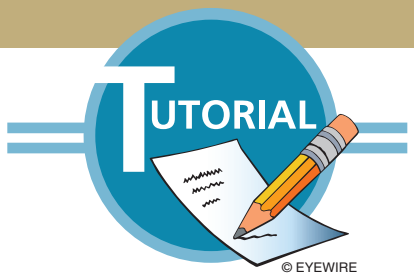


A Beginner's Guide to 6-D Vectors (Part 1)

What They Are, How They Work, and How to Use Them

BY ROY FEATHERSTONE

A rigid body has six degrees of motion freedom, so why not use six-dimensional (6-D) vectors to describe its motions and the forces acting upon it? In fact, some roboticists already do this, and the practice is becoming more common. The purpose of this tutorial is to present a beginner's guide to 6-D vectors in sufficient detail that a reader can begin to use them as a practical problem-solving tool right away. This tutorial covers the basics, and Part 2 will cover the application of 6-D vectors to a variety of robot kinematics and dynamics calculations.

6-D vectors come in various forms. The particular kind presented here is called *spatial vectors*. They are the tool that the author has been using for nearly 30 years to invent dynamics algorithms and write dynamics calculation software. Other kinds of 6-D vector include screws, motors, and Lie algebras. More will be said about them at the end of this tutorial. The differences between the various kinds of 6-D vector are relatively small. The more you understand any one of them, the easier it gets to understand the others.

The obvious advantage of 6-D vectors is that they cut the volume of algebra. Instead of having to define two three-dimensional (3-D) vectors to describe a force, another two to describe an acceleration, and writing two equations of motion for each body, a 6-D vector notation lets you pair up corresponding 3-D vectors and equations. The immediate result is a tidier, more compact notation involving fewer quantities and fewer equations. However, anyone who thinks that 6-D vectors are only a convenient notation for organizing 3-D vectors is missing half the point. 6-D vectors are tools for thought. They have their own physical meanings and mathematical properties. They let you solve a problem more directly, and at a higher level of abstraction, by letting you think in 6-D, which is easier than it sounds.

Using spatial vectors (and other kinds of 6-D vector) lets you formulate a problem more succinctly, solve it more quickly and in fewer steps, present the solution more clearly to others, implement it in fewer lines of code, and debug the software more easily. Furthermore, there is no loss of efficiency: spatial-vector software can be just as efficient as 3-D-vector software, despite the higher level of abstraction.

The rest of this tutorial is chiefly concerned with explaining what spatial vectors are and how to use them. It highlights the differences between solving a rigid-body problem using 3-D vectors and solving the same problem using spatial vectors, so that the reader can get an idea of what it means to think in 6-D.

A Note on Notation

When using spatial vectors, it is convenient to employ symbols like $\boldsymbol{f}$, $\boldsymbol{v}$, and $\boldsymbol{a}$ (or $\dot{\boldsymbol{v}}$) to denote quantities like force, velocity, and acceleration. However, these same symbols are equally useful for 3-D vectors. Thus, whenever spatial and 3-D vectors are discussed together, there is a possibility of name clashes. To resolve these clashes, we shall use the following rule: in any context where a spatial symbol needs to be distinguished from a 3-D symbol, the spatial symbol is given a hat (e.g., $\hat{\boldsymbol{f}}$ and $\hat{\boldsymbol{v}}$). These hats are dropped when they are no longer needed. An

Digital Object Identifier 10.1109/MRA.2010.937853

1070-9932/10/$26.00©2010 IEEE

Solving a Two-Body Dynamics Problem Using 3-D Vectors

We are given a rigid-body system consisting of two bodies, B_1 and B_2, connected by a revolute joint [S1] The bodies have masses of m_1 and m_2, centers of mass located at the points C_1 and C_2, and rotational inertias of I_1 and I_2 about their respective centers of mass. Both bodies are initially at rest. The joint's axis of rotation passes through the point P in the direction given by s. A system of forces acts on B_1, causing both bodies to accelerate. This system is equivalent to a single force f acting on a line passing through C_1 together with couple n. These forces impart an angular acceleration of $\dot{\omega}_1$ to B_1 and a linear acceleration of a_1 to its center of mass. The problem is to express a_1 and $\dot{\omega}_1$ in terms of f and n (Figure S1).

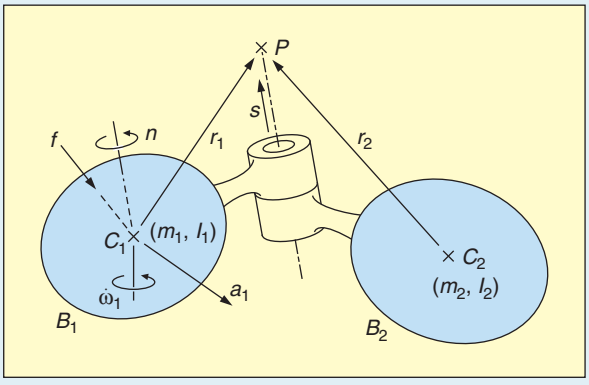


Figure S1. Problem diagram using 3-D vectors.

Solution

Let f_1, n_1, f_2, and n_2 be the net forces and couples acting on B_1 and B_2, respectively, where the lines of action of f_1 and f_2 pass through C_1 and C_2, respectively; let $\dot{\omega}_2$ and a_2 be the angular acceleration of B_2 and the linear acceleration of its center of mass; let ${}^P a_1$, ${}^P\dot{\omega}_1$, ${}^P a_2$, and ${}^P\dot{\omega}_2$ be the linear and angular accelerations of B_1 and B_2 expressed at P; and let ${}^P f_2$, ${}^P n_2$, ${}^1 f_2$, and ${}^1 n_2$ be the net force and couple acting on B_2 expressed at P and C_1, respectively. As the system of applied forces acts only on B_1, the net force and couple acting on B_2 are also the net force and couple transmitted through the joint. Let us also define $r_1 = \overrightarrow{C_1 P}$ and $r_2 = \overrightarrow{C_2 P}$, and let α be the joint acceleration variable.

The equations of motion of the two bodies, expressed at their centers of mass, are

$$f_1 = m_1 a_1, \tag{S1}$$
$$n_1 = I_1 \dot{\omega}_1, \tag{S2}$$
$$f_2 = m_2 a_2, \tag{S3}$$

and

$$n_2 = I_2 \dot{\omega}_2. \tag{S4}$$

There are no velocity terms because the bodies are at rest. The rules for transferring forces and accelerations (of bodies at rest) from one point to another provide us with the following relationships between quantities referred to C_1, C_2, and P:

$${}^P a_1 = a_1 - r_1 \times \dot{\omega}_1, \tag{S5}$$
$${}^P a_2 = a_2 - r_2 \times \dot{\omega}_2, \tag{S6}$$
$${}^P\dot{\omega}_1 = \dot{\omega}_1, \tag{S7}$$
$${}^P\dot{\omega}_2 = \dot{\omega}_2, \tag{S8}$$
$${}^1 f_2 = {}^P f_2 = f_2, \tag{S9}$$
$${}^P n_2 = n_2 - r_2 \times f_2, \tag{S10}$$

and

$${}^1 n_2 = n_2 + (r_1 - r_2) \times f_2. \tag{S11}$$

If B_1 exerts ${}^1 f_2$ and ${}^1 n_2$ on B_2 then B_2 exerts $-{}^1 f_2$ and $-{}^1 n_2$ on B_1 (Newton's third law expressed at C_1); so, the net force and couple acting on B_1 are

$$f_1 = f - {}^1 f_2,$$
$$n_1 = n - {}^1 n_2,$$

from which we get [via (S9) and (S11)]

$$f = f_1 + f_2, \tag{S12}$$

and

$$n = n_1 + n_2 + (r_1 - r_2) \times f_2. \tag{S13}$$

The joint allows B_2 one degree of motion freedom relative to B_1 and imposes one constraint on the couple transmitted from B_1 to B_2. Expressed at P, the constraint equations are

$${}^P a_2 = {}^P a_1, \tag{S14}$$
$${}^P\dot{\omega}_2 = {}^P\dot{\omega}_1 + s\,\alpha, \tag{S15}$$

and

$$s^{\mathrm{T}}\, {}^P n_2 = 0, \tag{S16}$$

where α is the unknown joint acceleration variable. There is no constraint on ${}^P f_2$: (S16) is sufficient to ensure that the force and couple transmitted by the joint perform no work in the direction of relative motion permitted by the joint.

We are now ready to solve the problem. Let us start by calculating a_2 and $\dot{\omega}_2$ in terms of a_1, $\dot{\omega}_1$ and α. From (S8), (S15), and (S7), we have

$$\begin{aligned}\dot{\omega}_2 &= {}^P\dot{\omega}_2 \\ &= {}^P\dot{\omega}_1 + s\,\alpha \\ &= \dot{\omega}_1 + s\,\alpha,\end{aligned} \tag{S17}$$

and from (S6), (S14), (S17), and (S5) we have

$$\begin{aligned}a_2 &= {}^P a_2 + r_2 \times \dot{\omega}_2 \\ &= {}^P a_1 + r_2 \times (\dot{\omega}_1 + s\,\alpha) \\ &= a_1 - r_1 \times \dot{\omega}_1 + r_2 \times (\dot{\omega}_1 + s\,\alpha) \\ &= a_1 + (r_2 - r_1) \times \dot{\omega}_1 + r_2 \times s\,\alpha.\end{aligned} \tag{S18}$$

Now let us calculate α. From (S16), (S10), (S3), (S4), (S17), and (S18), we get

$$
\begin{aligned}
0 &= s^{\mathrm{T}\,p}n_2 \\
&= s^{\mathrm{T}}(n_2 - r_2 \times f_2) \\
&= s^{\mathrm{T}}(I_2\,\dot{\omega}_2 - m_2\,r_2 \times a_2) \\
&= s^{\mathrm{T}}(I_2\,(\dot{\omega}_1 + s\,\dot{\alpha}) - m_2\,r_2 \times \\
&\quad (a_1 + (r_2 - r_1)\times\dot{\omega}_1 + r_2 \times s\,\alpha))\,.
\end{aligned}
$$

Collecting terms in α gives

$$
\begin{aligned}
&s^{\mathrm{T}}(I_2\,s - m_2\,r_2 \times (r_2 \times s))\,\alpha \\
&\qquad + s^{\mathrm{T}}(I_2\,\dot{\omega}_1 - m_2\,r_2 \times (a_1 + (r_2 - r_1)\times\dot{\omega}_1)) = 0\,,
\end{aligned}
$$

hence

$$
\alpha = -\frac{s^{\mathrm{T}}(I_2\,\dot{\omega}_1 - m_2\,r_2 \times (a_1 + (r_2 - r_1)\times\dot{\omega}_1))}{s^{\mathrm{T}}(I_2\,s - m_2\,r_2 \times (r_2 \times s))}\,. \tag{S19}
$$

This equation is only valid if the denominator is not equal to zero, so we must investigate the necessary conditions for it to be nonzero. This problem can be solved using the following trick. For any two vectors u and v, the cross product $u \times v$ can be expressed in the form $u \times v = \tilde{u}\,v$, where $\tilde{u}$ is the skew-symmetric matrix:

$$
\tilde{u} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}.
$$

Using this trick, we can express the denominator in the form $s^{\mathrm{T}}J\,s$, where

$$
\begin{aligned}
J &= I_2 - m_2\,\tilde{r}_2\,\tilde{r}_2 \\
&= I_2 + m_2\,\tilde{r}_2^{\mathrm{T}}\,\tilde{r}_2\,.
\end{aligned} \tag{S20}
$$

J is therefore the sum of an SPD matrix and an SPSD matrix, hence itself also SPD, so the denominator of (S19) is guaranteed to be strictly greater than zero. Substituting (S20) into (S19) gives us the following simplified expression for α:

$$
\alpha = -\frac{s^{\mathrm{T}}(J\,\dot{\omega}_1 - m_2\,r_2 \times (a_1 - r_1 \times \dot{\omega}_1))}{s^{\mathrm{T}}J\,s}. \tag{S21}
$$

The next step is to express f and n in terms of a_1, $\dot{\omega}_1$, and α, and then to eliminate α using (S21). Let us start with f. From (S12), (S1), (S3), and (S18), we get

$$
\begin{aligned}
f &= f_1 + f_2 \\
&= m_1\,a_1 + m_2\,a_2 \\
&= m_1\,a_1 + m_2(a_1 + (r_2 - r_1)\times\dot{\omega}_1 + r_2 \times s\,\alpha) \\
&= (m_1 + m_2)a_1 + m_2(r_2 - r_1)\times\dot{\omega}_1 + m_2\,r_2 \times s\,\alpha\,.
\end{aligned}
$$

Eliminating α using (S21) gives

$$
\begin{aligned}
f = (m_1 + m_2)a_1 + m_2(r_2 - r_1)\times\dot{\omega}_1 \\
- m_2\,\frac{r_2 \times s\,s^{\mathrm{T}}(J\,\dot{\omega}_1 - m_2\,r_2 \times (a_1 - r_1 \times \dot{\omega}_1))}{s^{\mathrm{T}}J\,s};
\end{aligned}
$$

and collecting terms in a_1 and $\dot{\omega}_1$ gives

$$
\begin{aligned}
f = &\left(m_1 + m_2 + m_2^2\,\frac{\tilde{r}_2\,s\,s^{\mathrm{T}}\,\tilde{r}_2}{s^{\mathrm{T}}J\,s}\right)a_1 \\
&+ \left(m_2(\tilde{r}_2 - \tilde{r}_1) - m_2\,\frac{\tilde{r}_2\,s\,s^{\mathrm{T}}(J + m_2\,\tilde{r}_2\,\tilde{r}_1)}{s^{\mathrm{T}}J\,s}\right)\dot{\omega}_1\,.
\end{aligned} \tag{S22}
$$

Repeating the procedure for n, (S13), (S2), (S3), (S4), (S17), and (S18) give

$$
\begin{aligned}
n &= n_1 + n_2 + (r_1 - r_2)\times f_2 \\
&= I_1\,\dot{\omega}_1 + I_2\,\dot{\omega}_2 + m_2(r_1 - r_2)\times a_2 \\
&= I_1\,\dot{\omega}_1 + I_2(\dot{\omega}_1 + s\,\dot{\alpha}) + m_2(r_1 - r_2)\times \\
&\quad (a_1 + (r_2 - r_1)\times\dot{\omega}_1 + r_2 \times s\,\alpha) \\
&= (I_1 + I_2 - m_2(\tilde{r}_1 - \tilde{r}_2)^2)\dot{\omega}_1 \\
&\quad + m_2(\tilde{r}_1 - \tilde{r}_2)a_1 + K s\,\alpha\,,
\end{aligned} \tag{S23}
$$

where

$$
\begin{aligned}
K &= I_2 + m_2(\tilde{r}_1 - \tilde{r}_2)\tilde{r}_2 \\
&= J + m_2\,\tilde{r}_1\,\tilde{r}_2\,.
\end{aligned} \tag{S24}
$$

Note that (S21) can now be simplified to

$$
\alpha = -\frac{s^{\mathrm{T}}(K^{\mathrm{T}}\dot{\omega}_1 - m_2\,\tilde{r}_2\,a_1)}{s^{\mathrm{T}}J\,s}\,. \tag{S25}
$$

Eliminating α from (S23) using (S25) gives

$$
\begin{aligned}
n = &(I_1 + I_2 - m_2(\tilde{r}_1 - \tilde{r}_2)^2)\dot{\omega}_1 \\
&+ m_2(\tilde{r}_1 - \tilde{r}_2)a_1 - \frac{K s\,s^{\mathrm{T}}(K^{\mathrm{T}}\dot{\omega}_1 - m_2\,\tilde{r}_2\,a_1)}{s^{\mathrm{T}}J\,s},
\end{aligned}
$$

and collecting terms in $\dot{\omega}_1$ and a_1 gives

$$
\begin{aligned}
n = &\left(I_1 + I_2 - m_2(\tilde{r}_1 - \tilde{r}_2)^2 - \frac{K s\,s^{\mathrm{T}}K^{\mathrm{T}}}{s^{\mathrm{T}}J\,s}\right)\dot{\omega}_1 \\
&+ \left(m_2(\tilde{r}_1 - \tilde{r}_2) + m_2\,\frac{K s\,s^{\mathrm{T}}\tilde{r}_2}{s^{\mathrm{T}}J\,s}\right)a_1\,.
\end{aligned} \tag{S26}
$$

The final step is to combine (S22) and (S26) into a single equation:

$$
\begin{bmatrix} f \\ n \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} a_1 \\ \dot{\omega}_1 \end{bmatrix}, \tag{S27}
$$

where

$$
A = (m_1 + m_2)\,1_{3\times3} + m_2^2\,\frac{\tilde{r}_2\,s\,s^{\mathrm{T}}\,\tilde{r}_2}{s^{\mathrm{T}}J\,s}, \tag{S28}
$$

$$
B = m_2(\tilde{r}_2 - \tilde{r}_1) - m_2\,\frac{\tilde{r}_2\,s\,s^{\mathrm{T}}K^{\mathrm{T}}}{s^{\mathrm{T}}J\,s}, \tag{S29}
$$

$$
C = m_2(\tilde{r}_1 - \tilde{r}_2) + m_2\,\frac{K s\,s^{\mathrm{T}}\tilde{r}_2}{s^{\mathrm{T}}J\,s}, \tag{S30}
$$

and

$$
D = I_1 + I_2 - m_2(\tilde{r}_1 - \tilde{r}_2)^2 - \frac{K s\,s^{\mathrm{T}}K^{\mathrm{T}}}{s^{\mathrm{T}}J\,s}\,. \tag{S31}
$$

$1_{3\times3}$ is an identity matrix. The solution to the original problem is then

$$
\begin{bmatrix} a_1 \\ \dot{\omega}_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1}\begin{bmatrix} f \\ n \end{bmatrix}. \tag{S32}
$$

COURTESY OF R. FEATHERSTONE, 2010.

Reference

[S1] R. Featherstone. (2010). Spatial vector algebra [Online]. Available: http://users.cecs.anu.edu.au/∼roy/spatial/

alternative strategy, which works just as well, is to distinguish the 3-D symbols from the spatial ones by marking them with arrows (e.g., $\vec{f}$ and $\vec{v}$). Rules like this provide a degree of flexibility to the user, and are preferable to simpler rules like "all spatial symbols have hats," which just lead to an unnecessary sea of hats—a nuisance to read and write.

