Computer Animation

Lesson 11 - Forward and Inverse Dynamics

Remi Ronfard, Nov 2019

11 Squash & Stretch

The most important principle is called *squash and stretch.* When an object is moved, the movement emphasizes any rigidity in the object. In real life, only the most rigid shapes (such as chairs, dishes and pans) remain so during motion. Anything composed of living flesh, no matter how bony, will show considerable movement in its shape during an action. For example, when a bent arm with swelling biceps straightens out, only the long sinews are apparent. A face, whether chewing, smiling, talking, or just showing a change of expression, is alive with changing shapes in the cheeks, the lips, and the eyes. [26]

The most important rule to squash and stretch is that, no matter how squashed or stretched out a particular object gets, its volume remains constant. If an object squashed down without its sides stretching, it would appear to shrink; if it stretched up without its sides squeezing in it would appear to grow. Consider the shape and volume of a half filled flour sack: when dropped on the floor, it squashed out to its fullest shape. If picked up by the top corners, it stretched out to its longest shape. It never changes volume. [26]

The standard animation test for all beginners is drawing a bouncing ball. The assignment is to represent the ball by a simple circle, and then have it drop, hit the ground, and bounce back into the air. A simple test, but it teaches the basic mechanics of animating a scene, introducing timing as well as squash and stretch. If the bottom drawing is flattened, it gives the appearance of bouncing. Elongating the drawings before and after the bounce increases the sense of speed, makes it easier to follow and gives more snap to the action. [26,3] (figure 2)

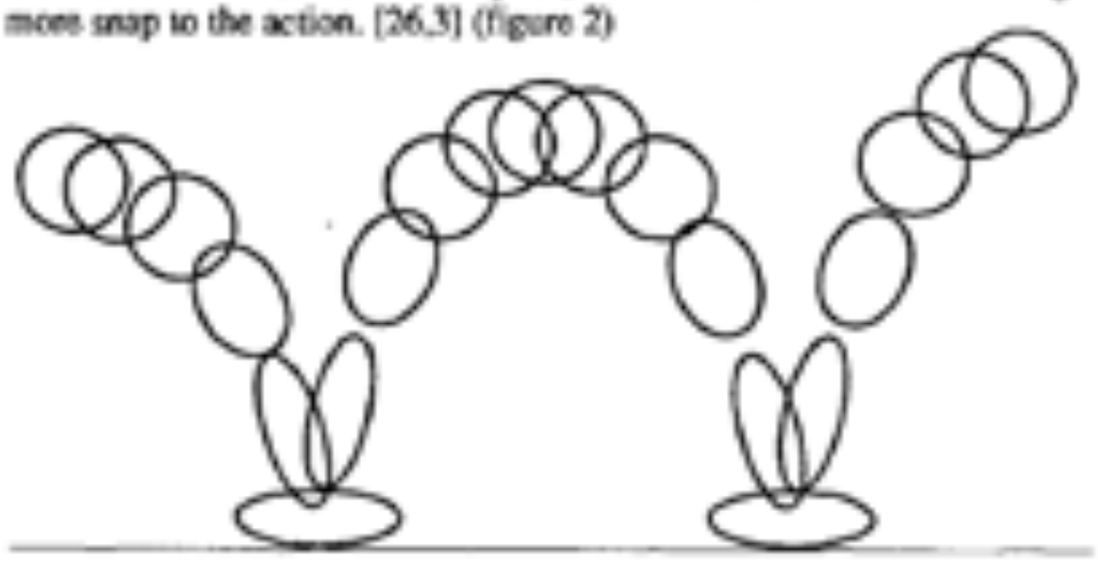


FIGURE 2. Squash & stretch in bouncing ball.

11 Squash & stretch

An object need not deform in order to squash and stretch. For instance, a hinged object like Luxo Jr. (from the film, *Luxo Jr.* [21]), squashes by folding over on itself, and stretches by extending out fully. (figure 3)

FIGURE 3. Squash & stretch in Luxo Jr.'s hop.

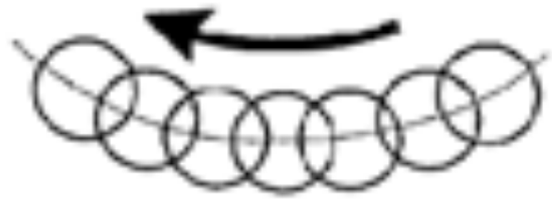


FIGURE 4a. In slow action, an object's position overlaps from frame to frame which gives the action a smooth appearance to the eye.

