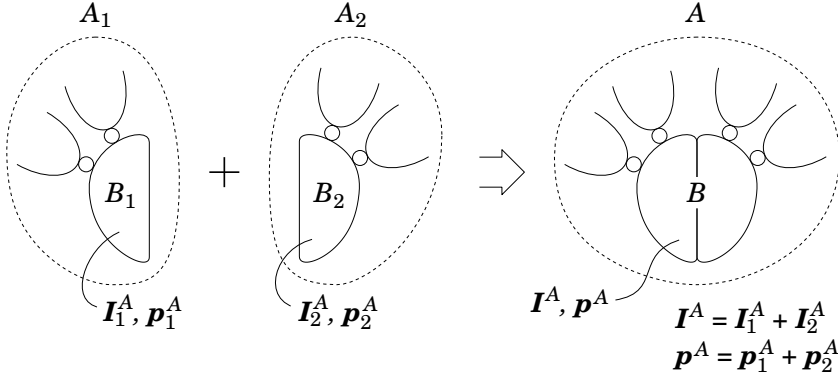
Figure 7.2: Addition of articulated-body inertias and bias forces

Articulated-body inertias depend only on the inertias of the member bodies and the constraints imposed by the member joints. Thus, they are functions of the joint position variables, but not the velocity variables or the various force terms. In this respect, they resemble generalized inertia matrices. The velocity terms, external force terms (other than $\boldsymbol{f}$), and so on, appear only in the bias forces. In 3D space, 21 parameters are required to define a general articulated-body inertia, compared with only 10 for a rigid-body inertia. In 2D space, 6 parameters are required to define a general articulated-body inertia, compared with only 4 for a rigid-body inertia. Note that 21 and 6 are the numbers of independent values in a symmetric 6×6 and 3×3 matrix, respectively. In general, the apparent mass of an articulated-body inertia varies with the direction of applied force, and it does not have a centre of apparent mass. (See Example 7.1.)

Addition

Addition is defined on articulated-body inertias, and the physical interpretation of this operation is illustrated in Figure 7.2. This figure shows two articulated bodies, A_1 and A_2, that are being joined to form a single articulated body, A. The connection is made by gluing bodies B_1 and B_2 together to form a new body, B. If the symbols $\boldsymbol{I}_i^A$, $\boldsymbol{p}_i^A$, $\boldsymbol{I}^A$ and $\boldsymbol{p}^A$ denote the articulated-body inertias and bias forces for B_i in A_i and B in A, respectively, then $\boldsymbol{I}^A = \boldsymbol{I}_1^A + \boldsymbol{I}_2^A$ and $\boldsymbol{p}^A = \boldsymbol{p}_1^A + \boldsymbol{p}_2^A$.

Inverse Inertia

Equation 7.2 assumes the handle has six degrees of freedom. If this is not the case, then the more general equation

$$\boldsymbol{a} = \boldsymbol{\Phi}^A \boldsymbol{f} + \boldsymbol{b}^A \tag{7.3}$$

must be used instead. In this equation, $\boldsymbol{\Phi}^A$ and $\boldsymbol{b}^A$ are the handle's articulated-body inverse inertia and bias acceleration, respectively (cf. Eq. 2.72). An equation in this form always exists. If the handle has six degrees of freedom, then $\boldsymbol{\Phi}^A = (\boldsymbol{I}^A)^{-1}$ and $\boldsymbol{b}^A = -\boldsymbol{\Phi}^A \boldsymbol{p}^A$. Otherwise, $\boldsymbol{\Phi}^A$ is singular, and its rank equals the handle's degree of motion freedom. Articulated-body inverse inertias are symmetric, positive-semidefinite matrices, and they transform like rigid-body inverse inertias.

Multiple Handles

It is possible for an articulated body to have more than one handle. If we number the handles from 1 to h, and define the vectors $\boldsymbol{f}_i$ and $\boldsymbol{a}_i$ to be the force and acceleration associated with handle i, then Eq. 7.3 generalizes to

$$\begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_h \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_1^A & \boldsymbol{\Phi}_{12}^A & \cdots & \boldsymbol{\Phi}_{1h}^A \\ \boldsymbol{\Phi}_{21}^A & \boldsymbol{\Phi}_2^A & \cdots & \boldsymbol{\Phi}_{2h}^A \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Phi}_{h1}^A & \boldsymbol{\Phi}_{h2}^A & \cdots & \boldsymbol{\Phi}_h^A \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \vdots \\ \boldsymbol{f}_h \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_1^A \\ \boldsymbol{b}_2^A \\ \vdots \\ \boldsymbol{b}_h^A \end{bmatrix} . \qquad (7.4)$$

In this equation, $\boldsymbol{\Phi}_i^A$ and $\boldsymbol{b}_i^A$ are the articulated-body inverse inertia and bias acceleration for handle i, and $\boldsymbol{\Phi}_{ij}^A$ is the cross-coupling term that describes the acceleration caused at handle i by the application of a force to handle j. The coefficient matrix is symmetric and positive-semidefinite, and its rank equals the total number of independent degrees of motion freedom possessed by the set of handles. If the rank is $6h$ then Eq. 7.4 can be inverted to obtain a multi-handle equivalent of Eq. 7.2.
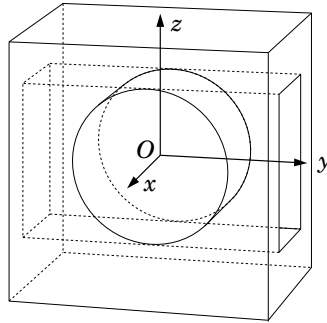
Figure 7.3: An articulated body

Example 7.1 *Figure 7.3 shows an articulated body consisting of a box with a slot, and a cylinder that fits inside the slot. The cylinder has two degrees of freedom relative to the box: it can rotate about the x axis, and it can translate in the y direction. The contact between the two bodies is assumed to*

be frictionless. Both bodies have their centres of mass at the origin. The box has a mass of m_1 and a rotational inertia about its centre of mass of I_1 (i.e., its rotational inertia matrix is I_1 times the identity matrix); and the cylinder has a mass of m_2 and a rotational inertia of I_2.

Let us treat the box as the handle, and calculate its articulated-body inertia. If we apply a force to the box along either the x or the z axis, then the two bodies will move as one, and the resulting acceleration will be $1/(m_1 + m_2)$ times the magnitude of the applied force; but if we apply a force along the y axis, then only the box will move, and the resulting acceleration will be $1/m_1$ times the magnitude of the applied force. Likewise, if a pure couple is applied to the box about either the y or the z axes, then the two bodies will rotate as one, and the apparent inertia will be $I_1 + I_2$; but a pure couple about the x axis will cause only the box to rotate, and the apparent inertia will be I_1. The complete articulated-body inertia matrix is therefore

$$I^A = \begin{bmatrix} I_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & I_1 + I_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & I_1 + I_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_1 + m_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & m_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & m_1 + m_2 \end{bmatrix}.$$

Observe that the apparent mass varies with direction: the box appears to have more mass in the x and z directions than in the y direction.

In this particular example, the handle exhibits a centre of apparent mass, which is located at the origin. Any pure force applied to the handle along a line passing through this point causes a pure linear acceleration of the handle. Such a point does not exist in general.

7.2 Calculating Articulated-Body Inertias

There are two basic strategies for calculating articulated-body inertias: projection and assembly. In the projection method, the equation of motion for the whole articulated body is projected onto the motion space of the handle (or handles). In the assembly method, an articulated body is assembled step-by-step from its component parts, and a sequence of articulated-body inertias are computed along the way. The projection method is commonly used to calculate operational-space inertias, which are used in robot control systems (Khatib, 1987, 1995); and the assembly method is used in $O(n)$ dynamics algorithms.

7.2.1 The Projection Method

An articulated body is a rigid-body system. It therefore has an equation of motion that can be expressed in generalized coordinates as

$$H\ddot{q} + C = \tau, \tag{7.5}$$

where C accounts for every force acting on the system except the force acting on the handle. Let v, a and f denote the spatial velocity and acceleration of the handle, and the spatial force acting on it. If J is the Jacobian that maps $\dot{q}$ to v according to $v = J\dot{q}$, then it defines the following relationships between f and τ, and between a and $\ddot{q}$:

$$\tau = J^{\mathrm{T}} f \tag{7.6}$$

and

$$a = J\ddot{q} + \dot{J}\dot{q}. \tag{7.7}$$

Combining these equations gives

$$
\begin{aligned}
a &= JH^{-1}(J^{\mathrm{T}}f - C) + \dot{J}\dot{q} \\
&= \Phi^A f + b^A,
\end{aligned}
\tag{7.8}
$$

where

$$\Phi^A = JH^{-1}J^{\mathrm{T}} \tag{7.9}$$

and

$$b^A = \dot{J}\dot{q} - JH^{-1}C. \tag{7.10}$$

These are the general equations for an articulated-body inverse inertia and bias acceleration. They apply to any articulated body, and they are independent of the choice of generalized coordinates. If the handle has a full six degrees of motion freedom, then Φ^A is invertible, and we can define

$$f = I^A a + p^A, \tag{7.11}$$

where

$$I^A = (JH^{-1}J^{\mathrm{T}})^{-1} \tag{7.12}$$

and

$$p^A = -I^A b^A. \tag{7.13}$$

Many of the basic properties of articulated-body inertias can be deduced from Eq. 7.12. For example, given that H is symmetric and positive-definite, it follows that I^A must also be symmetric and positive-definite; and it is obvious that I^A does not depend on velocity or force terms. Some basic properties of Φ^A that follow from Eq. 7.9 are:

$$
\begin{aligned}
\mathrm{rank}(\Phi^A) &= \mathrm{rank}(J), \\
\mathrm{range}(\Phi^A) &= \mathrm{range}(J), \\
\mathrm{null}(\Phi^A) &= \mathrm{null}(J^{\mathrm{T}}).
\end{aligned}
\tag{7.14}
$$
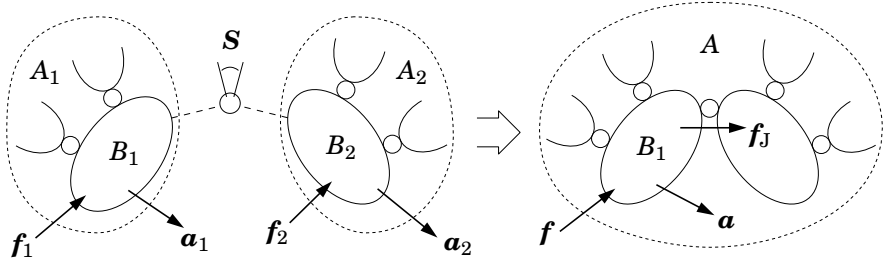
Figure 7.4: Construction of a larger articulated body from two smaller ones

Multiple Handles

The projection method is easily extended to articulated bodies with more than one handle. Suppose there are h handles, and let $\boldsymbol{a}_i$, $\boldsymbol{f}_i$ and $\boldsymbol{J}_i$ be the acceleration, force and Jacobian pertaining to handle i. If we redefine the quantities $\boldsymbol{a}$, $\boldsymbol{f}$ and $\boldsymbol{J}$ to be

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_h \end{bmatrix}, \qquad \boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_h \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_1 \\ \vdots \\ \boldsymbol{J}_h \end{bmatrix},$$

then Eqs. 7.6 to 7.14 immediately apply to the multiple-handle case.

7.2.2 The Assembly Method

In the assembly method, we regard a given articulated body as an assembly of smaller articulated bodies; and we calculate its inertia and bias force from those of its component parts. Thus, the assembly method is suitable for use in recursive algorithms. In principle, the assembly method is general. However, we shall consider here only the special case in which every articulated body is a *floating kinematic tree*—a kinematic tree with no connection to a fixed base. This is the case that is needed for the articulated-body algorithm. The special properties of this case are: that every articulated body has one handle; that every handle has a full six degrees of freedom; and that every assembly operation consists of connecting one handle to another via a single joint. More general cases are covered in Sections 7.4 and 7.5.

Figure 7.4 shows a pair of articulated bodies, A_1 and A_2, in the process of being assembled into a larger articulated body, A. The assembly operation consists of connecting the two handles, B_1 and B_2, with a joint. Prior to the assembly, the articulated-body equations for B_1 and B_2 are

$$\boldsymbol{f}_1 = \boldsymbol{I}_1^A \, \boldsymbol{a}_1 + \boldsymbol{p}_1^A \tag{7.15}$$

and

$$\boldsymbol{f}_2 = \boldsymbol{I}_2^A \, \boldsymbol{a}_2 + \boldsymbol{p}_2^A . \tag{7.16}$$

After the assembly, B_1 serves as the handle for the new articulated body, and its equation is

$$\boldsymbol{f} = \boldsymbol{I}^A \boldsymbol{a}_1 + \boldsymbol{p}^A \,. \tag{7.17}$$

The objective is to express $\boldsymbol{I}^A$ and $\boldsymbol{p}^A$ in terms of $\boldsymbol{I}_1^A$, $\boldsymbol{I}_2^A$, $\boldsymbol{p}_1^A$, $\boldsymbol{p}_2^A$ and the motion constraint imposed by the new joint.

When the assembly is made, the new joint introduces a new force, $\boldsymbol{f}_J$, which is the total force transmitted through the joint from B_1 to B_2. The relationships between the force variables are therefore

$$\begin{aligned} \boldsymbol{f}_1 &= \boldsymbol{f} - \boldsymbol{f}_J \,, \\ \boldsymbol{f}_2 &= \boldsymbol{f}_J \,. \end{aligned}$$

The value of $\boldsymbol{f}_J$ is unknown, but it imposes the motion constraint

$$\boldsymbol{a}_2 - \boldsymbol{a}_1 = \boldsymbol{S}\ddot{\boldsymbol{q}} + \boldsymbol{c} \,,$$

and it satisfies

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{f}_J = \boldsymbol{\tau} \,.$$

In these equations, $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ are the joint's acceleration and force variables, respectively, $\boldsymbol{S}$ is the joint's motion subspace matrix, and $\boldsymbol{c} = \dot{\boldsymbol{S}}\dot{\boldsymbol{q}}$ (or $\dot{\boldsymbol{S}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{\sigma}}$ if the joint depends on time). $\ddot{\boldsymbol{q}}$ is unknown, but $\boldsymbol{\tau}$, $\boldsymbol{S}$ and $\boldsymbol{c}$ are given. By combining these equations with Eq. 7.16, we can calculate $\ddot{\boldsymbol{q}}$ as follows:

$$\begin{aligned} \boldsymbol{\tau} &= \boldsymbol{S}^{\mathrm{T}}\boldsymbol{f}_2 \\ &= \boldsymbol{S}^{\mathrm{T}}(\boldsymbol{I}_2^A \boldsymbol{a}_2 + \boldsymbol{p}_2^A) \\ &= \boldsymbol{S}^{\mathrm{T}}(\boldsymbol{I}_2^A(\boldsymbol{a}_1 + \boldsymbol{c} + \boldsymbol{S}\ddot{\boldsymbol{q}}) + \boldsymbol{p}_2^A) \,, \end{aligned}$$

so

$$\ddot{\boldsymbol{q}} = (\boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}_2^A \boldsymbol{S})^{-1} (\boldsymbol{\tau} - \boldsymbol{S}^{\mathrm{T}}(\boldsymbol{I}_2^A(\boldsymbol{a}_1 + \boldsymbol{c}) + \boldsymbol{p}_2^A)) \,. \tag{7.18}$$

The matrix $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}_2^A \boldsymbol{S}$ is positive definite, and therefore invertible. Having solved for $\ddot{\boldsymbol{q}}$, we can now construct an equation relating $\boldsymbol{f}$ to $\boldsymbol{a}_1$ as follows:

$$\begin{aligned} \boldsymbol{f} &= \boldsymbol{f}_1 + \boldsymbol{f}_2 \\ &= \boldsymbol{I}_1^A \boldsymbol{a}_1 + \boldsymbol{I}_2^A \boldsymbol{a}_2 + \boldsymbol{p}_1^A + \boldsymbol{p}_2^A \\ &= \boldsymbol{I}_1^A \boldsymbol{a}_1 + \boldsymbol{I}_2^A(\boldsymbol{a}_1 + \boldsymbol{c} + \boldsymbol{S}\ddot{\boldsymbol{q}}) + \boldsymbol{p}_1^A + \boldsymbol{p}_2^A \\ &= \boldsymbol{I}_1^A \boldsymbol{a}_1 + \boldsymbol{I}_2^A(\boldsymbol{a}_1 + \boldsymbol{c} + \boldsymbol{S}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}_2^A \boldsymbol{S})^{-1}(\boldsymbol{\tau} - \boldsymbol{S}^{\mathrm{T}}(\boldsymbol{I}_2^A(\boldsymbol{a}_1 + \boldsymbol{c}) + \boldsymbol{p}_2^A))) \\ &\quad + \boldsymbol{p}_1^A + \boldsymbol{p}_2^A \,. \end{aligned}$$

This equation has the same form as Eq. 7.17, and yields the following formulae for $\boldsymbol{I}^A$ and $\boldsymbol{p}^A$:

$$\boldsymbol{I}^A = \boldsymbol{I}_1^A + \boldsymbol{I}_2^A - \boldsymbol{I}_2^A \boldsymbol{S} (\boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}_2^A \boldsymbol{S})^{-1} \boldsymbol{S}^{\mathrm{T}}\boldsymbol{I}_2^A \tag{7.19}$$

and

$$\boldsymbol{p}^A = \boldsymbol{p}_1^A + \boldsymbol{p}_2^A + \boldsymbol{I}_2^A \boldsymbol{c} + \boldsymbol{I}_2^A \boldsymbol{S} \, (\boldsymbol{S}^\mathrm{T} \boldsymbol{I}_2^A \boldsymbol{S})^{-1} \, (\boldsymbol{\tau} - \boldsymbol{S}^\mathrm{T} (\boldsymbol{I}_2^A \boldsymbol{c} + \boldsymbol{p}_2^A)) \,. \qquad (7.20)$$

With these formulae, we can compute the articulated-body inertia and bias force for any handle in any articulated body, provided only that the articulated body is a floating kinematic tree.

Assembling Subtrees

The three articulated bodies in Figure 7.4 were labelled A_1, A_2 and A merely for identification purposes, with no particular significance in the numbers 1 and 2. In practice, we would typically define a set of articulated bodies, $A_1 \cdots A_{N_B}$, such that A_i contained all of the bodies in the subtree rooted at body i, with body i serving as the handle (e.g. see Figure 7.5). In this circumstance, the most useful assembly formulae are the ones that tell how to calculate $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ from the inertias and bias forces pertaining to the children of body i; that is, the quantities $\boldsymbol{I}_j^A$ and $\boldsymbol{p}_j^A$ for all $j \in \mu(i)$. Without repeating the algebra, these formulae are

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \sum_{j \in \mu(i)} \boldsymbol{I}_j^a \qquad (7.21)$$

and

$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \sum_{j \in \mu(i)} \boldsymbol{p}_j^a \,, \qquad (7.22)$$

where $\boldsymbol{I}_i$ and $\boldsymbol{p}_i$ are the rigid-body inertia and bias force for body i, and

$$\boldsymbol{I}_j^a = \boldsymbol{I}_j^A - \boldsymbol{I}_j^A \boldsymbol{S}_j \, (\boldsymbol{S}_j^\mathrm{T} \boldsymbol{I}_j^A \boldsymbol{S}_j)^{-1} \, \boldsymbol{S}_j^\mathrm{T} \boldsymbol{I}_j^A \qquad (7.23)$$

and

$$\boldsymbol{p}_j^a = \boldsymbol{p}_j^A + \boldsymbol{I}_j^a \boldsymbol{c}_j + \boldsymbol{I}_j^A \boldsymbol{S}_j \, (\boldsymbol{S}_j^\mathrm{T} \boldsymbol{I}_j^A \boldsymbol{S}_j)^{-1} \, (\boldsymbol{\tau}_j - \boldsymbol{S}_j^\mathrm{T} \boldsymbol{p}_j^A) \,. \qquad (7.24)$$

Essentially, these equations are obtained by applying Eqs. 7.19 and 7.20 once for each child of body i. On each application, A_1 consists of body i plus every child subtree that has already been processed, and A_2 is the next child subtree to be added.

The quantities $\boldsymbol{I}_j^a$ and $\boldsymbol{p}_j^a$ are the result of propagating $\boldsymbol{I}_j^A$ and $\boldsymbol{p}_j^A$ across joint j. They can be regarded as the apparent inertia and bias force of a massless handle that is connected to body j through joint j. In this case, the formulae in Eqs. 7.21 and 7.22 have the effect of gluing all of these massless handles to body i (see Figure 7.2). $\boldsymbol{I}_j^a$ and $\boldsymbol{p}_j^a$ have one more interesting property: they express the force across a joint as a function of the parent body's acceleration, whereas $\boldsymbol{I}_j^A$ and $\boldsymbol{p}_j^A$ express this same force as a function of the child's acceleration:

$$\begin{aligned} \boldsymbol{f}_j &= \boldsymbol{I}_j^A \boldsymbol{a}_j + \boldsymbol{p}_j^A \\ &= \boldsymbol{I}_j^a \boldsymbol{a}_{\lambda(j)} + \boldsymbol{p}_j^a \,. \end{aligned} \qquad (7.25)$$
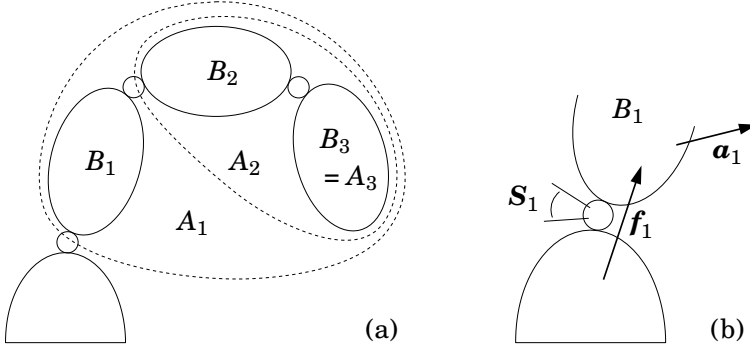
Figure 7.5: Aspects of the articulated-body algorithm: (a) defining the articulated bodies, and (b) solving for the acceleration of joint 1

I_j^a plays two roles in improving the efficiency of propagation calculations. First, it allows the two terms involving c in Eq. 7.20 to be collected into a single term, as shown in Eq. 7.24. Second, it has the property $I_j^a S_j = 0$. Depending on the value of S_j, this property implies that one or more rows and columns of I_j^a will be zero. For example, if $S_j = [0\ 0\ 1\ 0\ 0\ 0]^T$ (i.e., joint j is revolute), then row and column 3 of I_j^a will be zero. This fact can be exploited to achieve a modest reduction in the cost of calculating articulated-body inertias.

7.3 The Articulated-Body Algorithm

Suppose we have a kinematic tree containing N_B rigid bodies. We can define a set of articulated bodies, $A_1 \cdots A_{N_B}$, such that A_i contains all of the bodies and joints in the subtree rooted at body i. A simple example is shown in Figure 7.5(a). Observe that body i (B_i) is in A_i but joint i is not. We define B_i to be the handle of A_i; and define I_i^A and p_i^A to be the articulated-body inertia and bias force for B_i in A_i.

Consider the motion of B_1, as shown in Figure 7.5(b). If f_1 is the force transmitted across joint 1, then the equation of motion for B_1 is

$$f_1 = I_1^A a_1 + p_1^A. \tag{7.26}$$

Now, joint 1 constrains the acceleration of B_1 to satisfy

$$a_1 = a_0 + c_1 + S_1 \ddot{q}_1, \tag{7.27}$$

where a_0 is the known acceleration of the base, and $c_1 = \dot{S}_1 \dot{q}_1$. The joint also constrains the transmitted force to satisfy

$$S_1^T f_1 = \tau_1. \tag{7.28}$$

From these three equations we get

$$\boldsymbol{S}_1^{\mathrm{T}}(\boldsymbol{I}_1^A(\boldsymbol{a}_0 + \boldsymbol{c}_1 + \boldsymbol{S}_1\,\ddot{\boldsymbol{q}}_1) + \boldsymbol{p}_1^A) = \boldsymbol{\tau}_1\,,$$

which can be solved for $\ddot{\boldsymbol{q}}_1$, giving

$$\ddot{\boldsymbol{q}}_1 = (\boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{I}_1^A\boldsymbol{S}_1)^{-1}(\boldsymbol{\tau}_1 - \boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{I}_1^A\,(\boldsymbol{a}_0 + \boldsymbol{c}_1) - \boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{p}_1^A))\,. \qquad (7.29)$$

Consider what we have just achieved. Simply knowing $\boldsymbol{I}_1^A$ and $\boldsymbol{p}_1^A$ has allowed us to calculate $\ddot{\boldsymbol{q}}_1$ directly, without needing to know the values of any of the other acceleration variables. This is possible because Eq. 7.26 already accounts for the dynamic effect of every body in A_1 on the acceleration response of B_1. Having calculated $\ddot{\boldsymbol{q}}_1$, it can be substituted into Eq. 7.27 to give the spatial acceleration of B_1. This, in turn, allows us to repeat the process with the children of B_1, and so on. In the general case, if the acceleration of body $\lambda(i)$ is known, then the acceleration of body i and joint i can be calculated as follows:

$$\ddot{\boldsymbol{q}}_i = (\boldsymbol{S}_i^{\mathrm{T}}\boldsymbol{I}_i^A\boldsymbol{S}_i)^{-1}(\boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}}\boldsymbol{I}_i^A\,(\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i) - \boldsymbol{S}_i^{\mathrm{T}}\boldsymbol{p}_i^A)) \qquad (7.30)$$

and

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i\,. \qquad (7.31)$$

Therefore, if the articulated-body inertias and bias forces are known, then the forward dynamics of the kinematic tree can be calculated using these two equations, with i ranging from 1 to N_B.

Algorithm

The articulated-body algorithm calculates the forward dynamics of a kinematic tree in $O(N_B)$ arithmetic operations, in exactly the manner outlined above. It makes three passes over the tree, as follows: an outward pass (root to leaves) to calculate velocity and bias terms; an inward pass to calculate articulated-body inertias and bias forces; and a second outward pass to calculate the accelerations.

Pass 1   The job of the first pass is to calculate the velocity-product accelerations, $\boldsymbol{c}_i$, and the rigid-body bias forces, $\boldsymbol{p}_i$, for use in later passes. Both are functions of the body velocities, $\boldsymbol{v}_i$, so it is necessary to calculate these as well. The equations of the first pass are:

$$\begin{aligned}
\boldsymbol{v}_{\mathrm{J}i} &= \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i\,, & (7.32) \\
\boldsymbol{c}_{\mathrm{J}i} &= \mathring{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i\,, & (7.33) \\
\boldsymbol{v}_i &= \boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i}\,, & (\boldsymbol{v}_0 = \boldsymbol{0}) & (7.34) \\
\boldsymbol{c}_i &= \boldsymbol{c}_{\mathrm{J}i} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}i}\,, & (7.35) \\
\boldsymbol{p}_i &= \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i - \boldsymbol{f}_i^x\,. & (7.36)
\end{aligned}$$

They are similar to the equations of the first pass in the recursive Newton-Euler algorithm, as shown in Table 5.1. The vectors $\boldsymbol{v}_{\mathrm{J}i}$ and $\boldsymbol{c}_{\mathrm{J}i}$ will have values of $\boldsymbol{S}_i \dot{\boldsymbol{q}}_i + \boldsymbol{\sigma}_i$ and $\mathring{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i + \mathring{\boldsymbol{\sigma}}_i$, respectively, if joint i depends explicitly on time (see §3.5); but the calculation details are handled by jcalc (§4.4). $\boldsymbol{f}_i^x$ is the external force, if any, acting on body i.

Pass 2 The next step is to calculate the articulated-body inertias and bias forces, $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$. This is done using the assembly formulae in Eqs. 7.21 to 7.24, which are reproduced below.

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \sum_{j \in \mu(i)} \boldsymbol{I}_j^a \tag{7.37}$$

$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \sum_{j \in \mu(i)} \boldsymbol{p}_j^a, \tag{7.38}$$

$$\boldsymbol{I}_j^a = \boldsymbol{I}_j^A - \boldsymbol{I}_j^A \boldsymbol{S}_j (\boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{S}_j)^{-1} \boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{I}_j^A \tag{7.39}$$

$$\boldsymbol{p}_j^a = \boldsymbol{p}_j^A + \boldsymbol{I}_j^a \boldsymbol{c}_j + \boldsymbol{I}_j^A \boldsymbol{S}_j (\boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{S}_j)^{-1} (\boldsymbol{\tau}_j - \boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{p}_j^A). \tag{7.40}$$

These calculations are performed for each value of i from N_B down to 1. This implies that $\boldsymbol{I}_j^a$ and $\boldsymbol{p}_j^a$ are not calculated for every j, but only for those values that satisfy $\lambda(j) \neq 0$. If body i has no children then $\boldsymbol{I}_i^A = \boldsymbol{I}_i$ and $\boldsymbol{p}_i^A = \boldsymbol{p}_i$.

Pass 3 The third pass calculates the accelerations using Eqs. 7.30 and 7.31, which we repeat here as

$$\ddot{\boldsymbol{q}}_i = (\boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^A \boldsymbol{S}_i)^{-1} (\boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^A (\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i) - \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{p}_i^A)) \tag{7.41}$$

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i. \qquad (\boldsymbol{a}_0 = -\boldsymbol{a}_g) \tag{7.42}$$

By initializing the base acceleration to $-\boldsymbol{a}_g$, rather than $\boldsymbol{0}$, we can simulate the effect of a uniform gravitational field with a fictitious upward acceleration. This is more efficient than treating gravity as an external force.

Common Subexpressions

Equations 7.39 to 7.42 contain a number of common subexpressions, which can be eliminated by defining the following intermediate results:

$$\boldsymbol{U}_i = \boldsymbol{I}_i^A \boldsymbol{S}_i, \tag{7.43}$$

$$\boldsymbol{D}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{U}_i, \tag{7.44}$$

$$\boldsymbol{u}_i = \boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{p}_i^A, \tag{7.45}$$

$$\boldsymbol{a}_i' = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i. \tag{7.46}$$

In terms of these quantities, Eqs. 7.39 to 7.42 simplify to

$$\boldsymbol{I}_j^a = \boldsymbol{I}_j^A - \boldsymbol{U}_j \boldsymbol{D}_j^{-1} \boldsymbol{U}_j^{\mathrm{T}}, \tag{7.47}$$

$$\boldsymbol{p}_j^a = \boldsymbol{p}_j^A + \boldsymbol{I}_j^a \boldsymbol{c}_j + \boldsymbol{U}_j \boldsymbol{D}_j^{-1} \boldsymbol{u}_j, \tag{7.48}$$

$$\ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i^{-1} (\boldsymbol{u}_i - \boldsymbol{U}_i^{\mathrm{T}} \boldsymbol{a}_i'), \tag{7.49}$$

$$\boldsymbol{a}_i = \boldsymbol{a}_i' + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i. \qquad (\boldsymbol{a}_0 = -\boldsymbol{a}_g) \tag{7.50}$$

Equations in Body Coordinates

The articulated-body algorithm works best in body coordinates. To convert the above equations to body coordinates, simply define each quantity that pertains to body i as being expressed in body i coordinates, and insert coordinate transforms where they are needed. The resulting equations are shown in Table 7.1.

Algorithm Details

The pseudocode for the articulated-body algorithm is shown in Table 7.1. It is organized into three passes, with one **for** loop for each pass. The quantities $\boldsymbol{X}_\mathrm{J}$, $\boldsymbol{v}_\mathrm{J}$ and $\boldsymbol{c}_\mathrm{J}$ are local variables in the first pass; $\boldsymbol{I}^a$ and $\boldsymbol{p}^a$ are local variables in the second pass; and $\boldsymbol{a}'$ is a local variable in the third pass. If the tree contains joints that depend explicitly on time, then add time as a fourth argument to jcalc. If the tree contains joints for which the velocity variables are not the derivatives of the position variables, then replace the symbols $\dot{\boldsymbol{q}}_i$ and $\ddot{\boldsymbol{q}}_i$ with $\boldsymbol{\alpha}_i$ and $\dot{\boldsymbol{\alpha}}_i$. If there are no external forces, then the calculation of ${}^i\boldsymbol{X}_0$ is not necessary. The pseudocode assumes that as soon as a transform $\boldsymbol{X}$ has been calculated, the transforms $\boldsymbol{X}^*$, $\boldsymbol{X}^{-1}$ and $(\boldsymbol{X}^*)^{-1}$ are immediately available for use. The functions that make this possible are described in Appendix A.

Equations 7.37 and 7.38 have been transformed from calculations using $\mu(i)$ to calculations using $\lambda(i)$. As a result, $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ are initialized to $\boldsymbol{I}_i$ and $\boldsymbol{p}_i$ at the bottom of the first **for** loop, and the remainder of the calculation is performed inside the **if** statement in the second **for** loop (cf. the implementation of Eq. 5.20 in §5.3).

7.4 Alternative Assembly Formulae

Section 7.3 describes the standard version of the articulated-body algorithm, which is based on the assembly formulae in Section 7.2.2. However, there are other formulae that could be used instead. This section presents a systematic procedure for deriving alternative assembly formulae, which give rise to alternative versions of the articulated-body algorithm. In most cases, the alternatives are more complicated and less efficient than the standard algorithm. However, they can solve a larger class of problems.

In Section 7.2.2 we solved the following problem: given the five equations

$$\boldsymbol{f}_1 = \boldsymbol{I}_1^A \, \boldsymbol{a}_1 + \boldsymbol{p}_1^A \,, \tag{7.51}$$

$$\boldsymbol{f}_2 = \boldsymbol{I}_2^A \, \boldsymbol{a}_2 + \boldsymbol{p}_2^A \,, \tag{7.52}$$

$$\boldsymbol{f} = \boldsymbol{f}_1 + \boldsymbol{f}_2 \,, \tag{7.53}$$

$$\boldsymbol{a}_2 = \boldsymbol{a}_1 + \boldsymbol{c} + \boldsymbol{S} \, \ddot{\boldsymbol{q}} \tag{7.54}$$

and

$$\boldsymbol{S}^\mathrm{T} \boldsymbol{f}_2 = \boldsymbol{\tau} \,, \tag{7.55}$$

Equations (in body coordinates): | Algorithm:

Pass 1

$$v_0 = \mathbf{0}$$

$$v_{\mathrm{J}i} = S_i\,\dot{q}_i$$

$$c_{\mathrm{J}i} = \mathring{S}_i\,\dot{q}_i$$

$$v_i = {}^iX_{\lambda(i)}\,v_{\lambda(i)} + v_{\mathrm{J}i}$$

$$c_i = c_{\mathrm{J}i} + v_i \times v_{\mathrm{J}i}$$

$$p_i = v_i \times^* I_i\,v_i - {}^iX_0^*\,f_i^x$$

Pass 2

$$I_i^A = I_i + \sum_{j \in \mu(i)} {}^iX_j^*\,I_j^a\,{}^jX_i$$

$$p_i^A = p_i + \sum_{j \in \mu(i)} {}^iX_j^*\,p_j^a$$

$$U_i = I_i^A\,S_i$$

$$D_i = S_i^{\mathrm{T}}\,U_i$$

$$u_i = \tau_i - S_i^{\mathrm{T}}\,p_i^A$$

$$I_i^a = I_i^A - U_i\,D_i^{-1}\,U_i^{\mathrm{T}}$$

$$p_i^a = p_i^A + I_i^a c_i + U_i\,D_i^{-1} u_i$$

Pass 3

$$a_0 = -a_g$$

$$a_i' = {}^iX_{\lambda(i)}\,a_{\lambda(i)} + c_i$$

$$\ddot{q}_i = D_i^{-1}(u_i - U_i^{\mathrm{T}} a_i')$$

$$a_i = a_i' + S_i\,\ddot{q}_i$$

Algorithm:

$v_0 = \mathbf{0}$
for $i = 1$ **to** N_B **do**
  $[X_{\mathrm{J}}, S_i, v_{\mathrm{J}}, c_{\mathrm{J}}] = \mathrm{jcalc}(\mathrm{jtype}(i), q_i, \dot{q}_i)$
  ${}^iX_{\lambda(i)} = X_{\mathrm{J}}\,X_{\mathrm{T}}(i)$
  if $\lambda(i) \neq 0$ **then**
    ${}^iX_0 = {}^iX_{\lambda(i)}\,{}^{\lambda(i)}X_0$
  end
  $v_i = {}^iX_{\lambda(i)}\,v_{\lambda(i)} + v_{\mathrm{J}}$
  $c_i = c_{\mathrm{J}} + v_i \times v_{\mathrm{J}}$
  $I_i^A = I_i$
  $p_i^A = v_i \times^* I_i\,v_i - {}^iX_0^*\,f_i^x$
end
for $i = N_B$ **to** 1 **do**
  $U_i = I_i^A\,S_i$
  $D_i = S_i^{\mathrm{T}}\,U_i$
  $u_i = \tau_i - S_i^{\mathrm{T}}\,p_i^A$
  if $\lambda(i) \neq 0$ **then**
    $I^a = I_i^A - U_i\,D_i^{-1}\,U_i^{\mathrm{T}}$
    $p^a = p_i^A + I^a c_i + U_i\,D_i^{-1} u_i$
    $I_{\lambda(i)}^A = I_{\lambda(i)}^A + {}^{\lambda(i)}X_i^*\,I^a\,{}^iX_{\lambda(i)}$
    $p_{\lambda(i)}^A = p_{\lambda(i)}^A + {}^{\lambda(i)}X_i^*\,p^a$
  end
end
$a_0 = -a_g$
for $i = 1$ **to** N_B **do**
  $a' = {}^iX_{\lambda(i)}\,a_{\lambda(i)} + c_i$
  $\ddot{q}_i = D_i^{-1}(u_i - U_i^{\mathrm{T}} a')$
  $a_i = a' + S_i\,\ddot{q}_i$
end

Table 7.1: The articulated body equations and algorithm

find expressions for the coefficients I^A and p^A in the target equation

$$f = I^A a_1 + p^A . \qquad (7.56)$$

By following a few simple rules, the five given equations can be assembled into the following composite equation:

$$
\begin{bmatrix}
1 & -1 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -I_2^A & 0 \\
0 & 0 & 0 & 1 & -S \\
0 & 0 & S^T & 0 & 0
\end{bmatrix}
\begin{bmatrix}
f \\
f_1 \\
f_2 \\
a_2 \\
\ddot{q}
\end{bmatrix}
=
\begin{bmatrix}
0 \\
p_1^A \\
p_2^A \\
c \\
\tau
\end{bmatrix}
+
\begin{bmatrix}
0 \\
I_1^A \\
0 \\
1 \\
0
\end{bmatrix}
a_1 .
$$

The rules for constructing this equation are:

1. the right-hand unknown in the target equation (a_1) becomes right-hand unknown in the composite equation;

2. the left-hand unknown in the target equation (f) becomes the top entry in the composite vector of unknowns;

3. the unknown vector with a dimension other than six ($\ddot{q}$) becomes the bottom entry in the composite vector of unknowns;

4. the given equation with a dimension other than six ($S^T f_2 = \tau$) occupies the bottom row in the composite equation; and

5. the remaining equations and unknowns are inserted in any order that results in the 5×5 composite matrix being as close as possible to upper-triangular form.

Having constructed this equation, the original problem can be solved using a process of Gaussian elimination and back-substitution. Once the coefficient matrix has been brought into upper-triangular form, we immediately have an equation expressing $\ddot{q}$ as a function of a_1; and the process of back-substitution eventually yields an equation giving f as a function of a_1. At this point, the coefficient of a_1 is the desired expression for I^A, and the remaining terms provide the desired expression for p^A.

The process is illustrated in Table 7.2 under the heading 'Standard Formulae'. At the end of the Gaussian elimination stage, the fifth row reads

$$S^T I_2^A S \ddot{q} = \tau - S^T p_2^A - S^T I_2^A c - S^T I_2^A a_1 ,$$

which implies

$$\ddot{q} = (S^T I_2^A S)^{-1}(\tau - S^T p_2^A - S^T I_2^A c - S^T I_2^A a_1)$$

(cf. Eq. 7.18). Having found $\ddot{q}$ in terms of a_1, back-substitution allows us to express a_2 in terms of a_1, and so on, until eventually we get an expression for f in terms of a_1, from which the formulae for I^A and p^A can be extracted.

<u>Standard Formulae</u>   obtain $f = I^A a_1 + p^A$ using I_i^A, p_i^A and S

$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -I_2^A & 0 \\ 0 & 0 & 0 & 1 & -S \\ 0 & 0 & S^{\mathrm{T}} & 0 & 0 \end{bmatrix} \begin{bmatrix} f \\ f_1 \\ f_2 \\ a_2 \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 \\ p_1^A \\ p_2^A \\ c \\ \tau \end{bmatrix} + \begin{bmatrix} 0 \\ I_1^A \\ 0 \\ 1 \\ 0 \end{bmatrix} a_1$$

$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -I_2^A & 0 \\ 0 & 0 & 0 & 1 & -S \\ 0 & 0 & 0 & S^{\mathrm{T}} I_2^A & 0 \end{bmatrix} \begin{bmatrix} f \\ f_1 \\ f_2 \\ a_2 \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 \\ p_1^A \\ p_2^A \\ c \\ \tau - S^{\mathrm{T}} p_2^A \end{bmatrix} + \begin{bmatrix} 0 \\ I_1^A \\ 0 \\ 1 \\ 0 \end{bmatrix} a_1$$

$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -I_2^A & 0 \\ 0 & 0 & 0 & 1 & -S \\ 0 & 0 & 0 & 0 & S^{\mathrm{T}} I_2^A S \end{bmatrix} \begin{bmatrix} f \\ f_1 \\ f_2 \\ a_2 \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 \\ p_1^A \\ p_2^A \\ c \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ I_1^A \\ 0 \\ 1 \\ -S^{\mathrm{T}} I_2^A \end{bmatrix} a_1$$

where $z = \tau - S^{\mathrm{T}} p_2^A - S^{\mathrm{T}} I_2^A c$

result:

$\ddot{q} = (S^{\mathrm{T}} I_2^A S)^{-1} (z - S^{\mathrm{T}} I_2^A a_1)$

$I^A = I_1^A + I_2^A - I_2^A S (S^{\mathrm{T}} I_2^A S)^{-1} S^{\mathrm{T}} I_2^A$

$p^A = p_1^A + p_2^A + I_2^A c + I_2^A S (S^{\mathrm{T}} I_2^A S)^{-1} z$

<u>Variant Formulae 1</u>   obtain $a_1 = \Phi^A f + b^A$ using Φ_i^A, b_i^A and T

$$\begin{bmatrix} 1 & 0 & -\Phi_1^A & 0 & 0 \\ 0 & 1 & 0 & -\Phi_2^A & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -T \\ -T^{\mathrm{T}} & T^{\mathrm{T}} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ f_1 \\ f_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} b_1^A \\ b_2^A \\ 0 \\ T_a \tau \\ T^{\mathrm{T}} c \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} f$$

$$\begin{bmatrix} 1 & 0 & -\Phi_1^A & 0 & 0 \\ 0 & 1 & 0 & -\Phi_2^A & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -T \\ 0 & 0 & -T^{\mathrm{T}} \Phi_1^A & T^{\mathrm{T}} \Phi_2^A & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ f_1 \\ f_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} b_1^A \\ b_2^A \\ 0 \\ T_a \tau \\ T^{\mathrm{T}} (c + b_1^A - b_2^A) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} f$$

(continued on next page)

Table 7.2: Alternative assembly formulae

<div align="center">(continued from previous page)</div>

$$
\begin{bmatrix}
1 & 0 & -\boldsymbol{\Phi}_1^A & 0 & 0 \\
0 & 1 & 0 & -\boldsymbol{\Phi}_2^A & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & -\boldsymbol{T} \\
0 & 0 & 0 & \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_1^A + \boldsymbol{\Phi}_2^A) & 0
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{\lambda}
\end{bmatrix}
=
\begin{bmatrix}
\boldsymbol{b}_1^A \\
\boldsymbol{b}_2^A \\
0 \\
\boldsymbol{T}_a \boldsymbol{\tau} \\
\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{c} + \boldsymbol{b}_1^A - \boldsymbol{b}_2^A)
\end{bmatrix}
+
\begin{bmatrix}
0 \\ 0 \\ 1 \\ 0 \\ \boldsymbol{T}^{\mathrm{T}}\boldsymbol{\Phi}_1^A
\end{bmatrix}
\boldsymbol{f}
$$

$$
\begin{bmatrix}
1 & 0 & -\boldsymbol{\Phi}_1^A & 0 & 0 \\
0 & 1 & 0 & -\boldsymbol{\Phi}_2^A & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & -\boldsymbol{T} \\
0 & 0 & 0 & 0 & \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_1^A + \boldsymbol{\Phi}_2^A)\boldsymbol{T}
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{\lambda}
\end{bmatrix}
=
\begin{bmatrix}
\boldsymbol{b}_1^A \\
\boldsymbol{b}_2^A \\
0 \\
\boldsymbol{T}_a \boldsymbol{\tau} \\
\boldsymbol{z}
\end{bmatrix}
+
\begin{bmatrix}
0 \\ 0 \\ 1 \\ 0 \\ \boldsymbol{T}^{\mathrm{T}}\boldsymbol{\Phi}_1^A
\end{bmatrix}
\boldsymbol{f}
$$

where $\boldsymbol{z} = \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{c} + \boldsymbol{b}_1^A - \boldsymbol{b}_2^A - (\boldsymbol{\Phi}_1^A + \boldsymbol{\Phi}_2^A)\boldsymbol{T}_a\boldsymbol{\tau})$

result:

$$\boldsymbol{\lambda} = (\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_1^A + \boldsymbol{\Phi}_2^A)\boldsymbol{T})^{-1}(\boldsymbol{z} + \boldsymbol{T}^{\mathrm{T}}\boldsymbol{\Phi}_1^A \boldsymbol{f})$$

$$\boldsymbol{\Phi}^A = \boldsymbol{\Phi}_1^A - \boldsymbol{\Phi}_1^A \boldsymbol{T}(\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_1^A + \boldsymbol{\Phi}_2^A)\boldsymbol{T})^{-1}\boldsymbol{T}^{\mathrm{T}}\boldsymbol{\Phi}_1^A$$

$$\boldsymbol{b}^A = \boldsymbol{b}_1^A - \boldsymbol{\Phi}_1^A \boldsymbol{T}_a\boldsymbol{\tau} - \boldsymbol{\Phi}_1^A \boldsymbol{T}(\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_1^A + \boldsymbol{\Phi}_2^A)\boldsymbol{T})^{-1}\boldsymbol{z}$$

Variant Formulae 2   obtain $\boldsymbol{f} = \boldsymbol{I}^A \boldsymbol{a}_1 + \boldsymbol{p}^A$ using $\boldsymbol{I}_i^A$, $\boldsymbol{p}_i^A$ and $\boldsymbol{T}$

$$
\begin{bmatrix}
1 & -1 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -\boldsymbol{I}_2^A & 0 \\
0 & 0 & 1 & 0 & -\boldsymbol{T} \\
0 & 0 & 0 & \boldsymbol{T}^{\mathrm{T}} & 0
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{f} \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{a}_2 \\ \boldsymbol{\lambda}
\end{bmatrix}
=
\begin{bmatrix}
0 \\ \boldsymbol{p}_1^A \\ \boldsymbol{p}_2^A \\ \boldsymbol{T}_a\boldsymbol{\tau} \\ \boldsymbol{T}^{\mathrm{T}}\boldsymbol{c}
\end{bmatrix}
+
\begin{bmatrix}
0 \\ \boldsymbol{I}_1^A \\ 0 \\ 0 \\ \boldsymbol{T}^{\mathrm{T}}
\end{bmatrix}
\boldsymbol{a}_1
$$

$$
\begin{bmatrix}
1 & -1 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -\boldsymbol{I}_2^A & 0 \\
0 & 0 & 0 & \boldsymbol{I}_2^A & -\boldsymbol{T} \\
0 & 0 & 0 & \boldsymbol{T}^{\mathrm{T}} & 0
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{f} \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{a}_2 \\ \boldsymbol{\lambda}
\end{bmatrix}
=
\begin{bmatrix}
0 \\ \boldsymbol{p}_1^A \\ \boldsymbol{p}_2^A \\ \boldsymbol{T}_a\boldsymbol{\tau} - \boldsymbol{p}_2^A \\ \boldsymbol{T}^{\mathrm{T}}\boldsymbol{c}
\end{bmatrix}
+
\begin{bmatrix}
0 \\ \boldsymbol{I}_1^A \\ 0 \\ 0 \\ \boldsymbol{T}^{\mathrm{T}}
\end{bmatrix}
\boldsymbol{a}_1
$$

$$
\begin{bmatrix}
1 & -1 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -\boldsymbol{I}_2^A & 0 \\
0 & 0 & 0 & \boldsymbol{I}_2^A & -\boldsymbol{T} \\
0 & 0 & 0 & 0 & \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{I}_2^A)^{-1}\boldsymbol{T}
\end{bmatrix}
\begin{bmatrix}
\boldsymbol{f} \\ \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{a}_2 \\ \boldsymbol{\lambda}
\end{bmatrix}
=
\begin{bmatrix}
0 \\ \boldsymbol{p}_1^A \\ \boldsymbol{p}_2^A \\ \boldsymbol{T}_a\boldsymbol{\tau} - \boldsymbol{p}_2^A \\ \boldsymbol{z}
\end{bmatrix}
+
\begin{bmatrix}
0 \\ \boldsymbol{I}_1^A \\ 0 \\ 0 \\ \boldsymbol{T}^{\mathrm{T}}
\end{bmatrix}
\boldsymbol{a}_1
$$

where $\boldsymbol{z} = \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{c} - (\boldsymbol{I}_2^A)^{-1}(\boldsymbol{T}_a\boldsymbol{\tau} - \boldsymbol{p}_2^A))$

result:

$$\boldsymbol{\lambda} = (\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{I}_2^A)^{-1}\boldsymbol{T})^{-1}(\boldsymbol{z} + \boldsymbol{T}^{\mathrm{T}}\boldsymbol{a}_1)$$

$$\boldsymbol{I}^A = \boldsymbol{I}_1^A + \boldsymbol{T}(\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{I}_2^A)^{-1}\boldsymbol{T})^{-1}\boldsymbol{T}^{\mathrm{T}}$$

$$\boldsymbol{p}^A = \boldsymbol{p}_1^A + \boldsymbol{T}_a\boldsymbol{\tau} + \boldsymbol{T}(\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{I}_2^A)^{-1}\boldsymbol{T})^{-1}\boldsymbol{z}$$

<div align="center">Table 7.2: Alternative assembly formulae</div>

The point of this exercise is that it makes the solution process systematic, so that it can be applied to a different set of given equations. There are two obvious changes we can try:

1. replace Eqs. 7.51, 7.52 and 7.56 with their inverses; that is, $a_1 = \Phi_1^A f_1 + b_1^A$, $a_2 = \Phi_2^A f_2 + b_2^A$ and $a_1 = \Phi^A f + b^A$; and

2. replace Eqs. 7.54 and 7.55 with the equations of the equivalent implicit joint constraint, which are $T^{\mathrm{T}} a_2 = T^{\mathrm{T}}(a_1 + c)$ and $f_2 = T_a \tau + T\lambda$.

Table 7.2 shows two of the alternative formulae that can be produced by this method: variant 1 is the result of making both changes; and variant 2 is the result of making only the second change. Each can be used as the basis for an alternative formulation of the articulated-body algorithm.[1]

Observe that this process gives rise to some potentially useful matrix identities. For example, the variant formula for I^A must be equal to the standard formula, and both must equal the inverse of the formula for Φ^A. Similar comments apply to p^A and b^A.

Any formula that uses inertias, rather than inverse inertias, is restricted by the requirement that the inertias must exist. Thus, the standard formulae, and those of variant 2, are only applicable if both handles have a full six degrees of freedom. In contrast, the formulae of variant 1 require only that the matrix $T^{\mathrm{T}}(\Phi_1^A + \Phi_2^A)T$ have full rank. Physically, this is equivalent to requiring that the joint does not impose a redundant constraint; that is, it does not duplicate a constraint that was already there. A sufficient condition for the application of variant 1 is that at least one of the two handles have a full six degrees of freedom. Thus, an algorithm based on variant 1 would be able to calculate articulated-body inverse inertias for bodies in any kinematic tree, and a few closed-loop systems, whereas the standard algorithm is limited to floating kinematic trees.

7.5 Multiple Handles

So far, we have applied the assembly method only to articulated bodies having one handle each. This is sufficient for assembling kinematic trees. However, if we wish to assemble general rigid-body systems, or if we want greater flexibility in the order of assembly, then we must be able to assemble articulated bodies having more than one handle. As an example of what can be accomplished, Featherstone (1999a,b) describes a divide-and-conquer algorithm to assemble both kinematic trees and closed-loop systems in a manner that allows their dynamics to be calculated in $O(\log(N_B))$ time on a parallel computer with $O(N_B)$ processors. Therefore, let us now develop an assembly formula for articulated bodies with multiple handles.

