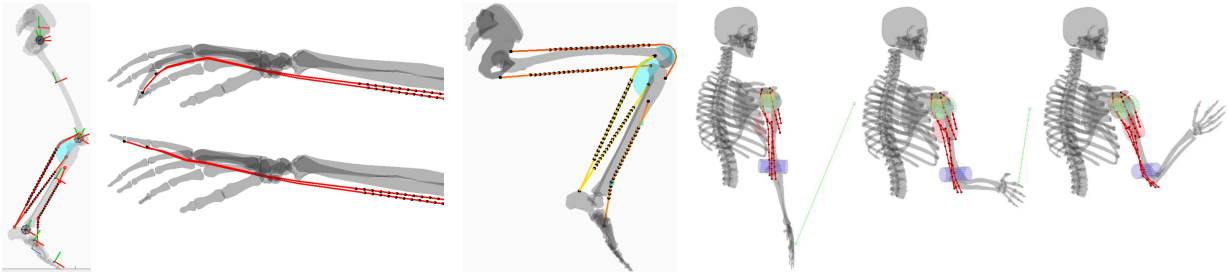
Differentiable Simulation of Inertial Musculotendons

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



(a) (b) (c) (d)

Fig. 1. Muscle inertia (a) changes the inverse dynamics result of running motion by up to 40%, and (b) stabilizes the simulation. Our framework (c) handles Hill-type muscles, complex joints, and higher-order integration, and (d) works flawlessly with the adjoint method for computing the simulation derivatives.

We propose a simple and practical approach for incorporating the effects of muscle inertia, which has been ignored by previous musculoskeletal simulators in both graphics and biomechanics. We approximate the inertia of the muscle by assuming that muscle mass is distributed along the centerline of the muscle. We express the motion of the musculotendons in terms of the motion of the skeletal joints using a chain of Jacobians, so that at the top level, only the reduced degrees of freedom of the skeleton are used to completely drive both bones and musculotendons. Our approach can handle all commonly used musculotendon path types, including those with multiple path points and wrapping surfaces. For muscle paths involving wrapping surfaces, we use neural networks to model the Jacobians, trained using existing wrapping surface libraries, which allows us to effectively handle the Jacobian discontinuities that occur when musculotendon paths collide with wrapping surfaces. We demonstrate support for higher-order time integrators, complex joints, inverse dynamics, Hill-type muscle models, and differentiability. In the limit, as the muscle mass is reduced to zero, our approach gracefully degrades to traditional simulators without support for muscle inertia. Finally, it is possible to mix and match inertial and non-inertial musculotendons, depending on the application.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; **Computer graphics**; **Neural networks**.

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.

0730-0301/2022/12-ART272 $15.00
https://doi.org/10.1145/3550454.3555490

Additional Key Words and Phrases: Biomechanics, Muscles, Tendons, Bones, Skeleton, Musculoskeletal, Musculotendon, Neural Networks

ACM Reference Format:
Ying Wang, Jasper Verheul, Sang-Hoon Yeo, Nima Khademi Kalantari, and Shinjiro Sueda. 2022. Differentiable Simulation of Inertial Musculotendons. *ACM Trans. Graph.* 41, 6, Article 272 (December 2022), 11 pages. https://doi.org/10.1145/3550454.3555490

1 INTRODUCTION

Computer animation researchers have been using and extending muscle-driven skeletal simulations for many applications—for example, for improved inverse kinematics [Komura et al. 2001], head/neck animation, [Lee and Terzopoulos 2006], hand animation [Sueda et al. 2008], real-time visualization of muscle activations [Murai et al. 2010], energy-minimizing gait animations [Wang et al. 2012], creation of imaginary bipedal characters, [Geijtenbeek et al. 2013], upper body animations [Lee et al. 2009; Si et al. 2015], and control of characters under various anatomical conditions [Lee et al. 2014; Lee et al. 2019]. However, almost all musculoskeletal simulators used in graphics and biomechanics ignore the effect of the inertia of the muscles as they slide with respect to the bones. Instead, the mass of the muscles is "lumped" to the bones at rest pose, and so the effect of the muscle inertia cannot be reflected in the dynamics of the system, even though around 40% of total body mass comes from skeletal muscles [Marieb and Hoehn 2010].

Missing inertia can change some important aspects of the simulation. The effect of the missing inertia is most pronounced when the muscle mass is large and far from the joints it acts on. For example, some of the muscles of the lower limb exhibit significant inertial effects. In the seminal paper, Pai [2010] notes that the triceps surae muscle of the human ankle can account for an additional 7.6% of the effective inertia of the joint. In §4.5 (Fig. 1a), we also show that

the combined effect of the muscle mass alters the inverse dynamics result of running motion by as much as 40%. As another example, consider the extrinsic muscles of the hand, which are located in the forearm (Fig. 1b & §4.4). The joints of the finger have very small inertia by themselves, but when the muscle masses are taken into account, the joint inertia increases significantly. With a traditional musculoskeletal simulator, these muscle masses are absorbed into the nearest segment (*i.e.,* forearm) and do not affect the inertia of the finger joints, whereas with our approach, these masses are coupled to all of the joints spanned by the musculotendons. This increase in inertia is important not only for simulation accuracy but also stability. If we apply an impulse to the fingertip (*e.g.,* flicking with the other hand), the distal joint quickly becomes unstable due to its small inertia, but if the effect of muscle inertia is taken into account, it remains stable under an impulse several times larger. Joint damping can be added to overcome some of these issues, but this would require manual tweaking of parameters, and the added damping would help stabilize both the simulation with and without muscle inertia. Furthermore, the muscle inertia provides coupling of the joints, naturally preventing the joints from moving independently.

In the past few years, biomechanics researchers have proposed techniques to deal with muscle inertia [Han et al. 2015; Guo et al. 2020], but these approaches can only be used for relatively simple muscle paths. We therefore propose a framework for incorporating the effects of muscle inertia for more complex muscle path types, including those with wrapping surfaces. To maximize interoperability with existing musculoskeletal simulators (*e.g.,* [Damsgaard et al. 2006; Seth et al. 2018]), we use the reduced coordinates of the articulated rigid body system representing the skeletal joints as the degrees of freedom. However, unlike existing musculoskeletal simulators, we take into account the inertia of the muscles as they slide with respect to the bones, by inserting mass points along the paths of the musculotendons. As the skeleton moves, these mass points move; since each musculotendon is assumed to be frictionless, the path moves such that its length is minimized.

Our main technical contribution is the derivation of this mapping (*i.e.,* Jacobian, plus its time derivative) from the skeletal motion to the muscle mass motion. To aid us in the derivation, we categorize musculotendon paths into three types (Fig. 2):

- I: Straight-line paths, whose Jacobians are derived in a straightforward manner (§3.1).
- II: Polyline paths through a sequence of points, whose Jacobians are derived by extending the Eulerian-on-Lagrangian framework [Sueda et al. 2011; Sachdeva et al. 2015] (§3.2).
- III: All others, but most importantly, curved paths wrapping over smooth surfaces, whose Jacobians are based on neural networks trained with our custom sampling strategy to handle parasitic discontinuities (§3.3).

To summarize, our contributions are:

- An Eulerian-on-Lagrangian approach for the inertia of polyline musculotendons composed of a sequence of path points.
- A neural network approach for the inertia of curved musculotendons wrapping over smooth surfaces.
- A framework compatible with various existing techniques, including higher-order integrators, inverse dynamics, Hill-type muscle models, and differentiability.

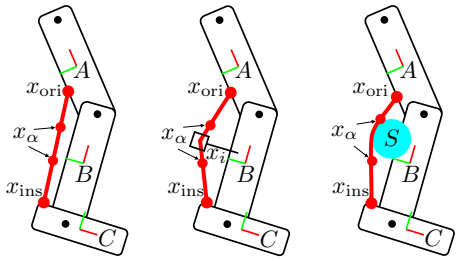


Fig. 2. Concrete running example for Types I, II, and III muscles. In all cases, there are three bones and one muscle. The origin is on body A, and the insertion is on body C. Type II muscle has a path point on body B, and Type III muscle has a wrapping surface S defined with respect to body B.

- A framework capable of handling musculotendons with inertia but can, in the limit, reproduce the results from existing simulators without inertia.
- A framework with support for mixing and matching of inertial and non-inertial muscles, so that the user can choose to add inertia only to muscles with substantial inertial effects.

2 RELATED WORK

Because of the importance of human character animation to graphics, many different types of approaches have been studied, starting with the seminal work on facial animation [Waters 1987; Terzopoulos and Waters 1990; Waters and Terzopoulos 1990]. Often in graphics, the causal relationship between the muscles and the bones is switched—the skeleton is first moved, and then the muscles/flesh are correspondingly simulated to add bulging effects to the character's skin [Scheepers et al. 1997; Wilhelms and Gelder 1997; Kim and Pollard 2011]. As important as these works are to graphics (*e.g.,* commercial products [Autodesk 2011; Ziva Dynamics 2018]), this paper focuses exclusively on muscle-driven systems.