FIGURE 4b. Strobing occurs in a faster action when the object's positions do not overlap and the eye perceives seperate images.

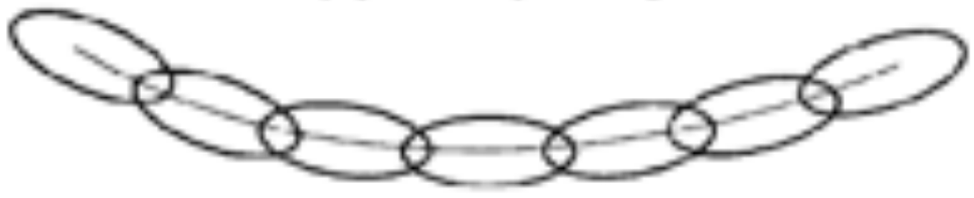


FIGURE 4c. Stretching the object so that it's positions overlap again will achieve the strobing effect.

1. Squash and Stretch

Physical animation of the human body

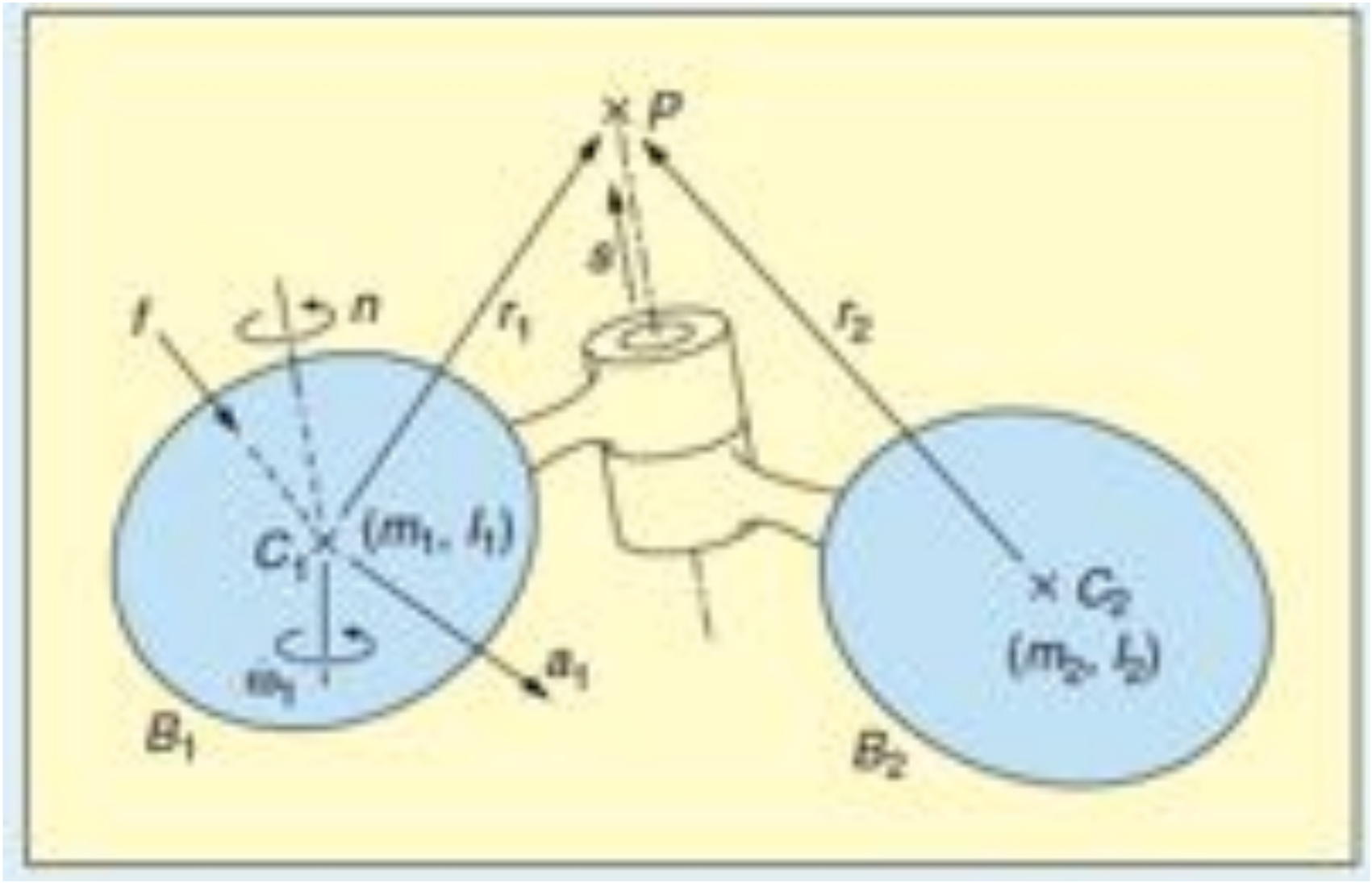
- Dynamics much more complex than kinematics
- 320 pairs = 640 "voluntary" skeletal muscles in the human body move the various parts of the body.

