INERTIAL MUSCULOTENDONS

A Dissertation

by

YING WANG

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Shinjiro Sueda
Committee Members,	John Keyser
	Vinayak Krishnamurthy
	Dylan Shell
Head of Department,	Scott Schaefer

December 2022

Major Subject: Computer Science

ABSTRACT

This thesis aims to develop a simple and practical framework for musculoskeletal simulation that accounts for the inertia of muscles.

Computer animation researchers have been using and extending muscle-driven skeletal simulations for many graphics applications. 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 are "lumped" to the bones at the 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 is from skeletal muscles. We present a novel framework for incorporating the effects of muscle inertia for all commonly used musculotendon path types, including those with multiple path points and wrapping surfaces. To maximize inter-operability with existing musculoskeletal simulators, 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 always at its local minimum. 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.

DEDICATION

This dissertation is dedicated to my parents, Xiang Wang and Zheren Wang, who have been a constant source of support and encouragement during the challenges of graduate school and life.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professors Shinjiro Sueda, John Keyser, Dylan Shell of the Department of Computer Science and Engineering and Professor Vinayak Krishnamurthy of the Department of Mechanical Engineering.

The analyses depicted in Chapters 3 and 4 were contributed in part by Nicholas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman and Professor Shinjiro Sueda, and were published in 2019 in an article listed in ACM Transactions on Graphics (SIGGRAPH 2019). The data in Chapter 6 were provided in part by Professors Jasper Verheul and Sang-Hoon Yeo. The network training part in Chapters 5 and 6 was advised in part by Professor Nima Khademi Kalantari, and were published in 2022 in an article listed in ACM Transactions on Graphics (SIGGRAPH ASIA 2022).

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported in part by the National Science Foundation (CAREER-1846368) and by Biotechnology and Biological Sciences Research Council (BB/S003762/1).

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION*

1.1 Problem Statement

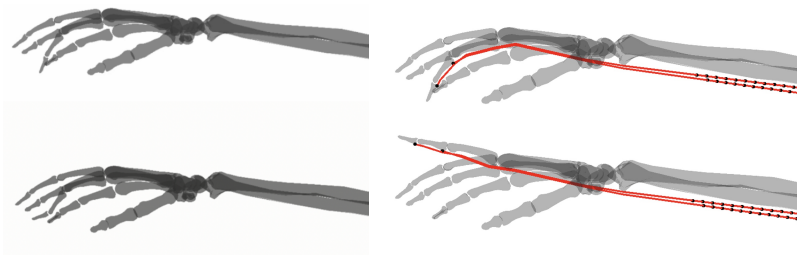
Musculoskeletal simulations have become an increasingly important component in modern graphics applications. Computer animation researchers have been using and extending muscle-driven skeletal simulations for many applications—to name some examples, improved inverse kinematics over methods without taking human body into account [1], head/neck animation, [2], hand animation [3], real-time visualization of muscle activations [4], energy-minimizing gait animations [5], creation of imaginary bipedal characters [6], upper body animations [7, 8], and control of characters under various anatomical conditions [9, 10].

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 the rest pose, meaning that the muscle masses are added to the bone masses. This means that the effect of the muscle inertia cannot be reflected in the dynamics of the system when the muscles slide with respect to the bones, even though around 40% of total body mass comes from skeletal muscles [11].

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/or 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 [12] notes that the triceps surae muscle of the human ankle can account for an additional 7.6 % of the effective inertia of the joint. As another example, let us consider the flexors of the hand during a finger flicking motion (Fig. 1.1b). 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

*Part of this chapter is reprinted with permission from Y. Wang, J. Verheul, S.-H. Yeo, N. K. Kalantari, and S. Sueda, "Differentiable simulation of inertial musculotendons," *ACM Transactions on Graphics*, vol. 41, Nov. 2022.



(a) (b)

Figure 1.1: We model flicking the fingertip with the other hand and add different amounts of forces. (a) Without muscle inertia, the simulation shows artifacts when applying 5N forces. (b) While with muscle inertia, the simulation shows artifacts when applying 20N forces. The simulation remains stable under an impulse that is several times larger. We can show that muscle inertia changes the dynamics result of flicking motion and stabilizes the simulation.

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 that is 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.

As we mentioned before, almost all musculoskeletal simulators used in graphics ignore the effect of the inertia of the muscles. Instead, they use the "segment lumping" method, which lumps the mass of the muscles to the bones at the rest pose. Fig. 1.2 illustrates what segment lumping means, and Fig. 1.3 illustrates our method, which does not lump the muscle mass.

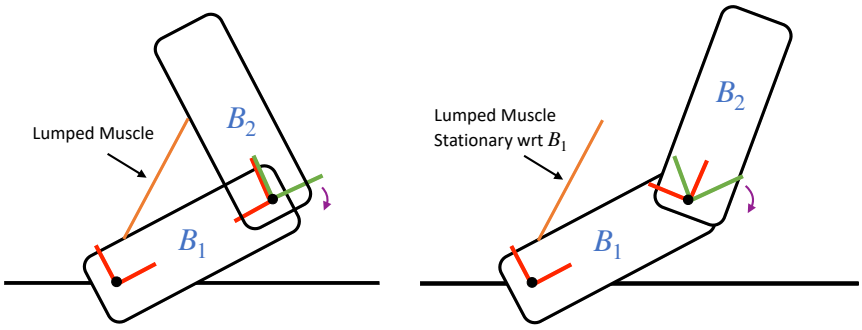


Figure 1.2: We use this simple example to briefly explain what segment lumping means. We have two rigid bodies in this case, B_1 and B_2. If we want to lump the mass of the muscle along with the bones within body B_1, we will model the muscle and B_1 as a single body in the model of dynamics, which means that even when the elbow joint angle changes during simulation, the muscle should be stationary with respect to B_1.

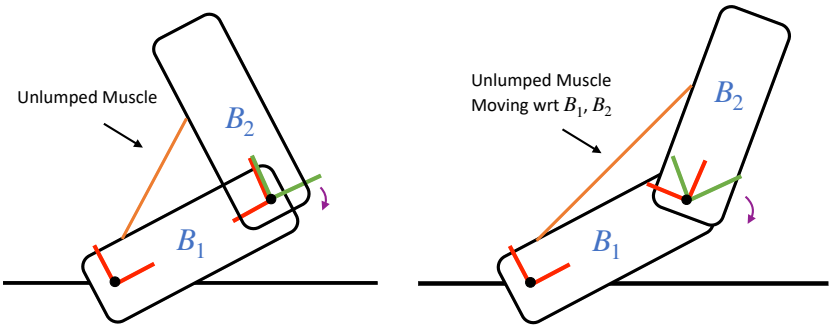


Figure 1.3: Here is an example of a unlumped muscle. The muscle should be moving with respect to B_1 when the elbow joint angle changes during simulation.

1.2 Our Approach

In the past few years, biomechanics researchers have proposed techniques to deal with muscle inertia [13, 14], but these approaches can only be used for relatively simple muscle paths. Therefore, we propose a framework for incorporating the effects of muscle inertia into musculoskeletal

simulations that contain more complex muscle paths, including those with wrapping surfaces. Our framework is based on the kinematic hierarchy shown in Fig. 1.4. This hierarchy allows us to compute the velocities of the muscle mass points (x_α) from the velocities of the skeletal joints (q_r). **The main technical contribution of this thesis is the derivation of this Jacobian mapping, $\mathbf{J}_{\alpha r}$, and its time derivative, $\dot{\mathbf{J}}_{\alpha r}$.**

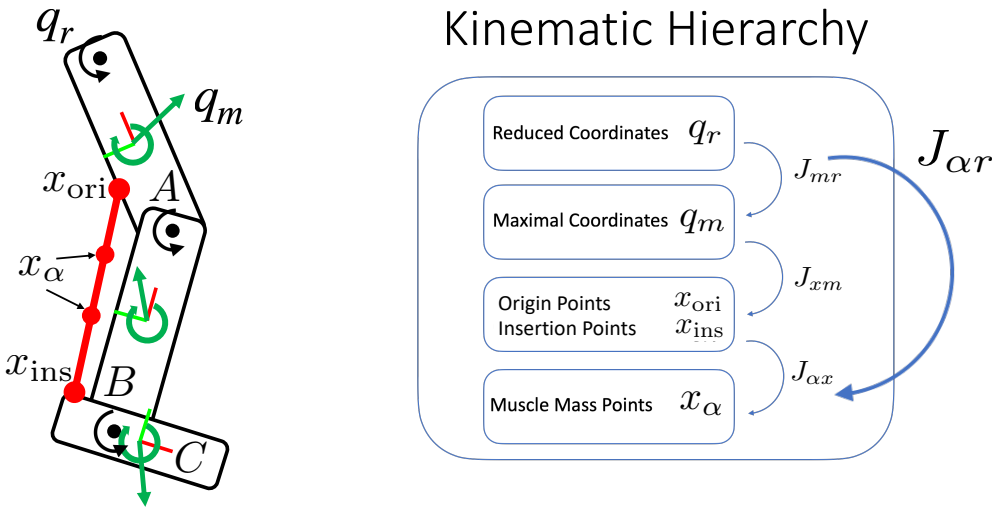


Figure 1.4: A simple musculoskeletal system with three bones and one muscle. The origin is on the first body and the insertion is on the third body. We use this simple system to illustrate our kinematic hierarchy of a musculoskeletal system with corresponding Jacobian mappings between different coordinates.

We now briefly describe the various Jacobian factors involved in this kinematic hierarchy. We express the motion of the musculotendons in terms of the motion of the skeletal joint using a chain of Jacobians so that at the top level, only the reduced degrees of freedom of the skeleton, shown in the illustration (Fig. 1.4), are used to drive both bones and musculotendons. The key to efficient inertial muscle simulation is to find a mapping between the change in the 3D world coordinates of these mass points and the change in the reduced coordinates of the articulated rigid body system.

This is accomplished through a series of Jacobian mappings:

- $\mathbf{J}_{mr}$ maps reduced coordinates q_r to maximal coordinates q_m. The maximal coordinates are represented by the rotational and translational components of the rigid bodies. These reduced coordinates q_r are composed of a series of joint angles. We will perform our dynamic system simulation using these reduced coordinates.

- $\mathbf{J}_{xm}$ maps maximal coordinates q_m to origin/insertion points $\mathbf{x}_{\text{ori}}$/$\mathbf{x}_{\text{ins}}$, where the origin and insertion refer to the muscle attachment points on the bones that are proximal and distal with respect to the muscle, respectively. To simulate the movement of the origin and insertion points, we can track their position using maximal coordinates q_m, which serve as the input for our muscle mass points algorithm.

- $\mathbf{J}_{\alpha x}$ maps origin/insertion points $\mathbf{x}_{\text{ori}}$/$\mathbf{x}_{\text{ins}}$ to muscle mass points x_α. To take into account the inertia of muscles, we insert mass points (the red dots on the path from $\mathbf{x}_{\text{ori}}$ to $\mathbf{x}_{\text{ins}}$ in the figure) 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. When sampled at a reasonable density, these insertion points can accurately approximate the path of inertial muscles.

These Jacobians are used to map between the velocities:

$$\dot{q}_m = \mathbf{J}_{mr}\, \dot{q}_r, \tag{1.1a}$$

$$\dot{q}_x = \mathbf{J}_{xm}\, \dot{q}_m, \tag{1.1b}$$

$$\dot{x}_a = \mathbf{J}_{\alpha x}\, \dot{q}_x. \tag{1.1c}$$

We chain these Jacobians together so that, ultimately, we can express the motion of the muscle mass points as a function of the joint angles of the skeleton:

$$\dot{x}_\alpha = \mathbf{J}_{\alpha x}\, \mathbf{J}_{xm}\, \mathbf{J}_{mr}\, \dot{q}_r. \tag{1.2}$$

1.2.1 Types of musculotendon paths

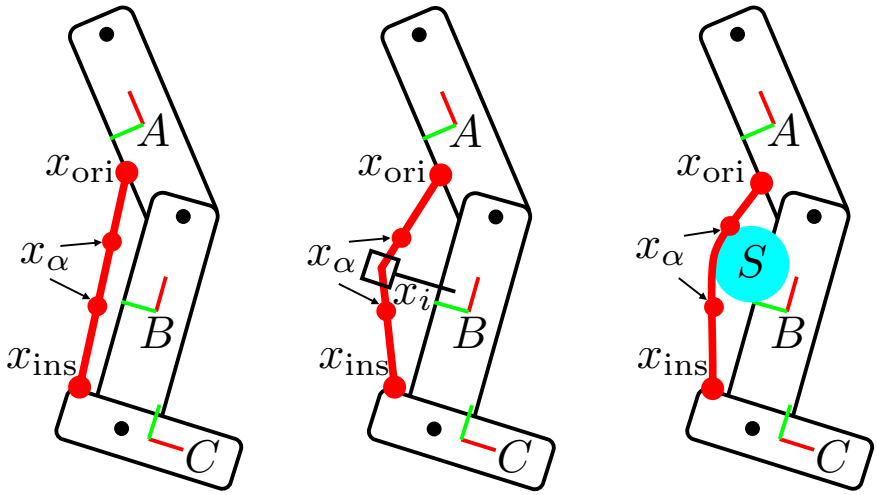


Figure 1.5: Concrete running example for Type I, II, and III muscles. In all cases, there are three bones and one muscle. The origin is on body A, and the insertion is on body C. Type II muscle has a path point on body B, and Type III muscle has a wrapping surface S defined with respect to body B.

To aid us in the derivation, we categorize musculotendon paths into three types (Fig. 1.5):

I: Straight-line paths, whose Jacobians are derived in a straightforward manner.

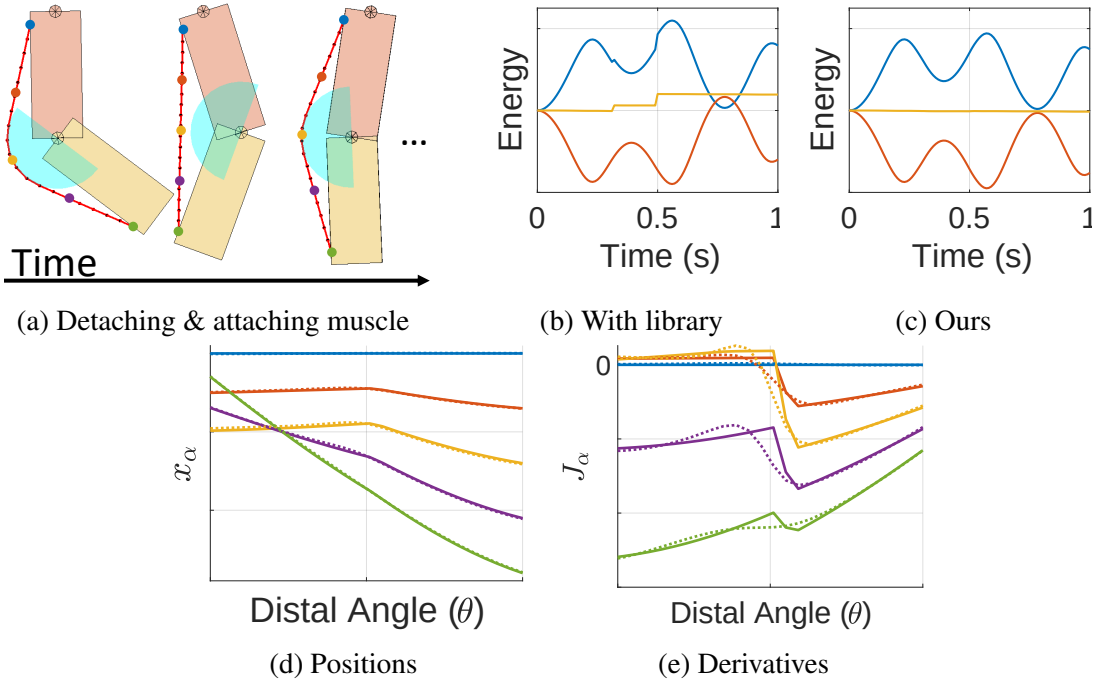
II: Polyline paths through a sequence of points, whose Jacobians are derived by extending the Eulerian-on-Lagrangian framework [15, 16].

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.

1.2.2 Why Neural Networks for Type III?

Existing muscle routing algorithms are highly efficient [17, 18, 19, 20], and 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*.



(a) Detaching & attaching muscle (b) With library (c) Ours

(d) Positions (e) Derivatives

Figure 1.6: (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.

As an illustration of this problem, suppose that we have a double pendulum with a musculotendon shown in Fig. 1.6a. 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. 1.6b. The reason for this discontinuity can be seen by looking at the motion of the mass points. Fig. 1.6d 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. 1.6e shows the corresponding derivatives. The jump in the value of the Jacobian manifests itself as an energy jump in the simulation. On the other hand, our neural network approach generates the smooth Jacobian plots in Fig. 1.6e, while keeping the position plots in Fig. 1.6d virtually indistinguishable from the output of the library code. This results in a smooth energy trajectory shown in Fig. 1.6c.

Because of our smooth Jacobians, our approach is compatible with various practical techniques, such as higher-order time integrators, inverse dynamics, complex joints, Hill-type muscle models, and differentiable dynamics. Importantly, as we decrease the amount of mass in the muscles, our simulator *gracefully degrades* to existing musculoskeletal simulators that do not support muscle inertia. It is even possible to *mix and match* musculotendons with/without mass, which can be useful for focusing the computational power on the most important muscles.

2. RELATED WORK*

We will first briefly discuss various methods of simulating articulated rigid bodies. And then we will review several muscle models used in computer graphics and biomechanics.

2.1 Articulated Rigid Body Dynamics

Articulated rigid body dynamics has been an active research area for many decades, especially in the field of robotics, where high-performance algorithms were required for low-power systems. A great deal of effort has been spent on both $O(n)$ and $O(n^3)$ methods, for example, to refine the performance for tree-configuration or closed-loop systems [21, 22]. Although asymptotically worse, $O(n^3)$ methods have attracted significant attention because they are more intuitive and are more easily parallelizable [23, 24, 25]. However, these methods do not work in the presence of the maximal stiffness matrix from linearly implicit integration, one of the most common integration methods in graphics [26].

The use of reduced/maximal coordinates with two-way coupling of articulated and deformable bodies has been of particular interest in computer graphics. Shinar et al. [27] used maximal coordinate dynamics with pre-stabilization, coupled with a finite element mesh. Their method achieves full two-way coupling, but their intricate time-stepping method makes it difficult to extend or to incorporate into existing simulators. Kim and Pollard [28] used reduced coordinate dynamics with explicit coupling between rigid and deformable bodies. Their method is extremely fast to evaluate; however their approach is limited to explicit integration schemes, since the reduced coordinates are integrated with the $O(n)$ recursive method. Jain and Liu [29] used reduced coordinates with fully implicit coupling between rigid and deformable bodies. However, with their formulation, each

*Part of this chapter is reprinted with permission from Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, "REDMAX: Efficient & flexible approach for articulated dynamics," *ACM Trans. Graph.*, vol. 38, Jul. 2019, and from Y. Wang, J. Verheul, S.-H. Yeo, N. K. Kalantari, and S. Sueda, "Differentiable simulation of inertial musculotendons," *ACM Transactions on Graphics*, vol. 41, Nov. 2022.

deformable body can only be influenced by a single rigid body. Liu et al. [30] used an off-the-shelf solver for maximal coordinate articulated bodies, which was time-stepped alongside a deformable body. Their coupling, however, was limited to explicit interactions—they only affected each other after taking a time step. To summarize, our method is unique in that it is capable of simultaneously using reduced coordinates, with fully implicit two-way coupling, and with the deformable mesh spanning multiple rigid bodies. We should clarify, however, that we limited our discussion to two-way coupling between articulated and deformable bodies—the works above provide many other important contributions.

There have also been a number of related works on the simulation of various phenomena using articulated rigid bodies, such as: trees [31], hair [32], and characters [33]. Linear time methods for *flexible* multibody systems have also been studied for decades, as described in the detailed survey by Wasfy and Noor [34]. Of particular importance to graphics, Bertails [35] showed that the recursive linear time approach can be used to simulate the dynamics of elastic rods. These efficient methods can only be used in the special case when all of the implicit forces are between topologically neighboring bodies (*e.g.,* joint springs), since then the topology of the reduced stiffness matrix will be the same as that of the reduced mass matrix. However, in the general case, the implicit forces are between *arbitrary* bodies, and so the recursive linear time approaches cannot be used.

2.2 Musculoskeletal Simulation

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 [36, 37, 38]. 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 [39, 40, 28]. As important as these works are to graphics (*e.g.,* commercial products [41, 42]), this thesis focuses exclusively on muscle-driven systems.

2.2.1 Line-based Methods

Line-based musculoskeletal methods were developed by adding line-of-action muscles to rigid body dynamics from robotics [43, 44]. 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 three works in computer graphics to use proper biomechanics-based muscle models are the works by Komura et al. [45, 46, 1], 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 [2] 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 [7] and swimming [8]. Wang et al. [5] simulate a variety of gaits, showing that optimizing for metabolic energy expenditure increases the realism of resulting animations. Geijtenbeek et al. [6] 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. [9] 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. [10] 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.

The strand-based muscles by Sueda et al. [3] and Sachedeva et al. [16] do have musculotendon inertia, but their systems cannot readily be used in conjunction with existing biomechanical simulators, because they explicitly simulate the two-way coupled system of muscles and bones, whereas existing simulators only work with the reduced degrees of freedom of articulated rigid bodies.

2.2.2 Volume-based Methods

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 [47] 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. [48] used eight-node brick elements with force generators between a set of linear FEM nodes. Lemos et al. [49] developed a general FEM framework that could support any nonlinear material as the background isotropic material. Ng-Thow-Hing [50] 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. [51] 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 [52] as well as facial muscles [53]. Fan et al. [54] used a blackbox deformation energy as an approximation for contractile mechanics in their volumetric muscles undergoing contact. Recently, Lee et al. [55] simulated volumetric muscles with Projective Dynamics, driven by per-element energy functions derived from a Hill-type muscle model.