---

[1]Neither variant calculates $\ddot{q}$ as a by-product; but this quantity can be calculated from a_1 and a_2 using the formula $\ddot{q} = T_a^{\mathrm{T}}(a_2 - a_1 - c)$. An algorithm based on variant 1 will be more efficient if the active joint force, $T_a \tau$, is replaced by an equivalent pair of external forces which contribute to the bias accelerations of the individual rigid bodies.
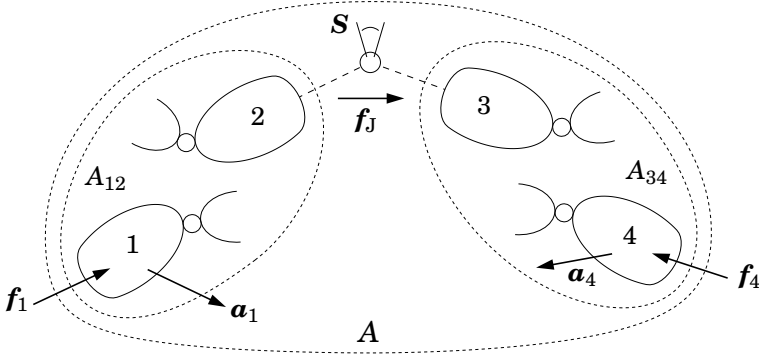
Figure 7.6: Assembly of articulated bodies with multiple handles

Figure 7.6 shows two articulated bodies, called A_{12} and A_{34}, which are to be assembled to form a larger articulated body called A. Body A_{12} contains two handles, numbered 1 and 2, while A_{34} contains handles 3 and 4. The idea is to assemble these bodies by connecting a joint between handles 2 and 3. The assembly operation is deemed to consume these two handles, leaving articulated body A with only handles 1 and 4. We shall assume that at least one of handles 2 and 3 has six degrees of freedom. The equations of motion for A_{12} and A_{34} before the assembly operation are

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_1 & \boldsymbol{\Phi}_{12} \\ \boldsymbol{\Phi}_{21} & \boldsymbol{\Phi}_2 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{7.57}$$

and

$$\begin{bmatrix} a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_3 & \boldsymbol{\Phi}_{34} \\ \boldsymbol{\Phi}_{43} & \boldsymbol{\Phi}_4 \end{bmatrix} \begin{bmatrix} f_3 \\ f_4 \end{bmatrix} + \begin{bmatrix} b_3 \\ b_4 \end{bmatrix} ; \tag{7.58}$$

and the equation of motion for A after the assembly operation is

$$\begin{bmatrix} a_1 \\ a_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi}_1^A & \boldsymbol{\Phi}_{14}^A \\ \boldsymbol{\Phi}_{41}^A & \boldsymbol{\Phi}_4^A \end{bmatrix} \begin{bmatrix} f_1 \\ f_4 \end{bmatrix} + \begin{bmatrix} b_1^A \\ b_4^A \end{bmatrix} . \tag{7.59}$$

(Observe that we have simplified the notation by omitting the superscripts from the coefficients of Eqs. 7.57 and 7.58.) We are using inverse inertias here because it is possible—and indeed quite likely—that the two handles within an articulated body do not have a full 12 degrees of motion freedom between them, even if they do have 6 degrees of freedom individually. In the absence of full motion freedom, the coefficient matrices in these equations will be singular. However, our assumption that at least one of the two handles 2 and 3 has six degrees of freedom on its own implies that at least one of the two matrices $\boldsymbol{\Phi}_2$ and $\boldsymbol{\Phi}_3$ is positive definite. This, in turn, implies that the sum $\boldsymbol{\Phi}_2 + \boldsymbol{\Phi}_3$ is positive definite.

When the assembly is made, the joint will impose the following motion constraint on handles 2 and 3:

$$\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{a}_3 - \boldsymbol{a}_2) = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{c}\,, \tag{7.60}$$

where $\boldsymbol{c} = \dot{\boldsymbol{S}}\dot{\boldsymbol{q}}$, and it will also impose the force constraint

$$\boldsymbol{f}_3 = \boldsymbol{T}\boldsymbol{\lambda} = -\boldsymbol{f}_2\,. \tag{7.61}$$

In formulating Eq. 7.61, we have assumed that any active force at the joint has already been accounted for in the bias accelerations of Eqs. 7.57 and 7.58. In other words, the active joint force, $\boldsymbol{T}_a\boldsymbol{\tau}$, has been replaced by a pair of equal and opposite external forces, $\boldsymbol{T}_a\boldsymbol{\tau}$ acting on handle 3 and $-\boldsymbol{T}_a\boldsymbol{\tau}$ acting on handle 2, and that the effects of these two forces on the motion of A_{12} and A_{34} have already been included in the vectors $\boldsymbol{b}_1 \cdots \boldsymbol{b}_4$.

The objective is to express the coefficients of Eq. 7.59 in terms of the coefficients of Eqs. 7.57, 7.58, 7.60 and 7.61. The first step in the solution process is to substitute for $\boldsymbol{a}_3$ and $\boldsymbol{a}_2$ in Eq. 7.60, giving

$$\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_3\boldsymbol{f}_3 + \boldsymbol{\Phi}_{34}\boldsymbol{f}_4 + \boldsymbol{b}_3 - \boldsymbol{\Phi}_{21}\boldsymbol{f}_1 - \boldsymbol{\Phi}_2\boldsymbol{f}_2 - \boldsymbol{b}_2) = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{c}\,.$$

The next step is to substitute for $\boldsymbol{f}_3$ and $\boldsymbol{f}_2$ in this equation, using Eq. 7.61, and collect terms in $\boldsymbol{\lambda}$, which gives

$$\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_2 + \boldsymbol{\Phi}_3)\boldsymbol{T}\boldsymbol{\lambda} = \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_{21}\boldsymbol{f}_1 - \boldsymbol{\Phi}_{34}\boldsymbol{f}_4 + \boldsymbol{c} - \boldsymbol{b}_3 + \boldsymbol{b}_2)\,.$$

Given that $\boldsymbol{\Phi}_2 + \boldsymbol{\Phi}_3$ is positive definite, the coefficient of $\boldsymbol{\lambda}$ is guaranteed to be nonsingular, and so we may solve for $\boldsymbol{\lambda}$ as follows:

$$\boldsymbol{\lambda} = \boldsymbol{Y}^{-1}\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_{21}\boldsymbol{f}_1 - \boldsymbol{\Phi}_{34}\boldsymbol{f}_4 + \boldsymbol{\beta})\,, \tag{7.62}$$

where

$$\boldsymbol{Y} = \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_2 + \boldsymbol{\Phi}_3)\boldsymbol{T}$$

and

$$\boldsymbol{\beta} = \boldsymbol{c} - \boldsymbol{b}_3 + \boldsymbol{b}_2\,.$$

The final step is to substitute for $\boldsymbol{f}_2$ and $\boldsymbol{f}_3$ in the expressions for $\boldsymbol{a}_1$ and $\boldsymbol{a}_4$ (drawn from Eqs. 7.57 and 7.58), using both Eq. 7.61 and Eq. 7.62. This gives

$$\boldsymbol{a}_1 = \boldsymbol{\Phi}_1\boldsymbol{f}_1 - \boldsymbol{\Phi}_{12}\boldsymbol{T}\boldsymbol{Y}^{-1}\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_{21}\boldsymbol{f}_1 - \boldsymbol{\Phi}_{34}\boldsymbol{f}_4 + \boldsymbol{\beta}) + \boldsymbol{b}_1$$
$$\boldsymbol{a}_4 = \boldsymbol{\Phi}_4\boldsymbol{f}_4 + \boldsymbol{\Phi}_{43}\boldsymbol{T}\boldsymbol{Y}^{-1}\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{\Phi}_{21}\boldsymbol{f}_1 - \boldsymbol{\Phi}_{34}\boldsymbol{f}_4 + \boldsymbol{\beta}) + \boldsymbol{b}_4\,.$$

These two equations express $\boldsymbol{a}_1$ and $\boldsymbol{a}_4$ as functions of $\boldsymbol{f}_1$ and $\boldsymbol{f}_4$ only, and are therefore the component parts of Eq. 7.59. Thus, the formulae for the coefficients of Eq. 7.59 are

$$\begin{aligned} \boldsymbol{\Phi}_1^A &= \boldsymbol{\Phi}_1 - \boldsymbol{\Phi}_{12}\boldsymbol{W}\boldsymbol{\Phi}_{21} & \boldsymbol{b}_1^A &= \boldsymbol{b}_1 - \boldsymbol{\Phi}_{12}\boldsymbol{W}\boldsymbol{\beta} \\ \boldsymbol{\Phi}_4^A &= \boldsymbol{\Phi}_4 - \boldsymbol{\Phi}_{43}\boldsymbol{W}\boldsymbol{\Phi}_{34} & \boldsymbol{b}_4^A &= \boldsymbol{b}_4 + \boldsymbol{\Phi}_{43}\boldsymbol{W}\boldsymbol{\beta} \\ \boldsymbol{\Phi}_{14}^A &= \boldsymbol{\Phi}_{12}\boldsymbol{W}\boldsymbol{\Phi}_{34} = (\boldsymbol{\Phi}_{41}^A)^{\mathrm{T}} \end{aligned} \tag{7.63}$$

where $\boldsymbol{W} = \boldsymbol{T}\boldsymbol{Y}^{-1}\boldsymbol{T}^{\mathrm{T}}$.

Generalizations

To generalize the number of handles, we replace the four individual handles with four handle sets. Set 1 contains every handle in A_{12} that will not be connected to a joint during assembly; set 2 contains every handle that will; and so on. Each handle in set 2 will be connected to one handle in set 3 via one joint; so sets 2 and 3 must contain the same number of handles. To permit multiple joints to be connected to the same body, we allow a body to have more than one handle. In this context, a handle should no longer be regarded as a body, but as a location on a body. Any kinematic tree can be obtained by a sequence of assembly operations involving one joint per assembly (i.e., sets 2 and 3 contain one handle each). However, the construction of closed-loop systems requires multi-joint assemblies.

Having defined these sets, the next step is to define $\boldsymbol{a}_1$, $\boldsymbol{f}_2$, $\boldsymbol{\Phi}_{34}$, and so on, to be composite vectors and matrices containing every relevant vector and matrix for their sets. For example, if set 1 contains handles 1a, 1b, 1c, ..., having accelerations of $\boldsymbol{a}_{1a}$, $\boldsymbol{a}_{1b}$, ..., then $\boldsymbol{a}_1$ is the composite vector $[\boldsymbol{a}_{1a}^{\mathrm{T}}\ \boldsymbol{a}_{1b}^{\mathrm{T}}\ \cdots]^{\mathrm{T}}$. Note that $\boldsymbol{T}$ will be a block-diagonal matrix composed of the constraint-force subspace matrices of the individual joints.

Given these composite vectors and matrices, the derivation of Eq. 7.63 proceeds in the same manner as shown in Eqs. 7.57 to 7.62, except that it may no longer be possible to assume that $\boldsymbol{Y}$ is invertible. Featherstone (1999b) explains what to do if $\boldsymbol{Y}$ is singular. It also explains how to incorporate constraint stabilization terms (see §8.3) into the assembly formula.

Chapter 8

Closed Loop Systems

This chapter covers the forward and inverse dynamics of rigid-body systems containing kinematic loops. The presence of kinematic loops brings a new level of complexity to the dynamics problem: new formulations are required; new problems arise; the systems exhibit new behaviours; and the inverse dynamics problem, in particular, changes its nature in the presence of kinematic loops. There are two main strategies for formulating the equations of motion for a closed-loop system: They are:

1. start with a set of unconstrained bodies, and apply all of the joint constraints simultaneously; or

2. start with a spanning tree of the system, and apply the loop-closure constraints.

A third possibility, which will not be considered here, is to use the assembly method described in Section 7.5. Method 1 results in large, sparse matrix equations, and is considered briefly in Section 8.13. Method 2 is the best choice for typical closed-loop systems, and is the main topic of this chapter. Some of the special properties of closed-loop systems are discussed in Section 8.10; the use of a loop-closure function is covered in Section 8.11; and inverse dynamics is covered in Section 8.12.

8.1   Equations of Motion

The equations of motion for general rigid-body systems, including closed-loop systems, were introduced in Chapter 3. In this section, we derive the equation of motion for a closed-loop system from that of its spanning tree. The procedure can be summarized as follows.

1. Formulate the equation of motion for a spanning tree of the closed-loop system.

2. Add terms to this equation representing the forces exerted on the tree by the loop joints.

3. Formulate a kinematic equation describing the motion constraints imposed on the tree by the loop joints.

4. Combine these two equations.

If the spanning tree has n degrees of freedom, and the loop joints impose n^c constraints on the tree, then this procedure results in a system of $n + n^c$ equations in $n + n^c$ unknowns.

Consider a rigid-body system containing N_B bodies, N_J joints and $N_L = N_J - N_B$ kinematic loops. We assume that a spanning tree has been chosen, and that the bodies and joints have been numbered accordingly. The number of tree-joint variables, n, and the number of loop-closure constraints, n^c, are given by the formulae

$$n = \sum_{i=1}^{N_B} n_i \qquad \text{and} \qquad n^c = \sum_{k=N_B+1}^{N_J} n_k^c \,, \tag{8.1}$$

where n_i is the number of freedoms allowed by tree joint i, and n_k^c is the number of constraints imposed by loop joint k. Observe that we are using the index variables i and k to range over tree-joint numbers and loop-joint numbers, respectively. More generally, we use the following index-variable convention in connection with kinematic loops: i and j range over tree-joint and/or body numbers (1 to N_B); l ranges over loop numbers (1 to N_L); and k identifies the loop joint that closes loop number l. With our standard numbering scheme, k and l are related by $k = l + N_B$, and k therefore ranges from $N_B + 1$ to N_J. From here on, wherever k and l appear together, it should be assumed that $k = l + N_B$.

The equation of motion for the spanning tree can be written

$$H\ddot{q} + C = \tau \,, \tag{8.2}$$

where $\ddot{q}$ and τ are vectors of tree-joint acceleration and force variables, respectively. Now, the only difference between the closed-loop system and its spanning tree is that the former contains the loop joints and the latter does not. Therefore, the equation of motion for the closed-loop system can be obtained from Eq. 8.2 by adding terms that account for the forces exerted on the tree by the loop joints. Thus, the equation of motion for the closed-loop system can be written

$$H\ddot{q} + C = \tau + \tau^c + \tau^a \,, \tag{8.3}$$

where τ^c and τ^a account for the constraint forces and active forces, respectively, produced by the loop joints. Active forces arise from springs, dampers, actuators, and the like, acting at the loop joints. If there are no such forces acting at a particular joint, then that joint is said to be passive; and if all of

the loop joints are passive, then $\boldsymbol{\tau}^a = \mathbf{0}$. In this case, the dynamics calculation software can be simplified by omitting the code required to calculate $\boldsymbol{\tau}^a$.

$\boldsymbol{\tau}^a$ is assumed to be known; but $\boldsymbol{\tau}^c$ is unknown, and it can be expressed as a function of n^c unknown constraint force variables (as in Eq. 8.5). Thus, Eq. 8.3 is a system of n equations in $n + n^c$ unknowns, and therefore needs to be supplemented with another n^c equations before it can be solved. These equations come from the kinematic constraints.

The loop joints impose a set of kinematic constraints on the tree, which can be collected into a single matrix equation of the form

$$\boldsymbol{K}\ddot{\boldsymbol{q}} = \boldsymbol{k}\,, \tag{8.4}$$

where $\boldsymbol{K}$ is an $n^c \times n$ matrix. We say that this equation expresses the constraints at the *acceleration level*, on the grounds that it has been differentiated a sufficient number of times for the acceleration variables to appear. Expressions for $\boldsymbol{K}$ and $\boldsymbol{k}$ are derived in Section 8.2.

As $\boldsymbol{\tau}^c$ is the force that imposes these constraints on the tree, it follows that $\boldsymbol{\tau}^c$ can be expressed in the form

$$\boldsymbol{\tau}^c = \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\lambda}\,, \tag{8.5}$$

where $\boldsymbol{\lambda}$ is a vector of n^c unknown constraint force variables. This step is covered in Section 8.4. (See also Eq. 3.15.)

Equations 8.3, 8.4 and 8.5 can now be assembled into a single matrix equation, being the complete equation of motion for the closed-loop system:

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{K}^{\mathrm{T}} \\ \boldsymbol{K} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}^a \\ \boldsymbol{k} \end{bmatrix}. \tag{8.6}$$

This is a system of $n + n^c$ equations in $n + n^c$ unknowns. The coefficient matrix is symmetric, but not positive definite. If this matrix has full rank, then the equation can be solved for both $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\lambda}$. If it does not have full rank, then some elements of $\boldsymbol{\lambda}$ will be indeterminate, but the equation can still be solved for $\ddot{\boldsymbol{q}}$. The rank of the coefficient matrix is $n + \mathrm{rank}(\boldsymbol{K})$.

8.2 Loop Constraint Equations

Let us now formulate the coefficients of the loop constraint equation, Eq. 8.4. We start with the velocity across loop joint k, which is denoted $\boldsymbol{v}_{\mathrm{J}k}$. This velocity is given by the equation

$$\boldsymbol{v}_{\mathrm{J}k} = \boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}\,, \tag{8.7}$$

where $p(k)$ and $s(k)$ are the predecessor and successor bodies, respectively, of joint k. In the general case, joint k will impose a constraint of the form

$$\boldsymbol{T}_k^{\mathrm{T}}\boldsymbol{v}_{\mathrm{J}k} = \boldsymbol{T}_k^{\mathrm{T}}\boldsymbol{\sigma}_k\,,$$

where $\boldsymbol{T}_k$ and $\boldsymbol{\sigma}_k$ denote the constraint-force subspace and bias velocity, respectively, of joint k. (See Eq. 3.38.) However, for the sake of simplicity, we will assume that $\boldsymbol{\sigma}_k = \boldsymbol{0}$ for all loop joints, so that the constraint equation simplifies to

$$\boldsymbol{T}_k^{\mathrm{T}} \boldsymbol{v}_{\mathrm{J}k} = \boldsymbol{0} \,. \tag{8.8}$$

Combining Eqs. 8.7 and 8.8 gives a velocity constraint equation,

$$\boldsymbol{T}_k^{\mathrm{T}} (\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) = \boldsymbol{0} \,, \tag{8.9}$$

which can be differentiated to obtain an acceleration constraint equation:

$$\boldsymbol{T}_k^{\mathrm{T}} (\boldsymbol{a}_{s(k)} - \boldsymbol{a}_{p(k)}) + \dot{\boldsymbol{T}}_k^{\mathrm{T}} (\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) = \boldsymbol{0} \,. \tag{8.10}$$

The velocity of any body in a kinematic tree can be expressed in terms of $\dot{\boldsymbol{q}}$ by an equation of the form

$$\boldsymbol{v}_i = \boldsymbol{J}_i \dot{\boldsymbol{q}} \,, \tag{8.11}$$

where $\boldsymbol{v}_i$ is the velocity of body i, and $\boldsymbol{J}_i$ is the Jacobian of body i. (See Example 4.2.) $\boldsymbol{J}_i$ is given by the equation

$$\boldsymbol{J}_i = \begin{bmatrix} \epsilon_{i1} \boldsymbol{S}_1 & \epsilon_{i2} \boldsymbol{S}_2 & \cdots & \epsilon_{iN_B} \boldsymbol{S}_{N_B} \end{bmatrix} \,, \tag{8.12}$$

where ϵ_{ij} takes the value 1 at every joint j that supports body i (i.e., the joint lies on the path between the body and the base), and zero otherwise. Expressed as an equation,

$$\epsilon_{ij} = \begin{cases} 1 & \text{if } j \in \kappa(i) \\ 0 & \text{otherwise.} \end{cases}$$

Likewise, the acceleration of any body in a kinematic tree can be expressed in terms of $\ddot{\boldsymbol{q}}$ by the derivative of Eq. 8.11:

$$\begin{aligned} \boldsymbol{a}_i &= \boldsymbol{J}_i \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i \dot{\boldsymbol{q}} \\ &= \boldsymbol{J}_i \ddot{\boldsymbol{q}} + \boldsymbol{a}_i^{vp} \,. \end{aligned} \tag{8.13}$$

$\boldsymbol{a}_i^{vp}$ is the 'velocity-product' acceleration of body i, which is the acceleration it would have if all the tree-joint acceleration variables were zero. It is available as one of the by-products of calculating $\boldsymbol{C}$. (See Section 8.6.)

Combining Eqs. 8.10 and 8.13 gives

$$\boldsymbol{T}_k^{\mathrm{T}} (\boldsymbol{J}_{s(k)} - \boldsymbol{J}_{p(k)}) \ddot{\boldsymbol{q}} + \boldsymbol{T}_k^{\mathrm{T}} (\boldsymbol{a}_{s(k)}^{vp} - \boldsymbol{a}_{p(k)}^{vp}) + \dot{\boldsymbol{T}}_k^{\mathrm{T}} (\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) = \boldsymbol{0} \,,$$

which we can simplify to

$$\boldsymbol{T}_k^{\mathrm{T}} (\boldsymbol{J}_{s(k)} - \boldsymbol{J}_{p(k)}) \ddot{\boldsymbol{q}} = \boldsymbol{k}_l \tag{8.14}$$

by defining

$$\boldsymbol{k}_l = -\boldsymbol{T}_k^{\mathrm{T}} (\boldsymbol{a}_{s(k)}^{vp} - \boldsymbol{a}_{p(k)}^{vp}) - \dot{\boldsymbol{T}}_k^{\mathrm{T}} (\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) \,. \tag{8.15}$$

Equation 8.14 is the acceleration constraint imposed on the spanning tree by loop joint k, and there is one such equation for each kinematic loop. Collecting these equations together, we get

$$\begin{bmatrix} \boldsymbol{T}_{N_B+1}^{\mathrm{T}} \left(\boldsymbol{J}_{s(N_B+1)} - \boldsymbol{J}_{p(N_B+1)} \right) \\ \vdots \\ \boldsymbol{T}_{N_J}^{\mathrm{T}} \left(\boldsymbol{J}_{s(N_J)} - \boldsymbol{J}_{p(N_J)} \right) \end{bmatrix} \ddot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{k}_1 \\ \vdots \\ \boldsymbol{k}_{N_L} \end{bmatrix}. \tag{8.16}$$

On comparing this equation with Eq. 8.4, it can be seen that

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{T}_{N_B+1}^{\mathrm{T}} \left(\boldsymbol{J}_{s(N_B+1)} - \boldsymbol{J}_{p(N_B+1)} \right) \\ \vdots \\ \boldsymbol{T}_{N_J}^{\mathrm{T}} \left(\boldsymbol{J}_{s(N_J)} - \boldsymbol{J}_{p(N_J)} \right) \end{bmatrix} \quad \text{and} \quad \boldsymbol{k} = \begin{bmatrix} \boldsymbol{k}_1 \\ \vdots \\ \boldsymbol{k}_{N_L} \end{bmatrix}. \tag{8.17}$$

The expression $\boldsymbol{J}_{s(k)} - \boldsymbol{J}_{p(k)}$ defines a *loop Jacobian*—a matrix that maps $\dot{\boldsymbol{q}}$ to the velocity across a loop-closing joint. The loop Jacobian for loop l, denoted $\boldsymbol{J}_{\mathrm{L}l}$, can be defined as follows:

$$\begin{aligned} \boldsymbol{J}_{\mathrm{L}l} &= \boldsymbol{J}_{s(k)} - \boldsymbol{J}_{p(k)} \\ &= \begin{bmatrix} \varepsilon_{l1}\boldsymbol{S}_1 & \varepsilon_{l2}\boldsymbol{S}_2 & \cdots & \varepsilon_{lN_B}\boldsymbol{S}_{N_B} \end{bmatrix} \end{aligned} \tag{8.18}$$

where $\varepsilon_{lj} = \epsilon_{s(k)j} - \epsilon_{p(k)j}$. These numbers identify the tree joints that take part in each loop. If we define the root of loop l to be the highest-numbered body that is a common ancestor of both $p(k)$ and $s(k)$, then the set of tree joints that take part in loop l is the union of two subsets: those that connect the root to $p(k)$, and those that connect the root to $s(k)$. The value of ε_{lj} is -1 for each joint in the first subset, $+1$ for each joint in the second, and zero for all other joints. Using Eq. 8.18, we can formulate an alternative expression for $\boldsymbol{K}$, which is the expression used to calculate it:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{11} & \cdots & \boldsymbol{K}_{1N_B} \\ \vdots & & \vdots \\ \boldsymbol{K}_{N_L1} & \cdots & \boldsymbol{K}_{N_L N_B} \end{bmatrix} \tag{8.19}$$

where

$$\boldsymbol{K}_{lj} = \varepsilon_{lj}\boldsymbol{T}_k^{\mathrm{T}}\boldsymbol{S}_j. \tag{8.20}$$

8.3   Constraint Stabilization

Equation 8.16 is theoretically correct, but is not stable during numerical integration. The problem is this: in principle, Eq. 8.16 behaves like a differential equation of the form

$$\ddot{e} = 0,$$

where e denotes a loop-closure position error; but in practice it behaves like

$$\ddot{e} = noise\,,$$

where *noise* is a small-magnitude signal representing numerical errors, such as the truncation errors in the numerical integration process. Thus, the use of Eq. 8.16 will ensure that the acceleration errors are kept small, but there is nothing to stop an unbounded accumulation of position and velocity errors. The problem can be solved by adding a stabilization term, $\boldsymbol{k}_{stab}$, to Eq. 8.4 (and therefore also to Eq. 8.6), so that it now reads

$$\boldsymbol{K}\ddot{\boldsymbol{q}} = \boldsymbol{k} + \boldsymbol{k}_{stab}\,. \tag{8.21}$$

The purpose of $\boldsymbol{k}_{stab}$ is to modify the behaviour of the constraint equation under numerical integration, so that it now behaves like an equation of the form

$$\ddot{e} + 2\alpha\,\dot{e} + \beta^2 e = noise\,,$$

where $\alpha > 0$ and $\beta \neq 0$. This method is due to Baumgarte, and is sometimes called Baumgarte stabilization (Baumgarte, 1972). The stabilization constants are usually chosen according to

$$\alpha = \beta = 1/T_{stab}\,,$$

where T_{stab} is the desired time constant for the decay of position and velocity errors. To achieve this effect, $\boldsymbol{k}_{stab}$ is defined as follows:

$$\boldsymbol{k}_{stab} = -2\alpha \begin{bmatrix} \boldsymbol{T}_{N_B+1}^{\mathrm{T}}\boldsymbol{v}_{\mathrm{J}N_B+1} \\ \vdots \\ \boldsymbol{T}_{N_J}^{\mathrm{T}}\boldsymbol{v}_{\mathrm{J}N_J} \end{bmatrix} - \beta^2 \begin{bmatrix} \boldsymbol{\delta}_1 \\ \vdots \\ \boldsymbol{\delta}_{N_L} \end{bmatrix} \tag{8.22}$$

where

$$\boldsymbol{\delta}_l = \mathrm{jcalc}(\,\mathrm{jtype}(k),\,{}^{s(k),k}\boldsymbol{X}_{p(k),k})\,. \tag{8.23}$$

$\boldsymbol{\delta}_l$ is a measure of the degree to which the position of the spanning tree violates the constraint imposed by loop joint k. It is a quantity of the form $\boldsymbol{\delta}_l = \boldsymbol{T}_k^{\mathrm{T}}\boldsymbol{d}_l$, where $\boldsymbol{d}_l$ is a spatial vector measuring the position error around loop l. This vector depends only on the joint type and the transform that locates the joint's successor frame relative to its predecessor frame. The latter is given by

$$\begin{aligned}
{}^{s(k),k}\boldsymbol{X}_{p(k),k} &= {}^{s(k),k}\boldsymbol{X}_{s(k)}\,{}^{s(k)}\boldsymbol{X}_{p(k)}\,{}^{p(k)}\boldsymbol{X}_{p(k),k} \\
&= \boldsymbol{X}_{\mathrm{S}}(l)\,{}^{s(k)}\boldsymbol{X}_{p(k)}\,\boldsymbol{X}_{\mathrm{P}}(l)^{-1}, \tag{8.24}
\end{aligned}$$

where $\boldsymbol{X}_{\mathrm{P}}$ and $\boldsymbol{X}_{\mathrm{S}}$ are the arrays of loop-joint predecessor and successor frame transforms stored in the system model. (See §4.2.)

One problem with Baumgarte stabilization is that there is no hard-and-fast rule for selecting T_{stab}. A reasonable strategy is to ask how many snapshots

per second of the moving system you would need in order not to miss anything important, and choose a value of T_{stab} that is similar to, or a little shorter than, the period between snapshots. For example, if the system is a large industrial robot, then $T_{stab} = 1$ is too large, $T_{stab} = 0.1$ is reasonable, and $T_{stab} = 0.01$ is a bit small, but still reasonable. If T_{stab} is chosen too small, then the differential equations become unnecessarily stiff. The exact value of T_{stab} is not critical—changing it by a factor of 2 makes only a small difference.

There is a temptation to choose T_{stab} as small as possible, so that position and velocity errors will decay as quickly as possible, in the belief that this will maximize the accuracy of the simulation. This is a bad strategy. The purpose of constraint stabilization is to achieve stability, not accuracy. If the simulation is not accurate enough, then the best way to improve it is to use a better integration method and/or a shorter integration time step.

Calculating δ

Let ^sX_p be the coordinate transform from the predecessor frame to the successor frame of a loop joint, as determined by the position variables of the spanning tree. In general, this transform will not comply exactly with the motion constraint imposed by the loop joint. Therefore, let us factorize this matrix as follows:

$$^sX_p = X_{err}\, X_J\,, \tag{8.25}$$

where X_J complies exactly with the joint constraint, and X_{err} represents the constraint error, which is assumed to be small. This factorization must be solved symbolically, using knowledge of the type of motion allowed by the joint, in order to obtain expressions for the elements of X_{err} in terms of ^sX_p. These expressions will be different for each joint type. Given that X_{err} represents a small displacement, there exists a motion vector, $d \in \mathsf{M}^6$, that satisfies

$$\mathbf{1} - d\times \simeq X_{err}\,. \tag{8.26}$$

This vector is (approximately) the displacement from where the successor frame ought to be to where it actually is. Having obtained d, δ is then given by

$$\delta = T^T d\,. \tag{8.27}$$

Formulae for δ for various common joint types are shown in Table 8.1. Note that a zero-DoF joint can serve as a *universal loop joint*, since any closed-loop system can be modified, by adding massless bodies and zero-DoF joints, in such a way that every loop-closing joint is a zero-DoF joint.

Example 8.1 *The formula in Table 8.1 for the cylindrical joint can be obtained as follows. By combining Eqs. 8.25 and 8.26, and choosing expressions for X_J and d that are appropriate for a cylindrical joint, we get the following*

Zero DoF	Revolute	Prismatic	Cylindrical
$\frac{1}{2}\begin{bmatrix} X_{23} - X_{32} \\ X_{31} - X_{13} \\ X_{12} - X_{21} \\ X_{53} - X_{62} \\ X_{61} - X_{43} \\ X_{42} - X_{51} \end{bmatrix}$	$\begin{bmatrix} X_{23} \\ -X_{13} \\ X_{53} \\ -X_{43} \\ X_{41}X_{21} + X_{42}X_{22} \end{bmatrix}$	$\begin{bmatrix} \frac{1}{2}(X_{23} - X_{32}) \\ \frac{1}{2}(X_{31} - X_{13}) \\ \frac{1}{2}(X_{12} - X_{21}) \\ X_{53} \\ -X_{43} \end{bmatrix}$	$\begin{bmatrix} X_{23} \\ -X_{13} \\ X_{53} \\ -X_{43} \end{bmatrix}$

$$\text{Spherical} \quad \begin{bmatrix} X_{51}X_{31} + X_{52}X_{32} + X_{53}X_{33} \\ X_{61}X_{11} + X_{62}X_{12} + X_{63}X_{13} \\ X_{41}X_{21} + X_{42}X_{22} + X_{43}X_{23} \end{bmatrix}$$

Table 8.1: Formulae for $\boldsymbol{\delta} = \text{jcalc}(type, \boldsymbol{X})$

approximate equation:

$$^{s}\boldsymbol{X}_p \simeq \begin{bmatrix} 1 & 0 & -d_2 & 0 & 0 & 0 \\ 0 & 1 & d_1 & 0 & 0 & 0 \\ d_2 & -d_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -d_5 & 1 & 0 & -d_2 \\ 0 & 0 & d_4 & 0 & 1 & d_1 \\ d_5 & -d_4 & 0 & d_2 & -d_1 & 1 \end{bmatrix} \begin{bmatrix} c & s & 0 & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & z & 0 & c & s & 0 \\ -z & 0 & 0 & -s & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

where $c = \cos(\theta)$, $s = \sin(\theta)$, and θ and z are the angular and linear displacements of the cylindrical joint (i.e., the joint position variables). Observe that we have set $d_3 = d_6 = 0$ as they are the coordinates of $\boldsymbol{d}$ in the directions of motion freedom of the joint. If we perform the multiplication, and examine the result, then we find that $d_1 = (^{s}X_p)_{23}$, $d_2 = -(^{s}X_p)_{13}$, and so on. Finally, $\boldsymbol{\delta} = [d_1 \ d_2 \ d_4 \ d_5]^{\mathrm{T}}$.

8.4   Loop Joint Forces

Let $\boldsymbol{f}_k$ denote the force transmitted across loop joint k in the closed-loop system. The effect of joint k on the spanning tree is therefore to exert a force of $\boldsymbol{f}_k$ on body $s(k)$ and a force of $-\boldsymbol{f}_k$ on body $p(k)$. Now, if a spatial force of $\boldsymbol{f}$ is applied to body i in a kinematic tree, then it has the same effect on the tree as a joint-space force of $\boldsymbol{\tau}$, where

$$\boldsymbol{\tau} = \boldsymbol{J}_i^{\mathrm{T}} \boldsymbol{f}.$$

Therefore, the effect of joint k on the spanning tree is equivalent to a joint-space force of

$$\boldsymbol{\tau} = (\boldsymbol{J}_{s(k)}^{\mathrm{T}} - \boldsymbol{J}_{p(k)}^{\mathrm{T}})\boldsymbol{f}_k = \boldsymbol{J}_{\mathrm{L}l}^{\mathrm{T}} \boldsymbol{f}_k.$$

Let us decompose $\boldsymbol{f}_k$ into the sum of an active force, $\boldsymbol{f}_k^a$, and a constraint force, $\boldsymbol{f}_k^c$, and consider their effects separately. If $\boldsymbol{\tau}^a$ represents the net effect on the spanning tree of all the active forces at the loop joints, then $\boldsymbol{\tau}^a$ is given by

$$\boldsymbol{\tau}^a = \sum_{l=1}^{N_L} \boldsymbol{J}_{Ll}^{\mathrm{T}} \boldsymbol{f}_k^a \, . \tag{8.28}$$

Likewise, if $\boldsymbol{\tau}^c$ represents the net effect of all the constraint forces at the loop joints, then $\boldsymbol{\tau}^c$ is given by

$$\boldsymbol{\tau}^c = \sum_{l=1}^{N_L} \boldsymbol{J}_{Ll}^{\mathrm{T}} \boldsymbol{f}_k^c \, . \tag{8.29}$$

However, the constraint force at joint k can also be expressed in the form

$$\boldsymbol{f}_k^c = \boldsymbol{T}_k \, \boldsymbol{\lambda}_k \, , \tag{8.30}$$

where $\boldsymbol{\lambda}_k$ is a vector of n_k^c unknown constraint force variables (cf. Eq. 3.34), so $\boldsymbol{\tau}^c$ can also be expressed as

$$\boldsymbol{\tau}^c = \sum_{l=1}^{N_L} \boldsymbol{J}_{Ll}^{\mathrm{T}} \boldsymbol{T}_k \, \boldsymbol{\lambda}_k \, . \tag{8.31}$$

On comparing this equation with the definition of $\boldsymbol{K}$ in Eq. 8.17, it can be seen that

$$\boldsymbol{\tau}^c = \boldsymbol{K}^{\mathrm{T}} \boldsymbol{\lambda} \tag{8.32}$$

where

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_{N_B+1}^{\mathrm{T}} & \cdots & \boldsymbol{\lambda}_{N_J}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \, . \tag{8.33}$$

As mentioned earlier, the active forces are assumed to be known. The best way to incorporate $\boldsymbol{\tau}^a$ into the equation of motion is to modify the algorithm for calculating $\boldsymbol{C}$ so that it calculates $\boldsymbol{C} - \boldsymbol{\tau}^a$ instead. A suitable algorithm is presented in Section 8.6.

8.5 Solving the Equations of Motion

Having formulated every term in the equations of motion, the next step is to solve them for the tree-joint accelerations. To recap, the equation of motion for the spanning tree, subject to the loop-closure forces, is

$$\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} = \boldsymbol{\tau} + \boldsymbol{K}^{\mathrm{T}} \boldsymbol{\lambda} + \boldsymbol{\tau}^a \, ; \tag{8.34}$$

the loop-closure constraint equation is

$$\boldsymbol{K}\ddot{\boldsymbol{q}} = \boldsymbol{k} + \boldsymbol{k}_{stab} \, ; \tag{8.35}$$

and the two can be combined into a single composite equation,

$$\begin{bmatrix} H & K^{\mathrm{T}} \\ K & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} \tau - C + \tau^a \\ k + k_{stab} \end{bmatrix}. \tag{8.36}$$

In these equations, H is an $n \times n$ symmetric, positive-definite matrix, while K is a general $n^c \times n$ matrix that may be rank-deficient. The coefficient matrix in Eq. 8.36 is therefore symmetric, and it has dimensions of $(n + n^c) \times (n + n^c)$; but it is not positive definite, and it has a rank of $n + r$, where $r = \mathrm{rank}(K)$, so it will be singular whenever $r < n^c$.

If $r = n^c$, then both $\ddot{q}$ and λ are uniquely determined by Eq. 8.36. However, if $r < n^c$, then $n^c - r$ of the constraints imposed on the spanning tree by the loop joints are linearly dependent on the other r constraints. As a result, some elements of λ will be indeterminate, but $\ddot{q}$ is still uniquely determined by the equation. Another problem that can arise if $r < n^c$ is that Eq. 8.35 can be slightly inconsistent; that is, $k + k_{stab}$ can lie slightly outside range(K). This problem is caused by numerical errors in the calculation of K, k and k_{stab}, and by inexact satisfaction of the loop-closure constraints for positions and velocities. When Eq. 8.35 is slightly inconsistent, it is acceptable to solve it in a least-squares sense.

There are various ways to solve the above equations for $\ddot{q}$. Let us now examine some of them. In particular, let us consider the following three methods:

1. solve Eq. 8.36 directly;

2. solve Eq. 8.36 for λ, and use the result to obtain $\ddot{q}$; or

3. solve Eq. 8.35 to express $\ddot{q}$ in terms of a vector of independent acceleration variables, and then solve Eq. 8.34.

Method 1: Solve Eq. 8.36 Directly

If the coefficient matrix in Eq. 8.36 is nonsingular, then this equation can be solved directly by a general-purpose linear equation solver, or a solver designed for symmetric, indefinite matrices. This is the simplest solution method, but not the most efficient. It is therefore appropriate whenever human effort is more important than computational efficiency. The coefficient matrix will be nonsingular if $r = n^c$, or if $n^c - r$ linearly-dependent rows are pruned out of Eq. 8.35 before it is incorporated into Eq. 8.36. The latter is practical if one happens to know in advance which rows to remove. This pruning tactic amounts to reducing n^c until $n^c = r$. Pruning can also be used in conjunction with methods 2 and 3.

Method 2: Solve for λ First

By subtracting $K H^{-1}$ times the first row from the second row in Eq. 8.36, we get an equation for λ of the form

$$A \lambda = b, \tag{8.37}$$

where

$$A = KH^{-1}K^{\mathrm{T}} \tag{8.38}$$

and

$$b = k + k_{stab} - KH^{-1}(\tau - C + \tau^a). \tag{8.39}$$

The coefficient matrix, A, is symmetric and positive-semidefinite. Its size is $n^c \times n^c$, and its rank is r. If $r = n^c$ then A is invertible, and Eq. 8.37 has the unique solution

$$\lambda = A^{-1}b. \tag{8.40}$$

If $r < n^c$ then A is singular, and Eq. 8.37 has an infinity of solutions. In this case, the general form of the solution is

$$\lambda = A^+ b + (1 - A^+A)z, \tag{8.41}$$

where A^+ is the pseudoinverse of A, z is an arbitrary vector, and the operator $1 - A^+A$ projects vectors onto the null space of A. If Eq. 8.37 is consistent, then Eq. 8.41 describes a family of exact solutions; otherwise, it describes a family of least-squares solutions. Equation 8.37 is consistent if and only if Eq. 8.35 is consistent.

Having obtained a value for λ, the final step is to substitute it into Eq. 8.34, and solve the resulting equation for $\ddot{q}$. Note that z has no effect on $\ddot{q}$, because

$$K^{\mathrm{T}}(1 - A^+A) = 0.$$

One is therefore free to choose any convenient value for z, such as $z = 0$.

The following procedure will calculate $\ddot{q}$ in an efficient manner. It uses the $L^{\mathrm{T}}L$ factorization and back-substitution algorithms described in Section 6.5, and therefore exploits any branch-induced sparsity that may be present in H. It can easily be modified to use the $L^{\mathrm{T}}DL$ factorization instead.

1. Factorize H into $H = L^{\mathrm{T}}L$.

2. Calculate $\tau' = \tau - C + \tau^a$.

3. Using back-substitution, calculate $Y = L^{-\mathrm{T}}K^{\mathrm{T}}$ and $z = L^{-\mathrm{T}}\tau'$.

4. Calculate $A = Y^{\mathrm{T}}Y$ and $b = k + k_{stab} - Y^{\mathrm{T}}z$.

5. Solve $A\lambda = b$ for λ by any appropriate method.

6. Solve $H\ddot{q} = \tau' + K^{\mathrm{T}}\lambda$ for $\ddot{q}$ using the factors from step 1.

Method 3: Solve Eq. 8.35 First

Suppose we have been given a general linear equation, $Ax = b$, in which A is an $m \times n$ matrix of rank r, and $b \in \mathrm{range}(A)$ (i.e., the equation is consistent). The task is to solve it for x. If $r = n$ then there will be only a single solution, which is given by the formula $x = A^+b$. If $r < n$ then there will be an infinite number of solutions, which are given by the formula $x = Cy + d$. In this

equation, $\boldsymbol{y}$ is an $(n-r)$-dimensional vector of unknowns, $\boldsymbol{d}$ is any one vector satisfying $\boldsymbol{Ad} = \boldsymbol{b}$, and $\boldsymbol{C}$ is a full-rank matrix having the property $\boldsymbol{AC} = \boldsymbol{0}$. Each solution has a unique corresponding value of $\boldsymbol{y}$, and vice versa (i.e., there is a $1:1$ correspondence between the two).

Let us apply this formula to Eq. 8.35. This equation has an $n^c \times n$ coefficient matrix of rank r. If $r = n$ then the equation has a unique solution, which is $\ddot{\boldsymbol{q}} = \boldsymbol{K}^+(\boldsymbol{k} + \boldsymbol{k}_{stab})$; but if $r < n$ then there are infinitely many solutions, all given by the formula

$$\ddot{\boldsymbol{q}} = \boldsymbol{G}\,\ddot{\boldsymbol{y}} + \boldsymbol{g}\,. \tag{8.42}$$

In this equation, $\boldsymbol{G}$ is an $n \times (n-r)$ full-rank matrix having the property

$$\boldsymbol{KG} = \boldsymbol{0}\,, \tag{8.43}$$

$\ddot{\boldsymbol{y}}$ is an $(n-r)$-dimensional vector of independent acceleration variables, and $\boldsymbol{g}$ is any one vector satisfying

$$\boldsymbol{Kg} = \boldsymbol{k} + \boldsymbol{k}_{stab}\,. \tag{8.44}$$

Typically, $\ddot{\boldsymbol{y}}$ is chosen to be a subset of the elements of $\ddot{\boldsymbol{q}}$. It can therefore be thought of as a vector of independent joint acceleration variables for the closed-loop system. An algorithm for calculating $\boldsymbol{G}$ and $\boldsymbol{g}$ is presented in Section 8.8.

We now have an equation that expresses $\ddot{\boldsymbol{q}}$ as a function of the unknown vector $\ddot{\boldsymbol{y}}$. The next step is therefore to find an expression for $\ddot{\boldsymbol{y}}$. Premultiplying Eq. 8.34 by $\boldsymbol{G}^{\mathrm{T}}$ causes $\boldsymbol{\lambda}$ to be eliminated, resulting in

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{H}\,\ddot{\boldsymbol{q}} = \boldsymbol{G}^{\mathrm{T}}(\boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}^a)\,;$$

and substituting Eq. 8.42 into this equation produces

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{HG}\,\ddot{\boldsymbol{y}} = \boldsymbol{G}^{\mathrm{T}}(\boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}^a - \boldsymbol{Hg}) \tag{8.45}$$

(cf. Eq. 3.20). This equation can be regarded as the equation of motion for the closed-loop system, expressed in terms of the independent joint acceleration variables in $\ddot{\boldsymbol{y}}$. The matrix $\boldsymbol{G}^{\mathrm{T}}\boldsymbol{HG}$ is symmetric and positive-definite, so Eq. 8.45 can be solved immediately for $\ddot{\boldsymbol{y}}$, giving

$$\ddot{\boldsymbol{y}} = (\boldsymbol{G}^{\mathrm{T}}\boldsymbol{HG})^{-1}\boldsymbol{G}^{\mathrm{T}}(\boldsymbol{\tau} - \boldsymbol{C} + \boldsymbol{\tau}^a - \boldsymbol{Hg})\,. \tag{8.46}$$

8.6   Algorithm for $C - \tau^a$

The best strategy for dealing with $\boldsymbol{\tau}^a$ is to choose the spanning tree such that every loop joint is passive. In this case, $\boldsymbol{f}_k^a = \boldsymbol{0}$ for every loop joint, and so $\boldsymbol{\tau}^a = \boldsymbol{0}$. The next best strategy is to modify the code for calculating $\boldsymbol{C}$ so that it calculates $\boldsymbol{C} - \boldsymbol{\tau}^a$ instead.

Original Equations:	Modified Equations:

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i \qquad (\boldsymbol{v}_0 = \boldsymbol{0})$$

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i \quad (\boldsymbol{a}_0 = -\boldsymbol{a}_g)$$

$$\boldsymbol{f}_i^B = \boldsymbol{I}_i\,\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i$$

$$\boldsymbol{f}_i = \boldsymbol{f}_i^B - \boldsymbol{f}_i^x + \sum_{j \in \mu(i)} \boldsymbol{f}_j$$

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{f}_i$$

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i \qquad (\boldsymbol{v}_0 = \boldsymbol{0})$$

$$\boldsymbol{a}_i^{vp} = \boldsymbol{a}_{\lambda(i)}^{vp} + \dot{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i \qquad (\boldsymbol{a}_0^{vp} = -\boldsymbol{a}_g)$$

$$\boldsymbol{f}_i^B = \boldsymbol{I}_i\,\boldsymbol{a}_i^{vp} + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i$$

$$\boldsymbol{f}_i = \boldsymbol{f}_i^B - \boldsymbol{f}_i^x - \sum_{k=N_B+1}^{N_J} e_{ik}\boldsymbol{f}_k^a + \sum_{j \in \mu(i)} \boldsymbol{f}_j$$

$$\text{where } e_{ik} = \begin{cases} +1 & \text{if } i = s(k) \\ -1 & \text{if } i = p(k) \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{C}'_i = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{f}_i$$

Table 8.2: Equations for calculating $C - \tau^a$

Table 8.2 shows the basic equations of the recursive Newton-Euler algorithm, as they appear in Table 5.1, alongside the equations for calculating $C - \tau^a$. On comparing the originals with the modified equations, the following differences can be seen.

1. The term $\boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$ has been dropped, as the acceleration variables are set to zero for calculating C.

2. The variable names $\boldsymbol{a}_i$ and $\boldsymbol{\tau}_i$ have been changed to $\boldsymbol{a}_i^{vp}$ and $\boldsymbol{C}'_i$ (where $\boldsymbol{C}' = \boldsymbol{C} - \boldsymbol{\tau}^a$). This is a purely cosmetic change.[1]

3. A new term has been added to the joint force calculation, which subtracts the active loop-joint forces at the same point where the general external forces are subtracted.

From the perspective of the spanning tree, the active loop-joint forces are indeed external forces, since they come from something that is not a part of the tree, and that is exactly how they are treated. The coefficients e_{ik} ensure that $\boldsymbol{f}_k^a$ gets subtracted only from $\boldsymbol{f}_{s(k)}$, and that $-\boldsymbol{f}_k^a$ gets subtracted from $\boldsymbol{f}_{p(k)}$.

Table 8.3 shows the pseudocode for an algorithm to calculate $C - \tau^a$ according to the equations in Table 8.2. The calculations are performed in body coordinates. The first and third loops in this algorithm closely resemble the corresponding code in Table 5.1, but the second loop is new. The last line in loop 1 initializes each vector $\boldsymbol{f}_i$ to the expression $\boldsymbol{f}_i^B - \boldsymbol{f}_i^x$. The purpose of loop 2 is then to subtract the term $\sum_{k=N_B+1}^{N_J} e_{ik}\boldsymbol{f}_k^a$ from each $\boldsymbol{f}_i$ before loop 3 starts to use them.

The second line in loop 2 uses jcalc to calculate $\boldsymbol{T}_k^a$, which is placed in the local variable $\boldsymbol{T}^a$. There is an assumption here that $\boldsymbol{T}_k^a$ is a constant, so that

---

[1]Strictly speaking $\boldsymbol{a}_i^{vp}$ in these equations has the value $\dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}} - \boldsymbol{a}_g$, rather than $\dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}}$ as defined in Eq. 8.13, but the constant offset $\boldsymbol{a}_g$ cancels out in the subtraction in Eq. 8.15.

$\boldsymbol{v}_0 = \boldsymbol{0}$

$\boldsymbol{a}_0 = -\boldsymbol{a}_g$

for $i = 1$ **to** N_B **do**

 $[\boldsymbol{X}_{\mathrm{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\mathrm{J}}, \boldsymbol{c}_{\mathrm{J}}] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$

 ${}^i\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}}\,\boldsymbol{X}_{\mathrm{T}}(i)$

 if $\lambda(i) \neq 0$ **then**

  ${}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{X}_0$

 end

 $\boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}}$

 $\boldsymbol{a}_i^{vp} = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)}^{vp} + \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}}$

 $\boldsymbol{f}_i = \boldsymbol{I}_i\,\boldsymbol{a}_i^{vp} + \boldsymbol{v}_i \times^*\boldsymbol{I}_i\,\boldsymbol{v}_i - {}^i\boldsymbol{X}_0^*\,\boldsymbol{f}_i^x$

end

$\vdots$

$\vdots$

for $l = 1$ **to** N_L **do**

 $k = l + N_B$

 $[\boldsymbol{T}^a] = \mathrm{jcalc}(\mathrm{jtype}(k))$

 $\boldsymbol{f}^a = \boldsymbol{X}_{\mathrm{S}}(l)^{\mathrm{T}}\,\boldsymbol{T}^a\,\boldsymbol{\tau}_k$

 $\boldsymbol{f}_{s(k)} = \boldsymbol{f}_{s(k)} - \boldsymbol{f}^a$

 $\boldsymbol{f}_{p(k)} = \boldsymbol{f}_{p(k)} + {}^{p(k)}\boldsymbol{X}_0^*\,{}^0\boldsymbol{X}_{s(k)}^*\,\boldsymbol{f}^a$

end

for $i = N_B$ **to** 1 **do**

 $\boldsymbol{C}_i' = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{f}_i$

 if $\lambda(i) \neq 0$ **then**

  $\boldsymbol{f}_i = \boldsymbol{f}_i + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{f}_i$

 end

end

Table 8.3: Algorithm to calculate $\boldsymbol{C} - \boldsymbol{\tau}^a$

jcalc only needs to know the type specifier for joint k in order to work out the correct value. If we did not make this assumption, then it would be necessary to work out the transform from the predecessor frame to the successor frame of joint k, so that it can be supplied as an argument to jcalc.

The expression $\boldsymbol{T}^a\boldsymbol{\tau}_k$ gives $\boldsymbol{f}_k^a$ in the successor-frame coordinates of joint k, which are the coordinates in which $\boldsymbol{T}^a$ is supplied. The third line in loop 2 calculates $\boldsymbol{f}_k^a$ in successor-frame coordinates, transforms it to body $s(k)$ coordinates, and stores the result in the local variable $\boldsymbol{f}^a$. The next two lines then subtract this quantity from $\boldsymbol{f}_{s(k)}$ and add it to $\boldsymbol{f}_{p(k)}$. Observe that $\boldsymbol{f}^a$ must be transformed from $s(k)$ to $p(k)$ coordinates before the addition is carried out.

8.7   Algorithm for $\boldsymbol{K}$ and $\boldsymbol{k}$

Table 8.4 shows the pseudocode for an algorithm to calculate $\boldsymbol{K}$ via Eqs. 8.19 and 8.20, $\boldsymbol{k}$ via Eq. 8.15, and $\boldsymbol{k}_{stab}$ via Eqs. 8.22 and 8.23. These calculations involve combining vectors and matrices that were originally obtained in various different coordinate systems. It is therefore necessary to apply coordinate transforms to at least some of these quantities before they can be combined. The strategy used in this algorithm is to transform everything to base coordinates and combine them there. This is the simplest strategy, but not necessarily the best.