Forward and inverse dynamics

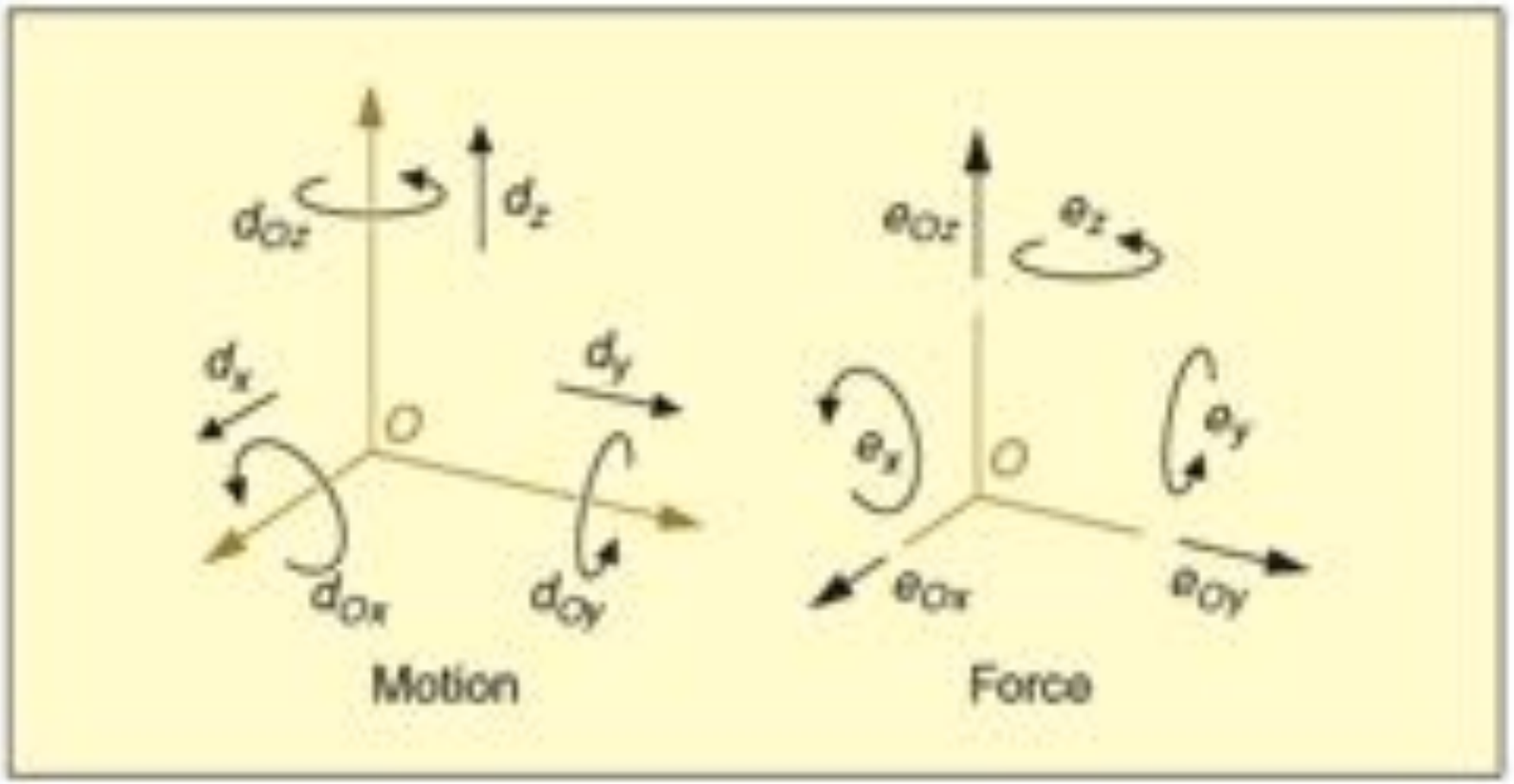
- Inverse dynamics (Newton-Euler)
  - from joint trajectories (positions, velocities, accelerations) to joint (muscle) forces
  - Useful for motion capture and control