Line-based musculoskeletal methods were developed by adding line-of-action muscles to rigid body dynamics from robotics [Damsgaard et al. 2006; Seth et al. 2018]. Almost always, these muscles are assumed to be massless, taking the shortest path between the origin and insertion, possibly being routed around path points and wrapping surfaces. Perhaps the first work in computer graphics to use proper biomechanics-based muscle models is the work by Komura et al. [1997; 2000; 2001], in which they show new types of animations, such as biomechanically based fatigue, which were not possible with previous joint torque-based approaches. Lee and Terzopoulos [2006] use line-based musculotendons to model the muscles of the neck, and in their follow-up works, they use these muscles to drive the volumetric mesh for upper-body motion [Lee et al. 2009] and swimming [Si et al. 2015]. Wang et al. [2012] simulate a variety of gaits, showing that optimizing for metabolic energy expenditure increases the realism of resulting animations. Geijtenbeek et al. [2013] use Hill-type muscle models for a range of bipedal characters, including humans, animals, and imaginary creatures. Unlike previous work, they also optimize for the placement and routing of these muscle lines so that the total error based on speed, orientation, and effort is minimized. Lee et al. [2014] propose a scalable biped controller that is able to solve for the activations of

more than one hundred muscles. Their controller is formulated as a quadratic program that can handle frictional contact based on Coulomb's model. Their results include motions that include muscle pain, muscle tightness, or joint dislocation. In their follow-up work, Lee et al. [2019] use deep reinforcement learning to control more than three hundred Hill-type muscles for full-body motions. They show that they can reproduce a wide range of motions, including muscle weakness, use of prostheses, and pathological gaits.

Although not directly related, we briefly cover volume-based muscle models because of their importance to graphics. Among those that do use biomechanically based muscle mechanics models, two subtypes of volume-based methods have been studied. The first subtype—those with embedded force generators— was initially used in animation. Chen and Zeltzer [1992] introduced the first biomechanics-based muscle mechanics model to computer animation. They used the finite element method (FEM) with twenty-node isoparametric brick elements, with the longitudinal edges of these elements acting as muscle force generators. Later, Zhu et al. [1998] used eight-node brick elements with force generators between a set of linear FEM nodes. Lemos et al. [2001] developed a general FEM framework that could support any nonlinear material as the background isotropic material. Ng-Thow-Hing [2001] used a similar approach to embed force generators inside a B-spline solid. Around the turn of the century, the second subtype—those with anisotropic muscle material models—became more popular in graphics. The seminal work by Teran et al. [2003] used a material model with a strain energy that includes an anisotropic muscle potential term. Similar muscle mechanics model is used in their follow-up work on larger scale simulation of skeletal muscles [Teran et al. 2005] as well as facial muscles [Sifakis et al. 2005]. Fan et al. [2014] used a blackbox deformation energy as an approximation for contractile mechanics in their volumetric muscles undergoing contact. Recently, Lee et al. [2018] simulated volumetric muscles with Projective Dynamics, driven by per-element energy functions derived from a Hill-type muscle model. Min et al. [2019] used quadratic strain energy to model contractile volumetric muscles of soft-bodied animals. Although in principle it is possible to use these volumetric simulators to compute the inertial effects of the muscle, they are *impractical or impossible* for the types of applications we are interested in, considering the high number of parameters and the computational complexity required by volumetric models.

3 METHODS

We use the reduced coordinates, $\mathbf{q}_r$, of the articulated rigid body system representing the skeletal joints as the degrees of freedom (DOFs) of the system. To take into account the inertia of the muscles as they slide with respect to the bones, we insert mass points along the path of the musculotendon. These mass points are *fixed* at a certain percentage length α along the path (*i.e.,* fixed at certain texture coordinates; see Fig. 3b); however, as the skeleton moves, these mass points *move in world space*, since each musculotendon is assumed to be frictionless—the path moves such that its length is minimized.

In this section, we derive the Jacobian $\mathbf{J}_{\alpha r}$ that maps the change in the reduced coordinates of the articulated rigid body system to the change in the 3D world coordinates of these muscle mass points:

$$\dot{\mathbf{x}}_\alpha = \mathbf{J}_{\alpha r}\dot{\mathbf{q}}_r, \tag{1}$$

where $\dot{\mathbf{q}}_r$ is the stacked vector of reduced (joint) velocities, and $\dot{\mathbf{x}}_\alpha$ is the stacked vector of muscle mass point velocities in world space. The size of $\dot{\mathbf{q}}_r$ depends on the joint types. For example, if all of the joints are revolute, then $\dot{\mathbf{q}}_r \in \mathbb{R}^n$, and if all of the joints are spherical, then $\dot{\mathbf{q}}_r \in \mathbb{R}^{3n}$, where n is the number of joints. The multiplication by the Jacobian $\mathbf{J}_{\alpha r}$, which depends nonlinearly on $\mathbf{q}_r$, produces the 3D world velocities of muscle mass points $\dot{\mathbf{x}}_\alpha \in \mathbb{R}^{3m}$, where m is the number of mass points.

We assume that we already have access to the Jacobian $\mathbf{J}_{mr}$ (and its time derivative $\dot{\mathbf{J}}_{mr}$) that maps between the reduced (joint) velocities and the maximal (body) velocities of the articulated rigid body system [Kim and Pollard 2011; Wang et al. 2019]:

$$\dot{\mathbf{q}}_m = \mathbf{J}_{mr}\dot{\mathbf{q}}_r, \tag{2}$$

where $\dot{\mathbf{q}}_m$ is the stacked vector of maximal velocities. Unlike reduced velocities, the size of the maximal velocity vector does not depend on the joint type: $\dot{\mathbf{q}}_m \in \mathbb{R}^{6n}$. In our work, we stack the rotational velocity, ω, and the translational velocity, v, together to form the maximal velocity, so that for each body, we have:

$$\dot{\mathbf{q}}_m = \phi = \begin{pmatrix}\omega \\ v\end{pmatrix}, \tag{3}$$

with both ω and v expressed in body-local coordinates [Murray et al. 2017].[1] In the rest of this section, we sometimes use ϕ as an alternative symbol for the maximal velocity (twist) of a *single* body.

The main technical contribution of our work is the derivation of Jacobian $\mathbf{J}_{\alpha m}$ (and its time derivative $\dot{\mathbf{J}}_{\alpha m}$) that maps the maximal velocities to the muscle mass point velocities (details in §3.1, §3.2, and §3.3). Once this Jacobian is derived, to compute the world velocities of the muscle mass points from the reduced velocities of the joints, we chain it together with $\mathbf{J}_{mr}$ to form the final Jacobian we are after:

$$\mathbf{J}_{\alpha r} = \mathbf{J}_{\alpha m}\mathbf{J}_{mr}. \tag{4}$$

Armed with this Jacobian, we can compute the 3D world accelerations of the muscle mass points as:

$$\begin{aligned}\ddot{\mathbf{x}}_\alpha &= \dot{\mathbf{J}}_{\alpha r}\dot{\mathbf{q}}_r + \mathbf{J}_{\alpha r}\ddot{\mathbf{q}}_r \\ \dot{\mathbf{J}}_{\alpha r} &= \dot{\mathbf{J}}_{\alpha m}\mathbf{J}_{mr} + \mathbf{J}_{\alpha m}\dot{\mathbf{J}}_{mr}.\end{aligned} \tag{5}$$

Plugging this into the equations of motion of the mass points $\mathbf{M}_\alpha\ddot{\mathbf{x}}_\alpha = \mathbf{f}_\alpha$ and applying the principle of virtual work, we obtain:

$$\mathbf{J}_{\alpha r}^\top\mathbf{M}_\alpha\mathbf{J}_{\alpha r}\ddot{\mathbf{q}}_r = \mathbf{J}_{\alpha r}^\top\left(\mathbf{f}_\alpha - \mathbf{M}_\alpha\dot{\mathbf{J}}_{\alpha r}\dot{\mathbf{q}}_r\right). \tag{6}$$

Here, $\mathbf{M}_\alpha \in \mathbb{R}^{3m\times 3m}$ is the constant diagonal inertia matrix of the m muscle mass points, and $\mathbf{f}_\alpha \in \mathbb{R}^{3m}$ is the force of gravity acting on these mass points. The muscle activation forces do not directly apply forces to these mass points. Instead, in order to keep our framework compatible with existing biomechanical simulators, we assume that the activation forces are applied to the skeleton, which in turn kinematically moves the mass points through the Jacobian $\mathbf{J}_{\alpha r}$. The last term in Eq. 6, which uses $\dot{\mathbf{J}}_{\alpha m}$, is the quadratic velocity vector (QVV) that results from the partial derivatives of the kinetic energy [Shabana 2013].

[1] Other conventions can be used; the derivations will need to be accordingly modified.

The reduced coordinates of the system also drive the bones, and so combining muscles and bones, we obtain the final equations of motion of the whole *musculoskeletal* system in reduced coordinates:

$$\tilde{\mathbf{M}}_r \ddot{\mathbf{q}}_r = \tilde{\mathbf{f}}_r \tag{7a}$$

$$\tilde{\mathbf{M}}_r = \mathbf{J}_{\alpha r}^\top \mathbf{M}_\alpha \mathbf{J}_{\alpha r} + \mathbf{J}_{mr}^\top \mathbf{M}_m \mathbf{J}_{mr} \tag{7b}$$

$$\tilde{\mathbf{f}}_r = \mathbf{J}_{\alpha r}^\top \left(\mathbf{f}_\alpha - \mathbf{M}_\alpha \dot{\mathbf{J}}_{\alpha r} \dot{\mathbf{q}}_r\right) + \mathbf{J}_{mr}^\top \left(\mathbf{f}_m - \mathbf{M}_m \dot{\mathbf{J}}_{mr} \dot{\mathbf{q}}_r\right) + \mathbf{f}_r, \tag{7c}$$

where $\mathbf{M}_m \in \mathbb{R}^{6n \times 6n}$ is the constant diagonal inertia of the n bones,[2] $\mathbf{f}_m \in \mathbb{R}^{6n}$ is the sum of maximal forces acting on these bones, such as gravity, Coriolis, and muscle activation forces, and $\mathbf{f}_r$ is the sum of reduced forces, such as joint torques. We can use any time integrator to step the system forward in time. In our implementation, we use forward Euler, BDF1, and SDIRK2 [Hairer et al. 2006].

