RedMax: Efficient & Flexible Approach for Articulated Dynamics; Supplemental Material

YING WANG, NICHOLAS J. WEIDNER, MARGARET A. BAXTER, YURA HWANG, Texas A&M University
DANNY M. KAUFMAN, Adobe Research
SHINJIRO SUEDA, Texas A&M University

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: Physical simulation, Rigid body dynamics, Constraints, Contact, Friction

ACM Reference Format:
Ying Wang, Nicholas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman, and Shinjiro Sueda. 2019. RedMax: Efficient & Flexible Approach for Articulated Dynamics; Supplemental Material. *ACM Trans. Graph.* 38, 4, Article 39 (July 2019), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 REFERENCE IMPLEMENTATION

We provide a reference implementation of the RedMax algorithm, written in object-oriented MATLAB (2018b). This code is not designed for performance but is rather designed for pedagogical purposes. Since it is object-oriented, porting to a more performant language should be straight-forward. To run the code, go to the directory containing `testRedMax.m` and type:

```
>> testRedMax(1,0)
```

This will show a swinging chain with alternating revolute/fixed joints (Fig. 1a). The first two arguments modify the integrator type and the scene ID:

(1) `itype`: Integrator type
  1: Use the recursive $O(n)$ algorithm [Kim and Pollard 2011; Kim 2012] with `ode45`.
  2: Use RedMax with `ode45`. This gives numerically the same solution as the recursive $O(n)$ algorithm.
  3: Use RedMax with linearly implicit Euler. This is the most feature-rich option.
(2) `sceneID`: There are many preset scenes. The first few are:
  0: Simple serial chain
  1: Different revolute axes
  2: Branching
  3: Spherical joint
  4: Loop
  5: ···

The full list of scenes is in `testRedMax.m`. The other arguments are used to control what gets displayed:

0730-0301/2019/7-ART39 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

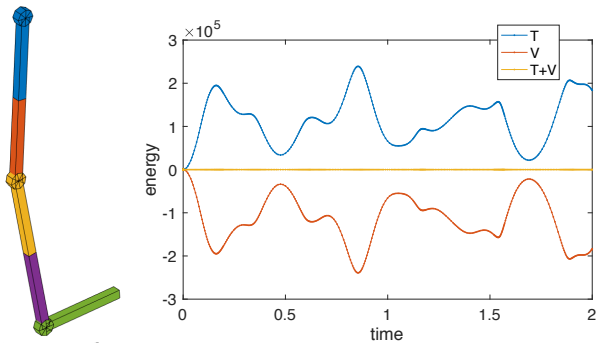

Fig. 1. Sample scene and its energy plot.

(3) `drawScene`: Whether to draw the scene. This can significantly speed up the program, since drawing in MATLAB is very slow.
(4) `plotH`: Whether to compute and plot the energy over time (Fig. 1b).

To run all the test cases, copy and paste the following into the command window:

```
clear; clc;
for itype = 1 : 3
  for sceneID = 0 : 33
    testRedMax(itype,sceneID,false,false);
  end
end
```

2 MAXIMAL COORDINATES

In this section and the next, we give a brief tutorial on rigid body dynamics. We first describe our maximal coordinate representation of rigid bodies, where 6 degrees of freedom (DOFs) are used for each body, and constraints are used to create joints between the bodies. Next, in §3, we describe our reduced coordinate representation of articulated rigid bodies, where the DOFs directly represent the joints between the rigid bodies.

In maximal coordinates, the configuration of a rigid body is represented by the usual 4×4 transformation matrix consisting of rotational and translational components:

$$ {}^0_i\mathsf{E} = \begin{pmatrix} {}^0_i\mathsf{R} & {}^0\boldsymbol{p} \\ 0 & 1 \end{pmatrix}. \tag{1} $$

The leading subscripts and superscripts indicate that the coordinates of rigid body (or frame) 'i' are defined with respect to the world frame, '0'. Given a local position ${}^i\boldsymbol{x}$ on a rigid body, its world position

is

$$ {}^0\boldsymbol{x} = {}^0_i\mathsf{E}\,{}^i\boldsymbol{x}, \tag{2} $$

where we have omitted the homogeneous coordinates for brevity.

The spatial velocity, ${}^i\phi_i$, also called a "twist," is composed of the angular component, ${}^i\boldsymbol{\omega}_i$, and the linear component, ${}^i\boldsymbol{\nu}_i$, both expressed in body coordinates:

$$ {}^i\phi_i = \begin{pmatrix} {}^i\boldsymbol{\omega}_i \\ {}^i\boldsymbol{\nu}_i \end{pmatrix}. \tag{3} $$

For a rigid body moving with spatial velocity, ${}^i\phi_i$, the world velocity of a point, ${}^i\boldsymbol{x}$, affixed to the rigid body is expressed as:

$$ {}^0\dot{\boldsymbol{x}} = {}^0_i\mathsf{R}\,\begin{pmatrix} [{}^i\boldsymbol{x}]^\top & I \end{pmatrix}\,{}^i\phi_i, \tag{4} $$

where $[\boldsymbol{a}]$ is the 3×3 cross-product matrix such that $[\boldsymbol{a}]\boldsymbol{b} = \boldsymbol{a}\times\boldsymbol{b}$.

The spatial velocity transforms from one frame to another according to the adjoint of the coordinate transform, which is defined from the rigid transform ${}^0_i\mathsf{E}$:

$$ {}^0_i\mathsf{Ad} = \begin{pmatrix} {}^0_i\mathsf{R} & 0 \\ [{}^0_i\boldsymbol{p}]\,{}^0_i\mathsf{R} & {}^0_i\mathsf{R} \end{pmatrix}. \tag{5} $$

The spatial velocity of the i^{th} rigid body in world coordinates is then

$$ {}^0\phi_i = {}^0_i\mathsf{Ad}\,{}^i\phi_i. \tag{6} $$

The time derivative of the adjoint, dropping the superscripts and subscripts for brevity, can be expressed as:

$$ \dot{\mathsf{Ad}} = \begin{pmatrix} \dot{\mathsf{R}} & 0 \\ [\dot{\boldsymbol{p}}]\mathsf{R} + [\boldsymbol{p}]\dot{\mathsf{R}} & \dot{\mathsf{R}} \end{pmatrix}. \tag{7} $$

This can be factored into a product of two matrices, $\mathsf{Ad}(\mathsf{E})$ and $\mathsf{ad}(\phi)$:

$$ \dot{\mathsf{Ad}}(\mathsf{E},\phi) = \underbrace{\begin{pmatrix} \mathsf{R} & 0 \\ [\boldsymbol{p}]\mathsf{R} & \mathsf{R} \end{pmatrix}}_{\mathsf{Ad}(\mathsf{E})} \underbrace{\begin{pmatrix} [\boldsymbol{\omega}] & 0 \\ [\boldsymbol{\nu}] & [\boldsymbol{\omega}] \end{pmatrix}}_{\mathsf{ad}(\phi)}, \tag{8} $$

where we have added the parameter list to more be explicit. The second factor, $\mathsf{ad} = \mathsf{Ad}^{-1}\dot{\mathsf{Ad}}$, is the spatial cross product matrix, which is the adjoint action of the Lie algebra on itself [Selig 2004; Kim 2012].