The local variables $\boldsymbol{X}_p$ and $\boldsymbol{X}_s$ contain the Plücker transforms from base coordinates to the predecessor and successor frames, respectively, of loop joint k; so $\boldsymbol{X}_p = {}^{p(k),k}\boldsymbol{X}_0$ and $\boldsymbol{X}_s = {}^{s(k),k}\boldsymbol{X}_0$. The expression $\boldsymbol{X}_s\,\boldsymbol{X}_p^{-1}$ is therefore the transform from the joint's predecessor frame to its successor frame, which is the argument to jcalc in Eq. 8.23. The call to jcalc requests both the loop position

$\boldsymbol{K} = \boldsymbol{0}$
for $l = 1$ to N_L do
 $k = l + N_B$
 $\boldsymbol{X}_p = \boldsymbol{X}_{\mathrm{P}}(l)\,{}^{p(k)}\boldsymbol{X}_0$
 $\boldsymbol{X}_s = \boldsymbol{X}_{\mathrm{S}}(l)\,{}^{s(k)}\boldsymbol{X}_0$
 $\boldsymbol{v}_p = {}^0\boldsymbol{X}_{p(k)}\,\boldsymbol{v}_{p(k)}$
 $\boldsymbol{v}_s = {}^0\boldsymbol{X}_{s(k)}\,\boldsymbol{v}_{s(k)}$
 $\boldsymbol{a}_p = {}^0\boldsymbol{X}_{p(k)}\,\boldsymbol{a}^{vp}_{p(k)}$
 $\boldsymbol{a}_s = {}^0\boldsymbol{X}_{s(k)}\,\boldsymbol{a}^{vp}_{s(k)}$
 $[\boldsymbol{\delta}, \boldsymbol{T}] = \mathrm{jcalc}(\mathrm{jtype}(k), \boldsymbol{X}_s\,\boldsymbol{X}_p^{-1})$
 $\boldsymbol{T} = \boldsymbol{X}_s^{\mathrm{T}}\,\boldsymbol{T}$
 $\boldsymbol{k}_{stab} = 2\alpha\,\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{v}_s - \boldsymbol{v}_p) + \beta^2\,\boldsymbol{\delta}$

$\boldsymbol{k}_l = -\boldsymbol{T}^{\mathrm{T}}(\boldsymbol{a}_s - \boldsymbol{a}_p + \boldsymbol{v}_s \times \boldsymbol{v}_p) - \boldsymbol{k}_{stab}$
$i = p(k)$
$j = s(k)$
while $i \neq j$ do
 if $i > j$ then
  $\boldsymbol{K}_{li} = -\boldsymbol{T}^{\mathrm{T}}\,{}^0\boldsymbol{X}_i\,\boldsymbol{S}_i$
  $i = \lambda(i)$
 else
  $\boldsymbol{K}_{lj} = \boldsymbol{T}^{\mathrm{T}}\,{}^0\boldsymbol{X}_j\,\boldsymbol{S}_j$
  $j = \lambda(j)$
 end
end
end

Table 8.4: Algorithm to calculate $\boldsymbol{K}$, $\boldsymbol{k}$ and $\boldsymbol{k}_{stab}$

error, $\boldsymbol{\delta}$, and the constraint-force subspace matrix, $\boldsymbol{T}$, for loop joint k. The line immediately after this call transforms $\boldsymbol{T}$ from successor-frame coordinates to base coordinates; but no transform is applied to $\boldsymbol{\delta}$ because it represents a quantity of the form $\boldsymbol{T}^{\mathrm{T}}\boldsymbol{d}$, where $\boldsymbol{d} \in \mathsf{M}^6$ is an infinitesimal displacement.

This algorithm assumes that $\mathring{\boldsymbol{T}} = \boldsymbol{0}$ and $\boldsymbol{\sigma} = \dot{\boldsymbol{\sigma}} = \boldsymbol{0}$ for every loop joint. Without this assumption, jcalc would need more arguments, and would be asked to return more quantities. The assumption $\mathring{\boldsymbol{T}} = \boldsymbol{0}$ implies $\dot{\boldsymbol{T}} = \boldsymbol{v}_s \times^* \boldsymbol{T}$, which allows us to simplify the right-hand side of Eq. 8.15 as follows:

$$
\begin{aligned}
\boldsymbol{k}_l &= -\boldsymbol{T}_k^{\mathrm{T}}(\boldsymbol{a}^{vp}_{s(k)} - \boldsymbol{a}^{vp}_{p(k)}) - \dot{\boldsymbol{T}}_k^{\mathrm{T}}(\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) \\
&= -\boldsymbol{T}_k^{\mathrm{T}}(\boldsymbol{a}^{vp}_{s(k)} - \boldsymbol{a}^{vp}_{p(k)}) - (\boldsymbol{v}_{s(k)} \times^* \boldsymbol{T}_k)^{\mathrm{T}}(\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) \\
&= -\boldsymbol{T}_k^{\mathrm{T}}(\boldsymbol{a}^{vp}_{s(k)} - \boldsymbol{a}^{vp}_{p(k)}) + \boldsymbol{T}_k^{\mathrm{T}}\boldsymbol{v}_{s(k)} \times (\boldsymbol{v}_{s(k)} - \boldsymbol{v}_{p(k)}) \\
&= -\boldsymbol{T}_k^{\mathrm{T}}(\boldsymbol{a}^{vp}_{s(k)} - \boldsymbol{a}^{vp}_{p(k)} + \boldsymbol{v}_{s(k)} \times \boldsymbol{v}_{p(k)}).
\end{aligned} \tag{8.47}
$$

This is the expression used in the algorithm.

$\boldsymbol{K}$ is calculated by first initializing the whole matrix to zero, and then calculating the nonzero submatrices. For each loop l, the submatrix $\boldsymbol{K}_{li}$ is nonzero for every tree joint i that lies on the path of the loop. These quantities are calculated in a **while** loop that employs two loop variables, i and j. These variables are initially set to $p(k)$ and $s(k)$, respectively, and then stepped back toward the base in an order that guarantees they meet at the nearest common ancestor of $p(k)$ and $s(k)$. At this point, every joint on the path of loop l is accounted for, and we have $i = j$, which is the termination condition for the loop.

The algorithm presented in Table 8.4 potentially involves some duplication of calculations. For example, if joint i participates in more than one loop, then

S_i will be transformed to base coordinates more than once. Likewise, if body i is the successor or predecessor of more than one loop joint, then v_i and a_i^{vp} will be transformed to base coordinates more than once, and the relevant transform (X_p or X_s) will be calculated more than once. This kind of duplication has a relatively small impact on the overall efficiency. Nevertheless, if maximum speed is required, then the code can be rewritten to avoid such duplications.

The Rounding-Error Problem

The algorithm's use of base coordinates achieves two things: it keeps the algorithm simple, and it minimizes the number of transformation matrices that are needed. However, there is a price to be paid: most floating-point calculations involve rounding errors, and some of these errors are sensitive to the distance between the frame in which the calculations are performed and the locations of the objects to which those calculations apply. Therefore, if the base coordinate frame is far away from the kinematic loop, then there will be a loss of precision in the calculations. This phenomenon is not a problem for most rigid-body systems; but it can be a problem for a floating-base system if it travels a great distance from its fixed-base origin. (Think of an aircraft, or a spacecraft.) In this case, a simple remedy is to perform the calculations for K, k and k_{stab} in floating-base coordinates instead of fixed-base coordinates.

8.8   Algorithm for G and g

This section shows how to solve a linear equation of the form

$$K\,x = k \tag{8.48}$$

to yield a solution of the form

$$x = G\,y + g\,. \tag{8.49}$$

The method is applicable to the task of solving Eq. 8.35 to get Eq. 8.42. In Equations 8.48 and 8.49, K is an $m \times n$ matrix of rank r, $k \in \text{range}(K)$ (so Eq. 8.48 is consistent), G is an $n \times (n-r)$ full-rank matrix, and Eq. 8.49 produces every possible solution to Eq. 8.48. In fact, Eq. 8.49 defines a $1:1$ mapping between y and the set of solutions to Eq. 8.48. If $r = n$ then Eq. 8.49 simplifies to $x = g$, and g is unique. If $r < n$ then G and g are not unique, and we seek any one pair of values that solves the problem.

Given that $\text{rank}(K) = r$, it follows that K must contain an $r \times r$ invertible submatrix. Therefore, there exists a permutation of K in which the elements of this submatrix occupy the top left corner. We may therefore write

$$K = Q_1 \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} Q_2\,,$$

in which $\boldsymbol{K}_{11}$ is an $r \times r$ invertible matrix, and $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are permutation matrices. Extending this permutation to the whole of Eq. 8.48 gives

$$\boldsymbol{Q}_1 \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{12} \\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \boldsymbol{Q}_1 \begin{bmatrix} \boldsymbol{k}_1 \\ \boldsymbol{k}_2 \end{bmatrix}, \tag{8.50}$$

where

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \boldsymbol{Q}_2\,\boldsymbol{x} \qquad \text{and} \qquad \boldsymbol{Q}_1 \begin{bmatrix} \boldsymbol{k}_1 \\ \boldsymbol{k}_2 \end{bmatrix} = \boldsymbol{k}\,.$$

Again, given that $\text{rank}(\boldsymbol{K}) = r$ and that Eq. 8.48 is consistent, it follows that the bottom row in Eq. 8.50 is a linear multiple of the top row; so there must exist a matrix $\boldsymbol{Y}$ such that $\boldsymbol{K}_{21} = \boldsymbol{Y}\boldsymbol{K}_{11}$, $\boldsymbol{K}_{22} = \boldsymbol{Y}\boldsymbol{K}_{12}$ and $\boldsymbol{k}_2 = \boldsymbol{Y}\boldsymbol{k}_1$. We can therefore rewrite Eq. 8.50 as

$$\boldsymbol{Q}_1 \begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{12} \\ \boldsymbol{Y}\boldsymbol{K}_{11} & \boldsymbol{Y}\boldsymbol{K}_{12} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \boldsymbol{Q}_1 \begin{bmatrix} \boldsymbol{k}_1 \\ \boldsymbol{Y}\boldsymbol{k}_1 \end{bmatrix}.$$

By inspection, we can see that one possible solution to this equation is

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} -\boldsymbol{K}_{11}^{-1}\boldsymbol{K}_{12} \\ \boldsymbol{1} \end{bmatrix} \boldsymbol{x}_2 + \begin{bmatrix} \boldsymbol{K}_{11}^{-1}\boldsymbol{k}_1 \\ \boldsymbol{0} \end{bmatrix}.$$

We may therefore conclude that one solution to the original problem is

$$\boldsymbol{G} = \boldsymbol{Q}_2^{\text{T}} \begin{bmatrix} -\boldsymbol{K}_{11}^{-1}\boldsymbol{K}_{12} \\ \boldsymbol{1} \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{g} = \boldsymbol{Q}_2^{\text{T}} \begin{bmatrix} \boldsymbol{K}_{11}^{-1}\boldsymbol{k}_1 \\ \boldsymbol{0} \end{bmatrix}. \tag{8.51}$$

This particular solution equates $\boldsymbol{y}$ with $\boldsymbol{x}_2$, which means that $\boldsymbol{y}$ is a subset of the elements of $\boldsymbol{x}$.

The expressions in Eq. 8.51 can be calculated using a standard Gaussian elimination algorithm, or an LU factorization, with the following modifications.

1. Complete pivoting must be used, with both row and column swaps. A record must be kept of the column swaps, as this data determines the value of $\boldsymbol{Q}_2$.

2. If r has not been supplied as an argument, then a rank test is needed to detect when the last of the independent rows has been processed.

3. The factorization process terminates once it has processed the last independent row. At this stage, the top-left $r \times r$ submatrix of $\boldsymbol{K}$ contains the upper-triangular matrix $\boldsymbol{U}$, where $\boldsymbol{L}\boldsymbol{U} = \boldsymbol{K}_{11}$; the top-right $r \times (n-r)$ submatrix contains $\boldsymbol{L}^{-1}\boldsymbol{K}_{12}$; and the top r rows of $\boldsymbol{k}$ contain $\boldsymbol{L}^{-1}\boldsymbol{k}_1$. Back-substitution on the top r rows therefore yields $\boldsymbol{K}_{11}^{-1}\boldsymbol{K}_{12}$ and $\boldsymbol{K}_{11}^{-1}\boldsymbol{k}_1$.

Various features can be added to this basic algorithm. For example, a mask can be supplied that identifies which columns of $\boldsymbol{K}$ correspond to joint variables that do not participate in any kinematic loop. These columns contain only zeros, and are therefore left unaltered by the factorization process. Another possibility is to allow the user to specify his own choice of independent variables. This eliminates the need for complete pivoting, so that partial pivoting can be used instead.
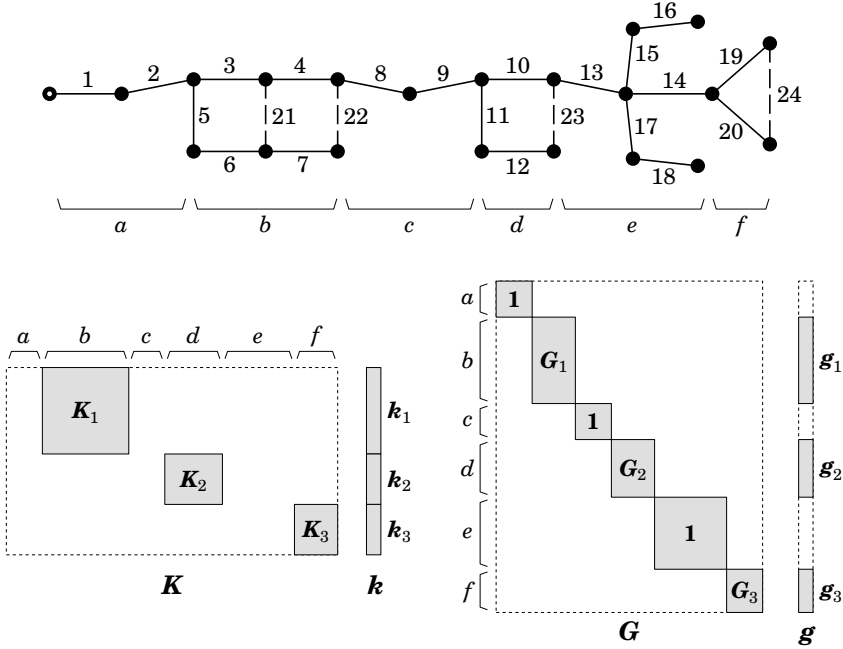
Figure 8.1: Loop clusters and the sparsity patterns of $\boldsymbol{K}$ and $\boldsymbol{G}$

8.9   Exploiting Sparsity in $\boldsymbol{K}$ and $\boldsymbol{G}$

It is clearly the case that both $\boldsymbol{K}$ and $\boldsymbol{G}$ may contain many zeros. For example, Eq. 8.20 implies that each submatrix $\boldsymbol{K}_{lj}$ of $\boldsymbol{K}$ will be zero for every l and j such that $\varepsilon_{lj} = 0$; and Eq. 8.51 implies that $\boldsymbol{G}$ contains at least $(n-r)(n-r-1)$ zeros, this being the number of zeros in an $(n-r) \times (n-r)$ identity matrix. However, a large proportion of all the sparsity in these two matrices can be attributed to a single fact, which is the subject of this section: they are both block-diagonal.

To reveal the block-diagonal structure, we must first organize the kinematic loops into a set of *minimal loop clusters*. A loop cluster is any set of loops with the property that no loop within the cluster is coupled to any loop outside it; and a minimal loop cluster is one that cannot be divided into two smaller clusters. A coupling exists between two kinematic loops if they have a joint in common. Given a set of minimal loop clusters, it is possible to devise a regular numbering of the spanning tree with the following property: the tree joints within each cluster are numbered consecutively. It is this numbering scheme that reveals the block-diagonal structure of $\boldsymbol{K}$ and $\boldsymbol{G}$.

An example is presented in Figure 8.1. This figure shows the connectivity graph of a closed-loop system containing four kinematic loops. They are numbered 1 to 4, and are closed by loop joints 21 to 24, respectively. (Loop numbers

are not shown in the diagram.) Loop 1 is coupled to loop 2, but there is no other coupling in the system. We can therefore identify three minimal clusters as follows: cluster 1 contains loops 1 and 2; cluster 2 contains loop 3 only; and cluster 3 contains loop 4 only. Having identified these three clusters, we can devise a regular numbering in which the tree joints are numbered consecutively within each cluster. In this example, the joints in cluster 1 are numbered 3 to 7, those in cluster 2 are numbered 10 to 12, and those in cluster 3 are numbered 19 and 20. The remaining tree joints do not lie on the path of any kinematic loop, and therefore do not belong to any cluster. They can be numbered in any manner that is consistent with the rules of regular numbering.

With this numbering scheme, $\boldsymbol{K}$ adopts an irregular block-diagonal structure with gaps: there is one block per cluster; blocks are generally rectangular; each block can be a different size and shape to the others; and there can be horizontal gaps of varying sizes between the blocks. These gaps are caused by the tree joints that do not belong to any cluster. This pattern of sparsity in $\boldsymbol{K}$ allows $\boldsymbol{G}$ to adopt an irregular block-diagonal structure without gaps: each block $\boldsymbol{K}_i$ in $\boldsymbol{K}$ gives rise to a block $\boldsymbol{G}_i$ in $\boldsymbol{G}$; and each gap in $\boldsymbol{K}$ gives rise to an identity-matrix block in $\boldsymbol{G}$. For easy identification, the joints in the connectivity graph have been divided into six sets, labelled a to f, and the same division is shown against the columns of $\boldsymbol{K}$ and the rows of $\boldsymbol{G}$.

Block-diagonal sparsity can be exploited in the calculation of $\boldsymbol{KH}^{-1}\boldsymbol{K}^{\mathrm{T}}$ and $\boldsymbol{G}^{\mathrm{T}}\boldsymbol{HG}$, and in the calculation of $\boldsymbol{G}$ from $\boldsymbol{K}$. In particular, $\boldsymbol{G}$ can be calculated from $\boldsymbol{K}$ one block at a time: as $\boldsymbol{x} = \boldsymbol{G}_i\,\boldsymbol{y} + \boldsymbol{g}_i$ is the solution to $\boldsymbol{K}_i\,\boldsymbol{x} = \boldsymbol{k}_i$, each $\boldsymbol{G}_i$ and $\boldsymbol{g}_i$ can be calculated directly from $\boldsymbol{K}_i$ and $\boldsymbol{k}_i$ using the algorithm in Section 8.8.

8.10 Some Properties of Closed-Loop Systems

Closed-loop systems exhibit a greater variety of properties and behaviours than kinematic trees, which can pose new difficulties for dynamics algorithms. This section looks at some of the special properties of closed-loop systems that are relevant to dynamics.

Mobility: Mobility is the degree of motion freedom of a rigid-body system. The mobility of a kinematic tree is simply n; but the mobility of a closed-loop system is given by the general formula

$$mobility = n - r \,. \tag{8.52}$$

If $\boldsymbol{y}$ is a vector of independent coordinates for a rigid-body system, then the dimension of $\boldsymbol{y}$ is $n - r$. Systems with the property $r = n^c$ are said to be *properly constrained*. For such a system, the mobility can be expressed by the formula

$$mobility = n_{tot} - 6N_L \,, \tag{8.53}$$
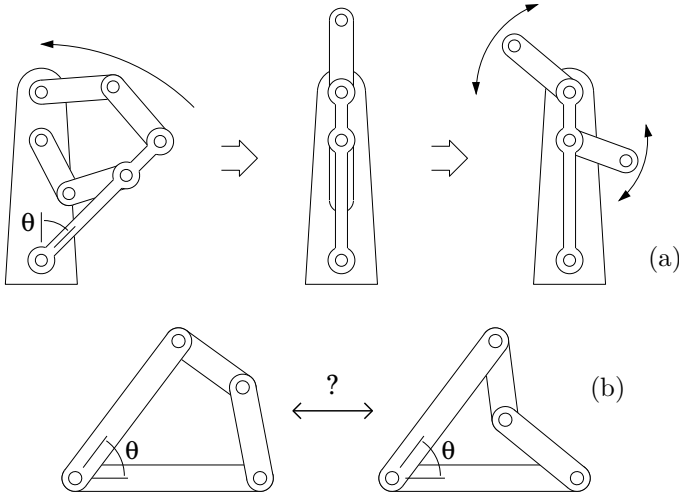
Figure 8.2: Examples of variable mobility (a) and configuration ambiguity (b)

where n_{tot} is the sum of the joint freedoms of all the joints in the system, not just the tree joints. This formula applies to rigid-body systems in a 3D Euclidean space. The equivalent formula in a 2D space is

$$mobility = n_{tot} - 3N_L \,. \tag{8.54}$$

Variable Mobility: The mobility of a kinematic tree is fixed; but the mobility of a closed-loop system depends on r, and r can vary with configuration. An example of variable mobility is shown in Figure 8.2(a). As long as the angle θ is nonzero, the mechanism has a mobility of 1; but if $\theta = 0$ then the two arms are able to move independently, and the mobility is 2. Furthermore, at the configuration shown in the middle diagram, which is the boundary between these two motion regimes, the mechanism has an instantaneous mobility of 3. In terms of n and r, we have $n = 5$ throughout, but $r = 4$ in the left-hand diagram, $r = 3$ in the right-hand diagram, and $r = 2$ in the middle diagram.

Configuration Ambiguity: If q and y are vectors of independent position variables for a kinematic tree and a closed-loop system, respectively, then q always identifies a unique configuration of the tree, but y does not necessarily identify a unique configuration of the closed-loop system. A simple example is shown in Figure 8.2(b): knowing the value of the independent variable, θ, is not enough to determine uniquely the configuration of the system. There are two ways to solve this problem:

1. use q instead of y, where q is the position vector of the spanning tree, or

2. supply a function that maps each value of y to a unique value of q.

The first option is the one used in previous sections, while the second is the subject of Section 8.11. For the system shown in Figure 8.2(b), option 2 is applicable if we know in advance that one of the two configurations cannot be reached. Otherwise, we must either try a different independent variable, or else resort to option 1.

Overconstraint: A closed-loop system is said to be overconstrained if $r < n^c$. Such systems have $n^c - r$ redundant constraints, which means there will be $n^c - r$ indeterminate constraint forces in the equations of motion. They also have *excess mobility* relative to a properly constrained system, as the mobility formula can be written

$$mobility = n_{tot} - 6N_L + (n^c - r). \tag{8.55}$$

Overconstrained systems are actually very common. For example, any system containing planar kinematic loops will be overconstrained. The planar loop shown in Figure 8.2(b) contains four revolute joints, so the numbers for this system are $n = 3$ and $n^c = 5$; but the mobility is 1, which means $r = 2$.

The dynamics of overconstrained systems is harder to calculate than that of properly constrained systems. For this reason, it is useful to convert the former to the latter wherever possible. The conversion is achieved by replacing the original loop joints with joints that impose fewer constraints, so that the value of n^c is reduced. For example, if the loop-closing revolute joint in a planar kinematic loop is replaced with a sphere-in-cylinder joint, which imposes only two constraints, then n^c will be reduced from 5 to 2, which means $n^c = r$.

8.11   Loop Closure Functions

Let $\boldsymbol{q}$ be the vector of position variables for the spanning tree of a given closed-loop system, and let $\boldsymbol{y}$ be a vector of independent position variables for the same system. $\boldsymbol{y}$ will typically be a subset of the elements of $\boldsymbol{q}$, but this is not necessary. We assume that $\boldsymbol{y}$ defines $\boldsymbol{q}$ uniquely. For this assumption to hold, it may be necessary to place restrictions on the value of $\boldsymbol{y}$. We therefore define a set $C \subseteq \mathsf{R}^{(n-r)}$ of acceptable values for $\boldsymbol{y}$, and assume $\boldsymbol{y} \in C$. Given these assumptions, there exists a function, $\boldsymbol{\gamma}$, that satisfies

$$\boldsymbol{q} = \boldsymbol{\gamma}(\boldsymbol{y}) \tag{8.56}$$

for all $\boldsymbol{y} \in C$. (Cf. Eq. 3.11.) Differentiating this equation gives

$$\dot{\boldsymbol{q}} = \boldsymbol{G}\,\dot{\boldsymbol{y}}, \tag{8.57}$$

where

$$\boldsymbol{G} = \frac{\partial \boldsymbol{\gamma}}{\partial \boldsymbol{y}}, \tag{8.58}$$

and differentiating it a second time gives

$$\ddot{\boldsymbol{q}} = \boldsymbol{G}\,\ddot{\boldsymbol{y}} + \boldsymbol{g}\,, \tag{8.59}$$

where

$$\boldsymbol{g} = \dot{\boldsymbol{G}}\,\dot{\boldsymbol{y}}\,. \tag{8.60}$$

Equation 8.59 has the same form as Eq. 8.42, and the two equations are identical when the loop-closure errors are zero.

   We now have a new method for obtaining $\boldsymbol{G}$ and $\boldsymbol{g}$, in which they are calculated from a vector of independent variables. This method can be summarized as follows.

   1. Define $\boldsymbol{y}$, and identify C.

   2. Find an expression for the function $\boldsymbol{\gamma}$ that maps $\boldsymbol{y}$ uniquely to $\boldsymbol{q}$, given $\boldsymbol{y} \in C$.

   3. Differentiate this expression symbolically to obtain expressions for $\boldsymbol{G}$ and $\boldsymbol{g}$.

   4. Write code to evaluate these expressions.

This method is less general that the one described in Section 8.5, mainly because step 2 is not always possible. It is also less convenient, since it requires the user to supply expressions (or code) for $\boldsymbol{G}$ and $\boldsymbol{g}$ in addition to the system model. Nevertheless, it is applicable to a wide variety of practical closed-loop systems, and the calculation of $\boldsymbol{G}$ and $\boldsymbol{g}$ by this method can be much cheaper than using the algorithms in Sections 8.7 and 8.8.

   Another advantage of this method is that loop-closure errors cannot occur. This is because we do not integrate $\ddot{\boldsymbol{q}}$ to get $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$, but instead integrate $\ddot{\boldsymbol{y}}$ to get $\dot{\boldsymbol{y}}$ and $\boldsymbol{y}$. This way, the vectors $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are calculated from $\boldsymbol{y}$ and $\dot{\boldsymbol{y}}$ using Eqs. 8.56 and 8.57, which means that their values automatically satisfy the loop constraints. There is therefore no need for constraint stabilization terms.

A Worked Example

Figure 8.3 shows a rigid-body system containing four bodies, five joints and one kinematic loop. The system is a planar mechanism, and its mobility is 2. Bodies B_1 and B_4 form a two-link arm, while B_2 and B_3 are the cylinder and piston, respectively, of a linear actuator. Joint 3 is prismatic, and the rest are revolute. Joints 1, 2, 3 and 5 participate in the kinematic loop, with joint 5 being the loop joint. The vector of tree-joint variables is therefore $\boldsymbol{q} = [q_1 \ q_2 \ q_3 \ q_4]^{\mathrm{T}}$, where q_3 is a distance and the rest are angles. In this example, we choose q_1 and q_4 to be the independent variables; that is, we define the vector of independent variables, $\boldsymbol{y} = [y_1 \ y_2]^{\mathrm{T}}$, such that $y_1 = q_1$ and $y_2 = q_4$. The main task is then to express q_2 and q_3 as functions of y_1.
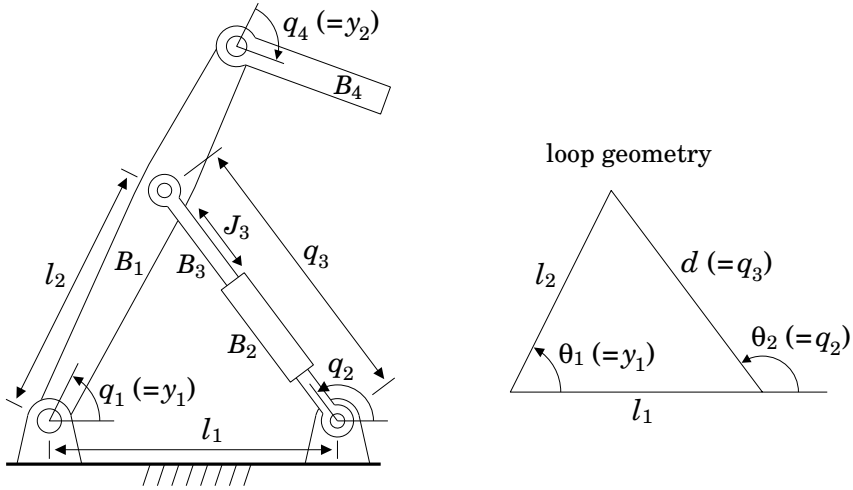
Triangle Formulae:

$$d^2 = l_1^2 + l_2^2 - 2l_1l_2\cos(\theta_1) \qquad d > 0$$

$$\sin(\theta_2) = \frac{l_2\sin(\theta_1)}{d}$$

$$\cos(\theta_2) = \frac{l_2\cos(\theta_1) - l_1}{d}$$

Solution:

$$\boldsymbol{\gamma}(\boldsymbol{y}) = \begin{bmatrix} y_1 \\ \theta_2 \\ d \\ y_2 \end{bmatrix} \qquad \text{where}$$

$$\theta_2 = \tan^{-1}\left(\frac{l_2\sin(y_1)}{l_2\cos(y_1) - l_1}\right)$$

$$d = \sqrt{l_1^2 + l_2^2 - 2l_1l_2\cos(y_1)}$$

$$\boldsymbol{G} = \frac{\partial\boldsymbol{\gamma}}{\partial\boldsymbol{y}} = \begin{bmatrix} 1 & 0 \\ u_1 & 0 \\ u_2 & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{where}$$

$$u_1 = \frac{l_2^2 - l_1l_2\cos(y_1)}{d^2}$$

$$u_2 = \frac{l_1l_2\sin(y_1)}{d}$$

$$\boldsymbol{g} = \dot{\boldsymbol{G}}\dot{\boldsymbol{y}} = \begin{bmatrix} 0 \\ u_3 \\ u_4 \\ 0 \end{bmatrix} \qquad \text{where}$$

$$u_3 = \frac{(l_1^2 - l_2^2)l_1l_2\sin(y_1)}{d^4}\dot{y}_1^2$$

$$u_4 = \frac{(l_1l_2\cos(y_1) - u_2^2)}{d}\dot{y}_1^2$$

Figure 8.3: Formulating $\boldsymbol{\gamma}$, $\boldsymbol{G}$ and $\boldsymbol{g}$ for a simple closed-loop system

The geometry of the kinematic loop is very simple: just a triangle. It is therefore possible to use standard trigonometric formulae to express q_2 and q_3 (labelled θ_2 and d on the triangle) as functions of y_1. To make the mapping unique, we impose the constraint $d > 0$, and we use the four-quadrant version of the arctangent function (the one that takes two arguments) to calculate θ_2. If $l_1 = l_2$ then we require $\theta_1 \neq 0$. In this case, C must exclude all vectors $\boldsymbol{y}$ having $y_1 = 0$. When a kinematic loop has an explicit solution formula, like this one, then it is said to have a *closed-form solution*.

Figure 8.3 also presents explicit formulae for $\boldsymbol{\gamma}$, $\boldsymbol{G}$ and $\boldsymbol{g}$. It can be seen that $\boldsymbol{G}$ is block-diagonal, having a 3×1 block and a 1×1 block. Note that the equations in this figure are all scalar equations, in contrast to the vector and matrix equations in Sections 8.7 and 8.8. It is therefore much cheaper to calculate $\boldsymbol{G}$ and $\boldsymbol{g}$ via the equations presented here.

In summary, this method places an additional burden on the user of having to supply code to calculate $\boldsymbol{\gamma}$, $\boldsymbol{G}$ and $\boldsymbol{g}$, but offers in return a potentially large improvement in the efficiency of this part of the dynamics calculation. This method is appropriate for systems containing kinematic loops with simple, closed-form solutions, such as might be found on excavator arms, large industrial robots, parallel robots and Stuart-Gough platforms (flight simulators).

8.12   Inverse Dynamics

Inverse dynamics is the problem of calculating the forces required to produce a given acceleration. In a kinematic tree, there is a $1 : 1$ relationship between $\boldsymbol{\tau}$ and $\ddot{\boldsymbol{q}}$, and the calculation of $\boldsymbol{\tau}$ from $\ddot{\boldsymbol{q}}$ is straightforward. In a closed-loop system, the given value of $\ddot{\boldsymbol{q}}$ must comply with the loop-closure constraints, and there are infinitely many values of $\boldsymbol{\tau}$ that produce the same acceleration. The nature of the inverse dynamics problem is therefore significantly altered by the presence of kinematic loops.

Let us now formulate the inverse dynamics equations for a closed-loop system. We start with the equation of motion for the spanning tree, which is

$$\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} - \boldsymbol{\tau}^a - \boldsymbol{K}^{\mathrm{T}}\boldsymbol{\lambda}. \tag{8.61}$$

In this equation, $\boldsymbol{\tau}$ is the vector of unknown forces acting at the tree joints. Any known forces, such as those due to springs and dampers, are assumed to be included in $\boldsymbol{C}$. It is also assumed that there are no unknown active forces at the loop joints. Thus, if $\boldsymbol{\tau}^a$ appears at all, then it is composed entirely of known forces.

Equation 8.61 contains two unknowns, $\boldsymbol{\tau}$ and $\boldsymbol{\lambda}$. As we are only interested in $\boldsymbol{\tau}$, the next step is to eliminate $\boldsymbol{\lambda}$. This is done by multiplying the equation by $\boldsymbol{G}^{\mathrm{T}}$, which gives

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau} = \boldsymbol{G}^{\mathrm{T}}(\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} - \boldsymbol{\tau}^a).$$

This equation can be written

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau} = \boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}_{\mathrm{ID}}\,, \tag{8.62}$$

where $\boldsymbol{\tau}_{\mathrm{ID}}$ is the force calculated by applying an inverse dynamics function to the spanning tree; that is,

$$\begin{aligned} \boldsymbol{\tau}_{\mathrm{ID}} &= \boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} - \boldsymbol{\tau}^a \\ &= \mathrm{ID}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\,. \end{aligned}$$

In this context, $\mathrm{ID}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ is the output of the algorithm described in Section 8.6 with the joint acceleration terms put back in. Bear in mind that $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ must all comply with the loop-closure constraints.

Equation 8.62 is the basic inverse dynamics equation for a closed-loop system. As $\boldsymbol{G}^{\mathrm{T}}$ is an $(n-r) \times n$ matrix, this equation imposes $n-r$ constraints on an n-dimensional vector of unknowns, leaving r freedoms of choice. In other words, there are ∞^r different values of $\boldsymbol{\tau}$ that produce the same acceleration. If we want a unique solution to the problem, then we must either impose more constraints or apply an optimality criterion.

Actuator Forces

A typical use for inverse dynamics is to calculate a vector of actuator forces to produce a given acceleration. Now, not every joint will be powered by an actuator, so let us define p and $\boldsymbol{u}$ to be the number of actuated freedoms in the tree, and the p-dimensional vector of actuator force values, respectively. $\boldsymbol{u}$ and $\boldsymbol{\tau}$ are then related by the equation

$$\boldsymbol{\tau} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{u}\\\boldsymbol{0}\end{bmatrix},$$

where $\boldsymbol{Q}$ is an $n \times n$ permutation matrix. This equation simply says that the elements of $\boldsymbol{\tau}$ are a permutation of the p elements of $\boldsymbol{u}$ and $n-p$ zeros. The zero-valued elements of $\boldsymbol{\tau}$ are those that belong to the unactuated degrees of freedom. Let us also define the two matrices $\boldsymbol{G}_u$ and $\boldsymbol{G}_0$, which contain the rows of $\boldsymbol{G}$ corresponding to actuated and unactuated freedoms, respectively. These matrices are related to $\boldsymbol{G}$ by the equation

$$\boldsymbol{G} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{G}_u\\\boldsymbol{G}_0\end{bmatrix}.$$

From these definitions, it follows that

$$\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau} = \begin{bmatrix}\boldsymbol{G}_u^{\mathrm{T}} & \boldsymbol{G}_0^{\mathrm{T}}\end{bmatrix}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q}\begin{bmatrix}\boldsymbol{u}\\\boldsymbol{0}\end{bmatrix} = \begin{bmatrix}\boldsymbol{G}_u^{\mathrm{T}} & \boldsymbol{G}_0^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\boldsymbol{u}\\\boldsymbol{0}\end{bmatrix} = \boldsymbol{G}_u^{\mathrm{T}}\boldsymbol{u}\,;$$

and substituting this result into Eq. 8.62 gives

$$\boldsymbol{G}_u^{\mathrm{T}}\boldsymbol{u} = \boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}_{\mathrm{ID}}\,, \tag{8.63}$$

which is the inverse dynamics equation for actuator forces. In solving this equation, there are three cases to consider.

1. $\mathrm{rank}(\boldsymbol{G}_u) < n - r$. In this case, the system is *underactuated*: it has more motion freedoms than the actuators can control.

2. $p > \mathrm{rank}(\boldsymbol{G}_u)$. In this case, the system is *redundantly actuated*: infinitely many different values of $\boldsymbol{u}$ will produce the same acceleration.

3. $p = \mathrm{rank}(\boldsymbol{G}_u) = n - r$. In this case, the system is *properly actuated*: $\boldsymbol{G}_u$ is invertible, and there is a unique solution to Eq. 8.63 given by $\boldsymbol{u} = \boldsymbol{G}_u^{-\mathrm{T}}\boldsymbol{G}^{\mathrm{T}}\boldsymbol{\tau}_{\mathrm{ID}}$.

Cases 1 and 2 are not mutually exclusive.

Example 8.2 *Consider the assignment of actuators to the mechanism shown in Figure 8.3. This mechanism has two degrees of freedom, and therefore requires two actuators. $\boldsymbol{G}_u$ will therefore be a 2×2 matrix containing two rows of $\boldsymbol{G}$; and these rows must be chosen so that $\boldsymbol{G}_u$ is invertible. Given the formula for $\boldsymbol{G}$ shown in the figure, it follows that $\boldsymbol{G}_u$ must contain row 4 of $\boldsymbol{G}$ and any one of the other three rows. This means that an actuator must be present at joint 4, and at any one of the other three joints. (As this mechanism is so simple, this result could have been obtained by inspection.)*

Let us suppose that joints 3 and 4 are actuated, which means that $\boldsymbol{G}_u = \begin{bmatrix} u_2 & 0 \\ 0 & 1 \end{bmatrix}$. The inverse dynamics formula for this system will then be

$$
\begin{aligned}
\boldsymbol{u} &= \boldsymbol{G}_u^{-\mathrm{T}}\,\boldsymbol{G}^{\mathrm{T}}\,\boldsymbol{\tau}_{\mathrm{ID}} \\
&= \begin{bmatrix} u_2 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & u_1 & u_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\tau}_{\mathrm{ID}} \\
&= \begin{bmatrix} 1/u_2 & u_1/u_2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\tau}_{\mathrm{ID}}\,,
\end{aligned}
$$

where

$$
\boldsymbol{\tau}_{\mathrm{ID}} = \mathrm{ID}(\boldsymbol{\gamma}(\boldsymbol{y}), \boldsymbol{G}\dot{\boldsymbol{y}}, \boldsymbol{G}\ddot{\boldsymbol{y}} + \boldsymbol{g})\,.
$$

Note that $\boldsymbol{u}$ is not a vector of generalized forces for this system, since the expression $\boldsymbol{u}^{\mathrm{T}}\dot{\boldsymbol{y}}$ does not equal the power delivered by $\boldsymbol{u}$. This discrepancy is caused by the fact that $\dot{q}_1$ is the first element in $\dot{\boldsymbol{y}}$, but τ_3 is the first element in $\boldsymbol{u}$. If you want $\boldsymbol{u}$ to be the generalized force vector, then the elements of $\boldsymbol{u}$ must be the same subset of $\boldsymbol{\tau}$ as $\dot{\boldsymbol{y}}$ is of $\dot{\boldsymbol{q}}$.

8.13   Sparse Matrix Method

Every algorithm we have studied so far works by applying a set of loop-closure constraints to a spanning tree. An alternative strategy is to dispense with the spanning tree, and work directly with the individual bodies and joints in the

system. This approach produces equations with large, sparse matrices; and any algorithm based on these equations must exploit the sparsity in order to be competitive. Such algorithms tend to be less efficient than those based on spanning trees, but they can be more efficient under special circumstances, as we shall see below.

The equation of motion for a collection of N_B rigid bodies, subject to the motion constraints imposed by a set of N_J joints, is given by

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{P}\,\boldsymbol{T} \\ \boldsymbol{T}^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}\,\boldsymbol{T}^a\boldsymbol{\tau} - \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v} + \boldsymbol{f}^x \\ -\dot{\boldsymbol{T}}^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}}\boldsymbol{v} \end{bmatrix}. \tag{8.64}$$

This equation is derived in Section 3.7. $\boldsymbol{v}$, $\boldsymbol{a}$ and $\boldsymbol{f}^x$ are vectors of body velocities, accelerations and external forces, respectively; $\boldsymbol{\lambda}$ is the vector of joint constraint-force variables; $\boldsymbol{T}^a\boldsymbol{\tau}$ is the vector of active forces transmitted across the joints; $\boldsymbol{P}^{\mathrm{T}}$ is the matrix that maps from body to joint velocities (i.e., $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{v}$ is the vector of joint velocities); and $\boldsymbol{I}$ and $\boldsymbol{T}$ are composite matrices of body inertias and joint constraint-force subspaces, respectively. $\boldsymbol{a}$ and $\boldsymbol{\lambda}$ are the unknowns; and the size of the coefficient matrix is $6N_B + n_{tot}^c$ square, where n_{tot}^c is the total number of constraints imposed by the joints.

Our chief interest lies in the sparsity pattern of the coefficient matrix, so let us examine its constituents. $\boldsymbol{I}$ and $\boldsymbol{T}$ are both block-diagonal, being defined by

$$\boldsymbol{I} = \mathrm{diag}(\boldsymbol{I}_1, \ldots, \boldsymbol{I}_{N_B}) \qquad \text{and} \qquad \boldsymbol{T} = \mathrm{diag}(\boldsymbol{T}_1, \ldots, \boldsymbol{T}_{N_J}).$$

$\boldsymbol{P}$ is also sparse, but not block-diagonal. It is defined by

$$\boldsymbol{P}_{ij} = \begin{cases} \boldsymbol{1}_{6 \times 6} & \text{if } i = s(j) \\ -\boldsymbol{1}_{6 \times 6} & \text{if } i = p(j) \\ \boldsymbol{0}_{6 \times 6} & \text{otherwise.} \end{cases}$$

$\boldsymbol{P}$ is therefore a $6N_B \times 6N_J$ matrix, organized as an $N_B \times N_J$ matrix of 6×6 blocks. It has two nonzero blocks in every column, except those columns for joints connecting directly to the base, which have only one nonzero block. From these definitions, it follows that the coefficient matrix contains N_B nonzero blocks on the main diagonal, and up to $4N_J$ nonzero off-diagonal blocks. The exact positions of the latter are determined by the connectivity.

Equation 8.64 is typical of the kind of equation that a sparse matrix arithmetic software library is designed to solve, and the use of such a library is probably the best way to proceed. A sparse matrix library would typically provide the following facilities:

- a function to analyse the sparsity pattern of a matrix, as a preprocessing step, in order to work out the best factorization order and which zero-valued elements get filled in with nonzero values during the course of the factorization;

- a data structure for compact storage of all the nonzero values in the matrix and all the zero-valued entries that will get altered during factorization;

- a function to perform a factorization in the order determined in the pre-processing step; and

- a function to perform back-substitution using the resulting factors.

If an $n \times n$ matrix contains only $O(n)$ nonzero elements, then it is sometimes possible to factorize the matrix in only $O(n)$ arithmetic operations. An $O(n)$ factorization of Eq. 8.64 is always possible if the system is a kinematic tree.[2] However, there are also many closed-loop systems for which an $O(n)$ factorization is possible, including some that cannot be solved in $O(n)$ by any algorithm using a spanning tree. In such cases, the sparse-matrix algorithm will out-perform spanning-tree algorithms for a sufficiently large n.

Figure 8.4 shows an example of a closed-loop system having $O(n)$ dynamics by the sparse matrix method. The connectivity graph is of a kind that is sometimes called a ladder. It can be extended indefinitely by adding more nodes to the right. The figure also shows the sparsity pattern of a symmetric permutation of the coefficient matrix. This permutation has been chosen so that the resulting matrix has a fixed bandwidth—every nonzero block appears on the main diagonal, or on one of the three nearest diagonals either side. A matrix with this property can always be factorized in $O(n)$ operations.

To explain the permutation, the nodes in the graph have been numbered in the same order as they appear in the matrix, and the joints have been given number pairs reflecting their connectivity: joint ij is the one that connects from body i to body j. Row 1 in the permuted matrix is the row in the original matrix that contained $\boldsymbol{I}_1$ (with body numbers as shown in the graph); row 2 is the row that contained $\boldsymbol{I}_2$; row 3, which is labelled 12, is the row in the original matrix containing the row in $\boldsymbol{T}^{\mathrm{T}}\boldsymbol{P}^{\mathrm{T}}$ pertaining to joint 12; and so on. Observe that each row labelled with a body number contains a nonzero block on the main diagonal, and that each row labelled with a number pair ij contains nonzero blocks only in columns i and j.

If we were to apply a spanning-tree algorithm to this system, then we would encounter the following problem: whatever spanning tree we choose, there will always be $O(n^2)$ nonzero elements in the joint-space inertia matrix, $\boldsymbol{H}$, and so it will be impossible to calculate the dynamics in only $O(n)$ operations.

Finally, the following details should be mentioned. First, if this algorithm is to be used in a dynamics simulator, then constraint stabilization is required. The necessary extra terms have not been included in Eq. 8.64. Second, we have assumed that the coefficient matrix has full rank; that is, we have assumed that the closed-loop system is properly constrained. If the matrix is singular, then the sparse factorization may not work.

---

[2]This is the basis of Baraff's algorithm (Baraff, 1996).
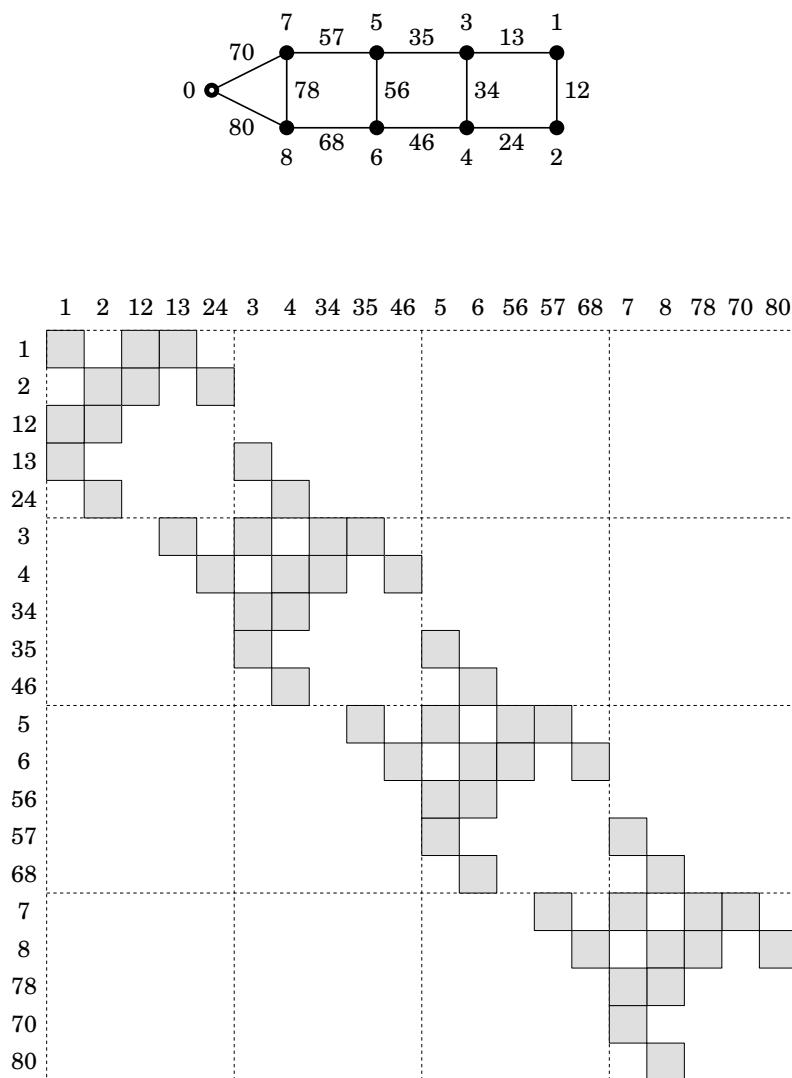
Figure 8.4: Sparsity pattern arising from a ladder-like connectivity graph

Chapter 9

Hybrid Dynamics and Other Topics

This chapter covers the following topics: hybrid dynamics, floating bases, gears and dynamic equivalence. Hybrid dynamics is a generalization of forward and inverse dynamics in which the forces are known at some joints, the accelerations at the rest, and the task is to calculate the unknown forces and accelerations. In effect, one is performing forward dynamics at some joints and inverse dynamics at the rest. A floating-base system is one in which the base is a moving body. Such systems can be modelled as fixed-base systems by installing a 6-DoF joint between the fixed base and the body representing the floating base. However, floating-base systems are an important class of rigid-body systems, and merit special treatment. Sections 9.1 to 9.5 present several algorithms for hybrid dynamics and floating bases. Gears impose constraints between joint variables. They therefore resemble kinematic loops, and Section 9.6 shows how gear constraints can be incorporated into a rigid-body system using a technique that was developed in Chapter 8 for kinematic loops. Two different rigid-body systems are dynamically equivalent if they have the same equation of motion. Section 9.7 explores this concept, and presents a method for modifying the inertia parameters of a rigid-body system without altering its equation of motion.

9.1 Hybrid Dynamics

For each joint i, there is a force variable, $\boldsymbol{\tau}_i$, and an acceleration variable, $\ddot{\boldsymbol{q}}_i$. In the forward-dynamics problem, we are given $\boldsymbol{\tau}_i$ for every joint, and the task is to calculate the accelerations. In the inverse-dynamics problem, we are given $\ddot{\boldsymbol{q}}_i$ for every joint, and the task is to calculate the forces. In the hybrid-dynamics problem, we are given either $\boldsymbol{\tau}_i$ or $\ddot{\boldsymbol{q}}_i$ at each joint, and the task is to calculate the unknown accelerations and forces. In effect, hybrid dynamics is like performing forward dynamics at some of the joints and inverse dynamics

at the rest. The hybrid-dynamics problem therefore includes both the forward-dynamics problem and the inverse-dynamics problem as special cases.

Hybrid dynamics is used to introduce prescribed motions into a rigid-body system. For example, it will let you study the dynamics of a mechanical linkage or transmission under prescribed motion conditions at the input, or the motion of a compound pendulum in response to a given kinematic excitation, or the tumbling behaviour of an animated character in mid air. It can also be used to help design trajectories for underactuated systems; for example, a sequence of limb movements for an animated character to leap, perform a somersault, and land on its feet.

Another use is to simplify a simulation by removing unimportant and/or high-frequency dynamics. For example, the motions of a robot's joints are typically controlled by high-gain feedback control systems. These control systems continually compare the actual motion of a joint with the commanded motion, and adjust the force output from the actuator so as to bring the difference quickly to zero. In a simulation of a complete robot, the simulator must calculate the dynamics of the actuators and control systems as well as the mechanical parts. However, if a particular control system can be regarded as perfect, meaning that the actual motions of the joints it controls are practically equal to the commanded motions, then it is possible to dispense with this control system and its actuators, and simply prescribe the motions of the affected joints. If the removed components were responsible for the highest-frequency dynamics in the original system, then the simulator can now use a longer integration time step. An example of this technique can be found in Hu et al. (2005).