- Forward dynamics (Featherstone)
- from joint (muscle) forces to joint trajectories (positions, velocities, accelerations)
  - Useful for animation and simulation

Spatial vectors



Spatial vectors



Spatial vectors

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

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

Spatial vectors

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 I and a velocity of v, then its momentum, $h \in \mathsf{F}^6$, is

$$h = Iv. \tag{23}$$

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

where m is the body's mass, c is a 3-D vector locating the body's center of mass, and $\bar{I}_c$ 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 c, and six in $\bar{I}_c$. 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.

Inverse dynamics : Newton-Euler

- From accelerations to forces
- Combine Newton and Euler equations of motion
  - Linear acceleration
  - Angular acceleration

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

- As a result of direct forces on the joint $\mathbf{f}_i$
- External forces $\mathbf{f}_i^x$
- And inertia matrix $\mathbf{I}_i$

Inverse dynamics : Newton-Euler

The inverse dynamics of a general kinematic tree can be calculated in three steps as follows:

1. Calculate the velocity and acceleration of each body in the tree.

2. Calculate the forces required to produce these accelerations.

3. Calculate the forces transmitted across the joints from the forces acting on the bodies.

Inverse dynamics : Newton-Euler

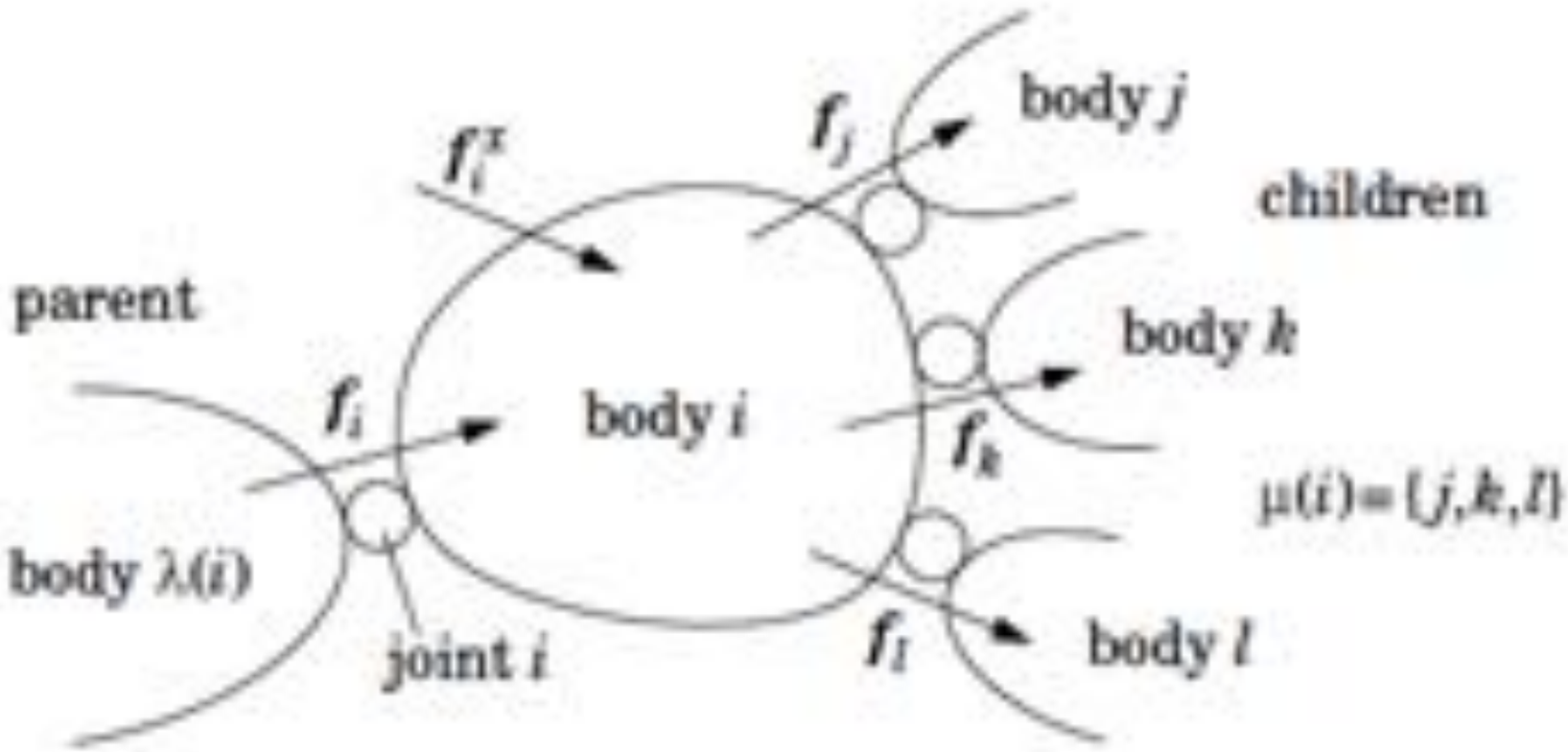


Figure 5.1: Forces acting on body i

Inverse dynamics : Newton-Euler

Step 1: Let v_i and a_i be the velocity and acceleration of body i. The velocity of any body in a kinematic tree can be defined recursively as the sum of the velocity of its parent and the velocity across the joint connecting it to its parent. Thus,

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{S}_i \dot{\boldsymbol{q}}_i \,. \qquad (\boldsymbol{v}_0 = \boldsymbol{0}) \tag{5.7}$$

(Initial values are shown in brackets.) The recurrence relation for accelerations is obtained by differentiating this equation, resulting in

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i \,. \qquad (\boldsymbol{a}_0 = \boldsymbol{0}) \tag{5.8}$$