It is sometimes useful to distinguish between a coordinate vector and the quantity it represents. This will be done by underlining the coordinate vector. Thus, you will occasionally see a symbol like $\underline{v}$ or $\underline{\hat{v}}$ used to denote the coordinate vector representing v or $\hat{v}$. This notational device is used only where needed.

A Worked Example

The main message of this tutorial is that spatial vectors are not merely a convenient way of pairing up 3-D vectors but are a problem-solving tool in their own right. Spatial vectors have their own physical interpretations, their own equations and formulae, and their own rules of use; and the best way to use them is to think directly in 6-D.

Perhaps the best way to explain is by means of a worked example, comparing the 3-D and 6-D approach to solving a rigid-body problem. "Solving a Two-Body Dynamics Problem Using 3-D Vectors" presents a detailed worked example of how to solve a simple two-body dynamics problem using

Solving a Two-Body Dynamics Problem Using Spatial Vectors

We are given a rigid-body system consisting of two bodies, B_1 and B_2, connected by a revolute joint [S2]. The bodies have inertias of I_1 and I_2, respectively, and they are initially at rest. The joint's rotation axis is s. A force f is applied to B_1, causing both bodies to accelerate. The problem is to calculate the acceleration of B_1 as a function of f (Figure S2).

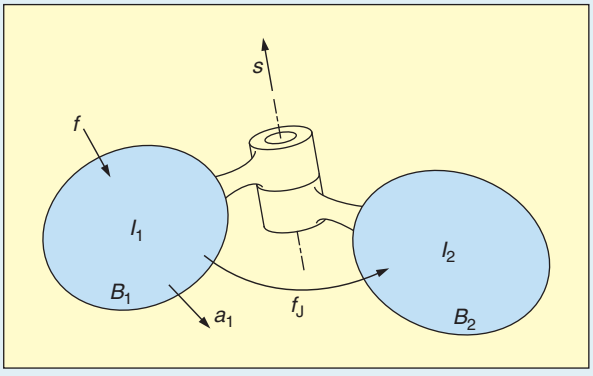


Figure S2. Problem diagram using spatial vectors.

Solution

Let a_1 and a_2 be the accelerations of the two bodies, and let f_J be the force transmitted from B_1 to B_2 through the joint. The net forces acting on the two bodies are therefore $f - f_J$ and f_J, respectively, and their equations of motion are

$$f - f_J = I_1 a_1 \tag{S33}$$

and

$$f_J = I_2 a_2. \tag{S34}$$

There are no velocity terms because the bodies are at rest. The joint permits B_2 to accelerate relative to B_1 about the axis specified by s; so, a_2 can be expressed in the form

$$a_2 = a_1 + s\alpha, \tag{S35}$$

where α is the joint acceleration variable. Again, there are no velocity terms because the bodies are at rest. This motion constraint is implemented by f_J, which is the joint constraint force, so f_J must satisfy

$$s^T f_J = 0, \tag{S36}$$

i.e., the constraint force does no work in the direction of motion allowed by the joint.

Given (S33)–(S36), the problem is solved as follows. First, substitute (S35) into (S34), giving

$$f_J = I_2 (a_1 + s\alpha). \tag{S37}$$

Substituting (S37) into (S36) gives

$$s^T I_2 (a_1 + s\alpha) = 0,$$

from which we get the following expression for α:

$$\alpha = -\frac{s^T I_2 a_1}{s^T I_2 s}. \tag{S38}$$

Substituting (S38) back into (S37) gives

$$f_J = I_2 \left(a_1 - \frac{s s^T I_2}{s^T I_2 s} a_1 \right),$$

and substituting this equation back into (S33) gives

$$\begin{aligned} f &= I_1 a_1 + I_2 a_1 - \frac{I_2 s s^T I_2}{s^T I_2 s} a_1 \\ &= \left(I_1 + I_2 - \frac{I_2 s s^T I_2}{s^T I_2 s} \right) a_1 . \end{aligned}$$

The expression in brackets is nonsingular and may therefore be inverted to express a_1 in terms of f:

$$a_1 = \left(I_1 + I_2 - \frac{I_2 s s^T I_2}{s^T I_2 s} \right)^{-1} f. \tag{S39}$$

Reference

[S2] R. Featherstone. (2010). Spatial vector algebra [Online]. Available: http://users.cecs.anu.edu.au/~roy/spatial/

3-D vectors, and "Solving a Two-Body Dynamics Problem Using Spatial Vectors" shows how to solve the same problem using spatial vectors. We shall refer to them as the 3-D example and the spatial example, respectively. The 3-D example employs the methods of 3-D vectorial dynamics, and the spatial example employs the methods of spatial vector algebra. Even the briefest glance reveals that the spatial example is simpler in every aspect: the problem statement is shorter, the diagram is simpler, and the solution is much shorter. Let us now examine these examples in more detail.

Starting with the 3-D example, eight quantities are needed to describe the rigid-body system: the mass, center of mass, and rotational inertia of each body, plus the quantities P and $\boldsymbol{s}$ to define the joint axis. A further two quantities are needed to describe the forces acting on B_1, and a further two to describe its resulting acceleration. Furthermore, it is not enough merely to state that $\boldsymbol{f}$, $\boldsymbol{n}$, $\boldsymbol{a}_1$, and $\dot{\boldsymbol{\omega}}_1$ describe the forces and accelerations—a complete description requires that we also identify the line of action of $\boldsymbol{f}$ and the particular point in B_1 to which $\boldsymbol{a}_1$ refers. Having introduced the three points C_1, C_2, and P as a necessary part of describing the problem, it becomes desirable to show these points on the diagram, as they will play a major role in the solution process.

In the terminology of the 3-D-vector approach, $\boldsymbol{f}$, $\boldsymbol{n}$, $\boldsymbol{a}_1$, and $\dot{\boldsymbol{\omega}}_1$ are said to be referred to (or expressed at) C_1, meaning that C_1 serves as the reference point for these quantities. Equations (S1) and (S2) are likewise referred to (expressed at) C_1. This need to define various points in space, and to refer various vectors and equations to these points, is a characteristic feature of the 3-D-vector approach to solving a rigid-body problem. It accounts for a large part of the algebraic complexity, and it forces the analyst to think explicitly in terms of which point will be used to express which equation and which quantities will have to be transferred from one reference point to another. A poor choice of reference points can render a complicated solution procedure even more complicated.

In the 3-D example, we can see that the equations of motion of each body have been expressed at their respective centers of mass, and that the equations of constraint (S14)–(S16) have been expressed at P. These are good choices, but they require us to define an extra eight quantities (${}^{P}\boldsymbol{a}_1$ to ${}^{1}\boldsymbol{n}_2$) and an extra seven equations (S5)–(S11) to manage all the necessary transfers of vectors from one reference point to another.

At the highest level, the solution strategy is this: express the acceleration of B_2 in terms of $\boldsymbol{a}_1$, $\dot{\boldsymbol{\omega}}_1$, and α, and then use the force-constraint equation (S16) to obtain an expression for α in terms of $\boldsymbol{a}_1$ and $\dot{\boldsymbol{\omega}}_1$. At this point, every force and acceleration in the system can be expressed in terms of $\boldsymbol{a}_1$ and $\dot{\boldsymbol{\omega}}_1$; so, the solution is obtained by expressing $\boldsymbol{f}$ and $\boldsymbol{n}$ in terms of $\boldsymbol{a}_1$ and $\dot{\boldsymbol{\omega}}_1$, and then inverting the equations to express the accelerations in terms of the forces.

Let us now examine the spatial-vector example. In this case, only three quantities ($\boldsymbol{I}_1$, $\boldsymbol{I}_2$, and $\boldsymbol{s}$) are required to describe the rigid-body system; only one quantity ($\boldsymbol{f}$) is required to describe the forces acting on B_1; and only one quantity ($\boldsymbol{a}_1$) is required to describe its acceleration. Furthermore, the solution procedure introduces only another three quantities ($\boldsymbol{a}_2$, $\boldsymbol{f}_J$, and α). So, the whole problem now involves only eight quantities.

Observe that there is no mention of any 3-D point anywhere in this example. The problem has been stated and solved without reference to any point in space. This absence of reference points is a characteristic feature of the spatial-vector approach (and some other 6-D formalisms) and is a key aspect of thinking in 6-D.

Referring back to the 3-D example, it is clearly possible to pair up corresponding 3-D vectors ($\boldsymbol{f}$ with $\boldsymbol{n}$, $\boldsymbol{a}_1$ with $\dot{\boldsymbol{\omega}}_1$, and so on) to make 6-D vectors, and this would result in some reduction in the volume of algebra. However, the points C_1, C_2, and P would still be an essential part of the problem statement and the solution process. Thus, the stacking of pairs of 3-D vectors is purely a notational device: the resulting vectors are 6-D, but the concepts, methods, and thought processes are all still 3-D.

Returning to the spatial-vector example, the diagram is clearly simpler, but the arrows now have different meanings. The arrow associated with $\boldsymbol{f}$, which points from empty space to B_1, indicates only that $\boldsymbol{f}$ is an external force acting on B_1. It does not convey any geometrical information (such as the line of action of a force). Likewise, the arrow associated with $\boldsymbol{f}_J$, which points from B_1 to B_2, indicates only that $\boldsymbol{f}_J$ is a force transmitted from B_1 to B_2, whereas the arrow associated with $\boldsymbol{a}_1$, which points out of B_1, indicates only that $\boldsymbol{a}_1$ is the acceleration of B_1. From the directions of the arrows (and knowledge of spatial vectors), we can immediately deduce that the net force on B_1 is $\boldsymbol{f} - \boldsymbol{f}_J$ and the net force on B_2 is $\boldsymbol{f}_J$. The arrow associated with $\boldsymbol{s}$, which is aligned with the joint's rotation axis, is the only one with any geometrical significance.

The reason why there are no 3-D points in the spatial-vector example and why most of the arrows have no geometrical significance is because all the necessary positional information is intrinsic to the relevant spatial quantities. The inertias $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ implicitly locate the centers of mass of the two bodies; the line of action of $\boldsymbol{f}$ (if it has one) can be deduced from its value; and $\boldsymbol{s}$ defines both the direction and the location of an axis of rotation in 3-D space. Acceleration is a little more complicated and will be discussed in a later section. Nevertheless, $\boldsymbol{a}_1$ does provide a complete description of a body's acceleration and does not need to be referred to any point.

The high-level solution strategy in the spatial-vector example is the same as that in the 3-D example: express $\boldsymbol{a}_2$ in terms of $\boldsymbol{a}_1$ and α; then, substitute into the force-constraint equation (S36) to obtain an expression for α in terms of $\boldsymbol{a}_1$; then, express $\boldsymbol{f}$ in terms of $\boldsymbol{a}_1$ and invert to express $\boldsymbol{a}_1$ in terms of $\boldsymbol{f}$. Using spatial vectors, the analyst is able to follow this high-level strategy directly, without having to think about the messy details associated with the 3-D-vector approach. Observe how the expression for α is obtained almost immediately, after just two simple substitutions, and the desired expression for $\boldsymbol{f}$ is obtained after just three more simple substitutions.

Another benefit of spatial vectors, which is not evident from this example, is that it is quite easy to prove that the expression in parentheses in (S39) is a symmetric, positive-definite matrix, and therefore invertible. The same is also true of the 6×6 matrix in (S27) of the 3-D example, but the proof (using 3-D vectors) is relatively complicated. Yet another advantage of spatial vectors is that a person who is new to the

example problem is likely to get the correct answer quickly using spatial vectors but is likely to get lost and take wrong turns while trying to solve it using 3-D vectors.

Some Formalities

To understand spatial vectors, it helps to review a few basic facts about vectors. First, there are many different types of vector, each having different mathematical properties. In fact, there are only two operations that are defined on all vectors: the addition of two vectors and the multiplication of a vector by a scalar (i.e., a real number). There are several more operators that are defined on particular types of vector and give them special properties. One important example is the Euclidean inner product, which is defined only on Euclidean vectors, and gives them the special properties of magnitude and direction.

Two types of vector are of special interest: Euclidean and coordinate vectors. Euclidean vectors have the special properties of magnitude and direction, and they are the elements of a Euclidean vector space, which we shall denote with the symbol E^n. (The superscript indicates the dimension.) Coordinate vectors are n-tuples of real numbers, and they are elements of the vector space R^n. The special property of coordinate vectors is that they have a first coordinate, a second coordinate, and so on.

Coordinate vectors are used to represent other kinds of vector via a basis. For example, if V is a general vector space and $\mathcal{E} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\} \subset V$ is a basis on V, then any vector $\boldsymbol{v} \in V$ can be expressed in the form $\boldsymbol{v} = \sum_{i=1}^{n} v_i \boldsymbol{e}_i$, where v_i are the coordinates of $\boldsymbol{v}$ in $\mathcal{E}$. If we assemble them into a coordinate vector $\underline{\boldsymbol{v}} = [v_1\ v_2 \cdots v_n]^{\mathrm{T}}$, then we can say that $\underline{\boldsymbol{v}} \in \mathsf{R}^n$ represents $\boldsymbol{v} \in V$ in the basis $\mathcal{E} \subset V$.

Some bases are more useful than others. For Euclidean vectors, the most useful is an orthonormal basis, which gives rise to a Cartesian coordinate system. The special property of Cartesian coordinates is this: if $\underline{\boldsymbol{v}}_1$ and $\underline{\boldsymbol{v}}_2$ are coordinate vectors representing the Euclidean vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in a Cartesian coordinate system, then $\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 = \underline{\boldsymbol{v}}_1^{\mathrm{T}} \underline{\boldsymbol{v}}_2$.

Spatial vectors are not Euclidean. They are the elements of two closely related vector spaces called M^6 and F^6. The former contains motion vectors, which describe the motions of rigid bodies, and the latter contains force vectors, which describe forces acting on rigid bodies. Formally, F^6 is the dual vector space of M^6 and vice versa. This notation can be extended to encompass other kinds of motion and force vector. For example, a generalized force vector could be described as an element of F^n, and a vector describing the motion of a set of N rigid bodies could be said to be an element of M^{6N}.

There is no inner product defined on spatial vectors. Instead, there is a scalar product that takes one argument from each space. If $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$, then the expressions $\boldsymbol{m} \cdot \boldsymbol{f}$ and $\boldsymbol{f} \cdot \boldsymbol{m}$ are defined (and are equal), but the expressions $\boldsymbol{m} \cdot \boldsymbol{m}$ and $\boldsymbol{f} \cdot \boldsymbol{f}$ are not. As we shall see later, if $\boldsymbol{m}$ is the velocity of a rigid body and $\boldsymbol{f}$ is the force acting on it, then $\boldsymbol{f} \cdot \boldsymbol{m}$ is the power delivered by $\boldsymbol{f}$.

A coordinate system for spatial vectors must span both M^6 and F^6. Thus, a total of 12 basis vectors are required: $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_6\} \subset \mathsf{M}^6$ and $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_6\} \subset \mathsf{F}^6$. Again, some bases are more useful than others. For spatial vectors, the most useful is a dual basis, which defines a dual coordinate system on M^6 and F^6. To qualify as a dual basis, the vectors $\boldsymbol{d}_i$ and $\boldsymbol{e}_i$ must satisfy the following condition:

$$\boldsymbol{d}_i \cdot \boldsymbol{e}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The special property of a dual coordinate system is this: if $\underline{\boldsymbol{m}}$ and $\underline{\boldsymbol{f}}$ are coordinate vectors representing $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$ in a dual coordinate system, then $\boldsymbol{m} \cdot \boldsymbol{f} = \underline{\boldsymbol{m}}^{\mathrm{T}} \underline{\boldsymbol{f}}$. Thus, dual coordinates are the spatial-vector equivalent of a Cartesian coordinate system on a Euclidean vector space. The Plücker coordinates that we shall meet later on are a special type of dual coordinate system having extra properties that make them especially convenient to use.

A general property of dual coordinate systems is that motion and force vectors obey different coordinate-transformation rules. If $\boldsymbol{X}$ is a coordinate transform for motion vectors, then the corresponding transform for force vectors is called $\boldsymbol{X}^*$, and the two are related by

$$\boldsymbol{X}^* = (\boldsymbol{X}^{-1})^{\mathrm{T}} = \boldsymbol{X}^{-\mathrm{T}}. \tag{2}$$

This relationship ensures that the scalar product is invariant with respect to any coordinate transformation, as can be seen from

$$(\boldsymbol{X}^* \boldsymbol{f})^{\mathrm{T}} (\boldsymbol{X} \boldsymbol{m}) = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{X}^{-1} \boldsymbol{X} \boldsymbol{m} = \boldsymbol{f}^{\mathrm{T}} \boldsymbol{m}. \tag{3}$$

What Is a Spatial Vector?

Let P be a particle in 3-D space. If we say that P has a velocity of $\boldsymbol{v}$, then we mean that $\boldsymbol{v}$ is a Euclidean vector ($\boldsymbol{v} \in \mathsf{E}^3$) whose magnitude and direction match the speed and direction of travel of P. Now suppose B is a rigid body. If we are told that B has a velocity of $\hat{\boldsymbol{v}}$, then we can immediately infer that $\hat{\boldsymbol{v}} \in \mathsf{M}^6$; but how exactly does $\hat{\boldsymbol{v}}$ describe the motion of B, and how is it possible for $\hat{\boldsymbol{v}}$ to describe the motion without using a reference point? An intuitive answer to these questions is provided by screw theory.

The most general motion of a rigid body, at any given instant, is a screwing motion along a directed line. (The body is behaving like a nut on a screw thread fixed somewhere in space.) In screw theory, such a motion is called a *twist*. A twist can be characterized by an angular magnitude, a linear magnitude, and a directed line. (The ratio of the two magnitudes is called the *pitch*.) These three quantities together define a twist, just as a magnitude and a direction together define a particle velocity. The two magnitudes describe the rate at which the body is rotating about and translating along the directed line, and the line itself describes the instantaneous screw axis of the motion. (As the name suggests, a rigid body will, in general, be screwing about two different axes at two different instants. Over a finite time interval, a body will have screwed about an infinity of axes, each one fixed in space and each one valid for a single instant.) If the linear magnitude is zero, then the motion is a pure rotation. If the angular magnitude is zero, then the motion is a pure translation, in which case the location of the line is irrelevant and only its direction matters.