Equations

Let us define fd to be the set of 'forward-dynamics joints'; that is, the set of joints for which the forces are known and the accelerations unknown. Any joint that is not a member of fd is therefore an 'inverse-dynamics joint', for which the acceleration is known and the force unknown. Let us also define $\boldsymbol{Q}$ to be the permutation matrix that reorders the elements of $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ so that the forward-dynamics variables come first. Thus,

$$\boldsymbol{Q}\,\ddot{\boldsymbol{q}} = \begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \ddot{\boldsymbol{q}}_2 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{Q}\,\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{\tau}_1 \\ \boldsymbol{\tau}_2 \end{bmatrix},$$

where $\ddot{\boldsymbol{q}}_1$ and $\boldsymbol{\tau}_1$ contain the acceleration and force variables pertaining to the forward-dynamics joints, and $\ddot{\boldsymbol{q}}_2$ and $\boldsymbol{\tau}_2$ contain the acceleration and force variables pertaining to the inverse-dynamics joints. The equation of motion for a kinematic tree can now be written

$$\begin{bmatrix} \boldsymbol{H}_{11} & \boldsymbol{H}_{12} \\ \boldsymbol{H}_{21} & \boldsymbol{H}_{22} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \ddot{\boldsymbol{q}}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_1 \\ \boldsymbol{\tau}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \end{bmatrix}, \tag{9.1}$$

where

$$\begin{bmatrix} \boldsymbol{H}_{11} & \boldsymbol{H}_{12} \\ \boldsymbol{H}_{21} & \boldsymbol{H}_{22} \end{bmatrix} = \boldsymbol{Q}\,\boldsymbol{H}\,\boldsymbol{Q}^{\mathrm{T}} \qquad \text{and} \qquad \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \end{bmatrix} = \boldsymbol{Q}\,\boldsymbol{C}\,.$$

Equation 9.1 is just a permutation of the standard equation of motion, $\boldsymbol{H}\ddot{\boldsymbol{q}} = \boldsymbol{\tau} - \boldsymbol{C}$ (e.g. see Eq. 6.1). The two unknowns in this equation are $\ddot{\boldsymbol{q}}_1$ and $\boldsymbol{\tau}_2$. If we bring them together on the left-hand side, then we get

$$\begin{bmatrix} \boldsymbol{H}_{11} & \mathbf{0} \\ \boldsymbol{H}_{21} & -\mathbf{1} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \boldsymbol{\tau}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_1 \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{C}'_1 \\ \boldsymbol{C}'_2 \end{bmatrix}, \tag{9.2}$$

where

$$\begin{bmatrix} \boldsymbol{C}'_1 \\ \boldsymbol{C}'_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_1 + \boldsymbol{H}_{12}\ddot{\boldsymbol{q}}_2 \\ \boldsymbol{C}_2 + \boldsymbol{H}_{22}\ddot{\boldsymbol{q}}_2 \end{bmatrix} = \boldsymbol{Q}\,\boldsymbol{C}'.$$

This is the basic equation for hybrid dynamics. Physically, $\boldsymbol{C}'$ is the force required to impart zero acceleration to each forward-dynamics joint and the given acceleration to each inverse-dynamics joint. Thus, $\boldsymbol{C}'$ can be calculated by an inverse-dynamics function as follows:

$$\boldsymbol{C}' = \mathrm{ID}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{Q}^{\mathrm{T}}\begin{bmatrix} \mathbf{0} \\ \ddot{\boldsymbol{q}}_2 \end{bmatrix}). \tag{9.3}$$

Example 9.1 *Equation 9.2 hides $\ddot{\boldsymbol{q}}_2$ within $\boldsymbol{C}'$. If we want to keep $\ddot{\boldsymbol{q}}_2$ visible, then the equation can be written*

$$\begin{bmatrix} \boldsymbol{H}_{11} & \mathbf{0} \\ \boldsymbol{H}_{21} & -\mathbf{1} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \boldsymbol{\tau}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{1} & -\boldsymbol{H}_{12} \\ \mathbf{0} & -\boldsymbol{H}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{\tau}_1 \\ \ddot{\boldsymbol{q}}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \end{bmatrix}. \tag{9.4}$$

Multiplying this equation by the inverse of the left-hand coefficient matrix produces Eq. 10 in Jain and Rodriguez (1993).

Algorithms

Equation 9.2 can be solved in four steps as follows.

1. Calculate $\boldsymbol{C}'$ using Eq. 9.3.

2. Calculate $\boldsymbol{H}_{11}$ as described below.

3. Solve $\boldsymbol{H}_{11}\ddot{\boldsymbol{q}}_1 = \boldsymbol{\tau}_1 - \boldsymbol{C}'_1$ for $\ddot{\boldsymbol{q}}_1$.

4. Calculate $\boldsymbol{\tau}_2$ via $\boldsymbol{\tau} = \boldsymbol{C}' + \mathrm{ID}_\delta(\boldsymbol{Q}^{\mathrm{T}}\begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \mathbf{0} \end{bmatrix})$.

The symbol ID_δ in step 4 denotes the differential inverse-dynamics function described in Table 6.1, which multiplies its argument by $\boldsymbol{H}$. Thus, the equation in step 4 evaluates to

$$\boldsymbol{\tau} = \boldsymbol{C}' + \boldsymbol{Q}^{\mathrm{T}}\begin{bmatrix} \boldsymbol{H}_{11}\ddot{\boldsymbol{q}}_1 \\ \boldsymbol{H}_{21}\ddot{\boldsymbol{q}}_1 \end{bmatrix}.$$

The simplest way to accomplish step 2 is to calculate $\boldsymbol{H}$ via the composite rigid body algorithm (§6.2) and extract the submatrix $\boldsymbol{H}_{11}$. However, if greater

$\boldsymbol{H} = \boldsymbol{0}$
for each i **in** $\nu(\mathit{fd})$ **do**
    $\boldsymbol{I}_i^{\mathrm{c}} = \boldsymbol{I}_i$
end
for $i = N_B$ **to** 1 **do**
    if $\lambda(i) \in \nu(\mathit{fd})$ **then**
        $\boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} = \boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} + {}^{\lambda(i)}\boldsymbol{X}_i^* \boldsymbol{I}_i^{\mathrm{c}}\, {}^i\boldsymbol{X}_{\lambda(i)}$
    end
    if $i \in \mathit{fd}$ **then**
        $\boldsymbol{F} = \boldsymbol{I}_i^{\mathrm{c}}\boldsymbol{S}_i$
        $\boldsymbol{H}_{ii} = \boldsymbol{S}_i^{\mathrm{T}}\boldsymbol{F}$

$\vdots$

        $j = i$
        while $\lambda(j) \in \nu(\mathit{fd})$ **do**
            $\boldsymbol{F} = {}^{\lambda(j)}\boldsymbol{X}_j^* \boldsymbol{F}$
            $j = \lambda(j)$
            if $j \in \mathit{fd}$ **then**
                $\boldsymbol{H}_{ij} = \boldsymbol{F}^{\mathrm{T}}\boldsymbol{S}_j$
                $\boldsymbol{H}_{ji} = \boldsymbol{H}_{ij}^{\mathrm{T}}$
            end
        end
    end
end

Table 9.1: Modified composite rigid body algorithm for calculating $\boldsymbol{H}_{11}$

efficiency is required, then it is possible to modify the algorithm so that it only performs the minimum amount of calculation necessary to obtain $\boldsymbol{H}_{11}$. This modified algorithm is shown in Table 9.1. One new quantity appears in this algorithm: $\nu(\mathit{fd})$, which is the set of bodies that are supported by at least one forward-dynamics joint. It is defined by the formula

$$\nu(\mathit{fd}) = \bigcup_{i \in \mathit{fd}} \nu(i)\,, \tag{9.5}$$

and it can be calculated from fd and λ as follows:[1]

$\nu(\mathit{fd}) = \mathit{fd}$
for $i = 1$ **to** N_B **do**
    if $\lambda(i) \in \nu(\mathit{fd})$ **then**
        $\nu(\mathit{fd}) = \nu(\mathit{fd}) \cup \{i\}$
    end
end

The cost of calculating $\nu(\mathit{fd})$ is small enough that it would be practical for a hybrid dynamics routine to calculate it each time the routine was called. However, as $\nu(\mathit{fd})$ depends only on fd and λ, its value would not normally change during the course of a single simulation run; so it would also be possible to calculate it just once at the beginning of the run, and store it for later use.

On comparing the modified algorithm with the original, in Table 6.2, it can be seen that they both have the same structure, but the modified algorithm performs only a subset of the calculations. For example, $\boldsymbol{I}_i^{\mathrm{c}}$ is calculated only

---

[1]In this code fragment, '$\nu(\mathit{fd})$' should be regarded as the name of a variable. In a practical implementation, $\nu(\mathit{fd})$ could be represented by an array of $N_B + 1$ Boolean values, the array being indexed from 0 to N_B, such that each element is either true or false according to whether its index is or is not an element of $\nu(\mathit{fd})$.

original graph   modified graph $\lambda' = [0,0,0,1,1]$
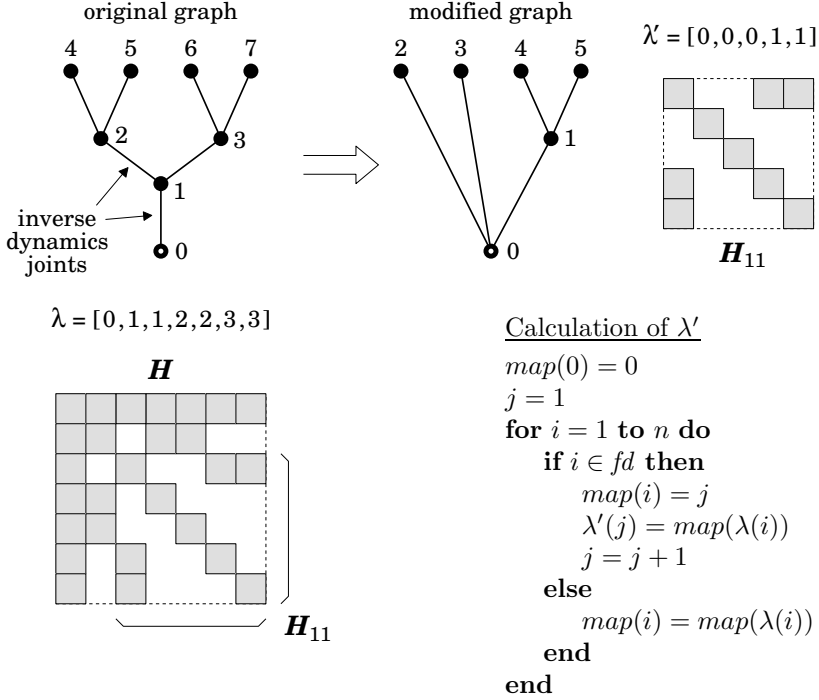
Figure 9.1: Branch-induced sparsity in $\boldsymbol{H}_{11}$

Calculation of λ'
$map(0) = 0$
$j = 1$
for $i = 1$ **to** n **do**
  if $i \in \mathit{fd}$ **then**
    $map(i) = j$
    $\lambda'(j) = map(\lambda(i))$
    $j = j + 1$
  else
    $map(i) = map(\lambda(i))$
  end
end

for those values of i satisfying $i \in \nu(\mathit{fd})$, and $\boldsymbol{H}_{ij}$ is calculated only for those values of i and j satisfying $i, j \in \mathit{fd}$.

Moving on to step 3, $\boldsymbol{H}_{11}$ is a symmetric, positive-definite matrix, so the equation in step 3 could be solved using any standard linear-equation solver for such matrices. However, as shown in Figure 9.1, $\boldsymbol{H}_{11}$ inherits a portion of whatever branch-induced sparsity is present in $\boldsymbol{H}$. We therefore have the opportunity to exploit this sparsity by using one of the algorithms described in Section 6.5.

The prerequisite for exploiting the sparsity in $\boldsymbol{H}_{11}$ is to obtain a modified parent array, λ', that describes the sparsity pattern in this matrix. Figure 9.1 shows how this can be done. Starting with the connectivity graph of the original system, a modified graph is constructed by merging every pair of nodes that are connected by an inverse-dynamics joint. Once this is done, the remaining nodes must be renumbered, as there are now fewer of them. In the example shown in the figure, joints 1 and 2 are inverse-dynamics joints, so nodes 1 and 2 are merged with node 0, and nodes 3 to 7 are renumbered 1 to 5.

Figure 9.1 also shows an algorithm for calculating λ' from λ and fd. As a by-product, this algorithm also calculates the node-number mapping from the original to the modified graph: $map(i)$ is the new number for original node i.

To follow this algorithm, it helps to know that i is counting node numbers in the original graph, while j is counting node numbers in the modified graph. Bear in mind that if any of the joints in the modified graph have more than one degree of freedom then λ' will have to be expanded, as explained in Section 6.5, before it can be used.

9.2   Articulated-Body Hybrid Dynamics

The articulated-body algorithm is easily adapted to perform hybrid dynamics: one simply modifies the algorithm so that it uses the normal articulated-body equations (as described in §7.3) at each forward-dynamics joint, and the equations developed below at each inverse-dynamics joint. The two sets of equations are shown side-by-side in Table 9.2, and the pseudocode is shown in Table 9.3. The latter is adapted from the code in Table 7.1.

Let $\boldsymbol{f}_i$ be the spatial force transmitted across joint i. $\boldsymbol{f}_i$ is related to the acceleration of body i by the equation

$$\boldsymbol{f}_i = \boldsymbol{I}_i^A \boldsymbol{a}_i + \boldsymbol{p}_i^A \, ; \tag{9.6}$$

and it is related to the acceleration of body $\lambda(i)$ by the equation

$$\boldsymbol{f}_i = \boldsymbol{I}_i^a \boldsymbol{a}_{\lambda(i)} + \boldsymbol{p}_i^a \, . \tag{9.7}$$

(See Eq. 7.25.) Furthermore, $\boldsymbol{a}_{\lambda(i)}$ is related to $\boldsymbol{a}_i$ by

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i \, . \tag{9.8}$$

In the articulated-body algorithm, the main step is the calculation of $\boldsymbol{I}_i^a$ and $\boldsymbol{p}_i^a$ from $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$. If joint i is a forward-dynamics joint, then the immediate problem is that $\ddot{\boldsymbol{q}}_i$ is unknown, so the first step is to find an expression for $\ddot{\boldsymbol{q}}_i$. However, no such difficulty exists for an inverse-dynamics joint. Substituting Eq. 9.8 into Eq. 9.6 gives

$$\boldsymbol{f}_i = \boldsymbol{I}_i^A (\boldsymbol{a}_{\lambda(i)} + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i) + \boldsymbol{p}_i^A \, ; $$

and comparing this equation with Eq. 9.7 reveals that

$$\boldsymbol{I}_i^a = \boldsymbol{I}_i^A \tag{9.9}$$

and

$$\boldsymbol{p}_i^a = \boldsymbol{p}_i^A + \boldsymbol{I}_i^A (\dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i) \, . \tag{9.10}$$

These two equations take the place of Eqs. 7.39 and 7.40 in Pass 2. Apart from this, only two other changes can be seen in the inverse-dynamics equations in Table 9.2: the term $\boldsymbol{S}_i \ddot{\boldsymbol{q}}_i$ has been added to $\boldsymbol{c}_i$ (which means that $\boldsymbol{a}_i'$ no longer appears in Pass 3), and a new equation has been added to Pass 3 to calculate $\boldsymbol{\tau}_i$ via $\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{f}_i$ and Eq. 9.6. (We could have used Eq. 9.7 instead.)

Forward Dynamics ($i \in fd$):	Inverse Dynamics ($i \notin fd$):
<u>Pass 1</u>	<u>Pass 1</u>
$\boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i$	$\boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i$
$\boldsymbol{v}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i} \qquad (\boldsymbol{v}_0 = \boldsymbol{0})$	$\boldsymbol{v}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i} \qquad (\boldsymbol{v}_0 = \boldsymbol{0})$
$\boldsymbol{c}_{\mathrm{J}i} = \mathring{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i$	$\boldsymbol{c}_{\mathrm{J}i} = \mathring{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i$
$\boldsymbol{c}_i = \boldsymbol{c}_{\mathrm{J}i} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}i}$	$\boldsymbol{c}_i = \boldsymbol{c}_{\mathrm{J}i} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}i} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$
$\boldsymbol{p}_i = \boldsymbol{v}_i \times^{*} \boldsymbol{I}_i\,\boldsymbol{v}_i - {}^{i}\boldsymbol{X}_0^{*}\,\boldsymbol{f}_i^{x}$	$\boldsymbol{p}_i = \boldsymbol{v}_i \times^{*} \boldsymbol{I}_i\,\boldsymbol{v}_i - {}^{i}\boldsymbol{X}_0^{*}\,\boldsymbol{f}_i^{x}$
<u>Pass 2</u>	<u>Pass 2</u>
$\boldsymbol{I}_i^{A} = \boldsymbol{I}_i + \displaystyle\sum_{j\in\mu(i)} {}^{i}\boldsymbol{X}_j^{*}\,\boldsymbol{I}_j^{a}\,{}^{j}\boldsymbol{X}_i$	$\boldsymbol{I}_i^{A} = \boldsymbol{I}_i + \displaystyle\sum_{j\in\mu(i)} {}^{i}\boldsymbol{X}_j^{*}\,\boldsymbol{I}_j^{a}\,{}^{j}\boldsymbol{X}_i$
$\boldsymbol{p}_i^{A} = \boldsymbol{p}_i + \displaystyle\sum_{j\in\mu(i)} {}^{i}\boldsymbol{X}_j^{*}\,\boldsymbol{p}_j^{a}$	$\boldsymbol{p}_i^{A} = \boldsymbol{p}_i + \displaystyle\sum_{j\in\mu(i)} {}^{i}\boldsymbol{X}_j^{*}\,\boldsymbol{p}_j^{a}$
$\boldsymbol{U}_i = \boldsymbol{I}_i^{A}\,\boldsymbol{S}_i$	$\boldsymbol{I}_i^{a} = \boldsymbol{I}_i^{A}$
$\boldsymbol{D}_i = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{U}_i$	$\boldsymbol{p}_i^{a} = \boldsymbol{p}_i^{A} + \boldsymbol{I}_i^{a}\boldsymbol{c}_i$
$\boldsymbol{u}_i = \boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}}\boldsymbol{p}_i^{A}$	
$\boldsymbol{I}_i^{a} = \boldsymbol{I}_i^{A} - \boldsymbol{U}_i\,\boldsymbol{D}_i^{-1}\,\boldsymbol{U}_i^{\mathrm{T}}$	
$\boldsymbol{p}_i^{a} = \boldsymbol{p}_i^{A} + \boldsymbol{I}_i^{a}\boldsymbol{c}_i + \boldsymbol{U}_i\,\boldsymbol{D}_i^{-1}\boldsymbol{u}_i$	
<u>Pass 3</u>	<u>Pass 3</u>
$\boldsymbol{a}_i' = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i \quad (\boldsymbol{a}_0 = -\boldsymbol{a}_g)$	$\boldsymbol{a}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i \quad (\boldsymbol{a}_0 = -\boldsymbol{a}_g)$
$\ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i^{-1}\,(\boldsymbol{u}_i - \boldsymbol{U}_i^{\mathrm{T}}\boldsymbol{a}_i')$	$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}\,(\boldsymbol{I}_i^{A}\boldsymbol{a}_i + \boldsymbol{p}_i^{A})$
$\boldsymbol{a}_i = \boldsymbol{a}_i' + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$	

Table 9.2: Articulated-body hybrid dynamics equations

$\boldsymbol{v}_0 = \boldsymbol{0}$
for $i = 1$ **to** N_B **do**
    $[\boldsymbol{X}_{\mathrm{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\mathrm{J}}, \boldsymbol{c}_{\mathrm{J}}] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
    ${}^{i}\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}}\, \boldsymbol{X}_{\mathrm{T}}(i)$
    if $\lambda(i) \neq 0$ **then**
        ${}^{i}\boldsymbol{X}_0 = {}^{i}\boldsymbol{X}_{\lambda(i)}\, {}^{\lambda(i)}\boldsymbol{X}_0$
    end
    $\boldsymbol{v}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}}$
    if $i \in \mathit{fd}$ **then**
        $\boldsymbol{c}_i = \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}}$
    else
        $\boldsymbol{c}_i = \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}} + \boldsymbol{S}_i\, \ddot{\boldsymbol{q}}_i$
    end
    $\boldsymbol{I}_i^A = \boldsymbol{I}_i$
    $\boldsymbol{p}_i^A = \boldsymbol{v}_i \times^* \boldsymbol{I}_i\, \boldsymbol{v}_i - {}^{i}\boldsymbol{X}_0^*\, \boldsymbol{f}_i^x$
end
for $i = N_B$ **to** 1 **do**
    if $i \in \mathit{fd}$ **then**
        $\boldsymbol{U}_i = \boldsymbol{I}_i^A\, \boldsymbol{S}_i$
        $\boldsymbol{D}_i = \boldsymbol{S}_i^{\mathrm{T}}\, \boldsymbol{U}_i$
        $\boldsymbol{u}_i = \boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}}\, \boldsymbol{p}_i^A$
        if $\lambda(i) \neq 0$ **then**
            $\boldsymbol{I}^a = \boldsymbol{I}_i^A - \boldsymbol{U}_i\, \boldsymbol{D}_i^{-1}\, \boldsymbol{U}_i^{\mathrm{T}}$
            $\boldsymbol{I}_{\lambda(i)}^A = \boldsymbol{I}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^*\, \boldsymbol{I}^a\, {}^{i}\boldsymbol{X}_{\lambda(i)}$
            $\boldsymbol{p}^a = \boldsymbol{p}_i^A + \boldsymbol{I}^a \boldsymbol{c}_i + \boldsymbol{U}_i\, \boldsymbol{D}_i^{-1}\boldsymbol{u}_i$
            $\boldsymbol{p}_{\lambda(i)}^A = \boldsymbol{p}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^*\, \boldsymbol{p}^a$
        end
    else
        if $\lambda(i) \neq 0$ **then**
            $\boldsymbol{I}^a = \boldsymbol{I}_i^A$
            $\boldsymbol{I}_{\lambda(i)}^A = \boldsymbol{I}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^*\, \boldsymbol{I}^a\, {}^{i}\boldsymbol{X}_{\lambda(i)}$
            $\boldsymbol{p}^a = \boldsymbol{p}_i^A + \boldsymbol{I}^a \boldsymbol{c}_i$
            $\boldsymbol{p}_{\lambda(i)}^A = \boldsymbol{p}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^*\, \boldsymbol{p}^a$
        end
    end
end
$\vdots$

$\vdots$
$\boldsymbol{a}_0 = -\boldsymbol{a}_g$
for $i = 1$ **to** N_B **do**
    if $i \in \mathit{fd}$ **then**
        $\boldsymbol{a}' = {}^{i}\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i$
        $\ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i^{-1}(\boldsymbol{u}_i - \boldsymbol{U}_i^{\mathrm{T}}\boldsymbol{a}')$
        $\boldsymbol{a}_i = \boldsymbol{a}' + \boldsymbol{S}_i\, \ddot{\boldsymbol{q}}_i$
    else
        $\boldsymbol{a}_i = {}^{i}\boldsymbol{X}_{\lambda(i)}\, \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i$
        $\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}(\boldsymbol{I}_i^A\boldsymbol{a}_i + \boldsymbol{p}_i^A)$
    end
end

Table 9.3: Articulated-body hybrid dynamics algorithm

9.3 Floating Bases

A floating-base rigid-body system is one in which the base body (body 0) is free to move, rather than being fixed in space. Any floating-base system can be converted into an equivalent fixed-base system by making the following changes:

1. Add 1 to all the body and joint numbers, so that the floating base is now body 1, and the lowest-numbered joint is joint 2.

2. Add a fixed base to the system, and a new 6-DoF joint connected between the fixed and floating bases; the former being the new body 0, and the latter the new joint 1.

3. As a consequence of step 2, add 1 to N_B and N_J, and add 6 to n.

Having made these changes, the floating-base system's dynamics can be analysed and calculated by any of the techniques and algorithms designed for fixed-base systems. Nevertheless, floating-base systems are an important special case, and they do merit special treatment.

Let us develop the equations of motion for a floating-base kinematic tree. The quickest approach is to start with the equation of motion for the equivalent fixed-base system. This equation can be written

$$\begin{bmatrix} \boldsymbol{H}_{11} & \boldsymbol{H}_{1*} \\ \boldsymbol{H}_{*1} & \boldsymbol{H}_{**} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_1 \\ \ddot{\boldsymbol{q}}_* \end{bmatrix} + \begin{bmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_* \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_1 \\ \boldsymbol{\tau}_* \end{bmatrix}, \tag{9.11}$$

where the subscripts 1 and $*$ refer to joint 1 and 'all other joints', respectively. This equation is already the correct equation of motion for a floating-base system—it just needs to be massaged into more appropriate notation, which is what we shall now do.

The special property of joint 1 is that is has six degrees of freedom. Its motion-subspace matrix, $\boldsymbol{S}_1$, must therefore be a 6×6 full-rank matrix. In fact, $\boldsymbol{S}_1$ could be any 6×6 full-rank matrix, as two different choices of $\boldsymbol{S}_1$ amount only to two different choices of velocity variables, rather than to any fundamental change in the equation. We therefore choose the most convenient value, which is the identity matrix. In particular, we choose $\boldsymbol{S}_1$ to be the identity matrix in floating-base coordinates.[2]

Having specified $\boldsymbol{S}_1 = \boldsymbol{1}_{6\times 6}$, several other quantities now take on special values. Starting with $\dot{\boldsymbol{q}}_1$, $\ddot{\boldsymbol{q}}_1$ and $\boldsymbol{\tau}_1$, we have

$$\begin{aligned} \boldsymbol{v}_1 &= \boldsymbol{S}_1\,\dot{\boldsymbol{q}}_1 & \boldsymbol{a}_1 &= \dot{\boldsymbol{S}}_1\,\dot{\boldsymbol{q}}_1 + \boldsymbol{S}_1\,\ddot{\boldsymbol{q}}_1 & \text{and} \quad \boldsymbol{\tau}_1 &= \boldsymbol{S}_1^{\mathrm{T}}\boldsymbol{f}_1 \\ &= \boldsymbol{1}_{6\times 6}\,\dot{\boldsymbol{q}}_1 & &= \boldsymbol{v}_1 \times \boldsymbol{S}_1\,\dot{\boldsymbol{q}}_1 + \boldsymbol{S}_1\,\ddot{\boldsymbol{q}}_1 & &= \boldsymbol{1}_{6\times 6}\,\boldsymbol{f}_1 \\ &= \dot{\boldsymbol{q}}_1\,, & &= \boldsymbol{v}_1 \times \boldsymbol{v}_1 + \boldsymbol{1}_{6\times 6}\,\ddot{\boldsymbol{q}}_1 & &= \boldsymbol{f}_1\,. \\ & & &= \ddot{\boldsymbol{q}}_1 \end{aligned}$$

---

[2]That is, $^1\boldsymbol{S}_1 = \boldsymbol{1}_{6\times 6}$; but in any other coordinate system we have $^i\boldsymbol{S}_1 = {}^i\boldsymbol{X}_1 \boldsymbol{1}_{6\times 6}$, which is not the identity matrix. As $\boldsymbol{S}_1$ is fixed in body 1, we have $\dot{\boldsymbol{S}}_1 = \boldsymbol{0}$ and $\dot{\boldsymbol{S}}_1 = \boldsymbol{v}_1 \times \boldsymbol{S}_1$.

So $\dot{q}_1$, $\ddot{q}_1$ and τ_1 contain the Plücker coordinates of the spatial vectors v_1, a_1 and f_1. Next, we observe that f_1 and f_1^x have essentially the same meaning, since they are both external forces acting on the floating base; so the total external force is actually given by the sum $f_1 + f_1^x$. In the equations below, f_1 (and therefore also τ_1) has been set to zero; but we always have the freedom to choose how to apportion the external force between f_1^x and f_1.

As C is the generalized bias force, it follows that C_1 is the value that τ_1 would have to take in order to produce zero joint acceleration at the floating base, assuming that all the other joint accelerations are zero. Given that $\ddot{q}_1 = a_1$ and $\tau_1 = f_1$, this means that C_1 is also the spatial bias force for the composite rigid body comprising the whole floating-base system. We shall use the symbol p_1^c to denote this quantity. Finally, from the definition of H_{ij} in Eq. 6.14, we have

$$H_{1i} = S_1^T I_i^c S_i = I_i^c S_i \qquad \text{and} \qquad H_{11} = S_1^T I_1^c S_1 = I_1^c \,.$$

Taking these changes into account, Eq. 9.11 can be written

$$\begin{bmatrix} I_1^c & H_{1*} \\ H_{*1} & H_{**} \end{bmatrix} \begin{bmatrix} a_1 \\ \ddot{q}_* \end{bmatrix} + \begin{bmatrix} p_1^c \\ C_* \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_* \end{bmatrix} \,.$$

Let us now revert to the original floating-base system, and adjust the notation accordingly. First, N_B, N_J and n revert to their original values, and the bodies and joints revert to their original numbers. The symbols a_1, p_1^c, I_1^c, and so on, therefore become a_0, p_0^c, I_0^c, and so on. Furthermore, as the 6-DoF joint 1 no longer exists, the symbols H_{**}, C_*, $\ddot{q}_*$ and τ_* all simplify to H, C, $\ddot{q}$ and τ, respectively. We replace H_{1*} with a new symbol, F, which is defined as follows:

$$F = \begin{bmatrix} F_1 & F_2 & \cdots & F_{N_B} \end{bmatrix} \tag{9.12}$$

where

$$F_i = I_i^c S_i \,.$$

F is a $6 \times n$ matrix whose columns are the spatial forces required at the floating base to support unit accelerations about each joint variable (cf. §6.3). With these changes, the equation of motion for a floating-base system now reads

$$\begin{bmatrix} I_0^c & F \\ F^T & H \end{bmatrix} \begin{bmatrix} a_0 \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} p_0^c \\ C \end{bmatrix} = \begin{bmatrix} 0 \\ \tau \end{bmatrix} \,. \tag{9.13}$$

This is a system of $n + 6$ equations in $n + 6$ unknowns, reflecting the fact that the floating-base system has $n + 6$ degrees of motion freedom. Note that a complete description of this system's acceleration requires the two vectors a_0 and $\ddot{q}$. Likewise, a complete description of the velocity requires v_0 and $\dot{q}$, and a complete description of the position requires the quantities ${}^0X_{ref}$ and q, where ${}^0X_{ref}$ is the Plücker transform from an inertial reference frame to floating-base coordinates.

Example 9.2 *Starting with Eq. 9.13, suppose we subtract $\boldsymbol{F}^{\mathrm{T}}(\boldsymbol{I}_0^{\mathrm{c}})^{-1}$ times the first row from the second. The resulting equation is*

$$
\begin{bmatrix} \boldsymbol{I}_0^{\mathrm{c}} & \boldsymbol{F} \\ \boldsymbol{0} & \boldsymbol{H} - \boldsymbol{F}^{\mathrm{T}}(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\boldsymbol{F} \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_0 \\ \ddot{\boldsymbol{q}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{p}_0^{\mathrm{c}} \\ \boldsymbol{C} - \boldsymbol{F}^{\mathrm{T}}(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\boldsymbol{p}_0^{\mathrm{c}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \end{bmatrix}.
$$

Extracting the bottom row, we have

$$
\boldsymbol{H}^{fl}\ddot{\boldsymbol{q}} + \boldsymbol{C}^{fl} = \boldsymbol{\tau}\,, \tag{9.14}
$$

where

$$
\boldsymbol{H}^{fl} = \boldsymbol{H} - \boldsymbol{F}^{\mathrm{T}}(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\boldsymbol{F} \tag{9.15}
$$

and

$$
\boldsymbol{C}^{fl} = \boldsymbol{C} - \boldsymbol{F}^{\mathrm{T}}(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\boldsymbol{p}_0^{\mathrm{c}}\,. \tag{9.16}
$$

Equation 9.14 is interesting because it presents us with a direct relationship between $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$, and the coefficients $\boldsymbol{H}^{fl}$ and $\boldsymbol{C}^{fl}$ can be regarded as floating-base analogues to the coefficients of the standard equation of motion. However, this equation is not necessarily a good choice for computational purposes because the subtraction of $\boldsymbol{F}^{\mathrm{T}}(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\boldsymbol{F}$ from $\boldsymbol{H}$ in Eq. 9.15 erases any branch-induced sparsity that may have been present in $\boldsymbol{H}$. Thus, $\boldsymbol{H}^{fl}$ is a dense matrix; and it is perfectly possible for $\boldsymbol{H}^{fl}$ to have more nonzero elements than $\boldsymbol{H}$, despite being smaller.

9.4 Floating-Base Forward Dynamics

If software is available to calculate fixed-base dynamics, and this software supports 6-DoF joints, then it can be used to calculate floating-base dynamics. Alternatively, if dedicated floating-base software is required, then both the articulated-body algorithm and the composite-rigid-body algorithm can be adapted to the task.

Table 9.4 shows the pseudocode for a floating-base articulated-body algorithm. On comparing this algorithm with the one in Table 7.1, the following differences can be seen: $\boldsymbol{v}_0$ is now an input; pass 2 has been extended to calculate $\boldsymbol{I}_0^A$ and $\boldsymbol{p}_0^A$; $\boldsymbol{a}_0$ is calculated from the equation

$$
\boldsymbol{I}_0^A\boldsymbol{a}_0 + \boldsymbol{p}_0^A = \boldsymbol{0}\,; \tag{9.17}
$$

and gravitational acceleration is added to $\boldsymbol{a}_0$ right at the end. This implementation expects the vectors $\boldsymbol{f}_i^x$ and $\boldsymbol{a}_g$ to be supplied in floating-base coordinates, rather than reference coordinates, so that it does not need to know the value of ${}^0\boldsymbol{X}_{ref}$. The symbols ${}^0\boldsymbol{f}_i^x$ and ${}^0\boldsymbol{a}_g$ are used accordingly. Note that ${}^0\boldsymbol{a}_g$ is not a constant, but will vary as the floating base rotates. $({}^0\boldsymbol{a}_g = {}^0\boldsymbol{X}_{ref}{}^{ref}\boldsymbol{a}_g.)$

The effect of a uniform gravitational field on a free-floating rigid-body system is to modify the acceleration of every body in the system by the same

for $i = 1$ **to** N_B **do**
$\quad [\boldsymbol{X}_{\mathrm{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\mathrm{J}}, \boldsymbol{c}_{\mathrm{J}}] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
$\quad {}^i\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}} \, \boldsymbol{X}_{\mathrm{T}}(i)$
$\quad$ **if** $\lambda(i) \neq 0$ **then**
$\quad\quad {}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{\lambda(i)} \, {}^{\lambda(i)}\boldsymbol{X}_0$
$\quad$ **end**
$\quad \boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)} \, \boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}}$
$\quad \boldsymbol{c}_i = \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}}$
$\quad \boldsymbol{I}_i^A = \boldsymbol{I}_i$
$\quad \boldsymbol{p}_i^A = \boldsymbol{v}_i \times^* \boldsymbol{I}_i \, \boldsymbol{v}_i - {}^i\boldsymbol{X}_0^* \, {}^0\boldsymbol{f}_i^x$
end
$\boldsymbol{I}_0^A = \boldsymbol{I}_0$
$\boldsymbol{p}_0^A = \boldsymbol{v}_0 \times^* \boldsymbol{I}_0 \, \boldsymbol{v}_0 - {}^0\boldsymbol{f}_0^x$
for $i = N_B$ **to** 1 **do**
$\quad \boldsymbol{U}_i = \boldsymbol{I}_i^A \, \boldsymbol{S}_i$
$\quad \boldsymbol{D}_i = \boldsymbol{S}_i^{\mathrm{T}} \, \boldsymbol{U}_i$
$\vdots$

$\vdots$
$\quad\quad \boldsymbol{u}_i = \boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}} \, \boldsymbol{p}_i^A$
$\quad\quad \boldsymbol{I}^a = \boldsymbol{I}_i^A - \boldsymbol{U}_i \, \boldsymbol{D}_i^{-1} \, \boldsymbol{U}_i^{\mathrm{T}}$
$\quad\quad \boldsymbol{p}^a = \boldsymbol{p}_i^A + \boldsymbol{I}^a \boldsymbol{c}_i + \boldsymbol{U}_i \, \boldsymbol{D}_i^{-1} \boldsymbol{u}_i$
$\quad\quad \boldsymbol{I}_{\lambda(i)}^A = \boldsymbol{I}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^* \, \boldsymbol{I}^a \, {}^i\boldsymbol{X}_{\lambda(i)}$
$\quad\quad \boldsymbol{p}_{\lambda(i)}^A = \boldsymbol{p}_{\lambda(i)}^A + {}^{\lambda(i)}\boldsymbol{X}_i^* \, \boldsymbol{p}^a$
$\quad$ **end**
$\quad \boldsymbol{a}_0 = -(\boldsymbol{I}_0^A)^{-1} \, \boldsymbol{p}_0^A$
$\quad$ **for** $i = 1$ **to** N_B **do**
$\quad\quad \boldsymbol{a}' = {}^i\boldsymbol{X}_{\lambda(i)} \, \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i$
$\quad\quad \ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i^{-1}(\boldsymbol{u}_i - \boldsymbol{U}_i^{\mathrm{T}} \boldsymbol{a}')$
$\quad\quad \boldsymbol{a}_i = \boldsymbol{a}' + \boldsymbol{S}_i \, \ddot{\boldsymbol{q}}_i$
$\quad$ **end**
$\quad \boldsymbol{a}_0 = \boldsymbol{a}_0 + {}^0\boldsymbol{a}_g$

Table 9.4: Floating-base articulated-body dynamics algorithm

amount. Thus, the relative accelerations of any two bodies are independent of gravity. The algorithm in Table 9.4 exploits this fact by calculating the motion of the system in the absence of a gravitational field, and then adding $\boldsymbol{a}_g$ to $\boldsymbol{a}_0$ right at the end. If you want this algorithm to return the accelerations of other bodies in the system, then $\boldsymbol{a}_g$ must also be added to these accelerations before they are returned.

Table 9.5 presents floating-base versions of the recursive Newton-Euler and composite-rigid-body algorithms. Together, they calculate the coefficients of Eq. 9.13, so that it can be solved for $\boldsymbol{a}_0$ and $\ddot{\boldsymbol{q}}$. Note that the coefficient matrix in this equation has the same branch-induced sparsity pattern as the matrix in Eq. 9.11, so it can be factorized using the algorithms of Section 6.5.

On comparing the modified Newton-Euler algorithm with the original in Table 5.1, the following differences can be seen: $\boldsymbol{v}_0$ is now an input; the term $\boldsymbol{S}_i \ddot{\boldsymbol{q}}_i$ has been removed from the expression for $\boldsymbol{a}_i$, which is duly renamed $\boldsymbol{a}_i^{vp}$, as it accounts for only the velocity-product acceleration terms; pass 2 has been extended to calculate $\boldsymbol{f}_0$; and a line has been added to set $\boldsymbol{p}_0^c = \boldsymbol{f}_0$. Once again, the vectors $\boldsymbol{a}_g$ and $\boldsymbol{f}_i^x$ are supplied in floating-base coordinates, so that the value of ${}^0\boldsymbol{X}_{ref}$ is not needed, and the symbols ${}^0\boldsymbol{a}_g$ and ${}^0\boldsymbol{f}_i^x$ have been used in the psuedocode accordingly. Setting $\boldsymbol{a}_0^{vp} = -\boldsymbol{a}_g$ at the beginning causes gravitational force terms to appear in $\boldsymbol{C}$ and $\boldsymbol{p}_0^c$, which ultimately has the effect of causing the correct gravitational acceleration term to appear in $\boldsymbol{a}_0$ when Eq. 9.13 is solved.

On comparing the modified composite-rigid-body algorithm with the orig-

Calculate $\boldsymbol{C}$ and $\boldsymbol{p}_0^{\mathrm{c}}$:

$\boldsymbol{a}_0^{vp} = -{}^0\boldsymbol{a}_g$
for $i = 1$ **to** N_B **do**
    $[\boldsymbol{X}_{\mathrm{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\mathrm{J}}, \boldsymbol{c}_{\mathrm{J}}] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
    ${}^i\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}}\,\boldsymbol{X}_{\mathrm{T}}(i)$
    if $\lambda(i) \neq 0$ **then**
        ${}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{X}_0$
    end
    $\boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}}$
    $\boldsymbol{a}_i^{vp} = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)}^{vp} + \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}}$
    $\boldsymbol{f}_i = \boldsymbol{I}_i\,\boldsymbol{a}_i^{vp} + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i - {}^i\boldsymbol{X}_0^*\,{}^0\boldsymbol{f}_i^x$
end
$\boldsymbol{f}_0 = \boldsymbol{I}_0\,\boldsymbol{a}_0^{vp} + \boldsymbol{v}_0 \times^* \boldsymbol{I}_0\,\boldsymbol{v}_0 - {}^0\boldsymbol{f}_0^x$
for $i = N_B$ **to** 1 **do**
    $\boldsymbol{C}_i = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{f}_i$
    $\boldsymbol{f}_{\lambda(i)} = \boldsymbol{f}_{\lambda(i)} + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{f}_i$
end
$\boldsymbol{p}_0^{\mathrm{c}} = \boldsymbol{f}_0$

Calculate $\boldsymbol{H}$, $\boldsymbol{F}$ and $\boldsymbol{I}_0^{\mathrm{c}}$:

$\boldsymbol{H} = \boldsymbol{0}$
for $i = 0$ **to** N_B **do**
    $\boldsymbol{I}_i^{\mathrm{c}} = \boldsymbol{I}_i$
end
for $i = N_B$ **to** 1 **do**
    $\boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} = \boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{I}_i^{\mathrm{c}}\,{}^i\boldsymbol{X}_{\lambda(i)}$
    $\boldsymbol{F}_i = \boldsymbol{I}_i^{\mathrm{c}}\,\boldsymbol{S}_i$
    $\boldsymbol{H}_{ii} = \boldsymbol{S}_i^{\mathrm{T}}\,\boldsymbol{F}_i$
    $j = i$
    while $\lambda(j) \neq 0$ **do**
        $\boldsymbol{F}_i = {}^{\lambda(j)}\boldsymbol{X}_j^*\,\boldsymbol{F}_i$
        $j = \lambda(j)$
        $\boldsymbol{H}_{ij} = \boldsymbol{F}_i^{\mathrm{T}}\,\boldsymbol{S}_j$
        $\boldsymbol{H}_{ji} = \boldsymbol{H}_{ij}^{\mathrm{T}}$
    end
    $\boldsymbol{F}_i = {}^0\boldsymbol{X}_j^*\,\boldsymbol{F}_i$
end

Table 9.5: Floating-base versions of the recursive Newton-Euler and composite-rigid-body algorithms

inal in Table 6.2, the following differences can be seen: the calculation of $\boldsymbol{I}_i^{\mathrm{c}}$ has been extended to include $\boldsymbol{I}_0^{\mathrm{c}}$; the old local variable $\boldsymbol{F}$ (which must not be confused with the new $6 \times n$ matrix $\boldsymbol{F}$ in Eq. 9.12) has been replaced by the variables $\boldsymbol{F}_i$, which are the block columns of $\boldsymbol{F}$ appearing in Eq. 9.12; and a line has been added after the **while** loop to transform each $\boldsymbol{F}_i$ to floating-base coordinates. (Each $\boldsymbol{F}_i$ is initially expressed in body i coordinates, and is progressively transformed to floating-base coordinates.)

The algorithms in these tables are not the only ways to solve the problem. Examples of some alternative approaches can be found in Hooker and Margulies (1965); Hooker (1970); Roberson and Schwertassek (1988); Wittenburg (1977).

9.5 Floating-Base Inverse Dynamics

The inverse-dynamics problem for a floating-base system is really a hybrid-dynamics problem: the joint accelerations are known, but the base acceleration is not. Therefore, it would be possible to solve this problem using one of the general hybrid-dynamics algorithms in Sections 9.1 and 9.2. Alternatively, we could design a special-purpose algorithm, which would solve the problem more efficiently. This section presents one such algorithm.

Let $\boldsymbol{f}_i$ be the spatial force that is exerted on body i by its parent. In the

case of body 0, we have $\boldsymbol{f}_0 = \boldsymbol{0}$ because this body does not have a parent. In all other cases, $\boldsymbol{f}_i$ is the force transmitted across joint i. In any kinematic tree, $\boldsymbol{f}_i$ supports the motion of all the bodies in the set $\nu(i)$, so we have

$$\boldsymbol{f}_i = \sum_{j \in \nu(i)} (\boldsymbol{I}_j \, \boldsymbol{a}_j + \boldsymbol{v}_j \times^* \boldsymbol{I}_j \, \boldsymbol{v}_j - \boldsymbol{f}_j^x) . \tag{9.18}$$

Our objective is to solve this equation for $\boldsymbol{a}_0$. To accomplish this, we first express $\boldsymbol{a}_i$ in the form

$$\boldsymbol{a}_i = \boldsymbol{a}_0 + \boldsymbol{a}_i^r ,$$

where $\boldsymbol{a}_i^r$ is the relative acceleration of body i, and then substitute this expression into Eq. 9.18. The result is

$$\boldsymbol{f}_i = \Big(\sum_{j \in \nu(i)} \boldsymbol{I}_j \Big) \boldsymbol{a}_0 + \sum_{j \in \nu(i)} (\boldsymbol{I}_j \, \boldsymbol{a}_j^r + \boldsymbol{v}_j \times^* \boldsymbol{I}_j \, \boldsymbol{v}_j - \boldsymbol{f}_j^x) . \tag{9.19}$$

The coefficient of $\boldsymbol{a}_0$ in this equation is the composite-rigid-body inertia $\boldsymbol{I}_i^c$. If we treat the rest of the right-hand side as a bias force, then we have

$$\boldsymbol{f}_i = \boldsymbol{I}_i^c \, \boldsymbol{a}_0 + \boldsymbol{p}_i^c , \tag{9.20}$$

where

$$\boldsymbol{p}_i^c = \sum_{j \in \nu(i)} (\boldsymbol{I}_j \, \boldsymbol{a}_j^r + \boldsymbol{v}_j \times^* \boldsymbol{I}_j \, \boldsymbol{v}_j - \boldsymbol{f}_j^x) . \tag{9.21}$$

$\boldsymbol{p}_i^c$ is the force that would be required to support the motion of all the bodies in $\nu(i)$ if $\boldsymbol{a}_0$ happened to be zero. It can be calculated recursively using the equation

$$\boldsymbol{p}_i^c = \boldsymbol{p}_i + \sum_{j \in \mu(i)} \boldsymbol{p}_j^c \tag{9.22}$$

where

$$\boldsymbol{p}_i = \boldsymbol{I}_i \, \boldsymbol{a}_i^r + \boldsymbol{v}_i \times^* \boldsymbol{I}_i \, \boldsymbol{v}_i - \boldsymbol{f}_i^x . \tag{9.23}$$

When $i = 0$, Eq. 9.20 can be solved directly for $\boldsymbol{a}_0$, giving

$$\boldsymbol{a}_0 = -(\boldsymbol{I}_0^c)^{-1} \, \boldsymbol{p}_0^c . \tag{9.24}$$

Once $\boldsymbol{a}_0$ is known, the joint forces can be calculated using

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^T \boldsymbol{f}_i = \boldsymbol{S}_i^T (\boldsymbol{I}_i^c \, \boldsymbol{a}_0 + \boldsymbol{p}_i^c) . \tag{9.25}$$

Table 9.6 presents the equations and pseudocode for an algorithm based on Eqs. 9.22 to 9.25 and the usual equations for body velocities, body accelerations and composite-rigid-body inertias. The equations are expressed in body coordinates, and include coordinate transforms where necessary. In line with the other floating-base algorithms, this one expects $\boldsymbol{a}_g$ and $\boldsymbol{f}_i^x$ to be supplied

Pass 1

$$\boldsymbol{a}_0^r = -{}^0\boldsymbol{a}_g$$

$$\boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_i$$

$$\boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}i}$$

$$\boldsymbol{c}_i = \mathring{\boldsymbol{S}}_i\,\dot{\boldsymbol{q}}_i + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}i}$$

$$\boldsymbol{a}_i^r = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$$

$$\boldsymbol{p}_i = \boldsymbol{I}_i\,\boldsymbol{a}_i^r + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i - {}^i\boldsymbol{X}_0^*\,{}^0\boldsymbol{f}_i^x$$

Pass 2

$$\boldsymbol{I}_i^{\mathrm{c}} = \boldsymbol{I}_i + \sum_{j \in \mu(i)} {}^i\boldsymbol{X}_j^*\,\boldsymbol{I}_j^{\mathrm{c}}\,{}^j\boldsymbol{X}_i$$

$$\boldsymbol{p}_i^{\mathrm{c}} = \boldsymbol{p}_i + \sum_{j \in \mu(i)} {}^i\boldsymbol{X}_j^*\,\boldsymbol{p}_j^{\mathrm{c}}$$

Pass 3

$$^0\boldsymbol{a}_0 = -(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\,\boldsymbol{p}_0^{\mathrm{c}}$$

$$^i\boldsymbol{a}_0 = {}^i\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{a}_0$$

$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}(\boldsymbol{I}_i^{\mathrm{c}}\,{}^i\boldsymbol{a}_0 + \boldsymbol{p}_i^{\mathrm{c}})$$

$\boldsymbol{a}_0^r = -{}^0\boldsymbol{a}_g$
for $i = 1$ **to** N_B **do**
    $[\boldsymbol{X}_{\mathrm{J}}, \boldsymbol{S}_i, \boldsymbol{v}_{\mathrm{J}}, \boldsymbol{c}_{\mathrm{J}}] = \mathrm{jcalc}(\mathrm{jtype}(i), \boldsymbol{q}_i, \dot{\boldsymbol{q}}_i)$
    ${}^i\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}}\,\boldsymbol{X}_{\mathrm{T}}(i)$
    if $\lambda(i) \neq 0$ **then**
        ${}^i\boldsymbol{X}_0 = {}^i\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{X}_0$
    end
    $\boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{\mathrm{J}}$
    $\boldsymbol{a}_i^r = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)}^r + \boldsymbol{c}_{\mathrm{J}} + \boldsymbol{v}_i \times \boldsymbol{v}_{\mathrm{J}} + \boldsymbol{S}_i\,\ddot{\boldsymbol{q}}_i$
    $\boldsymbol{I}_i^{\mathrm{c}} = \boldsymbol{I}_i$
    $\boldsymbol{p}_i^{\mathrm{c}} = \boldsymbol{I}_i\,\boldsymbol{a}_i^r + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i - {}^i\boldsymbol{X}_0^*\,{}^0\boldsymbol{f}_i^x$
end
$\boldsymbol{I}_0^{\mathrm{c}} = \boldsymbol{I}_0$
$\boldsymbol{p}_0^{\mathrm{c}} = \boldsymbol{I}_0\,\boldsymbol{a}_0^r + \boldsymbol{v}_0 \times^* \boldsymbol{I}_0\,\boldsymbol{v}_0 - {}^0\boldsymbol{f}_0^x$
for $i = N_B$ **to** 1 **do**
    $\boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} = \boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{I}_i^{\mathrm{c}}\,{}^i\boldsymbol{X}_{\lambda(i)}$
    $\boldsymbol{p}_{\lambda(i)}^{\mathrm{c}} = \boldsymbol{p}_{\lambda(i)}^{\mathrm{c}} + {}^{\lambda(i)}\boldsymbol{X}_i^*\,\boldsymbol{p}_i^{\mathrm{c}}$
end
$^0\boldsymbol{a}_0 = -(\boldsymbol{I}_0^{\mathrm{c}})^{-1}\boldsymbol{p}_0^{\mathrm{c}}$
for $i = 1$ **to** N_B **do**
    $^i\boldsymbol{a}_0 = {}^i\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{a}_0$
    $\boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}(\boldsymbol{I}_i^{\mathrm{c}}\,{}^i\boldsymbol{a}_0 + \boldsymbol{p}_i^{\mathrm{c}})$
end

Table 9.6: Floating-base inverse dynamics equations and algorithm

in floating-base coordinates, and the symbols $^0\boldsymbol{a}_g$ and $^0\boldsymbol{f}_i^x$ have been used accordingly. This algorithm handles gravity by initializing $\boldsymbol{a}_0^r$ to $-\boldsymbol{a}_g$ rather than zero. This causes gravitational force terms to appear in $\boldsymbol{p}_i^{\mathrm{c}}$, which in turn cause the correct gravitational acceleration to appear in $\boldsymbol{a}_0$. Alternatively, one could set $\boldsymbol{a}_0^r$ to zero at the beginning of pass 1, and then add $\boldsymbol{a}_g$ to $\boldsymbol{a}_0$ at the end of pass 3.

More on this topic, and on related topics, can be found in Dubowsky and Papadopoulos (1993); Fang and Pollard (2003); Longman et al. (1987); Umetani and Yoshida (1989); Vafa and Dubowsky (1990a,b); and many other papers.