Successive iterations of these two formulae, with i ranging from 1 to N_B, will calculate the velocity and acceleration of every body in the tree.

Inverse dynamics : Newton-Euler

Step 2: Let $\boldsymbol{f}_i^B$ be the net force acting on body i. This force is related to the acceleration of body i by the equation of motion

$$\boldsymbol{f}_i^B = \boldsymbol{I}_i\,\boldsymbol{a}_i + \boldsymbol{v}_i \times^* \boldsymbol{I}_i\,\boldsymbol{v}_i\,. \tag{5.9}$$

Inverse dynamics : Newton-Euler

Step 3: Referring to Figure 5.1, let $\boldsymbol{f}_i$ be the force transmitted from body $\lambda(i)$ to body i across joint i, and let $\boldsymbol{f}_i^x$ be the external force (if any) acting on body i. External forces can model a variety of environmental influences: force fields, physical contacts, and so on. They are regarded as inputs to the algorithm; that is, their values are assumed to be known. The net force on body i is then

$$\boldsymbol{f}_i^n = \boldsymbol{f}_i + \boldsymbol{f}_i^x - \sum_{j \in \mu(i)} \boldsymbol{f}_j \,,$$

which can be rearranged to give a recurrence relation for the joint forces:

$$\boldsymbol{f}_i = \boldsymbol{f}_i^n - \boldsymbol{f}_i^x + \sum_{j \in \mu(i)} \boldsymbol{f}_j \,. \tag{5.10}$$

Inverse dynamics : Newton-Euler

Having computed the spatial force across each joint, it remains only to calculate the generalized forces at the joints, which are given by

$$\tau_i = \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{f}_i \tag{5.11}$$

Inverse dynamics

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.

Inverse dynamics

```
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
```

Recursive Newton Euler

1 $v_0 = 0$
2 $a_0 = -a_g$
3 **for** $i = 1$ **to** N **do**
4 $[X_J, S_i] = \text{jcalc}(h_i, q_i)$
5 ${}^iX_{\lambda(i)} = X_J X_T(i)$
6 $v_i = {}^iX_{\lambda(i)} v_{\lambda(i)} + S_i \dot{q}_i$
7 $a_i = {}^iX_{\lambda(i)} a_{\lambda(i)} + S_i \ddot{q}_i + v_i \times S_i \dot{q}_i$
8 $f_i = I_i a_i + v_i \times^* I_i v_i$
9 **end**
10 **for** $i = N$ **to** 1 **do**
11 $\tau_i = S_i^T f_i$
12 **If** $\lambda(i) \neq 0$ **then**
13 $f_{\lambda(i)} = f_{\lambda(i)} + {}^{\lambda(i)}X_i^* f_i$
14 **end**
15 **end**

Forward 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

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

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.