We can now answer the question posed earlier. The elements of M^6 are twists and are characterized by two magnitudes and a directed line. If we say that $\hat{\boldsymbol{v}} \in \mathsf{M}^6$ is the velocity of B, then we mean that the directed line and the linear and angular magnitudes of $\hat{\boldsymbol{v}}$ match the instantaneous screw axis of the body's motion and its linear and angular rates of progression along and about that axis.

Spatial force vectors can be explained in a similar manner. The most general force acting on a rigid body consists of a screwing force acting along a directed line: a linear force acting along the line, together with a turning force (a couple) acting about the line. In screw theory, such a quantity is called a *wrench*. A wrench can be characterized by a linear magnitude, an angular magnitude, and a directed line—exactly the same as a twist. The two magnitudes describe the intensities of the linear and angular components of the wrench, and the directed line is the instantaneous screw axis. If the angular magnitude is zero, then the wrench is a pure force. If the linear magnitude is zero, then the wrench is a pure couple, in which case the location of the line is irrelevant and only its direction matters.

If the elements of F^6 are forces acting on rigid bodies, then they are wrenches, and they are characterized by two magnitudes and a directed line. If we say that a force $\hat{\boldsymbol{f}} \in \mathsf{F}^6$ is acting on body B, then we mean that the directed line and the linear and angular magnitudes of $\hat{\boldsymbol{f}}$ match the instantaneous screw axis and the linear and angular intensities of the wrench acting on body B.

If we introduce a reference point O, then it becomes possible to represent a spatial velocity $\hat{\boldsymbol{v}}$ by means of a pair of 3-D vectors $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$, and to represent a spatial force $\hat{\boldsymbol{f}}$ by means of a pair of 3-D vectors $\boldsymbol{f}$ and $\boldsymbol{n}_O$. Having explained the nature of rigid-body motion and force by means of twists and wrenches, it should now be clear that reference points are not an intrinsic necessity but merely an artifact of the 3-D-vector representation. In other words, you only need reference points if you are using 3-D vectors. Choosing a reference point is like choosing a coordinate system in which a spatial vector is to be represented by a pair of vector-valued coordinates. (This idea is explored in [6].)

Having obtained $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$, there are two ways to view the velocity they describe. According to one view, the body is rotating with an angular velocity of $\boldsymbol{\omega}$ about an axis passing through O while simultaneously translating with a linear velocity of $\boldsymbol{v}_O$. According to the other view, $\boldsymbol{\omega}$ defines the angular magnitude of the twist velocity and the direction of the instantaneous screw axis, while $\boldsymbol{v}_O$ (in combination with $\boldsymbol{\omega}$) defines the linear magnitude of the twist velocity and the location of the instantaneous screw axis relative to O. Both views are correct and useful. Similar comments apply to forces.

Using Spatial Vectors

Spatial vectors are a tool for expressing and analyzing the physical properties and behavior of rigid-body systems. The vectors describe physical quantities, and the equations describe relationships between them. So what do the rules of classical mechanics look like in spatial-vector form? Here is a short list of basic facts and formulae. It mentions some quantities that we have not yet met, but will be described in the next section.

- *Relative velocity*: If bodies B_1 and B_2 have velocities of $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, respectively, then the relative velocity of B_2 with respect to B_1 is $\boldsymbol{v}_{\text{rel}} = \boldsymbol{v}_2 - \boldsymbol{v}_1$. Obviously, this also means that $\boldsymbol{v}_2 = \boldsymbol{v}_1 + \boldsymbol{v}_{\text{rel}}$.
- *Summation of forces*: If forces $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ act on the same rigid body, then they are equivalent to a single force, $\boldsymbol{f}_{\text{tot}}$, given by $\boldsymbol{f}_{\text{tot}} = \boldsymbol{f}_1 + \boldsymbol{f}_2$.
- *Action and reaction*: If body B_1 exerts a force $\boldsymbol{f}$ on body B_2, then B_2 exerts a force $-\boldsymbol{f}$ on B_1. This is Newton's third law in spatial form.
- *Scalar product*: If a force $\boldsymbol{f}$ acts on a body having a velocity of $\boldsymbol{v}$, then the power delivered by that force is $\boldsymbol{f} \cdot \boldsymbol{v}$.
- *Scalar multiplication*: This operation affects a spatial vector's magnitudes but not its directed line. If a spatial vector $\boldsymbol{s}$ is characterized by magnitudes m_1 and m_2 and line $\boldsymbol{l}$, then $\alpha\boldsymbol{s}$ is characterized by αm_1, αm_2, and $\boldsymbol{l}$. A body having a velocity of $\alpha\boldsymbol{v}$ makes the same infinitesimal motion over a period of δt as a body with a velocity of $\boldsymbol{v}$ makes over a period of $\alpha\delta t$; and a force $\beta\boldsymbol{f}$ delivers β times as much power as a force $\boldsymbol{f}$ acting on the same body. So, $(\alpha\boldsymbol{v}) \cdot (\beta\boldsymbol{f}) = \alpha\beta(\boldsymbol{v} \cdot \boldsymbol{f})$.
- *Differentiation*: Spatial vectors are differentiated just like any other vector. The derivative of a motion vector is a motion vector, and the derivative of a force vector is a force vector. If $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$ are fixed in a body having a velocity of $\boldsymbol{v}$, then $\dot{\boldsymbol{m}} = \boldsymbol{v} \times \boldsymbol{m}$ and $\dot{\boldsymbol{f}} = \boldsymbol{v} \times^* \boldsymbol{f}$. [The operator $\times^*$ is defined in (18).]
- *Acceleration*: Spatial acceleration is the time derivative of spatial velocity ($\boldsymbol{a} = \dot{\boldsymbol{v}}$). For example, if $\boldsymbol{v}_2 = \boldsymbol{v}_1 + \boldsymbol{v}_{\text{rel}}$, then $\boldsymbol{a}_2 = \boldsymbol{a}_1 + \boldsymbol{a}_{\text{rel}}$. Spatial accelerations are elements of M^6 and therefore obey the same coordinate-transformation rule as velocities.
- *Summation of inertias*: If bodies B_1 and B_2, having inertias of $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$, respectively, are rigidly connected to form a single composite rigid body, then the inertia of the composite is $\boldsymbol{I}_{\text{tot}} = \boldsymbol{I}_1 + \boldsymbol{I}_2$.
- *Momentum*: If a rigid body has a velocity of $\boldsymbol{v}$ and an inertia of $\boldsymbol{I}$, then its momentum is $\boldsymbol{I}\boldsymbol{v}$.
- *Equation of motion*: The total force acting on a rigid body equals its rate of change of momentum. $\boldsymbol{f} = \mathrm{d}(\boldsymbol{I}\boldsymbol{v})/\mathrm{d}t = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v} \times^* \boldsymbol{I}\boldsymbol{v}$.
- *Motion constraints*: If the relative velocity of two rigid bodies is constrained to lie in a subspace $S \subseteq \mathsf{M}^6$, then the motion constraint is implemented by a constraint force lying in the subspace $T = \{\boldsymbol{f} \in \mathsf{F}^6 \mid \boldsymbol{f} \cdot \boldsymbol{v} = 0 \ \forall\, \boldsymbol{v} \in S\}$. This is a statement of D'Alembert's principle of virtual work (or Jourdain's principle of virtual power) expressed using spatial vectors.

Details

This section covers the practical details of coordinate systems, differentiation, inertia, the equation of motion, and motion constraints. By the end of this section, you should be able to understand the spatial-vector example clearly enough to be able to generalize it to the case where B_1 and B_2 have nonzero velocities.

Plücker Coordinates

Plücker coordinates (pronounced plooker) are the coordinate systems of choice for 6-D vectors. To set up a Plücker coordinate system, all that is needed is a Cartesian coordinate frame $Oxyz$ placed anywhere. The position and orientation of this frame defines a Plücker coordinate system. In fact, there is a 1:1 correspondence between the set of all positions and orientations of a Cartesian frame and the set of all possible Plücker coordinate systems.

The frame $Oxyz$ defines the following items: a point O, three mutually orthogonal directions x, y, and z, and three directed lines Ox, Oy, and Oz. (They pass through O in the x, y, and z directions.) With this data, we can define three bases as follows:

$$\mathcal{C} = \{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\} \subset \mathsf{E}^3, \tag{4}$$

$$\mathcal{D} = \{\boldsymbol{d}_{Ox}, \boldsymbol{d}_{Oy}, \boldsymbol{d}_{Oz}, \boldsymbol{d}_x, \boldsymbol{d}_y, \boldsymbol{d}_z\} \subset \mathsf{M}^6, \tag{5}$$

and

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

The elements of $\mathcal{C}$ are unit Euclidean vectors in the x, y, and z directions, and this basis defines a Cartesian coordinate system on E^3. The elements of $\mathcal{D}$ and $\mathcal{E}$ are as follows: $\boldsymbol{d}_{Ox}$, $\boldsymbol{d}_{Oy}$, and $\boldsymbol{d}_{Oz}$ are unit pure rotations about the lines Ox, Oy, and Oz, respectively; $\boldsymbol{d}_x$, $\boldsymbol{d}_y$, and $\boldsymbol{d}_z$ are unit pure translations in the x, y, and z directions, respectively; $\boldsymbol{e}_{Ox}$, $\boldsymbol{e}_{Oy}$, and $\boldsymbol{e}_{Oz}$ are unit pure forces acting along the lines Ox, Oy, and Oz, respectively; and $\boldsymbol{e}_x$, $\boldsymbol{e}_y$, and $\boldsymbol{e}_z$ are unit pure couples in the x, y, and z directions. These vectors are illustrated in Figure 1. The bases $\mathcal{D}$ and $\mathcal{E}$ together define a Plücker coordinate system on M^6 and F^6.

Now that we have the bases, let us work out the Plücker coordinates of a spatial velocity $\hat{\boldsymbol{v}} \in \mathsf{M}^6$ and a spatial force $\hat{\boldsymbol{f}} \in \mathsf{F}^6$. Starting with $\hat{\boldsymbol{v}}$, the first step is to identify the two 3-D vectors, $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$, that represent $\hat{\boldsymbol{v}}$ at reference point O. Once this has been done, the motion of the body can be regarded as the sum of a pure rotation of $\boldsymbol{\omega}$ about O (i.e., the axis of rotation passes through O) and a pure translation of $\boldsymbol{v}_O$. The next step is to express these vectors in basis $\mathcal{C}$:

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

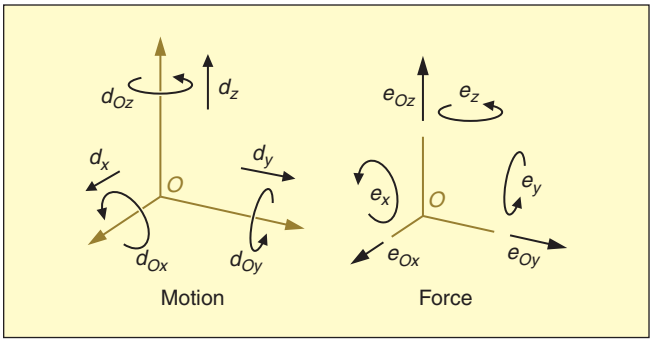


Figure 1. *Plücker basis vectors.*

and

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

We can now describe the spatial velocity $\hat{\boldsymbol{v}}$ as the sum of six components: three rotations of magnitudes ω_x, ω_y, and ω_z about the lines Ox, Oy, and Oz, respectively, plus three translations of magnitudes v_{Ox}, v_{Oy}, and v_{Oz} in the x, y, and z directions. On comparing this description with the definitions of the basis vectors in $\mathcal{D}$, it follows immediately that

$$\hat{\boldsymbol{v}} = \omega_x \boldsymbol{d}_{Ox} + \omega_y \boldsymbol{d}_{Oy} + \omega_z \boldsymbol{d}_{Oz} + v_{Ox} \boldsymbol{d}_x + v_{Oy} \boldsymbol{d}_y + v_{Oz} \boldsymbol{d}_z. \tag{7}$$

Applying the same procedure to $\hat{\boldsymbol{f}}$, we identify $\boldsymbol{f}$ and $\boldsymbol{n}_O$ as the 3-D vectors representing $\hat{\boldsymbol{f}}$ at O and express them in basis $\mathcal{C}$:

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

and

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

The force $\hat{\boldsymbol{f}}$ can then be described as the sum of six components: pure forces of magnitudes f_x, f_y, and f_z acting along the lines Ox, Oy, and Oz, respectively, and pure couples of magnitudes n_{Ox}, n_{Oy}, and n_{Oz} in the x, y, and z directions. On comparing this description with the definitions of the basis vectors in $\mathcal{E}$, it follows immediately that

$$\hat{\boldsymbol{f}} = n_{Ox} \boldsymbol{e}_x + n_{Oy} \boldsymbol{e}_y + n_{Oz} \boldsymbol{e}_z + f_x \boldsymbol{e}_{Ox} + f_y \boldsymbol{e}_{Oy} + f_z \boldsymbol{e}_{Oz}. \tag{8}$$

So, the Plücker coordinates of $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{f}}$ are none other than the Cartesian coordinates in $\mathcal{C}$ of the vectors $\boldsymbol{\omega}$, $\boldsymbol{v}_O$, $\boldsymbol{f}$, and $\boldsymbol{n}_O$. If $\underline{\hat{\boldsymbol{v}}}$ and $\underline{\hat{\boldsymbol{f}}}$ are the coordinate vectors representing $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{f}}$, then we can write them in full as

$$\underline{\hat{\boldsymbol{v}}} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_{Ox} \\ v_{Oy} \\ v_{Oz} \end{bmatrix}, \qquad \underline{\hat{\boldsymbol{f}}} = \begin{bmatrix} n_{Ox} \\ n_{Oy} \\ n_{Oz} \\ f_x \\ f_y \\ f_z \end{bmatrix}, \tag{9}$$

or we can write them in short form as

$$\underline{\hat{\boldsymbol{v}}} = \begin{bmatrix} \underline{\boldsymbol{\omega}} \\ \underline{\boldsymbol{v}}_O \end{bmatrix}, \qquad \underline{\hat{\boldsymbol{f}}} = \begin{bmatrix} \underline{\boldsymbol{n}}_O \\ \underline{\boldsymbol{f}} \end{bmatrix}, \tag{10}$$

where $\underline{\boldsymbol{\omega}} = [\omega_x \ \omega_y \ \omega_z]^{\mathrm{T}}$, etc. This short format is very convenient and popular, but it does have one small drawback: it looks like a stacked pair of 3-D vectors and is therefore capable of misleading readers into thinking that a spatial vector is a stacked pair of 3-D vectors; but in reality, it is only the coordinates that are being stacked. Equation (10) is nothing more nor less than a shorthand for (9).

Equations (9) and (10) list the angular coordinates above the linear ones. This order is not essential, and you will find

examples in the literature where the linear coordinates precede the angular ones. From a mathematical point of view, the difference is purely cosmetic: it is simply a consequence of the order in which we have chosen to list the basis vectors in $\mathcal{D}$ and $\mathcal{E}$. However, if you want to use spatial arithmetic software, then you will have to comply with the order expected by the software.

The pattern of coordinate names appearing in (9) and (10) is not the general case because we have used some special symbols. In particular, we used the standard symbol $\boldsymbol{\omega}$ for angular velocity and the (not quite so) standard symbol $\boldsymbol{n}$ for moment. In the general case, the coordinates' names are derived from the name of the vector they describe. For a generic motion vector $\hat{\boldsymbol{m}}$, the coordinates would be called m_x, m_y, m_z, m_{Ox}, m_{Oy}, and m_{Oz}. Of course, you could also number the coordinates (and basis vectors) if you prefer.

Plücker Transforms

Let A and B be two Cartesian frames defining two Plücker coordinate systems, which we shall also call A and B. Let ${}^A\boldsymbol{m}, {}^B\boldsymbol{m}, {}^A\boldsymbol{f}, {}^B\boldsymbol{f} \in \mathsf{R}^6$ be coordinate vectors representing $\boldsymbol{m} \in \mathsf{M}^6$ and $\boldsymbol{f} \in \mathsf{F}^6$ in A and B coordinates, respectively. The coordinate transformation rules for these vectors are as follows:

$$ {}^B\boldsymbol{m} = {}^B\boldsymbol{X}_A \, {}^A\boldsymbol{m}, $$

and

$$ {}^B\boldsymbol{f} = {}^B\boldsymbol{X}_A^* \, {}^A\boldsymbol{f}, $$

where ${}^B\boldsymbol{X}_A$ is the coordinate transformation matrix from A to B coordinates for motion vectors, and ${}^B\boldsymbol{X}_A^*$ is the corresponding matrix for force vectors. The two are related by

$$ {}^B\boldsymbol{X}_A^* = ({}^B\boldsymbol{X}_A)^{-\mathrm{T}}, $$

[cf. (2)]. The formulae for ${}^B\boldsymbol{X}_A$ and ${}^B\boldsymbol{X}_A^*$ depend only on the location of B relative to A and are given by

$$ {}^B\boldsymbol{X}_A = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ -\boldsymbol{r}\times & \boldsymbol{1} \end{bmatrix}, \tag{11} $$

and

$$ {}^B\boldsymbol{X}_A^* = \begin{bmatrix} \boldsymbol{E} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{E} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} & -\boldsymbol{r}\times \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}. \tag{12} $$

In these equations, $\boldsymbol{E}$ is the coordinate transform from $\mathcal{C}_A$ to $\mathcal{C}_B$ (the Cartesian coordinate systems defined by frames A and B), and $\boldsymbol{r}$ locates the origin of frame B in $\mathcal{C}_A$ coordinates (see Figure 2).

The symbols $\boldsymbol{0}$ and $\boldsymbol{1}$ denote zero and identity matrices of appropriate dimensions, and the expression $\boldsymbol{r}\times$ is the skew-symmetric matrix

$$ \boldsymbol{r}\times = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \times = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}, \tag{13} $$

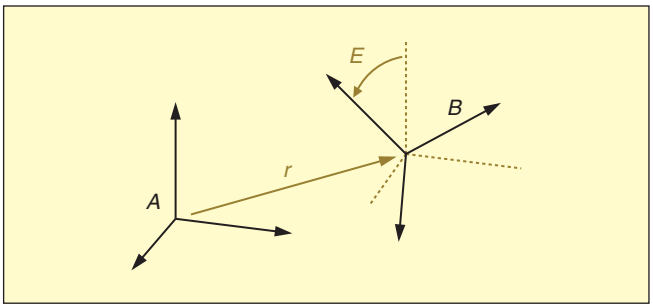


Figure 2. *Location of frame B relative to A.*

which maps any Euclidean vector $\boldsymbol{v}$ to the vector product $\boldsymbol{r} \times \boldsymbol{v}$. (It is the same idea as the matrix $\tilde{\boldsymbol{u}}$ that we encountered in the 3-D example.)