Finally, the Newton-Euler equations of motion of a single rigid body can be written in a compact form as:

$$ \begin{aligned} \mathsf{M}_i\dot{\phi}_i &= [\text{Coriolis forces}] + [\text{body forces (e.g., gravity)}] \\ &= \mathsf{ad}(\phi_i)^\top \mathsf{M}_i\phi_i + f_{\text{body}}(\mathsf{E}_i), \end{aligned} \tag{9} $$

where, M_i is the spatial inertia of the rigid body, $\dot{\phi}_i$ is the *acceleration* of the rigid body, and $\mathsf{ad}(\phi_i)$ is the spatial cross product matrix from Eq. (8). The mass matrix is diagonal because we express all velocities in body coordinates—*i.e.,* we use a body-fixed frame aligned with the principal axis of the body and whose origin is coincident with the center of mass of the body. By stacking all of the maximal acceleration DOFs together, $\ddot{\mathsf{q}}_m = \begin{pmatrix}\dot{\phi}_1^\top & \cdots & \dot{\phi}_n^\top\end{pmatrix}^\top$, we arrive at the following linear system for *maximal* DOFs:

$$ \mathsf{M}_m\ddot{\mathsf{q}}_m = \mathsf{f}_m. \tag{10} $$

3 REDUCED COORDINATES

In reduced coordinates, we express the state of the system using a reduced set of coordinates, such as joint angles. For the types of joints we are interested in, including highly complex joints, such as the spline curve and surface joints [Lee and Terzopoulos 2008], there is a linear relationship between the maximal and reduced velocities. Denoting by $\dot{\mathsf{q}}_m$ the vector of all maximal velocities and by $\dot{\mathsf{q}}_r$ the vector of all reduced velocities, we have:

$$ \dot{\mathsf{q}}_m = \mathsf{J}_{mr}\dot{\mathsf{q}}_r, \quad \ddot{\mathsf{q}}_m = \dot{\mathsf{J}}_{mr}\dot{\mathsf{q}}_r + \mathsf{J}_{mr}\ddot{\mathsf{q}}_r, \tag{11} $$

where J_{mr} is the Jacobian matrix that transforms velocities from reduced to maximal, which we derive later in this section. Then, combining Eq. (10) and Eq. (11), we obtain

$$ \left(\mathsf{J}_{mr}^\top \mathsf{M}_m \mathsf{J}_{mr}\right)\ddot{\mathsf{q}}_r = \mathsf{J}_{mr}^\top\left(\mathsf{f}_m - \mathsf{M}_m\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r\right). \tag{12} $$

The reduced inertia matrix, $\mathsf{M}_r = \mathsf{J}_{mr}^\top\mathsf{M}_m\mathsf{J}_{mr}$ is much smaller than their maximal counterpart. Furthermore, we do not require constraints, since the Jacobian automatically projects forces down to the reduced space. The last term, $-\mathsf{J}_{mr}^\top\mathsf{M}_m\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r$, is the extra *quadratic velocity vector* due to the change of coordinates, in analogy to the Coriolis force in Eq. (9) [Shabana 2013].

Assuming there are no loops, we have a tree structure describing the topology of the system. (Loops are handled with constraints.) Furthermore, we have a one-to-one relationship between a body and a joint—every body has a joint between itself and its parent body. The body frame is aligned to the body's inertial frame (as described below Eq. (9)), and the joint frame is aligned according to the joint type (*e.g.,* Z axis along the axis of rotation). We will use i to denote a body, j to denote its corresponding joint, and p to denote the parent *body* of i (or to the parent *joint* of j depending on the context).

The twists of bodies p and i at joint j are

$$ {}^j\phi_p = {}^j_p\mathsf{Ad}\,{}^p\phi_p, \quad {}^j\phi_i = {}^j_i\mathsf{Ad}\,{}^i\phi_i, \tag{13} $$

and their *relative* twist is

$$ \begin{aligned} {}^j\phi_j &= {}^j\phi_i - {}^j\phi_p \\ &= {}^j_i\mathsf{Ad}\,{}^i\phi_i - {}^j_p\mathsf{Ad}\,{}^p\phi_p \\ &= {}^j_i\mathsf{Ad}\,{}^i\phi_i - {}^j_i\mathsf{Ad}\,{}^i_0\mathsf{Ad}\,{}^0_p\mathsf{Ad}\,{}^p\phi_p, \end{aligned} \tag{14} $$

where 0 indicates the world frame. Since i owns the joint, ${}^j_i\mathsf{Ad}$ is constant. (It is constructed from ${}^j_i\mathsf{E}$, which represents where the i's body frame is wrt to the joint frame, which is set at initialization.) Note that in maximal coordinates, we store positions wrt the world and velocities wrt the body itself. In other words, for each body, we store ${}^0_i\mathsf{E}$ and ${}^i\phi_i$. So in the above expression, the adjoint matrices of the form ${}^0_i\mathsf{Ad}$ and ${}^i_0\mathsf{Ad}$ can be computed easily from ${}^0_i\mathsf{E}$. We can rearrange Eq. (14) to solve for body i's spatial velocity:

$$ \begin{aligned} {}^j_i\mathsf{Ad}\,{}^i\phi_i &= {}^j_i\mathsf{Ad}\,{}^i_0\mathsf{Ad}\,{}^0_p\mathsf{Ad}\,{}^p\phi_p + {}^j\phi_j \\ {}^i\phi_i &= {}^i_0\mathsf{Ad}\,{}^0_p\mathsf{Ad}\,{}^p\phi_p + {}^i_j\mathsf{Ad}\,{}^j\phi_j. \end{aligned} \tag{15} $$

What this expression implies is that if we know the parent body's velocity, ${}^p\phi_p$, and the joint's velocity, ${}^j\phi_j$, we can compute the child body's velocity, ${}^i\phi_i$. In reduced coordinates, we parameterize ${}^j\phi_j$ not with the full 6 degrees of freedom but with some subset $\dot{\mathsf{q}}_j \subseteq \mathbb{R}^6$.

