Differentiable Simulation of Inertial Musculotendons: Supplementary Document

YING WANG, Texas A&M University, USA
JASPER VERHEUL, Cardiff Metropolitan University, UK
SANG-HOON YEO, University of Birmingham, UK
NIMA KHADEMI KALANTARI, Texas A&M University, USA
SHINJIRO SUEDA, Texas A&M University, USA

ACM Reference Format:
Ying Wang, Jasper Verheul, Sang-Hoon Yeo, Nima Khademi Kalantari, and Shinjiro Sueda. 2022. Differentiable Simulation of Inertial Musculotendons: Supplementary Document. *ACM Trans. Graph.* 41, 6, Article 1 (December 2022), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 DERIVATION DETAILS

In this section, we provide additional derivation details for Type II and Type III muscle paths.

1.1 Type II: Path Point Muscles

For Path Point Muscles, we require

$$\begin{aligned} \mathbf{J}_{\alpha m} &= \mathbf{J}_{\alpha z}\mathbf{J}_{zm} \\ \dot{\mathbf{J}}_{\alpha m} &= \dot{\mathbf{J}}_{\alpha z}\mathbf{J}_{zm} + \mathbf{J}_{\alpha z}\dot{\mathbf{J}}_{zm}, \end{aligned} \tag{1}$$

where

$$\mathbf{J}_{zm} = \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{sx} \end{pmatrix} \mathbf{J}_{xm}, \quad \dot{\mathbf{J}}_{zm} = \begin{pmatrix} \mathbf{0} \\ \dot{\mathbf{J}}_{sx} \end{pmatrix} \mathbf{J}_{xm} + \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{sx} \end{pmatrix} \dot{\mathbf{J}}_{xm}. \tag{2}$$

In the rest of this subsection, we will derive $\mathbf{J}_{sx}$, $\dot{\mathbf{J}}_{sx}$, $\mathbf{J}_{\alpha z}$, and $\dot{\mathbf{J}}_{\alpha z}$.

1.1.1 *Derivation of $\mathbf{J}_{sx}$ and $\dot{\mathbf{J}}_{sx}$.*

Here, we follow the derivation of EOL strands by Sachdeva et al. [2015]. Let n be the number of path points between the origin and insertion, as shown in Fig. 1. Then there are $n+1$ line segments, and within each segment i, we define

$$\Delta\mathbf{x}_i = \mathbf{x}_{i+1} - \mathbf{x}_i, \quad \Delta s_i = s_{i+1} - s_i. \tag{3}$$

The length of each segment is then

$$l_i = \|\Delta\mathbf{x}_i\|, \tag{4}$$

and we further define

$$\Delta\bar{\mathbf{x}}_i = \frac{\Delta\mathbf{x}_i}{l_i}. \tag{5}$$

Authors' addresses: Ying Wang, Texas A&M University, USA, ying.wang@tamu.edu; Jasper Verheul, Cardiff Metropolitan University, UK, jpverheul@cardiffmet.ac.uk; Sang-Hoon Yeo, University of Birmingham, UK, s.yeo@bham.ac.uk; Nima Khademi Kalantari, Texas A&M University, USA, nimak@tamu.edu; Shinjiro Sueda, Texas A&M University, USA, sueda@tamu.edu.

2022. Manuscript submitted to ACM

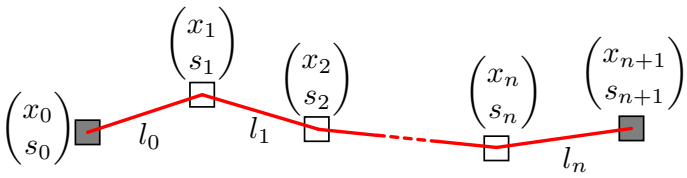


Fig. 1. An EOL strand with n path points between origin and insertion.