Min et al. [56] 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. MAXIMAL COORDINATES*

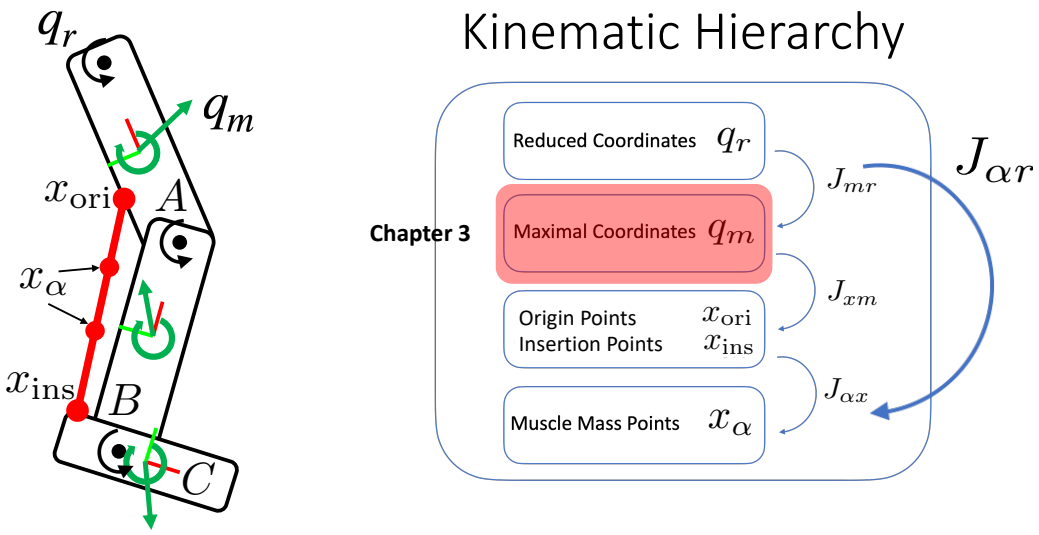


Figure 3.1: In this chapter, we dive into the kinematic hierarchy for our previously introduced example. Our focus is the mapping between reduced and maximal coordinates, represented by the Jacobian $\mathbf{J}_{mr}$. We will first introduce the maximal coordinates formulation and then later use it to derive $\mathbf{J}_{mr}$, which is the first component of the Jacobian chain.

In this chapter, we will introduce some background knowledge about maximal coordinates. (highlighted in red in Fig. 3.1). Dynamics with maximal coordinates use each rigid body's six degrees of freedom (DOFs). Constraints must be applied to model joints, and these constraints often must be stabilized to avoid drift [57, 58]. Later in Chapter 4, we will cover the other approach, reduced coordinates, which requires only a minimal set of degrees of freedom, such as joint angles for revolute joints and relative translations for prismatic joints, to represent the state of the system.

*Part of this chapter is reprinted with permission from Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, "REDMAX: Efficient & flexible approach for articulated dynamics," *ACM Trans. Graph.*, vol. 38, Jul. 2019.

Following the notation of Cline and Pai [58] and Kaufman et al. [59], we use bold letters, $\boldsymbol{x}$, to denote vectors and points in $\mathbb{R}^3$, and sans serif letters, q, to denote generalized coordinates and related quantities. Everywhere possible, q refers to the generalized coordinates of a body, and $\boldsymbol{x}$ refers to a point on the body in $\mathbb{R}^3$. Time derivatives are indicated with a dot: $\dot{\boldsymbol{x}} \equiv d\boldsymbol{x}/dt$; and material derivatives are indicated with a prime: $\boldsymbol{x}' \equiv d\boldsymbol{x}/du$.

3.1 Position Representation

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

$$ {}^{0}_{i}\mathsf{E} = \begin{pmatrix} {}^{0}_{i}\mathsf{R} & {}^{0}\boldsymbol{p} \\ 0 & 1 \end{pmatrix}. \tag{3.1} $$

The leading subscripts and superscripts indicate that the coordinates of a rigid body (or frame) i are defined with respect to the world frame, 0. Thus each column of the rotation matrix, ${}^{0}_{i}\mathsf{R}$, corresponds to the frame's basis vectors, $\boldsymbol{e}_k$, expressed in world coordinates, and ${}^{0}\boldsymbol{p}$ is the position of the frame's origin expressed in world coordinates. In other words, the first three columns of ${}^{0}_{i}\mathsf{E}$ express the i^{th} frame's x, y, and z axis directions in 0^{th} coordinate frame, and the last column of ${}^{0}_{i}\mathsf{E}$ expresses the i^{th} frame's position in the 0^{th} coordinate frame. Given a local position ${}^{i}\boldsymbol{x}$ on a rigid body, its world position is

$$ {}^{0}\boldsymbol{x} = {}^{0}_{i}\mathsf{E}\,{}^{i}\boldsymbol{x}, \tag{3.2} $$

where we have omitted the homogeneous coordinates for brevity. Unless otherwise stated, we assume that the reference frame is the world frame and use a trailing subscript to indicate the frame of a rigid body, as in E_i. With this notation, E_i transforms a position from the local space of the i^{th} rigid body to world space.

The rotation matrix, R, has the following properties:

$$ \mathsf{R}\mathsf{R}^{\top} = \mathsf{R}^{\top}\mathsf{R} = I, \quad \det(\mathsf{R}) = 1. \tag{3.3} $$

This implies that the columns of R are mutually orthonormal and follow the right-hand rule. Also, the inverse of a rotation matrix is the transpose, which is a very useful property! All 3×3 matrices with the properties above form a group called the *special orthogonal group in 3 dimensions*, or SO(3). We can write this as $\mathsf{R} \in$ SO(3).

The group of all 4×4 transformation matrices of the form Eq. 3.1 is called the *special Euclidean group in 3 dimensions*, or SE(3). We can write this as $\mathsf{E} \in$ SE(3). Because of its special structure, taking the inverse of E is easy:

$$\begin{pmatrix} \mathsf{R} & \boldsymbol{p} \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \mathsf{R}^\top & -\mathsf{R}^\top \boldsymbol{p} \\ 0 & 1 \end{pmatrix} \tag{3.4}$$

The inverse matrix reverses the transformation: ${}^0_i\mathsf{E}^{-1} = {}^i_0\mathsf{E}$. In other words, ${}^0_i\mathsf{E}$ transforms points from i to 0, whereas ${}^0_i\mathsf{E}^{-1}$ transforms points from 0 to i.

3.2 Velocity Representation

The spatial velocity (or the twist) ${}^i\phi_i$ of a rigid body ${}^0_i\mathsf{E}$ describes the motion of the rigid body at time t,

$${}^i\phi_i = \begin{pmatrix} {}^i\boldsymbol{\omega}_i \\ {}^i\boldsymbol{\nu}_i \end{pmatrix}. \tag{3.5}$$

This 6×1 vector can also be expressed as a 4×4 matrix similar to the transformation matrix in Eq. 3.1, with the rotational part in the 3×3 upper-left block and the translational part in the 3×1 upper-right block,

$$\left[{}^i\phi_i\right] = \begin{pmatrix} \left[{}^i\boldsymbol{\omega}_i\right] & {}^i\boldsymbol{\nu}_i \\ 0 & 0 \end{pmatrix}, \tag{3.6}$$

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

$$[\boldsymbol{a}] = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}. \tag{3.7}$$

The twist is related to the time derivative of the frame with:

$${}^0_i\dot{\mathsf{E}} = {}^0_i\mathsf{E} \begin{pmatrix} [{}^i\boldsymbol{\omega}_i] & {}^i\boldsymbol{\nu}_i \\ 0 & 0 \end{pmatrix}. \tag{3.8}$$

The intuition behind Eq. 3.8 is that we must multiply the twist by ${}^0_i\mathsf{E}$ to transform it to world space since ${}^0_i\dot{\mathsf{E}}$ is in world space whereas ${}^i\phi_i$ is in local space. Again, we sometimes suppress the leading superscript for brevity and write ϕ_i, assuming that all spatial velocity quantities are expressed in local coordinates. If we simplify Eq. 3.8, we see that the time derivative of the rotation matrix is its upper left 3×3 portion:

$${}^0_i\dot{\mathsf{R}} = {}^0_i\mathsf{R}\,[{}^i\boldsymbol{\omega}_i]. \tag{3.9}$$

.

3.3 Logarithms and Exponentials

Recall that:

- A rotation matrix belongs to the special orthogonal group in 3D: $\mathsf{R} \in$ SO(3).
- A rigid body configuration belongs to the special Euclidean group in 3D: $\mathsf{E} \in$ SE(3).

For each of these, there exists a corresponding velocity, or "Lie algebra" to be more technically precise:

- Angular velocity belongs to so(3), the Lie algebra of SO(3): $\boldsymbol{\omega} \in$ so(3).
- Spatial velocity belongs to se(3), the Lie algebra of SE(3): $\phi \in$ se(3).

As shown in Eq. 3.6 Angular velocity, $\boldsymbol{\omega} \in$ so(3), can also be expressed as a vector $\boldsymbol{\omega} \in \mathbb{R}^3$ or as a 3×3 skew symmetric matrix, $[\boldsymbol{\omega}]$. Similarly, spatial velocity, $\phi \in$ se(3), can also be expressed as a

vector $\phi \in \mathbb{R}^6$ or as a 4×4 matrix, $[\phi]$.

We use the matrix logarithm and exponential to go back and forth between SO(3) and so(3), as well as between SE(3) and se(3).

$$\begin{aligned} \mathsf{R} &= \exp([\boldsymbol{\omega}]), \quad [\boldsymbol{\omega}] = \log(\mathsf{R}), \\ \mathsf{E} &= \exp([\phi]), \quad [\phi] = \log(\mathsf{E}). \end{aligned} \tag{3.10}$$

Intuitive interpretation is that if a frame undergoes an angular velocity of $\boldsymbol{\omega}$ for 1 unit of time, then the frame will be rotated by R. Similarly, if a frame undergoes a spatial velocity of ϕ for 1 unit of time, then the frame will be transformed by E. For general matrices, these operations can be expensive, but for these types of matrices, there are efficient formulas.

3.4 Force Representation

Let a be a reference frame and $\boldsymbol{x}_a$ be a point in a rigid body. Suppose we have a force acting on the rigid body at that point $\boldsymbol{x}_a$, which can be represented by a vector $\boldsymbol{f}_a \in \mathbb{R}^3$. This force creates a torque or moment:

$$\tau_a = \boldsymbol{x}_a \times \boldsymbol{f}_a \in \mathbb{R}^3. \tag{3.11}$$

Then, we can introduce a spatial force or wrench by combining the torque and force into a single vector:

$$\mathsf{f}_a = \begin{pmatrix} \tau_a \\ \boldsymbol{f}_a \end{pmatrix} \in \mathbb{R}^6. \tag{3.12}$$

3.5 Material Jacobian

If a rigid body is moving with spatial velocity, ϕ_i, the world velocity of a point, ${}^i\boldsymbol{x}$, affixed to the rigid body is computed as ${}^0\dot{\boldsymbol{x}} = \mathrm{J}\phi_i$. More specifically,

$${}^0\dot{\boldsymbol{x}} = \underbrace{\mathsf{R}_i \overbrace{\begin{pmatrix} [{}^i\boldsymbol{x}]^\top & I \end{pmatrix}}^{\Gamma \in \mathbb{R}^{3\times 6}}}_{\mathrm{J} \in \mathbb{R}^{3\times 6}} \phi_i, \tag{3.13}$$

where the 3×6 matrix, J, the material Jacobian (consisted of the rotation matrix R_i and $\Gamma = ([^i\boldsymbol{x}]^\top\ I)$ that transforms twists to local point velocities), transforms the local spatial velocity of the rigid body, ϕ_i, into the velocity of a local point on the rigid body in world coordinates, ${}^0\dot{\boldsymbol{x}}$. Its transpose, a 6×3 matrix, transforms a point force in world space, $\boldsymbol{f}_0$, into a local wrench acting on the rigid body,

$$\mathsf{f}_i = \mathrm{J}^\top \boldsymbol{f}_0. \tag{3.14}$$

This "transpose" relationship works in a variety of generalized coordinate settings. If there is a matrix that maps generalized velocities into world velocities, then its transpose will map forces in world coordinates back to generalized forces.

3.6 Adjoint

Just like how 3D points and vectors must be in the same coordinate space before they can be added (and dotted, crossed, etc.), spatial velocities must also be in the same coordinate space. The spatial velocity transforms from one frame to another according to the adjoint of the coordinate transform, which is defined by the rigid transform ${}^0_i\mathsf{E}$,

$${}^0_i\mathsf{Ad} = \begin{pmatrix} {}^0_i\mathsf{R} & 0 \\ [{}^0_i\boldsymbol{p}]\,{}^0_i\mathsf{R} & {}^0_i\mathsf{R} \end{pmatrix}. \tag{3.15}$$

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

$${}^0\phi_i = {}^0_i\mathsf{Ad}\ {}^i\phi_i, \tag{3.16}$$

which is the spatial velocity of the i^{th} rigid body with respect to the world, now expressed in world coordinates.

Let's look at the time derivative of the adjoint, which we'll need to derivate the equations of

motion. Dropping the superscripts and subscripts for brevity, we have, from Eq. 3.15,

$$\dot{\mathsf{Ad}} = \begin{pmatrix} \dot{\mathsf{R}} & 0 \\ [\dot{\boldsymbol{p}}]\mathsf{R} + [\boldsymbol{p}]\dot{\mathsf{R}} & \dot{\mathsf{R}} \end{pmatrix}. \tag{3.17}$$

Looking at the rotational component of Eq. 3.9 gives us $\dot{\mathsf{R}} = \mathsf{R}[\omega]$. The point derivative, $[\dot{\boldsymbol{p}}]$, is a little trickier. (Note $[\dot{\boldsymbol{p}}] \neq [\boldsymbol{\nu}]$.) Instead, note that $\dot{\boldsymbol{p}}$ is the velocity of the frame origin expressed in world coordinates. So, $[\dot{\boldsymbol{p}}] = [\mathsf{R}\boldsymbol{\nu}]$, since $\boldsymbol{\nu}$ is expressed in local coordinates, and R rotates a vector from local to world coordinates. But $[\mathsf{R}\boldsymbol{\nu}] = \mathsf{R}[\boldsymbol{\nu}]\mathsf{R}^\top$, because for an arbitrary $\boldsymbol{x}$,

$$\begin{aligned} [\mathsf{R}\boldsymbol{\nu}](\mathsf{R}\boldsymbol{x}) &= (\mathsf{R}\boldsymbol{\nu}) \times (\mathsf{R}\boldsymbol{x}) \\ &= \mathsf{R}(\boldsymbol{\nu} \times \boldsymbol{x}) \\ &= \mathsf{R}[\boldsymbol{\nu}]\boldsymbol{x} \\ &= (\mathsf{R}[\boldsymbol{\nu}]\mathsf{R}^\top)(\mathsf{R}\boldsymbol{x}), \end{aligned} \tag{3.18}$$

and so $[\dot{\boldsymbol{p}}]\mathsf{R} = [\mathsf{R}\boldsymbol{\nu}]\mathsf{R} = \mathsf{R}[\boldsymbol{\nu}]$. So the final form of the time derivative of the adjoint is

$$\dot{\mathsf{Ad}} = \begin{pmatrix} \mathsf{R}[\boldsymbol{\omega}] & 0 \\ \mathsf{R}[\boldsymbol{\nu}] + [\boldsymbol{p}]\mathsf{R}[\boldsymbol{\omega}] & \mathsf{R}[\boldsymbol{\omega}] \end{pmatrix}. \tag{3.19}$$

This can be factored into a product of two matrices:

$$\dot{\mathsf{Ad}}(\mathsf{E}, \phi) = \underbrace{\begin{pmatrix} \mathsf{R} & 0 \\ [\boldsymbol{p}]\mathsf{R} & \mathsf{R} \end{pmatrix}}_{\mathsf{Ad}(\mathsf{E})} \underbrace{\begin{pmatrix} [\boldsymbol{\omega}] & 0 \\ [\boldsymbol{\nu}] & [\boldsymbol{\omega}] \end{pmatrix}}_{\mathsf{ad}(\phi)}, \tag{3.20}$$

where we have added the parameter list to be more 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 [60, 61].

3.7 Equations of Motion

The Newton–Euler equations of motion of a rigid body can be written in a compact form as

$$\begin{aligned} \mathsf{M}_i \dot{\phi}_i &= [\text{Coriolis forces}] + [\text{body forces (e.g., gravity)}] \\ &= \mathsf{ad}(\phi_i)^\top \mathsf{M}_i \phi_i + f_{\text{body}}(\mathsf{E}_i). \end{aligned} \tag{3.21}$$

Here, M_i is the spatial inertia of the rigid body, and $\mathsf{ad}(\phi_i)$ is the spatial cross product matrix from Eq. 3.20. If gravity is the only force involved, then the body force is

$$f_{\text{body}}(\mathsf{E}_i) = \begin{pmatrix} 0 \\ \mathsf{R}_i^\top m\boldsymbol{g} \end{pmatrix}, \tag{3.22}$$

where m is the linear mass and $\boldsymbol{g}$ is the gravity vector in world coordinates. The top zero indicates that gravity does not affect angular velocity. For the translational velocity, the multiplication by the transpose of the rotation matrix transforms the gravity direction into body coordinates.

Eq. 3.21 can be rearranged to take on the more familiar form as given by Murray et al. [62] in

their Equation (4.16). Let $\mathcal{I}$ be the rotational inertia, and mI be the translational inertia. Then

$$\begin{aligned}
\begin{pmatrix} \mathcal{I} & 0 \\ 0 & mI \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\omega}} \\ \dot{\boldsymbol{\nu}} \end{pmatrix} &= \begin{pmatrix} [\boldsymbol{\omega}]^\top & [\boldsymbol{\nu}]^\top \\ 0 & [\boldsymbol{\omega}]^\top \end{pmatrix} \begin{pmatrix} \mathcal{I} & 0 \\ 0 & mI \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{\nu} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= \begin{pmatrix} [\boldsymbol{\omega}]^\top & [\boldsymbol{\nu}]^\top \\ 0 & [\boldsymbol{\omega}]^\top \end{pmatrix} \begin{pmatrix} \mathcal{I}\boldsymbol{\omega} \\ m\boldsymbol{\nu} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= \begin{pmatrix} [\boldsymbol{\omega}]^\top \mathcal{I}\boldsymbol{\omega} + [\boldsymbol{\nu}]^\top m\boldsymbol{\nu} \\ [\boldsymbol{\omega}]^\top m\boldsymbol{\nu} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= \begin{pmatrix} [\boldsymbol{\omega}]^\top \mathcal{I}\boldsymbol{\omega} \\ [\boldsymbol{\omega}]^\top m\boldsymbol{\nu} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix} \\
&= - \begin{pmatrix} \boldsymbol{\omega} \times \mathcal{I}\boldsymbol{\omega} \\ \boldsymbol{\omega} \times m\boldsymbol{\nu} \end{pmatrix} + \begin{pmatrix} \tau \\ \boldsymbol{f} \end{pmatrix},
\end{aligned} \tag{3.23}$$

which is the same as the Newton–Euler Equation in body coordinates, as given by Murray et al. [62] in their Equation (4.16).

3.8 Maximal Inertia

Expressing the spatial velocity of a rigid body in local coordinates is advantageous in that the mass matrix is diagonal and can be precomputed at the beginning of the simulation. Wikipedia's article on "List of moments of inertia" is a good reference for some common shapes.* For a triangular mesh, we can compute the body-centered frame and its associated mass matrix using volume integration [63].

*https://en.wikipedia.org/wiki/List_of_moments_of_inertia

For example, the 6×6 mass matrix of a cuboid whose side lengths are $(\Delta x, \Delta y, \Delta z)$ is

$$\mathsf{M} = \begin{pmatrix} \frac{m}{12}(\Delta y^2 + \Delta z^2) & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{m}{12}(\Delta z^2 + \Delta x^2) & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{m}{12}(\Delta x^2 + \Delta y^2) & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \end{pmatrix}, \tag{3.24}$$

where $m = \rho \Delta x \Delta y \Delta z$ is the total mass of the cuboid with density ρ.

3.9 Euler Integration with Maximal Coordinates

For now, we will work with the simplest integration method. If we discretize the acceleration as

$$\dot{\phi} = \frac{\phi^{(k+1)} - \phi^{(k)}}{h}, \tag{3.25}$$

where $h = t^{(k+1)} - t^{(k)}$ is the time step size, then we can rewrite Eq. 3.21 to be at the velocity level at time $t^{(k)}$,

$$\mathsf{M}_i \phi_i^{(k+1)} = \mathsf{M}_i \phi_i^{(k)} + h \left(\mathsf{ad}(\phi_i^{(k)})^\top \mathsf{M}_i \phi_i^{(k)} + \mathsf{f}_{\text{body}}(\mathsf{E}_i^{(k)}) \right), \tag{3.26}$$

which can be solved for the new velocities, $\phi_i^{(k+1)}$.

The rigid body configuration $\mathsf{E}_i^{(k+1)}$ can be obtained by integrating $\phi_i^{(k+1)}$. We must be careful here because E_i belongs to a non-Euclidean space SE(3). As we mentioned before, we can use the matrix logarithm and exponential to go back and forth between SE(3) and se(3). We use the first order implicit discretization, with the time step h:

$$\mathsf{E}_i^{(k+1)} = \mathsf{E}_i^{(k)} \exp \left(h \begin{pmatrix} [\boldsymbol{\omega}_i^{(k+1)}] & \boldsymbol{\nu}_i^{(k+1)} \\ 0 & 0 \end{pmatrix} \right). \tag{3.27}$$

The matrix exponential can be computed efficiently using Rodrigues' formula [62].

3.10 Elastic and Damping Forces

Following the approach of Baraff and Witkin [26], we can add implicit damping and elastic forces using the linearly implicit Euler integration. They linearize the forces about the current position and velocity and move the resulting matrices to the left-hand-side (see §4.6.1):

$$\left(\mathsf{M} + h\mathsf{D} - h^2\mathsf{K}\right)\phi^{(k+1)} = \mathsf{M}\phi^{(k)} + h\mathsf{f}, \tag{3.28}$$

where D is the damping matrix, K is the stiffness matrix, and f is the sum of all forces. What goes into D, K, and f depends on the type of forces involved.

3.10.1 Damping Force

With simple viscous damping, $\mathsf{D} = dI$ is a diagonal matrix, where d is the damping coefficient. There is no contribution to the right-hand-side force vector, f.

3.10.2 Directional Point Force

Let's say that we want to pull on a point ${}^i\boldsymbol{x}$ on a rigid body in a particular direction ${}^0\boldsymbol{a}$, where ${}^i\boldsymbol{x}$ is in local coordinates, and ${}^0\boldsymbol{a}$ is in world coordinates. Then the linear wrench to be applied to the rigid body can be computed as follows:

$$\mathsf{f} = k\Gamma^\top \mathsf{R}^\top {}^0\boldsymbol{a}, \tag{3.29}$$

where k is the stiffness constant, Γ transforms twists to local point velocities (Eq. 3.13), and R is the rotation matrix of the rigid body. The corresponding potential energy is

$$V = -k\,{}^0\boldsymbol{x}^\top {}^0\boldsymbol{a}, \tag{3.30}$$

where ${}^0\boldsymbol{x}$ is the position of the force application point in world coordinates. The force in Eq. 3.29 is the negative gradient of this potential energy with respect to the 6 DOFs. We obtain the stiffness

matrix if we differentiate again:

$$\mathsf{K} = k \begin{pmatrix} [{}^{i}\boldsymbol{x}][\mathsf{R}^{\top}\,{}^{0}\boldsymbol{a}] & 0 \\ [\mathsf{R}^{\top}\,{}^{0}\boldsymbol{a}] & 0 \end{pmatrix}, \tag{3.31}$$

where we used the following identity for the derivatives with respect to the 6 DOFs:

$$\frac{\partial \mathsf{R}^{\top}\boldsymbol{a}}{\partial \boldsymbol{\omega}} = [\mathsf{R}^{\top}\boldsymbol{a}], \quad \frac{\partial \mathsf{R}\boldsymbol{a}}{\partial \boldsymbol{\omega}} = -\mathsf{R}[\boldsymbol{a}], \quad \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{\nu}} = \mathsf{R}. \tag{3.32}$$

The stiffness matrix is non-symmetric, which makes sense since the force is not a function of the rigid translations ($\boldsymbol{\nu}$), and so the second column is zero. We follow Baraff and Witkin [26] and symmetrize: $\mathsf{K} = \frac{1}{2}(\mathsf{K} + \mathsf{K}^{\top})$.