Throughout this section, we will use the concrete running example shown in Fig. 2. We will assume that each joint is a revolute joint, and so the reduced velocity is $\dot{\mathbf{q}}_r = (\dot{\theta}_A \ \dot{\theta}_B \ \dot{\theta}_C)^\top \in \mathbb{R}^3$. The maximal velocity is $\dot{\mathbf{q}}_m = (\dot{\phi}_A \ \dot{\phi}_B \ \dot{\phi}_C)^\top \in \mathbb{R}^{18}$, and $\mathbf{J}_{mr} \in \mathbb{R}^{18\times 3}$. The origin of the musculotendon is assumed to be on body A, and the insertion on body C. We will also assume that there is a single muscle with two mass points, so that $\dot{\mathbf{x}}_\alpha \in \mathbb{R}^6$, and $\mathbf{J}_{\alpha m} \in \mathbb{R}^{6\times 18}$. The final Jacobian is $\mathbf{J}_{\alpha r} \in \mathbb{R}^{6\times 3}$. For the Type II muscle, the path point is attached to body B. For the Type III muscle, the wrapping surface S is defined with respect to body B.

3.1 Type I: Straight Line Muscles

We start with the simple case of a straight line muscle between two bodies. This subsection is not a contribution, but the derivations and notations introduced here will help us with the rest of the paper.

To be explicit, for vectors, we will use a leading superscript to indicate which coordinate space the vector is defined in, and for matrices, we will use a leading sub/superscript to indicate from which to which space the matrix transforms a vector. Let ${}^A\mathbf{x}_{\text{ori}}$ be the 3D position of the origin in the local space of A, and ${}^C\mathbf{x}_{\text{ins}}$ be the 3D position of the insertion in the local space of C. Then the world velocities of the origin and insertion can be computed as:

$${}^W\dot{\mathbf{x}}_{\text{ori}} = {}^W_A\mathbf{R}\,\Gamma({}^A\mathbf{x}_{\text{ori}})\,\dot{\phi}_A, \quad {}^W\dot{\mathbf{x}}_{\text{ins}} = {}^W_C\mathbf{R}\,\Gamma({}^C\mathbf{x}_{\text{ins}})\,\dot{\phi}_C, \tag{8}$$

where ${}^W_X\mathbf{R} \in SO(3)$ is the rotation matrix of body X (*e.g.,* A or C), and $\Gamma(\mathbf{x}) = \begin{pmatrix}[\mathbf{x}]^\top & \mathbf{I}\end{pmatrix} \in \mathbb{R}^{3\times 6}$ is the material Jacobian matrix for computing the point velocity [Murray et al. 2017], with $[\cdot]$ the cross-product matrix. This gives us the following expression for the Jacobian between maximal velocities and world velocities of the origin/insertion for our concrete running example in Fig. 2:

$$\mathbf{J}_{xm} = \begin{pmatrix} {}^W_A\mathbf{R}\,\Gamma({}^A\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & {}^W_C\mathbf{R}\,\Gamma({}^C\mathbf{x}_{\text{ins}}) \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{9}$$

For a muscle mass point α, the world velocity is simply the weighted average of the world velocities of the origin and the insertion: ${}^W\dot{\mathbf{x}}_\alpha = (1-\alpha)\,{}^W\dot{\mathbf{x}}_{\text{ori}} + \alpha\,{}^W\dot{\mathbf{x}}_{\text{ins}}$. Thus, the Jacobian $\mathbf{J}_{\alpha x}$ is:

$$\mathbf{J}_{\alpha x} = \begin{pmatrix} (1-\alpha_1)\,\mathbf{I} & \alpha_1\,\mathbf{I} \\ (1-\alpha_2)\,\mathbf{I} & \alpha_2\,\mathbf{I} \end{pmatrix} \in \mathbb{R}^{6\times 6}, \tag{10}$$

[2]The maximal inertia is constant because of our choice of body-local coordinates.

where α_1 and α_2 are the percentage lengths of the two mass points. The product of these two Jacobians gives the final Jacobian for Type I muscles: $\mathbf{J}_{\alpha m} = \mathbf{J}_{\alpha x}\mathbf{J}_{xm} \in \mathbb{R}^{6\times 18}$.

The α value is fixed over time, as well as the origin and insertion positions with respect to their respective bodies. The time derivative of the Jacobian is then $\dot{\mathbf{J}}_{\alpha m} = \mathbf{J}_{\alpha x}\dot{\mathbf{J}}_{xm}$, where

$$\dot{\mathbf{J}}_{xm} = \begin{pmatrix} {}^W_A\dot{\mathbf{R}}\,[\omega_A]\Gamma({}^A\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & {}^W_C\dot{\mathbf{R}}\,[\omega_C]\Gamma({}^C\mathbf{x}_{\text{ins}}) \end{pmatrix}, \tag{11}$$

since $\dot{\mathbf{R}} = \mathbf{R}[\omega]$ for maximal velocities in body coordinates [Murray et al. 2017].

3.2 Type II: Path Point Muscles

Some musculotendons are constructed as a polyline going through a sequence of path points. To deal with these types of muscles, we extend the Eulerian-on-Lagrangian (EOL) strands framework [Sueda et al. 2011; Sachdeva et al. 2015]. Let $i = 0, 1, 2, \cdots, n+1$ be the indices of the path points (so that $i = 0$ corresponds to the origin, $i = n+1$ corresponds to the insertion, and there are n internal path points). With the EOL framework, we keep track of not only the world space position and velocity (Lagrangian quantities $\mathbf{x}_i$ and $\dot{\mathbf{x}}_i \in \mathbb{R}^3$) of the path points, but also the reference space position and velocity (Eulerian quantities s_i and $\dot{s}_i \in \mathbb{R}$) at these path points. This allows us to model the sliding motion of the underlying strand even when the world positions of the path points are fixed (*e.g.,* if $\dot{\mathbf{x}}_i = 0$ but $\dot{s}_i \neq 0$, the musculotendon material still moves in world space). Following the work by Sachdeva et al. [2015], we assume that all of the line segments of the polyline share the same strain value, which allows us to derive a Jacobian that maps from $\dot{\mathbf{x}}_i$ to $\dot{s}_i$ (see Eq. 3 [Sachdeva et al. 2015]):

$$\mathbf{J}_{sx} = -\mathbf{L}^{-1}\Delta\mathbf{S}\,\Delta\bar{\mathbf{X}}, \tag{12}$$

where $\Delta\mathbf{S}$ is a matrix constructed from the Eulerian coordinates s_i, $\Delta\bar{\mathbf{X}}$ is a matrix constructed from the Lagrangian coordinates $\mathbf{x}_i$, and $\mathbf{L}$ is constructed from the segment lengths between the path points.

Since Sachdeva et al. [2015] used *inextensible* EOL strands, they did not need to derive the time derivative of this Jacobian. However, in this work, the EOL strands are used for *extensible* musculotendons; therefore, we must also derive $\dot{\mathbf{J}}_{sx}$. Using the inverse derivative identity for $\mathbf{L}$, we obtain:

$$\dot{\mathbf{J}}_{sx} = -\mathbf{L}^{-1}\left(\dot{\mathbf{L}}\mathbf{J}_{sx} + \Delta\dot{\mathbf{S}}\,\Delta\bar{\mathbf{X}} + \Delta\mathbf{S}\,\Delta\dot{\bar{\mathbf{X}}}\right). \tag{13}$$

Further details are in the supplementary document.

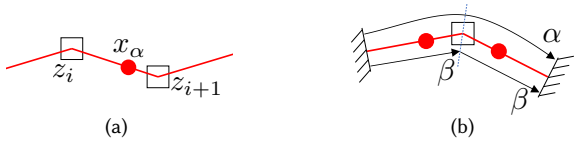


Fig. 3. (a) An EOL segment: the motion of the mass point $\mathbf{x}_\alpha$ depends on the motion of both Eulerian and Lagrangian motions of the path points $\mathbf{z}_i$ and $\mathbf{z}_{i+1}$. (b) A musculotendon with one path point between origin and insertion: α represents the percentage length along the whole musculotendon, whereas β represents the percentage length along each line segment.

So far, the Jacobians $\mathbf{J}_{sx}$ and $\dot{\mathbf{J}}_{sx}$ that we derived cannot be plugged into our system because they only map between $\dot{\mathbf{x}}_i$ and $\dot{s}_i$, rather than from $\dot{\mathbf{q}}_m$ to $\dot{\mathbf{x}}_\alpha$. In other words, these Jacobians only provide the mapping between the Lagrangian and Eulerian velocities of the path points of a musculotendon, rather than the mapping between the maximal velocities of the skeleton and the muscle mass point velocities. To tie the Jacobians $\mathbf{J}_{sx}$ and $\dot{\mathbf{J}}_{sx}$ to the rest of the system, we introduce a new notation $\mathbf{z}$ that represents the combined Lagrangian/Eulerian coordinates:

$$\mathbf{z}_i = \begin{pmatrix} \mathbf{x}_i \\ s_i \end{pmatrix} \in \mathbb{R}^4. \tag{14}$$

In the concrete example in Fig. 2, which contains a single internal path point, $\mathbf{z} = (\mathbf{x}_{\text{ori}}\ s_{\text{ori}}\ \mathbf{x}_1\ s_1\ \mathbf{x}_{\text{ins}}\ s_{\text{ins}})^\top \in \mathbb{R}^{12}$. The musculotendon material cannot flow past the origin or insertion, so $\dot{s}_{\text{ori}}$ and $\dot{s}_{\text{ins}}$ are always zero. Using this notation, the Jacobian that we are after can be written as:

$$\begin{aligned} \mathbf{J}_{\alpha m} &= \mathbf{J}_{\alpha z}\mathbf{J}_{zm} \in \mathbb{R}^{6\times 18} \\ \dot{\mathbf{J}}_{\alpha m} &= \dot{\mathbf{J}}_{\alpha z}\mathbf{J}_{zm} + \mathbf{J}_{\alpha z}\dot{\mathbf{J}}_{zm}. \end{aligned} \tag{15}$$