Differentiation

Spatial vectors are differentiated in the same way as any other vector, namely

$$ \frac{\mathrm{d}}{\mathrm{d}x}\boldsymbol{s}(x) = \lim_{\delta x \to 0} \frac{\boldsymbol{s}(x+\delta x) - \boldsymbol{s}(x)}{\delta x}. \tag{14} $$

The derivative of a motion vector is a motion vector, and the derivative of a force vector is a force vector. Unfortunately, the situation gets a little more complicated when we come to coordinate vectors because of the need to distinguish between the derivative of a coordinate vector and the coordinate vector representing a derivative.

To clarify this distinction, let $\boldsymbol{m} \in \mathsf{M}^6$ be a motion vector, and let ${}^A\boldsymbol{m} \in \mathsf{R}^6$ be a coordinate vector representing $\boldsymbol{m}$ in A coordinates. We can now identify the following quantities:

- $d\boldsymbol{m}/dx$: the derivative of $\boldsymbol{m}$
- ${}^A(d\boldsymbol{m}/dx)$: the vector representing $d\boldsymbol{m}/dx$ in A coordinates
- $d^A\boldsymbol{m}/dx$: the derivative of the coordinate vector ${}^A\boldsymbol{m}$.

The derivative of a coordinate vector is always its componentwise derivative. If the basis vectors do not vary with x, then we have ${}^A(d\boldsymbol{m}/dx) = d^A\boldsymbol{m}/dx$ (and similarly for forces); otherwise, these quantities will differ by a term depending on the derivatives of the basis vectors.

For the special case of a time derivative in a moving Plücker coordinate system, we have the following formulae:

$$ {}^A\left(\frac{d\boldsymbol{m}}{dt}\right) = \frac{d}{dt}{}^A\boldsymbol{m} + {}^A\boldsymbol{v}_A \times {}^A\boldsymbol{m}, \tag{15} $$

and

$$ {}^A\left(\frac{d\boldsymbol{f}}{dt}\right) = \frac{d}{dt}{}^A\boldsymbol{f} + {}^A\boldsymbol{v}_A \times^* {}^A\boldsymbol{f}. \tag{16} $$

In these equations, A is both the name of a Plücker coordinate system and the name of the frame that defines it, while ${}^A\boldsymbol{v}_A$ is the velocity of frame A expressed in A coordinates. These equations introduce two new operators, $\times$ and $\times^*$, which are the spatial-vector equivalents of the cross-product operator

If a rigid body has a velocity of v and an inertia of I, then its momentum is Iv.

for 3-D Euclidean vectors appearing in (13). They are defined (in Plücker coordinates) as follows:

$$\hat{\boldsymbol{v}}\times = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix}\times = \begin{bmatrix} \boldsymbol{\omega}\times & \mathbf{0} \\ \boldsymbol{v}_O\times & \boldsymbol{\omega}\times \end{bmatrix}, \tag{17}$$

and

$$\hat{\boldsymbol{v}}\times^* = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix}\times^* = \begin{bmatrix} \boldsymbol{\omega}\times & \boldsymbol{v}_O\times \\ \mathbf{0} & \boldsymbol{\omega}\times \end{bmatrix} = -\hat{\boldsymbol{v}}\times^{\mathrm{T}}. \tag{18}$$

These operators share many properties with their 3-D counterpart, which can be deduced from their definitions; e.g., $\hat{\boldsymbol{v}}\times\hat{\boldsymbol{v}} = \mathbf{0}$. The reason why there are two operators is because one ($\hat{\boldsymbol{v}}\times$) arises from the motion of the motion basis vectors, while the other ($\hat{\boldsymbol{v}}\times^*$) arises from the motion of the force basis vectors. The operator $\hat{\boldsymbol{v}}\times$ acts on a motion vector, producing a motion-vector result, and the operator $\hat{\boldsymbol{v}}\times^*$ acts on a force vector, producing a force-vector result.

A useful corollary of (15) and (16) is that if $\boldsymbol{m}$ and $\boldsymbol{f}$ are fixed in a body B and are varying only because B is in motion, then

$$\dot{\boldsymbol{m}} = \boldsymbol{v}\times\boldsymbol{m}, \tag{19}$$

and

$$\dot{\boldsymbol{f}} = \boldsymbol{v}\times^*\boldsymbol{f}, \tag{20}$$

where $\boldsymbol{v}$ is the velocity of B. A similar formula for rigid-body inertia appears in (27). If $\boldsymbol{s} \in \mathsf{M}^6$ denotes a revolute or prismatic joint axis that is fixed in body B, then $\dot{\boldsymbol{s}} = \boldsymbol{v}\times\boldsymbol{s}$.

Acceleration

Spatial acceleration is just the time derivative of spatial velocity. However, that seemingly innocuous definition contains a surprise, as we shall now discover. Let O be a fixed point in space, and let B be a rigid body whose spatial velocity is given by $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ at O. Let O' be a body-fixed point that happens to coincide with O at the current instant (time t), and let $\boldsymbol{r} = \overrightarrow{OO'}$. Thus, $\boldsymbol{r} = \mathbf{0}$ at time t, but $\boldsymbol{r} \neq \mathbf{0}$ in general. As O is stationary, the velocity and acceleration of O' are given by

$$\boldsymbol{v}_{O'} = \dot{\boldsymbol{r}},$$

and

$$\dot{\boldsymbol{v}}_{O'} = \ddot{\boldsymbol{r}}.$$

Now, the relationship between $\boldsymbol{v}_O$ and $\boldsymbol{v}_{O'}$ is

$$\boldsymbol{v}_O = \boldsymbol{v}_{O'} - \boldsymbol{\omega}\times\boldsymbol{r},$$

so,

$$\dot{\boldsymbol{v}}_O = \dot{\boldsymbol{v}}_{O'} - \dot{\boldsymbol{\omega}}\times\boldsymbol{r} - \boldsymbol{\omega}\times\dot{\boldsymbol{r}}.$$

Therefore, at the current instant (where $\boldsymbol{r} = \mathbf{0}$), we have

$$\begin{aligned} \boldsymbol{v}_{O'} &= \dot{\boldsymbol{r}}, \quad & \boldsymbol{v}_O &= \dot{\boldsymbol{r}}, \\ \dot{\boldsymbol{v}}_{O'} &= \ddot{\boldsymbol{r}}, \quad & \dot{\boldsymbol{v}}_O &= \ddot{\boldsymbol{r}} - \boldsymbol{\omega}\times\dot{\boldsymbol{r}}. \end{aligned} \tag{21}$$

The formula for spatial acceleration is therefore

$$\hat{\boldsymbol{a}} = \frac{\mathrm{d}}{\mathrm{d}t}\hat{\boldsymbol{v}} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\boldsymbol{r}} - \boldsymbol{\omega}\times\dot{\boldsymbol{r}} \end{bmatrix}. \tag{22}$$

We can explain this in words as follows. $\dot{\boldsymbol{r}}$ is the velocity of a particular body-fixed particle, but $\boldsymbol{v}_O$ is the velocity measured at O of the stream of body-fixed particles passing through O. Likewise, $\ddot{\boldsymbol{r}}$ is the acceleration of a particular body-fixed particle, but $\dot{\boldsymbol{v}}_O$ is the rate of change in the velocity at which successive body-fixed particles stream through O.

Despite the slightly greater complexity of (22) compared with the classical description of rigid-body acceleration using $\dot{\boldsymbol{\omega}}$ and $\ddot{\boldsymbol{r}}$, spatial acceleration is significantly easier to use. For example, spatial accelerations can be summed like velocities, and they obey the same coordinate-transformation rule. To take another example, if two bodies B_1 and B_2 are connected by a joint such that the two velocities obey

$$\boldsymbol{v}_2 = \boldsymbol{v}_1 + \boldsymbol{s}\,\dot{q},$$

where $\boldsymbol{s}$ describes the joint axis and $\dot{q}$ is a joint velocity variable, then the relationship between their accelerations is obtained immediately by differentiating the velocity equation:

$$\boldsymbol{a}_2 = \boldsymbol{a}_1 + \dot{\boldsymbol{s}}\,\dot{q} + \boldsymbol{s}\,\ddot{q}.$$

If $\boldsymbol{s}$ describes a joint axis that is fixed in B_2, then $\dot{\boldsymbol{s}} = \boldsymbol{v}_2\times\boldsymbol{s}$, which implies that $\dot{\boldsymbol{s}}\,\dot{q} = \boldsymbol{v}_1\times\boldsymbol{v}_2$. (Can you prove this?) The equivalent 3-D-vector equations are significantly more complicated. For more on the topic of spatial acceleration see, [5] and [7].

Inertia

The spatial inertia of a rigid body is a tensor that maps its velocity to its momentum (which is a force vector). If a body has an inertia of $\boldsymbol{I}$ and a velocity of $\boldsymbol{v}$, then its momentum, $\boldsymbol{h} \in \mathsf{F}^6$, is

$$\boldsymbol{h} = \boldsymbol{I}\boldsymbol{v}. \tag{23}$$

If rigid bodies $B_1 \cdots B_N$ are rigidly connected together to form a single composite rigid body, then the inertia of the composite is

$$\boldsymbol{I}_{\mathrm{tot}} = \sum_{i=1}^{N} \boldsymbol{I}_i, \tag{24}$$

where $\boldsymbol{I}_i$ is the inertia of B_i.

Expressed in any dual coordinate system, spatial inertia is a symmetric, positive-definite matrix (or, in special circumstances,

positive semidefinite). Expressed in Plücker coordinates, the spatial inertia of a rigid body is

$$\boldsymbol{I} = \begin{bmatrix} \bar{\boldsymbol{I}}_c + m\,\boldsymbol{c}\times\boldsymbol{c}\times^{\mathrm{T}} & m\,\boldsymbol{c}\times \\ m\,\boldsymbol{c}\times^{\mathrm{T}} & m\,\mathbf{1} \end{bmatrix}, \tag{25}$$

where m is the body's mass, $\boldsymbol{c}$ is a 3-D vector locating the body's center of mass, and $\bar{\boldsymbol{I}}_c$ is the body's rotational inertia about its center of mass. Observe that a rigid-body inertia is a function of ten parameters: one in m, three in $\boldsymbol{c}$, and six in $\bar{\boldsymbol{I}}_c$. More general kinds of spatial inertia, such as articulated-body and operational-space inertia, do not have the special form shown in (25), and they can be functions of up to 21 independent parameters.

All spatial inertias, whether rigid or not, obey the following coordinate-transformation rule:

$${}^{B}\boldsymbol{I} = {}^{B}\boldsymbol{X}_{A}^{*}\,{}^{A}\boldsymbol{I}\,{}^{A}\boldsymbol{X}_{B}. \tag{26}$$

This formula is valid for any dual coordinate system, not only Plücker coordinates. If a rigid body has a velocity of $\boldsymbol{v}$ and an inertia of $\boldsymbol{I}$, then the time derivative of its inertia is

$$\frac{\mathrm{d}}{\mathrm{d}t}\,\boldsymbol{I} = \boldsymbol{v}\times^{*}\boldsymbol{I} - \boldsymbol{I}\boldsymbol{v}\times. \tag{27}$$

Another useful equation is

$$E = \frac{1}{2}\,\boldsymbol{v}\cdot\boldsymbol{I}\boldsymbol{v}, \tag{28}$$

which gives the kinetic energy of a rigid body.

Equation of Motion

Expressed in spatial form, the equation of motion for a rigid body having a velocity of $\boldsymbol{v}$ and an inertia of $\boldsymbol{I}$ is

$$\boldsymbol{f} = \frac{\mathrm{d}}{\mathrm{d}t}\,(\boldsymbol{I}\boldsymbol{v}) = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{v}\times^{*}\boldsymbol{I}\boldsymbol{v}, \tag{29}$$

where $\boldsymbol{f}$ is the total force acting on the body, and $\boldsymbol{a}$ is the resulting acceleration. [Can you verify this equation using (27)?] In words, it says that the total force acting on a rigid body equals its rate of change of momentum. This equation incorporates both Newton's equation applied to the center of mass and Euler's equation for the rotation of the body about its center of mass.

It is often useful to write the equation of motion in the following simplified form:

$$\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p}, \tag{30}$$

where $\boldsymbol{p} \in \mathsf{F}^6$ is called a *bias force*. There are two main reasons why you might want to do this. First, the algebraic form of this equation is identical to the algebraic form of several other important equations of motion, such as the articulated-body equation of motion and the equation of motion of a rigid body expressed in generalized coordinates. Second, it offers the opportunity to split $\boldsymbol{f}$ into a known part and an unknown part, and incorporate the former into $\boldsymbol{p}$. For example, if the forces acting on the body consisted of an unknown force and a gravitational force, you could define $\boldsymbol{f}$ in (30) to be the unknown force and define $\boldsymbol{p}$ as follows:

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

where $\boldsymbol{f}_{\mathrm{g}}$ is the gravitational force. Incidentally, if $\boldsymbol{a}_{\mathrm{g}}$ is the acceleration due to gravity (in a uniform gravitational field), then the force of gravity acting on a rigid body with inertia $\boldsymbol{I}$ is

$$\boldsymbol{f}_{\mathrm{g}} = \boldsymbol{I}\boldsymbol{a}_{\mathrm{g}}.$$

Motion Constraints

In the simplest case, a motion constraint between two rigid bodies, B_1 and B_2, restricts their relative velocity to a vector subspace $S \subseteq \mathsf{M}^6$, which can vary with time. If r is the dimension of S, then the constraint allows r degrees of relative motion freedom between the two bodies, and consequently imposes $6 - r$ constraints. If $\boldsymbol{s}_1 \cdots \boldsymbol{s}_r$ are any set of vectors that span S (i.e., they form a basis on S), then the relative velocity can be expressed in the form

$$\boldsymbol{v}_{\mathrm{rel}} = \boldsymbol{v}_2 - \boldsymbol{v}_1 = \sum_{i=1}^{r} \boldsymbol{s}_i\,\dot{q}_i,$$

where $\dot{q}_i$ are a set of velocity variables. However, we usually collect the vectors together into a single $6\times r$ matrix $\boldsymbol{S}$, and express the relative velocity as follows:

$$\boldsymbol{v}_2 - \boldsymbol{v}_1 = \boldsymbol{S}\dot{\boldsymbol{q}}, \tag{31}$$

where $\dot{\boldsymbol{q}}$ is an r-dimensional coordinate vector containing the velocity variables. To obtain a constraint on the relative acceleration, we simply differentiate this equation, giving

$$\boldsymbol{a}_2 - \boldsymbol{a}_1 = \dot{\boldsymbol{S}}\dot{\boldsymbol{q}} + \boldsymbol{S}\ddot{\boldsymbol{q}}. \tag{32}$$

In a typical dynamics problem, the quantities $\dot{\boldsymbol{S}}$, $\dot{\boldsymbol{q}}$, and $\boldsymbol{S}$ would all be known, and $\ddot{\boldsymbol{q}}$ would be unknown. Expressions for $\boldsymbol{S}$ and $\dot{\boldsymbol{S}}$ will depend on the type of constraint. If the component vectors of $\boldsymbol{S}$ are fixed in body i ($i = 1$ or 2), then $\dot{\boldsymbol{S}} = \boldsymbol{v}_i \times \boldsymbol{S}$.

Motion constraints are implemented by constraint forces, and constraint forces all have the following special property: a constraint force does no work in any direction of motion permitted by the constraint.

This is simply a statement of D'Alembert's principle of virtual work, or Jourdain's principle of virtual power, depending on

A constraint force does no work in any direction of motion permitted by the constraint.

whether you interpret motion to mean infinitesimal displacement or velocity.

Let $\boldsymbol{f}_c$ be the constraint force implementing the above motion constraint. As the relative motion has been defined to be the motion of B_2 relative to B_1, so we must define $\boldsymbol{f}_c$ to be a force transmitted from B_1 to B_2 (i.e., $\boldsymbol{f}_c$ acts on B_2 and $-\boldsymbol{f}_c$ acts on B_1). To comply with D'Alembert's principle, $\boldsymbol{f}_c$ must satisfy $\boldsymbol{f}_c \cdot \boldsymbol{s}_i = 0$ for all i. Therefore, $\boldsymbol{f}_c$ must satisfy

$$\boldsymbol{S}^{\mathrm{T}} \boldsymbol{f}_c = \boldsymbol{0}. \tag{33}$$

Equations (32) and (33) together define the motion constraint between B_2 and B_1. To model a powered joint, we replace (33) with

$$\boldsymbol{S}^{\mathrm{T}} \boldsymbol{f}_{\mathrm{J}} = \boldsymbol{\tau}, \tag{34}$$

where $\boldsymbol{f}_{\mathrm{J}}$ is the total force transmitted across the joint—the sum of an active force and a constraint force—and $\boldsymbol{\tau}$ is a vector of generalized force variables. The elements of $\boldsymbol{\tau}$ must be defined such that $\boldsymbol{\tau}^{\mathrm{T}} \dot{\boldsymbol{q}}$ is the instantaneous power delivered by the joint to the system.

Further Reading

Spatial vectors are described in detail in [7] and in somewhat less detail in [8]. An older version of spatial vectors is described in [4]. Web-based materials are available from [9], including a slide show, a set of exercises with answers, the two examples appearing at the beginning of this tutorial, and software for MATLAB and Octave that implements spatial vector arithmetic and a selection of the most important dynamics algorithms for robotics. Similar materials, plus a lot of materials from other authors on screw theory, are available at [17].

Spatial vectors are closely related to screw theory, to motor algebra, and the Lie algebra se(3). Materials on these topics can be found in [1]–[3], [10]–[12], and [15], [16]. Screw theory emphasizes geometrical aspects of 6-D vectors, expressed in terms of straight lines, pitches (of screws), and magnitudes. Lie algebra takes a more formal approach: se(3) is the tangent space at the identity of the Lie group se(3); so you can expect notions of group theory, manifolds, and tangent spaces to appear. Motor algebra has two forms: one based on real numbers [2], [10], [11] and the other on dual numbers [3]. The latter is not suitable for dynamics because dual numbers don't work with inertias.