For example, if we are using a revolute joint about the Z axis, we can write

$$ {}^{j}\phi_j = S_j \dot{\mathrm{q}}_j, \quad S_j = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}^\top. \tag{16} $$

We use S here to follow the notation of Park et al. [1995] and Kim [2012]. S takes on this simple constant form for revolute joints, but in general is a nonlinear function of the joint's DOFs, $\dot{\mathrm{q}}_j$. Derivations of other joint types are included in §4 and §5. Combining Eq. (15) and Eq. (16), we obtain the recursive expression for the velocity of body i:

$$ {}^{i}\phi_i = {}^{i}_{p}\mathrm{Ad}\, {}^{p}\phi_p + {}^{i}_{j}\mathrm{Ad}\, S_j \dot{\mathrm{q}}_j, \tag{17} $$

where ${}^{i}_{p}\mathrm{Ad} = {}^{i}_{0}\mathrm{Ad}\, {}^{0}_{p}\mathrm{Ad}$.

This relationship can be recursively applied to form the system Jacobian. For example, for a serial chain with three links, the Jacobian is the following lower triangular matrix, where we have explicitly labeled each body quantity with i and joint quantity with j:

$$ \underbrace{\begin{pmatrix} {}^{i1}\phi_{i1} \\ {}^{i2}\phi_{i2} \\ {}^{i3}\phi_{i3} \end{pmatrix}}_{\dot{\mathrm{q}}_m} = \underbrace{\begin{pmatrix} {}^{i1}_{j1}\mathrm{Ad}\, S_{j1} & 0 & 0 \\ {}^{i2}_{i1}\mathrm{Ad}\, {}^{i1}_{j1}\mathrm{Ad}\, S_{j1} & {}^{i2}_{j2}\mathrm{Ad}\, S_{j2} & 0 \\ {}^{i3}_{i2}\mathrm{Ad}\, {}^{i2}_{i1}\mathrm{Ad}\, {}^{i1}_{j1}\mathrm{Ad}\, S_{j1} & {}^{i3}_{i2}\mathrm{Ad}\, {}^{i2}_{j2}\mathrm{Ad}\, S_{j2} & {}^{i3}_{j3}\mathrm{Ad}\, S_{j3} \end{pmatrix}}_{\mathrm{J}_{mr}} \underbrace{\begin{pmatrix} \dot{\mathrm{q}}_{j1} \\ \dot{\mathrm{q}}_{j2} \\ \dot{\mathrm{q}}_{j3} \end{pmatrix}}_{\dot{\mathrm{q}}_r}. \tag{18} $$

The pseudocode to fill J_{mr} is given in Alg. 1. This function must be called on the joints in a tree traversal order starting from the root. This function takes advantage of the recursive structure of the tree hierarchy—as we traverse through the ancestors, we use the products already computed by the ancestors. Since this matrix has $O(n^2)$ elements, it takes $O(n^2)$ time to fill, even with its recursive structure. If we only need the product of this matrix with a vector, we only need $O(n)$ time, as shown in Alg. 2 and Alg. 3, a strategy implicitly exploited by the recursive dynamics algorithm [Featherstone 1983; Kim 2012].

To compute $\dot{\mathrm{J}}$, we can take the derivative of each term in Eq. (18). For the diagonal terms, the derivative is

$$ \dot{\mathrm{J}}(i,j) = {}^{i}_{j}\mathrm{Ad}\, \dot{S}_j, \tag{19} $$

which is 0 for revolute joints, since S is constant. For off-diagonal terms, we traverse the hierarchy through the joint's ancestors back to the root, and the recursive expression for the derivative is

$$ \dot{\mathrm{J}}(i,a) = {}^{i}_{p}\dot{\mathrm{A}}\mathrm{d}\, \mathrm{J}(p,a) + {}^{i}_{p}\mathrm{Ad}\, \dot{\mathrm{J}}(p,a), \tag{20} $$

Algorithm 1 Fills the Jacobian matrix and its time derivative

while forward traversal **do**
  $\mathrm{J}(i,j) = {}^{i}_{j}\mathrm{Ad}\, S_j$
  $\dot{\mathrm{J}}(i,j) = {}^{i}_{j}\mathrm{Ad}\, \dot{S}_j$
  ancestor a = parent *joint* of j
  while $a \neq$ null **do**
    $\mathrm{J}(i,a) = {}^{i}_{p}\mathrm{Ad}\, \mathrm{J}(p,a)$ ▷ $p = p(i)$ is the parent *body* of i
    $\dot{\mathrm{J}}(i,a) = {}^{i}_{p}\dot{\mathrm{A}}\mathrm{d}\, \mathrm{J}(p,a) + {}^{i}_{p}\mathrm{Ad}\, \dot{\mathrm{J}}(p,a)$
    $a = a$'s parent joint
  end while
end while

Algorithm 2 Computes products $y = \mathrm{J}x$ and $z = \dot{\mathrm{J}}x$

while forward traversal **do**
  $y(i) = {}^{i}_{j}\mathrm{Ad}\, S_j\, x(j)$
  $z(i) = {}^{i}_{j}\mathrm{Ad}\, \dot{S}_j\, x(j)$
  if $p \neq$ null **then** ▷ $p = p(j)$ is the parent joint of j
    $y(i)$ += ${}^{i}_{p}\mathrm{Ad}\, y(p)$
    $z(i)$ += ${}^{i}_{p}\mathrm{Ad}\, z(p) + {}^{i}_{p}\dot{\mathrm{A}}\mathrm{d}\, y(p)$
  end if
end while

Algorithm 3 Computes product $x = \mathrm{J}^\top y$

while backward traversal **do**
  $y_i = y(i)$
  for all children c **do** ▷ $c = c(j)$ is a child joint of j
    y_i += α_c ▷ α is a temp variable stored by each joint
  end for
  $\alpha_i = {}^{i}_{p}\mathrm{Ad}^\top y_i$ ▷ to be used by j's parent later
  $x(j) = S_j^\top\, {}^{i}_{j}\mathrm{Ad}^\top\, y_i$
end while

where $a = a(j)$ is an ancestor *joint* of j, and $p = p(i)$ is the parent *body* of i. To compute ${}^{i}_{p}\dot{\mathrm{A}}\mathrm{d}$, we use Eq. (8) and the identity for taking the derivative of the matrix inverse: $\dot{A}^{-1} = -A^{-1}\dot{A}A^{-1}$:

$$ {}^{i}_{p}\dot{\mathrm{A}}\mathrm{d} = -{}^{i}_{0}\mathrm{Ad}\, {}^{0}_{i}\dot{\mathrm{A}}\mathrm{d}\, {}^{i}_{0}\mathrm{Ad}\, {}^{0}_{p}\mathrm{Ad} + {}^{i}_{0}\mathrm{Ad}\, {}^{0}_{p}\dot{\mathrm{A}}\mathrm{d}. \tag{21} $$

4 BASIC JOINT TYPES

To model most mechanisms, we only require just a few basic joint types. The most important are the fixed joint (§4.1), the prismatic joint (§4.2), and the revolute joint (§4.5).