The left Jacobian $\mathbf{J}_{\alpha z} \in \mathbb{R}^{6\times 12}$ represents the mapping from the Lagrangian/Eulerian velocities of the path points to the muscle mass point (Fig. 3a). This was already derived by Sueda et al. [2011] (Eq. 4), but we reproduce the expression here, for our concrete example with one path point and two mass points. The first mass point is between the origin and the path point, and the second mass point is between the path point and the insertion. Therefore, we get:

$$\mathbf{J}_{\alpha z} = \begin{pmatrix} (1-\beta_1)\mathbf{I} & -(1-\beta_1)\mathbf{F}_1 & \beta_1\mathbf{I} & -\beta_1\mathbf{F}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (1-\beta_2)\mathbf{I} & -(1-\beta_2)\mathbf{F}_2 & \beta_2\mathbf{I} & -\beta_2\mathbf{F}_2 \end{pmatrix}. \tag{16}$$

Here, we used β to represent the percentage location of $\mathbf{x}_\alpha$ *within* a particular line segment, as shown in Fig. 3b. $\mathbf{F} \in \mathbb{R}^3$ is the deformation gradient of the line segment: $\mathbf{F}_1 = (\mathbf{x}_1 - \mathbf{x}_{\text{ori}})/(s_1 - s_{\text{ori}})$ and $\mathbf{F}_2 = (\mathbf{x}_{\text{ins}} - \mathbf{x}_1)/(s_{\text{ins}} - s_1)$. The time derivatives of these quantities, which were not derived before by Sueda et al. [2011], are nevertheless needed for our extensible musculotendons. We list the detailed derivations of these derivatives in the supplementary document.

The right Jacobian $\mathbf{J}_{zm} \in \mathbb{R}^{12\times 18}$ in Eq. 15 represents the mapping from the maximal velocities of the bodies to the Lagrangian/Eulerian velocities of the path points. This can be accomplished by constructing a Jacobian that passes through the Lagrangian components while hitting the Eulerian components by $\mathbf{J}_{sx}$:

$$\mathbf{J}_{zm} = \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{sx} \end{pmatrix}\mathbf{J}_{xm}, \quad \dot{\mathbf{J}}_{zm} = \begin{pmatrix} \mathbf{0} \\ \dot{\mathbf{J}}_{sx} \end{pmatrix}\mathbf{J}_{xm} + \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{sx} \end{pmatrix}\dot{\mathbf{J}}_{xm}. \tag{17}$$

$\mathbf{J}_{xm}$ in our concrete example with an internal path point $\mathbf{x}_i$ attached to body B is:

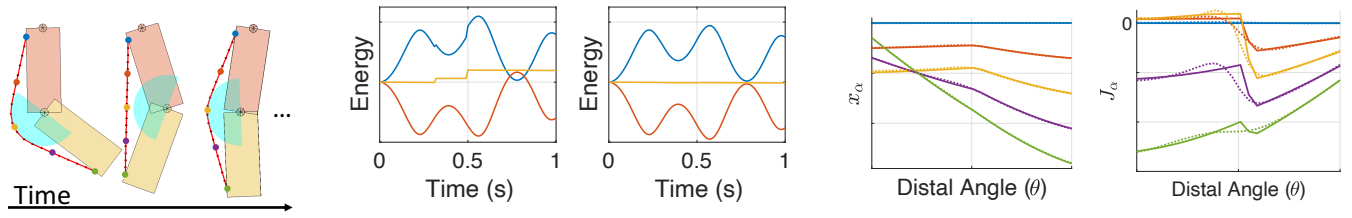
$$\mathbf{J}_{xm} = \begin{pmatrix} {}^W_A\mathbf{R}\,\Gamma({}^A\mathbf{x}_{\text{ori}}) & {}^W_B\mathbf{R}\,\Gamma({}^B\mathbf{x}_i) & \mathbf{0} \\ \mathbf{0} & {}^W_B\mathbf{R}\,\Gamma({}^B\mathbf{x}_i) & {}^W_C\mathbf{R}\,\Gamma({}^C\mathbf{x}_{\text{ins}}) \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{18}$$

Its time derivative, $\dot{\mathbf{J}}_{xm}$, can be derived similarly as in Eq. 11.

3.3 Type III: Wrapping Surface Muscles

Some musculotendons are constructed as 3D paths that wrap around smooth surfaces. To derive the Jacobians for these types of paths, we use neural networks. The reason for using neural networks may not be immediately obvious, since existing muscle routing algorithms are highly efficient [Garner and Pandy 2000; Scholz et al. 2016; Seth et al. 2018; Lloyd et al. 2020]. With some fairly minor modifications, we could use the output of these libraries to compute the Jacobians with finite differencing, which would not be prohibitively expensive due to the efficiency of these libraries. However, they cannot be used directly in our framework for inertial muscles because they all suffer from a massive problem: *Jacobian discontinuity*.

As an illustration of this problem, suppose that we have a double pendulum with a musculotendon shown in Fig. 4a. As the pendulum swings due to the force of gravity acting on both the bones and the musculotendon, the path of the musculotendon attaches and detaches from the wrapping surface. If we use a Jacobian computed using existing wrapping surface libraries and finite differencing, we observe discontinuities in the energy plot, as shown in Fig. 4b. These energy jumps occur because the velocities of the muscle mass points undergo sudden changes, even when the velocities of the joints vary smoothly. Fig. 4d shows the x-component of five of the mass points (each with its own color), as a function of the distal joint angle, zoomed in near a discontinuity. The values computed with an existing wrapping surface library are shown with solid lines, and ours with dotted lines. Fig. 4e shows the corresponding derivatives. The jump in the value of the Jacobian creates sudden changes in the



(a) Detaching & attaching muscle (b) With library (c) Ours (d) Positions (e) Derivatives

Fig. 4. (a) A double pendulum with a musculotendon, hitting a wrapping surface. (b) Energy plot (kinetic in blue, potential in red, total in yellow) of the simulation using an existing wrapping surface library. (c) Energy plot using our approach. (d) Plot of the x-component of five selected muscle mass points as a function of the distal joint angle, zoomed around a discontinuity. The solid lines are generated using an existing wrapping surface library. The dotted lines are generated using our approach. (e) The corresponding plots of the Jacobian. Unlike previous work (solid), our approach (dotted) generates smooth Jacobians.

velocities of the mass points, which in turn creates energy jumps in the simulation. On the other hand, our neural network approach generates the smooth Jacobian plots in Fig. 4e, while keeping the position plots in Fig. 4d virtually indistinguishable from the output of the library code. This results in a smooth energy trajectory shown in Fig. 4c.

One way to deal with the discontinuity is to detect these sudden state changes and apply a manual fix, *e.g.*, by computing the pre- and post-collision Jacobians and running a nonlinear optimization to compute the velocities that minimize the change in energy. However, such approaches are tricky to incorporate into implicit integrators, such as SDIRK2 [Hairer et al. 2006], as well as into differentiable simulation techniques, such as the adjoint method [McNamara et al. 2004; Geilinger et al. 2020; Xu et al. 2021], which our method supports naturally without any changes to the framework.

We instead choose to smooth the discontinuity. Smoothing would be easy with a uni-articular muscle spanning a hinge joint. As an offline process, we could pre-sample many points within the range of motion of the joint, and then apply a smoothing filter over the samples. During runtime, we could then use the filtered values to construct the Jacobian. However, high-dimensional smoothing would be required with a bi- or multi-articular muscle, as well as with a uni-articular muscle with a spherical joint. Therefore, we use neural networks for this high-dimensional smoothing problem. This approach is simple to implement and can be used with any existing muscle routing libraries.

3.3.1 Training the Network. We train the network with origin and insertion positions as the input, rather than the joint angle. This is an important choice, since it allows the same trained network to be used regardless of the type of the joints, how many joints the musculotendon spans, as well as with respect to which bodies the surface is defined. Using the cylinder wrapping surface as a concrete example, the input and output of our network are:

$$\begin{pmatrix} {}^{S}\mathbf{x}_{\text{ori}} \\ {}^{S}\mathbf{x}_{\text{ins}} \\ \alpha \\ r \end{pmatrix} \rightarrow \left({}^{S}\mathbf{x}_{\alpha} \right), \tag{19}$$

where r is the radius of the cylinder, and α is the percentage length along the musculotendon. The origin ${}^{S}\mathbf{x}_{\text{ori}}$, insertion ${}^{S}\mathbf{x}_{\text{ins}}$, and the output position ${}^{S}\mathbf{x}_{\alpha}$ are all defined with respect to the coordinate space of the wrapping surface S. During training, we use the ℓ^2-norm of the difference between the output of the network and the output of the wrapping library. We include samples with muscles in both attached and detached states, so that at runtime, we do not need to detect whether the muscle is in contact or not. Once trained, the network and the original wrapping library can be used interchangeably, except for one important difference: discontinuity.

To ensure that the network does not contain any discontinuities, we use the *hyperbolic tangent* activation function. Furthermore, we *throw away* the samples near the discontinuity before training. To detect whether a sample is close to a discontinuity, we use the following simple heuristics for all wrapping surfaces.

- Compute l, the length of the "wrapped" portion of the path.
- If $l = 0$, keep the sample.
- Compute L, the length of the whole path.
- If $l/L <$ thresh, discard the sample.
- Otherwise, keep the sample.