Forward dynamics

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.

Composite rigid body

```
1  M = 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
```

Forward dynamics: Featherstone

The articulated-body algorithm calculates the forward dynamics of a kinematic tree in $O(N_B)$ arithmetic operations, in exactly the manner outlined above. It makes three passes over the tree, as follows: an outward pass (root to leaves) to calculate velocity and bias terms; an inward pass to calculate articulated-body inertias and bias forces; and a second outward pass to calculate the accelerations.

Roy Featherstone. Rigid Body Dynamics Algorithms. Spinger, 2008.

Articulated body inertia

Articulated-body inertia is the inertia that a body appears to have when it is part of a rigid-body system. Consider the rigid body B shown in diagram (a) of Figure 7.1. The relationship between the applied force, $\boldsymbol{f}$, and the resulting acceleration, $\boldsymbol{a}$, is given by the body's equation of motion:

$$\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p}\,. \tag{7.1}$$

The coefficients $\boldsymbol{I}$ and $\boldsymbol{p}$ in this equation are the rigid-body inertia and bias force, respectively, of body B. Now consider the two-body system shown in diagram (b), in which a second body, B', has been connected to B via a joint. The effect of this second body is to alter the acceleration response of B, so that the relationship between $\boldsymbol{f}$ and $\boldsymbol{a}$ is now given by

$$\boldsymbol{f} = \boldsymbol{I}^{\boldsymbol{A}}\,\boldsymbol{a} + \boldsymbol{p}^{\boldsymbol{A}}\,. \tag{7.2}$$

Articulated body inertia

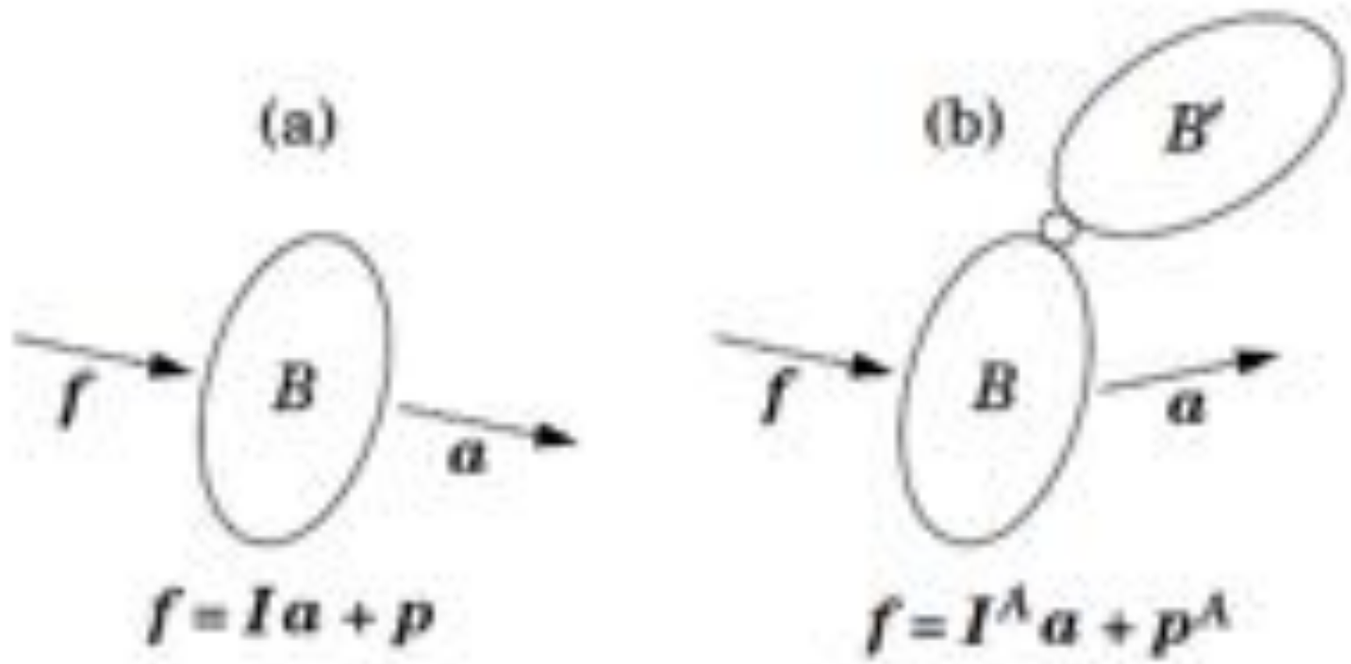


Figure 7.1: Comparing a rigid body (a) with an articulated body (b)