3.10.3 Point-to-Point Force

For a linear force between two points on two different bodies, the wrenches acting on these two bodies are

$$\mathsf{f} = k \begin{pmatrix} \Gamma_1^{\top}\mathsf{R}_1^{\top}\,\Delta\boldsymbol{x} \\ -\Gamma_2^{\top}\mathsf{R}_2^{\top}\,\Delta\boldsymbol{x} \end{pmatrix}, \tag{3.33}$$

where $\Delta\boldsymbol{x} = {}^{0}\boldsymbol{x}_2 - {}^{0}\boldsymbol{x}_1$, and ${}^{0}\boldsymbol{x}_1$ and ${}^{0}\boldsymbol{x}_2$ are the world coordinate positions of the two points, which are obtained by transforming the corresponding local coordinate positions. This force is the negative gradient of the potential energy:

$$V = \frac{1}{2}k\Delta\boldsymbol{x}^{\top}\Delta\boldsymbol{x}. \tag{3.34}$$

As before, we obtain the stiffness matrix by differentiating the force with respect to the DOFs:

$$\mathsf{K} = k \begin{pmatrix} [{}^1\boldsymbol{x}_1][\mathsf{R}_1^\top (\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & [{}^1\boldsymbol{x}_1] & [{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top\mathsf{R}_2[{}^2\boldsymbol{x}_2] & -[{}^1\boldsymbol{x}_1]\mathsf{R}_1^\top\mathsf{R}_2 \\ [\mathsf{R}_1^\top (\boldsymbol{p}_1 - {}^0\boldsymbol{x}_2)] & I & \mathsf{R}_1^\top\mathsf{R}_2[{}^2\boldsymbol{x}_2] & -\mathsf{R}_1^\top\mathsf{R}_2 \\ [{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top\mathsf{R}_1[{}^1\boldsymbol{x}_1] & -[{}^2\boldsymbol{x}_2]\mathsf{R}_2^\top\mathsf{R}_1 & [{}^2\boldsymbol{x}_2][\mathsf{R}_2^\top (\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & [{}^2\boldsymbol{x}_2] \\ \mathsf{R}_2^\top\mathsf{R}_1[{}^1\boldsymbol{x}_1] & -\mathsf{R}_2^\top\mathsf{R}_1 & [\mathsf{R}_2^\top (\boldsymbol{p}_2 - {}^0\boldsymbol{x}_1)] & I \end{pmatrix}. \quad (3.35)$$

Again, we symmetrize this: $\mathsf{K} = \frac{1}{2}(\mathsf{K} + \mathsf{K}^\top)$.

3.11 Constraints

3.11.1 Joint Constraints

Joint constraints between rigid bodies are implemented using the adjoint formulation [62], from which we can easily derive various types of joints simply by dropping different rows in the 6×6 adjoint matrix. Given two rigid bodies, i and k, and a joint frame defined with respect to the first body, ${}^i_j\mathsf{E}$, we constrain the rigid bodies' spatial velocities, ϕ_i and ϕ_k, with respect to the joint frame. Using Eq. 3.16, the relative velocity at joint j is given by

$$\begin{aligned} \delta\phi_j &= {}^j_i\mathsf{Ad}\,\phi_i - {}^j_k\mathsf{Ad}\,\phi_k \\ &= \begin{pmatrix} {}^j_i\mathsf{Ad} & -{}^j_i\mathsf{Ad}\,{}^i_0\mathsf{Ad}\,{}^0_k\mathsf{Ad} \end{pmatrix} \begin{pmatrix} \phi_i \\ \phi_k \end{pmatrix}. \end{aligned} \quad (3.36)$$

(Note ${}^i_0\mathsf{Ad} = {}^0_i\mathsf{Ad}^{-1}$.) For a rigid joint, we want the relative velocities to be zero, so we set $\delta\phi = 0$. From this, we can derive different types of joints by dropping various rows of the constraint equation: for example, the top three rows (corresponding to the three rotational DoFs) for a ball joint or the third row (corresponding to the rotation about the z-axis) for a hinge joint.

Setting $\delta\phi = 0$, we can write Eq. 3.36 in matrix form as $\mathsf{G}\Phi = 0$, where $\mathsf{G} \in \mathbb{R}^{6\times12}$ and $\Phi \in \mathbb{R}^{12\times1}$. (This is for a fixed joint. For a hinge joint, $\mathsf{G} \in \mathbb{R}^{5\times12}$.) As we add more bodies and joints, we add more rows to this "constraint" matrix. For example, if we have 3 bodies and 2 hinge joints between them, then G will have $5 + 5 = 10$ rows and $6 + 6 + 6 = 18$ columns. The entries in

G need to line up so that the correct terms get multiplied by each other. For example, if the two hinge joints are between bodies 1 and 2, and between 1 and 3, the constraint equation is

$$\begin{pmatrix} \mathsf{G}_{11} & \mathsf{G}_{12} & 0 \\ \mathsf{G}_{21} & 0 & \mathsf{G}_{23} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{3.37}$$

Before we incorporate these constraints into dynamics, let's first simplify the notation of Eq. 3.26, by letting $\phi_i = \phi_i^{(k+1)}$ and $\tilde{\mathsf{f}}_i = \mathsf{M}_i \phi_i^{(k)} + h\left(\mathsf{ad}(\phi_i^{(k)})^\top \mathsf{M}_i \phi_i^{(k)} + \mathsf{f}_{\text{body}}\left(\mathsf{E}_i^{(k)}\right)\right)$. Then we have $\mathsf{M}_i \phi_i = \tilde{\mathsf{f}}_i$, which is a 6×6 linear system. If we combine all bodies, we get $\mathsf{M}\Phi = \tilde{\mathsf{f}}$, which is a $6n \times 6n$ linear system, where n is the number of bodies. We can think of this linear system as the solution to the following quadratic minimization problem:

$$\underset{\Phi}{\text{minimize}} \quad \frac{1}{2}\Phi^\top \mathsf{M}\Phi - \Phi^\top \tilde{\mathsf{f}}. \tag{3.38}$$

To this quadratic objective, we add the equality constraint equation $\mathsf{G}\Phi = 0$, giving us

$$\begin{aligned} \underset{\Phi}{\text{minimize}} \quad & \frac{1}{2}\Phi^\top \mathsf{M}\Phi - \Phi^\top \tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{G}\Phi = 0. \end{aligned} \tag{3.39}$$

Since the maximal mass matrix, M, is always positive definite, the objective is convex, and using duality, we can convert this quadratic minimization problem into a single linear system, called a Karush-Kuhn-Tucker (KKT) system [64],

$$\begin{pmatrix} \mathsf{M} & \mathsf{G}^\top \\ \mathsf{G} & 0 \end{pmatrix} \begin{pmatrix} \Phi \\ \lambda \end{pmatrix} = \begin{pmatrix} \tilde{\mathsf{f}} \\ 0 \end{pmatrix}. \tag{3.40}$$

The top entry in the solution vector, Φ, is the new velocities of all the rigid bodies, and the bottom entry in the solution vector, λ, is the vector of Lagrange multipliers for the constraints. The joint reaction forces can be computed as $-\mathsf{G}^\top \lambda / h$. For intuition, the top and bottom rows can be rewritten

separately:

$$\begin{aligned} \mathsf{M}\Phi + \mathsf{G}^\top \lambda &= \tilde{\mathsf{f}} \\ \mathsf{G}\Phi &= 0. \end{aligned} \tag{3.41}$$

The top equation is the original discretized Newton–Euler equation but with constraint forces added, and the bottom equation is the constraint equation from the joints.

3.11.2 Contact Constraints

First, let's assume that a single body is colliding with the ground. A collision detector, which is outside the scope of this document, returns the collision point and the collision normal. Let's assume that these are both in world coordinates: ${}^0\boldsymbol{x}$ and ${}^0\boldsymbol{n}$. What we require from the colliding rigid body is that the velocity of the collision point is positive with respect to the collision normal. Using the material Jacobian, Eq. 3.13, the velocity of the collision point is

$${}^0\dot{\boldsymbol{x}} = {}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x})\,{}^i\phi_i, \tag{3.42}$$

where ${}^i\boldsymbol{x} = {}^i_0\mathsf{E}\,{}^0\boldsymbol{x}$ is the collision point in body local coordinates. We want this world velocity to be positive with respect to the collision normal, so the final constraint is

$$\begin{aligned} {}^0\boldsymbol{n}^\top\,{}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x})\,{}^i\phi_i \geq 0 \qquad &\text{or} \\ \mathsf{N}\,{}^i\phi_i \geq 0, & \end{aligned} \tag{3.43}$$

where $\mathsf{N} = \boldsymbol{n}^\top\,\mathsf{R}\,\Gamma$. In this simple case of a single collision point on a single body, N is a 1×6 matrix. If we have multiple contact points, we can add more rows to this constraint matrix, with each row having a slightly different entry because the contact point, ${}^i\boldsymbol{x}$, is going to be different. If the collision occurs with other objects in the scene (*i.e.,* not the ground), the collision normal may also be different. If there are multiple rigid bodies colliding with the world, then N will be of size $m \times 6n$, where m is the total number of collisions, and n is the number of rigid bodies.

Now let's see how we handle collisions between bodies i and j. What we do now is to constrain the *relative* velocity between the colliding bodies. The collision detector (usually) doesn't know whether things are moving, so it will just return a list of collision points and normals as before.[†] As before, we have collision point ${}^0\boldsymbol{x}$ and normal ${}^0\boldsymbol{n}$. The relative velocity, ${}^0\boldsymbol{v}_{\text{rel}}$, between the two points in contact, expressed in world coordinates, is

$$ {}^0\boldsymbol{v}_{\text{rel}} = {}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x})\,{}^i\phi_i - {}^0_j\mathsf{R}\,\Gamma({}^j\boldsymbol{x})\,{}^j\phi_j, \tag{3.44} $$

where ${}^i\boldsymbol{x} = {}^i_0\mathsf{E}\,{}^0\boldsymbol{x}$ and ${}^j\boldsymbol{x} = {}^j_0\mathsf{E}\,{}^0\boldsymbol{x}$. We want this relative velocity to be positive with respect to the collision normal: ${}^0\boldsymbol{n}^\top\,{}^0\boldsymbol{v}_{\text{rel}} \geq 0$. In matrix form, we have the following:

$$ \begin{pmatrix} {}^0\boldsymbol{n}^\top\,{}^0_i\mathsf{R}\,\Gamma({}^i\boldsymbol{x}) & -{}^0\boldsymbol{n}^\top\,{}^0_j\mathsf{R}\,\Gamma({}^j\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} {}^i\phi_i \\ {}^j\phi_j \end{pmatrix} \geq 0. \tag{3.45} $$

Each collision between bodies takes two block columns (12 columns total) of the C matrix. By combining all collisions into the contact constraint matrix, we can write $\mathsf{C}\Phi \geq 0$. For an example filling pattern, see Eq. 3.37.

By adding the constraint to Eq. 3.38, we obtain the following quadratic program:

$$ \begin{aligned} \underset{\Phi}{\text{minimize}} \quad & \frac{1}{2}\Phi^\top\mathsf{M}\Phi - \Phi^\top\tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{C}\Phi \geq 0. \end{aligned} \tag{3.46} $$

If there are joint constraints as well, we must solve

$$ \begin{aligned} \underset{\Phi}{\text{minimize}} \quad & \frac{1}{2}\Phi^\top\mathsf{M}\Phi - \Phi^\top\tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{C}\Phi \geq 0 \\ & \mathsf{G}\Phi = 0. \end{aligned} \tag{3.47} $$

[†] *Continuous* collision detector also takes into account velocities.

3.12 Summary

We have presented some background knowledge about maximal coordinates, which is the foundation of the whole thesis. We started with how to use maximal coordinates (each rigid body has six degrees of freedom) to represent each rigid body's position, velocity, force and shown how to transform those quantities from one frame to another according to the adjoint of the coordinate transform. Then, we derived and formed the equations of motion and introduced how to apply different forces and constraints in the system. The basic knowledge of maximal coordinates will help us derive the reduced approach in the next chapter.

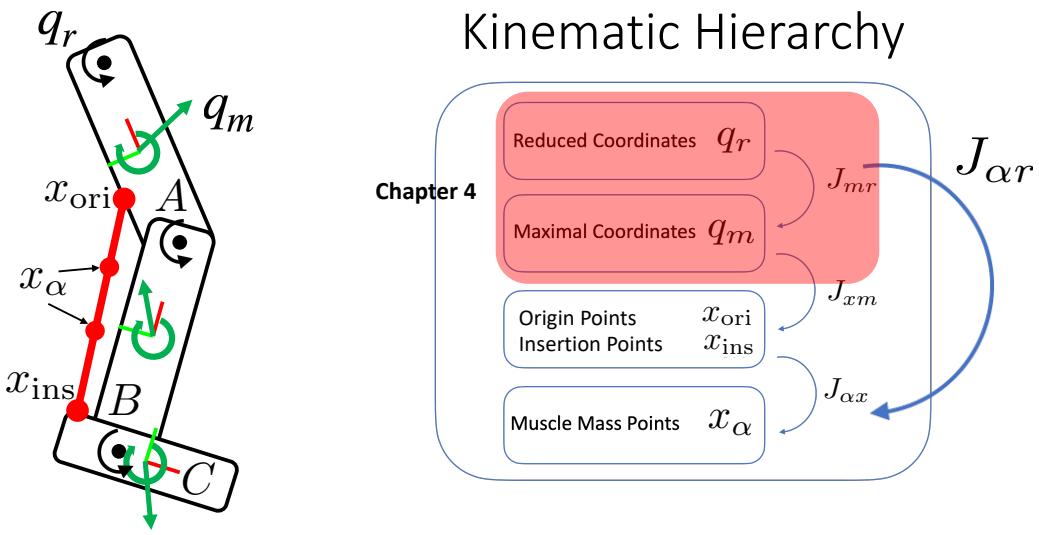


Figure 4.1: In this chapter, we continue the derivation of $\mathbf{J}_{mr}$ based on the maximal coordinates formulation we presented in the last chapter. Then, we designed and implemented the REDMAX framework that combined the advantages of both reduced and maximal coordinates. We will show how flexible it is to incorporate different forces and constraints in both coordinates, which lay the foundation for incorporating muscles.

In this chapter, we will introduce a new approach, which we call REDMAX, to solve the differential equations in the dynamic system of articulated rigid bodies in a more flexible way (highlighted in red in Fig. 4.1). Dynamics with reduced coordinates uses a minimal set of degree of freedom (DOFs), such as joint angles for revolute joints and relative translations for prismatic joints to represent the state of the system, while maximal coordinates, which we covered in Chapter 3,

*Part of this chapter is reprinted with permission from Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, "REDMAX: Efficient & flexible approach for articulated dynamics," *ACM Trans. Graph.*, vol. 38, Jul. 2019.

model all six degrees of freedom of each rigid body and need constraints to model joints.

Forward dynamics with reduced coordinates was originally developed for robotics applications [65, 66]. Unlike the maximal coordinate formulation, which requires $6n$ degrees of freedom (DOFs) and $5n$ constraints (for revolute/hinge joints), the reduced coordinate formulation requires only n DOFs and no constraints. Initially, linear time algorithm was known only for reduced coordinates, but Baraff [67] proposed a linear time algorithm that uses maximal coordinates. Baraff [67] argues that one of the advantages of maximal coordinates is that they're easier to understand and implement.

> "While $O(n)$ *inverse* reduced-coordinate approaches are easily understood, *forward* reduced-coordinate formulations with linear time complexity have an extremely steep learning curve, and make use of a formidable array of notational tools. The author admits (as do many practitioners the author has queried) to lacking a solid, intuitive understanding of these methods."

Baraff [67] goes on to argue for the use of maximal coordinates, since the resulting KKT matrix (Eq. 3.40) can be factored in linear time as long as there are no loops. (Loops can be supported for a small cost.) Baraff [67] does mention an important advantage of using reduced coordinates—lack of constraint drift. However, combining reduced coordinates with other types of simulation (*e.g.,* FEM) is again challenging. What we present here is not linear time, but is easy to understand and implement.

4.1 Updating the Transforms

Now, the main thing we need is a way to map between reduced coordinates and maximal coordinates since we want to express our system DOFs in reduced coordinates. For now, let's assume that we only have revolute (hinge) joints, so our reduced coordinates are composed of a series of joint angles. We'll also assume that there are no loops in the mechanism. Let q_r, $\dot{\mathbf{q}}_r$, and $\ddot{\mathbf{q}}_r$ be the reduced position, velocity, and acceleration. As before, for maximal coordinates, we'll also use E, ϕ, and $\dot{\phi}$ for individual rigid bodies, but we'll use q_m, $\dot{\mathbf{q}}_m$, $\ddot{\mathbf{q}}_m$, for the stacked vectors of all rigid bodies. Our goal here is to find a Jacobian, $\mathbf{J}_{mr}$, that maps reduced coordinates to maximal

coordinates:

$$\dot{\mathbf{q}}_m = \mathbf{J}_{mr}\dot{\mathbf{q}}_r. \tag{4.1}$$

Once we derive this Jacobian, we will be ready to start working out the dynamics in reduced coordinates. We start with the Newton–Euler equations of motion of rigid bodies in maximal coordinates Eq. 3.21. Instead of a single body, assume we have a system of bodies in the matrix form $\mathbf{M}_m\ddot{\mathbf{q}}_m = \mathbf{f}_m$, where $\mathbf{f}_m$ contains all forces including Coriolis forces. This system has $6n$ degrees of freedom, since it is in maximal coordinates. Using the Jacobian in Eq. 4.1, we can convert this into reduced coordinates. First, we need the mapping between reduced and maximal accelerations:

$$\ddot{\mathbf{q}}_m = \dot{\mathbf{J}}_{mr}\dot{\mathbf{q}}_r + \mathbf{J}_{mr}\ddot{\mathbf{q}}_r. \tag{4.2}$$

Therefore, we need not only the Jacobian, $\mathbf{J}$, but also its time derivative, $\dot{\mathbf{J}}$.

Incidentally, the reason inverse dynamics is easier than forward dynamics can be seen by looking at $\mathsf{f} = \mathsf{M}\ddot{\mathsf{q}}$. In inverse dynamics, we are given $\ddot{\mathsf{q}}$ and solve for f, whereas in forward dynamics, we are given f and solve for $\ddot{\mathsf{q}}$. Therefore, with inverse dynamics, we need to know how to multiply by the mass matrix, M, whereas with forward dynamics, we need to know how to multiply by the *inverse* mass matrix, M^{-1}, or *solve* with the mass matrix.

Let's take a closer look at the Jacobian. The Jacobian is easier to understand in terms of velocities, but we'll start with positions. We'll first assume that the joint hierarchy forms an acyclic graph, *i.e.,* a tree. If the system has loops, we first need to break them so that we get a spanning tree, and we will put these loops back later with constraints in §4.6.4. In a tree, each node only has one parent, with the root node having a null parent. So, we can assume that there is a one-to-one mapping between a body and a joint, and therefore we can use "body" and "joint" interchangeably. Joints and bodies always come in pairs, and for each pair, the body is always understood to be the child body connected to the joint. We will use the notation $B_0, B_1, \cdots, B_i$ for bodies and $J_0, J_1, \cdots, J_i$ for joints. B_i is the body attached to joint J_i and vice-versa.

The "world" body and joint, B_0 and J_0, coincide and are at the world origin. In Fig. 4.2, we have

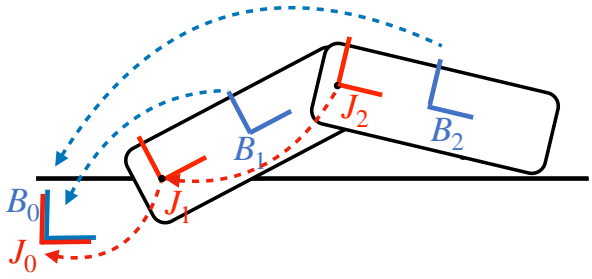


Figure 4.2: Bodies and joints. Body frames are labeled with 'B', and joint frames are labeled with 'J'. Bodies are expressed with respect to the world, and joints are expressed with respect to their parents.

two bodies with their corresponding joints. The body frames are located at the center of mass with their axes oriented according to the rotational inertia. The joint frames are located at the position of the joint on the body. With maximal coordinates, we use the configuration of each body with respect to the world as the DOFs, as indicated by the dotted blue arrows. With reduced coordinates, we use the relative configuration (*e.g.,* joint angle) between a child and its parent as the DOFs, as indicated by the dotted red arrows. So we must store the following information when coding:

- Each body i stores ${}^{0}_{B_i}\mathsf{E}$, the transformation of the i^{th} body with respect to the world. Because the 0^{th} body is the world, we use 0 instead of B_0 for the leading subscript.
- Each joint i stores ${}^{J_p}_{J_i}\mathsf{E}$, the transformation of the i^{th} joint with respect to its parent joint. If we have a serial chain, $p = i - 1$, but in the general case, $p \neq i - 1$. We use the convention that the parent index is always smaller than the child index.

Here, we again used the notation with leading superscript and subscript on E to represent these transformation matrices. These transformation matrices represent a configuration of the subscript frame with respect to the superscript frame. In other words, if we have a point ${}^{B_i}\boldsymbol{x}$ stored in B_i's coordinate frame, its position in the world coordinate frame is ${}^{0}\boldsymbol{x} = {}^{0}_{B_i}\mathsf{E}\ {}^{B_i}\boldsymbol{x}$.

Additionally, as we show in Fig. 4.3, we decompose the joint transform into two parts: $\overline{J}_i$, which represents where the joint is with respect to its parent at rest; and $Q_i(\mathsf{q}_i)$, which represents the

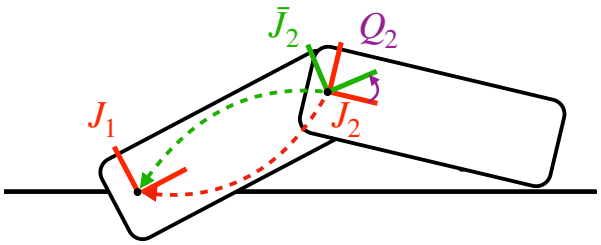


Figure 4.3: The transform for joint 2 decomposed into two parts. To go from J_2 to J_1, first apply Q_2 and then $\overline{J}_2$.

relative transformation due to the degree of freedom q_i. The joint transform is the product of these two: $J_i(\mathsf{q}_i) = \overline{J}_i\, Q_i(\mathsf{q}_i)$. $\overline{J}_i$ is constant and is set at initialization time, whereas $Q_i(\mathsf{q}_i)$ changes at run time depending on the current q_i. In the simple case, we only have revolute joints, $\overline{J}_i$ is a translation matrix, and $Q_i(\mathsf{q}_i)$ is a rotation matrix.