Both l and L are readily available from the wrapping surface library. In our current implementation, we use a threshold of 1%.

The trajectory of $\mathbf{x}_{\alpha}$ computed with the library is only C^0, but the trajectory computed by the network is C^{∞}. Despite this difference, the two trajectories are virtually indistinguishable. For example, if we closely inspect what happens to $\mathbf{x}_{\alpha}$ as it approaches and touches the wrapping surface, we find that it slightly penetrates the surface and then floats back to the surface. We also note that the wrapping surface path is already an approximation of the actual path taken by a real muscle, and so this slight discrepancy is within reason.

3.3.2 Incorporating the Network. We now describe how we use the trained network in our simulation framework. As described earlier, to maximize generality, we train the network with origin and insertion in the coordinate space of the wrapping surface as the input: ${}^{S}\mathbf{x}_{\text{ori}}$ and ${}^{S}\mathbf{x}_{\text{ins}}$. To compute the world velocity of the muscle mass point, ${}^{W}\dot{\mathbf{x}}_{\alpha}$, we first need to transform the network input into S space, use the network, and then transform the output back to world space.

Like with Type I and Type II muscles, our goal is to derive $\mathbf{J}_{\alpha m}$ and $\dot{\mathbf{J}}_{\alpha m}$. To derive $\mathbf{J}_{\alpha m}$, we must express the world velocity of $\mathbf{x}_{\alpha}$ using maximal velocities of the bodies. The world velocity of one mass point can be written as the sum of three terms:

$${}^{W}\dot{\mathbf{x}}_{\alpha} = {}^{W}\mathbf{v}_{\text{base}} + {}^{W}\mathbf{v}_{\text{ori}} + {}^{W}\mathbf{v}_{\text{ins}}. \tag{20}$$

The first term represents the *base motion* of the mass point as if it were fixed with respect to S. Since S itself could be moving, even if the mass point is stationary in S, its world velocity could be nonzero. The second term represents the contribution from the relative motion of the *origin* within the S space. Similarly, the third term represents the contribution from the relative motion of the *insertion* within the S space. Our goal is to rewrite each of the three terms so that ${}^{W}\dot{\mathbf{x}}_{\alpha} = \mathbf{J}_{\text{base}}\dot{\mathbf{q}}_m + \mathbf{J}_{\text{ori}}\dot{\mathbf{q}}_m + \mathbf{J}_{\text{ins}}\dot{\mathbf{q}}_m$. Then the Jacobian we are after is $\mathbf{J}_{\alpha m} = \mathbf{J}_{\text{base}} + \mathbf{J}_{\text{ori}} + \mathbf{J}_{\text{ins}}$.

For concreteness, we continue to assume that the origin is fixed to A, insertion is fixed to C, and the surface S is fixed to B (see Fig. 2). The first term in Eq. 20 is the motion of the mass point assuming that it is fixed in S. If we convert this to body B's space, we get:

$$\begin{aligned} {}^{W}\mathbf{v}_{\text{base}} &= {}^{W}_{S}\mathbf{R}\,\Gamma({}^{S}\mathbf{x}_{\alpha})\,\phi_S \\ &= {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha})\,\phi_B, \end{aligned} \tag{21}$$

where ${}^{B}_{S}\mathbf{E}$ is the transformation matrix of S with respect to B, which is fixed over time. The Jacobian for this term, assuming there are two mass points (Fig. 2), is then

$$\mathbf{J}_{\text{base}} = \begin{pmatrix} \mathbf{0} & {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha_1}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha_2}) & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{6\times 18}, \tag{22}$$

where ${}^{S}\mathbf{x}_{\alpha_1}$ and ${}^{S}\mathbf{x}_{\alpha_2}$ are the *values returned from the network*.

To compute $\mathbf{J}_{\text{ori}}$, we first need the relative velocity of the origin from the point of view of the surface. To do so, we must take into account the relative motions of the coordinate spaces, shown in Fig. 5. Since the origin is attached to A, we can compute its world

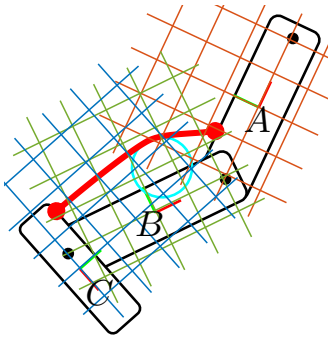


Fig. 5. Coordinate spaces for a wrapping surface muscle. A contains the origin, C contains the insertion, and B contains the wrapping surface S. The S coordinate space (not drawn in this figure) moves rigidly with B.

velocity ${}^{W}\dot{\mathbf{x}}_{\text{ori}}$ using Eq. 8. What we are after is the relative velocity of the origin if we temporarily imagine frame B to be stationary and transfer its motion to frame A. In other words, we subtract from ${}^{W}\dot{\mathbf{x}}_{\text{ori}}$ the hypothetical velocity of the origin attached to body B:

$$ {}^{W}\mathbf{v}^{\text{rel}}_{\text{ori}} = {}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}})\,\phi_A - {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}})\,\phi_B, \tag{23} $$

where ${}^{B}_{A}\mathbf{E} = {}^{W}_{B}\mathbf{E}^{-1}\,{}^{W}_{A}\mathbf{E}$, formed from the current configurations of bodies A and B. We then rotate this into surface space, hit it with the *network Jacobian*, and then rotate back to world:

$$ {}^{W}\mathbf{v}_{\text{ori}} = {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}\mathbf{v}^{\text{rel}}_{\text{ori}}. \tag{24} $$

The network Jacobian, ${}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}}$, is computed with backward differentiation of the network. Given that the input and output of the network are in S space, the network Jacobians are also in S space.

$$ {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}} = \frac{d\,{}^{S}\mathbf{x}_{\alpha}}{d\,{}^{S}\mathbf{x}_{\text{ori}}}, \quad {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{i}} = \frac{d\,{}^{S}\mathbf{x}_{\alpha}}{d\,{}^{S}\mathbf{x}_{\text{ins}}}. \tag{25} $$

Since ${}^{S}\mathbf{x}_{\alpha}$, ${}^{S}\mathbf{x}_{\text{ori}}$, and ${}^{S}\mathbf{x}_{\text{ins}}$ are all in $\mathbb{R}^3$, these network Jacobians are 3×3 matrices.

Combining Eq. 23 and Eq. 24 and extracting out the maximal velocities ϕ_A and ϕ_B, the Jacobian $\mathbf{J}_{\text{ori}}$ for the concrete running example becomes:

$$ \mathbf{J}_{\text{ori}} = \begin{pmatrix} {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \\ {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \end{pmatrix} - \begin{pmatrix} \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{26} $$

The Jacobian $\mathbf{J}_{\text{ins}}$ is derived similarly, except that the insertion is fixed to body C instead of A.

$$ \mathbf{J}_{\text{ins}} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \\ \mathbf{0} & \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \end{pmatrix} - \begin{pmatrix} \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{C}\mathbf{E}\,{}^{C}\mathbf{x}_{\text{ins}}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{C}\mathbf{E}\,{}^{C}\mathbf{x}_{\text{ins}}) & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{27} $$

The time derivatives of the individual quantities in $\dot{\mathbf{J}}_{\text{base}}, \dot{\mathbf{J}}_{\text{ori}}$, and $\dot{\mathbf{J}}_{\text{ins}}$ are listed in the supplementary material. We analytically derive all of the derivatives, except for the network Jacobians. For these, we perturb ${}^{S}\mathbf{x}_{\text{ori}}$ and ${}^{S}\mathbf{x}_{\text{ins}}$ in time to evaluate the network again to perform finite differencing:

$$ \begin{aligned} {}^{S}\mathbf{x}^{+}_{\text{ori}} &= {}^{S}\mathbf{x}_{\text{ori}} + \epsilon\,{}^{S}\mathbf{v}^{\text{rel}}_{\text{ori}}, & {}^{S}\dot{\mathbf{J}}^{\text{NN}}_{\alpha\text{o}} &= \left({}^{S}\mathbf{J}^{\text{NN+}}_{\alpha\text{o}} - {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}}\right)/\epsilon, \\ {}^{S}\mathbf{x}^{+}_{\text{ins}} &= {}^{S}\mathbf{x}_{\text{ins}} + \epsilon\,{}^{S}\mathbf{v}^{\text{rel}}_{\text{ins}}, & {}^{S}\dot{\mathbf{J}}^{\text{NN}}_{\alpha\text{i}} &= \left({}^{S}\mathbf{J}^{\text{NN+}}_{\alpha\text{i}} - {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{i}}\right)/\epsilon, \end{aligned} \tag{28} $$

where ${}^{S}\mathbf{v}^{\text{rel}}_{\text{ori}}$ is computed as ${}^{S}\mathbf{v}^{\text{rel}}_{\text{ori}} = {}^{S}_{W}\mathbf{R}\,{}^{W}\mathbf{v}^{\text{rel}}_{\text{ori}}$, and likewise for ${}^{S}\mathbf{v}^{\text{rel}}_{\text{ins}}$.

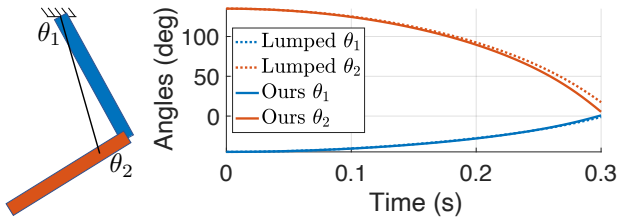
4 RESULTS

We implemented a prototype in MATLAB. The networks were trained on a computer with a Ryzen 7 5800X CPU with 32 GB of RAM and an RTX 3080 Ti GPU with 12 GB of RAM. We trained the networks using Adam [Kingma and Ba 2015] with the default parameters and a learning rate of 10^{-4}. For each network, we used 6 layers with 256 neurons per layer. We used tanh as the activation function for all layers. The trained networks were loaded and evaluated in MATLAB. We used around 30k samples, and the training took about 12 hours. More details are in the supplementary document.