For all joint types, the joint transform is a 4×4 matrix that defines where the joint, j, is wrt its parent joint, $p(j)$:

$$ {}^{p}_{j}\mathrm{E} = {}^{p}_{j}\mathrm{E}_0\, \mathrm{Q}_j(\mathrm{q}_j), \tag{22} $$

where ${}^{p}_{j}\mathrm{E}_0$ is the initial transform (often a translation) that specifies where the joint is wrt its parent joint, and $\mathrm{Q}_j(\mathrm{q}_j)$ is the transform that actually applies the degrees of freedom of that joint. Additionally, for each joint type, we require S and $\dot{S}$ for the computation of J and $\dot{\mathrm{J}}$.

4.1 Fixed Joint

A fixed joint is used for rigidly attaching two bodies together. For a fixed joint, $\mathrm{q}_j = \varnothing$, and $\mathrm{Q}_j(\mathrm{q}_j)$ is simply the 4×4 identity matrix. The joint Jacobian, S, is an empty 6×0 matrix.

4.2 Prismatic Joint

A prismatic joint allows one degree of translational freedom. Let $\boldsymbol{a}$ represent the axis along which the joint is able to translate. Then

$$ \mathrm{Q}_j(\mathrm{q}_j) = \begin{pmatrix} I & \boldsymbol{a}\mathrm{q}_j \\ 0 & 1 \end{pmatrix}, \tag{23} $$

which is a 4×4 translation matrix. The corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 \\ \boldsymbol{a} \end{pmatrix} \in \mathbb{R}^{6\times 1}. \tag{24}$$

4.3 Planar Joint

A planar joint allows translation in two directions. We assume that the joint is oriented so that the allowed motion is in the X-Y plane. Then

$$\mathrm{Q}_j(\mathrm{q}_j) = \begin{pmatrix} I & 0 & \mathrm{q}_j \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{25}$$

which is again a 4×4 translation matrix. The corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{6\times 2}. \tag{26}$$

4.4 Translational Joint

A translational joint allows full translation (but no rotation).

$$\mathrm{Q}_j(\mathrm{q}_j) = \begin{pmatrix} I & \mathrm{q}_j \\ 0 & 1 \end{pmatrix}, \tag{27}$$

and the corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 \\ I \end{pmatrix} \in \mathbb{R}^{6\times 3}. \tag{28}$$

4.5 Revolute Joint

A revolute joint allows rotation about an axis, $\boldsymbol{a}$. The rotation matrix is constructed from the (axis, angle) pair: $(\boldsymbol{a}, \mathrm{q}_j)$.

$$\mathrm{Q}_j(\mathrm{q}_j) = \begin{pmatrix} \mathrm{R}(\boldsymbol{a}, \mathrm{q}_j) & 0 \\ 0 & 1 \end{pmatrix}, \tag{29}$$

and the corresponding joint Jacobian is

$$S = \begin{pmatrix} \boldsymbol{a} \\ 0 \end{pmatrix} \in \mathbb{R}^{6\times 3}. \tag{30}$$

4.6 Universal Joint

A universal joint allows bending in X and Y but no twisting along Z. We start with the rotation matrix corresponding to the XYZ Euler angles:

$$\mathrm{R} = \begin{pmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + s_1s_2c_3 & c_1c_3 - s_1s_2s_3 & -s_1c_2 \\ s_1s_3 - c_1s_2c_3 & s_1c_3 + c_1s_2s_3 & c_1c_2 \end{pmatrix}, \tag{31}$$

where $c_1 = \cos(q_1), c_2 = \cos(q_2)$, etc. We then fix the third angle at 0, so that $c_3 = 1$ and $s_3 = 0$. This gives us

$$\mathrm{R} = \begin{pmatrix} c_2 & 0 & s_2 \\ s_1s_2 & c_1 & -s_1c_2 \\ -c_1s_2 & s_1 & c_1c_2 \end{pmatrix}. \tag{32}$$

Q is then

$$\mathrm{Q} = \begin{pmatrix} \mathrm{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{33}$$

The joint Jacobian, S, is going to be a 6×2 matrix. To get the 1st column of S, we take the derivative of R wrt q_1 and premultiply by $\mathrm{R}^\top$. After some cancellations, we get a skew symmetric matrix, from which the angular elements are extracted into the first column of S. We repeat this for the second column, and the resulting matrix is

$$S = \begin{pmatrix} c_2 & 0 \\ 0 & 1 \\ s_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{34}$$

The time derivative of the joint Jacobian is

$$\dot{S} = \begin{pmatrix} -s_2\dot{q}_2 & 0 \\ 0 & 0 \\ c_2\dot{q}_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{35}$$

4.7 Spherical Joint

No matter which 3-parameter representation we choose for a spherical joint, there is going to be a singularity somewhere. We could alternatively use a quaternion, but then we would need to add a constraint to keep the quaternion be of unit length. With Euler angles, to stay away from singularities, we need to switch the coordinate chart on the fly (*e.g.,* between ZYX and ZYZ). For these reasons, we use exponential coordinates [Grassia 1998; Gallego and Yezzi 2015]. Exponential coordinates also need to be reparameterized, but we do not need to keep track of the coordinate chart as with Euler angles.

Let $\mathrm{q} \in \mathrm{so}(3)$ (can also be thought of as $\mathbb{R}^3$) be the DOF of the spherical joint. Recall that every rotation matrix can be expressed as a matrix exponential of a skew symmetric matrix, and Q is then

$$\mathrm{R} = \exp([\mathrm{q}]), \quad \mathrm{Q} = \begin{pmatrix} \mathrm{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{36}$$