One more notation that deserves a mention is the spatial operator algebra of [13], [14]. This notation shows obvious signs of 3-D-vector thinking, but its most important feature is the way the authors have stacked up the 6-D vectors to make $6N$-dimensional vectors and $6N \times 6N$ matrices that describe properties of a whole rigid-body system comprising N bodies.

This concludes part 1 of this tutorial. Part 2 will show how spatial vectors are applied to several standard problems in robot kinematics and dynamics. In particular, it will show how a problem can be solved algebraically using spatial vectors, and the resulting solution translated directly into short, simple computer code for performing the desired calculations.

Keywords

Robot dynamics, spatial vectors, rigid body dynamics, Plücker coordinates, screw theory.

References

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

[2] E. J. Baker and K. Wohlhart, *Motor Calculus: A New Theoretical Device for Mechanics* (Transl. of [10] and [11]). Graz, Austria: Institute for Mechanics, TU Graz, 1996.

[3] L. Brand, *Vector and Tensor Analysis*, 4th ed. New York: Wiley, 1953.

[4] R. Featherstone, *Robot Dynamics Algorithms*. Boston: Kluwer, 1987.

[5] R. Featherstone, "The acceleration vector of a rigid body," *Int. J. Robot. Res.*, vol. 20, no. 11, pp. 841–846, 2001.

[6] R. Featherstone, "Plücker basis vectors," in *Proc. IEEE Int. Conf. Robotics and Automation*, Orlando, FL, May 15–19, 2006, pp. 1892–1897.

[7] R. Featherstone, *Rigid Body Dynamics Algorithms*. New York: Springer-Verlag, 2008.

[8] R. Featherstone and D. E. Orin, "Dynamics," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin: Springer-Verlag, 2008, pp. 35–65.

[9] R. Featherstone. (2010). Spatial vector algebra [Online]. Available: http://users.cecs.anu.edu.au/∼roy/spatial/

[10] R. von Mises, "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, 1924.

[11] R. von Mises, "Anwendungen der Motorrechnung [Applications of Motor Calculus]," *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 4, no. 3, pp. 193–213, 1924.

[12] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC, 1994.

[13] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "A spatial operator algebra for manipulator modelling and control," *Int. J. Robot. Res.*, vol. 10, no. 4, pp. 371–381, 1991.

[14] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "Spatial operator algebra for multibody system dynamics," *J. Astronaut. Sci.*, vol. 40, no. 1, pp. 27–50, 1992.

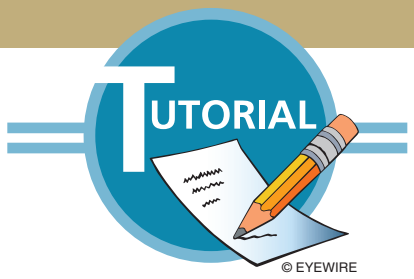
[15] J. M. Selig, *Geometrical Methods in Robotics*. New York: Springer-Verlag, 1996.

[16] J. M. Selig, *Geometric Fundamentals of Robotics*. New York: Springer-Verlag, 2009.

[17] D. Zlatanov. (2009). Screw-theory based methods in robotics [Online]. Available: http://www.dimec.unige.it/PMAR/summer_screws/SS09/PMAR_SuSc09_academic.htm and http://www.summerscrews.org

Roy Featherstone received his Ph.D. degree from Edinburgh University in 1984. He spent approximately seven years working in industry before moving to Oxford University in 1992 to take up an EPSRC Advanced Research Fellowship. He currently works for the ANU, which he joined in 2001. His research interests include robot motion, in particular, the kinematics, dynamics, control and enabling technology of complex and energetic robot motion. He is also interested in dynamics algorithms and simulation. He is a Fellow of the IEEE.

Address for Correspondence: Roy Featherstone, School of Engineering, RSISE Building 115, The Australian National University, Canberra, ACT 0200, Australia. E-mail: Roy.Featherstone@anu.edu.au.



A Beginner's Guide to 6-D Vectors (Part 2)

From Equations to Software

BY ROY FEATHERSTONE

Spatial vectors are six-dimensional (6-D) vectors that describe the motions of rigid bodies and the forces acting upon them. In Part 1, we saw how spatial vectors can simplify the process of expressing and analyzing the dynamics of a simple rigid-body system. In this tutorial, we shall examine the application of spatial vectors to various problems in robot kinematics and dynamics. To demonstrate that spatial vectors are both a tool for analysis and a tool for computation, we shall consider both the mathematical solution of a problem and the computer code to calculate the answer.

To illustrate the power of spatial vectors, we shall consider the class of robots having branched connectivity. This class includes legged robots, humanoids and multifingered grippers, as well as traditional serial robot arms; however, it does not include robots with kinematic loops, such as parallel robots. To cope with this degree of generality, we shall take a model-based approach: the robot mechanism is described by means of a standard set of quantities stored in a model data structure, and the equations, algorithms, and computer code are designed to use those quantities in their calculations.

Following the same pattern as Part 1, this tutorial starts with a specific example and proceeds to analyze it in detail; the example in this instance being the computer code to implement a model-based inverse dynamics calculation using the recursive Newton–Euler algorithm. Subsequent sections then examine a variety of topics in kinematics and present the two main recursive algorithms for forward dynamics: the composite-rigid-body algorithm and the articulated-body algorithm.

It is assumed that the readers have already read Part 1 [6], or equivalent material, and therefore, they are familiar with the notation and basic concepts of spatial vector algebra.

Digital Object Identifier 10.1109/MRA.2010.939560

A Computational Example

Inverse dynamics is the problem of calculating the forces required to produce a given acceleration. It is a relatively easy problem, and therefore, a good place to start. A model-based inverse dynamics calculation can be expressed mathematically as

$$\boldsymbol{\tau} = \mathrm{ID}(\mathrm{model}, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}), \tag{1}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$, and $\boldsymbol{\tau}$ denote vectors of joint position, velocity, acceleration, and force variables, respectively, and *model* denotes a data structure containing a description of the robot. The objective is to calculate the numeric value of ID given the numeric values of its arguments.

Figure 1 shows the MATLAB source code for an implementation of (1) using the recursive Newton–Euler algorithm. This is a complete implementation: you could type it in right now (minus the line numbers) and get it to work, provided you also typed in the (very short) definitions of the functions `jcalc`, `crm`, and `crf`, which are discussed later in this tutorial. The code in Figure 1 can calculate the inverse dynamics of

1070-9932/10/$26.00©2010 IEEE

any robot mechanism in which the bodies are connected together in the manner of a topological tree, and each joint is either revolute, prismatic, or helical (a screw joint).

The code is clearly very short. This degree of brevity would not be possible using three-dimensional (3-D) vectors. Once the basics of spatial vectors are understood, code like this requires relatively little effort to write, test, and debug compared with the equivalent 3-D-vector code. Furthermore, code like this is relatively easy for others to read, understand, and adapt to other purposes.

```
 1  function  tau = ID( model, q, qd, qdd )
 2  for i = 1:model.N
 3    [ XJ, s{i} ] = jcalc( model.pitch(i), q(i) );
 4    vJ = s{i}*qd(i);
 5    Xup{i} = XJ * model.Xtree{i};
 6    pi = model.parent(i);
 7    if pi == 0
 8      v{i} = vJ;
 9      a{i} = Xup{i} * [0;0;0;0;0;9.81] + s{i}*qdd(i);
10    else
11      v{i} = Xup{i}*v{pi} + vJ;
12      a{i} = Xup{i}*a{pi} + s{i}*qdd(i) + crm(v{i})*vJ;
13    end
14    f{i} = model.I{i}*a{i} + crf(v{i})*model.I{i}*v{i};
15  end
16  for i = model.N:-1:1
17    tau(i,1) = s{i}' * f{i};
18    pi = model.parent(i);
19    if pi ~= 0
20      f{pi} = f{pi} + Xup{i}'*f{i};
21    end
22  end
```

Figure 1. *MATLAB code for inverse dynamics calculation.*

Model Data Structure

Before we study the code in detail, let us first examine the contents of the model data structure. This structure contains the following fields:

- `model.N`: an integer specifying the number of bodies in the mechanism
- `model.parent`: an array of integers, called the parent array, describing the connectivity of the mechanism
- `model.Xtree`: an array of Plücker coordinate transforms describing the relative locations of the joints within each body
- `model.pitch`: an array of floating point numbers describing the pitch (and therefore, the type) of each joint
- `model.I`: an array of spatial inertias giving the inertia of each body expressed in link coordinates.

This data is sufficient to describe a general kinematic tree in which the joints are revolute, prismatic, or helical. The term "kinematic tree" simply means a rigid-body system in which the connectivity is that of a topological tree. It is derived from the older term "kinematic chain." The small set of joint types is not quite as limiting as it appears, because many common joint types can be emulated by a chain of revolute and prismatic joints connected together by massless bodies. For example, a spherical joint can be emulated by a chain of three revolute joints with axes passing through the rotation center of the spherical joint. This works so long as the chain does not enter a kinematic singularity.

At this point, you might be wondering why helical joints have been included. The short answer is to demonstrate how easy it is, when using spatial vectors, to go beyond the basic repertoire of revolute and prismatic joints. A longer answer is that helical joints are more general than revolute or prismatic ones, so their inclusion represents a genuine increase in generality. Also, helical joints are an example of a joint type that requires a parameter (the pitch of the helix), so their inclusion provides an opportunity to include joint parameters in a robot model.

The next three subsections explain how the fields in the model data structure are used to model a robot mechanism, and then, we shall return to the code in Figure 1 and the algorithm it implements.

Connectivity

The connectivity of a robot mechanism can be represented by a connectivity graph, which is an undirected graph in which the nodes represent bodies and the arcs represent joints. A couple of examples are shown in Figure 2. If the robot is a kinematic tree, then its connectivity graph is a topological tree. To describe the connectivity of a kinematic tree, we first number the bodies and joints according to a standard scheme. For a robot having a fixed base, the numbering proceeds as follows:

1) The fixed base is assigned the number 0 and serves as the root node of the tree.

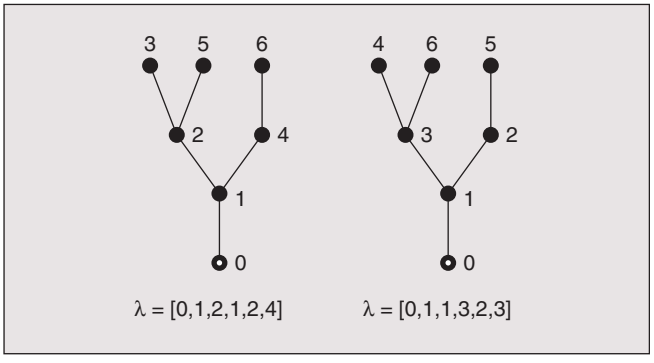


Figure 2. *Two numberings of a simple tree and their corresponding parent arrays.*

2) The remaining bodies are numbered consecutively from 1 to N in any order such that each body has a higher number than its parent.
3) The joints are then numbered from 1 to N such that joint i is the joint that connects body i to its parent.

Having numbered the bodies and joints, the connectivity can be described using a parent array, called λ, which is defined such that $\lambda(i)$ is the body number of the parent of body i. This array has the following special property, which is a consequence of the numbering scheme:

$$0 \leq \lambda(i) < i \qquad \text{for all } 1 \leq i \leq N. \tag{2}$$

Many algorithms rely on this property. Note that neither the numbering nor the parent array is unique. To illustrate this, Figure 2 shows two possible numberings of the same graph and their corresponding parent arrays. For the tree on the left, we have $\lambda(1) = 0$, $\lambda(2) = 1$, $\lambda(3) = 2$, and so on, indicating that body 0 is the parent of body 1, body 1 is the parent of body 2, and so on. Joint numbers have not been shown, because they can be deduced from the body numbers: joint i connects between bodies i and $\lambda(i)$.

To model a mobile robot, we first place a Cartesian coordinate frame at any convenient fixed location in space. This frame serves as a virtual fixed base. We then introduce a six-degree-of-freedom (six-DoF) joint between the fixed base and any one body of the mobile robot. The chosen body is then called the floating base. Now, a six-DoF joint does not introduce any kinematic constraints, so it does not restrict the mobility of the mobile robot. Instead, its purpose is to supply the extra variables needed to identify the position and orientation of the robot relative to its virtual fixed base. Having made these two modifications, the mobile robot can be treated as a fixed-base robot and numbered as described above. (The floating base will therefore be body number 1.)

Although λ alone already provides a complete description of the connectivity, it is often helpful to supplement λ with the following sets:

- $\mu(i)$: the set of children of body i,
- $\kappa(i)$: the set of joints on the path between body i and the root, and
- $\nu(i)$: the set of bodies in the subtree starting at body i.

For the left-hand tree in Figure 2, we have $\mu(2) = \{3, 5\}$, $\kappa(3) = \{1, 2, 3\}$, $\nu(4) = \{4, 6\}$, $\mu(5) = \emptyset$ (the empty set), and so on.

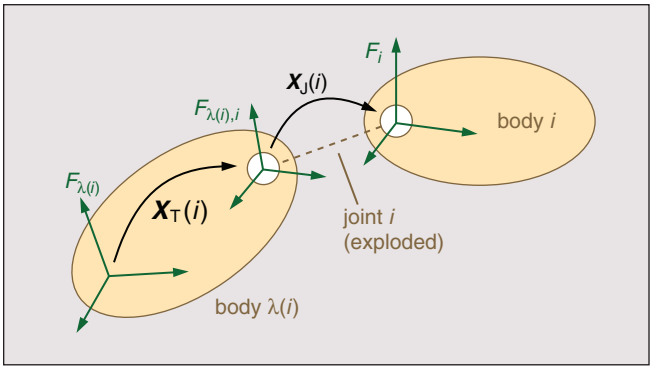


Figure 3. *Coordinate frames and transforms associated with joint i.*

Geometry

The geometrical part of a robot model is the part that specifies the relative locations of the joints in each body. It is also the part that defines a link coordinate system for each body so that quantities like `model.I{i}` can be expressed and stored in link-i coordinates. (Link is the technical term for a body in a mechanical linkage, so link and body can be used interchangeably here.)

To describe the geometry, the first step is to introduce a pair of coordinate frames for each joint: one fixed in each of the two bodies connected by the joint. For joint i, which connects between bodies i and $\lambda(i)$, we introduce a frame F_i that is fixed in body i and a frame $F_{\lambda(i),i}$ that is fixed in body $\lambda(i)$ (see Figure 3). We also introduce a special frame, F_0, which is fixed in body 0 and which serves as an absolute, world, or reference frame (take your pick) for the whole robot. For a mobile robot, F_0 is the frame that was introduced earlier to serve as a virtual fixed base.

The frames can be located anywhere in their respective bodies, provided they satisfy the following rules:

1) Frames F_i and $F_{\lambda(i),i}$ must coincide when the joint variable of joint i is zero.
2) Frames F_i and $F_{\lambda(i),i}$ must comply with the joint-specific alignment requirements of joint i.

As an example of rule 2, if joint i is revolute or helical, then the z-axes of F_i and $F_{\lambda(i),i}$ must lie on the joint's rotation or screw axis. If, instead, joint i is prismatic, then the two z-axes must be parallel to the joint's direction of translation. The purpose of this rule is to ensure that the joint coordinate transform (labeled $\boldsymbol{X}_J(i)$ in Figure 3) takes a canonical form for each joint type. For a revolute joint, $\boldsymbol{X}_J(i)$ is a pure rotation about the z-axis. (More information on this topic is provided in the next subsection.)

The above rules stipulate only the minimum necessary constraints on the placement of coordinate frames and do not constrain them completely. It is therefore possible to introduce additional rules for the purpose of further constraining their locations. The most well-known example is the scheme of Denavit and Hartenberg, which has the special property that the location of F_i relative to $F_{\lambda(i)}$ is a function of only four parameters, one of which serves as the joint variable [1], [3], [9].

At the end of this process, there are $2N + 1$ frames in total, of which $N + 1$ have names of the form F_i, and N have names of the form $F_{i,j}$, where $j \in \mu(i)$ (which is the same condition as $i = \lambda(j)$). Every body in the system, including the fixed base, now contains exactly one frame F_i plus a variable number of frames $F_{i,j}$, one for each $j \in \mu(i)$. At this point, we select F_i to define the link coordinate system for body i, i.e., link-i coordinates. Thus, the spatial inertia stored in `model.I{i}` is expressed in the (Plücker) coordinate system defined by frame F_i.

A complete description of the robot's geometry can now be obtained as follows. Let $\boldsymbol{X}_T(i)$ be the Plücker coordinate transform from link-$\lambda(i)$ coordinates to the coordinate system defined

by frame $F_{\lambda(i),i}$, as shown in Figure 3. Now, a Plücker transform implicitly describes the relative locations of two coordinate frames; so $\boldsymbol{X}_{\mathrm{T}}(i)$ serves to locate $F_{\lambda(i),i}$ relative to F_i, and a complete set of transforms, $\boldsymbol{X}_{\mathrm{T}}(1), \ldots, \boldsymbol{X}_{\mathrm{T}}(N)$, serves to locate every $F_{i,j}$ relative to its corresponding F_i. These transforms are stored in the array `model.Xtree`, so that `model.Xtree{i}` $= \boldsymbol{X}_{\mathrm{T}}(i)$.

Let ${}^{i}\boldsymbol{X}_{\lambda(i)}$ denote the Plücker coordinate transform from link-$\lambda(i)$ to link-i coordinates for motion vectors (the corresponding transform for force vectors is ${}^{i}\boldsymbol{X}^{*}_{\lambda(i)}$). This transform locates F_i relative to $F_{\lambda(i)}$ and therefore locates body i relative to body $\lambda(i)$. From Figure 3, we can see that

$$ {}^{i}\boldsymbol{X}_{\lambda(i)} = \boldsymbol{X}_{\mathrm{J}}(i)\,\boldsymbol{X}_{\mathrm{T}}(i). \tag{3} $$

Equation (3) brings together the connectivity data, $\lambda(i)$, the geometry data, $\boldsymbol{X}_{\mathrm{T}}(i)$, and the joint data, via $\boldsymbol{X}_{\mathrm{J}}(i)$, to express the relative locations of adjacent bodies as a function of the joint position variables. This calculation appears on line 5 of Figure 1, where ${}^{i}\boldsymbol{X}_{\lambda(i)}$ is calculated and stored in the variable `Xup{i}`. Quantities like ${}^{i}\boldsymbol{X}_{\lambda(i)}$ are called link-to-link coordinate transforms.