4.1 Comparison to Analytical Results

We start with comparisons to the simulation and analytical results by Pai [2010] to verify that our general framework is in agreement with published results. First we simulate the scene in Fig. 6a, which uses the same setup as *their* Fig. 2. As shown by the solid lines in *our* Fig. 6b, the two angles reach zero at 0.3 seconds, just like in the published result.

Pai also analytically computed the contributions to the self-inertia of the rat knee joint from the biceps femoris posterior muscle and the bones of the shank, and reported that the relative contribution from the muscle with respect to the bones is 45%. We also computed the inertia from the muscle and the bones using Eq. 7b, and obtained the value of 45.8%. The slight discrepancy goes down if we include more mass points, but we found that 10-20 are sufficient for most purposes. Furthermore, the discrete approach allows us to more easily model the non-uniform mass distribution along the musculotendon path.



(a) (b)

Fig. 6. Comparison to published results [Pai 2010]. (a) Two bones and one muscle, all with the same mass. (b) The solid lines show that after simulating the system with the muscle for 0.3 seconds, the two angles straighten out as in the previous work. The dotted lines show the same simulation but with the mass of the muscle lumped onto the bones.

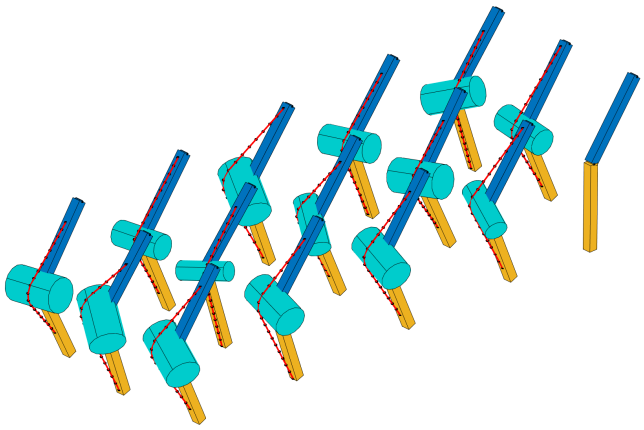
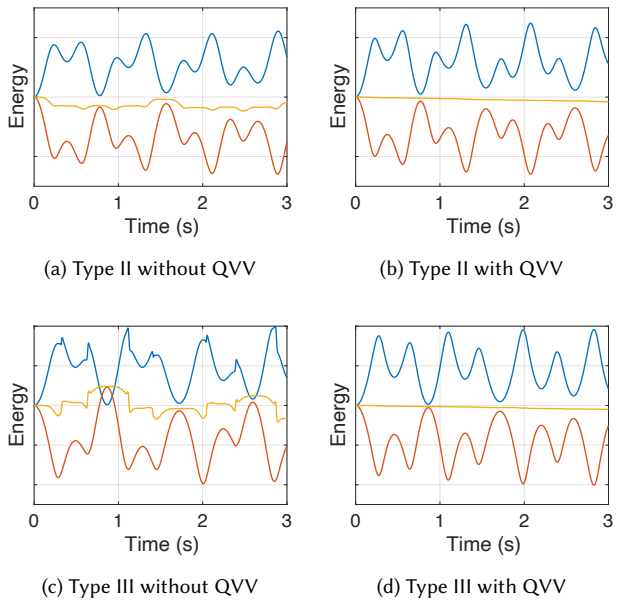


Fig. 7. Double pendulums with cylinder wrapping. The same trained network is used for a range of input parameters. For comparison, the right-most double pendulum is simulated without a muscle.



(a) Type II without QVV (b) Type II with QVV

(c) Type III without QVV (d) Type III with QVV

Fig. 8. (a-b) Energy plots from a Type II muscle with and without QVV. (c-d) Energy plots from a Type III muscle with and without QVV. Kinetic energy is shown in blue, potential energy in red, and total energy in yellow.

4.2 Network Jacobian

We simulate a group of double pendulums with varying origin, insertion, radius, and the initial rotation of the wrapping surface, as shown in Fig. 7. In these experiments, the masses of the proximal bone, the distal bone, and the muscle are set to be equal. For comparison, in the right-most pendulum, we remove the muscle, adding half of its mass to the proximal bone and the other half to the distal bone. Using the same trained network, the simulator is able to account for all the variations properly.

4.3 Energy Behavior

To show the importance of the $\dot{\mathbf{J}}_{\alpha m}$ term that we derived, we take one of the simulations from Fig. 7, and remove $\dot{\mathbf{J}}_{\alpha m}$, and consequently, the quadratic velocity vector (QVV) of the muscle mass points [Shabana 2013]. (We keep the QVV of the bones in the simulation.) As shown in Fig. 8c, even with the SDIRK2 time integrator, the energy oscillates wildly. On the other hand, as shown in Fig. 8d, the energy stays stable once we put the QVV of the muscle back in. Similarly, in Fig. 8a-8b, we show the same experiment with a Type II muscle. Again, without the QVV of the muscle, the energy fluctuates, but with the QVV of the muscle included, the energy remains stable.

4.4 Simulation Stability

The effect of muscle inertia is stronger when a relatively light bone is actuated by a relatively large muscle mass located away from the joint. In Fig. 1b, we show an example of such a case with the flexor digitorum profundus and superficialias muscles (FDP & FDS), which originate near the elbow and insert into the distal and middle phalanges, respectively. For our simulation, we modeled the bones and joints using open source data [Lee et al. 2015], and we manually modeled the FDP and FDS as Type II muscles, with the tendons routed through pulleys implemented as path points. We fixed all joints except for the three joints of the index finger, which we modeled as revolute joints. The masses of the bones are set from the meshes, with a relatively large density of 5 g cm^{-3} to account for the rest of the finger mass, and the mass of the muscles is set to 200 g each. With a fixed time step of 1 ms, we apply different amounts of force for the first two time steps of the simulation, to model flicking the fingertip with the other hand. With the traditional approach, the simulation becomes unstable when the force is increased to 5 N, whereas with our approach, the simulation becomes unstable when the force is increased to 20 N. This is due to the fact that with the traditional approach, the muscle inertia gets absorbed into the forearm segment, and thus the generalized inertia of the finger joints is not affected by the muscles, unlike with our approach. (The peak force during typing is around 2 N [Kim et al. 2014].) We also note that the inertia due to the muscles in this particular example is *substantially underestimated*, since we assume that strain is equal throughout the length of the musculotendon. If we also take into account the fact that the tendon is highly stiff, joint motion would cause more of the muscle mass to move, which would increase the inertia further.

4.5 Comparison to OpenSim

For our next experiment, we use marker-based motion-capture data to drive the skeleton and compute the resulting torques at the joints with inverse dynamics. We show a 0.5 second clip in Fig. 9. The figure shows the swing phase: from take-off to touch-down. We use OpenSim to scale the bone lengths/masses, joint locations, muscle origin/insertion, path points, and wrapping surfaces to the specific subject. The skeleton has 11 DOFs: 6 for pelvis, 3 for the right hip, 1 for the right knee, and 1 for the right ankle. We model four muscles that span the ankle: gastrocnemius lateral (Type III), gastrocnemius medial (Type III), soleus (Type I), and tibialis anterior (Type II). The

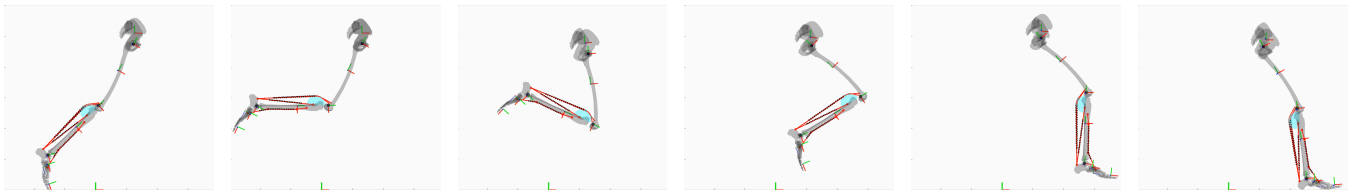


Fig. 9. The swing phase of a 19.1 km/h treadmill run, showing only the right leg. The four muscles (and their types) are: gastrocnemius lateral (Type III), gastrocnemius medial (Type III), soleus (Type I), and tibialis anterior (Type II).

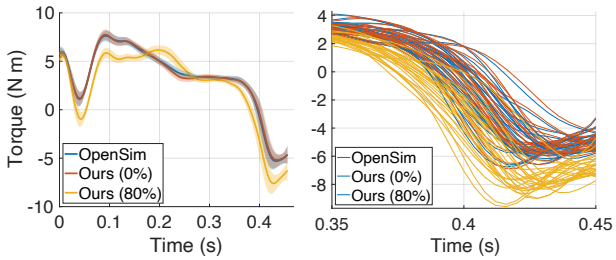


Fig. 10. (Left) Ankle torque computed by inverse dynamics, showing the mean and the standard deviation. Blue plot is generated by OpenSim, which does not support inertial muscles. Red plot is generated by our simulator with the muscles accounting for 0% of the total mass. Yellow plot is generated by our simulator with 80% of the tibia segment mass transferred to the muscles. (Right) The closeup of the final dip, showing the individual trajectories. Our simulator generates results that gracefully degrade to OpenSim's results as the inertia of the muscles is decreased to zero.