The joint Jacobian, S, is computed using the derivative formula described by Gallego and Yezzi [2015]. The derivative of R wrt q is a $3 \times 3 \times 3$ tensor, with each 3×3 slice given by:

$$\frac{\partial \mathrm{R}}{\partial q_i} = \frac{q_i[\mathrm{q}] + [[\mathrm{q}](I - \mathrm{R})\boldsymbol{e}_i]}{\mathrm{q}^\top \mathrm{q}}\mathrm{R}, \tag{37}$$

where $\boldsymbol{e}_i$ is the i^{th} standard basis in $\mathbb{R}^3$, and I is the identity matrix. If $\|\mathrm{q}\| < \varepsilon$, then we must take the limit as $\mathrm{q} \to 0$, which gives us $\mathrm{R} = I$, and $\partial \mathrm{R}/\partial q_i = [\boldsymbol{e}_i]$. Each column of the joint Jacobian is then

$$[S_i] = \mathrm{R}^\top \frac{\partial \mathrm{R}}{\partial q_i}. \tag{38}$$

The bracket around S_i implies that we need to unbracket the RHS to get each column of S. By contracting the $3 \times 3 \times 3$ tensor $\partial \mathrm{R}/\partial \mathrm{q}$ by $\dot{\mathrm{q}}$, we can compute the time derivative of the rotation matrix, $\dot{\mathrm{R}}$,

which is needed for $\dot{S}$:

$$\dot{\mathrm{R}} = \sum_i \frac{\partial \mathrm{R}}{\partial q_i}\dot{q}_i. \tag{39}$$

To aid in the derivation of $\dot{S}$, we first partition the derivative as

$$\begin{aligned} \frac{\partial \mathrm{R}}{\partial q_i} &= A_i\mathrm{R}, \quad A_i = (B_i + C_i)d, \\ B_i &= q_i[\mathrm{q}], \quad C_i = [[\mathrm{q}](I - \mathrm{R})\boldsymbol{e}_i], \quad d = \frac{1}{\mathrm{q}^\top \mathrm{q}}. \end{aligned} \tag{40}$$

Then each column of $\dot{S}$ can be expressed as

$$\begin{aligned} \left[\dot{S}_i\right] &= \dot{\mathrm{R}}^\top A_i \mathrm{R} + \mathrm{R}^\top \dot{A}_i \mathrm{R} + \mathrm{R}^\top A_i \dot{\mathrm{R}} \\ \dot{A}_i &= \left(\dot{B}_i + \dot{C}_i\right) d + \left(B_i + C_i\right) \dot{d} \\ \dot{B}_i &= \dot{\mathrm{q}}_i[\mathrm{q}] + \mathrm{q}_i[\dot{\mathrm{q}}] \\ \dot{C}_i &= \left[[\dot{\mathrm{q}}](I - \mathrm{R})\boldsymbol{e}_i - [\mathrm{q}]\dot{\mathrm{R}}\boldsymbol{e}_i\right] \\ \dot{d} &= \frac{-2\mathrm{q}^\top \dot{\mathrm{q}}}{(\mathrm{q}^\top \mathrm{q})^2}. \end{aligned} \tag{41}$$

We still need to deal with singularities. Quoting Grassia [1998], "The exponential map has singularities on the spheres of radius $2n\pi$ (for $n = 1, 2, 3, \ldots$). This makes sense, since a rotation of 2π about any axis is equivalent to no rotation at all—the entire shell of points 2π distant from the origin (and 4, etc.) collapses to the identity in SO(3)." They then show that a good way to avoid singularities is to check if $\|\mathrm{q}\|$ is close to 2π and if so, reparameterize as $\mathrm{q} = (1 - 2\pi/\|\mathrm{q}\|)\mathrm{q}$. Whenever q is reparameterized, we must also update $\dot{\mathrm{q}}$, S, and $\dot{S}$. To do so, we first recompute S with Eq. (38) using the reparameterized q. Then, we can compute the new velocities as $\dot{\mathrm{q}} = S^{-1}S_{\text{prev}}\dot{\mathrm{q}}_{\text{prev}}$. Finally, we can compute $\dot{S}$ using Eq. (41) with the new values of q and $\dot{\mathrm{q}}$.

4.8 Free Joint

A free joint is a joint that is completely unconstrained. This is useful if we want a structure that is not affixed to the ground, in which case the root joint will be implemented as a free joint. To implement a free joint, we concatenate a spherical joint and a translational joint together:

$$\mathrm{Q} = \mathrm{Q}_1\mathrm{Q}_2, \tag{42}$$

where $\mathrm{Q}_1 = \mathrm{Q}_{\text{spherical}}$ and $\mathrm{Q}_2 = \mathrm{Q}_{\text{translational}}$. The corresponding joint Jacobian is

$$S = \begin{pmatrix} \hat{S}_1 & 0 \\ -[\boldsymbol{q}_2]\hat{S}_1 & I \end{pmatrix} \in \mathbb{R}^{6\times 6}, \tag{43}$$

where $\hat{S}_1$ is the top three rows of S_1, and $\boldsymbol{q}_2 \in \mathbb{R}^3$ is the translational DOF of joint 2. The time derivative of the Jacobian is

$$\dot{S} = \begin{pmatrix} \dot{\hat{S}}_1 & 0 \\ -[\dot{\boldsymbol{q}}_2]\hat{S}_1 - [\boldsymbol{q}_2]\dot{\hat{S}}_1 & 0 \end{pmatrix}, \tag{44}$$

where $\dot{\hat{S}}_1$ is the top three rows of $\dot{S}_1$, and $\dot{\boldsymbol{q}}_2$ is the (translational) velocity of joint 2.

5 ADVANCED JOINT TYPES

RedMax also supports more advanced joint types, including the spline joint by Lee and Terzopoulos [2008].

5.1 Composite Joint

The free joint can be seen as a special case of a composite joint, where two joints are composed together: $\mathrm{Q} = \mathrm{Q}_1\mathrm{Q}_2$. This can be interpreted as a chaining of two joints, with a massless body in between, with 1 as a parent of 2. The corresponding joint Jacobian is

$$S = \begin{pmatrix} {}^2_1\mathrm{Ad}\, S_1 & S_2 \end{pmatrix} \in \mathbb{R}^{6\times(n_1+n_2)}, \tag{45}$$

where n_1 and n_2 are the number of DOFs of joints 1 and 2, respectively, and ${}^2_1\mathrm{Ad}$ is the 6×6 adjoint matrix that transforms from joint 1's coordinate space to joint 2's coordinate space. The time derivative of the right term, S_2, is simply $\dot{S}_2$, which is computed by joint 2. The time derivative of the left term is

$$\frac{d}{dt}\left({}^2_1\mathrm{Ad}\, S_1\right) = {}^2_1\dot{\mathrm{Ad}}\, S_1 + {}^2_1\mathrm{Ad}\, \dot{S}_1. \tag{46}$$

To compute ${}^2_1\dot{\mathrm{Ad}}$, note that joint 2 stores its transform wrt joint 1 (child wrt parent), which is ${}^1_2\mathrm{Ad}$. The transform of the parent wrt to the child involves the inverse, and so we have

$$\begin{aligned} {}^2_1\dot{\mathrm{Ad}} &= -{}^2_1\mathrm{Ad}\, {}^1_2\dot{\mathrm{Ad}}\, {}^2_1\mathrm{Ad} && (47) \\ &= -{}^2_1\mathrm{Ad}\, {}^1_2\mathrm{Ad}\, \mathrm{ad}\left({}^2\phi_2\right)\, {}^2_1\mathrm{Ad} && (48) \\ &= -\mathrm{ad}\left(S_2\dot{\mathrm{q}}_2\right)\, {}^2_1\mathrm{Ad}. && (49) \end{aligned}$$