Joint Models

A joint is a kinematic constraint between two bodies. To identify them individually, we call one body the predecessor and the other the successor. In a kinematic tree, the predecessor of joint i is body $\lambda(i)$, and its successor is body i. By convention, we define the velocity across a joint to be the velocity of the successor relative to the predecessor, and the force across a joint to be a force transmitted from the predecessor to the successor. Thus, if $\boldsymbol{v}_{\mathrm{J}i}$ and $\boldsymbol{f}_{\mathrm{J}i}$ are the spatial velocity and force across joint i, then

$$ \boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{v}_i - \boldsymbol{v}_{\lambda(i)}, \tag{4} $$

where $\boldsymbol{v}_i$ is the velocity of body i, and $\boldsymbol{f}_{\mathrm{J}i}$ is the force transmitted from body $\lambda(i)$ to body i through the joint.

A mathematical model of a joint consists of two quantities: a coordinate transform, $\boldsymbol{X}_{\mathrm{J}}$, and a motion subspace matrix, $\boldsymbol{S}$ (also known as a free-modes matrix). For joint i, $\boldsymbol{X}_{\mathrm{J}}(i)$ is the coordinate transform from $F_{\lambda(i),i}$ to F_i, as shown in Figure 3, and $\boldsymbol{S}_i$ defines the following relationships between the joint variables and spatial vectors:

$$ \boldsymbol{v}_{\mathrm{J}i} = \boldsymbol{S}_i\,\dot{\boldsymbol{q}}_{\mathrm{i}} \tag{5} $$

and

$$ \boldsymbol{\tau}_i = \boldsymbol{S}_i^{\mathrm{T}}\boldsymbol{f}_{\mathrm{J}i}, \tag{6} $$

where $\dot{\boldsymbol{q}}_i$ and $\boldsymbol{\tau}_i$ are the subvectors of $\dot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ that contain the velocity and force variables, respectively, for joint i. We can see an instance of (5) and (6) on lines 4 and 17, respectively, in Figure 1. Incidentally, $\dot{\boldsymbol{q}}_i$ and $\boldsymbol{\tau}_i$ also satisfy

$$ \boldsymbol{\tau}_i \cdot \dot{\boldsymbol{q}}_i = \boldsymbol{f}_{\mathrm{J}i} \cdot \boldsymbol{v}_{\mathrm{J}i}, \tag{7} $$

which is known as the power-balance equation. The scalar on the left is the mechanical power delivered to the robot mechanism at joint i, expressed in terms of joint variables, and the scalar on the right is the same physical quantity expressed in terms of spatial vectors. The two are necessarily equal.

```
function  [XJ,s] = jcalc( pitch, q )
if pitch == 0           % revolute joint
  XJ = rotz(q);
  s = [0;0;1;0;0;0];
elseif pitch == inf     % prismatic joint
  XJ = xlt([0 0 q]);
  s = [0;0;0;0;0;1];
else                    % helical joint
  XJ = rotz(q)*xlt([0 0 q*pitch]);
  s = [0;0;1;0;0;pitch];
end
```

Figure 4. *MATLAB code for function* `jcalc`.

A computational model of a joint consists of a piece of code (such as `jcalc` in Figure 4) that computes the numeric values of $\boldsymbol{X}_{\mathrm{J}}$ and $\boldsymbol{S}$ as a function of the numeric values of the joint variables and parameters (if any). If $\boldsymbol{S}$ varies as a function of the joint's position variables, then it is also necessary to compute the numeric value of a term that depends on $\partial \boldsymbol{S}/\partial \boldsymbol{q}$ (see $\boldsymbol{c}_{\mathrm{J}}$ on p. 80 of [3]).

If joint i permits n_i degrees of motion freedom, then $\boldsymbol{S}_i$ is a $6 \times n_i$ matrix and $\dot{\boldsymbol{q}}_i$ is an $n_i \times 1$ vector. The total number of joint variables is then

$$ n = \sum_{i=1}^{N} n_i\,. \tag{8} $$

This is the dimension of the vectors in (1). However, in this tutorial, we have chosen to limit the repertoire of joint types to revolute, prismatic, and helical. These are all single-DoF joints, so we have $n_i = 1$ for every joint in the mechanism and therefore also $n = N$. Two more simplifications are:

1) the motion subspace matrix simplifies to a joint axis vector $\boldsymbol{s}_i$, and
2) the variables for joint i are the ith elements of their corresponding joint-space vectors.

Item 2 refers to expressions like `q(i)` on line 3 of Figure 1 and `qd(i)` on line 4. These expressions simply extract the ith element of `q`, `qd`, etc. In the general case, the variables for joint i would be n_i-dimensional subvectors of `q`, `qd`, etc., and expressions such as `q(i)` and `qd(i)` would have to be replaced with something a little more complicated.

Another simplification is that revolute and prismatic joints can be regarded as helical joints having zero pitch and infinite pitch, respectively; so it is possible to use the array `model.pitch` both to define the type of each joint and to supply the pitch parameter for each helical joint. This tactic can be seen in the source code of `jcalc`, which is shown in Figure 4. As you can see from this code, a revolute joint implements a pure rotation about the (local) z-axis, a prismatic joint implements a pure translation in the z-direction, and a helical joint implements a screwing motion about the z-axis, in which the pitch

Table 1. Instant spatial vector arithmetic (based on Table A.2 of [3]).

3-D vector formulae $(c = \cos(\theta),\ s = \sin(\theta))$

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

$$\mathrm{ry}(\theta) = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} \qquad \boldsymbol{r}\times = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$$

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 $\boldsymbol{v} = \mathrm{XtoV}(\boldsymbol{X})$
$\boldsymbol{v} = \frac{1}{2}\begin{bmatrix} X_{23} - X_{32} \\ X_{31} - X_{13} \\ X_{12} - X_{21} \\ X_{53} - X_{62} \\ X_{61} - X_{43} \\ X_{42} - X_{51} \end{bmatrix}$
end

parameter determines the lead (per radian) of the screw. The functions `rotz` and `xlt` are defined in Table 1.

To represent a broader range of joint types, we could replace `model.pitch` with an array of joint-descriptor data structures: each descriptor contains a joint-type code and zero or more parameter values, the number of parameters being determined by the type code. More information on joint models can be found in [3], [4], and [8].

The code in Figure 1 shows a clear separation between the code that implements the dynamics algorithm and the code that handles joint-dependent calculations, the latter being hived off into the function `jcalc`. This organizational feature is standard practice with 6-D vectors, but it is harder to achieve using 3-D vectors. As a result, algorithms that are expressed using 3-D vectors tend to be restricted to revolute and prismatic joints, and their descriptions typically contain many statements of the form

if joint type is revolute **then**
  variable = one expression
else
  variable = another expression.

(For example, see the original description of the recursive Newton–Euler algorithm in [7].) Clearly, it is a nontrivial exercise to extend such an algorithm to accommodate a third joint type. This intertwining of algorithms with joint-specific details is an impediment to the development of clean, extensible, general-purpose code, and it is yet another reason to prefer 6-D vectors over 3-D vectors.

Algorithm

We now return to the code in Figure 1. As mentioned earlier, this code implements the recursive Newton–Euler algorithm for calculating inverse dynamics. The equations for this algorithm, expressed using spatial vectors, are as follows:

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{s}_i\dot{\boldsymbol{q}}_i \qquad (\boldsymbol{v}_0 = \boldsymbol{0}) \tag{9}$$

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{s}_i\ddot{\boldsymbol{q}}_i + \boldsymbol{v}_i \times \boldsymbol{s}_i\dot{\boldsymbol{q}}_i \qquad (\boldsymbol{a}_0 = -\boldsymbol{a}_g) \tag{10}$$

$$\boldsymbol{f}_{\mathrm{B}i} = \boldsymbol{I}_i\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\boldsymbol{v}_i \tag{11}$$

$$\boldsymbol{f}_{\mathrm{J}i} = \boldsymbol{f}_{\mathrm{B}i} + \sum_{j\in\mu(i)} \boldsymbol{f}_{\mathrm{J}j} \tag{12}$$

$$\tau_i = \boldsymbol{s}_i^{\mathrm{T}}\boldsymbol{f}_{\mathrm{J}i} \tag{13}$$

Equation (9) states that the velocity of body i is the sum of the velocity of its parent and the velocity across joint i [cf. (4) and (5)]. Equation (10) says the same for accelerations and is simply the derivative of (9). Observe that $\dot{\boldsymbol{s}}_i = \boldsymbol{v}_i \times \boldsymbol{s}_i$, because $\boldsymbol{s}_i$ is fixed in body i. The starting condition for (10) is $\boldsymbol{a}_0 = -\boldsymbol{a}_g$, where $\boldsymbol{a}_g$ is the acceleration caused by gravity. This is a trick that exploits the fact that a uniform gravitational field is indistinguishable from a constant linear acceleration. Therefore, instead of calculating the gravitational force acting on each body and incorporating those forces into (11), we can simply offset every body's spatial acceleration by giving the fixed base a fictitious acceleration of $-\boldsymbol{a}_g$ (see line 9 in Figure 1).

In (11)–(13), $\boldsymbol{f}_{\mathrm{B}i}$ is the net force acting on body i, and $\boldsymbol{f}_{\mathrm{J}i}$ is the force transmitted across joint i. As mentioned earlier, the spatial force across a joint is defined to be a force transmitted from its predecessor body to its successor; so $\boldsymbol{f}_{\mathrm{J}i}$ is a force transmitted from body $\lambda(i)$ to body i. In other words, joint i is causing a force of $+\boldsymbol{f}_{\mathrm{J}i}$ to act on body i and a force of $-\boldsymbol{f}_{\mathrm{J}i}$ to act on body $\lambda(i)$.

The equation of motion for body i is given by (11). As the accelerations are already known, the purpose of this equation is to work out the force required to produce the given acceleration. Equation (12) then calculates the (spatial) joint forces from the body forces. It works as follows: $\boldsymbol{f}_{\mathrm{B}i}$ is the net force acting on body i, so it must be the sum of all the individual forces acting on body i. Having accounted for gravity by means of a fictitious acceleration, instead of a gravitational force field, the only force acting on body i is those transmitted to it via the joints. So, $\boldsymbol{f}_{\mathrm{B}i} = \boldsymbol{f}_{\mathrm{J}i} + \sum_{j\in\mu(i)} (-\boldsymbol{f}_{\mathrm{J}j})$. A small rearrangement of this equation yields (12). Finally, (13) calculates the joint force variable for joint i from the spatial force transmitted across the joint, per (6).

Equations (9)–(13) provide a mathematical description of the recursive Newton–Euler algorithm; they describe the algorithm in principle but omit some calculation details. Translating these equations into a more explicit description of

the algorithm produces the pseudocode shown in Figure 5. On comparing the pseudocode with the equations, we can see that the following things have changed.

1) The vectors in (9)–(13) are tacitly assumed to be expressed in a single common coordinate system; however, their counterparts in Figure 5 are expressed in link coordinates, and the necessary link-to-link coordinate transforms (${}^{i}\boldsymbol{X}_{\lambda(i)}$ and ${}^{\lambda(i)}\boldsymbol{X}_{i}^{*}$) have been incorporated into lines 6, 7, and 13.
2) The method of calculating the link-to-link transforms has been made explicit in lines 4 and 5.
3) The sum over $\mu(i)$ in (12) has been replaced with code that performs the same summation but using $\lambda(i)$ instead.

Item 3 refers to lines 8 and 12–14. Line 8 initializes every variable $\boldsymbol{f}_i$ to have the value of $\boldsymbol{f}_{Bi}$ as given by (11). However, by the time $\boldsymbol{f}_i$ is used on line 11 to calculate τ_i, its value is equal to $\boldsymbol{f}_{Ji}$ as given in (12). The change is effected by lines 12–14, which add each vector $\boldsymbol{f}_i$ ($=\boldsymbol{f}_{Ji}$) back to its parent. By the time $\boldsymbol{f}_i$ is used on line 11, the contributions from all of its children have already been added in.

On comparing the pseudocode in Figure 5 with the MATLAB source code in Figure 1, it can be seen that the translation from pseudocode to source code is entirely straightforward. Again, look how short everything is: five equations have given rise to 15 lines of pseudocode and 22 lines of source code. This degree of brevity would not be possible using 3-D vectors. Of course, if you want source code in a language such as C or C++, then the code will be somewhat longer; however, the translation from pseudocode to source code will still be straightforward, provided you have access to a suitable library of spatial arithmetic functions.

Arithmetic

To perform arithmetic with spatial vectors, you need a spatial arithmetic library. Most arithmetic operations on spatial vectors are just standard matrix arithmetic. Therefore, if you are using a programming language that already has matrix arithmetic built in (such as MATLAB or Octave), then only a small number of additional functions are needed. Table 1 presents a small but sufficient set.

The functions `rotx`, `roty`, `rotz`, and `xlt` construct Plücker coordinate transforms (for motion vectors) from a current coordinate system to one that has been rotated or translated, as appropriate, relative to the current one. Examples of their use can be found in Figure 4. Note that the formulae listed for `rx`, `ry`, and `rz` are coordinate rotation matrices; they rotate the coordinate system in which the vector is represented. In many robotics textbooks (e.g., [1], p. 372), you will find formulae for rotation matrices that rotate the vector itself. These two types of matrix are inverses of each other.

The functions `crm` and `crf` implement the two spatial cross-product operators. Examples of their use can be found in Figure 1. The symbols $\boldsymbol{v}\times$ and $\boldsymbol{v}\times^{*}$ appearing in these two functions are the names of the return values. The expressions $\boldsymbol{v}_{1:3}$ and $\boldsymbol{v}_{4:6}$ are 3-D vectors formed from the first and last three Plücker coordinates of $\boldsymbol{v}$.

```
1  v0 = 0
2  a0 = -ag
3  for i = 1 to N do
4     [XJ, Si] = jcalc(hi,qi)
5     iXλ(i) = XJXT(i)
6     vi = iXλ(i)vλ(i) + sidq̇i
7     ai = iXλ(i)aλ(i) + sidq̈i + vi ×sidq̇i
8     fi = Iiai + vi ×*Iivi
9  end
10 for i = N to 1 do
11    τi  = Si^T fi
12    If λ(i) ≠ 0 then
13       fλ(i) = fλ(i) + λ(i)Xi* fi
14    end
15 end
```

Figure 5. *The recursive Newton–Euler algorithm.*

The function `mcI` constructs a spatial rigid-body inertia from arguments giving the body's mass (m), the position of its center of mass ($\boldsymbol{c}$), and its rotational inertia about its center of mass ($\boldsymbol{I}_C$). You would use this function to initialize the inertia matrices in a robot model data structure.

Finally, `XtoV` calculates a small-magnitude motion vector from the Plücker transform for a small change of coordinates. If A and B denote two Cartesian frames, and also the Plücker coordinate systems defined by those frames, then we can define `XtoV` as follows: if A and B are close together, and $\boldsymbol{X}$ is the Plücker coordinate transform from A to B, then `XtoV` ($\boldsymbol{X}$) approximates to the velocity vector that would move frame A to coincide with B after one time unit. The returned value also happens to be an invariant of $\boldsymbol{X}$ (i.e., $\boldsymbol{v}=\boldsymbol{X}\boldsymbol{v}$) so it has the same value in both A and B coordinates. An example of this function's use appears in the next section.

If you want to perform spatial arithmetic in a programming language such as C or C++, then you will need a more extensive library. Some guidelines on how to build such a library, and a collection of formulae for implementing highly efficient spatial arithmetic, can be found in [3], and a less comprehensive version appears in [3]. Implementations of the functions in Table 1 can be found in [5].

Kinematics

Spatial vectors can be used both for positional kinematics and for instantaneous kinematics. We have already encountered the latter in (9) and (10), which present recursive formulae for calculating body velocities and accelerations from joint velocity and acceleration variables. Body positions can be calculated recursively using the formula

$$ {}^{i}\boldsymbol{X}_{0} = {}^{i}\boldsymbol{X}_{\lambda(i)}\,{}^{\lambda(i)}\boldsymbol{X}_{0}, \qquad (\lambda(i)\neq 0) \tag{14} $$

which calculates the coordinate transform from reference coordinates (frame F_0) to the body coordinate frame (F_i) of each body in the mechanism. As mentioned earlier, a coordinate transform implicitly defines the relative locations of two

coordinate frames; so ${}^{i}\boldsymbol{X}_0$ effectively locates F_i, and therefore also body i, relative to F_0.