Then we form the following three matrices:

$$\mathbf{L} = \begin{pmatrix} l_0 + l_1 & -l_0 & & & & & \\ -l_2 & l_1 + l_2 & -l_1 & & & & \\ & -l_3 & l_2 + l_3 & -l_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -l_{n-1} & l_{n-2} + l_{n-1} & -l_{n-2} & \\ & & & & -l_n & l_{n-1} + l_n \end{pmatrix} \in \mathbb{R}^{n \times n} \tag{6}$$

$$\Delta\mathbf{S} = \begin{pmatrix} -\Delta s_1 & \Delta s_0 & & & \\ & -\Delta s_2 & \Delta s_1 & & \\ & & \ddots & \ddots & \\ & & & -\Delta s_n & \Delta s_{n-1} \end{pmatrix} \in \mathbb{R}^{n \times n+1} \tag{7}$$

$$\Delta\bar{\mathbf{X}} = \begin{pmatrix} -\Delta\bar{\mathbf{x}}_0^\top & \Delta\bar{\mathbf{x}}_0^\top & & & \\ & -\Delta\bar{\mathbf{x}}_1^\top & \Delta\bar{\mathbf{x}}_1^\top & & \\ & & \ddots & \ddots & \\ & & & -\Delta\bar{\mathbf{x}}_n & \Delta\bar{\mathbf{x}}_n \end{pmatrix} \in \mathbb{R}^{n+1 \times 3(n+2)}. \tag{8}$$

The Jacobian for mapping from the Lagrangian velocities to the Eulerian velocities can be expressed as:

$$\mathbf{J}_{sx} = -\mathbf{L}^{-1} \Delta\mathbf{S}\, \Delta\bar{\mathbf{X}} \in \mathbb{R}^{n \times 3(n+2)}, \tag{9}$$

This Jacobian maps the Lagrangian velocities of the $n + 2$ points (origin, insertion, and internal path points) to the Eulerian velocities of just the n internal path points (excluding origin and insertion). In other words, if we know the Lagrangian velocities of the origin, insertion, and the internal path points, then we can compute the Eulerian velocities of the internal path points. (The Eulerian velocities of the origin and insertion are always zero.)

The time derivative of the Jacobian is:

$$\dot{\mathbf{J}}_{sx} = -\mathbf{L}^{-1} \left(\dot{\mathbf{L}} \mathbf{J}_{sx} + \Delta\dot{\mathbf{S}}\, \Delta\bar{\mathbf{X}} + \Delta\mathbf{S}\, \Delta\dot{\bar{\mathbf{X}}} \right). \tag{10}$$

To derive these matrix quantities, we use the following scalar and vector quantities.

$$\Delta\dot{\mathbf{x}}_i = \dot{\mathbf{x}}_{i+1} - \dot{\mathbf{x}}_i \tag{11}$$

$$\Delta\dot{s}_i = \dot{s}_{i+1} - \dot{s}_i \tag{12}$$

$$\dot{l}_i = \frac{\Delta\mathbf{x}_i^\top}{l_i} \Delta\dot{\mathbf{x}}_i \tag{13}$$

$$\Delta\dot{\bar{\mathbf{x}}}_i = \left(\mathbf{I} - \Delta\bar{\mathbf{x}}_i \Delta\bar{\mathbf{x}}_i^\top\right) \frac{\Delta\dot{\mathbf{x}}_i}{l_i}. \tag{14}$$

1.1.2 *Derivation of $\mathbf{J}_{\alpha z}$ and $\dot{\mathbf{J}}_{\alpha z}$.*

As noted in the main text, we use α to denote the percentage length of a mass point along the whole path, and β to denote the percentage length of the same mass point within the particular line segment that the mass point is on. For convenience, at startup, we convert α into its corresponding Eulerian coordinate s. For example, if $\alpha = 0.5$ for a mass point, then we set s for this mass point to be $L/2$, where L is the total length of the path. Then at runtime, β can be computed from s as

$$\beta = \frac{s - s_0}{s_1 - s_0}, \tag{15}$$

where s_0 and s_1 are the Eulerian coordinates of the two path points that contain the mass point.

Given β, we can compute the world position of the mass point as:

$${}^W\mathbf{x}_\alpha = (1-\beta)\,{}^W\mathbf{x}_0 + \beta\,{}^W\mathbf{x}_1. \tag{16}$$