Note that Q_i and J_i represent the same frame pictorially—in Fig. 4.3, the purple and red frames are drawn on top of each other. The difference is in what they are relative to, as shown by the dotted arrows: J_i is relative with respect to J_p, and Q_i is relative with respect to $\overline{J}_i$. Using the notation with leading scripts, the transformation of J_2 with respect to J_1 is ${}^{J_1}_{J_2}\mathsf{E}(\mathsf{q}_2) = {}^{J_1}_{\overline{J}_2}\mathsf{E}\ {}^{\overline{J}_2}_{Q_2}\mathsf{E}\ {}^{Q_2}_{J_2}\mathsf{E}$, with the last transform being the identity matrix since Q_2 and J_2 are equivalent. With the simplified notation, we write $J_2(\mathsf{q}_2) = \overline{J}_2\, Q_2(\mathsf{q}_2)$.

The actual formulation for $Q_i(\mathsf{q}_i)$ depends on the joint type. For example, for a revolute joint about the Z-axis, we have

$$Q_i(\mathsf{q}_i) = {}^{\overline{J}_i}_{Q_i}\mathsf{E}(\mathsf{q}_i) = \begin{pmatrix} \cos(\mathsf{q}_i) & -\sin(\mathsf{q}_i) & 0 & 0 \\ \sin(\mathsf{q}_i) & \cos(\mathsf{q}_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.3}$$

When we derive the Jacobian, rather than using a rotation matrix directly as above, we will be using the matrix exponential, since we'll be working with spatial velocities. The rotation matrix above

can be written equivalently as

$$\mathsf{Q}_i(\mathsf{q}_i) = \exp\left([S\mathsf{q}_i]\right), \quad S = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}^\top. \tag{4.4}$$

In other words, the product $S\mathsf{q}_i$ gets constructed into a skew symmetric matrix using Eq. 3.6, which is then exponentiated to construct a transformation matrix. For other types of joints, we get other expressions for Q and S.

Although not always needed for dynamics calculations, sometimes it is useful to know where the joint frame is with respect to the world (*e.g.,* drawing the joint on the screen). For a joint i, its world transformation matrix, ${}^{0}_{J_i}\mathsf{E}$ (*i.e.,* where J_i is with respect to the world frame), can be computed by chaining the transforms matrices from the root to the joint. For a serial chain, we get

$$\begin{aligned} {}^{0}_{J_i}\mathsf{E} &= {}^{0}_{J_1}\mathsf{E}(\mathsf{q}_1)\ {}^{J_1}_{J_2}\mathsf{E}(\mathsf{q}_2) \cdots {}^{J_{i-1}}_{J_i}\mathsf{E}(\mathsf{q}_i) \\ &= J_1(\mathsf{q}_1) J_2(\mathsf{q}_2) \cdots J_i(\mathsf{q}_i) \\ &= \overline{J}_1\, Q_1(\mathsf{q}_1)\, \overline{J}_2\, Q_2(\mathsf{q}_2) \cdots \overline{J}_i\, Q_i(\mathsf{q}_i). \end{aligned} \tag{4.5}$$

In some situations, we also need to compute where the body is with respect to the world (*e.g.,* drawing the body on the screen or applying a maximal force to the bodies). This can be done by first computing where the joint is with respect to the world and then right multiplying by ${}^{J_i}_{B_i}\mathsf{E}$, the transform of B_i with respect to J_i. This represents where the body is with respect to the joint, and is constant over time.

4.2 Jacobian

We just saw how we compute the transform of the joint frame. To derive the dynamics, we also need to take into account the velocities. In particular, we need a way to compute the maximal (body) velocities, since the inertia is stored with respect to maximal coordinates. In this section, we will now derive the Jacobian to map between maximal and reduced velocities: $\Phi = \mathrm{J}\dot{\mathsf{q}}$, where Φ is the vector of maximal velocities, and $\dot{\mathsf{q}}$ is the vector of reduced velocities. (We wrote this as $\dot{\mathbf{q}}_m = \mathbf{J}_{mr}\dot{\mathbf{q}}_r$ before.)

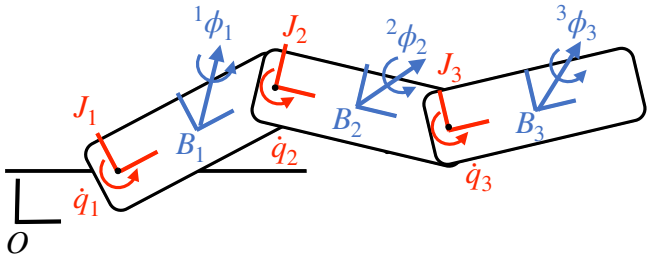


Figure 4.4: Velocities of bodies and joints. Joint J_i is between body B_i and its parent body B_{i-1}.

We'll use the following notation. An arbitrary body B_i in a tree hierarchy has a parent B_p, with a joint J_i between them. The joint will be assigned to B_i instead of B_p, since there is a one-to-one mapping between a body and a joint to the body's parent. On the other hand, B_p may have multiple children, so assigning the joint to B_p instead of to B_i can make the code more complex. For example, in Fig. 4.4, we have B_2 and its parent B_1 with J_2 between them.

The derivation of the Jacobian will be similar, but different from what we just saw in §4.1. We start by computing the relative twist between B_p and B_i at J_i. The twists of the two bodies B_p and B_i represented in J_i's frame are:

$$^{J_i}\phi_{B_p} = {}^{J_i}_{B_p}\mathsf{Ad}\ {}^{B_p}\phi_{B_p}, \quad {}^{J_i}\phi_{B_i} = {}^{J_i}_{B_i}\mathsf{Ad}\ {}^{B_i}\phi_{B_i}. \tag{4.6}$$

Their difference is the twist of J_i in its own frame:

$$\begin{aligned} {}^{J_i}\phi_{J_i} &= {}^{J_i}\phi_{B_i} - {}^{J_i}\phi_{B_p} \\ &= {}^{J_i}_{B_i}\mathsf{Ad}\ {}^{B_i}\phi_{B_i} - {}^{J_i}_{B_p}\mathsf{Ad}\ {}^{B_p}\phi_{B_p} \\ &= {}^{J_i}_{B_i}\mathsf{Ad}\ {}^{B_i}\phi_{B_i} - {}^{J_i}_{B_i}\mathsf{Ad}\ {}^{B_i}_{0}\mathsf{Ad}\ {}^{0}_{B_p}\mathsf{Ad}\ {}^{B_p}\phi_{B_p}, \end{aligned} \tag{4.7}$$

where 0 indicates the world frame. ${}^{J_i}_{B_i}\mathsf{Ad}$ is constant, and is constructed from ${}^{J_i}_{B_i}\mathsf{E}$, which represents where the i's body frame is with respect to the joint frame, which is set at initialization. Remember that in maximal coordinates, we store positions with respect to the world and velocities with respect

to the body itself. In other words, for each body, we store ${}^{0}_{B_i}\mathsf{E}$ and ${}^{B_i}\phi_{B_i}$. So in the above expression, the adjoint matrices of the form ${}^{0}_{B_i}\mathsf{Ad}$ and ${}^{B_i}_{0}\mathsf{Ad}$ can be computed easily from ${}^{0}_{B_i}\mathsf{E}$. We can rearrange this to solve for B_i's spatial velocity.

$$\begin{aligned} {}^{J_i}_{B_i}\mathsf{Ad}\ {}^{B_i}\phi_{B_i} &= {}^{J_i}_{B_i}\mathsf{Ad}\ {}^{B_i}_{0}\mathsf{Ad}\ {}^{0}_{B_p}\mathsf{Ad}\ {}^{B_p}\phi_{B_p} + {}^{J_i}\phi_{J_i} \\ {}^{B_i}\phi_{B_i} &= {}^{B_i}_{0}\mathsf{Ad}\ {}^{0}_{B_p}\mathsf{Ad}\ {}^{B_p}\phi_{B_p} + {}^{B_i}_{J_i}\mathsf{Ad}\ {}^{J_i}\phi_{J_i}. \end{aligned} \tag{4.8}$$

What this expression implies is that if we know the parent's velocity, ${}^{B_p}\phi_{B_p}$, and the joint's velocity, ${}^{J_i}\phi_{J_i}$, we can compute the child's velocity, ${}^{B_i}\phi_{B_i}$. In reduced coordinates, we parameterize ${}^{J_i}\phi_{J_i}$ not with the full 6 degrees of freedom but with some subset $\in \mathbb{R}^6$. For example, assuming we're using revolute joints about the Z axis, we can write

$${}^{J_i}\phi_{J_i} = S\dot{\mathsf{q}}_i, \quad S = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}^{\top}. \tag{4.9}$$

We use S here to follow the notation of Park et al. [66] and Kim [61]. S takes on this simple form for revolute joints, but it gets quite complicated for spherical joints, as we'll see later in §4.5. Combining Eq. 4.8 and Eq. 4.9, we get

$${}^{B_i}\phi_{B_i} = \underbrace{{}^{B_i}_{0}\mathsf{Ad}\ {}^{0}_{B_p}\mathsf{Ad}}_{{}^{B_i}_{B_p}\mathsf{Ad}}\ {}^{B_p}\phi_{B_p} + \underbrace{{}^{B_i}_{J_i}\mathsf{Ad}\ S}_{{}^{B_i}_{J_i}\mathsf{Ad}_S}\dot{\mathsf{q}}_i \tag{4.10}$$

This relationship can be recursively applied to get the system Jacobian. Let's assume we have a serial chain and use 0, 1, 2, ..., instead of p and i. So we have

$${}^{B_1}\phi_{B_1} = {}^{B_1}_{B_0}\mathsf{Ad}\,{}^{B_0}\phi_{B_0} + {}^{B_1}_{J_1}\mathsf{Ad}_S\,\dot{\mathsf{q}}_1, \tag{4.11}$$

but we can assume that the world frame is stationary, so ${}^{B_0}\phi_{B_0} = 0$. Continuing recursively,

$$\begin{aligned}
{}^{B_2}\phi_{B_2} &= {}^{B_2}_{B_1}\mathsf{Ad}\,{}^{B_1}\phi_{B_1} + {}^{B_2}_{J_2}\mathsf{Ad}_S\,\dot{\mathsf{q}}_2 \\
&= {}^{B_2}_{B_1}\mathsf{Ad}\,\left({}^{B_1}_{J_1}\mathsf{Ad}_S\,\dot{\mathsf{q}}_1\right) + {}^{B_2}_{J_2}\mathsf{Ad}_S\,\dot{\mathsf{q}}_2 \\
&= {}^{B_2}_{B_1}\mathsf{Ad}\,{}^{B_1}_{J_1}\mathsf{Ad}_S\,\dot{\mathsf{q}}_1 + {}^{B_2}_{J_2}\mathsf{Ad}_S\,\dot{\mathsf{q}}_2, \\
{}^{B_3}\phi_{B_3} &= {}^{B_3}_{B_2}\mathsf{Ad}\,{}^{B_2}\phi_{B_2} + {}^{B_3}_{J_3}\mathsf{Ad}_S\,\dot{\mathsf{q}}_3 \\
&= {}^{B_3}_{B_2}\mathsf{Ad}\,\left({}^{B_2}_{B_1}\mathsf{Ad}\,{}^{B_1}_{J_1}\mathsf{Ad}_S\,\dot{\mathsf{q}}_1 + {}^{B_2}_{J_2}\mathsf{Ad}_S\,\dot{\mathsf{q}}_2\right) + {}^{B_3}_{J_3}\mathsf{Ad}_S\,\dot{\mathsf{q}}_3 \\
&= {}^{B_3}_{B_2}\mathsf{Ad}\,{}^{B_2}_{B_1}\mathsf{Ad}\,{}^{B_1}_{J_1}\mathsf{Ad}_S\,\dot{\mathsf{q}}_1 + {}^{B_3}_{B_2}\mathsf{Ad}\,{}^{B_2}_{J_2}\mathsf{Ad}_S\,\dot{\mathsf{q}}_2 + {}^{B_3}_{J_3}\mathsf{Ad}_S\,\dot{\mathsf{q}}_3.
\end{aligned} \tag{4.12}$$

The pattern here is that initially, ${}^{B_1}\phi_{B_1}$ is just a function of $\dot{\mathsf{q}}_1$, but as we traverse the tree, ${}^{B_i}\phi_{B_i}$ becomes a function of all the ancestors of i. For a serial chain, this implies a lower triangular matrix.

$$\underbrace{\begin{pmatrix} {}^{B_1}\phi_{B_1} \\ {}^{B_2}\phi_{B_2} \\ {}^{B_3}\phi_{B_3} \end{pmatrix}}_{\dot{\mathbf{q}}_m} = \underbrace{\begin{pmatrix} {}^{B_1}_{J_1}\mathsf{Ad}_S & 0 & 0 \\ {}^{B_2}_{B_1}\mathsf{Ad}\,{}^{B_1}_{J_1}\mathsf{Ad}_S & {}^{B_2}_{J_2}\mathsf{Ad}_S & 0 \\ {}^{B_3}_{B_2}\mathsf{Ad}\,{}^{B_2}_{B_1}\mathsf{Ad}\,{}^{B_1}_{J_1}\mathsf{Ad}_S & {}^{B_3}_{B_2}\mathsf{Ad}\,{}^{B_2}_{J_2}\mathsf{Ad}_S & {}^{B_3}_{J_3}\mathsf{Ad}_S \end{pmatrix}}_{\mathbf{J}_{mr}} \underbrace{\begin{pmatrix} \dot{\mathsf{q}}_1 \\ \dot{\mathsf{q}}_2 \\ \dot{\mathsf{q}}_3 \end{pmatrix}}_{\dot{\mathbf{q}}_r}. \tag{4.13}$$

Note the recursive structure here. To fill a matrix element to the left of the diagonal, we take the element above and premultiply it by ${}^{B_i}_{B_{i-1}}\mathsf{Ad}$. As we iterate over all the columns to the left of the diagonal, what we are doing is that we are backtracing the ancestors all the way to the root. For a general tree structure, we cannot assume that the row directly above belongs to the parent. Instead, the parent is on some row above the current row.

The pseudocode of the Jacobian filling function is given in Alg. 1. Remember that the size of the Jacobian is $m \times r$, the number of maximal DOFs by the number of reduced DOFs. Therefore, the row are indexed using maximal (body) indices, and the columns are indexed using reduced (joint) indices. In the pseudocode, the notation $\mathrm{J}(B_i, J_i)$ implies using body B_i's indices for the rows and joint J_i's indices for the columns. The function is called in a forward traversal order, starting from the root joint. With this ordering, the parent is guaranteed to be processed before its children. 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. In the following, we use J_{ij} to denote matrix entries that have already been computed:

$$\begin{pmatrix} {}^{B_1}_{J_1}\mathsf{Ad}_S & 0 & 0 \\ \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot \end{pmatrix} \Rightarrow \begin{pmatrix} \mathrm{J}_{11} & 0 & 0 \\ {}^{B_2}_{B_1}\mathsf{Ad}\,\mathrm{J}_{11} & {}^{B_2}_{J_2}\mathsf{Ad}_S & 0 \\ \cdot & \cdot & \cdot \end{pmatrix} \Rightarrow \begin{pmatrix} \mathrm{J}_{11} & 0 & 0 \\ \mathrm{J}_{21} & \mathrm{J}_{22} & 0 \\ {}^{B_3}_{B_2}\mathsf{Ad}\,\mathrm{J}_{21} & {}^{B_3}_{B_2}\mathsf{Ad}\,\mathrm{J}_{22} & {}^{B_3}_{J_3}\mathsf{Ad}_S \end{pmatrix} \Rightarrow \begin{pmatrix} \mathrm{J}_{11} & 0 & 0 \\ \mathrm{J}_{21} & \mathrm{J}_{22} & 0 \\ \mathrm{J}_{31} & \mathrm{J}_{32} & \mathrm{J}_{33} \end{pmatrix}. \tag{4.14}$$

Since this matrix has $O(n^2)$ elements, it takes $O(n^2)$ time to fill, even with this recursive structure. If we only need the product of this matrix with a vector, we only need $O(n)$ time, a strategy taken by the recursive forward dynamics algorithm [65, 66, 61].

Algorithm 1 Filling the Jacobian matrix

while forward traversal **do** ▷ starting with root
  B_i = J_i's body
  $\mathrm{J}(B_i, J_i) = {}^{B_i}_{J_i}\mathsf{Ad}\ S(\mathsf{q}_i)$ ▷ Diagonal element
  J_p = J_i's parent joint
  B_p = J_p's body
  Form ${}^{B_i}_{B_p}\mathsf{Ad}(\mathsf{q}_i)$
  J_a = J_p ▷ J_i's ancestor, starting with J_i's parent
  while J_a != null **do**
    $\mathrm{J}(B_i, J_a) = {}^{B_i}_{B_p}\mathsf{Ad}(\mathsf{q}_i)\ \mathrm{J}(B_p, J_a)$ ▷ Off-diagonal element
    J_a = J_a's parent joint

4.3 Jacobian Time Derivative

As we saw in Eq. 4.2, we require the time derivative of J, which in turn requires the time derivative of the adjoint, $\dot{\mathsf{Ad}}$. For the diagonal terms, the derivative is

$$\dot{\mathrm{J}}(i,i) = {}^{i}_{J_i}\mathsf{Ad}\,\dot{S}, \tag{4.15}$$

which is 0 for revolute joints, since S is constant. (It isn't 0 for some other types of joints.) For off-diagonal terms, recall that we have the recurrence relation $\mathrm{J}(i,a) = {}^{i}_{p}\mathsf{Ad}\,\mathrm{J}(p,a)$, where p is the parent of i, and a is an ancestor of i. From this, we see that the derivative is

$$\dot{\mathrm{J}}(i,a) = {}^{i}_{p}\dot{\mathsf{Ad}}\,\mathrm{J}(p,a) + {}^{i}_{p}\mathsf{Ad}\,\dot{\mathrm{J}}(p,a). \tag{4.16}$$

To compute ${}^{i}_{p}\dot{\mathsf{Ad}}$, we use Eq. 3.19 and the identity for taking the derivative of the matrix inverse: $\dot{A}^{-1} = -A^{-1}\dot{A}A^{-1}$.

$$\begin{aligned}
{}^{i}_{p}\dot{\mathsf{Ad}} &= \frac{d}{dt}\left({}^{i}_{0}\mathsf{Ad}\,{}^{0}_{p}\mathsf{Ad}\right) \\
&= {}^{i}_{0}\dot{\mathsf{Ad}}\,{}^{0}_{p}\mathsf{Ad} + {}^{i}_{0}\mathsf{Ad}\,{}^{0}_{p}\dot{\mathsf{Ad}} \\
&= {}^{0}_{i}\dot{\mathsf{Ad}}^{-1}\,{}^{0}_{p}\mathsf{Ad} + {}^{i}_{0}\mathsf{Ad}\,{}^{0}_{p}\dot{\mathsf{Ad}} \\
&= -{}^{i}_{0}\mathsf{Ad}\,{}^{0}_{i}\dot{\mathsf{Ad}}\,{}^{i}_{0}\mathsf{Ad}\,{}^{0}_{p}\mathsf{Ad} + {}^{i}_{0}\mathsf{Ad}\,{}^{0}_{p}\dot{\mathsf{Ad}}.
\end{aligned} \tag{4.17}$$

The function is called in a forward traversal order, starting from the root. In this ordering, the parent is guaranteed to be processed before its children.

Algorithm 2 Filling the Jacobian matrix and its time derivative

1: **while** forward traversal **do**
2: $\quad\mathrm{J}(i,i) = {}^{i}_{J_i}\mathsf{Ad}\,S$ ▷ Diagonal element
3: $\quad\dot{\mathrm{J}}(i,i) = {}^{i}_{J_i}\mathsf{Ad}\,\dot{S}$ ▷ Diagonal element
4: $\quad$ancestor = parent
5: $\quad$**while** ancestor != null **do**
6: $\quad\quad\mathrm{J}(i,a) = {}^{i}_{p}\mathsf{Ad}\,\mathrm{J}(p,a)$ ▷ Off-diagonal element
7: $\quad\quad\dot{\mathrm{J}}(i,a) = {}^{i}_{p}\dot{\mathsf{Ad}}\,\mathrm{J}(p,a) + {}^{i}_{p}\mathsf{Ad}\,\dot{\mathrm{J}}(p,a)$ ▷ Off-diagonal element
8: $\quad\quad$ancestor = ancestor's parent

4.4 REDMAX Dynamics

Now that we have both $\mathbf{J}_{mr}$ and $\dot{\mathbf{J}}_{mr}$, we can finally form the reduced equations of motion. Combining Eq. 3.21 and Eq. 4.2, we have

$$\begin{aligned}
\mathbf{M}_m \left(\mathbf{J}_{mr}\, \ddot{\mathbf{q}}_r + \dot{\mathbf{J}}_{mr}\, \dot{\mathbf{q}}_r \right) &= \mathbf{f}_m \\
\mathbf{M}_m\, \mathbf{J}_{mr} \ddot{\mathbf{q}}_r &= \mathbf{f}_m - \mathbf{M}_m\, \dot{\mathbf{J}}_{mr}\, \dot{\mathbf{q}}_r \\
\left(\mathbf{J}_{mr}^\top \mathbf{M}_m\, \mathbf{J}_{mr} \right) \ddot{\mathbf{q}}_r &= \mathbf{J}_{mr}^\top \left(\mathbf{f}_m - \mathbf{M}_m\, \dot{\mathbf{J}}_{mr}\, \dot{\mathbf{q}}_r \right) \\
\mathsf{M}_r \ddot{\mathbf{q}}_r &= \mathsf{f}_r,
\end{aligned} \tag{4.18}$$

where the reduced mass matrix, $\mathsf{M}_r = \mathbf{J}_{mr}^\top\, \mathbf{M}_m\, \mathbf{J}_{mr}$, and the reduced force vector, $\mathsf{f}_r = \mathbf{J}_{mr}^\top \left(\mathbf{f}_m - \mathbf{M}_m\, \dot{\mathbf{J}}_{mr}\, \dot{\mathbf{q}}_r \right)$, are much smaller than their maximal counterparts (1/6 the size for revolute joints). Furthermore, we don't require constraints, since the Jacobians automatically take care of constraints. The last term, $-\mathbf{J}_{mr}^\top\, \mathbf{M}_m\, \dot{\mathbf{J}}_{mr}\, \dot{\mathbf{q}}_r$, is the extra *quadratic velocity vector* that results due to the change of coordinates [68].