Example 9.3 *The total spatial momentum of a rigid-body system is the sum of the momenta of its bodies. The momentum of a floating-base system is therefore given by*

$$momentum = \sum_{i=0}^{N_B} \boldsymbol{I}_i\,\boldsymbol{v}_i\,.$$

In the absence of external forces, this quantity is conserved. It turns out that

the momentum is also given by the expression $\boldsymbol{I}_0^c \boldsymbol{v}_0 + \boldsymbol{F}\dot{\boldsymbol{q}}$. We can prove this as follows:

$$
\begin{aligned}
\sum_{i=0}^{N_B} \boldsymbol{I}_i \, \boldsymbol{v}_i \; &= \; \sum_{i=0}^{N_B} \boldsymbol{I}_i \left(\boldsymbol{v}_0 + \sum_{j \in \kappa(i)} \boldsymbol{S}_j \, \dot{\boldsymbol{q}}_j \right) \\
&= \; \sum_{i=0}^{N_B} \boldsymbol{I}_i \, \boldsymbol{v}_0 + \sum_{j=1}^{N_B} \sum_{i \in \nu(j)} \boldsymbol{I}_i \, \boldsymbol{S}_j \, \dot{\boldsymbol{q}}_j \\
&= \; \boldsymbol{I}_0^c \, \boldsymbol{v}_0 + \sum_{j=1}^{N_B} \boldsymbol{I}_j^c \, \boldsymbol{S}_j \, \dot{\boldsymbol{q}}_j \; = \; \boldsymbol{I}_0^c \, \boldsymbol{v}_0 + \boldsymbol{F}\dot{\boldsymbol{q}} \, .
\end{aligned}
$$

(See Eq. 4.3 for summation over $\nu(j)$.) Using a similar argument, we can show that the kinetic energy of a floating-base system is

$$
T = \frac{1}{2} \sum_{i=0}^{N_B} \boldsymbol{v}_i^{\mathrm{T}} \boldsymbol{I}_i \, \boldsymbol{v}_i = \frac{1}{2} \begin{bmatrix} \boldsymbol{v}_0^{\mathrm{T}} & \dot{\boldsymbol{q}}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_0^c & \boldsymbol{F} \\ \boldsymbol{F}^{\mathrm{T}} & \boldsymbol{H} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_0 \\ \dot{\boldsymbol{q}} \end{bmatrix} .
$$

Example 9.4 *In a fixed-base system, the acceleration of body i is given by $\boldsymbol{a}_i = \boldsymbol{J}_i \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i \dot{\boldsymbol{q}}$, where $\boldsymbol{J}_i$ is the Jacobian of body i. In a floating-base system, this becomes*

$$
\boldsymbol{a}_i = \boldsymbol{J}_i \, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i \, \dot{\boldsymbol{q}} + \boldsymbol{a}_0 \, .
$$

However, from Eq. 9.13 we have

$$
\boldsymbol{I}_0^c \, \boldsymbol{a}_0 + \boldsymbol{F}\ddot{\boldsymbol{q}} + \boldsymbol{p}_0^c = \boldsymbol{0} \, .
$$

Putting these two equations together, we get

$$
\begin{aligned}
\boldsymbol{a}_i \; &= \; \boldsymbol{J}_i \, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i \, \dot{\boldsymbol{q}} - (\boldsymbol{I}_0^c)^{-1}(\boldsymbol{F}\ddot{\boldsymbol{q}} + \boldsymbol{p}_0^c) \\
&= \; \boldsymbol{J}_i^{fl} \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i \, \dot{\boldsymbol{q}} - (\boldsymbol{I}_0^c)^{-1} \boldsymbol{p}_0^c \, ,
\end{aligned}
$$

where

$$
\boldsymbol{J}_i^{fl} = \boldsymbol{J}_i - (\boldsymbol{I}_0^c)^{-1} \boldsymbol{F} \, .
$$

The quantity $\boldsymbol{J}_i^{fl}$ is, in effect, a floating-base Jacobian, since it describes the linear dependency of $\boldsymbol{a}_i$ on $\ddot{\boldsymbol{q}}$ (cf. Umetani and Yoshida (1989)). This matrix also relates changes in $\dot{\boldsymbol{q}}$ to changes in $\boldsymbol{v}_i$. For example, if an impulse within the system causes $\dot{\boldsymbol{q}}$ to change by an amount $\delta\dot{\boldsymbol{q}}$, then the consequential change in $\boldsymbol{v}_i$ is

$$
\delta\boldsymbol{v}_i = \boldsymbol{J}_i^{fl} \delta\dot{\boldsymbol{q}} \, .
$$

9.6　Gears

Many mechanical systems contain gear pairs, or components that perform a similar function. From a mathematical point of view, a gear pair is an algebraic
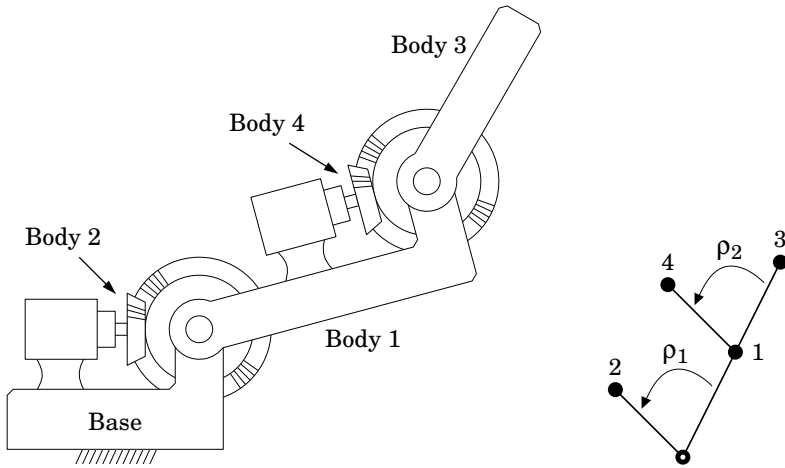
Figure 9.2: A two-link robot arm driven by motors and gears

equation involving two joint variables. Typically, one variable is a constant multiple of the other, but nonlinear relationships are also possible. Another possibility is that more than two variables can be involved. For example, a differential gear box usually has two outputs and either one or two inputs, and therefore involves either three or four variables in total.

Gear constraints can be incorporated into a kinematic tree using the same technique as was used in Section 8.11 for kinematic loops; namely, to choose a vector of independent joint variables, y, and define a function, γ, that maps y to q. Let us now study this technique with the aid of an example.

Figure 9.2 shows a simple two-link robot arm in which both joints are driven by electric motors via gear pairs. This system contains the following parts: a fixed base, an upper arm, a forearm, two motors, two large gear wheels and two small gear wheels. Each motor consists of two parts: a stator and a rotor. The two stators are fixed to the base and the upper arm, respectively, and the two rotors are each fixed to a small gear wheel. The two large gear wheels are fixed to the upper arm and forearm, respectively. The system therefore contains a total of four moving bodies, defined as follows:

body 1: the upper arm, the first large gear wheel and the stator of the second motor;

body 2: the first small gear wheel and the rotor of the first motor;

body 3: the forearm and the second large gear wheel;

body 4: the second small gear wheel and the rotor of the second motor.

Joints 1 and 3 are therefore the shoulder and elbow joints of the robot arm, while joints 2 and 4 are the bearings inside the motors that allow the rotors to rotate

relative to the stators. The two gear pairs impose the following constraints on the joint variables:

$$q_2 = \rho_1\, q_1 \qquad \text{and} \qquad q_4 = \rho_2\, q_3\,,$$

where ρ_1 and ρ_2 are gear ratios. Figure 9.2 also shows the connectivity graph of this system, and the gearing relationships between the joints. The arrows serve only to indicate the direction in which the stated gear ratio applies. Thus, the arrow labelled ρ_1 points from joint 1 to joint 2 because q_2 is ρ_1 times q_1.

To incorporate the gear constraints into the equation of motion, we first define a vector of independent variables, $\boldsymbol{y}$, and then define a function, $\boldsymbol{\gamma}$, that gives $\boldsymbol{q}$ as a function of $\boldsymbol{y}$. Let us choose q_1 and q_3 to be the independent variables. $\boldsymbol{y}$ is then the vector $[y_1\; y_2]^\mathrm{T}$, where $y_1 = q_1$ and $y_2 = q_3$. With this choice of variables, $\boldsymbol{\gamma}$ is defined as follows:

$$\boldsymbol{\gamma}(\boldsymbol{y}) = \begin{bmatrix} y_1 \\ \rho_1\, y_1 \\ y_2 \\ \rho_2\, y_2 \end{bmatrix}.$$

The next step is to derive the quantities $\boldsymbol{G} = \partial\boldsymbol{\gamma}/\partial\boldsymbol{y}$ and $\boldsymbol{g} = \dot{\boldsymbol{G}}\dot{\boldsymbol{y}}$ from $\boldsymbol{\gamma}$. For our example, these quantities are

$$\boldsymbol{G} = \frac{\partial\boldsymbol{\gamma}}{\partial\boldsymbol{y}} = \begin{bmatrix} 1 & 0 \\ \rho_1 & 0 \\ 0 & 1 \\ 0 & \rho_2 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{g} = \dot{\boldsymbol{G}}\,\dot{\boldsymbol{y}} = \boldsymbol{0}\,.$$

($\boldsymbol{g}$ would only ever be nonzero if there were a nonlinear gear pair in the system.) Finally, the equation of motion for the kinematic tree, taking into account the gear constraints, is

$$\boldsymbol{G}^\mathrm{T}\boldsymbol{H}\,\boldsymbol{G}\,\ddot{\boldsymbol{y}} + \boldsymbol{G}^\mathrm{T}(\boldsymbol{C} + \boldsymbol{H}\boldsymbol{g}) = \boldsymbol{G}^\mathrm{T}\boldsymbol{\tau}\,, \tag{9.26}$$

where $\boldsymbol{H}$ and $\boldsymbol{C}$ are the coefficients of the equation of motion in the absence of gear constraints. (See Eq. 8.45.) As the motors exert their torques about joints 2 and 4, elements τ_2 and τ_4 of $\boldsymbol{\tau}$ will contain the motor torques, while $\tau_1 = \tau_3 = 0$. The inverse dynamics of a geared kinematic tree can be computed using the method described in Section 8.12.

$\boldsymbol{G}$ will usually be a very sparse matrix. In fact, it will typically contain only one nonzero element per row, and many of those elements will be ones. It is therefore recommended to exploit this sparsity. By way of example, if we treat $\boldsymbol{G}$ and $\boldsymbol{H}$ as dense matrices, then the cost of calculating $\boldsymbol{G}^\mathrm{T}\boldsymbol{H}\boldsymbol{G}$ for a 4×2 matrix $\boldsymbol{G}$ is 44 multiplications and 33 additions. In contrast, if we use the actual values of $\boldsymbol{G}$ and $\boldsymbol{H}$ for the above example, then the cost is a mere

5 multiplications and 3 additions, as shown below. Obviously, the savings will be even greater with larger matrices.

$$
\boldsymbol{G}^{\mathrm{T}} \boldsymbol{H} \boldsymbol{G} = \begin{bmatrix} 1 & \rho_1 & 0 & 0 \\ 0 & 0 & 1 & \rho_2 \end{bmatrix} \begin{bmatrix} H_{11} & 0 & H_{13} & H_{14} \\ 0 & H_{22} & 0 & 0 \\ H_{13} & 0 & H_{33} & 0 \\ H_{14} & 0 & 0 & H_{44} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \rho_1 & 0 \\ 0 & 1 \\ 0 & \rho_2 \end{bmatrix}
$$

$$
= \begin{bmatrix} H_{11} + \rho_1^2 \, H_{22} & H_{13} + \rho_2 \, H_{14} \\ H_{13} + \rho_2 \, H_{14} & H_{33} + \rho_2^2 \, H_{44} \end{bmatrix} . \tag{9.27}
$$

Example 9.5 *In a typical robot mechanism, the rotors of the electric motors are much smaller than the other parts of the mechanism, and therefore account for only a small fraction of the total mass. However, they also reach much higher speeds than the bodies they are driving, and can therefore exert a substantial influence on the overall dynamics of the system.*

The method presented in this section is the correct way to account for the dynamic effects of rotors, but there is also a quick-and-dirty method that is popular in robotics. According to this method, we dispense with the rotors and the gearboxes, and model the actuator as a fictitious electric motor that develops ρ times the torque of the real motor, runs at $1/\rho$ times the speed, and has ρ^2 times the rotor inertia of the real motor, where ρ is the step-down gear ratio. These inertias, which are scalar constants, are simply added to the diagonal elements of $\boldsymbol{H}$. In particular, if I_i^{rot} is the scaled rotor inertia for the motor that is driving joint i, then I_i^{rot} is added to $\boldsymbol{H}_{ii}$. For this to work, joint i must be a 1-DoF joint, so that $\boldsymbol{H}_{ii}$ is a 1×1 matrix.

This method captures the largest inertial effects of the rotors, but it ignores the smaller inertial effects and all of the gyroscopic effects. A clue as to why this method works can be seen in Eq. 9.27. If the rotors are small, then H_{22}, H_{14} and H_{44} will be small compared with H_{11}, H_{13} and H_{33}, but ρ_1 and ρ_2 will be moderately large. (Gear ratios of $100:1$ are fairly typical.) The two rotors will therefore have their biggest effect on the two diagonal elements, where they get multiplied by the squares of the gear ratios.

The quick-and-dirty method also works with the articulated-body algorithm. In this case, we modify Eq. 7.44 to read

$$
\boldsymbol{D}_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{U}_i + I_i^{rot} . \tag{9.28}
$$

Again, joint i must be a 1-DoF joint, so that $\boldsymbol{D}_i$ is a 1×1 matrix.

9.7 Dynamic Equivalence

It is perfectly possible for two different rigid-body systems to have the same equation of motion. We shall describe such systems as *dynamically equivalent*. An important special case occurs when the two systems differ only in their inertia parameters. An immediate consequence of this kind of equivalence is that
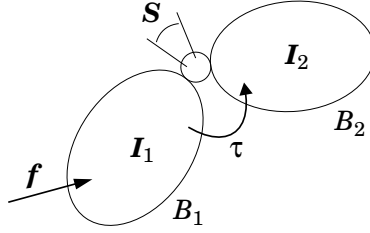
Figure 9.3: A two-body floating rigid-body system

the inertia parameters of a rigid-body system cannot be identified from measurements of its dynamic behaviour: one must instead identify an observable subset, which are called the base inertia parameters (Khalil and Dombre, 2002). Another consequence, which is the topic of this section, is that the inertia parameters of a system model can be adjusted without altering its behaviour. If the adjustments result in fewer parameters having nonzero values, then the cost of computing the dynamics can be reduced.

Consider the rigid-body system shown in Figure 9.3. It consists of two bodies, B_1 and B_2, having inertias of $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, respectively, which are connected by a joint having a motion subspace of $\boldsymbol{S}$. A spatial force $\boldsymbol{f}$ acts on B_1, and a generalized force $\boldsymbol{\tau}$ acts at the joint. The equation of motion for this system is

$$\begin{bmatrix} \boldsymbol{I}_1 + \boldsymbol{X}_{\mathrm{J}}^{\mathrm{T}} \boldsymbol{I}_2 \, \boldsymbol{X}_{\mathrm{J}} & \boldsymbol{X}_{\mathrm{J}}^{\mathrm{T}} \boldsymbol{I}_2 \, \boldsymbol{S} \\ \boldsymbol{S}^{\mathrm{T}} \boldsymbol{I}_2 \, \boldsymbol{X}_{\mathrm{J}} & \boldsymbol{S}^{\mathrm{T}} \boldsymbol{I}_2 \, \boldsymbol{S} \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_1 \\ \ddot{\boldsymbol{q}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{C}_f \\ \boldsymbol{C}_\tau \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{\tau} \end{bmatrix}, \qquad (9.29)$$

where

$$\begin{bmatrix} \boldsymbol{C}_f \\ \boldsymbol{C}_\tau \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_1 \times^* \boldsymbol{I}_1 \, \boldsymbol{v}_1 + \boldsymbol{X}_{\mathrm{J}}^{\mathrm{T}} (\boldsymbol{I}_2 \, \dot{\boldsymbol{S}} \, \dot{\boldsymbol{q}} + \boldsymbol{v}_2 \times^* \boldsymbol{I}_2 \, \boldsymbol{v}_2) \\ \boldsymbol{S}^{\mathrm{T}} (\boldsymbol{I}_2 \, \dot{\boldsymbol{S}} \, \dot{\boldsymbol{q}} + \boldsymbol{v}_2 \times^* \boldsymbol{I}_2 \, \boldsymbol{v}_2) \end{bmatrix}$$

and $\boldsymbol{v}_2 = \boldsymbol{X}_{\mathrm{J}} \boldsymbol{v}_1 + \boldsymbol{S} \dot{\boldsymbol{q}}$. Except for notational differences, this equation is just an application of Eq. 9.13 to a two-body system. In formulating this equation, we have chosen the joint's predecessor and successor frames to serve as link coordinate frames, so that $^2\boldsymbol{X}_1 = \boldsymbol{X}_{\mathrm{J}}$.

We now modify this system by adding a spatial inertia of $\boldsymbol{I}_\Delta$ to $\boldsymbol{I}_1$ and subtracting $\boldsymbol{I}_\Delta$ from $\boldsymbol{I}_2$. If we want this modification not to alter the coefficients of Eq. 9.29, then $\boldsymbol{I}_\Delta$ must meet the following conditions:[3]

$$\boldsymbol{I}_\Delta \boldsymbol{S} \equiv \boldsymbol{0}, \qquad \boldsymbol{I}_\Delta \equiv \boldsymbol{X}_{\mathrm{J}}^{\mathrm{T}} \boldsymbol{I}_\Delta \boldsymbol{X}_{\mathrm{J}} \qquad \text{and} \qquad \boldsymbol{S}^{\mathrm{T}} \boldsymbol{v}_1 \times^* \boldsymbol{I}_\Delta \boldsymbol{v}_1 \equiv \boldsymbol{0}. \qquad (9.30)$$

These conditions must be met at every joint position, and for every $\boldsymbol{v}_1 \in \mathsf{M}^6$. It therefore follows that the conditions depend only on the joint type, and that suitable values for $\boldsymbol{I}_\Delta$ can be found without knowing anything about the system other than the joint type.

---

[3]To obtain this set of conditions, it is necessary to assume that the joint's motion subspace, range($\boldsymbol{S}$), is a constant in either B_1 or B_2 coordinates. This assumption does not hold for all joint types.

Joint type	I_Δ		
revolute	$\begin{bmatrix} I & 0 & 0 & 0 & -mc_z & 0 \\ 0 & I & 0 & mc_z & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & mc_z & 0 & m & 0 & 0 \\ -mc_z & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \end{bmatrix}$		
prismatic	$\begin{bmatrix} I_{xx} & I_{xy} & I_{xz} & 0 & 0 & 0 \\ I_{xy} & I_{yy} & I_{yz} & 0 & 0 & 0 \\ I_{xz} & I_{yz} & I_{zz} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \bar{I} & 0 \\ 0 & 0 \end{bmatrix}$		
spherical	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & m\mathbf{1} \end{bmatrix}$		

Table 9.7: Values of I_Δ that do not alter the equation of motion

Table 9.7 shows the values of I_Δ that meet the conditions in Eq. 9.30 for some joint types. Let us examine the physical interpretation of these results. In the case of a spherical joint, I_Δ is the inertia of a particle of mass m located at the joint's rotation centre. This result says you can add particles of masses m and $-m$ to B_1 and B_2, respectively, both located at the rotation centre, and because these two particles always coincide, they cancel each other exactly. In the case of a revolute joint, I_Δ is the inertia of a pair of particles at different positions on the joint axis (which is the z axis). m and c_z are the total mass and mass centre of the pair, and I depends on the distance between them. Again, because the particles are located on the rotation axis, if we add them to B_1 and add their negatives to B_2 then they always coincide, and therefore always cancel. In the case of a prismatic joint, I_Δ is a disembodied pure rotational inertia. Such inertias are translationally invariant; so, if we add I_Δ to B_1 and subtract it from B_2, then the two will always cancel so long as B_2 never rotates relative to B_1.

Clearly, the subtraction of a sufficiently massive particle from B_2 will result in a negative mass, and the subtraction of a sufficiently large pure rotational inertia will result in a negative rotational inertia. It is therefore possible for B_2 to end up with a rigid-body inertia that is not physically realizable. Nevertheless, the equation of motion for the modified system is identical to the original.

If B_1 and B_2 are part of a larger rigid-body system, then the above results still apply—replacing a two-body subsystem with a dynamically equivalent sub-

system does not alter the equation of motion for the system as a whole.

Simplification of Rigid-Body systems

The results in Table 9.7 allow a dynamics simulator to simplify the inertia parameters of any given rigid-body system by adding and subtracting values of I_Δ that are designed to zero as many of the inertia parameters as possible. The procedure is as follows:

> **for** $i = N_B$ **to** 1 **do**
> $I_\Delta = \text{best\_match}(\text{jtype}(i), I_i)$
> $I_i = I_i - I_\Delta$
>   **if** $\lambda(i) \neq 0$ **then**
> $I_{\lambda(i)} = I_{\lambda(i)} + I_\Delta$
>   **end**
> **end**

The function best_match first looks in a table of I_Δ formulae to find an entry for the joint type jtype(i). If no entry is found, then it returns the value $\mathbf{0}_{6\times6}$. However, if a formula is found, then it returns the value of I_Δ that maximizes the number of zero-valued inertia parameters in the expression $I_i - I_\Delta$. For example, if joint i is spherical, then the returned value of I_Δ has the same mass as I_i, so that the mass of $I_i - I_\Delta$ is zero.

The point of this procedure is that it produces a predictable pattern of zero-valued parameters, which can be exploited by writing code for each special case.[4] Assuming that most joints are revolute, this technique can cut the cost of the recursive Newton-Euler algorithm by about 15%, but it is less effective on forward dynamics algorithms.

It is possible for a dynamics algorithm to fail on the modified system when it would have succeeded on the original. For example, if an algorithm uses the inverses of the rigid-body inertias, then it will only work if those inverses exist. If the modifications have brought the mass of body i to zero, then its modified inertia will be singular.

More on this topic, and related topics, can be found in Gautier and Khalil (1990); Khalil and Kleinfinger (1987); Khalil and Dombre (2002).

Augmented Bodies

The above technique resembles a much older technique called *augmented bodies*. They both involve inertia parameters, and they both have the same objective: cost reduction. However, the augmented-body technique relies on a rewriting of the equations of motion that collects inertia parameters into constant subexpressions, which can be computed in advance. These precomputed values can be regarded as the inertia parameters of a collection of augmented bodies.

---

[4]A better strategy is to use the symbolic simplification technique described in §10.4.

Starting with a kinematic tree, augmented body i is obtained by adding one particle to rigid body i for each joint attached to that body. The locations of these particles depend on the joints, but each has a mass equal to the mass of the subtree that is connected to body i via that joint. In a variant form of augmented body, a particle is added only for each joint in $\mu(i)$. The former tends to be used with floating-base systems, and the latter with fixed-base systems. More on this topic can be found in Balafoutis et al. (1988); Balafoutis and Patel (1989, 1991); Hooker and Margulies (1965); Hooker (1970); Roberson and Schwertassek (1988); Wittenburg (1977).

Chapter 10

Accuracy and Efficiency

In theory, the algorithms presented in this book are all exact. However, the presence of round-off errors in the calculations ensures that the results are rarely so. Other sources of error include the use of numerical integration, and the differences in behaviour between a rigid-body system and the physical system it represents. Section 10.1 examines these various sources of error, paying particular attention to the sources of round-off error in dynamics calculations and ways to avoid them.

If the acceleration of a rigid-body system changes greatly in response to a tiny change in the applied forces, then it is said to be sensitive. If you want to know the acceleration of such a system accurately, then the forces must be known even more accurately. It turns out that many kinematic trees have this property, and the phenomenon is examined in Section 10.2.

Section 10.3 explores the topic of efficiency. It explains how the efficiency of a dynamics algorithm is measured; and it presents tables and graphs comparing the efficiencies of several versions of the three main algorithms for kinematic trees. The efficiency of the composite-rigid-body algorithm is examined in detail, partly to show how an operations-count formula is obtained, and partly because the efficiency of this particular algorithm varies greatly with the amount of branching in the tree. This section also shows that the true asymptotic complexity of the composite-rigid-body algorithm is $O(nd)$, where d is the depth of the connectivity tree, and that the true complexity of the so-called $O(n^3)$ algorithm is $O(nd^2)$.

One of the disadvantages of model-based algorithms is that they have to cater for the general case. As such, they perform many calculations that are indeed necessary in the general case, but are frequently not necessary for some particular system model. One solution to this problem is the automatic generation of customized dynamics routines—stripped-down versions of the general algorithm that perform only those calculations that are necessary for a specific given system model. Section 10.4 presents a brief description of a technique for doing this, which is called symbolic simplification.

10.1 Sources of Error

Dynamics calculations are subject to three kinds of error: *round-off error*, which
is the result of using finite-precision floating-point arithmetic; *truncation error*,
which is the result of approximating an infinite series by a finite number of
terms; and *modelling error*, which refers to the discrepancy between the be-
haviour of the mathematical model and the behaviour of the physical system
it represents. Each kind of error is distinct from the others, so let us examine
them separately.

Round-off Error

Round-off errors occur during most floating-point operations, but they are usu-
ally tiny. Large errors can arise if two large-magnitude numbers are added or
subtracted, producing a small-magnitude result, and especially if this happens
several times during the course of a calculation. The simplest defence against
round-off error is to use double-precision floating-point arithmetic for all calcu-
lations, but this will not solve every problem.

Large round-off errors can arise during dynamics calculations for a variety
of reasons. Some of the more likely ones are:

1. using a far-away coordinate system;
2. large velocities;
3. large inertia ratios; and
4. nearly singular loop-closure constraints.

A coordinate system is far away from a rigid body if the distance between
the origin and the centre of mass is many times the body's radius of gyration.
Ideally, this distance should be no more than about one or two radii of gyration.
This effect is the reason why dynamics algorithms are more accurate when
they are implemented in body coordinates rather than absolute coordinates.
The large-velocity problem arises in systems with large mean velocities, like
spacecraft, in which the relative velocities of the component parts are small
compared with their absolute velocities. This problem can be solved by using
an inertial reference frame with a velocity close to the mean linear velocity of
the system. Large differences in inertia arise from relatively small differences
in scale. For example, if two solid spheres are made of the same substance,
and one is 10 times larger than the other, then the larger sphere will have 1000
times the mass of the smaller one, and 100,000 times the rotational inertia.

Let us examine in more detail how some of these round-off errors arise.
Figure 10.1 shows a planar rigid body having a centre of mass at the point C,
which is located relative to the origin by the vector $\boldsymbol{c} = \overrightarrow{OC} = [c_x\ c_y]^{\mathrm{T}}$. We shall
use planar vectors in this example, to keep the matrices small, but the round-off
problems are essentially the same for planar and spatial vectors. This body has
unit mass, and it has unit rotational inertia about C, so its radius of gyration
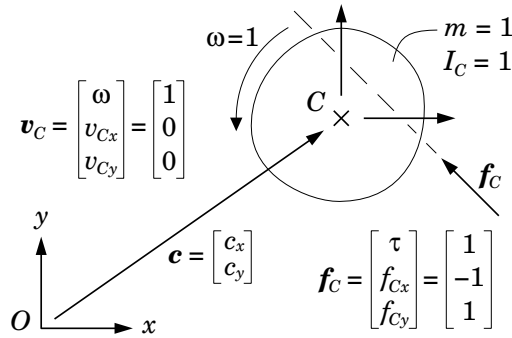
Figure 10.1: Calculating the acceleration of a planar rigid body

is 1. Expressed in C coordinates, the velocity of this body is $\boldsymbol{v}_C = [1\ 0\ 0]^{\mathrm{T}}$, and it is subject to an applied force of $\boldsymbol{f}_C = [1\ {-1}\ 1]^{\mathrm{T}}$. The former is a unit rotation about C, and the latter is a pure force having a magnitude of $\sqrt{2}$ and a line of action passing through the point $(0.5, 0.5)$ relative to C. Expressed in O coordinates, the equation of motion for this body is

$$\boldsymbol{I}_O\, \boldsymbol{a}_O = \boldsymbol{f}_O - \boldsymbol{v}_O \times^* \boldsymbol{I}_O\, \boldsymbol{v}_O\,, \tag{10.1}$$

where

$$\boldsymbol{v}_O = \begin{bmatrix} 1 \\ c_y \\ -c_x \end{bmatrix}, \qquad \boldsymbol{v}_O\times^* = \begin{bmatrix} 0 & c_x & c_y \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad \boldsymbol{f}_O = \begin{bmatrix} 1 + c_x + c_y \\ -1 \\ 1 \end{bmatrix}$$

and

$$\boldsymbol{I}_O = \begin{bmatrix} 1 + c_x^2 + c_y^2 & -c_y & c_x \\ -c_y & 1 & 0 \\ c_x & 0 & 1 \end{bmatrix}.$$

The exact solution to this equation is

$$\boldsymbol{a}_O = \begin{bmatrix} 1 \\ c_y - 1 \\ 1 - c_x \end{bmatrix}.$$

Suppose we set $c_x = 100$ and $c_y = 10$, and ask the computer to evaluate $\boldsymbol{a}_O$ using IEEE double-precision floating-point arithmetic, which is accurate to one part in 10^{16}. The computer will do this by solving the linear equation

$$\begin{bmatrix} 10101 & -10 & 100 \\ -10 & 1 & 0 \\ 100 & 0 & 1 \end{bmatrix} \boldsymbol{a}_O = \begin{bmatrix} 111 \\ -1 \\ 1 \end{bmatrix},$$

and the calculated value will differ from the exact value by about one part in 10^{12}. Thus, four significant figures of accuracy have been lost. The reason
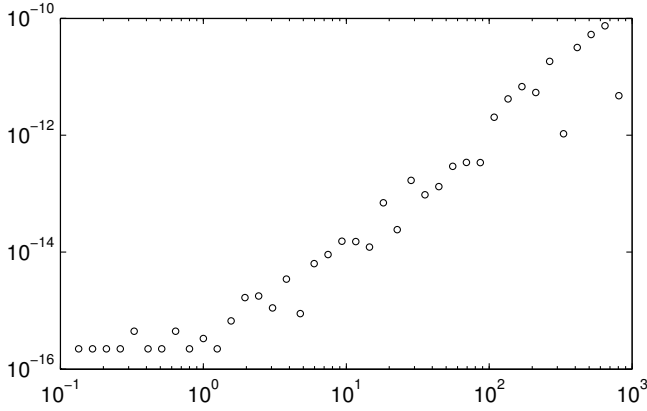
Figure 10.2: Round-off error in the value of $\boldsymbol{a}_O$, measured at C, versus $|\boldsymbol{c}|$

for this loss is that the coefficient matrix is highly ill-conditioned, having a condition number of 1.02×10^8. To give an idea of how the round-off error varies with $|\boldsymbol{c}|$, Figure 10.2 plots $\|\boldsymbol{a}_C - {}^C\boldsymbol{X}_O\boldsymbol{a}_O\|_\infty$ against $|\boldsymbol{c}|$, where $\boldsymbol{a}_C$ is the exact value and $\boldsymbol{a}_O$ the computed value. It can be seen that the round-off error is approximately constant when $|\boldsymbol{c}| < 1$ (i.e., when $|\boldsymbol{c}|$ is less than the radius of gyration), and grows approximately in proportion to $|\boldsymbol{c}|^2$ when $|\boldsymbol{c}| > 1$.

In this example, the linear velocity of the body's centre of mass was zero ($v_{Cx} = v_{Cy} = 0$). If instead the body were to have a large linear velocity, then there could be substantial round-off error in calculating the right-hand side of Eq. 10.1, even if $|\boldsymbol{c}|$ is relatively small.

Round-off Error in Dynamics Algorithms

Round-off error is known to increase with body count, and to increase more quickly with some algorithms than with others. The three main algorithms for kinematic trees are known to remain accurate, even when there are thousands of bodies in the system; and it is also known that the articulated-body algorithm is more accurate than the composite-rigid-body algorithm. However, the round-off behaviour of many other algorithms is not known. Two relevant articles on this subject are Ascher et al. (1997) and Featherstone (1999b).

Truncation Error

Truncation errors are the result of using finite approximations to infinite series, continuous curves, continuous functions and the like. In rigid-body dynamics, the main source of truncation error is the numerical integration process, which is only an approximation to exact mathematical integration. By its nature, the management of truncation error involves a compromise between efficiency

and accuracy: a better approximation will reduce the error, but will be more expensive to calculate. The management of integration error involves making an appropriate choice of integration method and time step.

If one plots the acceleration variables of a simulated rigid-body system against time, then it will often be the case that the resulting graph contains large spikes: brief periods when the accelerations are much higher than at other times. Under these circumstances, it may be beneficial to use an integration method that automatically adapts its step size to maintain a prescribed accuracy. Such methods take big steps when not much is happening, and tiny steps when the accelerations are high. This strategy can lead to an improvement in the overall efficiency of a simulation run by reducing the total number of times that the dynamics has to be calculated.

Modelling Error

Modelling errors are those that cause the behaviour of the mathematical model to differ from the behaviour of the physical system. If the mathematical model is a rigid-body system, then the main sources of modelling error are:

- the rigid-body assumption,

- the ideal-joint assumption (exact kinematic constraints),

- unmodelled dynamics, and

- inaccuracies in model parameters.

In reality, there is no such thing as a truly rigid body, and real joints suffer from imperfections such as friction, compliance and play. Unmodelled dynamics refers to aspects of the physical system that have been ignored in formulating the model. For example, one might choose to ignore the dynamics of the chain in a chain-and-sprocket drive, or one might be forced, by lack of data, to ignore temperature-related or position-dependent variations in the friction at an important joint bearing. Most model parameters refer either to geometry or to inertia. The geometric parameters of a mechanical system are usually known to a reasonable accuracy, but the inertia parameters may be less accurate, or even unknown. If the system can be dismantled, then one can measure the inertias of the individual parts; otherwise, one can use parameter-identification techniques, such as those described in Khalil and Dombre (2002).

10.2   The Sensitivity Problem

Some rigid-body systems can be extremely sensitive to small changes in applied forces, or small changes in model parameters. Sensitivity to geometric parameters is only to be expected in closed-loop systems having special geometries, or general geometries that are close to being special. However, the problem is
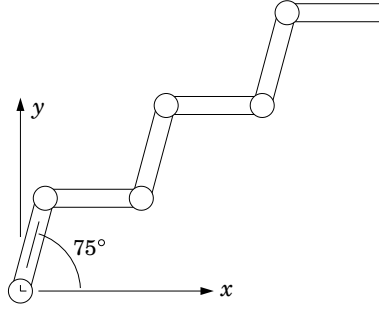
Figure 10.3: A 6-link planar mechanism in a zigzag configuration

broader than this, and, in particular, it is a problem that affects kinematic trees as well as closed-loop systems.

To illustrate the problem, consider the 6-link planar robot mechanism shown in Figure 10.3. Each link has unit length, unit mass, a centre of mass half way along its length, and a rotational inertia about its centre of mass of $1/12$. The joints are revolute, and the joint angles alternate between $+75°$ and $-75°$, as shown. The robot is initially at rest, and there are no forces acting on it other than the joint forces. The equation of motion is therefore

$$\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}}.$$

Suppose we wish to impart a desired acceleration of $\ddot{\boldsymbol{q}}_d = [1\ 1\ 1\ 1\ 1\ 1]^{\mathrm{T}}$ to this robot. The exact force required to do this is

$$\boldsymbol{\tau}_d = \boldsymbol{H}\ddot{\boldsymbol{q}}_d = \begin{bmatrix} 126.4936 \cdots \\ 97.4663 \cdots \\ 69.9762 \cdots \\ 43.7998 \cdots \\ 21.9371 \cdots \\ 6.1646 \cdots \end{bmatrix}.$$

If we apply exactly this force to the robot, then we will get exactly the desired acceleration. Now suppose that the robot's actuators are slightly imperfect, such that the forces they actually apply are equal to the commanded forces rounded to three significant figures. The force that will actually be applied to the robot is then

$$\boldsymbol{\tau}_a = \begin{bmatrix} 126 \\ 97.5 \\ 70.0 \\ 43.8 \\ 21.9 \\ 6.16 \end{bmatrix},$$

and the robot's response to this force will be

$$\ddot{\boldsymbol{q}}_a = \boldsymbol{H}^{-1}\boldsymbol{\tau}_a = \begin{bmatrix} 0.6952 \\ 1.3654 \\ 1.3808 \\ 0.5894 \\ 0.9057 \\ 1.0705 \end{bmatrix}.$$

Observe that some of the accelerations are as much as 40% off their correct values, despite the applied forces all being accurate to better than 0.5%. In effect, the mechanism has amplified the error by more than a factor of 80.

This problem has been studied in Featherstone (2004), where the condition number of $\boldsymbol{H}$ is used as an indication of the underlying sensitivity of the mechanism. This paper shows that the condition number can grow in proportion to the fourth power of the number of bodies, and that the worst-case condition number for a chain of identical links, like the mechanism in Figure 10.3, is approximately $4N_B^4$. The worst-case condition number for the 6-link robot is therefore approximately 5184, although the actual condition number of $\boldsymbol{H}$ in the above example is 725; and the worst case for a 10-link robot is 40,000. Numbers like this indicate that one should be cautious of simulating long chains of rigid bodies, and skeptical of the accuracy of the results.

This paper also investigates branched kinematic trees, and it shows that sensitivity increases with the depth of the tree, other things being kept equal. Thus, a branched kinematic tree will usually have a lower condition number than an unbranched tree made from the same set of bodies. The obvious implication is that one should choose a minimum-depth spanning tree for a closed-loop mechanism.

10.3 Efficiency

To compare the efficiencies of different algorithms, it is necessary to have a common measure of efficiency. In the case of dynamics algorithms, the most widely used measure is the floating-point operations count. This count is usually broken down into two subtotals: the number of multiplications and divisions, which are collectively referred to as multiplications, and the number of additions and subtractions, which are collectively referred to as additions. Furthermore, the subtotals are expressed as functions of a parameter that measures some aspect of the size of the rigid-body system, such as the number of bodies or the number of joint variables.

Some examples of operation-count formulae are shown in Table 10.1. The symbols m and a stand for multiplication and addition, respectively, and n is the number of joint variables. To obtain figures like this, it is necessary to agree on a standard set of rigid-body systems, one for each value of n, so that every algorithm is applied to the same set of systems. The most widely used standard

Algorithm	Cost	Source
RNEA	$(93n - 108)$m $+ (81n - 100)$a	3
RNEA	$(93n - 69)$m $+ (81n - 66)$a	1, 3
RNEA	$(130n - 68)$m $+ (101n - 56)$a	6
RNEA	$(150n - 48)$m $+ (131n - 48)$a	8
CRBA	$(10n^2 + 22n - 32)$m $+ (6n^2 + 37n - 43)$a	7, 14
CRBA	$(10n^2 + 31n - 41)$m $+ (6n^2 + 40n - 46)$a	6
CRBA	$(10n^2 + 41n - 51)$m $+ (6n^2 + 52n - 58)$a	9, 10
CRBA	$(11n^2 + 11n - 42)$m $+ (\frac{11}{2}n^2 + \frac{65}{2}n - 54)$a	2, 4
CRBA	$(12n^2 + 56n - 27)$m $+ (7n^2 + 67n - 53)$a	12
F&S	$(\frac{1}{6}n^3 + \frac{3}{2}n^2 - \frac{2}{3}n)$m $+ (\frac{1}{6}n^3 + n^2 - \frac{7}{6}n)$a	13
ABA	$(224n - 259)$m $+ (205n - 248)$a	11
ABA	$(250n - 222)$m $+ (220n - 198)$a	5
ABA	$(300n - 267)$m $+ (279n - 259)$a	6

Abbreviations
RNEA   recursive Newton-Euler algorithm
CRBA   composite-rigid-body algorithm
F&S   factorize and solve equation of motion $\boldsymbol{H\ddot{q} = \tau - C}$
ABA   articulated-body algorithm

Sources
 1   Balafoutis et al. (1988), Algorithm 3 in Table II
 2   Balafoutis and Patel (1989), Algorithm I in Table II
 3   Balafoutis and Patel (1991), Algorithm 5.7 in Tables 5.1 and 5.2
 4   Balafoutis and Patel (1991), Algorithm 6.2 in Table 6.2
 5   Brandl et al. (1988)
 6   Featherstone (1987), Table 8-6
 7   Featherstone (2005), Eq. 20 with $D_{0a} = n - 1$, $D_{1a} = (n^2 - n)/2$
     and $D_{0b} = D_{1b} = 0$
 8   Hollerbach (1980), Newton-Euler method in Table I
 9   Lilly and Orin (1991), method IV in Table 7
10   Lilly (1993), method IV in Table 3.7
11   McMillan et al. (1995), Table II (fixed base)
12   Walker and Orin (1982), method 3
13   Table 6.6 with Eq. 6.28
14   Eq. 10.3

Table 10.1: Operation counts for several published dynamics algorithms
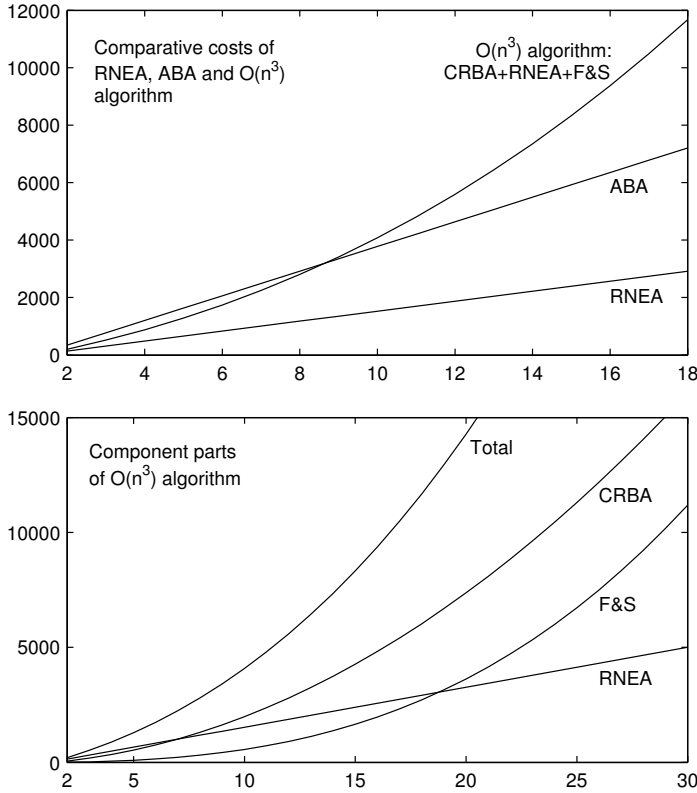
Figure 10.4: Operation counts versus n for various dynamics algorithms

is the set of unbranched kinematic chains in which all the joints are revolute, and all the bodies have general inertia and geometric parameters. For easy reference, let us call this set GU (general unbranched), and call its members $GU(1)$, $GU(2)$, and so on. $GU(n)$ is then the unbranched chain with n joint variables, and therefore also n joints and n moving bodies (i.e., $N_J = N_B = n$).

Given that the joints in GU are revolute, it is understood that one sine and one cosine calculation are required for each joint. As these calculations are the same for all algorithms that need them, they are omitted from the operation counts. The algorithms that need them are the recursive Newton-Euler and articulated-body algorithms.

To give a visual impression of the relative efficiencies of the algorithms, Figure 10.4 shows the total operation count for the best version of each algorithm, plotted against n. It also plots the three component parts of the $O(n^3)$ algorithm, so that their relative contributions can be gauged. Strictly speaking, the curves in these graphs are only correct if the cost of performing an addition is the same as the cost of performing a multiplication. However, in practice

it makes little difference whether one costs an addition as one multiplication, 0.5 multiplications or 0.25 multiplications: apart from the vertical scales, the graphs look almost the same.

According to these graphs, the $O(n^3)$ algorithm is slightly faster than the articulated-body algorithm for $n \leq 8$, and has risen to about 1.6 times the cost of the articulated-body algorithm by the time $n = 18$. The second graph shows that the cost of the composite-rigid-body algorithm (which is $O(n^2)$) is the dominant term in the $O(n^3)$ algorithm from $n = 8$ until well past $n = 30$. (The F&S curve crosses the CRBA curve at approximately $n = 45$.) It also shows that the cost of factorizing and solving the equation of motion is actually the smallest term when $n \leq 18$. The contribution of the n^3 component to the total cost is therefore relatively insignificant at small values of n.

10.3.1   Optimization

The figures quoted in Table 10.1 are the result of extensive optimizations, which are aimed at showing each algorithm in its best possible light. The most important of these optimizations are as follows.[1]

1. Use Denavit-Hartenberg coordinate frames. (See Figure 4.8.) This enables optimizations 2 and 3.

2. Exploit the fact that $\boldsymbol{S}_i = [0\ 0\ 1\ 0\ 0\ 0]^{\mathrm{T}}$ in link coordinates.

3. Implement link-to-link coordinate transformations as a sequence of two axial-screw transforms.

An axial-screw transform is a combination of a rotation about and translation along a single coordinate axis. Such transforms are particularly cheap to apply, and formulae and costings are presented in Section A.4. A transformation from one Denavit-Hartenberg coordinate frame to another can be accomplished by two axial-screw transforms: one about the x axis, and one about the z axis. The former is a constant, while the latter includes the joint variable.

To see how these optimizations are applied, let us calculate the operation-count formula for the composite-rigid-body algorithm. Table 10.2 reproduces the pseudocode of this algorithm (from Table 6.2), together with some annotations. The symbols ra, rx and vx represent the cost of adding two rigid-body inertias, transforming a rigid-body inertia and transforming a vector, respectively. We shall work out these costs in a moment.

The first step is to identify which lines contain floating-point calculations. Assignment and transpose operations do not count as calculations, so there are no floating-point calculations on Lines 1, 3 or 16. Using optimization 2, the

---

[1] One more optimization that ought to be mentioned, although it does not apply to the set GU, is to use planar vectors for calculating the dynamics of planar rigid-body systems. This is particularly beneficial for the articulated-body algorithm, as the number of independent parameters in an articulated-body inertia shrinks from 21 to 6.

	Algorithm:	Annotations:
1	$\boldsymbol{H} = \boldsymbol{0}$	
2	**for** $i = 1$ **to** N_B **do**	loop 1
3	$\quad \boldsymbol{I}_i^{\mathrm{c}} = \boldsymbol{I}_i$	
4	**end**	
5	**for** $i = N_B$ **to** 1 **do**	loop 2
6	$\quad$ **if** $\lambda(i) \neq 0$ **then**	
7	$\quad\quad \boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} = \boldsymbol{I}_{\lambda(i)}^{\mathrm{c}} + {}^{\lambda(i)}\boldsymbol{X}_i^* \, \boldsymbol{I}_i^{\mathrm{c}} \, {}^i\boldsymbol{X}_{\lambda(i)}$	operations: ra + rx
8	$\quad$ **end**	
9	$\quad \boldsymbol{F} = \boldsymbol{I}_i^{\mathrm{c}} \, \boldsymbol{S}_i$	$\boldsymbol{F}$ = column 3 of $\boldsymbol{I}_i^{\mathrm{c}}$
10	$\quad \boldsymbol{H}_{ii} = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{F}$	$\boldsymbol{H}_{ii}$ = element 3 of $\boldsymbol{F}$
11	$\quad j = i$	
12	$\quad$ **while** $\lambda(j) \neq 0$ **do**	loop 3
13	$\quad\quad \boldsymbol{F} = {}^{\lambda(j)}\boldsymbol{X}_j^* \boldsymbol{F}$	operations: vx
14	$\quad\quad j = \lambda(j)$	
15	$\quad\quad \boldsymbol{H}_{ij} = \boldsymbol{F}^{\mathrm{T}} \boldsymbol{S}_j$	$\boldsymbol{H}_{ij}$ = element 3 of $\boldsymbol{F}$
16	$\quad\quad \boldsymbol{H}_{ji} = \boldsymbol{H}_{ij}^{\mathrm{T}}$	
17	$\quad$ **end**	
18	**end**	

Table 10.2: Calculations performed by the composite-rigid-body algorithm

multiplication $\boldsymbol{I}_i^{\mathrm{c}} \, \boldsymbol{S}_i$ on line 9 simplifies to extracting column 3 from $\boldsymbol{I}_i^{\mathrm{c}}$, which does not require any calculation. Likewise, the multiplications $\boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{F}$ and $\boldsymbol{F}^{\mathrm{T}} \boldsymbol{S}_j$ on lines 10 and 15 both simplify to extracting element 3 from $\boldsymbol{F}$, which does not require any calculation. Thus, there are only two lines where floating-point calculations do occur: line 7, which performs the operations ra and rx, and line 13, which performs the operation vx.

If this algorithm is applied to the mechanism $GU(n)$, then the body of loop 2 will be executed n times. The condition $\lambda(i) \neq 0$ will be true for every value of i except $i = 1$, so line 7 will be executed a total of $n - 1$ times. On each iteration of loop 2, the body of loop 3 will be executed $i - 1$ times (because $\lambda(j) = j - 1$); so line 13 will be executed a total of $\sum_{i=1}^{n} (i - 1)$ times, which works out as $n(n - 1)/2$ times. The cost formula for the composite-rigid-body algorithm is therefore

$$cost = (n - 1)(\mathrm{ra} + \mathrm{rx}) + \frac{n(n - 1)}{2} \mathrm{vx} \, . \tag{10.2}$$

The next step is to express ra, rx and vx in terms of m and a. If rigid-body inertias are stored in compact form, as explained in Section A.3, then the cost of summing two inertias is ten floating-point additions; so ra = 10a. To obtain the lowest operation counts for rx and vx, we use optimizations 1 and 3. From the figures in Section A.4, the cost of a single axial-screw transform of a spatial

vector is $10\mathsf{m}+6\mathsf{a}$, but vx requires two, so $\mathsf{vx}=20\mathsf{m}+12\mathsf{a}$. The basic cost of an axial-screw transform of a rigid-body inertia is $16\mathsf{m}+15\mathsf{a}$, to which $2\mathsf{m}+3\mathsf{a}$ must be added each time the angle changes. However, we can shave $1\mathsf{m}$ off this figure by precomputing the products of the masses with the distance parameters of the screw transforms, since they are all constants. This results in a cost figure of $2(15\mathsf{m}+15\mathsf{a})+2\mathsf{m}+3\mathsf{a}=32\mathsf{m}+33\mathsf{a}$ for rx. Substituting these costs into Eq. 10.2 gives

$$cost = (10n^2 + 22n - 32)\mathsf{m} + (6n^2 + 37n - 43)\mathsf{a}\,. \qquad (10.3)$$

This formula is as far as we will take the optimization process, but there are further improvements in efficiency that could be made. For example, modest reductions in the costs of both ra and rx can be achieved by modifying the inertia parameters as described in Section 9.7. Another cost saving can be obtained by observing that only element 3 of $\boldsymbol{F}$ is needed to calculate $\boldsymbol{H}_{ij}$, so the last execution of line 13 in each invocation of loop 3 could be replaced by a simpler calculation that only calculates element 3 of the transformed vector.

Even this is not yet the last word in efficiency. However, it is not really worthwhile to implement such fiddly optimizations manually. A better strategy is to get the computer to do the optimization for you, and that is the subject of Section 10.4.

10.3.2   Branches

One of the problems with using GU as the benchmark set of rigid-body systems is that it does not contain any branched trees, and therefore does not allow the effect of branches to be seen. It turns out that branches have very little effect on the recursive Newton-Euler and articulated-body algorithms, but they can greatly reduce the cost of the composite-rigid-body algorithm, as well as the cost of factorizing and solving the equation of motion. In both cases, the cost savings are caused by branch-induced sparsity in the joint-space inertia matrix. (See §6.4.)

It therefore follows that the cost formulae in Table 10.1 and the curves in Figure 10.4 present the $O(n^3)$ algorithm in an unduly pessimistic light, by stating only its worst-case performance. Indeed, even the name '$O(n^3)$ algorithm' is misleading, since the true complexity of this algorithm is $O(nd^2)$, where d is the depth of the connectivity tree. If there are no branches, then $d = n$; but if there are branches, then d can be much smaller than n; and if d is subject to a fixed upper limit, then the complexity of this algorithm is only $O(n)$.

Let us recalculate the cost formula for the composite-rigid-body algorithm, assuming that it is being applied to a general kinematic tree with n revolute joints, and that the joint axes satisfy $\boldsymbol{S}_i = [0\ 0\ 1\ 0\ 0\ 0]^\mathrm{T}$ in link coordinates. Referring back to Table 10.2, it is still the case that lines 7 and 13 are the only two that perform floating-point calculations, but the number of times these lines get executed has changed. In the case of line 7, the expression $\lambda(i) \neq 0$

is false for every body i that is a child of body 0; that is, for every i satisfying $i \in \mu(0)$. Line 7 is therefore executed $n - |\mu(0)|$ times. In the case of line 13, loop 3 iterates over every ancestor of body i, starting with body i itself, and terminating on the ancestor nearest the root (which is the ancestor satisfying $j \in \mu(0)$). The body of loop 3 is therefore executed $|\kappa(i)| - 1$ times on each iteration of loop 2; and so line 13 is executed a total of $\sum_{i=1}^{n}(|\kappa(i)| - 1)$ times.

Let us define the quantities D_0 and D_1 as follows:

$$D_0 = n - |\mu(0)| \qquad (10.4)$$

and

$$D_1 = \sum_{i=1}^{n}(|\kappa(i)| - 1) \,. \qquad (10.5)$$

The cost of the composite-rigid-body algorithm can then be expressed as

$$cost = D_0(\mathsf{ra} + \mathsf{rx}) + D_1\mathsf{vx} \,. \qquad (10.6)$$

Furthermore, as $|\mu(0)|$ is bounded by $1 \le |\mu(0)| \le n$, and $|\kappa(i)|$ by $1 \le |\kappa(i)| \le i$, it follows that D_0 and D_1 are bounded by

$$0 \le D_0 \le n - 1 \qquad (10.7)$$

and

$$0 \le D_1 \le n(n-1)/2 \,. \qquad (10.8)$$

Both quantities reach their lower limit in a system where every body is connected directly to the base. In such a system, $\boldsymbol{H}$ is diagonal and constant, and the cost of calculating it is zero. At the other extreme, both quantities reach their upper limit in an unbranched kinematic tree. In this case, Eq. 10.6 is identical to Eq. 10.2. If d is the depth of the connectivity tree, then $|\kappa(i)|$ is also bounded by $|\kappa(i)| \le d$, which implies

$$0 \le D_1 \le n(d-1) \,. \qquad (10.9)$$

Thus, the true complexity of the composite-rigid-body algorithm is actually $O(nd)$, and the expression $O(n^2)$ only applies in the worst case. If we combine this with the $O(nd^2)$ complexity of the factorization algorithms in Section 6.5, then the complete forward-dynamics algorithm has a complexity of $O(nd^2)$.