The time derivative of β is

$$\dot{\beta} = -\frac{1}{\Delta s}\left((1-\beta)\dot{s}_0 + \beta\dot{s}_1\right), \tag{17}$$

where $\Delta s = s_1 - s_0$, so the world velocity of the mass point becomes:

$${}^W\dot{\mathbf{x}}_\alpha = (1-\beta)\,{}^W\dot{\mathbf{x}}_0 + \beta\,{}^W\dot{\mathbf{x}}_1 - \frac{\Delta\mathbf{x}}{\Delta s}\left((1-\beta)\dot{s}_0 + \beta\dot{s}_1\right), \tag{18}$$

where $\Delta\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_0$. This was derived previously by Sueda et al. [2011]. To ease the derivation of the time derivative, we define the deformation gradient of the line segment as:

$$\mathbf{F} = \frac{\Delta\mathbf{x}}{\Delta s}. \tag{19}$$

Rearranging Eq. 18, we obtain:

$${}^W\dot{\mathbf{x}}_\alpha = \underbrace{\begin{pmatrix}(1-\beta)\mathbf{I} & \beta\mathbf{I} & -(1-\beta)\mathbf{F} & -\beta\mathbf{F}\end{pmatrix}}_{\mathbf{J}_{\alpha z}} \begin{pmatrix} {}^W\dot{\mathbf{x}}_0 \\ {}^W\dot{\mathbf{x}}_1 \\ \dot{s}_0 \\ \dot{s}_1 \end{pmatrix}. \tag{20}$$

The time derivative, $\dot{\mathbf{J}}_{\alpha z}$, requires $\dot{\beta}$ (Eq. 17) as well as $\dot{\mathbf{F}}$. The time derivative of the deformation gradient is

$$\dot{\mathbf{F}} = -\frac{1}{\Delta s}\left(\Delta\dot{\mathbf{x}} + \mathbf{F}\Delta\dot{s}\right). \tag{21}$$

1.2 Type III: Wrapping Surface Muscles

1.2.1 *Derivation of $\dot{\mathbf{J}}_{base}$.*

The two quantities in $\mathbf{J}_{\text{base}}$ that we must take the time derivative of are ${}^W_B\mathbf{R}$ and $\Gamma({}^B_S\mathbf{E}\,{}^S\mathbf{x}_\alpha)$.

For a body with rotation matrix $\mathbf{R}$ and angular velocity ω in body space, we can take the time derivative (see Murray et al. [2017]):

$$\dot{\mathbf{R}} = \mathbf{R}[\omega], \tag{22}$$

and so ${}^W_B\dot{\mathbf{R}} = {}^W_B\mathbf{R}[\omega_B]$.

To compute the time derivative of $\Gamma({}^B_S\mathbf{E}\,{}^S\mathbf{x}_\alpha)$, we only need the time derivative of ${}^S\mathbf{x}_\alpha$, since ${}^B_S\mathbf{E}$ is constant.

$$\dot{\Gamma}({}^B_S\mathbf{E}\,{}^S\mathbf{x}_\alpha) = \begin{pmatrix} [{}^B_S\mathbf{R}\,{}^S\dot{\mathbf{x}}_\alpha]^\top & \mathbf{0} \end{pmatrix}. \tag{23}$$

For this *base* Jacobian, ${}^S\mathbf{x}_\alpha$ is the position of the muscle mass point assuming for a moment that it is tied to the surface S. Therefore, we can compute its time derivative as:

$${}^S\dot{\mathbf{x}}_\alpha = {}^S_B\mathbf{R}\,\Gamma({}^B_S\mathbf{E}\,{}^S\mathbf{x}_\alpha)\,\phi_B. \tag{24}$$

1.2.2 *Derivation of $\dot{\mathbf{J}}_{ori}$ and $\dot{\mathbf{J}}_{ins}$.*

For $\dot{\mathbf{J}}_{\text{ori}}$, we need the time derivatives of ${}^W_S\mathbf{R}$, ${}^S_W\mathbf{R}$, ${}^W_A\mathbf{R}$, $\Gamma({}^A\mathbf{x}_{\text{ori}})$, and $\Gamma({}^B_A\mathbf{E}\,{}^A\mathbf{x}_{\text{ori}})$.