The twist of joint 2, ${}^2\phi_2$, is the spatial velocity of 2 wrt 1, which is the product $S_2\dot{\mathrm{q}}_2$. For example, if joint 2 is a translational joint, then

$$S_2\dot{\mathrm{q}}_2 = \begin{pmatrix} 0 \\ \dot{\boldsymbol{q}}_2 \end{pmatrix} \in \mathbb{R}^6, \tag{50}$$

which is a translation-only twist. Combining Eq. (45), Eq. (46), and Eq. (47), the time derivative of S is, therefore,

$$\dot{S} = \begin{pmatrix} -\mathrm{ad}\left(S_2\dot{\mathrm{q}}_2\right)\, {}^2_1\mathrm{Ad}\, S_1 + {}^2_1\mathrm{Ad}\, \dot{S}_1 & \dot{S}_2 \end{pmatrix}. \tag{51}$$

Composite joints can be chained together. For example, a composite joint of three joints can be expressed as $\mathrm{Q} = \mathrm{Q}_1(\mathrm{Q}_2\mathrm{Q}_3)$.

5.2 Spline Curve Joint

With RedMax, it is easy to include more advanced joints, such as the Spline Joint by Lee and Terzopoulos [2008]. We'll start by reviewing some basic spline concepts. For concreteness, we'll be using uniform cubic B-spline curves.

Let $C \in \mathbb{R}^{3\times 4}$ be the matrix of 4 consecutive control points:

$$C = \begin{pmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \boldsymbol{c}_3 & \boldsymbol{c}_4 \end{pmatrix}, \tag{52}$$

and let $B \in \mathbb{R}^{4\times 4}$ be the cubic B-spline basis matrix:

$$B = \frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{53}$$

Then the spline position at $q \in [0, 1]$ can be written as

$$\boldsymbol{x}(q) = CB\vec{q}, \quad \vec{q} = \begin{pmatrix} 1 \\ q \\ q^2 \\ q^3 \end{pmatrix}. \tag{54}$$

Other types of splines can be swapped in by replacing the basis matrix, B. If there are more than 4 control points, then the matrix C

needs to be updated so that the appropriate 4 control points make up the 4 columns of the matrix, and the spline parameter, q, must always be mapped to be between 0 and 1.

We can expand Eq. (54) in terms of the control points, $\boldsymbol{c}_i$:

$$\boldsymbol{x}(q) = \boldsymbol{c}_1 B_1(q) + \boldsymbol{c}_2 B_2(q) + \boldsymbol{c}_3 B_3(q) + \boldsymbol{c}_4 B_4(q), \tag{55}$$

where the basis function, $B_i(q)$, is the product of the i^{th} row of B and $\vec{q}$.

The spline joint uses the cumulative form of basis functions, introduced by Kim et al. [1995]:

$$\boldsymbol{x}(q) = \boldsymbol{c}_1 \tilde{B}_1(q) + \Delta\boldsymbol{c}_2 \tilde{B}_2(q) + \Delta\boldsymbol{c}_3 \tilde{B}_3(q) + \Delta\boldsymbol{c}_4 \tilde{B}_4(q), \tag{56}$$

where the control point differences are computed as $\Delta\boldsymbol{c}_i = \boldsymbol{c}_i - \boldsymbol{c}_{i-1}$. By equating Eq. (55) and Eq. (56), the cumulative basis functions, $\tilde{B}_i(q)$, are:

$$\begin{aligned}
\tilde{B}_4(q) &= B_4(q) \\
\tilde{B}_3(q) &= B_3(q) + B_4(q) \\
\tilde{B}_2(q) &= B_2(q) + B_3(q) + B_4(q) \\
\tilde{B}_1(q) &= B_1(q) + B_2(q) + B_3(q) + B_4(q) = 1.
\end{aligned} \tag{57}$$

The derivatives, $\tilde{B}_i'(q)$ and $\tilde{B}_i''(q)$, are computed by differentiating $\vec{q}$:

$$B'(q) = \frac{1}{6}\begin{pmatrix} -3 & 3 & -1 \\ 0 & -6 & 3 \\ 3 & 3 & -3 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 2q \\ 3q^2 \end{pmatrix}, \quad B''(q) = \frac{1}{6}\begin{pmatrix} 3 & -1 \\ -6 & 3 \\ 3 & -3 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 2 \\ 6q \end{pmatrix}, \tag{58}$$

where we have removed the zero entries from $\vec{q}'$ and $\vec{q}''$, and the corresponding columns from B.

With the spline joint, instead of control *points* $\boldsymbol{c}_i \in \mathbb{R}^3$, we have control *frames* $\mathrm{C}_i \in \mathrm{SE}(3)$. We use the cumulative form, Eq. (56), but instead of subtracting to get the control point differences, we use the matrix logarithm to get the control frame differences. Following Eq. (56), the joint matrix can be expressed using products of exponentials instead of additions:

$$\mathrm{Q}(\mathrm{q}) = \mathrm{C}_1 \exp\left(\Delta\mathrm{C}_2 \tilde{B}_2(\mathrm{q})\right) \exp\left(\Delta\mathrm{C}_3 \tilde{B}_3(\mathrm{q})\right) \exp\left(\Delta\mathrm{C}_4 \tilde{B}_4(\mathrm{q})\right), \tag{59}$$

where the control frame differences are computed using logarithms: $\Delta\mathrm{C}_i = \log(\mathrm{C}_{i-1}^{-1}\mathrm{C}_i)$.

The recursive method for computing the corresponding joint Jacobian, $S = [\mathrm{Q}^{-1}(\partial\mathrm{Q}/\partial\mathrm{q})]$, and Hessian, $\partial S/\partial\mathrm{q}$, are given in the appendix of the spline joints paper [Lee and Terzopoulos 2008], which we reproduce in Alg. 4 for reference. Once we compute $\partial S/\partial\mathrm{q}$, $\dot{S}$ can be computed using the chain rule: $\dot{S} = (\partial S/\partial\mathrm{q})\dot{\mathrm{q}}$.