Suppose we want to move a particular body, say body b, so that its body coordinate frame, F_b, coincides with a given target frame, F_{d}. We can express the location of F_{d} by means of the coordinate transform ${}^{\mathrm{d}}\boldsymbol{X}_0$, which is assumed to be given. If F_b is already close to F_{d}, then we can use the function XtoV in Table 1 to calculate a small displacement, $\Delta\boldsymbol{p}$, as follows:

$$\Delta\boldsymbol{p} = \mathrm{XtoV}({}^{\mathrm{d}}\boldsymbol{X}_b) = \mathrm{XtoV}({}^{\mathrm{d}}\boldsymbol{X}_0\ {}^{0}\boldsymbol{X}_b). \tag{15}$$

This vector approximates to the small screw displacement that would bring F_b into coincidence with F_{d}, the error in the approximation diminishing quadratically with the angular magnitude of $\Delta\boldsymbol{p}$. Alternatively, $\Delta\boldsymbol{p}$ can be regarded as approximating the velocity that would bring F_b into coincidence with F_{d} in one time unit. As $\Delta\boldsymbol{p}$ is an invariant of ${}^{\mathrm{d}}\boldsymbol{X}_b$, it has the same value in both F_b and F_{d} coordinates.

```
 1  function  q = IKpos( model, body, Xd, q0 )
 2  q = q0;
 3  dpos = ones(6,1);
 4  while norm(dpos) > 1e-10
 5    X = bodypos( model, body, q );
 6    J = bodyJac( model, body, q );
 7    J = X * J;
 8    dpos = XtoV( Xd / X );
 9    dq = J' * ((J*J') \ dpos);
10    q = q + dq;
11  end
```

```
 1  function  X = bodypos( model, b, q )
 2  X = eye(6);
 3  while b > 0
 4    XJ = jcalc( model.pitch(b), q(b) );
 5    X = X * XJ * model.Xtree{b};
 6    b = model.parent(b);
 7  end
```

```
 1  function  J = bodyJac( model, body, q )
 2  e = zeros(1,model.N);
 3  while body ~= 0
 4    e(body) = 1;
 5    body = model.parent(body);
 6  end
 7  J = zeros(6,model.N);
 8  for i = 1:model.N
 9    if e(i)
10      [XJ,S] = jcalc( model.pitch(i), q(i) );
11      Xa{i} = XJ * model.Xtree{i};
12      if model.parent(i) ~= 0
13        Xa{i} = Xa{i} * Xa{model.parent(i)};
14      end
15      J(:,i) = Xa{i} \ S;
16    end
17  end
```

Figure 6. *MATLAB code for iterative inverse kinematics.*

If F_{d} is a reachable position for body b, then there will be a joint position change, $\Delta\boldsymbol{q}$, that causes a displacement of $\Delta\boldsymbol{p}$ in body b. In general, $\Delta\boldsymbol{q}$ will not be unique. The exact relationship between $\Delta\boldsymbol{p}$ and $\Delta\boldsymbol{q}$ may be difficult to obtain, but a first-order approximation is given by

$$\Delta\boldsymbol{p} = {}^{b}\boldsymbol{J}_b\,\Delta\boldsymbol{q}, \tag{16}$$

where ${}^{b}\boldsymbol{J}_b$ is the Jacobian for body b expressed in b coordinates (Jacobians are discussed in the next section). Equations (15) and (16) form the basis for an iterative inverse-kinematics algorithm as follows:

while not close enough **do**
  calculate ${}^{\mathrm{d}}\boldsymbol{X}_b$ and ${}^{b}\boldsymbol{J}_b$
  $\Delta\boldsymbol{p} = \mathrm{XtoV}({}^{\mathrm{d}}\boldsymbol{X}_b)$
  $\Delta\boldsymbol{q} = {}^{b}\boldsymbol{J}_b^{+}\Delta\boldsymbol{p}$
  $\boldsymbol{q} = \boldsymbol{q} + \Delta\boldsymbol{q}$
end,

where ${}^{b}\boldsymbol{J}_b^{+}$ is the pseudoinverse of ${}^{b}\boldsymbol{J}_b$.

Some MATLAB source code to implement this calculation is shown in Figure 6. The variables `q0` and `q` are the initial guess and computed final value of $\boldsymbol{q}$, and the variables `body` and `Xd` contain b and ${}^{\mathrm{d}}\boldsymbol{X}_0$. Line 3 sets `dpos` (= $\Delta\boldsymbol{p}$) to a dummy value that will pass the test on line 4, and the functions `bodypos` and `bodyJac` calculate ${}^{b}\boldsymbol{X}_0$ and ${}^{0}\boldsymbol{J}_b$, respectively. Line 7 calculates ${}^{b}\boldsymbol{J}_b$ from ${}^{0}\boldsymbol{J}_b$, line 8 calculates $\Delta\boldsymbol{p}$, and line 9 calculates $\Delta\boldsymbol{q}$ using the pseudoinverse of ${}^{b}\boldsymbol{J}_b$. Bear in mind that this is not a serious inverse-kinematics function, as it fails to check for a variety of things that can go wrong, such as singularities and unreachable positions.

Jacobians

In common robotics usage, a Jacobian is a matrix that maps the joint-space velocity vector, $\dot{\boldsymbol{q}}$, to some other kind of velocity. We have used the term body Jacobian, and the symbol $\boldsymbol{J}_b$, to refer to the matrix that maps $\dot{\boldsymbol{q}}$ to the spatial velocity of body b, as in

$$\boldsymbol{v}_b = \boldsymbol{J}_b\dot{\boldsymbol{q}}. \tag{17}$$

To obtain a formula for $\boldsymbol{J}_b$, we first express $\boldsymbol{v}_b$ in nonrecursive form:

$$\boldsymbol{v}_b = \sum_{i\in\kappa(b)} \boldsymbol{s}_i\dot{\boldsymbol{q}}_i. \tag{18}$$

This equation simply states that $\boldsymbol{v}_b$ is the sum of the joint velocities of all the joints on the path between body b and the fixed base. Rewriting this equation as

$$\boldsymbol{v}_b = \sum_{i=1}^{N} e_{bi}\boldsymbol{s}_i\dot{\boldsymbol{q}}_i, \tag{19}$$

where

$$e_{bi} = \begin{cases} 1 & \text{if } i \in \kappa(b) \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

yields the following expression for $\boldsymbol{J}_b$:

$$\boldsymbol{J}_b = \begin{bmatrix} e_{b1}\boldsymbol{s}_1 & e_{b2}\boldsymbol{s}_2 & \cdots & e_{bN}\boldsymbol{s}_N \end{bmatrix}. \tag{21}$$

Thus, the body Jacobian for body b is the $6 \times N$ matrix whose ith column is either $\boldsymbol{s}_i$ or zero, depending on whether joint i is or is not on the path between the fixed base and body b. More generally, a body Jacobian is a $1 \times N$ block matrix in which the ith block is the $6 \times n_i$ matrix ${}_{ei}\boldsymbol{S}_i$, and the overall dimension of the Jacobian is $6 \times n$ [cf. (8)]. The code for `bodyJac` in Figure 6 should now make sense: lines 2–6 calculate e_{bi} and lines 8–17 calculate the nonzero columns. For the special case where b is the end effector of a serial robot, we have $e_{bi} = 1$ for all i, which implies the following simplified formula for the end-effector Jacobian:

$$\boldsymbol{J}_{\text{ee}} = \begin{bmatrix} \boldsymbol{s}_1 & \boldsymbol{s}_2 & \dots & \boldsymbol{s}_N \end{bmatrix}. \tag{22}$$

If we need to be explicit about the coordinate system, then (17) and (21) can be written as

$${}^{A}\boldsymbol{v}_b = {}^{A}\boldsymbol{J}_b\,\dot{\boldsymbol{q}} \tag{23}$$

and

$${}^{A}\boldsymbol{J}_b = \begin{bmatrix} e_{b1}{}^{A}\boldsymbol{s}_1 & e_{b2}{}^{A}\boldsymbol{s}_2 & \dots & e_{bN}{}^{A}\boldsymbol{s}_N \end{bmatrix}, \tag{24}$$

where A is the name of a coordinate system. These equations show that every column of a body Jacobian must be expressed in the same coordinate system as the velocity vector it maps to. The coordinate transformation rule for a body Jacobian is therefore

$${}^{B}\boldsymbol{J}_b = {}^{B}\boldsymbol{X}_A\,{}^{A}\boldsymbol{J}_b. \tag{25}$$

Still on the subject of coordinate systems, here is a popular trap for the unwary. You will occasionally encounter an equation of the form

$$\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix} = \boldsymbol{J}\dot{\boldsymbol{q}}, \tag{26}$$

where $\boldsymbol{\omega}$ and $\boldsymbol{v}$ are described as the angular and linear velocity of the end effector (or some other body) expressed in absolute, reference, or base coordinates (i.e., the Cartesian coordinate system defined by frame F_0). In translating this equation from 3-D to spatial vectors, it is tempting to regard the left-hand side as being the Plücker coordinates of a spatial velocity expressed in frame F_0. However, this is nearly always incorrect, because the 3-D vector $\boldsymbol{v}$ nearly always refers to some particular point in the end effector, such as the tool center point, which does not coincide with the origin of F_0. The correct translation is this: the left-hand side of (26) contains the Plücker coordinates of the spatial velocity of the end effector expressed in a coordinate system that is parallel to absolute coordinates but has its origin at the particular point in the end effector to which $\boldsymbol{v}$ refers.

Jacobians can also map forces. If a robot makes contact with its environment through body b, and the environment responds by exerting a force of $\boldsymbol{f}_{\text{e}}$ on body b, then the effect of that force on the robot is equivalent to a joint-space force of $\boldsymbol{\tau}_{\text{e}}$ given by

$$\boldsymbol{\tau}_{\text{e}} = \boldsymbol{J}_b^{\text{T}}\boldsymbol{f}_{\text{e}}. \tag{27}$$

The robot's control system can resist this force by adding $-\boldsymbol{\tau}_{\text{e}}$ to its joint-force command. This works because $\boldsymbol{f}_{\text{e}}$ acting on body b has the same effect on the robot as $\boldsymbol{\tau}_{\text{e}}$ acting at the joints, and the two forces $+\boldsymbol{\tau}_{\text{e}}$ and $-\boldsymbol{\tau}_{\text{e}}$ cancel. In applications like this, it is important to be clear and unambiguous about whether a force is being exerted by the environment on the robot or the other way around. In this example, the environment exerts a force of $\boldsymbol{f}_{\text{e}}$ on the robot, and the robot exerts a force of $-\boldsymbol{f}_{\text{e}}$ on the environment.

Acceleration

Equation (10) provides us with a recursive formula for calculating body accelerations. A nonrecursive formula can be obtained by differentiating (18):

$$\boldsymbol{a}_b = \sum_{i \in \kappa(b)} (\boldsymbol{s}_i\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{s}}_i\dot{\boldsymbol{q}}_i). \tag{28}$$

If we assume that $\dot{\boldsymbol{s}}_i = \boldsymbol{v}_i \times \boldsymbol{s}_i$, then this equation can be further expanded to

$$\begin{aligned} \boldsymbol{a}_b &= \sum_{i \in \kappa(b)} \boldsymbol{s}_i\ddot{\boldsymbol{q}}_i + \sum_{i \in \kappa(b)} \Big(\sum_{j \in \kappa(i)} \boldsymbol{s}_j\dot{\boldsymbol{q}}_j \Big) \times \boldsymbol{s}_i\dot{\boldsymbol{q}}_i \\ &= \sum_{i \in \kappa(b)} \boldsymbol{s}_i\ddot{\boldsymbol{q}}_i + \sum_{i \in \kappa(b)} \sum_{j \in \kappa(i)} \boldsymbol{s}_j\dot{\boldsymbol{q}}_j \times \boldsymbol{s}_i\dot{\boldsymbol{q}}_i. \end{aligned} \tag{29}$$

It is sometimes useful to define a velocity-product acceleration, $\boldsymbol{a}_b^{\text{vp}}$, equal to the velocity terms on the right-hand side:

$$\boldsymbol{a}_b^{\text{vp}} = \sum_{i \in \kappa(b)} \sum_{j \in \kappa(i)} \boldsymbol{s}_j\dot{\boldsymbol{q}}_j \times \boldsymbol{s}_i\dot{\boldsymbol{q}}_i. \tag{30}$$

In dynamics applications, this quantity might also include the fictitious acceleration that simulates gravity. Velocity-product accelerations can be calculated efficiently by the recursive formula

$$\boldsymbol{a}_i^{\text{vp}} = \boldsymbol{a}_{\lambda(i)}^{\text{vp}} + \boldsymbol{v}_i \times \boldsymbol{s}_i\dot{\boldsymbol{q}}_i, \qquad (\boldsymbol{a}_0^{\text{vp}} = \boldsymbol{0} \text{ or } -\boldsymbol{a}_g) \tag{31}$$

which is obtained from (10) by setting $\ddot{\boldsymbol{q}}_i = 0$.

Another equation for the acceleration of body b can be obtained by differentiating (17):

$$\boldsymbol{a}_b = \boldsymbol{J}_b\,\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_b\,\dot{\boldsymbol{q}}. \tag{32}$$

At first sight, the term $\dot{\boldsymbol{J}}_b\,\dot{\boldsymbol{q}}$ looks like it might be difficult to calculate. However, a moments thought reveals that $\dot{\boldsymbol{J}}_b\,\dot{\boldsymbol{q}} = \boldsymbol{a}_b^{\text{vp}}$; so this equation can be written as

$$\boldsymbol{a}_b = \boldsymbol{J}_b\,\ddot{\boldsymbol{q}} + \boldsymbol{a}_b^{\text{vp}}. \tag{33}$$

If a two-handed robot has rigidly grasped a single object with both hands, then the kinematic acceleration constraint on that robot is $\boldsymbol{a}_l = \boldsymbol{a}_r$, where l and r are the body numbers of the left and right hands, respectively. Using (33), we can express this as a constraint on the joint accelerations as follows:

$$(\boldsymbol{J}_l - \boldsymbol{J}_r)\ddot{\boldsymbol{q}} = \boldsymbol{a}_r^{\mathrm{vp}} - \boldsymbol{a}_l^{\mathrm{vp}}. \tag{34}$$

If a robot mechanism contains kinematic loops, then the loop-closure constraints can be formulated in a similar manner to (34). However, if one wishes to simulate such a mechanism, then the acceleration constraints must be stabilized to prevent accumulation of position and velocity errors [3], [8].

Dynamics

We have already examined inverse dynamics in some detail, so let us now look at forward dynamics, which is the problem of calculating a robot's acceleration response to applied forces. In analogy with (1), we can express the forward-dynamics problem mathematically as

$$\ddot{\boldsymbol{q}} = \mathrm{FD}(\mathrm{model}, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}), \tag{35}$$

where the objective is to calculate the numeric value of the function FD from the numeric values of its arguments. There are many ways to do this; however, we shall consider only the two most efficient ways.

Composite Rigid-Body Algorithm

The joint-space equation of motion for a kinematic tree can be expressed in the following canonical form:

$$\boldsymbol{\tau} = \boldsymbol{H}\ddot{\boldsymbol{q}} + \boldsymbol{C}, \tag{36}$$

where $\boldsymbol{H}$ is the joint-space inertia matrix, and $\boldsymbol{C}$ is a vector containing the Coriolis, centrifugal, and gravitational terms. If we can calculate $\boldsymbol{H}$ and $\boldsymbol{C}$, then we can solve the forward-dynamics problem simply by solving (36) for $\ddot{\boldsymbol{q}}$. We already know how to calculate $\boldsymbol{C}$, because

$$\boldsymbol{C} = \mathrm{ID}(\mathrm{model}, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{0}) \tag{37}$$

[cf. (1)], so the only remaining problem is how to calculate $\boldsymbol{H}$. The best algorithm for this job is called the composite-rigid-body algorithm, which we shall now derive.

One of the defining properties of the joint-space inertia matrix is that the kinetic energy of a robot mechanism is given by

$$T = \frac{1}{2}\dot{\boldsymbol{q}}^{\mathrm{T}}\boldsymbol{H}\dot{\boldsymbol{q}} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij}\dot{\boldsymbol{q}}_i\dot{\boldsymbol{q}}_j. \tag{38}$$

However, the kinetic energy is also the sum of the kinetic energies of the individual bodies, which can be written in spatial-vector notation as

$$T = \sum_{k=1}^{N}\frac{1}{2}\boldsymbol{v}_k^{\mathrm{T}}\boldsymbol{I}_k\boldsymbol{v}_k. \tag{39}$$

Substituting for $\boldsymbol{v}_k$ using (18) gives

$$\begin{aligned} T &= \frac{1}{2}\sum_{k=1}^{N}\Big(\sum_{i\in\kappa(k)}\boldsymbol{s}_i\dot{\boldsymbol{q}}_i\Big)^{\mathrm{T}}\boldsymbol{I}_k\Big(\sum_{j\in\kappa(k)}\boldsymbol{s}_j\dot{\boldsymbol{q}}_j\Big) \\ &= \frac{1}{2}\sum_{k=1}^{N}\sum_{i\in\kappa(k)}\sum_{j\in\kappa(k)}\boldsymbol{s}_i^{\mathrm{T}}\boldsymbol{I}_k\boldsymbol{s}_j\dot{\boldsymbol{q}}_i\dot{\boldsymbol{q}}_j\,. \end{aligned} \tag{40}$$

Now, the expression on the right-hand side is a sum over all i, j, k triples in which both i and j are elements of $\kappa(k)$. This same set of triples can also be described as the set of all i, j, k triples in which $k \in \nu(i)$ and $k \in \nu(j)$. So we can rewrite (40) as follows:

$$T = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k\in\nu(i)\cap\nu(j)}\boldsymbol{s}_i^{\mathrm{T}}\boldsymbol{I}_k\boldsymbol{s}_j\dot{\boldsymbol{q}}_i\dot{\boldsymbol{q}}_j\,. \tag{41}$$

On comparing (41) with (38), if we take into account that both equations must be true for all $\dot{\boldsymbol{q}}$, and also that $n = N$ for the class of robots we are considering, then it follows that

$$H_{ij} = \sum_{k\in\nu(i)\cap\nu(j)}\boldsymbol{s}_i^{\mathrm{T}}\boldsymbol{I}_k\boldsymbol{s}_j. \tag{42}$$

There are two simplifications we can make to this equation. The first is that

$$\nu(i)\cap\nu(j) = \begin{cases} \nu(i) & \text{if } i \in \nu(j) \\ \nu(j) & \text{if } j \in \nu(i) \\ \emptyset & \text{otherwise.} \end{cases} \tag{43}$$

The second is that we can define a composite rigid-body inertia, $\boldsymbol{I}_i^c$, which is the inertia of all the bodies in the subtree $\nu(i)$ treated as a single composite rigid body. This inertia is given by

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

but the best way to calculate it is via the recursive formula

$$\boldsymbol{I}_i^c = \boldsymbol{I}_i + \sum_{j\in\mu(i)}\boldsymbol{I}_j^c. \tag{44}$$

With these two simplifications, we can rewrite (42) as

$$H_{ij} = \begin{cases} \boldsymbol{s}_i^T\boldsymbol{I}_i^c\boldsymbol{s}_j & \text{if } i \in \nu(j) \\ \boldsymbol{s}_i^T\boldsymbol{I}_j^c\boldsymbol{s}_j & \text{if } j \in \nu(i) \\ 0 & \text{otherwise.} \end{cases} \tag{45}$$

Equations (44) and (45) together define the composite-rigid-body algorithm.

Before moving on, let us review what we have just achieved. Using only a small amount of algebra, we have obtained a very compact expression for H_{ij} in (42) and a compact description of the composite-rigid-body algorithm in (44) and (45). Along the way, we have not had to worry about whether joint i is revolute, prismatic, or helical and write different equations for each case; nor have we written separate expressions for the linear and angular components of kinetic energy; nor have we defined a point in each body, expressed equations at that point, and transferred them from one point to another; and nor have we written equations to calculate the center of mass of a composite body or use the

parallel-axes theorem to calculate a rotational inertia about a new center of mass. In short, we have benefited considerably from the use of spatial-vector notation. Readers may wish to compare this derivation with the original 3-D-vector derivation in [10], bearing in mind that the original applied only to unbranched chains with revolute and prismatic joints.