Let's first try the simple Euler integration scheme from §3.9. The acceleration in Eq. 4.18 is discretized as

$$\ddot{\mathbf{q}}_r = \frac{\dot{\mathbf{q}}_r^{(k+1)} - \dot{\mathbf{q}}_r^{(k)}}{h}, \tag{4.19}$$

which results in

$$\mathsf{M}_r \dot{\mathbf{q}}_r^{(k+1)} = \mathsf{M}_r \dot{\mathbf{q}}_r^{(k)} + h \mathsf{f}_r. \tag{4.20}$$

This is a small, dense linear system that gives the new reduced velocities, $\dot{\mathbf{q}}_r^{(k+1)}$. If desired, the maximal velocities can be computed using the Jacobian. The reduced positions are integrated as $\mathsf{q}_r{}^{(k+1)} = \mathsf{q}_r{}^{(k)} + h \dot{\mathbf{q}}_r^{(k+1)}$.

Often it is advantageous to use a more sophisticated integrator, such as MATLAB's `ode45` integrator, which allows adaptive time steps. To use these general integrators, we must convert a

2nd order ODE into a system of 1st order ODEs, by stacking the positions and velocities together.

$$\frac{d}{dt}\begin{pmatrix} \mathsf{q}_r \\ \dot{\mathsf{q}}_r \end{pmatrix} = \begin{pmatrix} \dot{\mathsf{q}}_r \\ \mathsf{M}_r^{-1}\mathsf{f}_r \end{pmatrix}. \tag{4.21}$$

An adaptive integrator is much more stable for rigid body dynamics because it takes small time steps as needed. This is important especially if there is no damping, since even a simple two-link system can result in chaotic behavior.*

With `ode45`, integrating Eq. 4.21 gives numerically the same solution as the recursive forward dynamics method outlined by Kim[61]. Because we need to invert the reduced mass matrix, our method is $O(n^3)$, whereas recursive forward dynamics is $O(n)$. (Somehow, the linear method automagically computes the product of the reduced mass matrix with the right-hand-side!) Our method, however, is much simpler to implement, easier to understand, and easier to combine with deformable object simulations (*e.g.,* FEM).

4.5 Other Joint Types

These are based on the source code by Kim [61].†

4.5.1 Fixed Joint

A fixed joint is used for rigidly attaching two bodies together. Recall that the joint transform is defined with respect to the parent joint:

$$\overset{j-1}{j}\mathsf{E} = \overset{j-1}{j}\mathsf{E}_0 \, \mathsf{Q}_j(\mathsf{q}_j), \tag{4.22}$$

where $\overset{j-1}{j}\mathsf{E}_0$ is the initial transform (often a translation), and $\mathsf{Q}_j(\mathsf{q}_j)$ is the transform that actually applies the degrees of freedom of that joint. For a fixed joint, $\mathsf{q}_j = \varnothing$, and $\mathsf{Q}_j(\mathsf{q}_j)$ is simply the 4×4 identity matrix. The joint Jacobian, S, is an empty 6×0 matrix.

*See a video of a "double pendulum" here: `https://youtu.be/U39RMUzCjiU`.

†GEAR: Geometric Engine for Articulated Rigid-body simulation `https://github.com/junggon/gear`

4.5.2 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

$$\mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} I & \boldsymbol{a}\mathsf{q}_j \\ 0 & 1 \end{pmatrix}, \tag{4.23}$$

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

$$S = \begin{pmatrix} 0 \\ \boldsymbol{a} \end{pmatrix} \in \mathbb{R}^{6\times 1}. \tag{4.24}$$

4.5.3 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

$$\mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} I & 0 & \mathsf{q}_j \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{4.25}$$

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

$$S = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{6\times 2}. \tag{4.26}$$

4.5.4 Translational Joint

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

$$\mathsf{Q}_j(\mathsf{q}_j) = \begin{pmatrix} I & \mathsf{q}_j \\ 0 & 1 \end{pmatrix}, \tag{4.27}$$

and the corresponding joint Jacobian is

$$S = \begin{pmatrix} 0 \\ I \end{pmatrix} \in \mathbb{R}^{6\times 3}. \tag{4.28}$$

4.5.5 Spherical Joint

Representing 3D rotation with reduced coordinates is nontrivial. With any 3-parameter representation, there will be a singularity somewhere. With more than 3 parameters, we require constraints, which we will get to in §4.6.4. One option for a 3-parameter rotation representation is Euler angles. (Another option is exponential coordinates, which we describe later in §4.5.8. We start here with Euler angles because they're also used for universal joints, described next in §4.5.6.) Once we choose a sequence of axes to rotate by (*e.g.,* XZX), we can multiply out the 3 rotation matrices and obtain a single rotation matrix parameterized by the 3 angles.[‡]

Recall from Eq. 4.9 that for a revolute joint, $S \in \mathbb{R}^{6\times 1}$, because $\mathsf{q} \in \mathbb{R}$. For a spherical joint parameterized by Euler angles, $S \in \mathbb{R}^{6\times 3}$, and $\mathsf{q} \in \mathbb{R}^3$. Intuitively, each column of S is the derivative of E with respect to q, expressed in local coordinates. In other words, each column i of S is defined as:

$$[S_i] \equiv \mathsf{E}^{-1}\frac{d\mathsf{E}}{d\mathsf{q}_i}, \tag{4.29}$$

where the bracket operator is from Eq. 3.6. (Note the similarity to Eq. 3.8.) By "unbracketing" this LHS matrix, we obtain the i^{th} column of S. We can simplify this a little bit because only rotations

[‡]Formulas on Wikipedia: `https://en.wikipedia.org/wiki/Euler_angles`.

are involved for a spherical joint. Since E is a rotation matrix for a spherical joint, we can instead write

$$[S_i] \equiv \mathsf{R}^\top \frac{d\mathsf{R}}{d\mathsf{q}}, \tag{4.30}$$

where the bracket operator corresponds only to the rotational part, as in Eq. 3.7.

Let's use XZX Euler angles as a concrete example. The corresponding rotation matrix is:

$$\mathsf{R} = \begin{pmatrix} c_2 & -s_2c_3 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -s_1c_3 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + s_1c_2c_3 & c_1c_3 - s_1c_2s_3 \end{pmatrix}, \tag{4.31}$$

where $c_1 = \cos(q_1), c_2 = \cos(q_2)$, etc. Q is then

$$\mathsf{Q} = \begin{pmatrix} \mathsf{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.32}$$

Since $\mathsf{q} = (q_1\ q_2\ q_3)^\top \in \mathbb{R}^3$, we take the derivative separately three times to get the three columns of S. To get the 1st column of S, we take the derivative of R with respect to q_1 and premultiply by $\mathsf{R}^\top$. After lots of cancellations, the resulting product is a skew symmetric matrix:

$$\mathsf{R}^\top \frac{d\mathsf{R}}{dq_1} = \begin{pmatrix} 0 & -s_2s_3 & -c_3s_2 \\ s_2s_3 & 0 & -c_2 \\ c_3s_2 & c_2 & 0 \end{pmatrix}. \tag{4.33}$$

If we "unbracket" this 3×3 skew symmetric matrix, we obtain a 3×1 vector. This forms the top three rows of the 1st column of S. The bottom three rows are zero, because translations are not parameterized by a spherical joint. Repeating for the 2nd and 3rd rows, we obtain the final form of

S for a spherical joint *parameterized by XZX Euler angles*:

$$S = \begin{pmatrix} c_2 & 0 & 1 \\ -c_3 s_2 & s_3 & 0 \\ s_2 s_3 & c_3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.34}$$

For this joint Jacobian, the time derivative, $\dot{S}$, is nonzero, and is needed for the computation of $\dot{\mathrm{J}}$:

$$\dot{S} = \begin{pmatrix} -s_2\dot{q}_2 & 0 & 0 \\ s_3 s_2 \dot{q}_3 - c_3 c_2 \dot{q}_2 & c_3\dot{q}_3 & 0 \\ c_2 s_3 \dot{q}_2 + s_2 c_3 \dot{q}_3 & -s_3\dot{q}_3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.35}$$

4.5.6 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:

$$\mathsf{R} = \begin{pmatrix} c_2 c_3 & -c_2 s_3 & s_2 \\ c_1 s_3 + s_1 s_2 c_3 & c_1 c_3 - s_1 s_2 s_3 & -s_1 c_2 \\ s_1 s_3 - c_1 s_2 c_3 & s_1 c_3 + c_1 s_2 s_3 & c_1 c_2 \end{pmatrix}, \tag{4.36}$$

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

$$\mathsf{R} = \begin{pmatrix} c_2 & 0 & s_2 \\ s_1 s_2 & c_1 & -s_1 c_2 \\ -c_1 s_2 & s_1 & c_1 c_2 \end{pmatrix}. \tag{4.37}$$

Q is then

$$\mathsf{Q} = \begin{pmatrix} \mathsf{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.38}$$

The joint Jacobian, S, is going to be a 6×2 matrix. As with the spherical joint, to get the 1st column of S, we take the derivative of R with respect to q_1 and premultiply by $\mathsf{R}^\top$. 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

$$S = \begin{pmatrix} c_2 & 0 \\ 0 & 1 \\ s_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{4.39}$$

The time derivative of the joint Jacobian is

$$\dot{S} = \begin{pmatrix} -s_2 \dot{q}_2 & 0 \\ 0 & 0 \\ c_2 \dot{q}_2 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{4.40}$$

4.5.7 Revolute joint

Because revolute joints are one of the simplest and most useful joints, we used it as an introductory example in §4.2. Here, we replicate the derivations for completeness. We will assume that the joint allows bending along the Z axis.

$$\mathsf{Q}(\mathsf{q}) = \exp\left([S\mathsf{q}]\right) = \begin{pmatrix} \cos(\mathsf{q}) & -\sin(\mathsf{q}) & 0 & 0 \\ \sin(\mathsf{q}) & \cos(\mathsf{q}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad S = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.41}$$

4.5.8 Spherical Joint with Exponential Coordinates

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). With exponential coordinates [69, 70], we also need to reparameterize, but we do not need to keep track of the coordinate chart.

Let $\mathsf{q} \in$ 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

$$\mathsf{R} = \exp([\mathsf{q}]), \quad \mathsf{Q} = \begin{pmatrix} \mathsf{R} & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.42}$$

The joint Jacobian, S, is computed using the derivative formula described by Gallego and Yezzi

[70]. The derivative of R with respect to q is a $3 \times 3 \times 3$ tensor, where each 3×3 slice is given by

$$\frac{\partial \mathsf{R}}{\partial q_i} = \frac{q_i[\mathsf{q}] + [[\mathsf{q}](I - \mathsf{R})\boldsymbol{e}_i]}{\mathsf{q}^\top \mathsf{q}} \mathsf{R}, \tag{4.43}$$

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

$$[S_i] = \mathsf{R}^\top \frac{\partial \mathsf{R}}{\partial q_i}. \tag{4.44}$$

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 $\partial \mathsf{R}/\partial \mathsf{q}$ by $\dot{\mathsf{q}}$, we can compute the time derivative of the rotation matrix, $\dot{\mathsf{R}}$, which is needed for $\dot{S}$:

$$\dot{\mathsf{R}} = \sum_i \frac{\partial \mathsf{R}}{\partial q_i} \dot{q}_i. \tag{4.45}$$

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

$$\frac{\partial \mathsf{R}}{\partial q_i} = A_i \mathsf{R}, \quad A_i = (B_i + C_i)d, \quad B_i = q_i[\mathsf{q}], \quad C_i = [[\mathsf{q}](I - \mathsf{R})\boldsymbol{e}_i], \quad d = \frac{1}{\mathsf{q}^\top \mathsf{q}}. \tag{4.46}$$

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

$$\begin{aligned}
\left[\dot{S}_i\right] &= \dot{\mathsf{R}}^\top A_i \mathsf{R} + \mathsf{R}^\top \dot{A}_i \mathsf{R} + \mathsf{R}^\top A_i \dot{\mathsf{R}} \\
\dot{A}_i &= \left(\dot{B}_i + \dot{C}_i\right) d + (B_i + C_i)\, \dot{d} \\
\dot{B}_i &= \dot{\mathsf{q}}_i[\mathsf{q}] + \mathsf{q}_i[\dot{\mathsf{q}}] \\
\dot{C}_i &= \left[[\dot{\mathsf{q}}](I - \mathsf{R})\boldsymbol{e}_i - [\mathsf{q}]\dot{\mathsf{R}}\boldsymbol{e}_i\right] \\
\dot{d} &= \frac{-2\mathsf{q}^\top \dot{\mathsf{q}}}{(\mathsf{q}^\top \mathsf{q})^2}.
\end{aligned} \tag{4.47}$$

Quoting Grassia [69], "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 $\|\mathsf{q}\|$ is close to 2π and if so, reparameterize as $\mathsf{q} = (1 - 2\pi/\|\mathsf{q}\|)\mathsf{q}$. Whenever q is reparameterized, we must also update $\dot{\mathsf{q}}$, S, and $\dot{S}$. To do so, we first recompute S with Eq. 4.44 using the reparameterized q. Then, we can compute the new velocities as $\dot{\mathsf{q}} = S^{-1} S_{\text{prev}} \dot{\mathsf{q}}_{\text{prev}}$. Finally, we can compute $\dot{S}$ using Eq. 4.47 with the new values of q and $\dot{\mathsf{q}}$.

4.5.9 Composite Joint

In a composite joint, two joints are composed together: $\mathsf{Q} = \mathsf{Q}_1 \mathsf{Q}_2$. 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

$$S = \begin{pmatrix} {}^2_1\mathsf{Ad}\, S_1 & S_2 \end{pmatrix} \in \mathbb{R}^{6 \times (n_1 + n_2)}, \tag{4.48}$$

where n_1 and n_2 are the number of DOFs of joints 1 and 2, respectively, and ${}^2_1\mathsf{Ad}$ 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

$$\frac{d}{dt}\left({}^2_1\mathsf{Ad}\, S_1\right) = {}^2_1\dot{\mathsf{Ad}}\, S_1 + {}^2_1\mathsf{Ad}\, \dot{S}_1. \tag{4.49}$$

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

$$\begin{aligned}
{}^2_1\dot{\mathsf{Ad}} &= -{}^2_1\mathsf{Ad}\, {}^1_2\dot{\mathsf{Ad}}\, {}^2_1\mathsf{Ad} && \text{Using the identity for the derivative of the inverse} \\
&= -{}^2_1\mathsf{Ad}\, {}^1_2\mathsf{Ad}\, \mathsf{ad}\left({}^2\phi_2\right)\, {}^2_1\mathsf{Ad} && \text{Using Eq. 3.20} \\
&= -\mathsf{ad}(S_2 \dot{\mathsf{q}}_2)\, {}^2_1\mathsf{Ad}.
\end{aligned} \tag{4.50}$$

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

$$S_2\dot{\mathsf{q}}_2 = \begin{pmatrix} 0 \\ \dot{\boldsymbol{q}}_2 \end{pmatrix} \in \mathbb{R}^6, \tag{4.51}$$

which is a translation-only twist. Combining Eq. 4.48, Eq. 4.49, and Eq. 4.50, the time derivative of S is, therefore,

$$\dot{S} = \begin{pmatrix} -\mathsf{ad}(S_2\dot{\mathsf{q}}_2)\ {}^2_1\mathsf{Ad}\, S_1 + {}^2_1\mathsf{Ad}\, \dot{S}_1 & \dot{S}_2 \end{pmatrix}. \tag{4.52}$$

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

4.5.10 2D Free Joint

A 2D free joint is a joint that is completely unconstrained in 2D. 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 2D free joint, we concatenate a X-Y planar joint and a Z revolute joint together into a composite joint:

$$\mathsf{Q} = \mathsf{Q}_1\mathsf{Q}_2, \tag{4.53}$$

where $\mathsf{Q}_1 = \mathsf{Q}_{\text{planar}}$ and $\mathsf{Q}_2 = \mathsf{Q}_{\text{revolute}}$. Their corresponding joint Jacobians are:

$$S_1 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad S_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{4.54}$$

and $\dot{S} = 0$ for both. After some simplification, the Jacobian for the 2D free joint is then

$$S = \begin{pmatrix} {}^{2}_{1}\mathsf{Ad}\, S_1 & S_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \mathsf{Q}_2^{xx} & \mathsf{Q}_2^{yx} & 0 \\ \mathsf{Q}_2^{xy} & \mathsf{Q}_2^{yy} & 0 \\ \mathsf{Q}_2^{xz} & \mathsf{Q}_2^{yz} & 0 \end{pmatrix}. \tag{4.55}$$

The bottom three rows of the first column contain the 1st row of the Q_2 matrix, and the bottom three rows of the second column contain the 2nd row of the Q_2 matrix.

4.5.11 3D Free Joint

A 3D free joint is a joint that is completely unconstrained in 3D. 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 3D free joint, we concatenate a translational joint and a spherical joint together into a composite joint:

$$\mathsf{Q} = \mathsf{Q}_1 \mathsf{Q}_2, \tag{4.56}$$

where $\mathsf{Q}_1 = \mathsf{Q}_{\text{translational}}$ and $\mathsf{Q}_2 = \mathsf{Q}_{\text{spherical}}$. Their corresponding joint Jacobians are:

$$S_1 = \begin{pmatrix} 0 \\ I \end{pmatrix}, \qquad \dot{S}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad S_2 = \begin{pmatrix} \hat{S}_2 \\ 0 \end{pmatrix}, \qquad \dot{S}_2 = \begin{pmatrix} \hat{\dot{S}}_2 \\ 0 \end{pmatrix}, \tag{4.57}$$

where $\hat{S}_2$ implies taking the top 3 rows of S_2. (The bottom 3 rows are zeros, since the 2nd joint is a spherical joint.) After some simplification, the Jacobian is:

$$S = \begin{pmatrix} {}^{2}_{1}\mathsf{Ad}\, S_1 & S_2 \end{pmatrix} = \begin{pmatrix} 0 & \hat{S}_2 \\ \mathsf{R}_2^{\top} & 0 \end{pmatrix}, \tag{4.58}$$

where R_2 is the rotational part of Q_2. The time derivative is:

$$\dot{S} = \begin{pmatrix} -\mathsf{ad}(S_2\dot{\mathsf{q}}_2)\ {}^2_1\mathsf{Ad}\, S_1 + {}^2_1\mathsf{Ad}\, \cancel{\dot{S}_1}^{\,0} & \dot{S}_2 \end{pmatrix} = \begin{pmatrix} 0 & \hat{\dot{S}}_2 \\ -[\hat{S}_2\dot{\mathsf{q}}_2]\mathsf{R}_2^\top & 0 \end{pmatrix}. \tag{4.59}$$

We can also concatenate the two joints in reverse order. This works just as well with `ode45`, but with Euler, it may cause more drift.

$$\mathsf{Q} = \mathsf{Q}_1\mathsf{Q}_2, \tag{4.60}$$

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

$$S = \begin{pmatrix} \hat{S}_1 & 0 \\ -[\boldsymbol{q}_2]\hat{S}_1 & I \end{pmatrix} \in \mathbb{R}^{6\times 6}, \tag{4.61}$$

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

$$\dot{S} = \begin{pmatrix} \hat{\dot{S}}_1 & 0 \\ -[\dot{\boldsymbol{q}}_2]\hat{S}_1 - [\boldsymbol{q}_2]\hat{\dot{S}}_1 & 0 \end{pmatrix}, \tag{4.62}$$

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

4.5.12 Spline Curve Joint

With REDMAX, it is easy to include more advanced joints, such as the Spline Joint by Lee and Terzopooulos [71]. 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:

$$C = \begin{pmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \boldsymbol{c}_3 & \boldsymbol{c}_4 \end{pmatrix}, \tag{4.63}$$

and let $B \in \mathbb{R}^{4\times4}$ be the cubic B-spline basis matrix:

$$B = \frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.64}$$

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

$$\boldsymbol{x}(q) = CB\vec{q}, \quad \vec{q} = \begin{pmatrix} 1 \\ q \\ q^2 \\ q^3 \end{pmatrix}. \tag{4.65}$$

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. 4.65 in terms of the control points, $\boldsymbol{c}_i$:

$$\boldsymbol{x}(q) = \boldsymbol{c}_1 B_1(q) + \boldsymbol{c}_2 B_2(q) + \boldsymbol{c}_3 B_3(q) + \boldsymbol{c}_4 B_4(q), \tag{4.66}$$

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.[72]:

$$\boldsymbol{x}(q) = \boldsymbol{c}_1 \tilde{B}_1(q) + \Delta\boldsymbol{c}_2 \tilde{B}_2(q) + \Delta\boldsymbol{c}_3 \tilde{B}_3(q) + \Delta\boldsymbol{c}_4 \tilde{B}_4(q), \tag{4.67}$$

where the control point differences are computed as $\Delta\boldsymbol{c}_i = \boldsymbol{c}_i - \boldsymbol{c}_{i-1}$. By equating Eq. 4.66 and

Eq. 4.67, the cumulative basis functions, $\tilde{B}_i(q)$, are:

$$\begin{aligned}
\tilde{B}_4(q) &= B_4(q) \\
\tilde{B}_3(q) &= B_3(q) + B_4(q) \\
\tilde{B}_2(q) &= B_2(q) + B_3(q) + B_4(q) \\
\tilde{B}_1(q) &= B_1(q) + B_2(q) + B_3(q) + B_4(q) = 1.
\end{aligned} \tag{4.68}$$

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

$$B'(q) = \frac{1}{6}\begin{pmatrix} -3 & 3 & -1 \\ 0 & -6 & 3 \\ 3 & 3 & -3 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 2q \\ 3q^2 \end{pmatrix}, \quad B''(q) = \frac{1}{6}\begin{pmatrix} 3 & -1 \\ -6 & 3 \\ 3 & -3 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 2 \\ 6q \end{pmatrix}, \tag{4.69}$$

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* $\mathsf{C}_i \in$ SE(3). We use the cumulative form, Eq. 4.67, but instead of subtracting to get the control point differences, we use the matrix logarithm to get the control frame differences. Following Eq. 4.67, the joint matrix can be expressed using products of exponentials instead of additions:

$$\mathsf{Q}(\mathsf{q}) = \mathsf{C}_1 \exp\left(\Delta\mathsf{C}_2\tilde{B}_2(\mathsf{q})\right) \exp\left(\Delta\mathsf{C}_3\tilde{B}_3(\mathsf{q})\right) \exp\left(\Delta\mathsf{C}_4\tilde{B}_4(\mathsf{q})\right), \tag{4.70}$$

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

The recursive method for computing the corresponding joint Jacobian, $S = [\mathsf{Q}^{-1}(\partial\mathsf{Q}/\partial\mathsf{q})]$, and Hessian, $\partial S/\partial\mathsf{q}$, are given in the appendix of the spline joints paper [71], which we reproduce in Alg. 3 for reference. Once we compute $\partial S/\partial\mathsf{q}$, $\dot{S}$ can be computed using the chain rule: $\dot{S} = (\partial S/\partial\mathsf{q})\dot{\mathsf{q}}$.