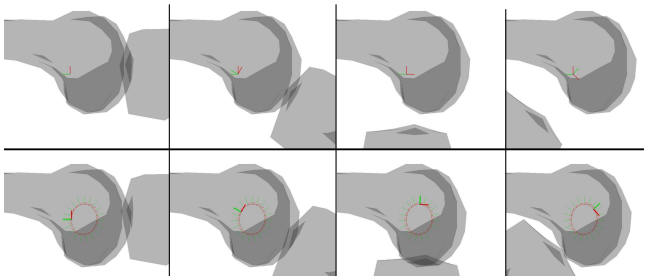


Fig. 11. Our approach supports complex joint types. Top row: Knee with a revolute joint—the tibia separates from the femur. Bottom row: Knee with a spline joint—the tibia stays close to the femur.

subject runs on a treadmill at 19.1 km/h, and we use OpenSim to reconstruct the motion of the skeleton from the marker data. We collect the ankle torque computed with inverse dynamics from the swing phases from two 10-second trials using OpenSim and our simulator. We overlay the swing phases on top of each other and plot the results in Fig. 10. We show the torque results generated by:

- OpenSim (blue), which does not support inertial muscles.
- Our simulator (red) with the muscles accounting for 0% of the total mass of the tibia segment.
- Our simulator (yellow) with 80% of the tibia mass transferred to the muscles.

The relative masses of the four muscles are taken from the literature [Ward et al. 2009]. For each muscle, the mass is distributed into 20 equally spaced points in the middle portion of the musculotendon that correspond to the muscle (as opposed to the tendons). Fig. 10b shows the closeup of the final dip. Comparing the blue and red plots, we confirm that our simulator generates results that *gracefully degrades* to OpenSim's results, as the inertia of the muscles are decreased to zero. On the other hand, comparing the red and yellow plots, we note that the ankle moment can differ by as much as 40% due to the effect of muscle inertia. In the supplementary material, we show how our result gracefully degrade to OpenSim's result.

4.6 Spline Joint Knee with Hill-Type Muscles

To demonstrate the generality and flexibility of our approach, we take the same scene setup as above, but replace the revolute joint of the knee with a spline joint [Lee and Terzopoulos 2008] and add the semimembranosus (Type I) and the rectus femoris (Type III). As shown in Fig. 11, we manually model a spline joint to better model the motion of the tibia with respect to the femur. (OpenSim uses a similar technique called a "mobilizer" [Seth et al. 2010].) We also use Hill-type muscles [Zajac 1989] to drive the knee and ankle joints, as opposed to using mocap as in §4.5. We use the damped equilibrium model with active force-length, active force-velocity, passive force-length, and tendon force-length curves taken from the biomechanics literature [Millard et al. 2013]. We manually set the excitation levels of the gastrocnemius lateral/medial muscles to a low level. We use a proportional controller based on the ankle joint angle to set the excitation levels of the tibialis anterior and the soleus muscles. Then we manually excite the rectus femoris and semimembranosus muscles, which results in the extension and flexion of the knee, as shown in Fig. 12.

4.7 Differentiable Reaching with Adjoint Method

For the final result, we use the adjoint method [McNamara et al. 2004; Geilinger et al. 2020; Xu et al. 2021] to compute the simulation derivatives to optimize for a reaching task using an arm model [Chadwick et al. 2014] with manually placed muscles, shown in Fig. 1d. For the three heads of the deltoid muscle, we use sphere-capped cylinders, and for the three heads of the triceps brachii muscle, we use cylinders. The task objective is to move the hand to the specified target, and the task parameters are the constant torques to be applied to the shoulder (3 DOF) and elbow (1 DOF) joints. We use `fminunc` as the optimizer with our analytical derivatives. As a comparison, when we run `fminunc` in gradient-free mode, it takes an order-of-magnitude more time to optimize, requiring many more

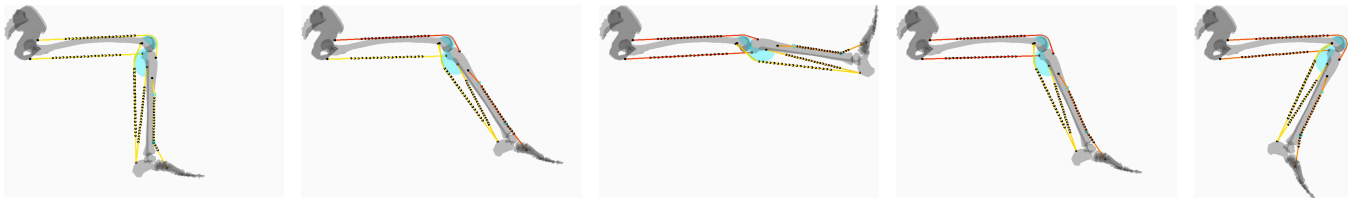


Fig. 12. For §4.6, we add a spline joint knee and Hill-type muscles to the model used in §4.5. We manually excite the rectus femoris and the semimembranosus muscles. The excitation levels of the soleus and the tibialis anterior muscles are computed automatically with a proportional controller.

simulation runs. Our inertial muscles, however, work seamlessly with the adjoint method. Furthermore, more objectives can be added, such as having the hand come to a rest, or more generally, following a preset trajectory.

5 CONCLUSION & FUTURE WORK

We presented an approach to account for the inertia of the muscles in a musculoskeletal simulation. We are able to handle a wide variety of musculotendon paths, including (I) straight, (II) polyline, and (III) curved paths over wrapping surfaces. For Type II muscles, we use the Eulerian-on-Lagrangian framework, and for Type III muscles, we use neural networks. Our approach is compatible with existing simulation techniques, such as inverse dynamics and differentiable dynamics, and the motion can be driven by muscle activations or joint torques. In the limit, as the mass of the muscles is transferred to the bones, our simulation results gracefully degrade to results obtained using traditional musculoskeletal simulators without inertial muscles. Finally, it is possible to mix and match inertial and non-inertial musculotendons, depending on the application.

We use the centerline to account for the muscle mass, which is still an approximation, but this is a prudent choice, since using a full, volumetric mesh is impractical for these experiments, at least currently. For example, it would be a challenge to produce results with FEM that can gracefully degrade to OpenSim results the way our method can. It may be possible to tweak the FEM simulation parameters to produce the desired output, but we believe that using FEM for these target applications is extremely challenging if not impossible, considering the high number of parameters and the computational complexity required by the volume model. Future work may address these difficulties with volumetric FEM. We believe that such work, along with ours, would pave the way toward a fully comprehensive simulation framework.

Some models use path points that move based on the skeletal DOFs (*e.g.,* LBS waypoints [Ryu et al. 2021], moving muscle points [Seth et al. 2018]). Although we have not implemented these, they can be categorized as Type II path points with their corresponding Jacobians between the skeletal DOFs and these points.

For muscles with long tendons, our approach still underestimates the muscle inertia because we assume that the strain is equal along the entire length of the musculotendon. For future work, we would like to derive the kinematics of the muscle points while incorporating inextensible tendons to reduce this underestimation.

We plan to train on more wrapping surface types, including ellipsoid, torus, sphere, and double cylinder [Seth et al. 2018; Garner and Pandy 2000]. In theory, our neural network approach can be used for any path. However, some wrapping surfaces require many parameters, which could make training more difficult and slower. For example, to train a double cylinder, it would require five more parameters than a single cylinder. (The first cylinder can be defined along the Z-axis. Assuming that the second cylinder is not orthogonal to the first, we need two parameters for a point and two for the direction, plus the radius.) Similarly, using a network for an arbitrary shape [Lloyd et al. 2020] could be a challenge, depending on the number of parameters of the surface.

Network evaluation is a bottleneck in our current implementation, which is written in MATLAB. We expect that evaluating the network on the GPU and batching the input as much as possible would increase the performance significantly. Furthermore, since our framework allows mixing and matching of inertial and non-inertial musculotendons (*e.g.,* §4.5), it is possible to find a subset of musculotendons to add inertia to, in order to find the sweet spot in terms of efficiency and efficacy. Automatically determining the set of musculotendons that affects the total inertia the most is an interesting avenue of future research.

Finally, given that our approach is compatible with the adjoint method, it would be interesting to optimize for tasks involving ground contact [Geilinger et al. 2020; Xu et al. 2021]. In our current implementation, as with most other musculoskeletal simulators [Millard et al. 2013], musculoskeletal dynamics and muscle/tendon dynamics are integrated separately, and so the adjoint method cannot use muscle excitations as parameters. Going further, we could add another layer on top of the adjoint method to compute for the muscle excitations rather than joint torques.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This work was sponsored in part by the National Science Foundation (CAREER-1846368) and by Biotechnology and Biological Sciences Research Council (BB/S003762/1).

REFERENCES

Autodesk. 2011. *Maya Muscle.* Autodesk.

Edward K Chadwick, Dimitra Blana, Robert F Kirsch, and Antonie J Van Den Bogert. 2014. Real-time simulation of three-dimensional shoulder girdle and arm dynamics. *IEEE Transactions on Biomedical Engineering* 61, 7 (2014), 1947–1956.

David T. Chen and David Zeltzer. 1992. Pump It up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method. *SIGGRAPH Comput. Graph.* 26, 2 (Jul. 1992), 89–98.

Michael Damsgaard, John Rasmussen, Søren Tørholm Christensen, Egidijus Surma, and Mark De Zee. 2006. Analysis of musculoskeletal systems in the AnyBody Modeling System. *Simulation Modelling Practice and Theory* 14, 8 (2006), 1100–1111.

Ye Fan, Joshua Litven, and Dinesh K. Pai. 2014. Active Volumetric Musculoskeletal Systems. *ACM Trans. Graph.* 33, 4, Article 152 (Jul. 2014), 9 pages.