Equations (44) and (45) provide us with a basic mathematical description of the algorithm. If we want to implement it on a computer, then we must first decide what coordinate systems to use. The best choice, for all but the largest rigid-body systems, is to use link coordinates. We can express the algorithm in link coordinates as follows:

$$\boldsymbol{I}_i^c = \boldsymbol{I}_i + \sum_{j \in \mu(i)} {}^i\boldsymbol{X}_j^* \boldsymbol{I}_j^{c\,j}\boldsymbol{X}_i, \tag{46}$$

$${}^{\lambda(j)}\boldsymbol{f}_i = {}^{\lambda(j)}\boldsymbol{X}_j^{*\,j}\boldsymbol{f}_i, \qquad ({}^i\boldsymbol{f}_i = \boldsymbol{I}_i^c \boldsymbol{s}_i) \tag{47}$$

$$H_{ij} = \begin{cases} {}^j\boldsymbol{f}_i^{\mathrm{T}}\boldsymbol{s}_j & \text{if } i \in v(j) \\ H_{ji} & \text{if } j \in v(i) \\ 0 & \text{otherwise.} \end{cases} \tag{48}$$

These equations show explicitly where the coordinate transforms are performed. Note that the quantities $\boldsymbol{s}_i$, $\boldsymbol{I}_i$, and $\boldsymbol{I}_i^c$ appearing in these equations are expressed in link-i coordinates, whereas the same symbols in previous equations were tacitly assumed to be expressed in a single unidentified common coordinate system.

The symbol ${}^j\boldsymbol{f}_i$ in (47) is the spatial force, expressed in link-j coordinates, that imparts an acceleration of $\boldsymbol{s}_i$ (i.e., a unit acceleration about the axis of joint i) to a composite rigid body comprising all of the bodies in subtree $v(i)$. The algorithm requires the calculation of ${}^i\boldsymbol{f}_i$ for every i and ${}^j\boldsymbol{f}_i$ for every $j \in \kappa(i) \backslash \{i\}$.

The pseudocode for this algorithm is shown in Figure 7. It employs the same tactic as was used in the recursive Newton–Euler algorithm to convert the summation over $\mu(i)$ in (46) into code that uses only λ: each variable $\boldsymbol{I}_i^c$ is initialized to $\boldsymbol{I}_i$ in the first loop, and then each $\boldsymbol{I}_i^c$ is added to its parent in the second loop. By the time $\boldsymbol{I}_i^c$ is used on line 9 to calculate ${}^i\boldsymbol{f}_i$, which is stored in the local variable $\boldsymbol{f}$, it has already received the contributions from all of its children and, therefore, has the correct final value.

The statement $\boldsymbol{H} = \boldsymbol{0}$ on line 1 is necessary, because the remaining code will initialize only the nonzero elements of $\boldsymbol{H}$. Certain elements of $\boldsymbol{H}$ will automatically be zero, simply because of the connectivity of the robot. This phenomenon is called branch-induced sparsity, and it arises from the third case in (48). This phenomenon is discussed in detail in [2] and [3] along with methods to greatly accelerate the solution of (36) by exploiting the sparsity.

Articulated-Body Algorithm

The articulated-body algorithm is an $O(N)$ algorithm that solves the forward-dynamics problem by the following strategy: at the outset, we know neither the acceleration of body i nor the force transmitted across joint i; however, we do know that the relationship between them must be linear. It must therefore be possible to express the relationship between these two vectors in an equation of the form

$$\boldsymbol{f}_i = \boldsymbol{I}_i^A \boldsymbol{a}_i + \boldsymbol{p}_i^A. \tag{49}$$

The two coefficients in this equation, $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$, are called the articulated-body inertia and bias force, respectively, of body i; they describe the acceleration response of body i to an applied spatial force, taking into account the influence of all the other bodies in the subtree $v(i)$. These coefficients have two special properties that form the basis of the articulated-body algorithm. The special properties are that

1) they can be calculated recursively from the tips of the tree to the base, and
2) once they have been calculated, they allow the accelerations of the bodies and joints to be calculated recursively from the base to the tips.

The calculation of $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ closely resembles the two-body example presented in Part 1 [6]. Referring to Figure 8, we initially assume that body i has only one child, which is labeled body j. The relevant equations for body i are then

$$\boldsymbol{f}_i - \boldsymbol{f}_j = \boldsymbol{I}_i \boldsymbol{a}_i + \boldsymbol{p}_i, \tag{50}$$

$$\boldsymbol{f}_j = \boldsymbol{I}_j^A \boldsymbol{a}_j + \boldsymbol{p}_j^A, \tag{51}$$

$$\boldsymbol{a}_j = \boldsymbol{a}_i + \boldsymbol{c}_j + \boldsymbol{s}_j \ddot{\boldsymbol{q}}_j \tag{52}$$

```
1  H = 0
2  for i = 1 to N do
3     I_i^c = I_i
4  end
5  for i = N to 1 do
6     If λ(i) ≠ 0 then
7        I_λ(i)^c = I_λ(i)^c + λ(i)X_i^* I_i^c iX_λ(i)
8     end
9     f = I_i^c s_i
10    H_ii = f^T s_i
11    j = i
12    While λ(j) ≠ 0 do
13       f = λ(j)X_j^* f
14       j = λ(j)
15       H_ij = f^T s_j
16       H_ji = H_ij
17    end
18 end
```

Figure 7. *The composite-rigid-body algorithm.*

Spatial vectors can be used both for positional kinematics and for instantaneous kinematics.

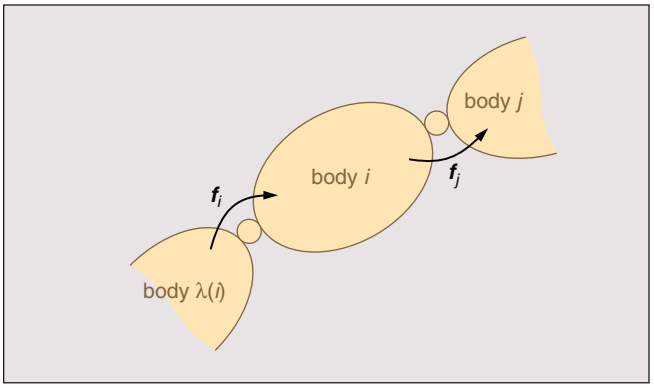


Figure 8. *Calculating articulated-body inertias.*

and

$$\tau_j = \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{f}_j, \tag{53}$$

where

$$\boldsymbol{p}_i = \boldsymbol{v}_i \times^* \boldsymbol{I}_i \boldsymbol{v}_i \tag{54}$$

and

$$\boldsymbol{c}_j = \boldsymbol{v}_j \times \boldsymbol{s}_j \dot{\boldsymbol{q}}_j. \tag{55}$$

Equation (50) is the equation of motion for body i, which we have written in terms of the rigid-body inertia and bias force, $\boldsymbol{I}_i$ and $\boldsymbol{p}_i$, to make it obvious that the rigid-body and articulated-body equations of motion have the same algebraic form. Equation (51) is the articulated-body equation of motion for body j, which describes the relationship between $\boldsymbol{f}_j$ and $\boldsymbol{a}_j$, taking into account the dynamics of every body and joint in the subtree $\nu(j)$. We assume that $\boldsymbol{I}_j^A$ and $\boldsymbol{p}_j^A$ are known. Equations (52) and (53) are the acceleration and force constraint equations for joint j.

The objective is to solve (50)–(53) to obtain an equation having the same form as (49), which is an equation involving only the two unknowns $\boldsymbol{f}_i$ and $\boldsymbol{a}_i$. To obtain this result, the first step is to solve (51)–(53) for the unknown acceleration $\ddot{\boldsymbol{q}}_j$. We can do this by substituting (51) and (52) into (53) as follows:

$$\begin{aligned}\tau_j &= \boldsymbol{s}_j^{\mathrm{T}} (\boldsymbol{I}_j^A \boldsymbol{a}_j + \boldsymbol{p}_j^A) \\ &= \boldsymbol{s}_j^{\mathrm{T}} (\boldsymbol{I}_j^A (\boldsymbol{a}_i + \boldsymbol{c}_j + \boldsymbol{s}_j \ddot{\boldsymbol{q}}_j) + \boldsymbol{p}_j^A),\end{aligned}$$

which yields the following equation for $\ddot{\boldsymbol{q}}_j$:

$$\ddot{\boldsymbol{q}}_j = \frac{\tau_j - \boldsymbol{s}_j^{\mathrm{T}} (\boldsymbol{I}_j^A (\boldsymbol{a}_i + \boldsymbol{c}_j) + \boldsymbol{p}_j^A)}{\boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{s}_j}. \tag{56}$$

At this point, we can simplify (56) a little by introducing the quantity

$$u_j = \tau_j - \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{p}_j^A. \tag{57}$$

Substituting (57) in (56) gives

$$\ddot{\boldsymbol{q}}_j = \frac{u_j - \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A (\boldsymbol{a}_i + \boldsymbol{c}_j)}{\boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{s}_j}. \tag{58}$$

Having found an expression for $\ddot{\boldsymbol{q}}_j$, the remainder of the problem is solved by substituting (51), (52), and (58) back into (50) as follows:

$$\begin{aligned}\boldsymbol{f}_i &= \boldsymbol{I}_i \boldsymbol{a}_i + \boldsymbol{p}_i + \boldsymbol{f}_j \\ &= \boldsymbol{I}_i \boldsymbol{a}_i + \boldsymbol{I}_j^A \boldsymbol{a}_j + \boldsymbol{p}_i + \boldsymbol{p}_j^A \\ &= \boldsymbol{I}_i \boldsymbol{a}_i + \boldsymbol{I}_j^A (\boldsymbol{a}_i + \boldsymbol{c}_j + \boldsymbol{s}_j \ddot{\boldsymbol{q}}_j) + \boldsymbol{p}_i + \boldsymbol{p}_j^A \\ &= \boldsymbol{I}_i \boldsymbol{a}_i + \boldsymbol{I}_j^A \left(\boldsymbol{a}_i + \boldsymbol{c}_j + \frac{\boldsymbol{s}_j (u_j - \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A (\boldsymbol{a}_i + \boldsymbol{c}_j))}{\boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{s}_j} \right) + \boldsymbol{p}_i + \boldsymbol{p}_j^A.\end{aligned} \tag{59}$$

On comparing this equation with (49), we get the following expressions for $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$:

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \boldsymbol{I}_j^a \tag{60}$$

and

$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \boldsymbol{p}_j^a, \tag{61}$$

where

$$\boldsymbol{I}_j^a = \boldsymbol{I}_j^A - \frac{\boldsymbol{I}_j^A \boldsymbol{s}_j \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A}{\boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{s}_j} \tag{62}$$

and

$$\boldsymbol{p}_j^a = \boldsymbol{I}_j^a \boldsymbol{c}_j + \frac{\boldsymbol{I}_j^A \boldsymbol{s}_j u_j}{\boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{s}_j} + \boldsymbol{p}_j^A. \tag{63}$$

The next step is to drop the assumption that body i has only one child. If body i has multiple children, then it is possible to process them one at a time using the above procedure. This works because spatial inertias are additive and rigid-body and articulated-body equations have the same algebraic form. In processing the rth child, we simply replace $\boldsymbol{I}_i$ and $\boldsymbol{p}_i$ in (50) with the articulated-body inertia and bias force that account for the first $r-1$ children. The end result is the following pair of equations, which replace (60) and (61):

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \sum_{j \in \mu(i)} \boldsymbol{I}_j^a \tag{64}$$

and

$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \sum_{j \in \mu(i)} \boldsymbol{p}_j^a. \tag{65}$$

The definitions of $\boldsymbol{I}_j^a$ and $\boldsymbol{p}_j^a$ remain unchanged.

The final step is to calculate the accelerations. We already have the necessary equations, being (52) and (58), but the calculation can be performed slightly more efficiently as follows:

$$\boldsymbol{a}_i' = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i, \qquad (\boldsymbol{a}_0 = -\boldsymbol{a}_g) \tag{66}$$

$$\ddot{\boldsymbol{q}}_i = \frac{u_i - \boldsymbol{s}_i^{\mathrm{T}} \boldsymbol{I}_i^A \boldsymbol{a}_i'}{\boldsymbol{s}_i^{\mathrm{T}} \boldsymbol{I}_i^A \boldsymbol{s}_i}, \tag{67}$$

Pass 1

$$\boldsymbol{v}_0 = \boldsymbol{0}$$
$$\boldsymbol{v}_i = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{v}_{\lambda(i)} + \boldsymbol{s}_i\dot{q}_i$$
$$\boldsymbol{c}_i = \boldsymbol{v}_i \times \boldsymbol{s}_i\dot{q}_i$$
$$\boldsymbol{p}_i = \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i$$

Pass 2

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \sum_{j\in\mu(i)} {}^i\boldsymbol{X}_j^*\,\boldsymbol{I}_j^a\,{}^j\boldsymbol{X}_i$$
$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \sum_{j\in\mu(i)} {}^i\boldsymbol{X}_j^*\,\boldsymbol{p}_j^a$$
$$\boldsymbol{h}_i = \boldsymbol{I}_i^A\boldsymbol{s}_i$$
$$d_i = \boldsymbol{s}_i^{\mathrm{T}}\boldsymbol{h}_i$$
$$u_i = \tau_i - \boldsymbol{s}_i^{\mathrm{T}}\boldsymbol{p}_i^A$$
$$\boldsymbol{I}_i^a = \boldsymbol{I}_i^A - \boldsymbol{h}_i\boldsymbol{h}_i^{\mathrm{T}}/d_i$$
$$\boldsymbol{p}_i^a = \boldsymbol{p}_i^A + \boldsymbol{I}_i^a\boldsymbol{c}_i + \boldsymbol{h}_i u_i/d_i$$

Pass 3

$$\boldsymbol{a}_0 = -\boldsymbol{a}_g$$
$$\boldsymbol{a}_i' = {}^i\boldsymbol{X}_{\lambda(i)}\,\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i$$
$$\ddot{q}_i = (u_i - \boldsymbol{h}_i^{\mathrm{T}}\boldsymbol{a}_i')/d_i$$
$$\boldsymbol{a}_i = \boldsymbol{a}_i' + \boldsymbol{s}_i\ddot{q}_i$$

Figure 9. *Equations of the articulated-body algorithm.*

$$\boldsymbol{a}_i = \boldsymbol{a}_i' + \boldsymbol{s}_i\ddot{\boldsymbol{q}}_i. \tag{68}$$

The complete equations for the articulated-body algorithm, expressed in link coordinates, are shown in Figure 9. The algorithm makes a total of three passes through the tree: an outward pass (base to tips) to calculate the velocity terms $\boldsymbol{c}_i$ and $\boldsymbol{p}_i$, an inward pass (tips to base) to calculate $\boldsymbol{I}_i^A$, $\boldsymbol{p}_i^A$, and related terms, and a second outward pass to calculate the accelerations. A more detailed description of this algorithm can be found in [3], and source code can be obtained from [5].

Deriving the articulated-body algorithm is an example of a dynamics problem that would be forbiddingly difficult to attempt using 3-D vectors. Whereas other algorithms described in this tutorial were invented using 3-D vectors, the articulated-body algorithm was invented using spatial vectors. In fact, spatial vectors themselves were invented as a side effect of trying to invent the articulated-body algorithm. This algorithm, and many others that have followed it, make an important statement about spatial vectors: they are a tool for discovery; they let you go beyond what is feasible to attempt using 3-D vectors.

Conclusion

This tutorial has demonstrated the use of spatial vectors in a variety of kinematics and dynamics calculations. A model-based approach was adopted in which a description of the robot mechanism is stored in a model data structure, and the various equations and algorithms are designed to use this data in their calculations. The class of robots considered was the class of general kinematic trees having revolute, prismatic, and helical joints; the idea being to show how easily spatial vectors cope with a high degree of generality. The focus of this tutorial has ranged from mathematics to computer code to make the point that spatial vectors are both an analytical tool and a computational tool. In both this tutorial and Part 1, the emphasis has been on human productivity: fewer equations, simpler problem solving, and shorter code. If your application also needs high computational efficiency, then see Appendix A of [3]. A mastery of spatial vectors gives you a different perspective on rigid-body kinematics and dynamics and is a worthwhile skill for a roboticist.

Keywords

Dynamics, kinematics, spatial vectors, dynamics algorithms, software.

References

[1] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

[2] R. Featherstone, "Efficient factorization of the joint space inertia matrix for branched kinematic trees," *Int. J. Robot. Res.*, vol. 24, no. 6, pp. 487–500, 2005.

[3] R. Featherstone, *Rigid Body Dynamics Algorithms*. New York: Springer-Verlag, 2008.

[4] R. Featherstone and D. E. Orin, "Dynamics," *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin: Springer-Verlag, 2008, pp. 35–65.

[5] R. Featherstone. (2010). "Spatial vector algebra," [Online]. Available: http://users.cecs.anu.edu.au/~roy/spatial/

[6] R. Featherstone, "A beginner's guide to 6-D vectors (part 1)," *IEEE Robot. Automat. Mag.*, vol. 17, no. 3, pp. 83–94, 2010.

[7] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *Trans. ASME J. Dyn. Syst., Meas. Control*, vol. 102, no. 2, pp. 69–76, 1980.

[8] R. E. Roberson and R. Schwertassek, *Dynamics of Multibody Systems*. Berlin: Springer-Verlag, 1988.

[9] K. Waldron and J. Schmiedeler, "Kinematics," *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin: Springer-Verlag, 2008, pp. 9–33.

[10] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Trans. ASME J. Dyn. Syst., Meas. Control*, vol. 104, no. 3, pp. 205–211, 1982.

Roy Featherstone received his Ph.D. degree from Edinburgh University in 1984. He spent approximately seven years working in industry before moving to Oxford University in 1992 to take up an EPSRC Advanced Research Fellowship. He currently works for the Australian National University, which he joined in 2001. His research interests include robot motion, in particular, the kinematics, dynamics, control and enabling technology of complex and energetic robot motion. He is also interested in dynamics algorithms and simulation. He is a Fellow of the IEEE.

Address for Correspondence: Roy Featherstone, School of Engineering, RSISE Building 115, The Australian National University, Canberra, ACT 0200, Australia. E-mail: Roy.Featherstone@anu.edu.au.