Algorithm 3 Cubic Spline Joint transform, Jacobian, and Hessian

$\mathsf{Q} = \mathsf{C}_1 \exp(\Delta \mathsf{C}_2 \tilde{B}_2(\mathsf{q}))$
$S = \Delta \mathsf{C}_2 \tilde{B}_2'(\mathsf{q})$
$\partial S / \partial \mathsf{q} = \Delta \mathsf{C}_2 \tilde{B}_2''(\mathsf{q})$
for $i = 3, 4$ **do**
    $\mathsf{Q}_i = \exp(\Delta \mathsf{C}_i \tilde{B}_i(\mathsf{q}))$
    $\mathsf{Q} = \mathsf{Q}\mathsf{Q}_i$
    $\mathsf{Ad}_i = \mathsf{Ad}(\mathsf{Q}_i^{-1})$
    $\mathsf{ad}_i = \mathsf{ad}(S)$
    $S = \Delta \mathsf{C}_i \tilde{B}_i'(\mathsf{q}) + \mathsf{Ad}_i S$
    $\partial S / \partial \mathsf{q} = \Delta \mathsf{C}_i \tilde{B}_i''(\mathsf{q}) + \mathsf{Ad}_i (\partial S / \partial \mathsf{q} + \mathsf{ad}_i \Delta \mathsf{C}_i \tilde{B}_i(\mathsf{q}))$

4.5.13 Spline Surface Joint

For the spline surface joint (called the multi-DOF spline joint by Lee and Terzopoulos [71]), we will again use uniform cubic B-splines, and we will limit ourselves to $n = 2$, which means we have a tensor product surface:

$$f(C, q_1, q_2) = \vec{q}_1^{\top} B^{\top} C B \vec{q}_2, \quad \vec{q}_i = \begin{pmatrix} 1 & q_i & q_i^2 & q_i^3 \end{pmatrix}^{\top}, \tag{4.71}$$

where B is the spline basis matrix from Eq. 4.64, and C is the 4×4 matrix of control values. The derivatives of the tensor product surface are:

$$\begin{aligned}
&\frac{\partial f}{\partial q_1} = \vec{q}_1'^{\top} B^{\top} C B \vec{q}_2, \quad \frac{\partial f}{\partial q_2} = \vec{q}_1^{\top} B^{\top} C B \vec{q}_2', \quad \vec{q}_i' = \begin{pmatrix} 0 & 1 & 2q_i & 3q_i^2 \end{pmatrix}^{\top} \\
&\frac{\partial^2 f}{\partial q_1^2} = \vec{q}_1''^{\top} B^{\top} C B \vec{q}_2, \quad \frac{\partial^2 f}{\partial q_2^2} = \vec{q}_1^{\top} B^{\top} C B \vec{q}_2'', \quad \frac{\partial^2 f}{\partial q_1 q_2} = \frac{\partial^2 f}{\partial q_2 q_1} = \vec{q}_1'^{\top} B^{\top} C B \vec{q}_2', \\
&\vec{q}_i'' = \begin{pmatrix} 0 & 0 & 2 & 6q_i \end{pmatrix}^{\top},
\end{aligned} \tag{4.72}$$

In the multi-DOF spline joint, Lee and Terzopoulos [71] 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.*, $\mathsf{E} = \mathsf{TR}$ is more intuitive than $\mathsf{E} = \mathsf{RT}$, 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:

$$\hat{e}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad \hat{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \quad \hat{e}_4 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_5 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \hat{e}_6 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (4.73)$$

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

$$\mathsf{Q}(\mathsf{q}) = \prod_{k=1}^{6} \exp\left(\hat{e}_k f(C_k, \mathsf{q})\right), \quad (4.74)$$

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

The joint Jacobian, $S \in \mathbb{R}^{6\times2}$, and Hessian, $\partial S/\partial \mathsf{q} \in \mathbb{R}^{6\times2\times2}$, can be computed recursively, as shown in Alg. 4. 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.

4.6 REDMAX Flexibilility

4.6.1 Joint Stiffness and Damping

Adding joint stiffness and damping is much easier in reduced coordinates than in maximal coordinates. We'll use linear stiffness here, but non-linear stiffness can be implemented trivially. The joint torque due to the stiffness of the joint is

$$\tau_{\text{K}} = -K\mathsf{q}_r, \quad (4.75)$$

Algorithm 4 Cubic Spline Surface Joint transform, Jacobian, and Hessian

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

where K is the scalar stiffness parameter. We assumed here that the rest state of the joint is at $\mathsf{q}_r = 0$, but again, it is trivial to have other values. This joint torque goes into the appropriate rows of the reduced force, f_r, which can simply be added to the reduced equations of motion (Eq. 4.18):

$$\left(\mathbf{J}_{mr}^{\top} \mathbf{M}_m \, \mathbf{J}_{mr}\right) \ddot{\mathbf{q}}_r = \mathsf{f}_r + \mathbf{J}_{mr}^{\top} \left(\mathbf{f}_m - \mathbf{M}_m \, \dot{\mathbf{J}}_{mr} \, \dot{\mathbf{q}}_r\right). \tag{4.76}$$

Similarly, for joint damping, the torque is

$$\tau_{\text{D}} = -D \dot{\mathbf{q}}_r, \tag{4.77}$$

where D is the scalar damping parameter.

With linearly implicit Euler integration, we evaluate the force at the next time step by expanding around the current time step [26]. For the joint stiffness force, we get:

$$\begin{aligned} {\tau_{\text{K}}}^{(k+1)} &= {\tau_{\text{K}}}^{(k)} + \frac{\partial \tau_{\text{K}}}{\partial \mathsf{q}_r} \left(\mathsf{q}_r^{(k+1)} - \mathsf{q}_r^{(k)}\right) \\ &= {\tau_{\text{K}}}^{(k)} - Kh \dot{\mathbf{q}}_r^{(k+1)}, \end{aligned} \tag{4.78}$$

since $\dot{\mathbf{q}}_r^{(k+1)} = \left(\mathsf{q}_r^{(k+1)} - \mathsf{q}_r^{(k)}\right)/h$ with implicit Euler. So with linearly implicit Euler, the joint stiffness force gets an extra "implicit" term that goes on the left hand side. Similarly, for the joint damping force, we get:

$$\begin{aligned}
{\tau_{\mathrm{D}}}^{(k+1)} &= {\tau_{\mathrm{D}}}^{(k)} + \frac{\partial \tau_{\mathrm{D}}}{\partial \dot{\mathbf{q}}_r}\left(\dot{\mathbf{q}}_r^{(k+1)} - \dot{\mathbf{q}}_r^{(k)}\right) \\
&= {\tau_{\mathrm{D}}}^{(k)} - D\left(\dot{\mathbf{q}}_r^{(k+1)} - \dot{\mathbf{q}}_r^{(k)}\right) \\
&= {\tau_{\mathrm{D}}}^{(k)} - D\dot{\mathbf{q}}_r^{(k+1)} - {\tau_{\mathrm{D}}}^{(k)} \\
&= -D\dot{\mathbf{q}}_r^{(k+1)}.
\end{aligned} \tag{4.79}$$

So with linearly implicit Euler, the joint damping force gets an "implicit" term that goes on the left hand side, and completely disappears from the right hand side. By moving all the factors of $\dot{\mathbf{q}}_r^{(k+1)}$ from both forces to the right hand side, we get

$$\left(\mathbf{J}_{mr}^{\top}\mathbf{M}_m\,\mathbf{J}_{mr} + h\mathsf{D}_r + h^2\mathsf{K}_r\right)\dot{\mathbf{q}}_r^{(k+1)} = \left(\mathbf{J}_{mr}^{\top}\mathbf{M}_m\,\mathbf{J}_{mr}\right)\dot{\mathbf{q}}_r^{(k)} + h\left(\mathsf{f}_r^{(k)} + \mathbf{J}_{mr}^{\top}\left(\mathsf{f}_m^{(k)} - \mathbf{M}_m\,\dot{\mathbf{J}}_{mr}\,\dot{\mathbf{q}}_r^{(k)}\right)\right). \tag{4.80}$$

For linear stiffness and linear damping (Eq. 4.75 & Eq. 4.77),

$$\mathsf{K}_r = -\frac{\partial \tau_{\mathrm{K}}}{\partial \mathsf{q}_r} = \begin{pmatrix} K_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & K_n \end{pmatrix}, \qquad \mathsf{D}_r = -\frac{\partial \tau_{\mathrm{D}}}{\partial \dot{\mathbf{q}}_r} = \begin{pmatrix} D_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & D_n \end{pmatrix}. \tag{4.81}$$

(Note: sometimes people move the negative signs around to get $(\mathsf{M} + h\mathsf{D} - h^2\mathsf{K})$ on the left hand side [26].) In general, we can combine the linearly implicit terms for both reduced and maximal coordinates:

$$\begin{aligned}
&\left(\mathbf{J}_{mr}^{\top}\left(\mathbf{M}_m + h\mathsf{D}_m - h^2\mathsf{K}_m\right)\mathbf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r\right)\dot{\mathbf{q}}_r^{(k+1)} \\
&\qquad = \left(\mathbf{J}_{mr}^{\top}\mathbf{M}_m\,\mathbf{J}_{mr}\right)\dot{\mathbf{q}}_r^{(k)} + h\left(\mathsf{f}_r^{(k)} + \mathbf{J}_{mr}^{\top}\left(\mathsf{f}_m^{(k)} - \mathbf{M}_m\,\dot{\mathbf{J}}_{mr}\,\dot{\mathbf{q}}_r^{(k)}\right)\right).
\end{aligned} \tag{4.82}$$

4.6.2 Hyper Reduced Coordinates

We can further reduce the degrees of freedom by chaining more Jacobians. For example, let's say we have a chain of rigid bodies connected by revolute joints, and we want the joint angle to be all the same. This can be accomplished by adding constraints as shown in §4.6.4, but if we use a chained Jacobian, we end up with a single DOF system. The reduced equation of motion from before, written out in full, is

$$\mathbf{J}_{mr}^{\top}\,\mathbf{M}_m\,\mathbf{J}_{mr}\,\ddot{\mathbf{q}}_r = \mathbf{J}_{mr}^{\top}\left(\mathbf{f}_m - \mathbf{M}_m\dot{\mathbf{J}}_{mr}\dot{\mathbf{q}}_r\right), \tag{4.83}$$

where $\dot{\mathbf{q}}_r$ contains all of the joint velocities. We now want to apply another Jacobian, so that these joint angles become the same. This can be expressed using the following relationship:

$$\begin{gathered}\begin{pmatrix}\dot{\theta}_1\\ \vdots\\ \dot{\theta}_n\end{pmatrix} = \begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}\dot{\theta}\\ \dot{\mathbf{q}}_r = \mathrm{J}_{r\mathrm{R}}\,\dot{\mathsf{q}}_{\mathrm{R}},\end{gathered} \tag{4.84}$$

where $\dot{\mathsf{q}}_{\mathrm{R}}$ represents the new (hyper) reduced coordinates. If we define

$$\mathrm{J}_{m\mathrm{R}} = \mathbf{J}_{mr}\,\mathrm{J}_{r\mathrm{R}}, \quad \dot{\mathrm{J}}_{m\mathrm{R}} = \dot{\mathbf{J}}_{mr}\,\mathrm{J}_{r\mathrm{R}} + \mathbf{J}_{mr}\,\dot{\mathrm{J}}_{r\mathrm{R}}, \tag{4.85}$$

then the hyper reduced equation of motion is

$$\mathrm{J}_{m\mathrm{R}}^{\top}\mathbf{M}_m\,\mathrm{J}_{m\mathrm{R}}\,\ddot{\mathsf{q}}_{\mathrm{R}} = \mathrm{J}_{m\mathrm{R}}^{\top}\left(\mathbf{f}_m - \mathbf{M}_m\dot{\mathrm{J}}_{m\mathrm{R}}\dot{\mathsf{q}}_{\mathrm{R}}\right). \tag{4.86}$$

In the following sections, we use lower cased subscripts (*e.g.,* $\mathbf{J}_{mr}$ instead of $\mathrm{J}_{m\mathrm{R}}$) to slightly lighten the notation, but with the understanding that the reduced coordinates can also be hyper reduced.

4.6.3 Adding Deformable Bodies

One of the strengths of the REDMAX algorithm is the ease in which deformable objects (*e.g.,* FEM) can be added. Without loss of generality, we show how this can be done with a mass-spring system. Let a spring be defined by a sequence of nodes connected in series, and let $\boldsymbol{x}$ be the nodal positions. For each node, the kinetic energy and the gravitational potential energy can be expressed as

$$T = \frac{1}{2} m \dot{\boldsymbol{x}}^\top \dot{\boldsymbol{x}}, \quad V = -m \boldsymbol{g}^\top \boldsymbol{x}, \tag{4.87}$$

where m is the mass of the node. This results in mass matrix $M = mI$ and gravity force $\mathsf{f} = m\boldsymbol{g}$. Between consecutive nodes $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$, the elastic potential energy can be expressed as

$$\begin{aligned} V &= \frac{K}{2} \varepsilon^2 \\ \varepsilon &= \frac{l - L}{L} \\ l &= \|\Delta \boldsymbol{x}\| \\ \Delta \boldsymbol{x} &= \boldsymbol{x}_1 - \boldsymbol{x}_0, \end{aligned} \tag{4.88}$$

where K is the stiffness. The corresponding force is the negative gradient of the energy:

$$\boldsymbol{f}_0 = \frac{K\varepsilon}{L} \frac{\Delta \boldsymbol{x}}{l}, \tag{4.89}$$

and $\boldsymbol{f}_1 = -\boldsymbol{f}_0$. These quantities for all of the springs can be collected into a mass matrix M_s and a force vector f_s.

We can combine this with REDMAX by modifying the Jacobian, whose job is to map reduced coordinates into maximal coordinates. Let $\boldsymbol{x}_f$ and $\boldsymbol{x}_a$ denote the "free" and "attached" vertices of the spring. To attach vertices to the rigid bodies, we work in maximal coordinates. The world

velocity of the attached vertex can be expressed as:

$$\dot{\boldsymbol{x}}_a = \underbrace{\mathsf{R}\Gamma\left({}^i\boldsymbol{x}_a\right)}_{J_{am}} \phi_i, \tag{4.90}$$

where R is the rotation matrix of the body, ${}^i\boldsymbol{x}_a$ is the position of the attached vertex in body coordinates, and $\Gamma \in \mathbb{R}^{3\times 6}$ is the material Jacobian from Eq. 3.13. The time derivative of this Jacobian, J_{am}, is:

$$\dot{J}_{am} = \mathsf{R}\left[\boldsymbol{\omega}_i\right]\Gamma. \tag{4.91}$$

By collecting all attachment Jacobians into a single global matrix, we obtain $\mathbf{J}_{\alpha m}$, which transforms maximal velocities of rigid bodies to velocities of attached vertices. Let q_f and q_a denote the concatenation of *free* and *attached* vertices. Then we have:

$$\begin{aligned}
\begin{pmatrix} \dot{\mathbf{q}}_m \\ \dot{\mathsf{q}}_a \\ \dot{\mathsf{q}}_f \end{pmatrix} &= \begin{pmatrix} \mathbf{J}_{mr} & 0 \\ \mathbf{J}_{\alpha m}\,\mathbf{J}_{mr} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_r \\ \dot{\mathsf{q}}_f \end{pmatrix} \\
\begin{pmatrix} \ddot{\mathbf{q}}_m \\ \ddot{\mathsf{q}}_a \\ \ddot{\mathsf{q}}_f \end{pmatrix} &= \begin{pmatrix} \dot{\mathbf{J}}_{mr} & 0 \\ \dot{\mathbf{J}}_{\alpha m}\,\mathbf{J}_{mr} + \mathbf{J}_{\alpha m}\,\dot{\mathbf{J}}_{mr} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_r \\ \dot{\mathsf{q}}_f \end{pmatrix} + \begin{pmatrix} \mathbf{J}_{mr} & 0 \\ \mathbf{J}_{\alpha m}\,\mathbf{J}_{mr} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}}_r \\ \ddot{\mathsf{q}}_f \end{pmatrix}.
\end{aligned} \tag{4.92}$$

The equations above allow us to express the maximal degrees of freedom as a linear function of the reduced degrees of freedom. By using an identity block to pass through the free vertices of the deformable bodies, we can use them as the reduced DOFs but at the same time create maximal quantities (mass matrix, force vector, etc.) for them, without the hassle of reduced coordinates. Let q_{M} be the concatenation of the maximal degrees of freedom: q_m, q_a, and q_f, and let q_{R} be the concatenation of the reduced degrees of freedom: q_r and q_f. Then defining J_{MR} and $\dot{\mathrm{J}}_{\text{MR}}$

appropriately, Eq. 4.92 can be expressed compactly as

$$\begin{aligned} \dot{\mathsf{q}}_{\mathrm{M}} &= \mathrm{J}_{\mathrm{MR}} \dot{\mathsf{q}}_{\mathrm{R}} \\ \ddot{\mathsf{q}}_{\mathrm{M}} &= \dot{\mathrm{J}}_{\mathrm{MR}} \dot{\mathsf{q}}_{\mathrm{R}} + \mathrm{J}_{\mathrm{MR}} \ddot{\mathsf{q}}_{\mathrm{R}}. \end{aligned} \tag{4.93}$$

We define the maximal mass matrix and the maximal force vector as

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_m & 0 & 0 \\ 0 & \mathsf{M}_a & 0 \\ 0 & 0 & \mathsf{M}_f \end{pmatrix}, \qquad \mathsf{f}_{\mathrm{M}} = \begin{pmatrix} \mathbf{f}_m \\ \mathsf{f}_a \\ \mathsf{f}_f \end{pmatrix}, \tag{4.94}$$

and the resulting reduced equation of motion is

$$\mathrm{J}_{\mathrm{MR}}^{\top} \mathbf{M} \; \mathrm{J}_{\mathrm{MR}} \, \ddot{\mathsf{q}}_{\mathrm{R}} = \mathrm{J}_{\mathrm{MR}}^{\top} \left(\mathsf{f}_{\mathrm{M}} - \mathbf{M} \, \dot{\mathrm{J}}_{\mathrm{MR}} \, \dot{\mathsf{q}}_{\mathrm{R}} \right), \tag{4.95}$$

where $\mathsf{M}_{\mathrm{R}} = \mathrm{J}_{\mathrm{MR}}^{\top} \mathbf{M} \; \mathrm{J}_{\mathrm{MR}}$ and $\mathsf{f}_{\mathrm{R}} = \mathrm{J}_{\mathrm{MR}}^{\top} \left(\mathsf{f}_{\mathrm{M}} - \mathbf{M} \, \dot{\mathrm{J}}_{\mathrm{MR}} \, \dot{\mathsf{q}}_{\mathrm{R}} \right)$ are the reduced mass matrix and force vector, respectively. In the following sections, we use lower cased subscripts (*e.g.,* $\mathbf{J}_{mr}$ instead of J_{MR}) to slightly lighten the notation, but it is important to note that "reduced coordinates" can mean rigid bodies *with or without* an attached deformable bodies.

4.6.4 Adding Constraints

The Jacobian-based dynamics (Eq. 4.18) and recursive forward dynamics [65, 66, 61] need constraints to support closed loops. These "loop-closing" constraints are implemented in a similar fashion as the joints in maximal coordinate dynamics (§3.11.1). We're looking for G such that $\mathsf{G}\Phi = 0$.

For example, let's consider forming a four-bar linkage by adding a loop-closing constraint to a system composed of four bodies in a series. The last body will be attached to the first body using a constraint. The world velocities of these two bodies, B and A, are

$${}^{0}\dot{\boldsymbol{x}}_A = {}^{0}_{A}\mathsf{R}\,\Gamma({}^{A}\boldsymbol{x}_A)\,{}^{A}\phi_A, \quad {}^{0}\dot{\boldsymbol{x}}_B = {}^{0}_{B}\mathsf{R}\,\Gamma({}^{B}\boldsymbol{x}_B)\,{}^{B}\phi_B, \tag{4.96}$$

where ${}^A\boldsymbol{x}_A$ and ${}^B\boldsymbol{x}_B$ are the positions of the constrained point expressed in A and B, respectively. We want these two velocities to be equal. We must be careful though, because if we simply form a constraint by equating these two, we get a singular system. To see why, consider the number of degrees of freedom and constraints in the system. Before adding the loop-closing constraint, the four-bar linkage has 3 degrees of freedom. If we add a 3-dimensional constraint, how many actual degrees of freedom are we left with? What would happen if we apply this 3D constraint is that we get a singular matrix with a 1D nullspace that corresponds to the actual, single degree of freedom of a four-bar linkage. Resolving this numerically is rather expensive, but fortunately there is a trivial way to get rid of this nullspace beforehand. If we look at the axis of rotation of the joint attached to A, we can obtain the two directions orthonormal to A's hinge axis. (B would work just as well.) Let ${}^0\boldsymbol{a}$ be the axis of rotation in world space. We can create two directions orthonormal to ${}^0\boldsymbol{a}$ as follows (dropping the superscript for brevity).

$$\begin{aligned}
\boldsymbol{v}_1 &= (1\ 0\ 0)^\top \qquad \text{// put 1 in the location with the smallest element in abs}(\boldsymbol{a}) \\
\boldsymbol{v}_2 &= \frac{\boldsymbol{a} \times \boldsymbol{v}_1}{\|\boldsymbol{a} \times \boldsymbol{v}_1\|} \\
\boldsymbol{v}_1 &= \frac{\boldsymbol{v}_2 \times \boldsymbol{a}}{\|\boldsymbol{v}_2 \times \boldsymbol{a}\|}.
\end{aligned} \tag{4.97}$$

Then ${}^0\boldsymbol{v}_1$ and ${}^0\boldsymbol{v}_2$ are both vectors in world space orthogonal to ${}^0\boldsymbol{a}$ and to each other. The constraint we want is that the relative velocity must be equal along these two directions.

$$\underbrace{\begin{pmatrix} {}^0\boldsymbol{v}_1^\top \\ {}^0\boldsymbol{v}_2^\top \end{pmatrix} \begin{pmatrix} {}^0_A\mathsf{R}\,\Gamma({}^A\boldsymbol{x}_A) & -{}^0_B\mathsf{R}\,\Gamma({}^B\boldsymbol{x}_B) \end{pmatrix}}_{\mathsf{G}_m} \underbrace{\begin{pmatrix} {}^A\phi_A \\ {}^B\phi_B \end{pmatrix}}_{\dot{\mathbf{q}}_m} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{4.98}$$

If we are working at the acceleration level, we need to take the time derivative, like we did for the Jacobian in Eq. 4.2. The constraint on the acceleration is

$$\mathsf{G}_m\dot{\mathbf{q}}_m = 0 \quad \rightarrow \quad \dot{\mathsf{G}}_m\dot{\mathbf{q}}_m + \mathsf{G}_m\ddot{\mathbf{q}}_m = 0. \tag{4.99}$$

For the G_m in Eq. 4.98, the only factors that depend on time are the two rotation matrices. Taking their time derivative as in Eq. 3.9,

$$\dot{\mathsf{G}}_m = \begin{pmatrix} {}^0\boldsymbol{v}_1^\top \\ {}^0\boldsymbol{v}_2^\top \end{pmatrix} \begin{pmatrix} {}^0_A\mathsf{R}\left[{}^A\boldsymbol{\omega}_A\right]\Gamma({}^A\boldsymbol{x}_A) & -{}^0_B\mathsf{R}\left[{}^B\boldsymbol{\omega}_B\right]\Gamma({}^B\boldsymbol{x}_B) \end{pmatrix}. \tag{4.100}$$