Let us now express the cost formula in Eq. 10.6 in terms of m and a. At this point, a new complication arises: only one child per nonterminal body can benefit from the cost savings afforded by Denavit-Hartenberg coordinate frames. We therefore rewrite Eq. 10.6 as follows:

$$cost = D_0\mathsf{ra} + D_0^{dh}\mathsf{rx}^{dh} + D_0^{g}\mathsf{rx}^{g} + D_1^{dh}\mathsf{vx}^{dh} + D_1^{g}\mathsf{vx}^{g} \,, \qquad (10.10)$$

where the superscripts dh and g refer to Denavit-Hartenberg and general coordinate transforms, respectively. So rx^{dh} is the cost of calculating ${}^{\lambda(i)}\boldsymbol{X}_i^{*}\,{}^{c}\boldsymbol{I}_i\,{}^{i}\boldsymbol{X}_{\lambda(i)}$

when ${}^i\boldsymbol{X}_{\lambda(i)}$ is a Denavit-Hartenberg transform (two axial screws); vx^g is the cost of calculating ${}^{\lambda(j)}\boldsymbol{X}_j^* \boldsymbol{F}$ when ${}^{\lambda(j)}\boldsymbol{X}_j^*$ is a general coordinate transform; D_0^g is the number of times that rx^g is performed; and so on. Note that $D_0^{dh} + D_0^g = D_0$ and $D_1^{dh} + D_1^g = D_1$.

To obtain expressions for the various D quantities, we introduce two new sets: π^{dh} and π^g. π^{dh} is the set of all bodies i such that $\lambda(i) \neq 0$ and ${}^i\boldsymbol{X}_{\lambda(i)}$ is a Denavit-Hartenberg transform; and π^g is the set of all bodies i such that $\lambda(i) \neq 0$ and ${}^i\boldsymbol{X}_{\lambda(i)}$ is a general transform. Thus, $\pi^{dh} \cup \pi^g$ is the set of all bodies that are not children of the root node (i.e., not elements of $\mu(0)$), and $|\pi^{dh} \cup \pi^g| = n - |\mu(0)| = D_0$. Given these sets, we can define the D quantities as follows:

$$D_0^{dh} = |\pi^{dh}|, \qquad D_1^{dh} = \sum_{i \in \pi^{dh}} |\nu(i)|,$$

$$D_0^g = |\pi^g|, \qquad D_1^g = \sum_{i \in \pi^g} |\nu(i)|. \tag{10.11}$$

The expressions for D_1^{dh} and D_1^g are obtained as follows. When we first evaluated D_1, we asked the question 'How many times is loop 3 executed?' Another approach would be to ask how many times ${}^{\lambda(j)}\boldsymbol{X}_j^*$ is used, for each possible value of j. The answer to this question is that if $\lambda(j) = 0$ then this transform is never used; otherwise, it is used once for each body in $\nu(j)$, which means it is used $|\nu(j)|$ times. The expressions in Eq. 10.11 are simply the sums of these usage counts.

The final step is to express ra, rx^{dh}, etc., in terms of m and a. From Sections A.3 and A.4, these expressions are: $\mathsf{ra} = 10\mathsf{a}$, $\mathsf{rx}^{dh} = 32\mathsf{m} + 33\mathsf{a}$, $\mathsf{rx}^g = 47\mathsf{m} + 48\mathsf{a}$ (three axial screws), $\mathsf{vx}^{dh} = 20\mathsf{m} + 12\mathsf{a}$ and $\mathsf{vx}^g = 24\mathsf{m} + 18\mathsf{a}$. Substituting these expressions into Eq. 10.10 gives

$$\begin{aligned} cost = \; & D_0^{dh}(32\mathsf{m} + 43\mathsf{a}) + D_0^g(47\mathsf{m} + 58\mathsf{a}) + \\ & D_1^{dh}(20\mathsf{m} + 12\mathsf{a}) + D_1^g(24\mathsf{m} + 18\mathsf{a}) \,. \end{aligned} \tag{10.12}$$

This is the cost of the composite-rigid-body algorithm when it is applied to a general kinematic tree in which all the joints are revolute. On an unbranched tree, we have $D_0^{dh} = n - 1$, $D_1^{dh} = n(n-1)/2$ and $D_0^g = D_1^g = 0$, which makes Eq. 10.12 the same as Eq. 10.3. More on this subject, including a cost formula for floating-base kinematic trees, can be found in Featherstone (2005).

To give an idea of how big a difference branches can make, Figure 10.5 reproduces the cost curves from Figure 10.4 for the articulated-body and $O(n^3)$ algorithms, the latter now labelled $O(nd^2)$, and it adds one new curve: the cost of applying the $O(nd^2)$ algorithm to a binary tree. This curve is almost identical to the curve for the articulated-body algorithm, and very different from the curve for the $O(nd^2)$ algorithm applied to an unbranched chain. The reason for this is that d is $O(\log(n))$ on a binary tree, so the complexity of the $O(nd^2)$ algorithm is only $O(n \log(n)^2)$ on the binary tree, which is very close to being $O(n)$.
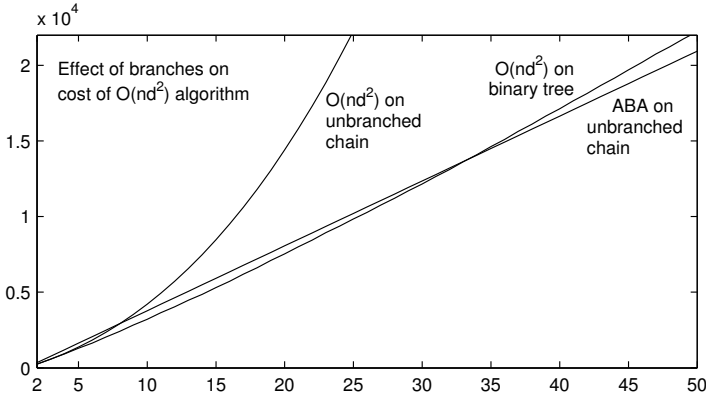
Figure 10.5: Effect of branches on the cost of the $O(nd^2)$ algorithm

10.4   Symbolic Simplification

The advantage of model-based dynamics software is that each algorithm can be implemented once, and the resulting code will work on any rigid-body system that is supplied to it as an argument. The disadvantage is that the code must cater for every possibility that is allowed by the system model, and cannot take short cuts that depend on prior knowledge of the system. In particular, many of the parameters in a typical system will be zero, and the model-based software will spend much time calculating terms that are bound to evaluate to zero because one of their factors is a zero-valued model parameter.

The alternative is to write custom code that is designed to work only on a single rigid-body system, or a narrow class of similar rigid-body systems. Such code can exploit prior knowledge of joint types, geometric and inertia parameter values, and so on, with the end result that fewer arithmetic operations are required to perform the calculation, as compared with a model-based coding of the same algorithm.

The manual construction of custom code is not practical, except for the simplest of rigid-body systems; but the automatic construction of such code is entirely practical, and the procedure for doing so is called *symbolic simplification*. Figure 10.6 illustrates the process. In the first stage, an algorithm and a system model are fed to a symbolic executer. This is a program that applies the algorithm to the data, but, instead of actually performing the numerical calculations, it merely records them, and outputs the record as a data file. The contents of this file are a list of assignment statements detailing the floating-point calculations that the given algorithm would have performed on the given model if it had been executed for real. This file also identifies the input and output variables.

In the second stage, the assignment-statement file is passed to a symbolic simplifier program. This program reads the file, applies a few basic simplifica-
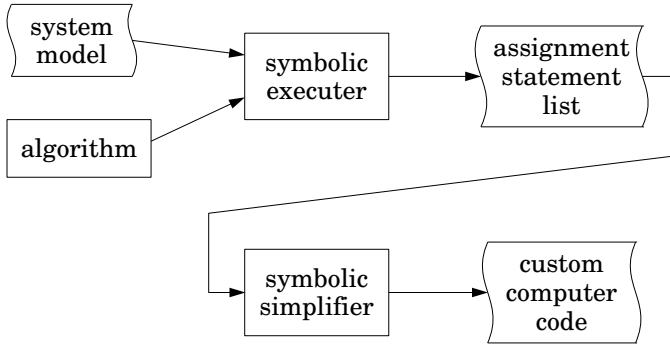
Figure 10.6: The symbolic simplification process

tion rules, and outputs the result in the form of computer source code expressed
in a suitable programming language. The most important simplifications are
as follows:

- Remove any assignment statement that calculates a quantity that is never
  used. (Being output counts as being used.)

- Remove assignment statements of the form $a = b$, where a is not an
  output variable, and b is either a constant or a variable, and replace all
  subsequent instances of a with b.

- Perform some basic algebraic simplifications, such as: $x \cdot 0 \to 0$, $x \cdot 1 \to x$,
  $x + 0 \to x$, and so on.

- Evaluate constant expressions.

These rules must be applied exhaustively, as the application of one rule can
create new opportunities for the application of another. They must also be
cheap, as there can easily be tens of thousands of assignments in the list.

Table 10.3 presents an example of how the process works. In this example,
v is an input variable, τ_{out} is an output variable, and I and s are quantities
defined in the system model. The algorithm specifies how to calculate τ_{out}
from v. If we were to implement this algorithm directly, then it would require
one $3 \times 3 \times 1$ matrix multiplication, one 3D vector cross product and one 3D
vector dot product. This adds up to a total of 29 floating-point arithmetic
operations, which can be seen in the assignment-statement file. The symbolic
executer deals only with scalars, so it expands vector operations into their
scalar equivalents. It also processes only one vector operation at a time, so it
introduces temporary variables as required. Apart from that, the only other
thing it does is to substitute model parameters with their actual values.

Using only the simplifications listed above, the simplifier is able to cut the
total operations count from 29 down to 5. If the simplifier had used a more

calculation specified in the algorithm	$p = v \times I\,v$ $\tau_{out} = p \cdot s$
data supplied in the model	$I = \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.6 & 0 \\ 0 & 0 & 1.7 \end{bmatrix}, \quad s = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
contents of the assignment-statement file	``` tmp001 = 1.5 * v(1) + 0 * v(2) + 0 * v(3) tmp002 = 0 * v(1) + 1.6 * v(2) + 0 * v(3) tmp003 = 0 * v(1) + 0 * v(2) + 1.7 * v(3) p(1) = v(2) * tmp003 - v(3) * tmp002 p(2) = v(3) * tmp001 - v(1) * tmp003 p(3) = v(1) * tmp002 - v(2) * tmp001 tau_out = p(1) * 0 + p(2) * 0 + p(3) * 1 ```
contents of the source-code file	``` tmp001 = 1.5 * v(1) tmp002 = 1.6 * v(2) p(3) = v(1) * tmp002 - v(2) * tmp001 tau_out = p(3) ```

Table 10.3: Symbolic simplification example

complex and sophisticated set of rules, then it might have found the optimal solution, which is $\tau_{out} = 0.1v_1v_2$, but the process would have required much more CPU time. Even so, the simple rule set is enough to find 90% of the possible cost savings. In this example, the simplifier cut the cost by almost a factor of 6. In practice, this factor is highly sensitive to the number of zeros in the system model. Factors as high as 10 have been reported in the literature (Wittenburg and Wolz, 1985); but factors close to 1 are also possible.

Several variations are possible on the above scheme. For example, a set of predefined algorithms can be built into the symbolic executer, so that the user only has to supply the system model. This makes the software easier to use, and easier to implement, but it removes the user's freedom to supply an algorithm of his own. Another variation is that the symbolic executer and simplifier could be two parts of the same program, rather than two separate programs. In this case, the assignment-statement file would not exist, although an equivalent internal data structure would still be required. Another possibility is to allow the model to define only some of the parameter values. The undefined parameters would then be treated as unknown quantities to be supplied at run time. This feature allows the creation of custom code that caters for a set of rigid-body systems, each differing from the others only in the values of specific parameters.

One potential disadvantage of symbolic simplification is that it reduces vector operations to scalar operations. On a computer with vector-arithmetic hardware, the resulting code may make poor use of the available processing power,

and could end up slower than the original vector-oriented code. Another possible disadvantage is that the size of the customized code grows linearly with the operations count (because it is just a list of assignment statements). For a sufficiently complicated rigid-body system, the custom code will be too large to fit in the computer's instruction cache. When this happens, the computer is forced to fetch instructions from main memory, which is a much slower process. The net effect is that execution speed goes down because the CPU is repeatedly having to wait for the next batch of instructions.

Several programs have been written that implement this technique, or something similar, and two have become commercially available: SD/FAST and Autolev.[2] More on the subject of automatic generation and simplification of dynamics equations can be found in Murray and Neuman (1984); Neuman and Murray (1987); Wittenburg and Wolz (1985).

---

[2]At the time of writing, further information on these two programs could be found at http://www.autolev.com and http://www.sdfast.com.

Chapter 11

Contact and Impact

When two rigid bodies come into contact, they become subject to a *contact constraint* which says that the two bodies are not allowed to penetrate. If ϕ is a measure of the signed distance between them, meaning that $\phi < 0$ if they overlap, then the contact constraint is given by the inequality $\phi \geq 0$. The forces that impose this constraint are similarly one-sided: they can act to prevent penetration, but not to prevent separation—they can repel, but not attract. If two bodies meet with velocities that are not consistent with the contact constraint between them, then an impulse is generated that causes a step change in their velocities. We call this event an *impact*.

The first four sections in this chapter develop the equations of motion for rigid-body systems containing point contacts. Then Section 11.5 examines ways of solving these equations, and Section 11.6 looks at some of the geometric aspects of the problem, including how to represent line and surface contacts in terms of point contacts. Section 11.7 develops the impulsive equations of motion for rigid-body systems, and the equations of impact between two bodies. Finally, Section 11.8 presents the alternative to rigid-body contact and impact dynamics, which is to incorporate compliance into the contacting surfaces.

11.1 Single Point Contact

Consider the rigid-body system shown in Figure 11.1. It contains a single rigid body, B, which is making contact with a fixed surface at a single point, C. The body has a velocity of $\boldsymbol{v}$ and an acceleration of $\boldsymbol{a}$; and it is subject to both an applied force of $\boldsymbol{f}$ and a contact force of $\boldsymbol{f}_c$. The contact is characterized by a contact normal vector, $\boldsymbol{n} \in \mathsf{F}^6$, which is a unit force acting along the line of the contact normal. We define $\boldsymbol{n}$ to point out of the surface so that positive scalar multiples of $\boldsymbol{n}$ will repel the body from the surface. The contact is assumed to be frictionless; and the body is assumed to be subject only to finite forces (i.e., no impulses) during and immediately before and after the current instant. The
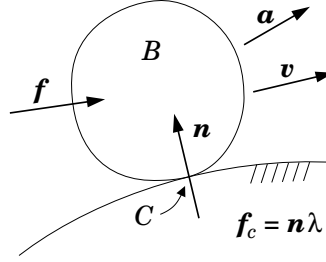
Figure 11.1: A rigid body making contact with a fixed surface at a single point

two unknowns in this system are $\boldsymbol{f}_c$ and $\boldsymbol{a}$, and the problem is to find them.

Let C' denote the point on the body's surface that coincides with C at the current instant. The fixed surface exerts a constraint force of $\boldsymbol{f}_c$ on the body, which acts to prevent C' from penetrating into the surface. As the contact is frictionless, this force must act along the contact normal, and must therefore be a scalar multiple of $\boldsymbol{n}$. We therefore introduce a new scalar variable, λ, which is the unknown contact force magnitude. Thus,

$$\boldsymbol{f}_c = \boldsymbol{n}\,\lambda\,. \tag{11.1}$$

As $\boldsymbol{f}_c$ can act only to repel the body, λ must satisfy the constraint $\lambda \geq 0$.

We now introduce a second scalar variable, ζ, to denote the *contact separation velocity*, which is defined as the normal component of the linear velocity of C'. It is given by the expression

$$\zeta = \boldsymbol{n}\cdot\boldsymbol{v}\,. \tag{11.2}$$

The correctness of this equation can be verified by placing a coordinate frame with its origin at C and its z axis along the normal: the Plücker coordinates of $\boldsymbol{n}$ will then be $[0\ 0\ 0\ 0\ 0\ 1]^{\mathrm{T}}$, so $\boldsymbol{n}\cdot\boldsymbol{v} = v_{Cz}$, which is the z coordinate of the linear velocity of C'.

It can be shown that $\zeta = 0$ at the current instant. This implies that the given values of $\boldsymbol{n}$ and $\boldsymbol{v}$ must satisfy $\boldsymbol{n}\cdot\boldsymbol{v} = 0$. The argument proceeds as follows: if the body is subject only to finite forces, then its acceleration must be finite; so, if $\zeta < 0$ at the current instant, then the body will penetrate the surface in the immediate future; and if $\zeta > 0$ then the body has penetrated the surface in the immediate past; but penetration is not allowed at any time, so $\zeta = 0$. A corollary to this argument is that $\zeta \neq 0$ implies an impulse.

We are also interested in the *contact separation acceleration*, which is the derivative of ζ. This quantity is given by

$$\dot{\zeta} = \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{n}\cdot\boldsymbol{v}) = \boldsymbol{n}\cdot\boldsymbol{a} + \dot{\boldsymbol{n}}\cdot\boldsymbol{v}\,, \tag{11.3}$$

and its value at the current instant is unknown. The term $\dot{\boldsymbol{n}}\cdot\boldsymbol{v}$ (or the vector $\dot{\boldsymbol{n}}$) is assumed to be known, since it is a function of the position and velocity

of B and the shapes of the contacting surfaces. Having established that $\zeta = 0$, it can be seen that $\dot{\zeta}$ cannot be negative, since that would lead to penetration, so it must satisfy the constraint $\dot{\zeta} \geq 0$.

The exact behaviour of the contact constraint can now be stated as follows: either the contact breaks at the current instant, or it persists. If it breaks, then the separation acceleration must be strictly positive, and the contact force must be zero (because contact is being lost). If it persists, then the separation acceleration must be zero. To summarize, we have

$$\dot{\zeta} > 0 \quad \text{and} \quad \lambda = 0 \qquad \text{if the contact breaks, and}$$
$$\dot{\zeta} = 0 \quad \text{and} \quad \lambda \geq 0 \qquad \text{if the contact persists.}$$

This same behaviour is described succinctly by the constraint equation

$$\dot{\zeta} \geq 0, \quad \lambda \geq 0, \quad \dot{\zeta}\lambda = 0. \tag{11.4}$$

The next step is to express $\dot{\zeta}$ as a function of λ. This is where the dynamics of the body come into play. The equation of motion for B is

$$\boldsymbol{I}\boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v} = \boldsymbol{f} + \boldsymbol{n}\lambda, \tag{11.5}$$

where $\boldsymbol{f}$ accounts for every force acting on the body other than the contact force. Expressing $\boldsymbol{a}$ as a function of λ, we have

$$\boldsymbol{a} = \boldsymbol{I}^{-1}(\boldsymbol{n}\lambda + \boldsymbol{f} - \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}). \tag{11.6}$$

Substituting this expression into Eq. 11.3 gives

$$\begin{aligned}
\dot{\zeta} &= \boldsymbol{n} \cdot \boldsymbol{I}^{-1}(\boldsymbol{n}\lambda + \boldsymbol{f} - \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}) + \dot{\boldsymbol{n}} \cdot \boldsymbol{v} \\
&= M\lambda + d,
\end{aligned} \tag{11.7}$$

where

$$M = \boldsymbol{n} \cdot \boldsymbol{I}^{-1}\boldsymbol{n} \tag{11.8}$$

and

$$d = \boldsymbol{n} \cdot \boldsymbol{I}^{-1}(\boldsymbol{f} - \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}) + \dot{\boldsymbol{n}} \cdot \boldsymbol{v}. \tag{11.9}$$

Equations 11.4 and 11.7 can now be solved by inspection for $\dot{\zeta}$ and λ, the solution being

$$\begin{aligned}
\dot{\zeta} = d \quad \text{and} \quad \lambda = 0 \qquad &\text{if } d \geq 0 \\
\dot{\zeta} = 0 \quad \text{and} \quad \lambda = -d/M \qquad &\text{if } d < 0.
\end{aligned} \tag{11.10}$$

(Note that M is strictly positive because $\boldsymbol{I}$ is positive definite.) Having solved for λ, the values of $\boldsymbol{f}_c$ and $\boldsymbol{a}$ follow from Eqs. 11.1 and 11.6, and the problem is solved. Observe that d is a linear function of $\boldsymbol{f}$, but λ is only a piecewise-linear function of d. Thus, the acceleration of B is not a linear function of the applied force, but only a piecewise-linear function. In this respect, contact dynamics differs fundamentally from the dynamics of systems with equality constraints.
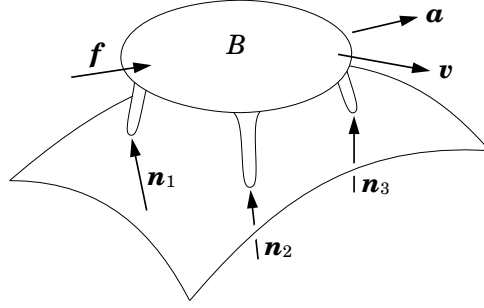
Figure 11.2: A rigid body making multiple point contacts with a fixed surface

11.2   Multiple Point Contacts

We now extend the analysis in Section 11.1 to the case of a single rigid body making multiple frictionless point contacts with a fixed surface. An example of such a system is shown in Figure 11.2. Let there be n_c contacts in total, numbered from 1 to n_c; and let contact i be characterized by a contact point, C_i, and a contact normal vector, $\boldsymbol{n}_i$, as defined in Section 11.1. We also define C'_i to be the body-fixed point that coincides with C_i at the current instant.

The fixed surface can exert a constraint force on the body through each contact. The force exerted through contact i is $\boldsymbol{n}_i \lambda_i$, where λ_i is the unknown contact force magnitude, and this force acts to prevent C'_i from penetrating into the surface. As the contact force can act only to repel C'_i, it must be a positive multiple of $\boldsymbol{n}_i$; so λ_i must satisfy the constraint $\lambda_i \geq 0$. If $\boldsymbol{f}_c$ is the total constraint force acting on the body, then

$$\boldsymbol{f}_c = \sum_{i=1}^{n_c} \boldsymbol{n}_i \, \lambda_i \,, \tag{11.11}$$

which replaces Eq. 11.1.

Let ζ_i denote the separation velocity at contact i. This quantity is the component of the linear velocity of C'_i in the direction of $\boldsymbol{n}_i$, and is therefore given by

$$\zeta_i = \boldsymbol{n}_i \cdot \boldsymbol{v} \,. \tag{11.12}$$

Observe that each contact has its own separation velocity, even though there is only a single moving body in the system. Using the same argument as in Section 11.1, we can show that every separation velocity must be zero at the current instant. This implies that the body's velocity must satisfy the following set of equations:

$$\boldsymbol{n}_i \cdot \boldsymbol{v} = 0 \qquad (i = 1, \ldots, n_c) \,.$$

If the contact normals span the whole of F^6 then the only solution will be $\boldsymbol{v} = \boldsymbol{0}$. However, this does not necessarily mean that the body cannot move, since it

may be able to accelerate away from (and therefore break) one or more of the contacts.

The separation acceleration at contact i is the derivative of the separation velocity, and is therefore given by

$$\dot{\zeta}_i = \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{n}_i \cdot \boldsymbol{v}) = \boldsymbol{n}_i \cdot \boldsymbol{a} + \dot{\boldsymbol{n}}_i \cdot \boldsymbol{v}\,, \tag{11.13}$$

which replaces Eq. 11.3. Given that $\zeta_i = 0$, the value of $\dot{\zeta}_i$ cannot be negative, since that would lead to penetration locally at contact i, so we have $\dot{\zeta}_i \geq 0$.

The exact behaviour of this set of contacts can now be stated as follows: at the current instant, each contact either breaks or it persists. If contact i breaks, then $\dot{\zeta}_i > 0$ and $\lambda_i = 0$, otherwise $\dot{\zeta}_i = 0$ and $\lambda_i \geq 0$. Thus, we have

$$\dot{\zeta}_i \geq 0\,, \quad \lambda_i \geq 0\,, \quad \dot{\zeta}_i \lambda_i = 0 \qquad (i = 1, \ldots, n_c)\,, \tag{11.14}$$

which replaces Eq. 11.4. At the outset, we do not know which contacts will break and which will persist. Finding this out is part of the solution process.

At this point, let us introduce three new quantities, $\boldsymbol{N}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\zeta}$, which are defined as follows:

$$\boldsymbol{N} = \begin{bmatrix} \boldsymbol{n}_1 & \boldsymbol{n}_2 & \cdots & \boldsymbol{n}_{n_c} \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{n_c} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\zeta} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_{n_c} \end{bmatrix}. \tag{11.15}$$

$\boldsymbol{N}$ is therefore a $6 \times n_c$ matrix, and the other two are n_c-element vectors. In terms of these quantities, Eqs. 11.11, 11.13 and 11.14 can be written

$$\boldsymbol{f}_c = \boldsymbol{N}\boldsymbol{\lambda}\,, \tag{11.16}$$

$$\dot{\boldsymbol{\zeta}} = \boldsymbol{N}^{\mathrm{T}}\boldsymbol{a} + \dot{\boldsymbol{N}}^{\mathrm{T}}\boldsymbol{v} \tag{11.17}$$

and

$$\dot{\boldsymbol{\zeta}} \geq \boldsymbol{0}\,, \quad \boldsymbol{\lambda} \geq \boldsymbol{0}\,, \quad \dot{\boldsymbol{\zeta}}^{\mathrm{T}}\boldsymbol{\lambda} = 0\,. \tag{11.18}$$

An expression of the form $\boldsymbol{a} \geq \boldsymbol{b}$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors, means that $a_i \geq b_i$ for every value of the index i. The constraint $\dot{\boldsymbol{\zeta}}^{\mathrm{T}}\boldsymbol{\lambda} = 0$, in combination with the other two constraints in Eq. 11.18, implies that $\dot{\zeta}_i \lambda_i = 0$ for every i.

Continuing with the analysis, the equation of motion for body B is

$$\boldsymbol{I}\boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v} = \boldsymbol{f} + \boldsymbol{N}\boldsymbol{\lambda}\,,$$

which can be rearranged to express $\boldsymbol{a}$ as a function of $\boldsymbol{\lambda}$:

$$\boldsymbol{a} = \boldsymbol{I}^{-1}(\boldsymbol{N}\boldsymbol{\lambda} + \boldsymbol{f} - \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v})\,. \tag{11.19}$$

Substituting this expression into Eq. 11.17 gives

$$\dot{\boldsymbol{\zeta}} = \boldsymbol{M}\boldsymbol{\lambda} + \boldsymbol{d}\,, \tag{11.20}$$
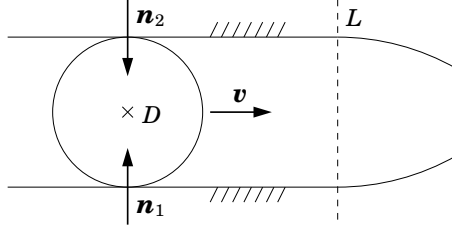
Figure 11.3: Example of a contact system with no solution

where
$$M = N^{\mathrm{T}} I^{-1} N \tag{11.21}$$
and
$$d = N^{\mathrm{T}} I^{-1} (f - v \times^* I v) + \dot{N}^{\mathrm{T}} v . \tag{11.22}$$

The acceleration of B can now be found by first solving Eqs. 11.18 and 11.20 for λ, and then calculating a from Eq. 11.19. However, unlike Eqs. 11.4 and 11.7, which could be solved by inspection, the solution of Eqs. 11.18 and 11.20 requires a special algorithm. This topic is discussed in Section 11.5, but a few remarks are in order here.

1. If $d \geq 0$ then one possible solution is $\lambda = 0$.

2. If M is positive-definite then there is exactly one solution; otherwise, there could be one solution, no solution, or infinitely many solutions.

3. If there are multiple solutions for λ then they all produce the same value for a.

It can be seen from Eq. 11.21 that M is symmetric and positive-semidefinite; but it will only be positive-definite if the contact normal vectors are linearly independent. Thus, the possibility that there might be no solution, or that there might be more than one solution for λ, can only arise if the contact normals are linearly dependent.

Example 11.1 *The possibility that there might be no solution to Eqs. 11.18 and 11.20 implies that there exist circumstances in which no value of λ can prevent a violation of the contact constraints. A 2D example of this phenomenon is shown in Figure 11.3. This figure shows a disc sliding in a slot. To the left of the line L, the slot's walls are straight, and the width of the slot equals the diameter of the disc. The disc therefore makes two point contacts with the slot, with contact normals n_1 and n_2 as shown. These vectors satisfy $n_2 = -n_1$, and are therefore linearly dependent. To the right of L, the slot walls are circular arcs that curve inward, making the slot narrower. Thus, the disc's centre point, D, can lie on or to the left of L, but it cannot pass to the right of L because the slot is too narrow.*

Consider the situation that arises when D lies on L: the disc cannot proceed any further to the right, but the two contact normals are pointing straight up and down, and are therefore incapable of affecting the horizontal motion of the disc. If a force is applied to the disc that pushes it to the right, then we have a paradoxical situation in which rightward motion must not occur, but the contact forces cannot prevent it. If this situation arises in a simulation, then the practical solution is to allow D to move slightly to the right of L, so that the two contact normals change direction and become able to resist any further motion to the right.

If D reaches L with a linear velocity of $\boldsymbol{v}$, as shown, then an impulse is required to arrest the motion of the disc. Again, the simulator must allow D to pass slightly beyond L before this impulse can be applied. In both cases, the simulator must tolerate a small amount of penetration between the disc and the slot.

Given that $\boldsymbol{n}_2 = -\boldsymbol{n}_1$, it can be shown that $\boldsymbol{M}$ takes the form $\begin{bmatrix} m & -m \\ -m & m \end{bmatrix}$, where m is a positive scalar. It therefore follows (from Eq. 11.20) that

$$\dot{\zeta}_1 + \dot{\zeta}_2 = d_1 + d_2 \,.$$

If a solution exists, then it must satisfy $\dot{\zeta}_1 + \dot{\zeta}_2 \geq 0$; so the condition for there to be no solution is $d_1 + d_2 < 0$.

11.3   A Rigid-Body System with Contacts

Let us now consider the case of a general rigid-body system with multiple frictionless point contacts. The big difference between this and the previous case is that the equation of motion for a general rigid-body system is expressed in joint space.[1] Thus, the equation of motion for this system will take the form

$$\boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C} = \boldsymbol{\tau} + \boldsymbol{\tau}_c \,, \tag{11.23}$$

where $\boldsymbol{\tau}_c$ is a joint-space force expressing the total contact force, and $\ddot{\boldsymbol{q}}$ will be subject to motion constraints from the contacts. It is therefore necessary to translate the individual contact force and acceleration expressions into joint space before they can be combined with the equation of motion.

Another difference is that the contacts can now take place between different pairs of bodies, so it becomes necessary to identify which bodies take part in each contact. To aid in this identification, we use the terms *predecessor* and *successor* to refer to the two bodies that take part in a given contact. A contact force is defined to be a force transmitted from the predecessor to the successor; and a separation acceleration is defined to be an acceleration of the successor relative to the predecessor. If this convention were applied to Figures 11.1 and 11.2, then body B would be identified as the successor. Note that contacts can

---

[1]Strictly speaking, we mean the space of independent joint variables, since the system may contain kinematic loops.

now take place between a pair of moving bodies as well as between a moving body and the fixed base.

Let there be n_c contacts in total, numbered from 1 to n_c; and let pc and sc be two arrays of integers such that $pc(i)$ and $sc(i)$ are the body numbers of the predecessor and successor bodies involved in contact i. Each contact is further characterized by a contact point, C_i, which lies on the surface of the predecessor; a second point, C_i', which coincides with C_i at the current instant but is fixed in the successor; and a normal vector, $\boldsymbol{n}_i$, which is a unit spatial force acting along the line of the contact normal. $\boldsymbol{n}_i$ is directed from the predecessor to the successor, so that positive scalar multiples of $\boldsymbol{n}_i$ act to repel the two bodies locally at contact i.

Given that the contacts are frictionless, any contact force that is present at contact i must be a scalar multiple of $\boldsymbol{n}_i$. Furthermore, as the force must be repelling in nature, it must be a positive scalar multiple of $\boldsymbol{n}_i$. We therefore introduce the variable λ_i to denote the unknown contact force magnitude at contact i, and note that it must satisfy the constraint $\lambda_i \geq 0$.

The force at contact i is transmitted from body $pc(i)$ to body $sc(i)$. It is therefore equivalent to a force of $\boldsymbol{n}_i \lambda_i$ acting on body $sc(i)$ and a force of $-\boldsymbol{n}_i \lambda_i$ acting on body $pc(i)$. To convert this pair of forces into an equivalent joint-space force, we make use of the general formula $\boldsymbol{\tau} = \boldsymbol{J}_i^{\mathrm{T}} \boldsymbol{f}$, where $\boldsymbol{f}$ is a spatial force applied to body i, $\boldsymbol{\tau}$ is the equivalent joint-space force, and $\boldsymbol{J}_i$ is the body Jacobian for body i. If $\boldsymbol{\tau}_{ci}$ denotes the joint-space force due to contact i, then this formula gives us

$$\boldsymbol{\tau}_{ci} = (\boldsymbol{J}_{sc(i)}^{\mathrm{T}} - \boldsymbol{J}_{pc(i)}^{\mathrm{T}}) \, \boldsymbol{n}_i \, \lambda_i \,. \tag{11.24}$$

At this point, we introduce a new quantity, $\boldsymbol{t}_i$, defined as follows:

$$\boldsymbol{t}_i = (\boldsymbol{J}_{sc(i)}^{\mathrm{T}} - \boldsymbol{J}_{pc(i)}^{\mathrm{T}}) \, \boldsymbol{n}_i \,. \tag{11.25}$$

It can be regarded as the contact normal vector for contact i, expressed in joint space. Substituting Eq. 11.25 into Eq. 11.24 gives

$$\boldsymbol{\tau}_{ci} = \boldsymbol{t}_i \, \lambda_i \,.$$

The joint-space force expressing the total contact force (i.e., the combined effect of all the contact forces) can now be written

$$\boldsymbol{\tau}_c = \sum_{i=1}^{n_c} \boldsymbol{t}_i \, \lambda_i \,, \tag{11.26}$$

which replaces Eq. 11.11.

Let ζ_i denote the separation velocity at contact i. As both bodies may have nonzero velocities, we define the separation velocity to be the normal component of the relative linear velocity of C_i' with respect to C_i. Thus,

$$\zeta_i = \boldsymbol{n}_i \cdot (\boldsymbol{v}_{sc(i)} - \boldsymbol{v}_{pc(i)}) \,, \tag{11.27}$$

which is a generalization of Eq. 11.12. To express ζ_i as a function of the joint-space velocity, $\dot{\boldsymbol{q}}$, we use the general formula $\boldsymbol{v}_i = \boldsymbol{J}_i\,\dot{\boldsymbol{q}}$, where $\boldsymbol{v}_i$ and $\boldsymbol{J}_i$ are the velocity and body Jacobian for body i. Applying this formula to Eq. 11.27 gives

$$\zeta_i = \boldsymbol{n}_i \cdot (\boldsymbol{J}_{sc(i)} - \boldsymbol{J}_{pc(i)})\,\dot{\boldsymbol{q}}\,.$$

However, on comparing this equation with the definition of $\boldsymbol{t}_i$ in Eq. 11.25, it can be seen that

$$\zeta_i = \boldsymbol{t}_i^{\mathrm{T}}\dot{\boldsymbol{q}}\,. \tag{11.28}$$

This is the joint-space equivalent of Eq. 11.12.

Having obtained an expression for the separation velocity, the separation acceleration is just the derivative of this expression, which is

$$\dot{\zeta}_i = \frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{t}_i^{\mathrm{T}}\dot{\boldsymbol{q}}) = \boldsymbol{t}_i^{\mathrm{T}}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{t}}_i^{\mathrm{T}}\dot{\boldsymbol{q}}\,. \tag{11.29}$$

This equation replaces Eq. 11.13. Employing the same argument as in previous sections, it can be shown that $\zeta_i = 0$ at the current instant, which implies $\dot{\zeta}_i \geq 0$. The term $\dot{\boldsymbol{t}}_i^{\mathrm{T}}\dot{\boldsymbol{q}}$ can be calculated from the expression

$$\dot{\boldsymbol{t}}_i^{\mathrm{T}}\dot{\boldsymbol{q}} = \boldsymbol{n}_i \cdot (\boldsymbol{a}_{sc(i)}^{vp} - \boldsymbol{a}_{pc(i)}^{vp}) + \dot{\boldsymbol{n}}_i \cdot (\boldsymbol{v}_{sc(i)} - \boldsymbol{v}_{pc(i)})\,, \tag{11.30}$$

where $\boldsymbol{a}_i^{vp}$ is the velocity-product acceleration of body i, which is equal to $\dot{\boldsymbol{J}}_i\,\dot{\boldsymbol{q}}$.

At this point, we introduce the vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\zeta}$, as defined in Eq. 11.15, and a new $n \times n_c$ matrix, $\boldsymbol{T}$, which is defined as follows:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1 & \boldsymbol{t}_2 & \cdots & \boldsymbol{t}_{n_c} \end{bmatrix}\,. \tag{11.31}$$

In terms of these quantities, Eqs. 11.26 and 11.29 can be written

$$\boldsymbol{\tau}_c = \boldsymbol{T}\,\boldsymbol{\lambda} \tag{11.32}$$

and

$$\dot{\boldsymbol{\zeta}} = \boldsymbol{T}^{\mathrm{T}}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{T}}^{\mathrm{T}}\dot{\boldsymbol{q}}\,, \tag{11.33}$$

which replace Eqs. 11.16 and 11.17. However, the constraints on $\boldsymbol{\lambda}$ and $\dot{\boldsymbol{\zeta}}$ are still correctly described by Eq. 11.18.

The final step is to formulate an equation expressing $\dot{\boldsymbol{\zeta}}$ as a function of $\boldsymbol{\lambda}$. Substituting Eq. 11.32 into Eq. 11.23, and rearranging to bring $\ddot{\boldsymbol{q}}$ over to the left-hand side, gives

$$\ddot{\boldsymbol{q}} = \boldsymbol{H}^{-1}(\boldsymbol{T}\,\boldsymbol{\lambda} + \boldsymbol{\tau} - \boldsymbol{C})\,;$$

and substituting this expression into Eq. 11.33 gives

$$\dot{\boldsymbol{\zeta}} = \boldsymbol{M}\boldsymbol{\lambda} + \boldsymbol{d}\,, \tag{11.34}$$

where

$$\boldsymbol{M} = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{H}^{-1}\boldsymbol{T} \tag{11.35}$$

State 1   State 2a   State 2b   State 3

$\phi > 0$ $\phi = 0$ $\phi = 0$ $\phi = 0$
$\dot{\phi}$ unconstrained $\dot{\phi} < 0$ $\dot{\phi} > 0$ $\dot{\phi} = 0$
$\ddot{\phi}$ unconstrained impulse $\ddot{\phi}$ unconstrained $\ddot{\phi} \geq 0$
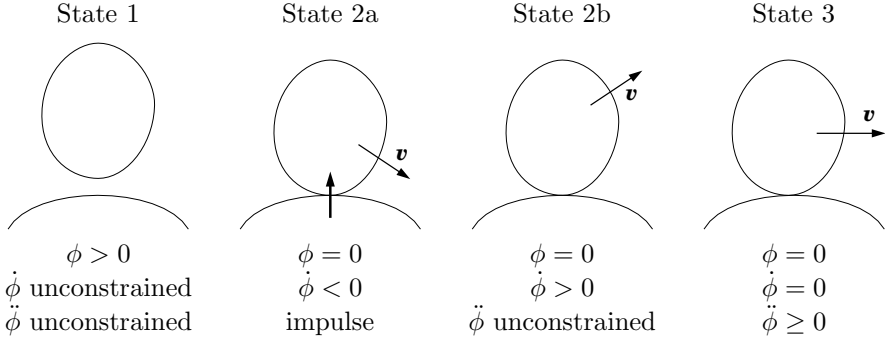
Figure 11.4: The possible states of $\phi(\boldsymbol{q}) \geq 0$

and

$$\boldsymbol{d} = \boldsymbol{T}^{\mathrm{T}} \boldsymbol{H}^{-1}(\boldsymbol{\tau} - \boldsymbol{C}) + \dot{\boldsymbol{T}}^{\mathrm{T}} \dot{\boldsymbol{q}} \,. \tag{11.36}$$

Equations 11.35 and 11.36 replace 11.21 and 11.22, but Eq. 11.34 is essentially the same as Eq. 11.20. In fact, the only difference between these two equations is that the rank of $\boldsymbol{M}$ in Eq. 11.20 is limited to 6 by Eq. 11.21, whereas the rank of $\boldsymbol{M}$ in Eq. 11.34 is potentially unlimited. The solution procedure is exactly the same as in Section 11.2: solve Eq. 11.34 subject to Eq. 11.18 using the methods described in Section 11.5.

11.4   Inequality Constraints

Let B_1 and B_2 be two bodies in a general rigid-body system; and let S_1 and S_2 be two surfaces, or surface patches, such that S_1 is fixed in B_1 and S_2 is fixed in B_2. A contact constraint between these two surfaces is a constraint that says they are not allowed to penetrate. Suppose we have a function, $\phi(\boldsymbol{q})$, with the property that $\phi(\boldsymbol{q})$ is greater than, equal to, or less than zero, according to whether S_1 and S_2 are apart, touching, or penetrating each other, respectively, at configuration $\boldsymbol{q}$. For example, $\phi(\boldsymbol{q})$ could be a measure of the separation distance when S_1 and S_2 are apart, and the negative of the penetration distance when they are penetrating each other. Given a function with this property, the contact constraint can be expressed by the equation

$$\phi(\boldsymbol{q}) \geq 0 \,. \tag{11.37}$$

A constraint equation of this form is called an *inequality constraint*, or, alternatively, a *unilateral constraint*. The latter name refers to the one-sided nature of the constraint: if the two surfaces are touching, then motion in one direction is allowed, but motion in the opposite direction is not.

An inequality constraint can have different effects on a rigid-body system, depending on the values of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. To understand these effects, we identify four states, as shown in Figure 11.4, and consider each in turn.

State 1 is characterized by $\phi(\boldsymbol{q}) > 0$, and therefore applies at every configuration $\boldsymbol{q}$ satisfying $\phi(\boldsymbol{q}) > 0$. In this state, the surfaces are separated by a distance greater than zero; so there is no contact, no constraint on the velocity or acceleration of the system, and no contact force. An inequality constraint in this state has no effect on the instantaneous dynamics of the system, and can be regarded as inactive.

State 2a is characterized by $\phi = 0$ and $\dot{\phi} < 0$. ($\dot{\phi}$ is a function of both $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.) This state arises at the moment of collision between the two surfaces, and it causes an impulse to be exerted between the two colliding bodies. Immediately after this impulse, the system will either be in state 2b or state 3, depending on whether the collision is elastic (bounce) or plastic (no bounce). State 2a lasts only for a single instant, but it causes a step change in the velocity of the system, which must be calculated using impulsive dynamics equations.

State 2b is characterized by $\phi = 0$ and $\dot{\phi} > 0$. In this state, the surfaces are touching, but they are flying apart with a positive separation velocity. This state can occur only in the immediate aftermath of an impulse, or at the moment of a geometric loss of contact (see §11.6). It lasts only for a single instant, because the constraint will transition to state 1 immediately after the current instant. State 2b has no effect on the instantaneous dynamics of the system, and can be treated like state 1.

State 3 is characterized by $\phi = 0$ and $\dot{\phi} = 0$, and is the state associated with prolonged contact. In this state, the two surfaces are in contact, and their separation velocity is zero. The separation acceleration is therefore constrained to be non-negative, and a constraint force (which is the contact force) will act to impose this constraint on the system. To be more specific, one of two things can happen at the current instant: either the two surfaces accelerate away from each other ($\ddot{\phi} > 0$), or they remain in contact ($\ddot{\phi} = 0$). In the former case, the inequality constraint will transition to state 1 immediately after the current instant, and the contact force will be zero. In the latter case, the constraint will remain in state 3, and a contact force will act to prevent $\ddot{\phi} < 0$.

It follows from this description that Sections 11.1 to 11.3 have all been concerned with the analysis of inequality constraints in state 3. Another connection is revealed when we differentiate $\phi(\boldsymbol{q})$:

$$\dot{\phi} = \boldsymbol{t}^{\mathrm{T}} \dot{\boldsymbol{q}} \tag{11.38}$$

and

$$\ddot{\phi} = \boldsymbol{t}^{\mathrm{T}} \ddot{\boldsymbol{q}} + \dot{\boldsymbol{t}}^{\mathrm{T}} \dot{\boldsymbol{q}}, \tag{11.39}$$

where

$$\boldsymbol{t}^{\mathrm{T}} = \frac{\partial \phi}{\partial \boldsymbol{q}}. \tag{11.40}$$

On comparing these equations with Eqs. 11.28 and 11.29, it can be seen that ζ_i and $\dot{\zeta}_i$ correspond to $\dot{\phi}_i$ and $\ddot{\phi}_i$ for an appropriate choice of function $\phi_i(\boldsymbol{q})$. Likewise, $\boldsymbol{t}_i$ in Eq. 11.25 corresponds to $(\partial \phi_i/\partial \boldsymbol{q})^{\mathrm{T}}$.

11.5 Solving Contact Equations

According to Section 11.3, the equation of motion for a rigid-body system subject to point-contact constraints is

$$H\ddot{q} + C = \tau + T\lambda, \qquad (11.41)$$

where λ is a solution to the problem

$$\dot{\zeta} = M\lambda + d, \quad \dot{\zeta} \geq 0, \quad \lambda \geq 0, \quad \dot{\zeta}^{\mathrm{T}}\lambda = 0. \qquad (11.42)$$

The equations developed in Sections 11.1 and 11.2 are just special cases of these equations in which Eq. 11.41 is replaced by the equation of motion for a single rigid body.

Equations like 11.41 have been covered in earlier chapters, but 11.42 is new. In fact, the set of equations and inequalities in Eq. 11.42 define a standard problem in mathematics known as a *linear complementarity problem* (LCP). The mathematics of this problem, and several algorithms for solving it, can be found in Cottle et al. (1992) and Cottle and Dantzig (1968); and an algorithm for the special case when M is positive definite can be found in Featherstone (1987).

In contact-dynamics applications, M will always be a symmetric, positive-semidefinite matrix, and it will be positive definite if the columns of T (i.e., the contact normal vectors) are linearly independent. The LCP for a symmetric, positive-semidefinite matrix has the following special properties:

1. If M is positive definite then there will be exactly one solution; otherwise there may be no solution, one solution, or infinitely many solutions.

2. If a solution exists, then the values of $T\lambda$ and $\dot{\zeta}$ are unique (i.e., if multiple solutions exist, then they differ only in the value of λ, and the differences lie in the null space of T).

3. The LCP in Eq. 11.42 is equivalent to the following quadratic program:[2]

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\lambda^{\mathrm{T}}M\lambda + \lambda^{\mathrm{T}}d \\ \text{subject to} & \lambda \geq 0. \end{array} \qquad (11.43)$$

Both Eq. 11.42 and Eq. 11.43 have the same solutions, but 11.43 has the advantage that software to solve quadratic programs is more widely available than software to solve LCPs. Two more formulations are mentioned in Lötstedt (1982). One of them is

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}(\ddot{q} - H^{-1}(\tau - C))^{\mathrm{T}}H(\ddot{q} - H^{-1}(\tau - C)) \\ \text{subject to} & T^{\mathrm{T}}\ddot{q} + \dot{T}^{\mathrm{T}}\dot{q} \geq 0. \end{array} \qquad (11.44)$$

---

[2]The contents of Eq. 11.42 are the Kuhn-Tucker conditions for the solution of Eq. 11.43.

1. Integrate forward in time, treating active contacts as equality constraints, and ignoring all other contacts.

2. As the integration proceeds, monitor the system for two kinds of event: geometric events (contact gains and losses) and negative contact forces.

3. If one or more such events are detected during the most recent integration step, then interpolate the system back to the moment of the earliest event.

4. If the event is a contact gain (e.g. a collision), then apply impulsive dynamics as described in Section 11.7, and identify the new set of current contacts and the new set of active contacts.

5. If the event is a geometric contact loss, then remove the lost contacts from the sets of current and active contacts.

6. If the event is a negative contact force, then formulate and solve either Eq. 11.42 or 11.43 and identify the new sets of current and active contacts.

7. Go to step 1.

Table 11.1: General procedure for contact dynamics simulation

This, too, is a quadratic program, but minimizing over the variable $\ddot{q}$ instead of $\boldsymbol{\lambda}$. This formulation will produce a unique solution, if one exists, and it may have a computational advantage if $n_c > n$, but recovering $\boldsymbol{\lambda}$ from $\ddot{q}$ is not straightforward. Equation 11.44 can be regarded as a generalization of Gauss' principle of least constraint, making it applicable to rigid-body systems with inequality constraints.

If there are many contacts, then the computational cost of solving Eq. 11.42 or 11.43 will be relatively large compared with the cost of imposing an equivalent set of equality constraints using the techniques of Chapter 8. It is therefore desirable to minimize the number of times that Eq. 11.42 or 11.43 is solved. A procedure to accomplish this is shown in Table 11.1.

According to this procedure, the simulator maintains a set of current contacts and a set of active contacts, the latter being a subset of the former. The set of current contacts contains every contact that is currently in state 3, as defined in Figure 11.4, and that also satisfies $\dot{\zeta}_i = 0$ (i.e., $\ddot{\phi}_i = 0$) for the appropriate index i. If a current contact is found to satisfy $\dot{\zeta}_i > 0$, then it is deemed to have broken at the current instant, and is removed from the set.

The set of active contacts is a set whose contact normals form a basis on range($\boldsymbol{T}$), and which contains every contact having a strictly positive contact force; that is, every contact for which $\lambda_i > 0$. If we treat the active contacts as equality constraints, then the following statement is true: for as long as the contact force variables remain non-negative, there is no change in the state of

contact. Thus, it is possible to get away with solving Eq. 11.42 or 11.43 only when a negative λ_i is detected on an active contact, or when a collision occurs. A negative λ_i will always cause a change in the set of active contacts, but not necessarily a change in the set of current contacts.

Most algorithms for solving LCPs and quadratic programs return a *basic* solution to the problem, which is a solution having the smallest possible number of nonzero variables. A basic solution to Eq. 11.42 or 11.43 has the property that the contact normals associated with the positive contact force variables are linearly independent. Given a basic solution to Eq. 11.42 or 11.43, the new current-contact set will be the set of contacts satisfying $\dot{\zeta}_i = 0$, and the new active-contact set will be the set of contacts satisfying $\lambda_i > 0$. On rare occasions, this set will be insufficient to form a basis, and will need to be supplemented with contacts satisfying $\dot{\zeta}_i = \lambda_i = 0$.

Interpolating an Integration Time Step

The idea that an integration time step can be interpolated needs some explanation. Suppose we wish to integrate the differential equation $\dot{y} = f(y, t)$ from t_0 to t_1, where $t_1 = t_0 + h$ and h is the integration step size. Many methods for numerical integration work by fitting a polynomial to $\dot{y}$ over the interval $[t_0, t_1]$, and adding the integral of this polynomial to $y(t_0)$. In effect, these methods all have the general form

$$\dot{y}(t) = p(t), \qquad t_0 \leq t \leq t_1$$
$$y_1 = y_0 + \int_{t_0}^{t_1} p(t) \, dt \tag{11.45}$$

where $y_0 = y(t_0)$, $y_1 = y(t_1)$, and $p(t)$ is a polynomial. The degree of p is one less than the order of the method. It therefore follows that we can calculate $y(t)$ for any intermediate value of t using the formula

$$y(t) = y_0 + \int_{t_0}^{t} p(t) \, dt. \tag{11.46}$$

To give a concrete example, Euler's integration method works by approximating $\dot{y}$ with a constant over the integration time step. The formula for this method is

$$k = f(y_0, t_0)$$
$$y_1 = y_0 + hk, \tag{11.47}$$

which implies that $p(t) = k$. The interpolating polynomial is therefore

$$y(t) = y_0 + (t - t_0)k. \tag{11.48}$$

Let $\delta = (t - t_0)/h$ be an interpolating variable that varies from 0 to 1 as t varies from t_0 to t_1. Written in terms of δ, the interpolating polynomial for Euler's method is

$$y(\delta) = y_0 + hk\delta. \tag{11.49}$$

Turning to a more complicated example, the formula for (one version of) Heun's integration method (Gear, 1971) is

$$
\begin{aligned}
k_1 &= f(y_0, t_0) \\
k_2 &= f(y_0 + \tfrac{2}{3}hk_1, t_0 + \tfrac{2}{3}h) \\
y_1 &= y_0 + h(\tfrac{1}{4}k_1 + \tfrac{3}{4}k_2).
\end{aligned}
\tag{11.50}
$$

This method approximates $\dot{y}$ with a linear polynomial having the value k_1 at t_0 and k_2 at $t_0 + \tfrac{2}{3}h$. The interpolating polynomial for this method will therefore be quadratic, and is given by the formula

$$
y(\delta) = y_0 + h((\delta - \tfrac{3}{4}\delta^2)k_1 + \tfrac{3}{4}\delta^2 k_2). \tag{11.51}
$$

Formulae such as this allow a dynamics simulator to calculate the values of the state variables at any intermediate moment within a completed integration time step, and therefore also to shorten the most recent time step by any desired amount.

11.6 Contact Geometry

A contact dynamics simulator involves both dynamics calculations and geometrical calculations. So far, we have said almost nothing about the latter. In this section, we examine some of the geometrical aspects of contact dynamics: the representation of shape, geometric events (contact gains and losses), and edge and surface contact.