Assuming that the surface is attached to body B, The rotation matrix ${}^W_S\mathbf{R}$ is

$${}^W_S\mathbf{R} = {}^W_B\mathbf{R}\,{}^B_S\mathbf{R}. \tag{25}$$

The surface does not move with respect to the body, so using Eq. 22,

$${}^W_S\dot{\mathbf{R}} = {}^W_B\mathbf{R}[\omega_B]\,{}^B_S\mathbf{R}. \tag{26}$$

The time derivative of the inverse rotation is simply the transpose of the time derivative:

$${}^S_W\dot{\mathbf{R}} = {}^W_S\dot{\mathbf{R}}^\top. \tag{27}$$

Next, using Eq. 22 again,

$${}^W_A\dot{\mathbf{R}} = {}^W_A\mathbf{R}[\omega_A]. \tag{28}$$

The position of the origin with respect to A is fixed, so

$$\dot{\Gamma}({}^A\mathbf{x}_{\text{ori}}) = 0. \tag{29}$$

However, the position of the origin with respect to B changes over time, since ${}^B_A\mathbf{E} = {}^W_B\mathbf{E}^{-1}\,{}^W_A\mathbf{E}$ changes over time:

$${}^B_A\dot{\mathbf{E}} = \frac{d}{dt}\{{}^W_B\mathbf{E}^{-1}\}\,{}^W_A\mathbf{E} + {}^W_B\mathbf{E}^{-1}\frac{d}{dt}\{{}^W_A\mathbf{E}\}. \tag{30}$$

Using the inverse derivative identity and the fact that $\dot{\mathbf{E}} = \mathbf{E}[\phi]$ (see [Murray et al. 2017]), we have, after some rearranging:

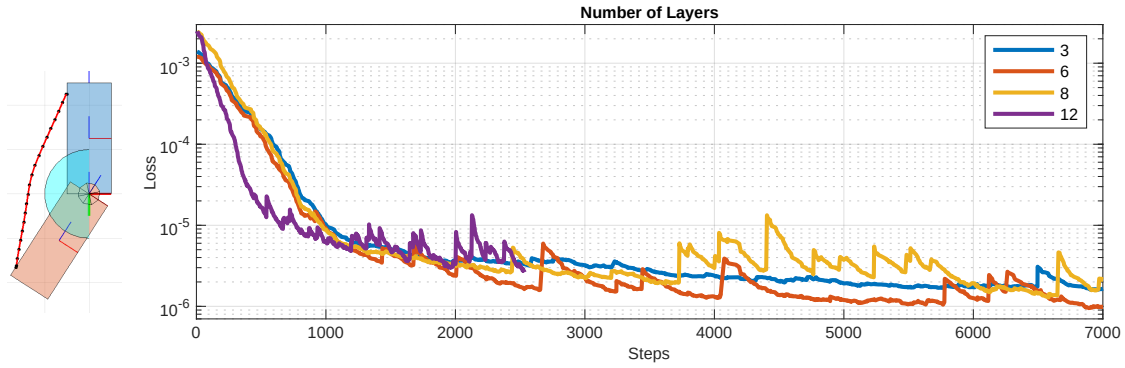
$${}^B_A\dot{\mathbf{E}} = {}^B_A\mathbf{E}[\phi_A] - [\phi_B]\,{}^B_A\mathbf{E}. \tag{31}$$

Therefore, the time derivative of $\Gamma({}^B_A\mathbf{E}\,{}^A\mathbf{x}_{\text{ori}})$ is

$$\dot{\Gamma}({}^B_A\mathbf{E}\,{}^A\mathbf{x}_{\text{ori}}) = \begin{pmatrix} \left[({}^B_A\mathbf{E}[\phi_A] - [\phi_B]\,{}^B_A\mathbf{E})\,{}^A\mathbf{x}_{\text{ori}}\right]^\top & \mathbf{0} \end{pmatrix}. \tag{32}$$

2 SAMPLING & NETWORK ARCHITECTURES

In this section, we provide further experiments with the sampling threshold and the number of layers for the neural network for Type III muscles. To eliminate the Jacobian discontinuity, we throw away data samples near sharp features. By throwing away more data samples around sharp features, it becomes easier for the network to converge. However, if we throw away too many samples, the resulting simulation produces obvious artifacts, such as the one shown in Fig. 2a, where the threshold was set to 20% (l/L in §3.3.1 in the main text). If we set the threshold to 10%, there remains some small artifacts. Therefore, we set the threshold to 1%, which we empirically determined to give almost negligible artifacts while still giving us acceptable convergence.