G_m and $\dot{\mathsf{G}}_m$ are both 2×12, and they get placed into global constraint matrices as discussed in §3.11.1. To apply these constraints, we form a KKT system as in Eq. 3.40. Because the constraint is being applied on the maximal coordinates, $\dot{\mathbf{q}}_m$, we must right-multiply the constraints by $\mathbf{J}_{mr}$ first to convert them to reduced coordinates. The constrained dynamics equation is then

$$\begin{pmatrix} \mathsf{M}_r & \mathbf{J}_{mr}^\top \mathsf{G}_m^\top \\ \mathsf{G}_m \mathbf{J}_{mr} & 0 \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{f}_r \\ -\left(\dot{\mathsf{G}}_m \mathbf{J}_{mr} + \mathsf{G}_m \dot{\mathbf{J}}_{mr}\right) \dot{\mathbf{q}}_r \end{pmatrix}. \tag{4.101}$$

If, instead, we are working at the velocity level, the constraint is $\mathsf{G}_m \mathbf{J}_{mr} \dot{\mathbf{q}}_r = 0$, and so we get

$$\begin{pmatrix} \mathsf{M}_r & \mathbf{J}_{mr}^\top \mathsf{G}_m^\top \\ \mathsf{G}_m \mathbf{J}_{mr} & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_r^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{M}_r \dot{\mathbf{q}}_r^{(k)} + h\mathsf{f}_r{}^{(k)} \\ 0 \end{pmatrix}. \tag{4.102}$$

If, instead, the constraint applies directly to the reduced coordinates rather than maximal coordinates, then the KKT system does not need the $\mathbf{J}_{mr}$ factors in the constraints. At the acceleration level,

$$\begin{pmatrix} \mathsf{M}_r & \mathsf{G}_r^\top \\ \mathsf{G}_r & 0 \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{f}_r \\ -\dot{\mathsf{G}}_r \dot{\mathbf{q}}_r \end{pmatrix}, \tag{4.103}$$

and at the velocity level,

$$\begin{pmatrix} \mathsf{M}_r & \mathsf{G}_r^\top \\ \mathsf{G}_r & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_r^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{M}_r \dot{\mathbf{q}}_r^{(k)} + h\mathsf{f}_r{}^{(k)} \\ 0 \end{pmatrix}. \tag{4.104}$$

Sometimes we may want both types of constraints: those acting on maximal coordinates, and

those acting on reduced coordinates. Then substituting

$$\bar{\mathsf{G}}_r = \begin{pmatrix} \mathsf{G}_r \\ \mathsf{G}_m \, \mathbf{J}_{mr} \end{pmatrix}, \quad \dot{\bar{\mathsf{G}}}_r = \begin{pmatrix} \dot{\mathsf{G}}_r \\ \dot{\mathsf{G}}_m \, \mathbf{J}_{mr} + \mathsf{G}_m \, \dot{\mathbf{J}}_{mr} \end{pmatrix} \tag{4.105}$$

into Eq. 4.103 will automatically apply both types of constraints. When expanded out, the expression turns into:

$$\begin{pmatrix} \mathsf{M}_r & \mathsf{G}_r^\top & \mathbf{J}_{mr}^\top \mathsf{G}_m^\top \\ \mathsf{G}_r & 0 & 0 \\ \mathsf{G}_m \, \mathbf{J}_{mr} & 0 & 0 \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}}_r \\ \lambda_r \\ \lambda_m \end{pmatrix} = \begin{pmatrix} \mathsf{f}_r \\ -\dot{\mathsf{G}}_r \dot{\mathbf{q}}_r \\ -\left(\dot{\mathsf{G}}_m \, \mathbf{J}_{mr} + \mathsf{G}_m \, \dot{\mathbf{J}}_{mr} \right) \dot{\mathbf{q}}_r \end{pmatrix}. \tag{4.106}$$

Quadratic programs for inequality constraints can be constructed similarly. In the most general case, we have inequality and equality constraints on both maximal and reduced coordinates, represented by constraint matrices C_m, C_r, G_m, and G_r, respectively. In addition to Eq. 4.105 and let

$$\bar{\mathsf{C}}_r = \begin{pmatrix} \mathsf{C}_r \\ \mathsf{C}_m \, \mathbf{J}_{mr} \end{pmatrix}, \quad \dot{\bar{\mathsf{C}}}_r = \begin{pmatrix} \dot{\mathsf{C}}_r \\ \dot{\mathsf{C}}_m \, \mathbf{J}_{mr} + \mathsf{C}_m \, \dot{\mathbf{J}}_{mr} \end{pmatrix}. \tag{4.107}$$

Then the resulting quadratic program is

$$\begin{aligned} \underset{\ddot{\mathbf{q}}_r}{\text{minimize}} \quad & \frac{1}{2} \ddot{\mathbf{q}}_r^\top \mathsf{M}_r \ddot{\mathbf{q}}_r - \ddot{\mathbf{q}}_r^\top \mathsf{f}_r \\ \text{subject to} \quad & \bar{\mathsf{C}}_r \ddot{\mathbf{q}}_r \geq -\dot{\bar{\mathsf{C}}}_r \dot{\mathbf{q}}_r \\ & \bar{\mathsf{G}}_r \ddot{\mathbf{q}}_r = -\dot{\bar{\mathsf{G}}}_r \dot{\mathbf{q}}_r. \end{aligned} \tag{4.108}$$

4.6.5 Hybrid Dynamics

In forward dynamics, we compute the accelerations given the forces, and in inverse dynamics, we compute the forces given the accelerations. In the REDMAX formulation, it is easy to mix these two into "hybrid" dynamics. Let p indicate the subset of joints whose motion are prescribed.

Then we can apply an equality constraint on the prescribed accelerations: ${}^{p}\mathsf{G}_r\,\ddot{\mathbf{q}}_r = {}^{p}\ddot{\mathbf{q}}_r$, where ${}^{p}\mathsf{G}_r$ contains the identity matrix in the appropriate columns so that the prescribed joints will be affected (note ${}^{p}\dot{\mathsf{G}}_r = 0$). The KKT system is then

$$\begin{pmatrix} \mathsf{M}_r & \bar{\mathsf{G}}_r^{\top} \\ \bar{\mathsf{G}}_r & 0 \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{q}}_r \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{f}_r \\ {}^{p}\ddot{\mathbf{q}}_r - \dot{\bar{\mathsf{G}}}_r\,\dot{\mathbf{q}}_r \end{pmatrix}, \tag{4.109}$$

where we have included the $-\dot{\bar{\mathsf{G}}}_r\,\dot{\mathbf{q}}_r$ term since other constraints may have non-zero $\dot{\bar{\mathsf{G}}}_r$. The required joint torques can be computed with the resulting Lagrange multiplier: ${}^{p}\tau = {}^{p}\mathsf{G}_r^{\top}\,{}^{p}\lambda$, where ${}^{p}\mathsf{G}_r$ and ${}^{p}\lambda$ are the appropriate rows and columns of G_r and λ, respectively. At the velocity level,

$$\begin{pmatrix} \mathsf{M}_r & \bar{\mathsf{G}}_r^{\top} \\ \bar{\mathsf{G}}_r & 0 \end{pmatrix} \begin{pmatrix} \dot{\mathbf{q}}_r^{(k+1)} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathsf{M}_r\dot{\mathbf{q}}_r^{(k)} + h\mathsf{f}_r{}^{(k)} \\ h\,{}^{p}\ddot{\mathbf{q}}_r + {}^{p}\mathsf{G}_r\dot{\mathbf{q}}_r^{(k)} \end{pmatrix}. \tag{4.110}$$

The 2nd row of the KKT system, $\bar{\mathsf{G}}_r\dot{\mathbf{q}}_r = h\,{}^{p}\ddot{\mathbf{q}}_r + {}^{p}\mathsf{G}_r\dot{\mathbf{q}}_r^{(k)}$, contains an extra term on the right hand side because the joint acceleration, rather than velocity, is being prescribed. It is also possible to prescribe the velocity by replacing the 2nd row with ${}^{p}\mathsf{G}_r\,\dot{\mathbf{q}}_r = {}^{p}\dot{\mathbf{q}}_r$.

4.7 Summary

In this chapter, we designed and implemented the REDMAX framework to simulate articulated rigid body. We showed how we use the reduced approach to get a denser and smaller matrix to simulate the dynamics of the system. First, we describe the formulation of our framework, which exposes all the reduced and maximal quantites. Then, we presented how flexible it is to apply implicit and explicit forces, and bilateral and unilateral constraints on both reduced and maximal coordinates. Finally, we have shown how to incorporate different constraints by forming a quadratic program, and use them to handle different applications, such as closing loops, attaching FEM nodes to bodies, and joint limits.

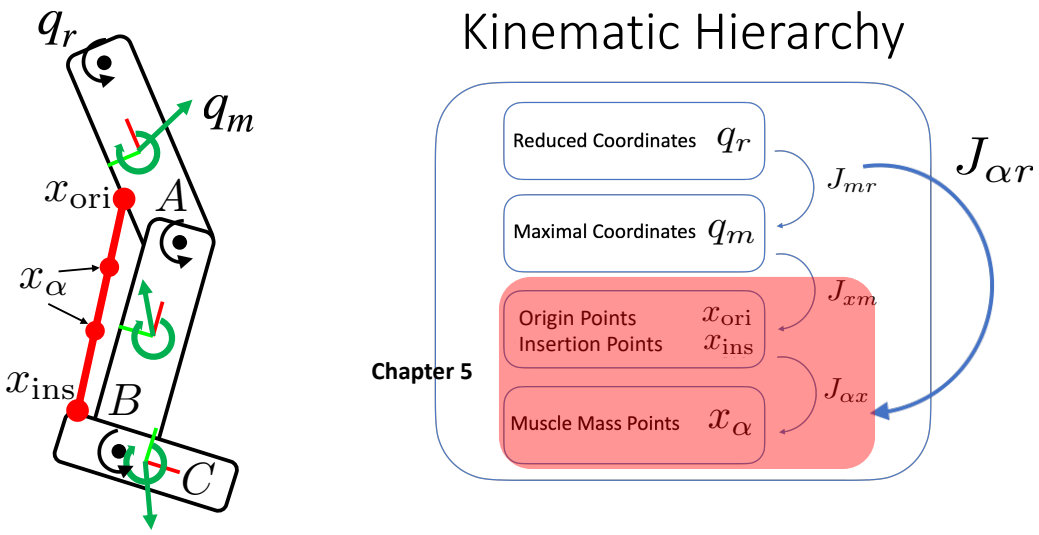


Figure 5.1: In this chapter, we will focus on the last component of kinematic hierarchy, which is to derive the mapping between the 3D coordinates of the muscle mass points and those of the origin/insertion points, represented by $\mathbf{J}_{\alpha x}$. By chaining the previously introduced Jacobian mappings and $\mathbf{J}_{\alpha x}$ together, we can express the motion of the musculotendons in terms of the motion of the skeletal joint and drive the simulation using only the reduced coordinates.

In this chapter, we will introduce a new approach to incorporate the effects of muscle inertia into efficient musculoskeletal simulations (highlighted in red in Fig. 5.1). This new approach extends on the Jacobian mapping $\mathbf{J}_{mr}$ developed in REDMAX, and is able to handle a wide variety of musculotendon paths, including straight, polyline, and curved paths over wrapping surfaces using our new Jacobians $\mathbf{J}_{\alpha m}$. We will also demonstrate how a novel neural network solution can be employed to solve the difficult challenge of Jacobian discontinuity, which arises in certain types of

*Part of this chapter is reprinted with permission from Y. Wang, J. Verheul, S.-H. Yeo, N. K. Kalantari, and S. Sueda, "Differentiable simulation of inertial musculotendons," *ACM Transactions on Graphics*, vol. 41, Nov. 2022.

muscle path.

5.1 Types of musculotendon paths

To aid us in the derivation, we categorize musculotendon paths into three types (Fig. 1.5): (Type I) straight-line paths; (Type II) polyline paths through a sequence of points; and (Type III) all others, but most importantly, curved paths wrapping over smooth surfaces. For Type I, we derive the Jacobian in a straight-forward manner. For Type II, we extend the Eulerian-on-Lagrangian framework [15, 16] to model the sliding motion of the musculotendon material through a series of points. For Type III, we use neural networks.

5.2 Jacobian discontinuity

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 [17, 19, 44, 20]. 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. 1.6a. 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. 1.6b. 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. 1.6d 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. 1.6e

shows the corresponding derivatives. The jump in the value of the Jacobian creates sudden changes in the 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. 1.6e, while keeping the position plots in Fig. 1.6d virtually indistinguishable from the output of the library code. This results in a smooth energy trajectory shown in Fig. 1.6c.

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 [73], as well as into differentiable simulation techniques, such as the adjoint method [74, 75, 76], 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.

5.3 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. 5.2b); 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:

$$\dot{\mathbf{x}}_\alpha = \mathbf{J}_{\alpha r}\dot{\mathbf{q}}_r, \tag{5.1}$$

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 [28, 77]:

$$\dot{\mathbf{q}}_m = \mathbf{J}_{mr}\dot{\mathbf{q}}_r, \tag{5.2}$$

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, ν, together to form the maximal velocity, so that for each body, we have:

$$\dot{\mathbf{q}}_m = \phi = \begin{pmatrix} \omega \\ \nu \end{pmatrix}, \tag{5.3}$$

with both ω and ν expressed in body-local coordinates [78].* 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 §5.3.1, §5.3.2, and §5.3.3). 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

*Other conventions can be used; the derivations will need to be accordingly modified.

the final Jacobian we are after:

$$\mathbf{J}_{\alpha r} = \mathbf{J}_{\alpha m}\mathbf{J}_{mr}. \tag{5.4}$$

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

$$\begin{aligned} \ddot{\mathbf{x}}_\alpha &= \dot{\mathbf{J}}_{\alpha r}\dot{\mathbf{q}}_r + \mathbf{J}_{\alpha r}\ddot{\mathbf{q}}_r \\ \dot{\mathbf{J}}_{\alpha r} &= \dot{\mathbf{J}}_{\alpha m}\mathbf{J}_{mr} + \mathbf{J}_{\alpha m}\dot{\mathbf{J}}_{mr}. \end{aligned} \tag{5.5}$$

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:

$$\mathbf{J}_{\alpha r}^\top \mathbf{M}_\alpha \mathbf{J}_{\alpha r}\ddot{\mathbf{q}}_r = \mathbf{J}_{\alpha r}^\top \left(\mathbf{f}_\alpha - \mathbf{M}_\alpha \dot{\mathbf{J}}_{\alpha r}\dot{\mathbf{q}}_r\right). \tag{5.6}$$

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. 5.6, which uses $\dot{\mathbf{J}}_{\alpha m}$, is the quadratic velocity vector (QVV) that results from the partial derivatives of the kinetic energy [68].

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:

$$\tilde{\mathbf{M}}_r\ddot{\mathbf{q}}_r = \tilde{\mathbf{f}}_r \tag{5.7a}$$

$$\tilde{\mathbf{M}}_r = \mathbf{J}_{\alpha r}^\top \mathbf{M}_\alpha \mathbf{J}_{\alpha r} + \mathbf{J}_{mr}^\top \mathbf{M}_m \mathbf{J}_{mr} \tag{5.7b}$$

$$\tilde{\mathbf{f}}_r = \mathbf{J}_{\alpha r}^\top \left(\mathbf{f}_\alpha - \mathbf{M}_\alpha \dot{\mathbf{J}}_{\alpha r}\dot{\mathbf{q}}_r\right) + \mathbf{J}_{mr}^\top \left(\mathbf{f}_m - \mathbf{M}_m \dot{\mathbf{J}}_{mr}\dot{\mathbf{q}}_r\right) + \mathbf{f}_r, \tag{5.7c}$$

where $\mathbf{M}_m \in \mathbb{R}^{6n \times 6n}$ is the constant diagonal inertia of the n bones,[†] $\mathbf{f}_m \in \mathbb{R}^{6n}$ 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 [73].

Throughout this section, we will use the concrete running example shown in Fig. 1.5. 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 $\dot{\mathbf{q}}_m = (\phi_A \ \phi_B \ \phi_C)^\top \in \mathbb{R}^{18}$, 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.

5.3.1 Type I: Straight Line Muscles

We start with the simple case of a straight line muscle between two bodies. We do not claim this subsection as a contribution, but the derivations and notations introduced here will help us with the rest of the thesis.

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:

$$ {}^{W}\dot{\mathbf{x}}_{\text{ori}} = {}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}})\,\phi_A, \quad {}^{W}\dot{\mathbf{x}}_{\text{ins}} = {}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}})\,\phi_C, \tag{5.8} $$

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 [78], with $[\cdot]$ the cross-product matrix. This gives us the following expression for the Jacobian between maximal velocities and

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

world velocities of the origin/insertion for our concrete running example in Fig. 1.5:

$$\mathbf{J}_{xm} = \begin{pmatrix} {}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & {}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{5.9}$$

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:

$$\mathbf{J}_{\alpha x} = \begin{pmatrix} (1-\alpha_1)\,\mathbf{I} & \alpha_1\,\mathbf{I} \\ (1-\alpha_2)\,\mathbf{I} & \alpha_2\,\mathbf{I} \end{pmatrix} \in \mathbb{R}^{6\times 6}, \tag{5.10}$$

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

$$\dot{\mathbf{J}}_{xm} = \begin{pmatrix} {}^{W}_{A}\mathbf{R}\,[\omega_A]\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & {}^{W}_{C}\mathbf{R}\,[\omega_C]\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \end{pmatrix}, \tag{5.11}$$

since $\dot{\mathbf{R}} = \mathbf{R}[\omega]$ for maximal velocities in body coordinates [78].

5.3.2 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 [15, 16]. 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. [16], 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 [16]):

$$\mathbf{J}_{sx} = -\mathbf{L}^{-1} \Delta\mathbf{S}\, \Delta\bar{\mathbf{X}}, \tag{5.12}$$

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. [16] 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:

$$\dot{\mathbf{J}}_{sx} = -\mathbf{L}^{-1} \left(\dot{\mathbf{L}}\, \mathbf{J}_{sx} + \Delta\dot{\mathbf{S}}\, \Delta\bar{\mathbf{X}} + \Delta\mathbf{S}\, \Delta\dot{\bar{\mathbf{X}}} \right). \tag{5.13}$$

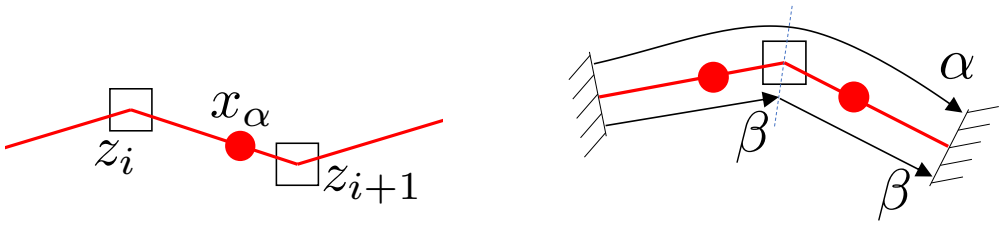
So far, the Jacobians $\mathbf{J}_{sx}$ and $\dot{\mathbf{J}}_{sx}$ that we derived cannot be plugged into our system because they only map between $\dot{\mathbf{x}}_i$ and $\dot{s}_i$, rather than from $\dot{\mathbf{q}}_m$ to $\dot{\mathbf{x}}_\alpha$. In other words, these Jacobians only provides the 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:

$$\mathbf{z}_i = \begin{pmatrix} \mathbf{x}_i \\ s_i \end{pmatrix} \in \mathbb{R}^4. \tag{5.14}$$

In the concrete example in Fig. 1.5, 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:

$$\begin{aligned}\mathbf{J}_{\alpha m} &= \mathbf{J}_{\alpha z}\mathbf{J}_{zm} \in \mathbb{R}^{6\times 18} \\ \dot{\mathbf{J}}_{\alpha m} &= \dot{\mathbf{J}}_{\alpha z}\mathbf{J}_{zm} + \mathbf{J}_{\alpha z}\dot{\mathbf{J}}_{zm}.\end{aligned} \tag{5.15}$$



(a) (b)

Figure 5.2: (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.

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. 5.2a). This was already derived by Sueda et al. [15] (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:

$$\mathbf{J}_{\alpha z} = \begin{pmatrix} (1-\beta_1)\mathbf{I} & -(1-\beta_1)\mathbf{F}_1 & \beta_1\mathbf{I} & -\beta_1\mathbf{F}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (1-\beta_2)\mathbf{I} & -(1-\beta_2)\mathbf{F}_2 & \beta_2\mathbf{I} & -\beta_2\mathbf{F}_2 \end{pmatrix}. \tag{5.16}$$

Here, we used β to represent the percentage location of $\mathbf{x}_\alpha$ *within* a particular line segment, as shown in Fig. 5.2b. $\mathbf{F} \in \mathbb{R}^3$ 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. [15], are nevertheless needed for our extensible musculotendons. We list the

detailed derivations of these derivatives as follows.