Algorithm 4 Cubic Spline Joint transform, Jacobian, and Hessian

```
Q = C_1 exp(ΔC_2 B̃_2(q))
S = ΔC_2 B̃_2'(q)
∂S/∂q = ΔC_2 B̃_2''(q)
for i = 3, 4 do
    Q_i = exp(ΔC_i B̃_i(q))
    Q = Q Q_i
    Ad_i = Ad(Q_i^{-1})
    ad_i = ad(S)
    S = ΔC_i B̃_i'(q) + Ad_i S
    ∂S/∂q = ΔC_i B̃_i''(q) + Ad_i (∂S/∂q + ad_i ΔC_i B̃_i(q))
end for
```

5.3 Spline Surface Joint

For the spline surface joint (called the multi-DOF spline joint by Lee and Terzopoulos [2008]), we will again use uniform cubic B-splines, and we will limit ourselves to $n = 2$, which means we have a tensor product surface:

$$f(C, q_1, q_2) = \vec{q}_1^{\,\top} B^\top C B \vec{q}_2, \quad \vec{q}_i = \begin{pmatrix} 1 & q_i & q_i^2 & q_i^3 \end{pmatrix}^\top, \tag{60}$$

where B is the spline basis matrix from Eq. (53), and C is the 4×4 matrix of control values. The derivatives of the tensor product surface are:

$$\begin{aligned}
\vec{q}_i' &= \begin{pmatrix} 0 & 1 & 2q_i & 3q_i^2 \end{pmatrix}^\top, \quad \vec{q}_i'' = \begin{pmatrix} 0 & 0 & 2 & 6q_i \end{pmatrix}^\top, \\
\frac{\partial f}{\partial q_1} &= \vec{q}_1'^{\,\top} B^\top C B \vec{q}_2, \quad \frac{\partial f}{\partial q_2} = \vec{q}_1^{\,\top} B^\top C B \vec{q}_2', \\
\frac{\partial^2 f}{\partial q_1^2} &= \vec{q}_1''^{\,\top} B^\top C B \vec{q}_2, \quad \frac{\partial^2 f}{\partial q_2^2} = \vec{q}_1^{\,\top} B^\top C B \vec{q}_2'', \\
\frac{\partial^2 f}{\partial q_1 q_2} &= \frac{\partial^2 f}{\partial q_2 q_1} = \vec{q}_1'^{\,\top} B^\top C B \vec{q}_2',
\end{aligned} \tag{61}$$

In the multi-DOF spline joint, Lee and Terzopoulos [2008] suggest using splines to process the 3 rotational and 3 translational degrees of freedom individually. They also suggest putting the translational basis in front of the rotational basis, so that the resulting transformation matrix behaves more intuitively. (*I.e.*, E = TR is more intuitive than E = RT, because the translation values in T go directly into the last column of the E matrix rather than being rotated by R.) The 6 basis vectors are then:

$$\begin{aligned}
\hat{e}_1 &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \hat{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \\
\hat{e}_4 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_5 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_6 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.
\end{aligned} \tag{62}$$

The resulting transformation is a spline-weighted product of the matrix exponentials of these basis vectors:

$$\mathrm{Q}(\mathrm{q}) = \prod_{k=1}^{6} \exp\left(\hat{e}_k f(C_k, \mathrm{q})\right), \tag{63}$$

where $C_k \in \mathbb{R}^{4\times4}$ is matrix of control values for the k^{th} basis. For example, C_1 through C_3 are the x, y, and z positions of the 16 control frames. By multiplying the rotations together, the spline frame acts as XYZ Euler angles, and so Lee and Terzopoulos [2008] warn against

Algorithm 5 Cubic Spline Surface Joint transform, Jacobian, and Hessian

$\mathsf{Q} = \exp(\hat{e}_1 f(C_1, \mathsf{q}))$
for $i = 1, 2$ **do**
  // S_i is the i^{th} column of S; $\partial f_i = \frac{\partial f}{\partial q_i}$
  $S_i = \hat{e}_1 \partial f_i(\mathsf{C}_1, \mathsf{q}) \in \mathbb{R}^6$
  for $j = 1, 2$ **do**
    // ∂S_{ij} is the $(i,j)^{th}$ column of $\partial S/\partial \mathsf{q}$; $\partial^2 f_{ij} = \frac{\partial^2 f}{\partial q_i q_j}$
    $\partial S_{ij} = \hat{e}_1 \partial^2 f_{ij}(C_1, \mathsf{q}) \in \mathbb{R}^6$
  end for
end for
for $k = 2, \cdots, 6$ **do**
  $\mathsf{Q}_k = \exp(\hat{e}_k f(C_k, \mathsf{q}))$
  $\mathsf{Q} = \mathsf{Q}\mathsf{Q}_k$
  $\mathsf{Ad}_k = \mathsf{Ad}(\mathsf{Q}_k^{-1})$
  for $i = 1, 2$ **do**
    $\mathsf{ad}_i = \mathsf{ad}(S_i)$
    $S_i = \hat{e}_k \partial f_i(C_k, \mathsf{q}) + \mathsf{Ad}_k S_i$
    for $j = 1, 2$ **do**
      $\partial S_{ij} = \hat{e}_k \partial^2 f_{ij}(C_k, \mathsf{q}) + \mathsf{Ad}_k(\partial S_{ij} + \mathsf{ad}_i \hat{e}_k \partial f_j(C_k, \mathsf{q}))$
    end for
  end for
end for

gimbal locks. This should not be a problem as long as the rotations are small ($< \pi/4$).

The joint Jacobian, $S \in \mathbb{R}^{6\times 2}$, and Hessian, $\partial S/\partial \mathsf{q} \in \mathbb{R}^{6\times 2\times 2}$, can be computed recursively, as shown in Alg. 5. Once we compute $\partial S/\partial \mathsf{q}$, $\dot{S}$ can be computed using the chain rule: $\dot{S} = (\partial S/\partial \mathsf{q})\dot{\mathsf{q}}$, which is a tensor product. In MATLAB notation, this can be written as `Sdot = dSdq(:,:,1)*qdot(1) + dSdq(:,:,2)*qdot(2)`.

6 RECURSIVE HYBRID DYNAMICS

For reference, we show in Alg. 6 the recursive hybrid dynamics algorithm by Kim and Pollard [2011], with minor notational changes. Here, j refers to joint frame, p refers to the parent of j, c refers to a child of j, and i refers to the body owned by j.