Shape

The most commonly used computer representation of shape is the polyhedron. A large body of technical knowledge exists, and a large body of software for creating, editing and displaying polyhedra, and for performing efficient collision detection. Unfortunately, polyhedra are not well suited to dynamics. Some examples of the kind of problem that can arise are shown in Figure 11.5. The left-hand diagrams compare a true sphere with a polyhedral approximation to a sphere (drawn as a polygon) rolling down a slope. Whereas the sphere rolls smoothly, the polyhedron makes a succession of impacts with the slope as each
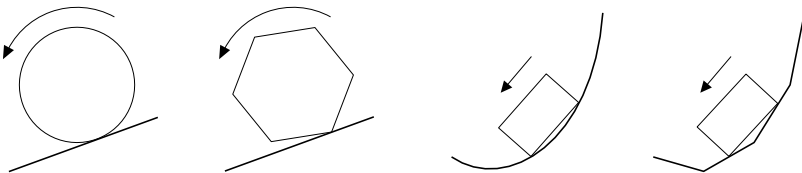
Figure 11.5: Some problems with polyhedral approximations

new vertex strikes the surface. How are these impacts to be handled? If they are treated as plastic collisions, then there is a loss of energy at every impact; but if they are treated as elastic collisions, then the polyhedron starts to bounce.

The problem is caused by the flat faces and straight edges of the polyhedron, which allow it to alternate between point, edge and surface contact as the motion proceeds. The cure is to distort the polyhedron by making the faces and edges curve outward, so that the resulting shape is strictly convex. Ideally, this distorted polyhedron should closely resemble a perfect sphere. However, even if it were not perfectly spherical, it would still be able to roll without causing a succession of impacts.

Another example is shown in the right-hand diagrams of Figure 11.5: a rectangular block sliding down a concave slope. If the slope is replaced by a polyhedral approximation, then the block will experience an impact each time one of its vertices reaches an edge in the slope. This time, the problem is caused by the sharp concave edges in the slope, and the cure is to round them.

Both of these examples are manifestations of a single underlying problem. Suppose p denotes a point on the surface of a body, and $n(p)$ is the surface normal at that point. On a curved surface, n varies continuously with p, and this variation finds its way into the equations of motion via the quantity $\dot{n}$. The basic difficulty with using polyhedra to approximate curved surfaces in a dynamics simulation is that $n(p)$ for a polyhedron is piecewise constant, and therefore fails to model dn/dp in the original curved surface.

Geometric Events

We define a geometric event to be a step change in the state of contact that is caused by the motions of the bodies. There are basically two kinds of geometric event: contact gain and contact loss. A collision is a special case of a contact-gain event in which the two participating bodies were not previously in contact. Examples of these events are shown in Figure 11.6. In general, a contact-gain event will cause an impulse, and a contact-loss event will cause a reduction in the set of current contacts (as defined in Section 11.5).

Suppose we use the term 'dynamic contact loss' to refer to the kind of contact loss studied in earlier sections. The difference between a dynamic contact loss and a geometric contact loss, as shown in Figure 11.6, is that the former happens when two surfaces accelerate away from each other because of the

collision   contact gain   contact loss
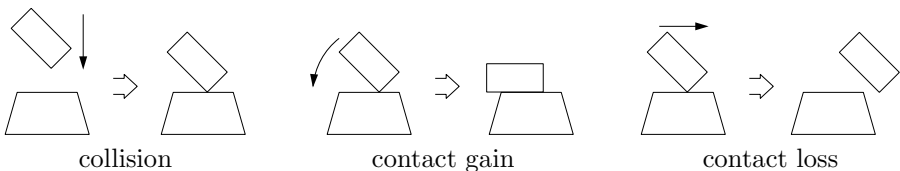
Figure 11.6: Geometric events

forces acting on the system, whereas the latter happens when there is a step change in the separation velocity because a vertex has run off the edge of a face, or one edge has run off the end of another. A geometric contact-loss event cannot be detected by solving equations of motion, but must instead be detected geometrically.

The detection of collisions and contact gains between moving bodies has the potential to be a very expensive computation. This is because accurate geometric models of realistic physical objects can contain hundreds, thousands, or even tens of thousands of geometric primitives. Fortunately, many efficient algorithms have been proposed for these and related geometric calculations, and the interested reader is referred to Bobrow (1989); Gilbert et al. (1988); Gottschalk et al. (1996); Jimenez et al. (2001); Mirtich (1998).

Line and Surface Contact

If a contact extends along a line, or over a surface, then it is mathematically equivalent to an infinite number of point contacts: one for each point on the line or surface. Thus, a line or surface contact could be described by an infinite set of point contacts. However, this set typically contains many more contacts than the minimum required to model the original state of contact. One can obtain a minimal contact set by exhaustively applying the following rule:

> If a point contact has a normal vector that is positively dependent[3] on other normals in the set, then it does not contribute to the contact constraint, and can be removed.

Figure 11.7 shows some examples of line and surface contacts. Diagram (a) shows a line contact between a straight edge on one body and a planar surface on another. In this case, the line contact can be modelled by a pair of point contacts: one at each end of the line. Diagram (b) shows a line contact between a straight edge and a ruled surface. The latter is formed by the equation $z = x \tan(y)$, where the z axis points up and the y axis lies on the line of contact. In this example, the curvature of the ruled surface makes every contact normal point in a different direction. Furthermore, none of the normal vectors in the point-contact set are positively dependent on other normals, so the minimal contact set for this example contains one contact for each point on the line.

Diagram (c) shows a surface contact between two planar surfaces, in which the convex hull of the contact area is a polygon. (A convex hull is the smallest convex shape containing a given shape.) In this case, the convex hull is the pentagon shown in grey. A surface contact like this can be modelled by a finite set of point contacts: one at each vertex of the polygon. Diagram (d) also shows a surface contact between two planar surfaces, but the contact area is a circular disc. The convex hull is therefore also a circular disc. In this example,

---

[3]A vector $\boldsymbol{v}$ is said to be positively dependent on a set of vectors $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \cdots\}$ if it can be expressed in the form $\boldsymbol{v} = a_1\boldsymbol{v_1} + a_2\boldsymbol{v_2} + \cdots$, where all of the scalars a_i are non-negative.
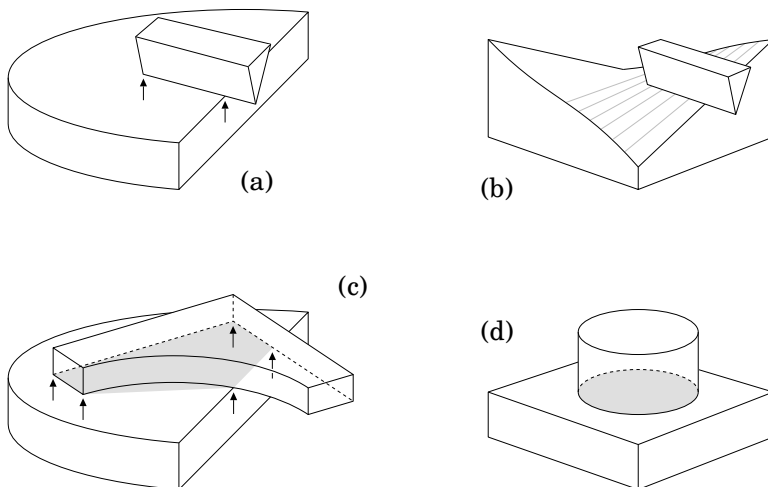
Figure 11.7: Examples of line and surface contacts

every normal vector in the interior of the disc is positively dependent on a pair of normals on the boundary, but no normal on the boundary is positively dependent on other normals in the set. Thus, the minimal contact set for this example consists of every point on the boundary circle.

Observe that examples (a) and (c) in Figure 11.7 cover every possible line and surface contact between two polyhedra; so any state of contact between two polyhedra can always be expressed by a finite set of point contacts. In examples (b) and (d), an infinite number of point contacts are required to express the contact constraint exactly, but an approximation to any desired accuracy can be obtained using only a finite number of points.

11.7   Impulsive Dynamics

Impulse is the time-integral of force. If a force $\boldsymbol{f}(t)$ is applied to a rigid body over a period from t_0 to t_1, then the body receives an impulse, $\boldsymbol{\iota}$, given by

$$\boldsymbol{\iota} = \int_{t_0}^{t_1} \boldsymbol{f}(t)\,\mathrm{d}t\,. \tag{11.52}$$

The effect of this impulse is to alter the momentum of the body as follows:

$$\boldsymbol{\iota} = \int_{t_0}^{t_1} \boldsymbol{f}(t)\,\mathrm{d}t = \int_{t_0}^{t_1} \frac{\mathrm{d}\boldsymbol{h}}{\mathrm{d}t}\,\mathrm{d}t = \boldsymbol{h}(t_1) - \boldsymbol{h}(t_0)\,, \tag{11.53}$$

where $\boldsymbol{f} = \mathrm{d}\boldsymbol{h}/\mathrm{d}t$ is the body's equation of motion, and $\boldsymbol{h}$ is its momentum ($\boldsymbol{h} = \boldsymbol{I}\boldsymbol{v}$). Thus, the impulse received by a rigid body over a given time interval equals its net change in momentum over that interval.

If two hard bodies collide, then they exert a large contact force on each other for a short period of time. If we increase their hardness to infinity, so that they become rigid, then the contact force diverges to infinity and the period converges to zero, but the impulse remains finite. Thus, for rigid-body dynamics, we are interested in a special kind of impulse that is the limit as $\delta t \to 0$ of a force $\boldsymbol{f}/\delta t$ applied over a period of duration δt:

$$\boldsymbol{\iota} = \lim_{\delta t \to 0} \int_t^{t+\delta t} \frac{\boldsymbol{f}(t)}{\delta t}\, \mathrm{d}t\,. \tag{11.54}$$

The rest of this section is concerned with impulses of this kind. Their basic properties are as follows: they are force vectors (i.e., they are elements of F^6); they arise from collisions (or contact gains) between rigid bodies; and they cause step changes in rigid-body velocities.

Let us now derive the impulsive equation of motion for a rigid body. Let $\boldsymbol{\iota}$ be an impulse applied to a body having an inertia of $\boldsymbol{I}$ and a velocity of $\boldsymbol{v}$, and let $\Delta \boldsymbol{v}$ be the step change in velocity caused by $\boldsymbol{\iota}$. The relationship between $\Delta \boldsymbol{v}$ and $\boldsymbol{\iota}$ is obtained as follows:

$$\begin{aligned}
\Delta \boldsymbol{v} &= \lim_{\delta t \to 0} \boldsymbol{v}(t + \delta t) - \boldsymbol{v}(t) \\
&= \lim_{\delta t \to 0} \int_t^{t+\delta t} \boldsymbol{a}(t)\, \mathrm{d}t \\
&= \lim_{\delta t \to 0} \int_t^{t+\delta t} \boldsymbol{I}^{-1}\big(\frac{\boldsymbol{f}}{\delta t} - \boldsymbol{v} \times^* \boldsymbol{I} \boldsymbol{v}\big)\, \mathrm{d}t \\
&= \lim_{\delta t \to 0} \boldsymbol{I}^{-1}(t) \int_t^{t+\delta t} \frac{\boldsymbol{f}}{\delta t}\, \mathrm{d}t - \lim_{\delta t \to 0} \int_t^{t+\delta t} \boldsymbol{I}^{-1} \boldsymbol{v} \times^* \boldsymbol{I} \boldsymbol{v}\, \mathrm{d}t \\
&= \boldsymbol{I}^{-1} \boldsymbol{\iota}\,,
\end{aligned}$$

so

$$\boldsymbol{\iota} = \boldsymbol{I}\, \Delta \boldsymbol{v}\,. \tag{11.55}$$

On the third line, the body's equation of motion ($\boldsymbol{I} \boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I} \boldsymbol{v} = \boldsymbol{f}/\delta t$) is used to express its acceleration in terms of the applied force $\boldsymbol{f}/\delta t$. On the fourth line, $\boldsymbol{I}^{-1}$ can be taken outside the integral because $\boldsymbol{I}$ varies only with position, which is a continuous function of time, so $\boldsymbol{I}^{-1}(t + \delta t) \to \boldsymbol{I}^{-1}(t)$ as $\delta t \to 0$. The second limit on the fourth line evaluates to zero because the integrand remains finite as $\delta t \to 0$.

Using the same argument, we can show that the impulsive equation of motion for the handle of an articulated body is

$$\boldsymbol{\iota} = \boldsymbol{I}^A \Delta \boldsymbol{v}\,, \tag{11.56}$$

where $\boldsymbol{I}^A$ is the handle's articulated-body inertia. Likewise, the impulsive equation of motion for a general rigid-body system is

$$\boldsymbol{u} = \boldsymbol{H}\, \Delta \dot{\boldsymbol{q}}\,, \tag{11.57}$$
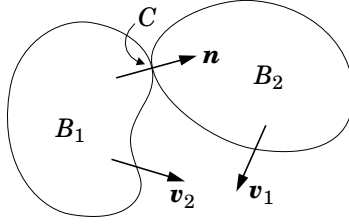
Figure 11.8: Two-body collision

where $\boldsymbol{u}$ is a joint-space (or generalized) impulse and $\Delta\dot{\boldsymbol{q}}$ is the step change in the joint (or generalized) velocity vector. If $\boldsymbol{u}$ is caused by a collision between body i and a body external to the system, such that body i receives an impulse of $\boldsymbol{\iota}$ from the external body, causing a step change of $\Delta\boldsymbol{v}_i$ in its velocity, then $\boldsymbol{u}$, $\boldsymbol{\iota}$, $\Delta\dot{\boldsymbol{q}}$ and $\Delta\boldsymbol{v}_i$ are related by

$$\boldsymbol{u} = \boldsymbol{J}_i^{\mathrm{T}}\boldsymbol{\iota} \tag{11.58}$$

and

$$\Delta\boldsymbol{v}_i = \boldsymbol{J}_i\,\Delta\dot{\boldsymbol{q}}\,, \tag{11.59}$$

where $\boldsymbol{J}_i$ is the body Jacobian for body i.

Caution: Equations 11.56, 11.57 and 11.58 are mathematically correct, but they do not necessarily offer an accurate prediction of the impulsive behaviour of a real multibody system. This is because real impacts involve physical processes that are not modelled by the rigid-body assumption, such as the propagation of compression waves through solids and across joint bearings.

Two-Body Collisions

Figure 11.8 shows two rigid bodies, B_1 and B_2, that collide at a single point C. Their velocities before the impact are $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, and their velocities afterwards are $\boldsymbol{v}_1 + \Delta\boldsymbol{v}_1$ and $\boldsymbol{v}_2 + \Delta\boldsymbol{v}_2$. We assume initially that there is no friction at the contact. This implies that the impulse transmitted from B_1 to B_2 can be expressed in the form $\boldsymbol{n}\lambda$, where $\boldsymbol{n}$ is the contact normal and $\lambda \geq 0$ is the impulse magnitude. The impulsive equations of motion for the two bodies are therefore

$$\Delta\boldsymbol{v}_1 = -\boldsymbol{I}_1^{-1}\boldsymbol{n}\lambda \tag{11.60}$$

and

$$\Delta\boldsymbol{v}_2 = \boldsymbol{I}_2^{-1}\boldsymbol{n}\lambda\,. \tag{11.61}$$

Let ζ and $\Delta\zeta$ denote the separation velocity at C before the impact, and the step change in separation velocity caused by the impulse, respectively. They are given by the equations

$$\zeta = \boldsymbol{n} \cdot (\boldsymbol{v}_2 - \boldsymbol{v}_1) \tag{11.62}$$

and

$$\Delta\zeta = \mathbf{n} \cdot (\Delta\mathbf{v}_2 - \Delta\mathbf{v}_1).$$ (11.63)

These scalars must satisfy $\zeta \leq 0$ and $\zeta + \Delta\zeta \geq 0$. We need one more equation to solve the problem. This is supplied by the coefficient of restitution, e, which is a number between 0 and 1 that expresses the ratio of the separation velocities before and after the impact. Specifically,

$$e = \frac{\zeta + \Delta\zeta}{-\zeta},$$

which implies

$$\Delta\zeta = -(1+e)\zeta.$$ (11.64)

Substituting Eqs. 11.62 and 11.63 into Eq. 11.64 gives

$$\mathbf{n} \cdot (\Delta\mathbf{v}_2 - \Delta\mathbf{v}_1) = -(1+e)\,\mathbf{n} \cdot (\mathbf{v}_2 - \mathbf{v}_1),$$

and substituting Eqs. 11.60 and 11.61 into this equation gives

$$\lambda = \frac{-(1+e)\,\mathbf{n} \cdot (\mathbf{v}_2 - \mathbf{v}_1)}{\mathbf{n} \cdot (\mathbf{I}_1^{-1} + \mathbf{I}_2^{-1})\,\mathbf{n}},$$ (11.65)

which solves the problem.

Friction

If we allow Coulomb friction at the contact, then the behaviour of the system becomes more complicated. In general, we must consider the following effects: tangential friction, torsional friction, tangential restitution, torsional restitution, and whether or not the tangential and/or torsional sliding motion is arrested before the end of the contact period. (Many of these complexities are examined in Brach (1991).) Torsional effects refer to a friction couple that opposes angular motion about the line of the contact normal. They arise because real bodies are not truly rigid, and local deformation of the contacting surfaces cause the point contact to become an area contact. Torsional effects are therefore more apparent with softer bodies than with harder ones. Even so, torsional effects are typically of lesser magnitude than tangential effects, and can often be ignored. Tangential and torsional restitution are caused by the recovery of local elastic deformations of the contacting bodies in the tangential and torsional directions, respectively. Some examples of these effects are shown in Figure 11.9.

Let us conclude this section by examining a relatively simple example of collision with friction. We shall assume that two bodies collide as shown in Figure 11.8, and that there are no torsional effects, no tangential restitution, and the tangential motion is arrested by the friction impulse. This example can be solved by almost the same procedure as for the frictionless case.
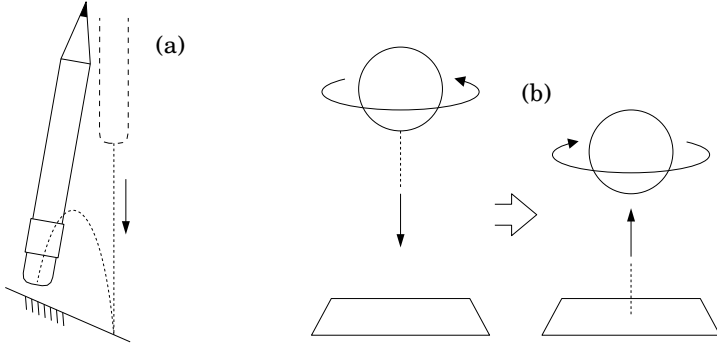
Figure 11.9: Examples of tangential restitution (a) and torsional restitution (b)

The first step is to realise that allowing a tangential friction component adds two degrees of freedom to the collision impulse, so that it is now a function of three unknown scalars instead of only one. At the same time, two constraints are added to the system by the condition that the tangential component of the relative velocity at C must be zero after the impact.

Let $\boldsymbol{\iota}$ be the total impulse transmitted from B_1 to B_2. The equations of motion for the two bodies are then

$$\Delta \boldsymbol{v}_1 = -\boldsymbol{I}_1^{-1} \boldsymbol{\iota} \tag{11.66}$$

and

$$\Delta \boldsymbol{v}_2 = \boldsymbol{I}_2^{-1} \boldsymbol{\iota} \,. \tag{11.67}$$

We can express the impulse in the form

$$\boldsymbol{\iota} = \boldsymbol{N} \boldsymbol{\lambda} \,, \tag{11.68}$$

where $\boldsymbol{N} = [\boldsymbol{n}_x \; \boldsymbol{n}_y \; \boldsymbol{n}_z]$ is a 6×3 matrix spanning the space of possible impulses, and $\boldsymbol{\lambda} = [\lambda_x \; \lambda_y \; \lambda_z]^{\mathrm{T}}$ is a vector of unknown impulse coefficients. $\boldsymbol{n}_z$ is the contact normal vector, so $\boldsymbol{n}_z$ and λ_z correspond to $\boldsymbol{n}$ and λ in the frictionless case, while $\boldsymbol{n}_x$ and $\boldsymbol{n}_y$ lie in the tangent plane of the contact, and span the space of possible friction impulses. All three vectors have lines of action passing through C.

Let $\boldsymbol{\zeta} = [\zeta_x \; \zeta_y \; \zeta_z]^{\mathrm{T}}$ be a vector of linear velocity coordinates measuring the relative linear velocity of the two bodies at C; and let $\Delta \boldsymbol{\zeta} = [\Delta \zeta_x \; \Delta \zeta_y \; \Delta \zeta_z]^{\mathrm{T}}$ be the step change in $\boldsymbol{\zeta}$ caused by the impulse. These vectors are given by

$$\boldsymbol{\zeta} = \boldsymbol{N}^{\mathrm{T}}(\boldsymbol{v}_2 - \boldsymbol{v}_1) \tag{11.69}$$

and

$$\Delta \boldsymbol{\zeta} = \boldsymbol{N}^{\mathrm{T}}(\Delta \boldsymbol{v}_2 - \Delta \boldsymbol{v}_1) \,. \tag{11.70}$$

The relationship between $\boldsymbol{\zeta}$ and $\Delta\boldsymbol{\zeta}$ is given partly by the coefficient of restitution, which applies in the normal direction, and partly by the condition that the tangential velocity be brought to zero. Basically, we have three scalar equations:

$$\zeta_x + \Delta\zeta_x = 0\,, \qquad \zeta_y + \Delta\zeta_y = 0 \qquad \text{and} \qquad \zeta_z + \Delta\zeta_z = -e\,\zeta_z\,,$$

which can be expressed as a matrix equation

$$\Delta\boldsymbol{\zeta} = -(\mathbf{1} + \boldsymbol{E})\boldsymbol{\zeta} \tag{11.71}$$

where

$$\boldsymbol{E} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & e \end{bmatrix}.$$

To solve the problem, we simply substitute Eqs. 11.69 and 11.70 into 11.71, giving

$$\boldsymbol{N}^{\mathrm{T}}(\Delta\boldsymbol{v}_2 - \Delta\boldsymbol{v}_1) = -(\mathbf{1} + \boldsymbol{E})\boldsymbol{N}^{\mathrm{T}}(\boldsymbol{v}_2 - \boldsymbol{v}_1)\,,$$

and then substitute Eqs. 11.66, 11.67 and 11.68 into this equation, giving

$$\boldsymbol{\lambda} = -(\boldsymbol{N}^{\mathrm{T}}(\boldsymbol{I}_1^{-1} + \boldsymbol{I}_2^{-1})\boldsymbol{N})^{-1}(\mathbf{1} + \boldsymbol{E})\boldsymbol{N}^{\mathrm{T}}(\boldsymbol{v}_2 - \boldsymbol{v}_1)\,. \tag{11.72}$$

11.8   Soft Contact

So far, we have studied only the phenomenon of contact between pairs of rigid bodies. However, many physical systems contain mixtures of hard and soft bodies, or hard bodies with soft surfaces, or bodies that are hard enough to be considered rigid overall, but soft enough to experience significant local deformations during contact. In these cases, one must be able to model the dynamics of contact between a rigid surface and a compliant one, or between two compliant surfaces.

Another reason for being interested in compliant contact is that it can be implemented more easily than rigid contact: there are no impulses at the moment of collision, so there is no need for impulsive dynamics calculations; and contact loss can be determined from position and velocity data, so there is no need to solve a linear complementarity problem. It is also easier to implement Coulomb friction at a compliant contact. For these reasons, it may be more sensible to use compliant contact in place of rigid contact, even in cases where rigid contact is a better model of the physical system, if implementation effort is an issue.

The main disadvantage of compliant contact is that it introduces high-frequency dynamics into the system, due to the presence of stiff springs in the compliant surfaces. If these dynamics require the integration routine to take smaller steps, then the speed of simulation will be reduced.

This section presents a technique for modelling compliant contact, including Coulomb friction, in which the compliant surface is modelled as a first-order
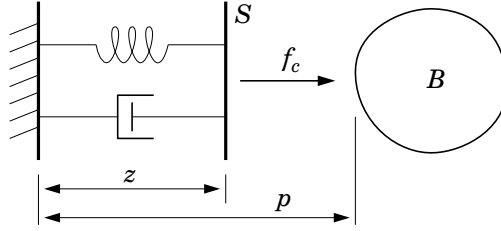
Figure 11.10: Compliant contact model

dynamical system—a system containing springs and dampers, but no mass. In such a system, an applied force causes a velocity, and an imposed velocity meets with a reaction force. The big advantage of this technique is that it separates the rigid-body dynamics from the contact dynamics: the contact forces can be calculated from the position and velocity variables, and then supplied to the rigid-body dynamics routine as a collection of known force inputs. It is therefore possible to use any standard forward-dynamics algorithm with this contact model, such as those described in Chapters 6, 7 and 8.

Contact Normal Force

Consider the one-dimensional rigid-body system shown in Figure 11.10. It consists of a rigid body, B, and a massless surface patch, S, the latter being connected to a fixed base by a spring and a damper. The variables z and p denote the positions of S and B, respectively, and f_c is the contact force transmitted from S to B. If B and S are in contact, then $p = z$ and $f_c \geq 0$; otherwise, $p > z$ and $f_c = 0$. The spring has a stiffness of K, and it exerts a force of $-Kz$ on S. Likewise, the damper has a damping coefficient of D, and it exerts a force of $-D\dot{z}$ on S.

Now, S has no mass, so the net force acting on it must be zero. The three forces that act on S are $-Kz$, $-D\dot{z}$ and $-f_c$, so we have

$$Kz + D\dot{z} + f_c = 0 \,.$$

Rearranging this equation gives

$$\dot{z} = -(Kz + f_c)/D \,, \tag{11.73}$$

which is the equation of motion for S. Observe that this is a first-order differential equation: it involves z and $\dot{z}$, but not $\ddot{z}$. Observe, too, that the contact force is linearly related to the *velocity* of S, not its acceleration. The two unknowns in this equation are $\dot{z}$ and f_c, and the state variable is z.

Let us now calculate f_c. If $p > z$ then there is no contact, and $f_c = 0$. If $p = z$ then we proceed as follows. If the contact is to persist, then we must have $\dot{p} = \dot{z}$ at the current instant. The contact force required to achieve this is

$$f_c = -Kz - D\dot{p} \,.$$

If this expression is zero or positive, then it is the correct value for f_c, and the contact does indeed persist. However, if this expression turns out to be negative, then the correct value for f_c is zero, and we have $\dot{p} > \dot{z}$, which implies that the contact is breaking. Assembling this argument into a single equation, we have

$$f_c = \begin{cases} 0 & \text{if } p > z \\ \max(0, -Kz - D\dot{p}) & \text{if } p = z. \end{cases} \qquad (11.74)$$

Equations 11.74 and 11.73, together with the equation of motion for B (to calculate $\ddot{p}$ from f_c), define the dynamic behaviour of the system.

Extensions

This contact model is easily extended to more general cases. For example, to model a compliant contact between two moving bodies, just redefine p to be the relative position of the two bodies; and to model a contact between two compliant surfaces, simply use one spring-damper pair and one state variable (i.e., one z variable) for each surface. If the ratio K/D is the same for both surfaces, then the two compliances can be lumped together.

It is possible to implement this model without using an explicit state variable. This trick is made possible by the fact that Eq. 11.73 has a simple analytical solution when $f_c = 0$. Thus, we have $z = p$ whenever there is contact, and $z = z_0 e^{-K(t-t_0)/D}$ at all other times, where t_0 and z_0 are the time at which the most recent contact loss occurred and the value of z at that time, respectively. This trick removes the need to integrate Eq. 11.73 numerically, which may improve the efficiency of the simulation.

It is possible to incorporate nonlinear springs and dampers into this model. The benefit of doing so is that a nonlinear spring-damper combination provides a more accurate model of the physical behaviour of colliding solids (Marhefka and Orin, 1999). Another possibility is to use a more general visco-elastic element, such as a spring-damper pair in series with a spring. More complex models may require additional state variables.

To apply this model to a point contact between two 3D bodies, we need two quantities calculated from the shapes and positions of the bodies: the contact normal vector, $\boldsymbol{n}$, and the negative of the penetration depth. The latter gives us the value of p. If the deformations are small, then it is reasonable to calculate $\boldsymbol{n}$ from the undeformed shapes of the surfaces. $\dot{p}$ is equivalent to ζ in earlier sections, and can be calculated from an equation of the form $\dot{p} = \boldsymbol{n} \cdot (\boldsymbol{v}_B - \boldsymbol{v}_S)$, where $\boldsymbol{v}_B$ and $\boldsymbol{v}_S$ are the velocities of the contacting bodies. Likewise, f_c is equivalent to λ in earlier sections, and the spatial contact force transmitted to body B is $\boldsymbol{n} f_c$. The accelerations of individual rigid bodies, or a complete rigid-body system, can then be calculated directly from equations like 11.5, 11.19 or 11.23, given that the contact force magnitudes are already known.

If the contact point moves over the compliant surface, then the spring and damper should follow the point. If it is desired to model the work done in moving the locus of compression over the surface, then this work can be approximated
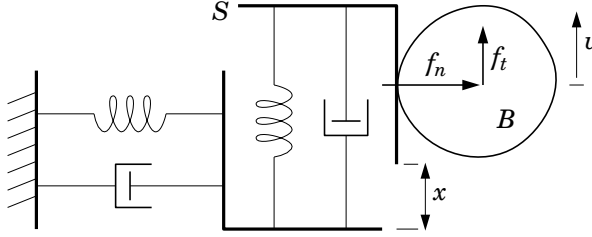
Figure 11.11: Coulomb friction model

by a viscous friction term in the tangent plane. Alternatively, one could embed an array of spring-damper pairs at fixed locations in the compliant surface, and employ an interpolation scheme based on the location of the contact point.

Coulomb Friction

The surface in Figure 11.10 is compliant only in the normal direction. We now add a tangential compliance, resulting in the system shown in Figure 11.11. (To extend this model to 3D, we would have to add a second tangential compliance.) This additional compliance can be used to implement a Coulomb friction model in which the friction force is a function of only the position and velocity variables. Thus, both the contact normal force and the friction force can be computed ahead of the main rigid-body dynamics calculation, and can be treated as known forces by the forward dynamics algorithm.

We introduce two new variables, x and v, which are the tangential counterparts of z and $\dot{p}$. x is the displacement in the tangent direction of the surface patch from its equilibrium position; while v is the tangential velocity of the point in B that is currently in contact with S (i.e., the tangential velocity of the point C' as defined in Section 11.1). So v refers to a different point in B at each instant. v is used to calculate the slip velocity at the contact.

The objective is to calculate both the contact normal force, f_n, and the tangent force, f_t, which is the friction force. The first step is to calculate f_n using Eqs. 11.74 and 11.73. This gives us

$$f_n = \begin{cases} 0 & \text{if } p > z \\ \max(0, -K_n z - D_n \dot{p}) & \text{if } p = z \end{cases} \tag{11.75}$$

and

$$\dot{z} = -(K_n z + f_n)/D_n \,, \tag{11.76}$$

where K_n and D_n are the spring and damper coefficients in the normal direction. The next step is to formulate the equation of motion for S in the tangent direction. Using the same argument as for Eq. 11.73, this equation is

$$\dot{x} = -(K_t x + f_t)/D_t \,, \tag{11.77}$$

where K_t and D_t are the spring and damper coefficients in the tangent direction. The next step is to work out what the tangent force would have to be in order to prevent any slippage between B and S. The condition for there to be no slippage is $\dot{x} = v$, and the force required to produce this value of $\dot{x}$ is

$$f_{stick} = -K_t x - D_t v \,. \tag{11.78}$$

The rule for calculating the Coulomb friction force can now be stated as follows:

$$f_t = \begin{cases} -\mu f_n & \text{if } f_{stick} < -\mu f_n \\ \mu f_n & \text{if } f_{stick} > \mu f_n \\ f_{stick} & \text{otherwise,} \end{cases} \tag{11.79}$$

where μ is the coefficient of friction. In words, f_t is constrained to lie in the friction cone, which means it must lie in the range $-\mu f_n$ to μf_n. If there is a force in this range that can prevent slippage between B and S, then that is the value of f_t; otherwise, f_t lies on the nearest range boundary, and some slippage does occur.

11.9 Further Reading

Contact and impact dynamics are large subjects, and this chapter has provided little more than a brief introduction. From here, the logical next step would be to read the books by Coutinho (Coutinho, 2001) and Brach (Brach, 1991). The former covers both contact and impact dynamics, while the latter specializes in impact. For an introduction to the literature, try the review articles by Gilardi and Sharf (Gilardi and Sharf, 2002) and Stewart (Stewart, 2000). The latter is more mathematical. Another item for the mathematically-minded is a compilation of lecture notes edited by Pfeiffer and Glocker (Pfeiffer and Glocker, 2000). The modelling of friction is another large subject that has barely been touched in this chapter. More on this topic can be found in Armstrong-Hélouvry et al. (1994); Dupont et al. (2002); Haessig and Friedland (1991). And finally, some more relevant material can be found in Lötstedt (1981, 1982, 1984); Marhefka and Orin (1999); Nakamura and Yamane (2000); Wittenburg (1977); Yamane (2004).

Appendix A

Spatial Vector Arithmetic

This appendix presents two approaches to implementing spatial vector arithmetic: one that emphasizes simplicity, and one that emphasizes efficiency. The simple method is presented for both planar and spatial vectors, but the efficient method only for spatial vectors. The simple method is appropriate whenever human productivity is more important than computing efficiency. It assumes the user has access to a general-purpose matrix arithmetic tool, like Matlab,[1] and it exploits this tool by providing only those few spatial arithmetic functions that are not standard matrix operations. The efficient method uses compact data structures to store spatial quantities efficiently, and a variety of optimizations and efficiency tricks to cut the cost of performing spatial arithmetic.

A.1 Simple Planar Arithmetic

Almost all of the basic operations in planar vector arithmetic are standard matrix operations. Thus, if one has access to a general-purpose matrix arithmetic tool, like Matlab, then only a few extra functions are required, and these are shown in Table A.1.

The function Xpln constructs a planar coordinate transformation matrix. Referring to the diagram immediately below it, the two arguments r and θ specify the position and orientation of coordinate frame B relative to frame A; and the matrix returned by Xpln is the motion-vector transform from A coordinates to B coordinates (i.e., $\boldsymbol{X} = {}^{B}\boldsymbol{X}_{A}$). This same convention is used in all other transform-making functions: the arguments describe where the new frame is relative to the current frame, and the function returns the motion-vector coordinate transform from the current frame to the new one. To obtain a force-vector transform, one can use the formula ${}^{B}\boldsymbol{X}_{A}^{*} = ({}^{B}\boldsymbol{X}_{A})^{-\mathrm{T}}$.

The functions crm and crf implement the planar cross operators. In both cases, the argument is a planar vector, $\boldsymbol{v} = [\omega \ v_{Ox} \ v_{Oy}]^{\mathrm{T}}$, specifying a velocity.

---

[1]Matlab is a registered trademark of The MathWorks, Inc.

---
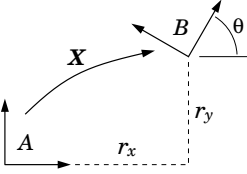
function $\boldsymbol{X} = \mathrm{Xpln}(\theta, \boldsymbol{r})$
   $c = \cos(\theta)$
   $s = \sin(\theta)$
$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 & 0 \\ s\,r_x - c\,r_y & c & s \\ c\,r_x + s\,r_y & -s & c \end{bmatrix}$$
end

function $\boldsymbol{v}\times = \mathrm{crm}(\boldsymbol{v})$
$$\boldsymbol{v}\times = \begin{bmatrix} 0 & 0 & 0 \\ v_{Oy} & 0 & -\omega \\ -v_{Ox} & \omega & 0 \end{bmatrix}$$
end

function $\boldsymbol{v}\times^* = \mathrm{crf}(\boldsymbol{v})$
$$\boldsymbol{v}\times^* = \begin{bmatrix} 0 & -v_{Oy} & v_{Ox} \\ 0 & 0 & -\omega \\ 0 & \omega & 0 \end{bmatrix}$$
end

function $\boldsymbol{I} = \mathrm{mcI}(m, \boldsymbol{c}, I_C)$
$$\boldsymbol{I} = \begin{bmatrix} I_C + m(c_x^2 + c_y^2) & -m\,c_y & m\,c_x \\ -m\,c_y & m & 0 \\ m\,c_x & 0 & m \end{bmatrix}$$
end

function $\boldsymbol{v} = \mathrm{XtoV}(\boldsymbol{X})$
   version 1:
$$\boldsymbol{v} = \begin{bmatrix} X_{23} \\ X_{31} \\ -X_{21} \end{bmatrix}$$
   version 2:
$$\boldsymbol{v} = \begin{bmatrix} X_{23} \\ (X_{31} + X_{22}X_{31} + X_{23}X_{21})/2 \\ (-X_{21} - X_{22}X_{21} + X_{23}X_{31})/2 \end{bmatrix}$$
end

Table A.1: Planar vector arithmetic functions

The expressions $\boldsymbol{v}\times$ and $\boldsymbol{v}\times^*$ in these functions are the *names* of the output variables. Strictly speaking, one does not really need both functions, since $\mathrm{crf}(\boldsymbol{v}) = -\mathrm{crm}(\boldsymbol{v})^{\mathrm{T}}$.

The function mcI constructs a rigid-body inertia matrix for a planar rigid body. The arguments m, $\boldsymbol{c} = [c_x\ c_y]^{\mathrm{T}}$ and I_C specify the body's mass, the position of its centre of mass, and its rotational inertia about the centre of mass, respectively. This function would typically be used to initialize the inertia matrices in a system-model data structure.

Finally, the function XtoV calculates a motion vector from a coordinate transform in the following manner. Given any two coordinate frames, A and B, there exists a velocity, $\boldsymbol{u}$, with the property that if frame A travels at this velocity for one time unit, then it will end up coincident with frame B. The argument to XtoV is the transform from A to B coordinates, which contains all the information needed to work out where frame B is relative to frame A; and the computed vector, $\boldsymbol{v}$, is an approximation to $\boldsymbol{u}$ that converges to the exact value as the angle between A and B approaches zero. Thus, $\boldsymbol{v}$ is exactly equal to $\boldsymbol{u}$ when $\boldsymbol{X}$ is a pure translation. Two versions of this function are given. The first is simpler, but the second is more accurate, and has two special properties:

(1) v is an invariant of X, i.e., $v = Xv$, and (2) v is a scalar multiple of u, and the two are related by the formula $u = v \sin^{-1}(X_{23})/X_{23}$. This function is used to calculate position errors in kinematic constraint equations. Its main use in dynamics is to calculate loop-closure position errors for use in loop constraint stabilization (see §8.3).

A.2 Simple Spatial Arithmetic

Almost every operation in spatial vector arithmetic is a standard matrix operation. Thus, if one has access to a general-purpose matrix arithmetic tool, like Matlab, then it can provide nearly all of the necessary operations. Only a few extra functions are required, and these are shown in Table A.2.

The functions rotx, roty and rotz construct Plücker transform matrices for rotations of the coordinate frame about the x, y and z axes, respectively. In particular, X transforms a motion vector from A to B coordinates, where frame B is rotated by an angle θ relative to frame A about their common x, y or z axis, as appropriate. In general, the transform matrix for a force vector can be obtained using the formula $X^* = X^{-\mathrm{T}}$. However, for a pure rotation, it just so happens that $X^* = X$. The functions rx, ry and rz calculate the appropriate 3×3 rotation matrices using the formulae at the top of the table.

The function xlt constructs the Plücker transform matrix for a translation of the coordinate frame by an amount r. The argument is a 3D vector specifying the position of frame B relative to frame A, and the returned value is the Plücker transform from A to B coordinates.

The functions crm and crf implement the spatial cross operators. In both cases, the argument is a motion vector representing a spatial velocity. The expressions $v_{1:3}$ and $v_{4:6}$ denote 3D vectors formed from the first three and last three Plücker coordinates of v, respectively; the expressions $v_{1:3}\times$ and $v_{4:6}\times$ are the 3×3 cross-product matrices calculated from $v_{1:3}$ and $v_{4:6}$ according to the formula at the top of the table; but the expressions $v\times$ and $v\times^*$ in these two functions are the *names* of the output variables.

The function mcI constructs a rigid-body inertia matrix. The arguments specify the body's mass, the position of its centre of mass, and its rotational inertia about the centre of mass. c is a 3D vector, and I_C is a 3×3 matrix.

Finally, the function XtoV calculates a motion vector from a coordinate transform in the same way as the equivalent planar function. Given any two coordinate frames, A and B, there exists a velocity, u, with the property that if frame A were to travel at this velocity for one time unit, then it would end up coincident with frame B. The argument to this function is the transform from A to B coordinates, which contains all the information needed to work out where frame B is relative to frame A; and the computed vector, v, is an approximation to u that converges to the exact value as the angle between A and B approaches zero. Thus, v is exactly equal to u when X is a pure translation. v is also an invariant of X, so $v = Xv = X^{-1}v$, but it is not a scalar multiple of u. This

$$\mathrm{rx}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} \qquad \mathrm{ry}(\theta) = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} \qquad \mathrm{rz}(\theta) = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{v}\times = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \qquad\qquad\qquad \begin{aligned} c &= \cos(\theta) \\ s &= \sin(\theta) \end{aligned}$$

---

function $\boldsymbol{X} = \mathrm{rotx}(\theta)$
  $\boldsymbol{E} = \mathrm{rx}(\theta)$
  $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{E} \end{bmatrix}$
end

function $\boldsymbol{X} = \mathrm{roty}(\theta)$
  $\boldsymbol{E} = \mathrm{ry}(\theta)$
  $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{E} \end{bmatrix}$
end

function $\boldsymbol{X} = \mathrm{rotz}(\theta)$
  $\boldsymbol{E} = \mathrm{rz}(\theta)$
  $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{E} \end{bmatrix}$
end

function $\boldsymbol{X} = \mathrm{xlt}(\boldsymbol{r})$
  $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{1}_{3\times3} & \boldsymbol{0}_{3\times3} \\ -\boldsymbol{r}\times & \boldsymbol{1}_{3\times3} \end{bmatrix}$
end

function $\boldsymbol{v}\times = \mathrm{crm}(\boldsymbol{v})$
  $\boldsymbol{v}\times = \begin{bmatrix} \boldsymbol{v}_{1:3}\times & \boldsymbol{0}_{3\times3} \\ \boldsymbol{v}_{4:6}\times & \boldsymbol{v}_{1:3}\times \end{bmatrix}$
end

function $\boldsymbol{v}\times^* = \mathrm{crf}(\boldsymbol{v})$
  $\boldsymbol{v}\times^* = -\mathrm{crm}(\boldsymbol{v})^{\mathrm{T}}$
end

function $\boldsymbol{I} = \mathrm{mcI}(m, \boldsymbol{c}, \boldsymbol{I}_C)$
  $\boldsymbol{I} = \begin{bmatrix} \boldsymbol{I}_C - m\,\boldsymbol{c}\times\boldsymbol{c}\times & m\,\boldsymbol{c}\times \\ -m\,\boldsymbol{c}\times & m\,\boldsymbol{1}_{3\times3} \end{bmatrix}$
end

function $\boldsymbol{v} = \mathrm{XtoV}(\boldsymbol{X})$
  $\boldsymbol{v} = \dfrac{1}{2} \begin{bmatrix} X_{23} - X_{32} \\ X_{31} - X_{13} \\ X_{12} - X_{21} \\ X_{53} - X_{62} \\ X_{61} - X_{43} \\ X_{42} - X_{51} \end{bmatrix}$
end

Table A.2: Spatial vector arithmetic functions and supporting 3D formulae

Quantity	Mathematical Form	Data Structure	Size
motion vector	$\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix}$	$\mathrm{mv}(\boldsymbol{\omega}, \boldsymbol{v})$	6
force vector	$\begin{bmatrix} \boldsymbol{n} \\ \boldsymbol{f} \end{bmatrix}$	$\mathrm{fv}(\boldsymbol{n}, \boldsymbol{f})$	6
Plücker transform	$\begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ -\boldsymbol{E}\,\boldsymbol{r}\times & \boldsymbol{E} \end{bmatrix}$	$\mathrm{plx}(\boldsymbol{E}, \boldsymbol{r})$	12
rigid-body inertia	$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{h}\times \\ -\boldsymbol{h}\times & m\,\boldsymbol{1} \end{bmatrix}$	$\mathrm{rbi}(m, \boldsymbol{h}, \mathrm{lt}(\boldsymbol{I}))$	10
artic.-body inertia	$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{H} \\ \boldsymbol{H}^{\mathrm{T}} & \boldsymbol{M} \end{bmatrix}$	$\mathrm{abi}(\mathrm{lt}(\boldsymbol{M}), \boldsymbol{H}, \mathrm{lt}(\boldsymbol{I}))$	21

Table A.3: Data structures for compact representation of spatial quantities

function is used to calculate position errors in kinematic constraint equations. Its main use in dynamics is to calculate loop-closure position errors for use in loop constraint stabilization (see §8.3).

A.3 Compact Representations

Most of the inefficiencies in the pure matrix implementation of spatial arithmetic come from the use of general 6×6 matrices to represent Plücker transforms, rigid-body inertias and articulated-body inertias. To give an extreme example, the calculation of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{X}$, where $\boldsymbol{X}$ is a Plücker transform and $\boldsymbol{I}$ is a rigid-body inertia, would require 432 multiplications and 360 additions (two $6 \times 6 \times 6$ matrix multiplications) if $\boldsymbol{X}$ and $\boldsymbol{I}$ are treated as general 6×6 matrices; but it is possible to perform this operation in only 52 multiplications and 56 additions if one exploits fully the special structure and properties of these two matrices. This source of inefficiency can be eliminated by defining special data structures to store these quantities, and special functions that implement optimized versions of the operations of spatial arithmetic.

Table A.3 shows how this can be done. Expressions like $\mathrm{mv}(\boldsymbol{\omega}, \boldsymbol{v})$ are a short-hand way of specifying the type and value of a data object. In the case of $\mathrm{mv}(\boldsymbol{\omega}, \boldsymbol{v})$, the type is 'motion vector' and the value is a pair of 3D vectors. On comparing this expression with the mathematical quantity it represents, it can be seen that $\boldsymbol{\omega}$ contains the first three Plücker coordinates of a spatial motion vector, and $\boldsymbol{v}$ contains the rest. In the data structures for inertias, the expression $\mathrm{lt}(\cdots)$ means 'lower triangle of $\cdots$'. Thus, the data structure $\mathrm{rbi}(m, \boldsymbol{h}, \mathrm{lt}(\boldsymbol{I}))$ contains the scalar m, the 3D vector $\boldsymbol{h}$ and the lower triangle

of the 3×3 matrix $\boldsymbol{I}$, making a total of $1 + 3 + 6 = 10$ numbers.

Strictly speaking, the data types mv and fv are no more compact than the 6D vectors they represent, which raises the question of why they are worth having. The answer is *type safety*. By forcing the programmer to declare each vector variable to be either a motion vector or a force vector, it becomes possible for the compiler to detect a variety of incorrect expressions, like $\boldsymbol{m} + \boldsymbol{f}$, $\boldsymbol{m} \cdot \boldsymbol{m}$, $\boldsymbol{f} \times \boldsymbol{f}$, $\boldsymbol{f} = \boldsymbol{m}$, $\boldsymbol{If}$ and $\boldsymbol{X}^* \boldsymbol{m}$. With some programming languages, it is possible to go even further, and have the compiler automatically select the correct version of an operation from the types of its operands. An example is shown in the following code fragment (in a fictitious programming language).

```
variable  X : PluXfm;
variable  f1, f2 : SpFvec;
variable  I1, I2 : SpRBI;
X := plx(......);
f1 := fv(......);
I1 := rbi(......);
f2 := X.apply(f1);
I2 := X.apply(I1);
```

In this example, the first six lines declare five variables and initialize three of them; and then the last two lines apply the Plücker transform described by X to the force vector described by f1 and the rigid-body inertia described by I1. The idea here is that the function `apply` exists in several different versions in the software library—one for each kind of object that can be transformed—and the compiler picks the right one for the type of argument. If the argument is a force vector, then the compiler picks the version that calculates $\boldsymbol{X}^* \boldsymbol{f}$; and if the argument is a rigid-body inertia, then the compiler picks the version that calculates $\boldsymbol{X}^* \boldsymbol{I} \boldsymbol{X}^{-1}$. By designing the software in this manner, it is possible to relieve the programmer from having to remember the coordinate transformation rule for each type of object.

Any well-designed spatial arithmetic package would also include a function `invApply`, which would apply the inverse of the transform described by X. Thus, if we were to add the lines

```
variable  f3 : SpFvec;
f3 := X.invApply(f2);
```

to the above example, then f3 would be identical to f1 except for rounding errors. This is more efficient than explicitly inverting X and applying the inverse. Again, the compiler would pick the appropriate version of `invApply` according to the type of object to be transformed.