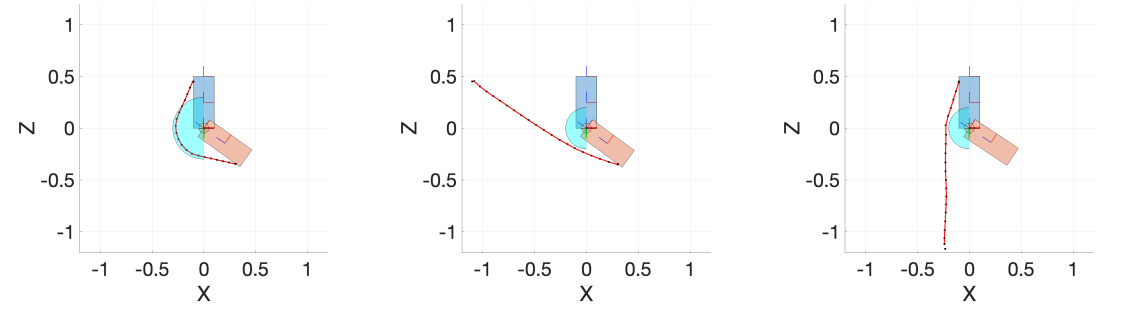
(a) (b)

Fig. 2. (a) When the threshold is set to 20%, the simulation produces noticeable artifacts near the Jacobian discontinuity. (b) Convergence plots with 3 (blue), 6 (red), 8 (yellow), and 12 (purple) layers.

With the threshold fixed at 1%, we tried training the network with 3, 6, 8, and 12 layers. As shown in Fig. 2b, all four networks converged adequately well. We chose to use a network with 6 layers, since it gave us a good tradeoff between speed and accuracy; however, with more hyperparameter tweaking, fewer layers may become more suitable.

3 FAILURE CASES

If we use values outside the training range, the network is not able to generate good results. Fig. 3 shows three such examples. In Fig. 3a, the runtime radius is outside of the training range. The resulting muscle path visibly penetrates the wrapping surface. In Fig. 3b, the origin point is outside of the training range. The resulting muscle path is visibly curved. In Fig. 3c, the insertion point is outside of the training range. The resulting muscle path is visibly curved and irregular.

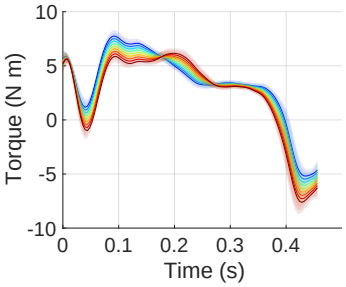


(a) Radius (b) Origin (c) Insertion

Fig. 3. Three failure cases: (a) The network was trained on radius between 0.01 and 0.2. Here, the network fails for a radius value of 0.25. (b) The network was trained on origin between $(-1.0, -1.0, -1.0)$ and $(1.0, 1.0, 1.0)$ in the coordinate space of the wrapping surface. Here, the network fails for an origin value of $(-1.1, 1.1, 0.45)$. (c) The network was trained on insertion between $(-1.0, -1.0, -1.0)$ and $(1.0, 1.0, 1.0)$ in the coordinate space of the wrapping surface. Here, the network fails for an insertion value of $(-0.22, 1.17, -1.1)$. Note that the $\alpha = 1$ point clearly deviates from its intended position.

4 GRACEFUL DEGRADATION

In §4.5 of the main text, we showed that the inverse dynamics output computed by our simulator matches the output computed with OpenSim [Delp et al. 2007], if we remove the muscle mass. We then showed that if we put 80% of the segment mass into the muscles, we obtain a torque value that is different by as much as 40%. In the inset figure, we plot the output of our simulator as we smoothly vary the muscle mass percentage from 0% (dark blue) to 80% (dark red), showing graceful degradation of our simulator to OpenSim.



REFERENCES

Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. 2007. OpenSim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering* 54, 11 (2007), 1940–1950.

Richard M Murray, Zexiang Li, and S Shankar Sastry. 2017. *A mathematical introduction to robotic manipulation.* CRC press.

Prashant Sachdeva, Shinjiro Sueda, Susanne Bradley, Mikhail Fain, and Dinesh K. Pai. 2015. Biomechanical Simulation and Control of Hands and Tendinous Systems. *ACM Trans. Graph.* 34, 4, Article 42 (Jul. 2015), 10 pages.

Shinjiro Sueda, Garrett L. Jones, David I. W. Levin, and Dinesh K. Pai. 2011. Large-Scale Dynamic Simulation of Highly Constrained Strands. *ACM Trans. Graph.* 30, 4, Article 39 (Jul. 2011), 10 pages.