Brian A Garner and Marcus G Pandy. 2000. The obstacle-set method for representing muscle paths in musculoskeletal models. *Computer methods in biomechanics and biomedical engineering* 3, 1 (2000), 1–30.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Trans. Graph.* 32, 6, Article 206 (Nov. 2013), 11 pages.

Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. 2020. ADD: Analytically Differentiable Dynamics for Multi-Body Systems with Frictional Contact. *ACM Trans. Graph.* 39, 6, Article 190 (Nov. 2020).

Jianqiao Guo, Hongshi Huang, Yuanyuan Yu, Zixuan Liang, Jorge Ambrósio, Zhihua Zhao, Gexue Ren, and Yingfang Ao. 2020. Modeling muscle wrapping and mass flow using a mass-variable multibody formulation. *Multibody System Dynamics* (2020), 1–22.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Vol. 31. Springer Science & Business Media.

Minyeon Han, Jisoo Hong, and FC Park. 2015. Musculoskeletal dynamics simulation using shape-varying muscle mass models. *Multibody System Dynamics* 33, 4 (2015), 367–388.

Junggon Kim and Nancy S. Pollard. 2011. Fast Simulation of Skeleton-driven Deformable Body Characters. *ACM Trans. Graph.* 30, 5, Article 121 (Oct. 2011), 19 pages.

Jeong Ho Kim, Lovenoor Aulck, Michael C. Bartha, Christy A. Harper, and Peter W. Johnson. 2014. Differences in typing forces, muscle activity, comfort, and typing performance among virtual, notebook, and desktop keyboards. *Applied Ergonomics* 45, 6 (2014), 1406–1413.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015*.

Taku Komura, Yoshihisa Shinagawa, and Tosiyasu L Kunii. 1997. A Muscle-based Feed-forward Controller of the Human Body. In *Computer Graphics Forum*, Vol. 16. Wiley Online Library, C165–C176.

Taku Komura, Yoshihisa Shinagawa, and Tosiyasu L Kunii. 2000. Creating and retargetting motion by the musculoskeletal human body model. *The visual computer* 16, 5 (2000), 254–270.

Taku Komura, Yoshihisa Shinagawa, and Tosiyasu L Kunii. 2001. An inverse kinematics method based on muscle dynamics. In *Proceedings. Computer Graphics International 2001*. IEEE, 15–22.

Jong Hwa Lee, Deanna S Asakawa, Jack T Dennerlein, and Devin L Jindrich. 2015. Finger muscle attachments for an OpenSim upper-extremity model. *PloS one* 10, 4 (2015), e0121712.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. 2019. Scalable Muscle-Actuated Human Simulation and Control. *ACM Trans. Graph.* 38, 4, Article 73 (July 2019).

Seunghwan Lee, Ri Yu, Jungnam Park, Mridul Aanjaneya, Eftychios Sifakis, and Jehee Lee. 2018. Dexterous Manipulation and Control with Volumetric Muscles. *ACM Trans. Graph.* 37, 4, Article 57 (Jul. 2018), 13 pages.

Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2009. Comprehensive Biomechanical Modeling and Simulation of the Upper Body. *ACM Trans. Graph.* 28, 4, Article 99 (Sep. 2009), 17 pages.

Sung-Hee Lee and Demetri Terzopoulos. 2006. Heads up! Biomechanical Modeling and Neuromuscular Control of the Neck. *ACM Trans. Graph.* 25, 3 (Jul. 2006), 1188–1198.

Sung-Hee Lee and Demetri Terzopoulos. 2008. Spline Joints for Multibody Dynamics. *ACM Trans. Graph.* 27, 3, Article 22 (Aug. 2008), 8 pages.

Yoonsang Lee, Moon Seok Park, Taesoo Kwon, and Jehee Lee. 2014. Locomotion Control for Many-Muscle Humanoids. *ACM Trans. Graph.* 33, 6, Article 218 (Nov. 2014), 11 pages.

Robson Lemos, Marcelo Epstein, Walter Herzog, and Brian Wyvill. 2001. Realistic skeletal muscle deformation using finite element analysis. In *Proceedings XIV Brazilian Symposium on Computer Graphics and Image Processing*. IEEE, 192–199.

John E Lloyd, François Roewer-Després, and Ian Stavness. 2020. Muscle Path Wrapping on Arbitrary Surfaces. *IEEE Trans. Biomedical Engineering* 68, 2 (2020), 628–638.

Elaine Marieb and Katja Hoehn. 2010. *Human Anatomy & Physiology* (8 ed.). Benjamin Cummings.

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 449–456.

Matthew Millard, Thomas Uchida, Ajay Seth, and Scott L Delp. 2013. Flexing computational muscle: modeling and simulation of musculotendon dynamics. *Journal of biomechanical engineering* 135, 2 (2013), 021005.

Sehee Min, Jungdam Won, Seunghwan Lee, Jungnam Park, and Jehee Lee. 2019. SoftCon: Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators. *ACM Trans. Graph.* 38, 6, Article 208 (Nov. 2019), 12 pages.

Akihiko Murai, Kosuke Kurosaki, Katsu Yamane, and Yoshihiko Nakamura. 2010. Musculoskeletal-see-through mirror: Computational modeling and algorithm for whole-body muscle activity visualization in real time. *Progress in biophysics and molecular biology* 103, 2-3 (2010), 310–317.

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

Victor Ng-Thow-Hing. 2001. *Anatomically-based models for physical & geometric reconstruction of humans & other animals*. Ph. D. Dissertation. University of Toronto.

Dinesh K Pai. 2010. Muscle mass in musculoskeletal models. *Journal of Biomechanics* 43, 11 (2010), 2093–2098.

Hoseok Ryu, Minseok Kim, Seungwhan Lee, Moon Seok Park, Kyoungmin Lee, and Jehee Lee. 2021. Functionality-Driven Musculature Retargeting. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 341–356.

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.

Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. 1997. Anatomy-based modeling of the human musculature. In *Proc. SIGGRAPH 97 (Annual Conference Series)*. ACM, 163–172.

Andreas Scholz, Michael Sherman, Ian Stavness, Scott Delp, and Andrés Kecskeméthy. 2016. A fast multi-obstacle muscle wrapping method using natural geodesic variations. *Multibody System Dynamics* 36, 2 (2016), 195–219.

Ajay Seth, Jennifer L Hicks, Thomas K Uchida, Ayman Habib, Christopher L Dembia, James J Dunne, Carmichael F Ong, Matthew S DeMers, Apoorva Rajagopal, Matthew Millard, et al. 2018. OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLoS computational biology* 14, 7 (2018), e1006223.

Ajay Seth, Michael Sherman, Peter Eastman, and Scott Delp. 2010. Minimal formulation of joint motion for biomechanisms. *Nonlinear dynamics* 62, 1 (2010), 291–303.

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

Weiguang Si, Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2015. Realistic Biomechanical Simulation and Control of Human Swimming. *ACM Trans. Graph.* 34, 1, Article 10 (Dec. 2015), 15 pages.

Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. 2005. Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data. *ACM Trans. Graph.* 24, 3 (Jul. 2005), 417–425.

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.

Shinjiro Sueda, Andrew Kaufman, and Dinesh K. Pai. 2008. Musculotendon Simulation for Hand Animation. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–8.

Joseph Teran, Silvia Blemker, Victor Ng-Thow-Hing, and Ronald Fedkiw. 2003. Finite Volume Methods for the Simulation of Skeletal Muscle. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.* (San Diego, California). 68–74.

Joseph Teran, Eftychios Sifakis, Silvia S. Blemker, Victor Ng-Thow-Hing, Cynthia Lau, and Ronald Fedkiw. 2005. Creating and Simulating Skeletal Muscle from the Visible Human Data Set. *IEEE TVCG* 11, 3 (May 2005), 317–328.

Demetri Terzopoulos and Keith Waters. 1990. Physically-based facial modelling, analysis, and animation. *Journal of Vis. & Comp. Anim.* 1, 2 (1990), 73–80.

Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. 2012. Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives. *ACM Trans. Graph.* 31, 4, Article 25 (Jul. 2012), 11 pages.

Ying Wang, Nicholas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman, and Shinjiro Sueda. 2019. RedMax: Efficient & Flexible Approach for Articulated Dynamics. *ACM Trans. Graph.* 38, 4, Article 104 (Jul. 2019), 10 pages.

Samuel R Ward, Carolyn M Eng, Laura H Smallwood, and Richard L Lieber. 2009. Are current measurements of lower extremity muscle architecture accurate? *Clinical orthopaedics and related research* 467, 4 (2009), 1074–1082.

Keith Waters. 1987. A muscle model for animation three-dimensional facial expression. *ACM SIGGRAPH Computer Graphics* 21, 4 (1987), 17–24.

Keith Waters and Demetri Terzopoulos. 1990. A physical model of facial tissue and muscle articulation. In *Proc. Conf. on Vis. in Biomedical Computing*. IEEE, 77–78.

Jane Wilhelms and Allen Van Gelder. 1997. Anatomically based modeling. In *Proc. SIGGRAPH 97 (Annual Conference Series)*. ACM, 173–180.

Jie Xu, Tao Chen, Lara Zlokapa, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. 2021. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Robotics: Science and Systems*.

Felix E Zajac. 1989. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical reviews in biomedical engineering* 17, 4 (1989), 359–411.

Qing-hong Zhu, Yan Chen, and Arie Kaufman. 1998. Real-time biomechanically-based muscle volume deformation using FEM. In *Computer Graphics Forum*, Vol. 17. Wiley Online Library, 275–284.

Ziva Dynamics. 2018. Ziva VFX. https://zivadynamics.com/ziva-vfx.