List of symbols:

- $\mathsf{q}_j, \dot{\mathsf{q}}_j, \ddot{\mathsf{q}}_j \in \mathbb{R}^{n_j}$: generalized position, velocity, acceleration of the joint.
- $\tau_j \in \mathbb{R}^{n_j}$: torque (or generalized force) on joint.
- ${}^p_j\mathsf{E} \in \mathrm{SE}(3)$: Transformation matrix from p to j, a function of q_j.
- $S_j = \left({}^p_j\mathsf{E}^{-1} \frac{d\, {}^p_j\mathsf{E}}{d\mathsf{q}_1} \quad \cdots \quad {}^p_j\mathsf{E}^{-1} \frac{d\, {}^p_j\mathsf{E}}{d\mathsf{q}_{n_j}} \right) \in \mathbb{R}^{6\times n_j}$: Jacobian of ${}^p_j\mathsf{E}$, viewed in j.
- $\mathsf{V}_j \in \mathrm{se}(3)$: twist at the joint. The twist at the body center is ${}^i\phi_i = {}^i_j\mathsf{Ad}\, \mathsf{V}_j$ and vice-versa.
- $\dot{\mathsf{V}}_j \in \mathrm{se}(3)$: Component-wise time derivative of V_j.
- $\mathsf{M}_j \in \mathbb{R}^{6\times 6}$: Body inertia at the joint. Since the joint is not at the body center, it is not diagonal. It can be calculated from body-centered inertia as $\mathsf{M}_j = {}^i_j\mathsf{Ad}^\top \mathsf{M}_i\, {}^i_j\mathsf{Ad}$.

Algorithm 6 Recursive Hybrid Dynamics

while forward traversal **do** ▷ p = parent joint of j
  ${}^p_j\mathsf{E}$ = function of q_j
  $\mathsf{V}_j = {}^j_p\mathsf{Ad}\, \mathsf{V}_p + S_j \dot{\mathsf{q}}_j$
  $\eta_j = \mathsf{ad}(\mathsf{V}_j) S_j \dot{\mathsf{q}}_j + \dot{S}_j \dot{\mathsf{q}}_j$
end while
while backward traversal **do** ▷ c = child joint of j
  $\hat{\mathsf{M}}_j = \mathsf{M}_j + \sum_c {}^c_j\mathsf{Ad}^\top \Pi_c\, {}^c_j\mathsf{Ad}$
  $\hat{\mathcal{B}}_j = -\mathsf{ad}(\mathsf{V}_j)^\top \mathsf{M}_j \mathsf{V}_j - \mathsf{F}_j^{\mathrm{ext}} + \sum_c {}^c_j\mathsf{Ad}^\top \beta_c$
  if prescribed acceleration **then**
    $\Pi_j = \hat{\mathsf{M}}_j$
    $\beta_j = \hat{\mathcal{B}}_j + \hat{\mathsf{M}}_j(\eta_j + S_j \ddot{\mathsf{q}}_j)$
  else
    $\Psi_j = (S_j^\top \hat{\mathsf{M}}_j S_j)^{-1}$
    $\Pi_j = \hat{\mathsf{M}}_j - \hat{\mathsf{M}}_j S_j \Psi_j S_j^\top \hat{\mathsf{M}}_j$
    $\beta_j = \hat{\mathcal{B}}_j + \hat{\mathsf{M}}_j \left(\eta_j + S_j \Psi_j \left(\tau_j - S_j^\top \left(\hat{\mathsf{M}}_j \eta_j + \hat{\mathcal{B}}_j \right) \right) \right)$
  end if
end while
while forward traversal **do** ▷ p = parent joint of j
  if prescribed acceleration **then**
    $\dot{\mathsf{V}}_j = {}^j_p\mathsf{Ad}\, \dot{\mathsf{V}}_p + S_j \ddot{\mathsf{q}}_j + \eta_j$
    $\mathsf{F}_j = \hat{\mathsf{M}}_j \dot{\mathsf{V}}_j + \hat{\mathcal{B}}_j$
    $\tau_j = S_j^\top \mathsf{F}_j$
  else
    $\ddot{\mathsf{q}}_j = \Psi_j \left(\tau_j - S_j^\top \hat{\mathsf{M}}_j \left({}^j_p\mathsf{Ad}\, \dot{\mathsf{V}}_p + \eta_j \right) - S_j^\top \hat{\mathcal{B}}_j \right)$
    $\dot{\mathsf{V}}_j = {}^j_p\mathsf{Ad}\, \dot{\mathsf{V}}_p + S_j \ddot{\mathsf{q}}_j + \eta_j$
    $\mathsf{F}_j = \hat{\mathsf{M}}_j \dot{\mathsf{V}}_j + \hat{\mathcal{B}}_j$
  end if
end while

- $\mathsf{F}_j \in \mathrm{dse}(3)$: Generalized force acting on the joint due to the connection to the parent, viewed in j. (dse(3) is the space of generalized forces acting on SE(3))
- $\mathsf{F}_j^{\mathrm{ext}} \in \mathrm{dse}(3)$: Generalized external force viewed in i. Gravity can be calculated as $\mathsf{F}_j^{\mathrm{ext}} = {}^i_j\mathsf{Ad}^\top \mathsf{F}_i^{\mathrm{grav}}$, where $\mathsf{F}_i^{\mathrm{grav}}$ is the force of gravity acting on the body center (zeros in the top three rows and $m\boldsymbol{g}$ in the bottom three rows).

REFERENCES

R. Featherstone. 1983. The Calculation of Robot Dynamics Using Articulated-body Inertias. *INT J ROBOT RES* 2, 1 (1983), 13–30.

G. Gallego and A. Yezzi. 2015. A Compact Formula for the Derivative of a 3-D Rotation in Exponential Coordinates. *J. Math. Imaging Vision* 51, 3 (2015), 378–384.

F. S. Grassia. 1998. Practical Parameterization of Rotations Using the Exponential Map. *J. Graph. Tools* 3, 3 (March 1998), 29–48.

J. Kim. 2012. *Lie Group Formulation of Articulated Rigid Body Dynamics.* Technical Report. Carnegie Mellon University.

J. Kim and N. S. Pollard. 2011. Fast Simulation of Skeleton-driven Deformable Body Characters. *ACM Trans. Graph.* 30, 5, Article 121 (Oct. 2011).

M.-J. Kim, M.-S. Kim, and S. Y. Shin. 1995. A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives. In *Annual Conference Series (Proc. SIGGRAPH).* ACM, New York, NY, USA, 369–376.

S.-H. Lee and D. Terzopoulos. 2008. Spline Joints for Multibody Dynamics. *ACM Trans. Graph.* 27, 3, Article 22 (Aug. 2008).

F. C. Park, J. E. Bobrow, and S. R. Ploen. 1995. A Lie Group Formulation of Robot Dynamics. *INT J ROBOT RES* 14, 6 (1995), 609–618.

J. M. Selig. 2004. Lie Groups and Lie Algebras in Robotics. In *Computational Noncommutative Algebra and Applications*. Springer, 101–125.

A. A. Shabana. 2013. *Dynamics of Multibody Systems*. Cambridge University press.