Table A.4 lists most of the arithmetic operations that should be provided in an efficient spatial arithmetic package, together with the formulae for implementing them using compact data structures, and their computational cost

Operation	Formula	Cost
$\alpha\,\hat{\boldsymbol{v}}$	$\mathrm{mv}(\,\alpha\boldsymbol{\omega},\ \alpha\boldsymbol{v}\,)$	6m
$\alpha\,\hat{\boldsymbol{f}}$	$\mathrm{fv}(\,\alpha\boldsymbol{n},\ \alpha\boldsymbol{f}\,)$	6m
$\alpha\,\hat{\boldsymbol{I}}$	$\mathrm{rbi}(\,\alpha m,\ \alpha\boldsymbol{h},\ \mathrm{lt}(\alpha\boldsymbol{I})\,)$	10m
$\alpha\,\boldsymbol{I}^A$	$\mathrm{abi}(\,\mathrm{lt}(\alpha\boldsymbol{M}),\ \alpha\boldsymbol{H},\ \mathrm{lt}(\alpha\boldsymbol{I})\,)$	21m
$\hat{\boldsymbol{v}}_1 + \hat{\boldsymbol{v}}_2$	$\mathrm{mv}(\,\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2,\ \boldsymbol{v}_1 + \boldsymbol{v}_2\,)$	6a
$\hat{\boldsymbol{f}}_1 + \hat{\boldsymbol{f}}_2$	$\mathrm{fv}(\,\boldsymbol{n}_1 + \boldsymbol{n}_2,\ \boldsymbol{f}_1 + \boldsymbol{f}_2\,)$	6a
$\hat{\boldsymbol{I}}_1 + \hat{\boldsymbol{I}}_2$	$\mathrm{rbi}(\,m_1 + m_2,\ \boldsymbol{h}_1 + \boldsymbol{h}_2,\ \mathrm{lt}(\boldsymbol{I}_1 + \boldsymbol{I}_2)\,)$	10a
$\boldsymbol{I}_1^A + \boldsymbol{I}_2^A$	$\mathrm{abi}(\,\mathrm{lt}(\boldsymbol{M}_1 + \boldsymbol{M}_2),\ \boldsymbol{H}_1 + \boldsymbol{H}_2,\ \mathrm{lt}(\boldsymbol{I}_1 + \boldsymbol{I}_2)\,)$	21a
$\boldsymbol{I}_1^A + \hat{\boldsymbol{I}}_2$	$\mathrm{abi}(\,\mathrm{lt}(\boldsymbol{M}_1 + m_2\boldsymbol{1}),\ \boldsymbol{H}_1 + h_2\times,\ \mathrm{lt}(\boldsymbol{I}_1 + \boldsymbol{I}_2)\,)$	15a
$\hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{f}}$	$\boldsymbol{\omega}\cdot\boldsymbol{n} + \boldsymbol{v}\cdot\boldsymbol{f}$	6m + 5a
$\hat{\boldsymbol{v}}_1 \times \hat{\boldsymbol{v}}_2$	$\mathrm{mv}(\,\boldsymbol{\omega}_1\times\boldsymbol{\omega}_2,\ \boldsymbol{\omega}_1\times\boldsymbol{v}_2 + \boldsymbol{v}_1\times\boldsymbol{\omega}_2\,)$	18m + 12a
$\hat{\boldsymbol{v}} \times^* \hat{\boldsymbol{f}}$	$\mathrm{fv}(\,\boldsymbol{\omega}\times\boldsymbol{n} + \boldsymbol{v}\times\boldsymbol{f},\ \boldsymbol{\omega}\times\boldsymbol{f}\,)$	18m + 12a
$\hat{\boldsymbol{I}}\hat{\boldsymbol{v}}$	$\mathrm{fv}(\,\boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{h}\times\boldsymbol{v},\ m\boldsymbol{v} - \boldsymbol{h}\times\boldsymbol{\omega}\,)$	24m + 18a
$\boldsymbol{I}^A\hat{\boldsymbol{v}}$	$\mathrm{fv}(\,\boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{H}\boldsymbol{v},\ \boldsymbol{M}\boldsymbol{v} + \boldsymbol{H}^{\mathrm{T}}\boldsymbol{\omega}\,)$	36m + 30a
$\boldsymbol{X}_1\boldsymbol{X}_2$	$\mathrm{plx}(\,\boldsymbol{E}_1\boldsymbol{E}_2,\ \boldsymbol{r}_2 + \boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{r}_1\,)$	33m + 24a
$\boldsymbol{X}^{-1}$	$\mathrm{plx}(\,\boldsymbol{E}^{\mathrm{T}},\ -\boldsymbol{E}\boldsymbol{r}\,)$	9m + 6a
$\boldsymbol{X}\hat{\boldsymbol{v}}$	$\mathrm{mv}(\,\boldsymbol{E}\boldsymbol{\omega},\ \boldsymbol{E}(\boldsymbol{v} - \boldsymbol{r}\times\boldsymbol{\omega})\,)$	24m + 18a
$\boldsymbol{X}^{-1}\hat{\boldsymbol{v}}$	$\mathrm{mv}(\,\boldsymbol{E}^{\mathrm{T}}\boldsymbol{\omega},\ \boldsymbol{E}^{\mathrm{T}}\boldsymbol{v} + \boldsymbol{r}\times\boldsymbol{E}^{\mathrm{T}}\boldsymbol{\omega}\,)$	24m + 18a
$\boldsymbol{X}^*\hat{\boldsymbol{f}}$	$\mathrm{fv}(\,\boldsymbol{E}(\boldsymbol{n} - \boldsymbol{r}\times\boldsymbol{f}),\ \boldsymbol{E}\boldsymbol{f}\,)$	24m + 18a
$\boldsymbol{X}^{\mathrm{T}}\hat{\boldsymbol{f}}$	$\mathrm{fv}(\,\boldsymbol{E}^{\mathrm{T}}\boldsymbol{n} + \boldsymbol{r}\times\boldsymbol{E}^{\mathrm{T}}\boldsymbol{f},\ \boldsymbol{E}^{\mathrm{T}}\boldsymbol{f}\,)$	24m + 18a
$\boldsymbol{X}^*\hat{\boldsymbol{I}}\boldsymbol{X}^{-1}$	$\mathrm{rbi}(\,m,\ \boldsymbol{E}(\boldsymbol{h} - m\boldsymbol{r}),$ $\mathrm{lt}(\boldsymbol{E}(\boldsymbol{I} + \boldsymbol{r}\times\boldsymbol{h}\times + (\boldsymbol{h} - m\boldsymbol{r})\times\boldsymbol{r}\times)\boldsymbol{E}^{\mathrm{T}})\,)$	52m + 56a
$\boldsymbol{X}^{\mathrm{T}}\hat{\boldsymbol{I}}\boldsymbol{X}$	$\mathrm{rbi}(\,m,\ \boldsymbol{E}^{\mathrm{T}}\boldsymbol{h} + m\boldsymbol{r},\ \mathrm{lt}(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{E} -$ $\boldsymbol{r}\times(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{h})\times - (\boldsymbol{E}^{\mathrm{T}}\boldsymbol{h} + m\boldsymbol{r})\times\boldsymbol{r}\times)\,)$	52m + 56a
$\boldsymbol{X}^*\boldsymbol{I}^A\boldsymbol{X}^{-1}$	$\mathrm{abi}(\,\mathrm{lt}(\boldsymbol{E}\boldsymbol{M}\boldsymbol{E}^{\mathrm{T}}),\ \boldsymbol{E}(\boldsymbol{H} - \boldsymbol{r}\times\boldsymbol{M})\boldsymbol{E}^{\mathrm{T}},$ $\mathrm{lt}(\boldsymbol{E}(\boldsymbol{I} - \boldsymbol{r}\times\boldsymbol{H}^{\mathrm{T}} + (\boldsymbol{H} - \boldsymbol{r}\times\boldsymbol{M})\boldsymbol{r}\times)\boldsymbol{E}^{\mathrm{T}})\,)$	137m + 137a
$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{I}^A\boldsymbol{X}$	$\mathrm{abi}(\,\mathrm{lt}(\boldsymbol{M}'),\ \boldsymbol{H}' + \boldsymbol{r}\times\boldsymbol{M}',$ $\mathrm{lt}(\boldsymbol{E}^{\mathrm{T}}\boldsymbol{I}\boldsymbol{E} + \boldsymbol{r}\times\boldsymbol{H}'^{\mathrm{T}} - (\boldsymbol{H}' + \boldsymbol{r}\times\boldsymbol{M}')\boldsymbol{r}\times))$ where $\boldsymbol{M}' = \boldsymbol{E}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{E}$ and $\boldsymbol{H}' = \boldsymbol{E}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{E}$	137m + 137a

Table A.4: Spatial arithmetic operation formulae and cost figures

figures. The symbols m and a in the cost column refer to floating-point multiplications and additions, respectively, with subtractions counted as additions. To avoid name clashes, the symbols $\hat{\boldsymbol{v}}$, $\hat{\boldsymbol{f}}$ and $\hat{\boldsymbol{I}}$ denote spatial motion (velocity), force and rigid-body inertia, respectively. The cost figures quoted here are for optimized implementations of the calculations. Thus, a calculation like $\boldsymbol{r}{\times}\boldsymbol{M}$, for example, is treated as the product of a skew-symmetric matrix with a symmetric one, which costs only $15\mathsf{m} + 9\mathsf{a}$. Likewise, a calculation like $\boldsymbol{E}\boldsymbol{M}\boldsymbol{E}^{\mathrm{T}}$ is assumed to be implemented using the appropriate efficiency trick from Section A.5, which costs only $28\mathsf{m}+29\mathsf{a}$. The coordinate transform formulae that would be used by X.apply(...) and X.invApply(...) are as follows:

X.apply(...)	X.invApply(...)
$\boldsymbol{X}\hat{\boldsymbol{v}}$	$\boldsymbol{X}^{-1}\hat{\boldsymbol{v}}$
$\boldsymbol{X}^{*}\hat{\boldsymbol{f}}$	$\boldsymbol{X}^{\mathrm{T}}\hat{\boldsymbol{f}}$
$\boldsymbol{X}^{*}\hat{\boldsymbol{I}}\boldsymbol{X}^{-1}$	$\boldsymbol{X}^{\mathrm{T}}\hat{\boldsymbol{I}}\boldsymbol{X}$
$\boldsymbol{X}^{*}\boldsymbol{I}^{A}\boldsymbol{X}^{-1}$	$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{I}^{A}\boldsymbol{X}$

Bear in mind that the cost figures in this table are not the last word in efficiency. In particular, significant further improvements can be obtained by using axial screw transforms instead of general Plücker transforms, as explained in Section A.4.

To see how the formulae and cost figures in Table A.4 are obtained, let us examine the operation $\boldsymbol{X}^{*}\hat{\boldsymbol{I}}\boldsymbol{X}^{-1}$ in detail. Expressed in terms of $6{\times}6$ matrices, we have

$$
\begin{aligned}
\boldsymbol{X}^{*}\hat{\boldsymbol{I}}\boldsymbol{X}^{-1} &= \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} & -\boldsymbol{r}\times \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{h}\times \\ -\boldsymbol{h}\times & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{r}\times & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{I}+\boldsymbol{r}\times\boldsymbol{h}\times & \boldsymbol{h}\times - m\boldsymbol{r}\times \\ -\boldsymbol{h}\times & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{r}\times & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{I}+\boldsymbol{r}\times\boldsymbol{h}\times & \boldsymbol{y}\times \\ -\boldsymbol{h}\times & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{r}\times & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{I}+\boldsymbol{r}\times\boldsymbol{h}\times + \boldsymbol{y}\times\boldsymbol{r}\times & \boldsymbol{y}\times \\ -\boldsymbol{h}\times + m\boldsymbol{r}\times & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z} & \boldsymbol{y}\times \\ -\boldsymbol{y}\times & m\boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E}^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}} & \boldsymbol{E}\boldsymbol{y}\times\boldsymbol{E}^{\mathrm{T}} \\ -\boldsymbol{E}\boldsymbol{y}\times\boldsymbol{E}^{\mathrm{T}} & m\boldsymbol{1} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}} & (\boldsymbol{E}\boldsymbol{y})\times \\ -(\boldsymbol{E}\boldsymbol{y})\times & m\boldsymbol{1} \end{bmatrix},
\end{aligned}
$$

where

$$\boldsymbol{y} = \boldsymbol{h} - m\boldsymbol{r} \quad \text{and} \quad \boldsymbol{Z} = \boldsymbol{I} + \boldsymbol{r}\times\boldsymbol{h}\times + \boldsymbol{y}\times\boldsymbol{r}\times .$$

This agrees completely with the formula in the table. To obtain the cost figure, we proceed as follows:

quantity	cost	quantity	cost
$\boldsymbol{h} - m\boldsymbol{r}$	3m + 3a	$\mathrm{lt}(\boldsymbol{I} + \boldsymbol{r}\times\boldsymbol{h}\times + \boldsymbol{y}\times\boldsymbol{r}\times)$	12a
$\mathrm{lt}(\boldsymbol{r}\times\boldsymbol{h}\times)$	6m + 3a	$\boldsymbol{E}\boldsymbol{y}$	9m + 6a
$\mathrm{lt}(\boldsymbol{y}\times\boldsymbol{r}\times)$	6m + 3a	$\boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}}$	28m + 29a
		Total:	52m + 56a

(See Section A.5 for how to calculate $\boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}}$.)

A.4   Axial Screw Transforms

An axial screw transform is a combination of a rotation about and translation along a single coordinate axis. We shall use the expressions $\mathrm{scrx}(\theta, d)$, $\mathrm{scry}(\theta, d)$ and $\mathrm{scrz}(\theta, d)$ to denote axial screw transforms about the x, y and z axes, respectively, in which the angle of rotation is θ and the amount of translation is d. Thus,

$$\mathrm{scrx}(\theta, d) = \mathrm{rotx}(\theta)\, \mathrm{xlt}([d\ 0\ 0]^{\mathrm{T}}) = \mathrm{xlt}([d\ 0\ 0]^{\mathrm{T}})\, \mathrm{rotx}(\theta)$$

(because the rotation and translation commute), and so on. A compact data structure describing an axial screw transform would contain the three basic quantities $c = \cos(\theta)$, $s = \sin(\theta)$ and d, as well as the derived quantities s^2, cs, $2cs$ and $1 - 2s^2$, which are used to reduce the recurrent cost of rotating matrices, as explained in Section A.5.

Axial screw transforms are useful because they can be performed efficiently. Table A.5 shows the cost of applying an axial screw transform to various spatial quantities. The quantity $\boldsymbol{I}^a$ has been included because of its relevance to the articulated-body algorithm. It will be discussed later in this section. The total cost has been divided into a recurring cost, which is the cost of applying the transform, and a one-time cost, which is the cost of calculating the quantities s^2, cs, $2cs$ and $1 - 2s^2$ mentioned above. If the angle is a constant, then this

transformed quantity	computational cost recurring	one-time	screw axis
$\boldsymbol{m}, \boldsymbol{f}$	10m + 6a		
$\boldsymbol{I}$	16m + 15a	2m + 3a	
$\boldsymbol{I}^A$	38m + 40a	2m + 3a	
$\boldsymbol{I}^a$	28m + 34a	2m + 3a	z
$\boldsymbol{I}^a$	32m + 28a	2m + 3a	x, y

row and column 3 of $\boldsymbol{I}^a$ are zero

Table A.5: Computational costs of axial screw transforms

calculation can be performed off-line, or at the start of program execution; otherwise, it must be performed each time the rotation angle changes.

On comparing the costs in Table A.5 with those in Table A.4, it can be seen that the cost of an axial screw transform varies from about 40% (for a vector) to less than a third (for an inertia) of the cost of a general Plücker transform. Now, a general Plücker transform can be achieved by three successive axial screws; but the transform from one Denavit-Hartenberg coordinate frame to another can be achieved using only two axial screws. (See Figure 4.8.) Therefore, a significant cost saving can be achieved, relative to the cost figures in Table A.4, by using a combination of Denavit-Hartenberg frames in the system model and axial-screw transforms in the calculations.

Vectors

Written out in full, the expression for $\boldsymbol{Xm}$, where $\boldsymbol{X} = \mathrm{scrz}(\theta, d)$, is

$$\begin{bmatrix} c & s & 0 & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -sd & cd & 0 & c & s & 0 \\ -cd & -sd & 0 & -s & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_x \\ m_y \\ m_z \\ m_{Ox} \\ m_{Oy} \\ m_{Oz} \end{bmatrix} = \begin{bmatrix} cm_x + sm_y \\ cm_y - sm_x \\ m_z \\ c(m_{Ox} + m_y d) + s(m_{Oy} - m_x d) \\ c(m_{Oy} - m_x d) - s(m_{Ox} + m_y d) \\ m_{Oz} \end{bmatrix}.$$

The cost of this operation is clearly $10\mathsf{m}+6\mathsf{a}$. It is easily shown that the cost of calculating $\boldsymbol{X}^*\boldsymbol{f}$ is also $10\mathsf{m}+6\mathsf{a}$, and that the costs of screw transforms about the other two axes are the same.

Rigid-Body Inertia

Axial screw transforms are applied to rigid-body inertias using the formulae in Table A.4, but the cost of the calculation is reduced because of the special form of $\boldsymbol{E}$ and $\boldsymbol{r}$. The cost breakdown for the operation $\boldsymbol{X}^*\hat{\boldsymbol{I}}\boldsymbol{X}^{-1}$ is

| | computational cost | |
operation	recurring	one-time
$\boldsymbol{y} = \boldsymbol{h} - m\boldsymbol{r}$	$1\mathsf{m} + 1\mathsf{a}$	
$\boldsymbol{Z} = \boldsymbol{I} + \boldsymbol{r}{\times}\boldsymbol{h}{\times} + \boldsymbol{y}{\times}\boldsymbol{r}{\times}$	$3\mathsf{m} + 5\mathsf{a}$	
$\boldsymbol{E}\boldsymbol{y}$	$4\mathsf{m} + 2\mathsf{a}$	
$\boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}}$	$8\mathsf{m} + 7\mathsf{a}$	$2\mathsf{m} + 3\mathsf{a}$
Totals:	$16\mathsf{m} + 15\mathsf{a}$	$2\mathsf{m} + 3\mathsf{a}$

The calculation of $\boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}}$ is accomplished using the appropriate efficiency trick from Section A.5. The calculation of $\boldsymbol{Z}$ proceeds as follows: if $\boldsymbol{r} = [0\ 0\ d]^{\mathrm{T}}$, then

$$\mathrm{lt}(\boldsymbol{r}{\times}\boldsymbol{h}{\times} + \boldsymbol{y}{\times}\boldsymbol{r}{\times}) = \begin{bmatrix} -(h_z + y_z)d & \cdot & \cdot \\ 0 & -(h_z + y_z)d & \cdot \\ h_x d & h_y d & 0 \end{bmatrix}.$$

The cost of calculating this expression is $3\mathsf{m} + 1\mathsf{a}$, and the cost of adding it to $\mathrm{lt}(\boldsymbol{I})$ is $4\mathsf{a}$. If $\boldsymbol{r} = [d\ 0\ 0]^{\mathrm{T}}$ or $[0\ d\ 0]^{\mathrm{T}}$ then the details will be different, but the cost will be the same.

Articulated-Body Inertias

Axial screw transforms are applied to articulated-body inertias using the formulae in Table A.4, but the cost of the calculation is reduced because of the special form of $\boldsymbol{E}$ and $\boldsymbol{r}$. The cost breakdown for the operation $\boldsymbol{X}^* \boldsymbol{I}^A \boldsymbol{X}^{-1}$ is

	computational cost	
operation	recurring	one-time
$\boldsymbol{E}\boldsymbol{M}\boldsymbol{E}^{\mathrm{T}}$	$8\mathsf{m} + 7\mathsf{a}$	$2\mathsf{m} + 3\mathsf{a}$
$\boldsymbol{Y} = \boldsymbol{H} - \boldsymbol{r}{\times}\boldsymbol{M}$	$5\mathsf{m} + 6\mathsf{a}$	
$\boldsymbol{E}\boldsymbol{Y}\boldsymbol{E}^{\mathrm{T}}$	$12\mathsf{m} + 12\mathsf{a}$	$2\mathsf{m}$
$\boldsymbol{Z} = \boldsymbol{I} - \boldsymbol{r}{\times}\boldsymbol{H}^{\mathrm{T}} + \boldsymbol{Y}\boldsymbol{r}{\times}$	$5\mathsf{m} + 8\mathsf{a}$	
$\boldsymbol{E}\boldsymbol{Z}\boldsymbol{E}^{\mathrm{T}}$	$8\mathsf{m} + 7\mathsf{a}$	$2\mathsf{m} + 3\mathsf{a}$
Totals:	$38\mathsf{m} + 40\mathsf{a}$	$2\mathsf{m} + 3\mathsf{a}$

(The cost breakdown and total cost of $\boldsymbol{X}^{\mathrm{T}} \boldsymbol{I}^A \boldsymbol{X}$ are the same.) The three rotations are performed using the appropriate efficiency tricks in Section A.5. The $2\mathsf{m}$ one-time cost for calculating $\boldsymbol{E}\boldsymbol{Y}\boldsymbol{E}^{\mathrm{T}}$ is subsumed in the $2\mathsf{m} + 3\mathsf{a}$ one-time cost for the other two rotations. The cost figures for calculating $\boldsymbol{Y}$ and $\boldsymbol{Z}$ can be understood as follows. If $\boldsymbol{r} = [0\ 0\ d]^{\mathrm{T}}$, then

$$
\boldsymbol{r}{\times}\boldsymbol{M} = \begin{bmatrix} 0 & -d & 0 \\ d & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} = \begin{bmatrix} -M_{21}d & -M_{22}d & -M_{23}d \\ M_{11}d & M_{12}d & M_{13}d \\ 0 & 0 & 0 \end{bmatrix}
$$

and

$$
\mathrm{lt}(-\boldsymbol{r}{\times}\boldsymbol{H}^{\mathrm{T}} + \boldsymbol{Y}\boldsymbol{r}{\times}) = \begin{bmatrix} (H_{12} + Y_{12})d & \cdot & \cdot \\ (Y_{22} - H_{11})d & -(H_{21} + Y_{21})d & \cdot \\ H_{32}d & -H_{31}d & 0 \end{bmatrix}.
$$

The cost of calculating $\boldsymbol{r}{\times}\boldsymbol{M}$ is $5\mathsf{m}$ (because $M_{12} = M_{21}$); the cost of subtracting it from $\boldsymbol{H}$ is $6\mathsf{a}$; the cost of calculating $\mathrm{lt}(-\boldsymbol{r}{\times}\boldsymbol{H}^{\mathrm{T}} + \boldsymbol{Y}\boldsymbol{r}{\times})$ is $5\mathsf{m} + 3\mathsf{a}$; and the cost of adding it to $\mathrm{lt}(\boldsymbol{I})$ is $5\mathsf{a}$. If $\boldsymbol{r} = [d\ 0\ 0]^{\mathrm{T}}$ or $[0\ d\ 0]^{\mathrm{T}}$ then the details will be different, but the costs will be the same.

Articulated-Body Inertias with Zero-Valued Rows and Columns

The single most expensive operation in the articulated-body algorithm is the transformation of articulated-body inertias from one coordinate system to another. However, the quantity that gets transformed is not a general articulated-body inertia, but one of the form

$$
\boldsymbol{I}^a = \boldsymbol{I}^A - \boldsymbol{I}^A \boldsymbol{S} (\boldsymbol{S}^{\mathrm{T}} \boldsymbol{I}^A \boldsymbol{S})^{-1} \boldsymbol{S}^{\mathrm{T}} \boldsymbol{I}^A .
$$

This quantity has the following special property: if S is a $6 \times d$ matrix having $6 - d$ zero-valued rows, then all of the rows and columns of I^a corresponding to the nonzero rows of S will be zero. For example, if $S = [0\ 0\ 1\ 0\ 0\ 0]^\mathrm{T}$ (a revolute joint) then row and column 3 of I^a will be zero, and if $S = [\mathbf{0}_{3\times2}\ \mathbf{1}_{3\times3}\ \mathbf{0}_{3\times1}]^\mathrm{T}$ (a planar joint) then rows and columns 3, 4 and 5 of I^a will all be zero.

A significant improvement in the speed of the articulated-body algorithm can be achieved by exploiting this special property of I^a. To create a customized coordinate transformation procedure, one starts with the procedure for a general articulated-body inertia, and makes whatever simplifications accrue from the special form of I^a. Cost figures for two such customized transforms appear in Table A.5, and a cost breakdown for each appears below. In both cases, row and column 3 of I^a are zero. Note that scrz preserves the zeros in this row and column, but scrx and scry do not. In typical use, I^a is transformed from one Denavit-Hartenberg coordinate system to another by first applying scrz, and then applying scrx to the result. The total cost of this procedure, including the one-time cost for scrz only (because the x rotation angle is a constant), is $62m + 65a$. This is less than half the cost of the general transform in Table A.4, and about 10% better than the figures reported in McMillan and Orin (1995). More on this topic can be found in McMillan and Orin (1995).

z axis screw	recurring cost	**x or y axis screw**	recurring cost
EME^T	$8m + 7a$	$M' = EME^\mathrm{T}$	$8m + 7a$
$Y = H - r{\times}M$	$5m + 6a$	$H' = EHE^\mathrm{T}$	$10m + 6a$
EYE^T	$8m + 10a$	$Y = H' - r{\times}M'$	$5m + 6a$
$Z = I - r{\times}H^\mathrm{T} + Yr{\times}$	$3m + 6a$	$I' = EIE^\mathrm{T}$	$4m + 1a$
EZE^T	$4m + 5a$	$I' - r{\times}H'^\mathrm{T} + Yr{\times}$	$5m + 8a$
	Total: $28m + 34a$		Total: $32m + 28a$

A.5   Some Efficiency Tricks

Product of Rotation Matrices

Any one row or column of a 3×3 rotation matrix is the vector product of the other two. Thus, if F and G are two rotation matrices, and $E = FG$, then E can be calculated in the following manner:

$$\begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \\ E_{31} & E_{32} \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \\ G_{31} & G_{32} \end{bmatrix},$$

$$\begin{bmatrix} E_{13} \\ E_{23} \\ E_{33} \end{bmatrix} = \begin{bmatrix} E_{11} \\ E_{21} \\ E_{31} \end{bmatrix} \times \begin{bmatrix} E_{12} \\ E_{22} \\ E_{32} \end{bmatrix}.$$

The total cost of this calculation is $24m + 15a$, which amounts to a saving of $3m + 3a$ over standard matrix multiplication. More generally, the cost of the calculation $E = F_1 F_2 \cdots F_n G$ is $(18m + 12a)n + 6m + 3a$.

Rotation of a General Matrix

Consider the calculation $B = EAE^T$, where E is a rotation matrix and A is a general 3×3 matrix. This calculation can be performed as two $3 \times 3 \times 3$ multiplications, for a total cost of $54m + 36a$. However, it is possible to reduce this cost to $40m + 39a$ as follows. First, split A into three components:

$$A = L + D + v\times,$$

where

$$L = \begin{bmatrix} A_{11} - A_{33} & A_{12} & 0 \\ A_{21} & A_{22} - A_{33} & 0 \\ A_{31} + A_{13} & A_{32} + A_{23} & 0 \end{bmatrix}, \quad D = \begin{bmatrix} A_{33} & 0 & 0 \\ 0 & A_{33} & 0 \\ 0 & 0 & A_{33} \end{bmatrix}, \quad v = \begin{bmatrix} -A_{23} \\ A_{13} \\ 0 \end{bmatrix}.$$

B can now be expressed as

$$\begin{aligned} B &= E(L + D + v\times)E^T \\ &= ELE^T + D + (Ev)\times. \end{aligned}$$

Defining $Y = EL$ and $Z = YE^T$, we calculate

$$\begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \\ Y_{31} & Y_{32} \end{bmatrix} = \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \\ L_{31} & L_{32} \end{bmatrix},$$

$$\begin{bmatrix} \cdot & Z_{12} & Z_{13} \\ Z_{21} & Z_{22} & Z_{23} \\ Z_{31} & Z_{32} & Z_{33} \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \\ Y_{31} & Y_{32} \end{bmatrix} \begin{bmatrix} E_{11} & E_{21} & E_{31} \\ E_{12} & E_{22} & E_{32} \end{bmatrix}$$

and

$$Z_{11} = L_{11} + L_{22} - Z_{22} - Z_{33}.$$

(This last equation exploits the invariance of the trace under rotation. Alternatively, Z_{11} could have been calculated with the rest of Z at a cost of $2m + 1a$.) Finally, Ev is calculated by a $3 \times 2 \times 1$ matrix multiplication between the leftmost two columns of E and the top two elements of v, and the quantities Z, D and $(Ev)\times$ are summed. The total cost of this procedure is calculated as follows:

quantity	cost	quantity	cost
L	4a	Z_{11}	3a
Y	$18m + 12a$	Ev	$6m + 3a$
Z	$16m + 8a$	$Z + D + (Ev)\times$	9a
			Total: $40m + 39a$

This is slightly more than the recurring cost of three successive rotations about the coordinate axes.

Rotation of a Symmetric Matrix

Now consider the calculation $\boldsymbol{B} = \boldsymbol{E}\boldsymbol{A}\boldsymbol{E}^{\mathrm{T}}$, where $\boldsymbol{E}$ is a general 3×3 rotation matrix and $\boldsymbol{A}$ is a symmetric matrix. This calculation can be performed using the exact same procedure as for a general matrix $\boldsymbol{A}$, but with some simplifications allowed by the symmetry. Specifically, we only need to calculate the lower triangle of $\boldsymbol{B}$. As a consequence, there is no need to calculate Z_{12}, Z_{13} or Z_{23}; and, as a further consequence, there is no need to calculate Y_{11} or Y_{12}. Thus, the calculations of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ simplify to

$$
\begin{bmatrix} Y_{21} & Y_{22} \\ Y_{31} & Y_{32} \end{bmatrix} = \begin{bmatrix} E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \\ L_{31} & L_{32} \end{bmatrix}
$$

and

$$
\begin{bmatrix} Z_{21} & Z_{22} & \cdot \\ Z_{31} & Z_{32} & Z_{33} \end{bmatrix} = \begin{bmatrix} Y_{21} & Y_{22} \\ Y_{31} & Y_{32} \end{bmatrix} \begin{bmatrix} E_{11} & E_{21} & E_{31} \\ E_{12} & E_{22} & E_{32} \end{bmatrix},
$$

with Z_{11} calculated as before. The total cost of this simplified procedure is:

quantity	cost	quantity	cost
$\boldsymbol{L}$	4a	Z_{11}	3a
$\boldsymbol{Y}$	12m + 8a	$\boldsymbol{Ev}$	6m + 3a
$\boldsymbol{Z}$	10m + 5a	$\boldsymbol{Z} + \boldsymbol{D} + (\boldsymbol{Ev})\times$	6a
		Total:	28m + 29a

This is a substantial improvement on the cost of two $3 \times 3 \times 3$ matrix multiplications, but it is not quite as fast as performing three successive rotations about the coordinate axes.

Rotation of a General Matrix about a Coordinate Axis

Suppose we wish to calculate $\boldsymbol{B} = \boldsymbol{E}\boldsymbol{A}\boldsymbol{E}^{\mathrm{T}}$, where $\boldsymbol{E}$ is $\mathrm{rx}(\theta)$, $\mathrm{ry}(\theta)$ or $\mathrm{rz}(\theta)$, and $\boldsymbol{A}$ is a general 3×3 matrix. The formulae for these three rotations can be written in the following manner.[2]

$$
\mathrm{rx}(\theta)\,\boldsymbol{A}\,\mathrm{rx}(\theta)^{\mathrm{T}} = \begin{bmatrix} A_{11} & cA_{12} + sA_{13} & cA_{13} - sA_{12} \\ cA_{21} + sA_{31} & A_{22} + \alpha_x & A_{23} + \beta_x \\ cA_{31} - sA_{21} & A_{32} + \beta_x & A_{33} - \alpha_x \end{bmatrix}
$$

$$
\mathrm{ry}(\theta)\,\boldsymbol{A}\,\mathrm{ry}(\theta)^{\mathrm{T}} = \begin{bmatrix} A_{11} - \alpha_y & cA_{12} - sA_{32} & A_{13} + \beta_y \\ cA_{21} - sA_{23} & A_{22} & cA_{23} + sA_{21} \\ A_{31} + \beta_y & cA_{32} + sA_{12} & A_{33} + \alpha_y \end{bmatrix}
$$

$$
\mathrm{rz}(\theta)\,\boldsymbol{A}\,\mathrm{rz}(\theta)^{\mathrm{T}} = \begin{bmatrix} A_{11} + \alpha_z & A_{12} + \beta_z & cA_{13} + sA_{23} \\ A_{21} + \beta_z & A_{22} - \alpha_z & cA_{23} - sA_{13} \\ cA_{31} + sA_{32} & cA_{32} - sA_{31} & A_{33} \end{bmatrix}
$$

---

[2]These formulae, and those in the next trick, are based on the work of McMillan (1994).

where

$$\alpha_x = cs(A_{23} + A_{32}) + s^2(A_{33} - A_{22})$$

$$\alpha_y = cs(A_{31} + A_{13}) + s^2(A_{11} - A_{33})$$

$$\alpha_z = cs(A_{12} + A_{21}) + s^2(A_{22} - A_{11})$$

$$\beta_x = cs(A_{33} - A_{22}) - s^2(A_{23} + A_{32})$$

$$\beta_y = cs(A_{11} - A_{33}) - s^2(A_{31} + A_{13})$$

$$\beta_z = cs(A_{22} - A_{11}) - s^2(A_{12} + A_{21})$$

$$s = \sin(\theta) \quad \text{and} \quad c = \cos(\theta).$$

In all cases, the cost is the same, and is calculated as follows:

	computational cost	
quantity	recurring	one-time
cs, s^2		2m
α, β	4m + 4a	
rest of $\boldsymbol{B}$	8m + 8a	
Totals:	12m + 12a	2m

Observe that the total cost has been divided into a recurring cost and a one-time cost. The idea here is that an axial-rotation data structure is defined, which contains the values c, s, cs and s^2, and all four quantities are available at the time the rotation is performed. Thus, the one-time cost is the cost incurred when the data structure is initialized or updated,[3] and the recurring cost is the cost incurred each time a rotation is performed. This tactic allows one to exploit the fact that many rotation angles in dynamics calculations are known constants, and that even where the rotation angle varies, the data structure will often be used more than once between changes.

Rotation of a Symmetric Matrix about a Coordinate Axis

Now consider the calculation of $\boldsymbol{B} = \boldsymbol{E}\boldsymbol{A}\boldsymbol{E}^{\mathrm{T}}$, where $\boldsymbol{E}$ is rx(θ), ry(θ) or rz(θ), and $\boldsymbol{A}$ is a symmetric 3×3 matrix. The formulae for these rotations are simplified versions of the formulae for rotating a general matrix, and can be written as follows:

$$\mathrm{lt}(\,\mathrm{rx}(\theta)\,\boldsymbol{A}\,\mathrm{rx}(\theta)^{\mathrm{T}}) = \begin{bmatrix} A_{11} & \cdot & \cdot \\ cA_{21} + sA_{31} & A_{22} + \alpha_x & \cdot \\ cA_{31} - sA_{21} & \beta_x & A_{33} - \alpha_x \end{bmatrix}$$

---

[3]Strictly speaking, the cost of initializing this data structure should include the cost of performing one sine and one cosine calculation. However, these trigonometric calculations are normally excluded from the cost figures because they are assumed to be the same for all methods.

$$\mathrm{lt}(\,\mathrm{ry}(\theta)\,\boldsymbol{A}\,\mathrm{ry}(\theta)^{\mathrm{T}}) = \begin{bmatrix} A_{11} - \alpha_y & \cdot & \cdot \\ cA_{21} - sA_{23} & A_{22} & \cdot \\ \beta_y & cA_{32} + sA_{12} & A_{33} + \alpha_y \end{bmatrix}$$

$$\mathrm{lt}(\,\mathrm{rz}(\theta)\,\boldsymbol{A}\,\mathrm{rz}(\theta)^{\mathrm{T}}) = \begin{bmatrix} A_{11} + \alpha_z & \cdot & \cdot \\ \beta_z & A_{22} - \alpha_z & \cdot \\ cA_{31} + sA_{32} & cA_{32} - sA_{31} & A_{33} \end{bmatrix}$$

where

$$\alpha_x = 2cs A_{32} + s^2(A_{33} - A_{22})$$

$$\alpha_y = 2cs A_{31} + s^2(A_{11} - A_{33})$$

$$\alpha_z = 2cs A_{21} + s^2(A_{22} - A_{11})$$

$$\beta_x = cs(A_{33} - A_{22}) + (1 - 2s^2)A_{32}$$

$$\beta_y = cs(A_{11} - A_{33}) + (1 - 2s^2)A_{31}$$

$$\beta_z = cs(A_{22} - A_{11}) + (1 - 2s^2)A_{21}\,.$$

In all cases, the cost is the same, and is calculated as follows:

quantity	computational cost	
	recurring	one-time
cs, s^2		2m
$2cs$, $(1 - 2s^2)$		3a
α, β	4m + 3a	
rest of $\boldsymbol{B}$	4m + 4a	
Totals:	8m + 7a	2m + 3a

Once again, the total cost has been divided into a recurring cost and a one-time cost. The axial-rotation data structure is assumed to contain the values c, s, cs, s^2, $2cs$ and $1 - 2s^2$, all of which are available at the time the rotation is performed. Thus, the one-time cost is incurred when the data structure is initialized or updated, and the recurring cost is incurred each time a rotation is performed.

Bibliography

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

Angeles, J. 2003. *Fundamentals of Robotic Mechanical Systems* (2nd ed.) New York: Springer-Verlag.

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

Armstrong-Hélouvry, B., Dupont, P., and Canudas de Wit, C. 1994. A Survey of Models, Analysis Tools and Compensation Methods for the Control of Machines with Friction. *Automatica*, vol. 30, no. 7, pp. 1083–1138.

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

Baker, E. J., and Wohlhart, K. 1996. Motor Calculus: A New Theoretical Device for Mechanics. Graz, Austria: Institute for Mechanics, TU Graz. (Translation of von Mises 1924a and 1924b.)

Balafoutis, C. A., Patel, R. V., and Misra, P. 1988. Efficient Modeling and Computation of Manipulator Dynamics Using Orthogonal Cartesian Tensors. *IEEE J. Robotics & Automation*, vol. 4, no. 6, pp. 665–676.

Balafoutis, C. A., and Patel, R. V. 1989. Efficient Computation of Manipulator Inertia Matrices and the Direct Dynamics Problem. *IEEE Trans. Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1313–1321.

Balafoutis, C. A., and Patel, R. V. 1991. *Dynamic Analysis of Robot Manipulators: A Cartesian Tensor Approach*. Boston: Kluwer Academic Publishers.

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

Baraff, D. 1996. Linear-Time Dynamics using Lagrange Multipliers. Proc. ACM SIGGRAPH '96, New Orleans, August 4–9, pp. 137–146.

Baumgarte, J. 1972. Stabilization of Constraints and Integrals of Motion in Dynamical Systems. *Computer Methods in Applied Mechanics and Engineering*, vol. 1, pp. 1–16.

Bobrow, J. E. 1989. A Direct Minimization Approach for Obtaining the Distance Between Convex Polyhedra. *Int. J. Robotics Research*, vol. 8, no. 3, pp. 65–76.

Brach, R. M. 1991. *Mechanical Impact Dynamics*. New York: Wiley.

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

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

Brandl, H., Johanni, R., and Otter, M. 1988. A Very Efficient Algorithm for the Simulation of Robots and Similar Multibody Systems Without Inversion of the Mass Matrix. In *Theory of Robots*, P. Kopacek, I. Troch & K. Desoyer (eds.), Oxford: Pergamon Press, pp. 95–100.

Cottle, R. W., and Dantzig, G. B. 1968. Complementary Pivot Theory of Mathematical Programming. *Linear Algebra and its Applications*, vol. 1, pp. 103–125.

Cottle, R. W., Pang, J.-S., and Stone, R. E. 1992. *The Linear Complementarity Problem*. Boston: Academic Press.

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

Craig, J. J. 2005. *Introduction to Robotics: Mechanics and Control* (3ed). Upper Saddle River, NJ: Pearson Prentice Hall.

Denavit, J., and Hartenberg, R. S. 1955. A Kinematic Notation for Lower Pair Mechanisms Based on Matrices. *Trans. ASME J. Applied Mechanics*, vol. 22, pp. 215–221.

Dubowsky, S., and Papadopoulos, E. 1993. The Kinematics, Dynamics, and Control of Free-Flying and Free-Floating Space Robotic Systems. *IEEE Trans. Robotics & Automation*, vol. 9, no. 5, pp. 531–543.

Duff, I.S., Erisman, A. M., and Reid, J. K. 1986. *Direct Methods for Sparse Matrices*. Oxford: Clarendon Press.

Duffy, J. 1990. The Fallacy of Modern Hybrid Control Theory that is Based on "Orthogonal Complements" of Twist and Wrench Spaces. *J. Robotic Systems*, vol. 7, no. 2, pp. 139–144.

Dupont, P., Hayward, V., Armstrong, B., and Altpeter, F. 2002. Single State Elastoplastic Friction Models. *IEEE Trans. Automatic Control*, vol. 47, no. 5, pp. 787–792.

Fang, A. C., and Pollard, N. S. 2003. Efficient Synthesis of Physically Valid Human Motion. *ACM Trans. Graphics* (SIGGRAPH 2003), vol. 22, no. 3, pp. 417–426.

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

Featherstone, R. 1984. Robot Dynamics Algorithms. *Ph. D. Thesis*, Edinburgh University.

Featherstone, R. 1987. *Robot Dynamics Algorithms*. Boston: Kluwer Academic Publishers.

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

Featherstone, R. 1999b. 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 Research*, vol. 18, no. 9, pp. 876–892.

Featherstone, R. 2004. An Empirical Study of the Joint Space Inertia Matrix. *Int. J. Robotics Research*, vol. 23, no. 9, pp. 859–871.

Featherstone, R. 2005. Efficient Factorization of the Joint Space Inertia Matrix for Branched Kinematic Trees. *Int. J. Robotics Research*, vol. 24, no. 6, pp. 487–500.

Gautier, M., and Khalil, W. 1990. Direct Calculation of Minimum Set of Inertial Parameters of Serial Robots. *IEEE Trans. Robotics & Automation*, vol. 6, no. 3, pp. 368–373.

Gear, C. W. 1971. *Numerical Initial Value Problems in Ordinary Differential Equations*. Englewood Cliffs, NJ: Prentice-Hall.

George, A., and Liu. J. W. H. 1981. *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs, NJ: Prentice-Hall.

Gilardi, G., and Sharf, I. 2002. Literature Survey of Contact Dynamics Modelling. *Mechanism & Machine Theory*, vol. 37, no. 10, pp. 1213–1239.

Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. 1988. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE J. Robotics & Automation*, vol. 4, no. 2, pp. 193–203.

Gottschalk, S., Lin, M. C., and Manocha, D. 1996. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Proc. ACM SIGGRAPH '96, New Orleans, August 4–9*, pp. 171–180.

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

Haessig, D. A., and Friedland, B. 1991. On the Modeling and Simulation of Friction. *Trans. ASME J. Dynamic Systems, Measurement & Control*, vol. 113, no. 3, pp. 354–362.

He, X., and Goldenberg, A. A. 1989. An Algorithm for Efficient Computation of Dynamics of Robotic Manipulators. *Proc. Fourth Int. Conf. Advanced Robotics*, (Columbus, OH), pp. 175–188, June.

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

Hooker, W. W., and Margulies, G. 1965. The Dynamical Attitude Equations for an n-Body Satellite. *J. Astronautical Sciences*, vol. 12, no. 4, pp. 123–128.

Hooker, W. W. 1970. A Set of r Dynamical Attitude Equations for an Arbitrary n-Body Satellite having r Rotational Degrees of Freedom. *AIAA Journal*, vol. 8(2), no. 7, pp. 1205–1207.

Hu, W., Marhefka, D. W., and Orin, D. E. 2005. Hybrid Kinematic and Dynamic Simulation of Running Machines. *IEEE Trans. Robotics*, vol. 21, no. 3, pp. 490–497.

Hunt, K. H. 1978. *Kinematic Geometry of Mechanisms*. Oxford: Oxford University Press.

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

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

Jain, A., and Rodriguez, G. 1993. An Analysis of the Kinematics and Dynamics of Underactuated Manipulators. *IEEE Trans. Robotics and Automation*, vol. 9, no. 4, pp. 411–422.

Jimenez, P., Thomas, F., and Torras, C. 2001. 3D Collision Detection: A Survey. *Computers & Graphics-UK*, vol. 25, no. 2, pp. 269–285.

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

Khalil, W., and Kleinfinger, J. F. 1987. Minimum Operations and Minimum Parameters of the Dynamic Models of Tree Structure Robots. *IEEE J. Robotics & Automation*, vol. 3, no. 6, pp. 517–526.

Khalil, W., and Dombre, E. 2002. *Modeling, Identification and Control of Robots*. New York, NY: Taylor & Francis.

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

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

Lathrop, R. H. 1985. Parallelism in Manipulator Dynamics. *Int. J. Robotics Research*, vol. 4, no. 2, pp. 80–102.

Lilly, K. W., and Orin, D. E. 1991. Alternate Formulations for the Manipulator Inertia Matrix. *Int. J. Robotics Research*, vol. 10, no. 1, pp. 64–74.

Lilly, K. W. 1993. *Efficient Dynamic Simulation of Robotic Mechanisms*. Boston: Kluwer Academic Publishers.

Lötstedt, P. 1981. Coulomb Friction in Two-Dimensional Rigid Body Systems. *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 61, no. 12, pp. 605–615.

Lötstedt, P. 1982. Mechanical Systems of Rigid Bodies Subject to Unilateral Constraints. *SIAM J. Applied Mathematics*, vol. 42, no. 2, pp. 281–296.

Lötstedt, P. 1984. Numerical Simulation of Time-Dependent Contact and Friction Problems in Rigid Body Mechanics. *SIAM J. Scientific and Statistical Computing*, vol. 5, no. 2, pp. 370–393.

Longman, R. W., Lindberg, R. E., and Zedd, M. F. 1987. Satellite-Mounted Robot Manipulators—New Kinematics and Reaction Moment Compensation. *Int. J. Robotics Research*, vol. 6, no. 3, pp. 87–103.

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

Luh, J. Y. S., Walker, M. W., and Paul, R. P. C. 1980b. Resolved-Acceleration Control of Mechanical Manipulators. *IEEE Trans. Automatic Control*, vol. 25, no. 3, pp. 468–474.

McMillan, S. 1994. Computational Dynamics for Robotic Systems on Land and Under Water. Ph. D. Thesis, The Ohio State University, Dept. Electrical Engineering.

McMillan, S., and Orin, D. E. 1995. Efficient Computation of Articulated-Body Inertias Using Successive Axial Screws. *IEEE Trans. Robotics & Automation*, vol. 11, no. 4, pp. 606–611.

McMillan, S., Orin, D. E., and McGhee, R. B. 1995. Efficient Dynamic Simulation of an Underwater Vehicle with a Robotic Manipulator. *IEEE Trans. Systems, Man & Cybernetics*, vol. 25, no. 8, pp. 1194–1206.

McMillan, S., and Orin, D. E. 1998. Forward Dynamics of Multilegged Vehicles Using the Composite Rigid Body Method. *Proc. IEEE Int. Conf. Robotics and Automation*, (Leuven, Belgium), pp. 464–470, May.

Marhefka, D. W., and Orin, D. E. 1999. A Compliant Contact Model with Nonlinear Damping for Simulation of Robotic Systems. *IEEE Trans. Systems, Man & Cybernetics—Part A*, vol. 29, no. 6, pp. 566–572.

Mirtich, B. 1998. V-Clip: Fast and Robust Polyhedral Collision Detection. *ACM Trans. Graphics*, vol. 17, no. 3, pp. 177–208.

von Mises, R. 1924a. Motorrechnung, ein neues Hilfsmittel der Mechanik [Motor Calculus: a new Theoretical Device for Mechanics]. *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 4, no. 2, pp. 155–181. (English translation: Baker and Wohlhart 1996.)

von Mises, R. 1924b. Anwendungen der Motorrechnung [Applications of Motor Calculus]. *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 4, no. 3, pp. 193–213. (English translation: Baker and Wohlhart 1996.)

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

Murray, J. J., and Neuman, C. P. 1984. ARM: An Algebraic Robot Dynamic Modeling Program. *Proc. IEEE Int. Conf. Robotics and Automation*, Atlanta, Georgia, pp. 103–114, March 13–15.

Murray, R. M., Li, Z., and Sastry, S. S. 1994. *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC Press.

Nakamura, Y., and Yamane, K. 2000. Dynamics Computation of Structure-Varying Kinematic Chains and Its Application to Human Figures. *IEEE Trans. Robotics and Automation*, vol. 16, no. 2, pp. 124–134.

Neuman, C. P., and Murray, J. J. 1987. Customized Computational Robot Dynamics. *J. Robotic Systems*, vol. 4, no. 4, pp. 503–526.

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

Orlandea, N., Chace, M. A., and Calahan, D. A. 1977. A Sparsity-Oriented Approach to the Dynamic Analysis and Design of Mechanical Systems—Part 1. *Trans. ASME J. Engineering for Industry*, vol. 99, no. 3, pp. 773–779.

Park, F. C., Bobrow, J. E., and Ploen, S. R. 1995. A Lie Group Formulation of Robot Dynamics. *Int. J. Robotics Research*, vol. 14, no. 6, pp. 609–618.

Paul, B. 1975. Analytical Dynamics of Mechanisms—A Computer Oriented Overview. *Mechanism and Machine Theory*, vol. 10, no. 6, pp. 481–507.

Pfeiffer, F., and Glocker, C. (eds) 2000. *Multibody Dynamics with Unilateral Contacts*. Vienna: Springer-Verlag.

Plücker, J. 1866. Fundamental Views Regarding Mechanics. *Philosophical Transactions*, vol. 156, pp. 361–380.

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

Rodriguez, G. 1987. Kalman Filtering, Smoothing, and Recursive Robot Arm Forward and Inverse Dynamics. *IEEE J. Robotics & Automation*, vol. RA-3, no. 6, pp. 624–639.

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

Rooney, J. 1977. A Survey of Representations of Spatial Rotations About a Fixed Point. *Environment and Planning B*, vol. 4, no. 2, pp. 185–210.

Rosenthal, D. E. 1990. An Order n Formulation for Robotic Systems. *J. Astronautical Sciences*, vol. 38, no. 4, pp. 511–529.

Saha, S. K. 1997. A Decomposition of the Manipulator Inertia Matrix. *IEEE Trans. Robotics & Automation*, vol. 13, no. 2, pp. 301–304.

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

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

Siciliano, B., and Khatib, O. (eds) 2008. *Springer Handbook of Robotics*. Berlin: Springer.

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

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

Stewart, D. E. 2000. Rigid-Body Dynamics with Friction and Impact. *SIAM Review*, vol. 42, no. 1, pp. 3–39.

Uicker, J. J. 1965. *On the Dynamic Analysis of Spatial Linkages Using 4 by 4 Matrices.* PhD thesis, Northwestern University.

Uicker, J. J. 1967. Dynamic Force Analysis of Spatial Linkages. *Trans. ASME J. Applied Mechanics*, vol. 34, pp. 418–424.

Umetani, Y., and Yoshida, K. 1989. Resolved Motion Rate Control of Space Manipulators with Generalized Jacobian Matrix. *IEEE Trans. Robotics & Automation*, vol. 5, no. 3, pp. 303–314.

Vafa, Z., and Dubowsky, S. 1990a. The Kinematics and Dynamics of Space Manipulators: The Virtual Manipulator Approach. *Int. J. Robotics Research*, vol. 9, no. 4, pp. 3–21.

Vafa, Z., and Dubowsky, S. 1990b. On the Dynamics of Space Manipulators Using the Virtual Manipulator, with applications to Path Planning. *J. Astronautical Sciences*, vol. 38, no. 4, pp. 441–472.

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

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

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

Wittenburg, J., and Wolz, U. 1985. The Program Mesa Verde for Robot Dynamics Simulations. In *Robotics Research: The Third International Symposium*, O. D. Faugeras and G. Giralt (eds.), Cambridge, MA: The MIT Press, pp. 197–204.

Woo, L. S., and Freudenstein, F. 1970. Application of Line Geometry to Theoretical Kinematics and the Kinematic Analysis of Mechanical Systems. *J. Mechanisms*, vol. 5, no. 3, pp. 417–460.

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

Symbols

The following list shows the basic associations between symbols and their meanings in this book. The list is not comprehensive. Subscripts and superscripts refine the basic meanings. For example, v_J = velocity across a joint; a_g = gravitational acceleration; I_i = inertia of body i; and so on. Leading superscripts denote coordinate systems. Many symbols have more than one meaning, and a few meanings have more than one symbol.

$\times^*$, $\boldsymbol{X}^*$ dual (i.e., force-vector version of) cross-product operator ($\times$) and coordinate transform ($\boldsymbol{X}$)

$\boldsymbol{\alpha}$, $\dot{\boldsymbol{\alpha}}$ vectors of velocity and acceleration variables, especially as an alternative to $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$

$\boldsymbol{\gamma}$ explicit motion constraint function ($\boldsymbol{q} = \boldsymbol{\gamma}(\boldsymbol{y})$)

$\boldsymbol{\delta}$ loop-closure position error; vector with one nonzero element

$\boldsymbol{\zeta}$, $\dot{\boldsymbol{\zeta}}$ vectors of contact-separation velocity and acceleration variables

$\boldsymbol{\iota}$ spatial impulse vector

$\lambda(i)$, $\mu(i)$, $\kappa(i)$, $\nu(i)$ parent of (λ), set of children of (μ), set of joints that support (κ), and set of bodies in subtree starting at (ν) body i (§4.1.4)

$\boldsymbol{\lambda}$ vector of constraint force variables; vector of contact normal forces

$\boldsymbol{\sigma}$ spatial bias velocity in (e.g.) rheonomic constraint due to explicit dependence on time

$\boldsymbol{\tau}$ vector of joint-space or generalized force variables; active force variables for a joint ($\boldsymbol{\tau} = \boldsymbol{S}^\mathrm{T}\boldsymbol{f}_\mathrm{J}$)

$\boldsymbol{\phi}$ implicit motion constraint function ($\boldsymbol{\phi}(\boldsymbol{q}) = \boldsymbol{0}$)

$\boldsymbol{\Phi}$ spatial inverse inertia

$\boldsymbol{\omega}$ 3D angular velocity vector

$\boldsymbol{a}$ spatial acceleration vector

$\boldsymbol{b}$ spatial bias acceleration (as in $\boldsymbol{a} = \boldsymbol{\Phi}\boldsymbol{f} + \boldsymbol{b}$)

$\boldsymbol{c}$ velocity-dependent spatial acceleration term; 3D vector locating centre of mass

$\boldsymbol{C}$ joint-space or generalized bias force (as in $\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C}$)

$\boldsymbol{d}$, $\boldsymbol{e}$ (Plücker) basis vectors on motion ($\boldsymbol{d}$) and force ($\boldsymbol{e}$) spaces

E 3D rotation matrix

f 3D linear force; spatial force (sometimes $\hat{f}$); general element of F^6

G, g coefficients of explicit motion constraint equation ($\dot{q} = G\dot{y}$, $\ddot{q} = G\ddot{y}+g$ where $G = \partial\gamma/\partial y$ (if γ defined), $g = \dot{G}\dot{y}$)

h spatial momentum vector; first moment of mass ($h = mc$)

H joint-space or generalized inertia

$\bar{I}$ 3D rotational inertia

I spatial inertia (I^A, I^a: articulated-body, I^c: composite-rigid-body)

J, J_{L} body (J) and loop (J_{L}) Jacobian: $v = J\dot{q}$ and $v_{\mathrm{J}} = J_{\mathrm{L}}\dot{q}$, where v and v_{J} are body and loop-joint velocities, respectively

K, k coefficients of implicit motion constraint equation ($K\dot{q} = 0$, $K\ddot{q} = k$; $K = \partial\phi/\partial q$, $k = -\dot{K}\dot{q}$)

m general spatial motion vector (any element of M^6)

$\mathsf{M}^n, \mathsf{F}^n$ vector spaces of generalized ($n=n$), spatial ($n=6$) and planar ($n=3$) motion (M) and force (F) vectors

N, n number of things (N) and variables (n); e.g., N_B, N_J, N_L, n: number of bodies, joints, kinematic loops and degrees of motion freedom

n moment or couple; spatial contact normal

N collection of contact normals: $N = [n_1 \; n_2 \; \cdots]$

O, P points in space; origins of coordinate frames; names of coordinate systems having those origins

$p(i), s(i)$ predecessor (p) and successor (s) bodies of joint i

p spatial bias force (as in $f = Ia + p$ or $f = I^A a + p^A$)

P system-wide mapping between vectors pertaining to bodies and vectors pertaining to joints ($6N_B \times 6N_J$ matrix)

$q, \dot{q}, \ddot{q}$ vectors of joint-space or generalized position, velocity and acceleration variables; such variables for a single joint

r 3D position vector of a point (e.g. an origin or a point in a body)

S, T general vector subspaces; motion freedom (S) and constraint and active force (T and T_a) subspaces

S, T matrices representing S and T; (T only) collection of contact normals: $T = [t_1 \; t_2 \; \cdots]$

t joint-space or generalized contact normal

u general vector; generalized force corresponding to $\dot{y}$; actuated subset of τ; generalized impulse

v general vector; 3D linear velocity; spatial velocity (sometimes $\hat{v}$)

X coordinate transform matrix (^jX_i: transform from i to j coordinates; X_{J}: joint transform; X_{T}, X_{P} and X_{S}: see §4.2)

$y, \dot{y}, \ddot{y}$ independent variables in explicit constraint equations; independent subset of variables q, $\dot{q}$ and $\ddot{q}$ after application of constraint

Index