Forward dynamics: Featherstone

From the base to the tip

Pass 1 The job of the first pass is to calculate the velocity-product accelerations, $\boldsymbol{c}_i$, and the rigid-body bias forces, $\boldsymbol{p}_i$, for use in later passes. Both are functions of the body velocities, $\boldsymbol{v}_i$, so it is necessary to calculate these as well. The equations of the first pass are:

$$\boldsymbol{v}_{Ji} = \boldsymbol{S}_i \dot{\boldsymbol{q}}_i , \tag{7.32}$$

$$\boldsymbol{c}_{Ji} = \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i , \tag{7.33}$$

$$\boldsymbol{v}_i = \boldsymbol{v}_{\lambda(i)} + \boldsymbol{v}_{Ji} , \qquad (\boldsymbol{v}_0 = \boldsymbol{0}) \tag{7.34}$$

$$\boldsymbol{c}_i = \boldsymbol{c}_{Ji} + \boldsymbol{v}_i \times \boldsymbol{v}_{Ji} , \tag{7.35}$$

$$\boldsymbol{p}_i = \boldsymbol{v}_i \times^* \boldsymbol{I}_i \boldsymbol{v}_i - \boldsymbol{f}_i^x . \tag{7.36}$$

Forward dynamics: Featherstone

From the tip to the base

Pass 2 The next step is to calculate the articulated-body inertias and bias forces, $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$. This is done using the assembly formulae in Eqs. 7.21 to 7.24, which are reproduced below.

$$\boldsymbol{I}_i^A = \boldsymbol{I}_i + \sum_{j \in \mu(i)} \boldsymbol{I}_j^a \tag{7.37}$$

$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \sum_{j \in \mu(i)} \boldsymbol{p}_j^a , \tag{7.38}$$

$$\boldsymbol{I}_j^a = \boldsymbol{I}_j^A - \boldsymbol{I}_j^A \boldsymbol{S}_j (\boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{S}_j)^{-1} \boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{I}_j^A \tag{7.39}$$

$$\boldsymbol{p}_j^a = \boldsymbol{p}_j^A + \boldsymbol{I}_j^a \boldsymbol{c}_j + \boldsymbol{I}_j^A \boldsymbol{S}_j (\boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{I}_j^A \boldsymbol{S}_j)^{-1} (\boldsymbol{\tau}_j - \boldsymbol{S}_j^{\mathrm{T}} \boldsymbol{p}_j^A) . \tag{7.40}$$

These calculations are performed for each value of i from N_B down to 1. This implies that $\boldsymbol{I}_j^a$ and $\boldsymbol{p}_j^a$ are not calculated for every j, but only for those values that satisfy $\lambda(j) \neq 0$. If body i has no children then $\boldsymbol{I}_i^A = \boldsymbol{I}_i$ and $\boldsymbol{p}_i^A = \boldsymbol{p}_i$.

Forward dynamics: Featherstone

From the base to the tip

Pass 3 The third pass calculates the accelerations using Eqs. 7.30 and 7.31, which we repeat here as

$$\ddot{\boldsymbol{q}}_i = (\boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^A \boldsymbol{S}_i)^{-1} (\boldsymbol{\tau}_i - \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{I}_i^A (\boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i) - \boldsymbol{S}_i^{\mathrm{T}} \boldsymbol{p}_i^A)) \tag{7.41}$$

$$\boldsymbol{a}_i = \boldsymbol{a}_{\lambda(i)} + \boldsymbol{c}_i + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i \,. \qquad (\boldsymbol{a}_0 = -\boldsymbol{a}_g) \tag{7.42}$$

By initializing the base acceleration to $-\boldsymbol{a}_g$, rather than $\boldsymbol{0}$, we can simulate the effect of a uniform gravitational field with a fictitious upward acceleration. This is more efficient than treating gravity as an external force.

Paper 11 - Space-time constraints