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

Here, we follow the derivation of EOL strands by Sachdeva et al. [16]. Let there be n path points between the origin and insertion, as shown in Fig. 5.2a. Then there are $n+1$ line segments, and within each segment i, we define

$$\Delta\mathbf{x}_i = \mathbf{x}_{i+1} - \mathbf{x}_i, \quad \Delta s_i = s_{i+1} - s_i. \tag{5.17}$$

The length of each segment is then

$$l_i = \|\Delta\mathbf{x}_i\|, \tag{5.18}$$

and we further define

$$\Delta\bar{\mathbf{x}}_i = \frac{\Delta\mathbf{x}_i}{l_i}. \tag{5.19}$$

Then we form the following three matrices:

$$\mathbf{L} = \begin{pmatrix} l_0 + l_1 & -l_0 & & & & \\ -l_2 & l_1 + l_2 & -l_1 & & & \\ & -l_3 & l_2 + l_3 & -l_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -l_{n-1} & l_{n-2} + l_{n-1} & -l_{n-2} \\ & & & & -l_n & l_{n-1} + l_n \end{pmatrix} \in \mathbb{R}^{n\times n} \tag{5.20}$$

$$\Delta\mathbf{S} = \begin{pmatrix} -\Delta s_1 & \Delta s_0 & & & \\ & -\Delta s_2 & \Delta s_1 & & \\ & & \ddots & \ddots & \\ & & & -\Delta s_n & \Delta s_{n-1} \end{pmatrix} \in \mathbb{R}^{n\times n+1} \tag{5.21}$$

$$\Delta\bar{\mathbf{X}} = \begin{pmatrix} -\Delta\bar{\mathbf{x}}_0^\top & \Delta\bar{\mathbf{x}}_0^\top & & & \\ & -\Delta\bar{\mathbf{x}}_1^\top & \Delta\bar{\mathbf{x}}_1^\top & & \\ & & \ddots & \ddots & \\ & & & -\Delta\bar{\mathbf{x}}_n & \Delta\bar{\mathbf{x}}_n \end{pmatrix} \in \mathbb{R}^{n+1\times 3(n+2)}. \tag{5.22}$$

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

$$\mathbf{J}_{sx} = -\mathbf{L}^{-1}\Delta\mathbf{S}\,\Delta\bar{\mathbf{X}} \in \mathbb{R}^{n\times 3(n+2)}, \tag{5.23}$$

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:

$$\dot{\mathbf{J}}_{sx} = -\mathbf{L}^{-1}\left(\dot{\mathbf{L}}\,\mathbf{J}_{sx} + \Delta\dot{\mathbf{S}}\,\Delta\bar{\mathbf{X}} + \Delta\mathbf{S}\,\Delta\dot{\bar{\mathbf{X}}}\right). \tag{5.24}$$

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

$$\Delta\dot{\mathbf{x}}_i = \dot{\mathbf{x}}_{i+1} - \dot{\mathbf{x}}_i \tag{5.25}$$

$$\Delta\dot{s}_i = \dot{s}_{i+1} - \dot{s}_i \tag{5.26}$$

$$\dot{l}_i = \frac{\Delta\mathbf{x}_i^\top}{l_i}\Delta\dot{\mathbf{x}}_i \tag{5.27}$$

$$\Delta\dot{\bar{\mathbf{x}}}_i = \left(\mathbf{I} - \Delta\bar{\mathbf{x}}_i\Delta\bar{\mathbf{x}}_i^\top\right)\frac{\Delta\dot{\mathbf{x}}_i}{l_i}. \tag{5.28}$$

5.3.2.2 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

$$\beta = \frac{s - s_0}{s_1 - s_0}, \tag{5.29}$$

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:

$$^W\mathbf{x}_\alpha = (1 - \beta)\, ^W\mathbf{x}_0 + \beta\, ^W\mathbf{x}_1. \tag{5.30}$$

The time derivative of β is

$$\dot{\beta} = -\frac{1}{\Delta s}\left((1 - \beta)\dot{s}_0 + \beta\dot{s}_1\right), \tag{5.31}$$

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

$$^W\dot{\mathbf{x}}_\alpha = (1 - \beta)\, ^W\dot{\mathbf{x}}_0 + \beta\, ^W\dot{\mathbf{x}}_1 - \frac{\Delta\mathbf{x}}{\Delta s}\left((1 - \beta)\dot{s}_0 + \beta\dot{s}_1\right), \tag{5.32}$$

where $\Delta\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_0$. This was derived earlier by Sueda et al.[15]. To ease the derivation of the time derivative, we define the deformation gradient of the line segment as:

$$\mathbf{F} = \frac{\Delta\mathbf{x}}{\Delta s}. \tag{5.33}$$

Rearranging Eq. 5.32, we obtain:

$$^W\dot{\mathbf{x}}_\alpha = \underbrace{\begin{pmatrix}(1-\beta)\mathbf{I} & \beta\mathbf{I} & -(1-\beta)\mathbf{F} & -\beta\mathbf{F}\end{pmatrix}}_{\mathbf{J}_{\alpha z}} \begin{pmatrix} ^W\dot{\mathbf{x}}_0 \\ ^W\dot{\mathbf{x}}_1 \\ \dot{s}_0 \\ \dot{s}_1 \end{pmatrix}. \tag{5.34}$$

The time derivative, $\dot{\mathbf{J}}_{\alpha z}$, requires $\dot{\beta}$ (Eq. 5.31) as well as $\dot{\mathbf{F}}$. The time derivative of the deformation gradient is

$$\dot{\mathbf{F}} = -\frac{1}{\Delta s}\left(\Delta\dot{\mathbf{x}} + \mathbf{F}\Delta\dot{s}\right). \tag{5.35}$$

The right Jacobian $\mathbf{J}_{zm} \in \mathbb{R}^{12\times 18}$ in Eq. 5.15 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 $\mathbf{J}_{sx}$:

$$\mathbf{J}_{zm} = \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{sx} \end{pmatrix} \mathbf{J}_{xm}, \quad \dot{\mathbf{J}}_{zm} = \begin{pmatrix} \mathbf{0} \\ \dot{\mathbf{J}}_{sx} \end{pmatrix} \mathbf{J}_{xm} + \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{sx} \end{pmatrix} \dot{\mathbf{J}}_{xm}. \tag{5.36}$$

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

$$\mathbf{J}_{xm} = \begin{pmatrix} {}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}\mathbf{x}_{i}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}\mathbf{x}_{i}) & {}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{5.37}$$

Its time derivative, $\dot{\mathbf{J}}_{xm}$, can be derived similarly as in Eq. 5.11.

5.3.3 Type III: Wrapping Surface Muscles

5.3.3.1 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:

$$\begin{pmatrix} {}^{S}\mathbf{x}_{\text{ori}} \\ {}^{S}\mathbf{x}_{\text{ins}} \\ \alpha \\ r \end{pmatrix} \rightarrow \begin{pmatrix} {}^{S}\mathbf{x}_{\alpha} \end{pmatrix}, \tag{5.38}$$

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 wrapping 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.

5.3.3.2 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 ${}^W\dot{\mathbf{x}}_\alpha$, the world velocity of the muscle mass point, 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:

$$ {}^W\dot{\mathbf{x}}_\alpha = {}^W\mathbf{v}_{\text{base}} + {}^W\Delta\mathbf{v}_{\text{ori}} + {}^W\Delta\mathbf{v}_{\text{ins}}. \tag{5.39} $$

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. 1.5). The first term in Eq. 5.39 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:

$$\begin{aligned} {}^{W}\mathbf{v}_{\text{base}} &= {}^{W}_{S}\mathbf{R}\,\Gamma({}^{S}\mathbf{x}_{\alpha})\,\phi_{S} \\ &= {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha})\,\phi_{B}, \end{aligned} \tag{5.40}$$

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. 1.5), is then

$$\mathbf{J}_{\text{base}} = \begin{pmatrix} \mathbf{0} & {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha_1}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha_2}) & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{6\times 18}, \tag{5.41}$$

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

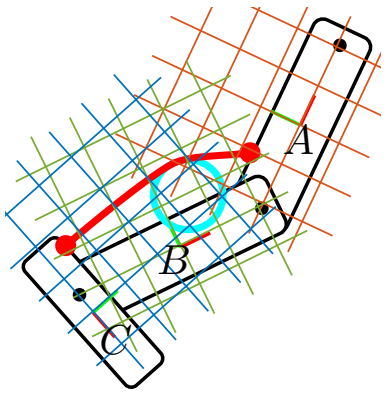


Figure 5.3: Coordinate spaces for a wrapping surface muscle. A contains the origin, C contains the insertion, and B contains the wrapping surface S. The S coordinate space (not drawn in this figure) moves rigidly with B.

To compute $\mathbf{J}_{\text{ori}}$, we first need the relative velocity of the origin from point of view of the surface. To do so, we must take into account the relative motions of the coordinate spaces, shown in Fig. 5.3. Since the origin is attached to A, we can compute its world velocity ${}^{W}\dot{\mathbf{x}}_{\text{ori}}$ using Eq. 5.8. 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:

$$ {}^{W}\mathbf{v}_{\text{ori}}^{\text{rel}} = {}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}})\,\phi_A - {}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}})\,\phi_B, \tag{5.42} $$

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:

$$ {}^{W}\Delta\mathbf{v}_{\text{ori}} = {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}\mathbf{v}_{\text{ori}}^{\text{rel}}. \tag{5.43} $$

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,

$$ {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}} = \frac{d\,{}^{S}\mathbf{x}_{\alpha}}{d\,{}^{S}\mathbf{x}_{\text{ori}}}, \quad {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{i}} = \frac{d\,{}^{S}\mathbf{x}_{\alpha}}{d\,{}^{S}\mathbf{x}_{\text{ins}}}. \tag{5.44} $$

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. 5.42 and Eq. 5.43 and extracting out the maximal velocities ϕ_A and ϕ_B, the Jacobian $\mathbf{J}_{\text{ori}}$ for the concrete running example becomes:

$$ \begin{aligned} \mathbf{J}_{\text{ori}} = &\begin{pmatrix} {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \\ {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{A}\mathbf{R}\,\Gamma({}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} & \mathbf{0} \end{pmatrix} \\ &- \begin{pmatrix} \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{o}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}}) & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{6\times 18}. \end{aligned} \tag{5.45} $$

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

$$\mathbf{J}_{\text{ins}} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \\ \mathbf{0} & \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{C}\mathbf{R}\,\Gamma({}^{C}\mathbf{x}_{\text{ins}}) \end{pmatrix} - \begin{pmatrix} \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_1\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{C}\mathbf{E}\,{}^{C}\mathbf{x}_{\text{ins}}) & \mathbf{0} \\ \mathbf{0} & {}^{W}_{S}\mathbf{R}\,{}^{S}\mathbf{J}^{\text{NN}}_{\alpha_2\text{i}}\,{}^{S}_{W}\mathbf{R}\,{}^{W}_{B}\mathbf{R}\,\Gamma({}^{B}_{C}\mathbf{E}\,{}^{C}\mathbf{x}_{\text{ins}}) & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{6\times 18}. \tag{5.46}$$

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 derived as follows.

5.3.3.3 *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 have [78]:

$$\dot{\mathbf{R}} = \mathbf{R}[\omega], \tag{5.47}$$

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,

$$\dot{\Gamma}({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha}) = \begin{pmatrix} [{}^{B}_{S}\mathbf{R}\,{}^{S}\dot{\mathbf{x}}_{\alpha}]^{\top} & \mathbf{0} \end{pmatrix}. \tag{5.48}$$

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:

$${}^{S}\dot{\mathbf{x}}_{\alpha} = {}^{S}_{B}\mathbf{R}\,\Gamma({}^{B}_{S}\mathbf{E}\,{}^{S}\mathbf{x}_{\alpha})\,\phi_B. \tag{5.49}$$

5.3.3.4 *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

$$ {}^{W}_{S}\mathbf{R} = {}^{W}_{B}\mathbf{R}\,{}^{B}_{S}\mathbf{R}. \tag{5.50} $$

The surface does not move with respect to the body, so using Eq. 5.47,

$$ {}^{W}_{S}\dot{\mathbf{R}} = {}^{W}_{B}\mathbf{R}[\omega_B]\,{}^{B}_{S}\mathbf{R}. \tag{5.51} $$

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

$$ {}^{S}_{W}\dot{\mathbf{R}} = {}^{W}_{S}\dot{\mathbf{R}}^{\top}. \tag{5.52} $$

Next, using Eq. 5.47 again,

$$ {}^{W}_{A}\dot{\mathbf{R}} = {}^{W}_{A}\mathbf{R}[\omega_A]. \tag{5.53} $$

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

$$ \dot{\Gamma}({}^{A}\mathbf{x}_{\text{ori}}) = 0. \tag{5.54} $$

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:

$$ {}^{B}_{A}\dot{\mathbf{E}} = \frac{d}{dt}\{{}^{W}_{B}\mathbf{E}^{-1}\}\,{}^{W}_{A}\mathbf{E} + {}^{W}_{B}\mathbf{E}^{-1}\frac{d}{dt}\{{}^{W}_{A}\mathbf{E}\}. \tag{5.55} $$

Using the inverse derivative identity and the fact that $\dot{\mathbf{E}} = \mathbf{E}[\phi]$ [78], we have, after some rearranging:

$$ {}^{B}_{A}\dot{\mathbf{E}} = {}^{B}_{A}\mathbf{E}[\phi_A] - [\phi_B]\,{}^{B}_{A}\mathbf{E}. \tag{5.56} $$

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

$$ \dot{\Gamma}({}^{B}_{A}\mathbf{E}\,{}^{A}\mathbf{x}_{\text{ori}}) = \begin{pmatrix} \left[({}^{B}_{A}\mathbf{E}[\phi_A] - [\phi_B]\,{}^{B}_{A}\mathbf{E})\,{}^{A}\mathbf{x}_{\text{ori}}\right]^{\top} & \mathbf{0} \end{pmatrix}. \tag{5.57} $$

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:

$$\begin{aligned} {}^{S}\mathbf{x}^{+}_{\text{ori}} &= {}^{S}\mathbf{x}_{\text{ori}} + \epsilon\, {}^{S}\mathbf{v}^{\text{rel}}_{\text{ori}}, \quad {}^{S}\dot{\mathbf{J}}^{\text{NN}}_{\alpha\text{o}} = ({}^{S}\mathbf{J}^{\text{NN+}}_{\alpha\text{o}} - {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{o}})/\epsilon, \\ {}^{S}\mathbf{x}^{+}_{\text{ins}} &= {}^{S}\mathbf{x}_{\text{ins}} + \epsilon\, {}^{S}\mathbf{v}^{\text{rel}}_{\text{ins}}, \quad {}^{S}\dot{\mathbf{J}}^{\text{NN}}_{\alpha\text{i}} = ({}^{S}\mathbf{J}^{\text{NN+}}_{\alpha\text{i}} - {}^{S}\mathbf{J}^{\text{NN}}_{\alpha\text{i}})/\epsilon, \end{aligned} \tag{5.58}$$

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 ${}^{S}\mathbf{v}^{\text{rel}}_{\text{ins}}$.

6. RESULTS*

The results described in the first part of this chapter was presented in SIGGRAPH 2019. And the second part of this chapter will be presented in SIGGRAPH ASIA 2022.

6.1 Flexibility of REDMAX

This section demonstrate the results from Chapter 4. With REDMAX's flexibility, we can easily incorporate constraints, attach FEM, and use advanced joints, etc. First, we can easily incorporate different combinations of constraints, for instance, reduced unilateral constraint for joint limits, maximal unilateral constraint for collisions, reduced bilateral constraint for gears, and finally maximal bilateral constraint for loop closure. It also becomes trivial to handle implicit two-way coupling between articulated and deformable bodies, such as FEM when attaching a finite element mesh to the skeleton, we can directly form and solve the KKT linear system, which is composed of the skeleton inside the body, and then the deformable bodies, finally the bilateral constraints that bind some vertices to be fixed with respect to the skeleton (see Fig. 6.1). With REDMAX, it is easy to combine forward and inverse dynamics into hybrid dynamics. In forward dynamics, we compute the motion $\dot{q}$ given the forces f, and in inverse dynamics, we compute the forces f given the motion $\dot{q}$. In REDMAX, it's also easy to combine them into hybrid dynamics, where some DOFs are forward, some DOFs are inverse. We emphasize that we can easily apply hybrid dynamics in both reduced and maximal coordinates.

6.1.1 Experiment Settings

We implemented our system in C++ and ran the simulations on a consumer desktop with an Intel Core i7-7700 CPU @ 3.6 Ghz and 16 GB of RAM. We use Eigen for dense linear algebra,

*Part of this chapter is reprinted with permission from Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, "REDMAX: Efficient & flexible approach for articulated dynamics," *ACM Trans. Graph.*, vol. 38, Jul. 2019, and from Y. Wang, J. Verheul, S.-H. Yeo, N. K. Kalantari, and S. Sueda, "Differentiable simulation of inertial musculotendons," *ACM Transactions on Graphics*, vol. 41, Nov. 2022.

Pardiso for sparse linear solves, and Mosek for quadratic programs.

6.1.2 Adding Deformable Bodies

We use a starfish example to showcase how simple and straightforward it is to add deformable bodies to our simulation system using REDMAX framework and achieve two-way coupling effects.

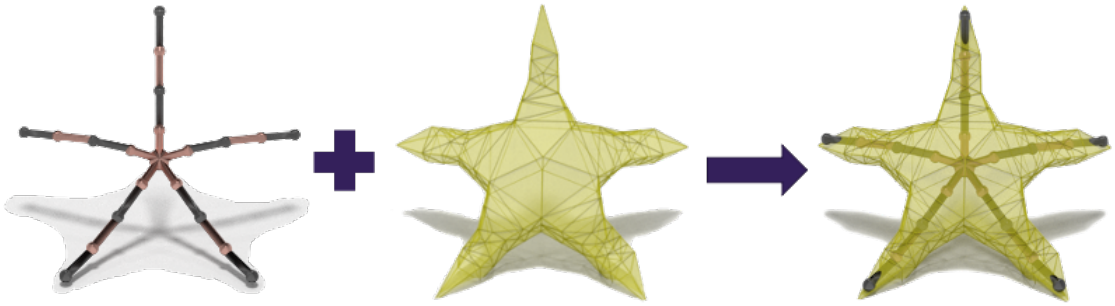


Figure 6.1: We model a starfish with a skeleton consisting of 20 joints and a coarse FEM mesh consisting of 221 vertices. For display and collision, we embed a fine mesh with 7909 vertices inside the coarse simulation mesh. We use co-rotated elasticity, but any material model can be used [79].

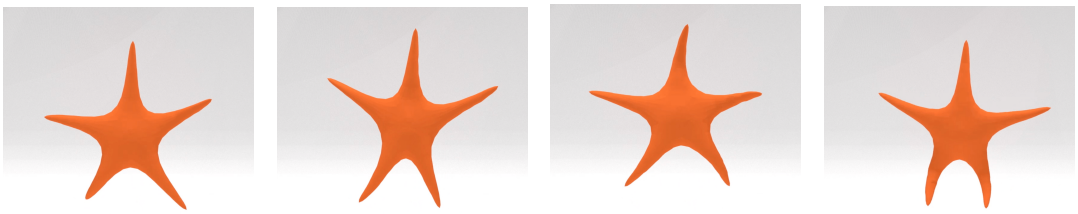


Figure 6.2: STARFISH, showing fully two-way coupled integration between articulated and deformable bodies.

STARFISH (Fig. 6.2): We use REDMAX hybrid dynamics to animate the starfish—we procedurally prescribe some of the joints as well as some specific points on the skeleton. The rest of the

skeleton and the FEM mesh are passively simulated with fully implicit two-way coupling.

6.1.3 Hybrid Dynamics

We use a hand simulation example to showcase the hybrid dynamics (forward dynamics and inverse dynamics) using the REDMAX framework.

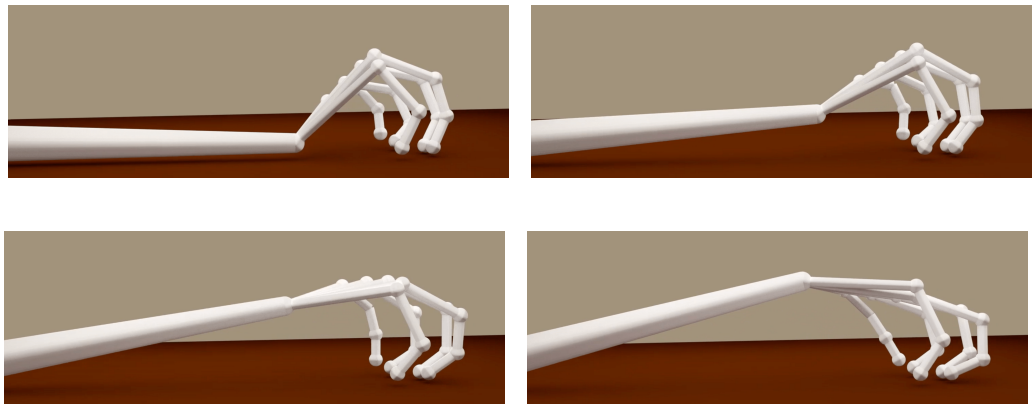


Figure 6.3: HAND simulation with coupled interphalangeal joint motions. We fixed the fingertips, and at the same time, prescribed the elbow angle using reduced inverse dynamics.

HAND (Fig. 6.3): The two distal joints of a healthy human finger exhibits coupled interphalangeal joint motions [80]. Specifically, the most distal joint (DIP) usually flexes by half the joint angle of the second most distal joint (PIP). We model a human hand where we use hyper reduced coordinates on the DIP/PIP of the four fingers. We then animate the hand with REDMAX hybrid dynamics. The fingertip positions are prescribed using maximal inverse dynamics, and at the same time, the elbow angle is prescribed using reduced inverse dynamics (Fig. 6.3).

6.2 Muscle Inertia Simulations

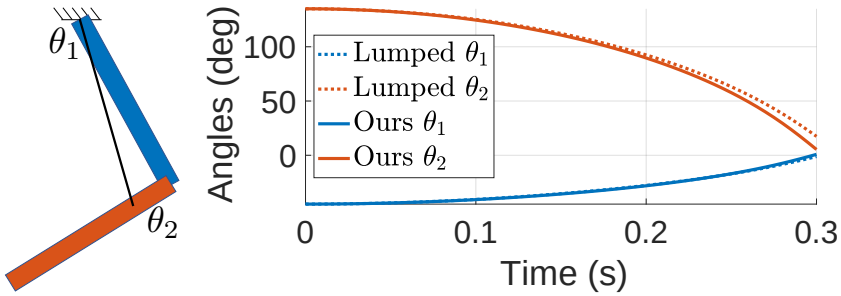
The next section, we will show some results of our reduced inertia muscle from Chapter 5. We first verify that our framework is in agreement with published results. And we demonstrate how our neural network approach can handle all the variations of the wrapping surfaces properly. Finally,

we show that our framework can handle musculotendons with higher-order integrators, inverse dynamics, and differentiability.

6.2.1 Experiment Settings

We implemented a prototype in MATLAB [81]. 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 [82] with the default parameters and a learning rate of 10^{-4}. For each network, we used 6 hidden layers with 256 neurons per layer. We use $\tanh$ 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.

6.2.2 Comparison to Analytical Results



(a) (b)

Figure 6.4: Comparison to published results [12]. (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.

We start with comparisons to the simulation and analytical results by Pai [12] to verify that our general framework is in agreement with published results. First we simulate the scene in Fig. 6.4a, which uses the same setup as *their* Fig. 2. As shown by the solid lines in *our* Fig. 6.4b, the two angles reach zero at 0.3 s. Our results matched the analytical results by Pai [12].

Pai [12] 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. 5.7b, 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.

6.2.3 Network Jacobian

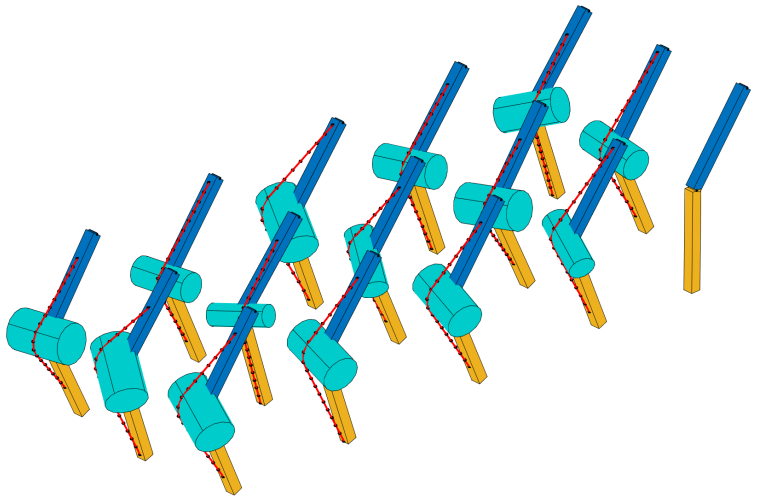
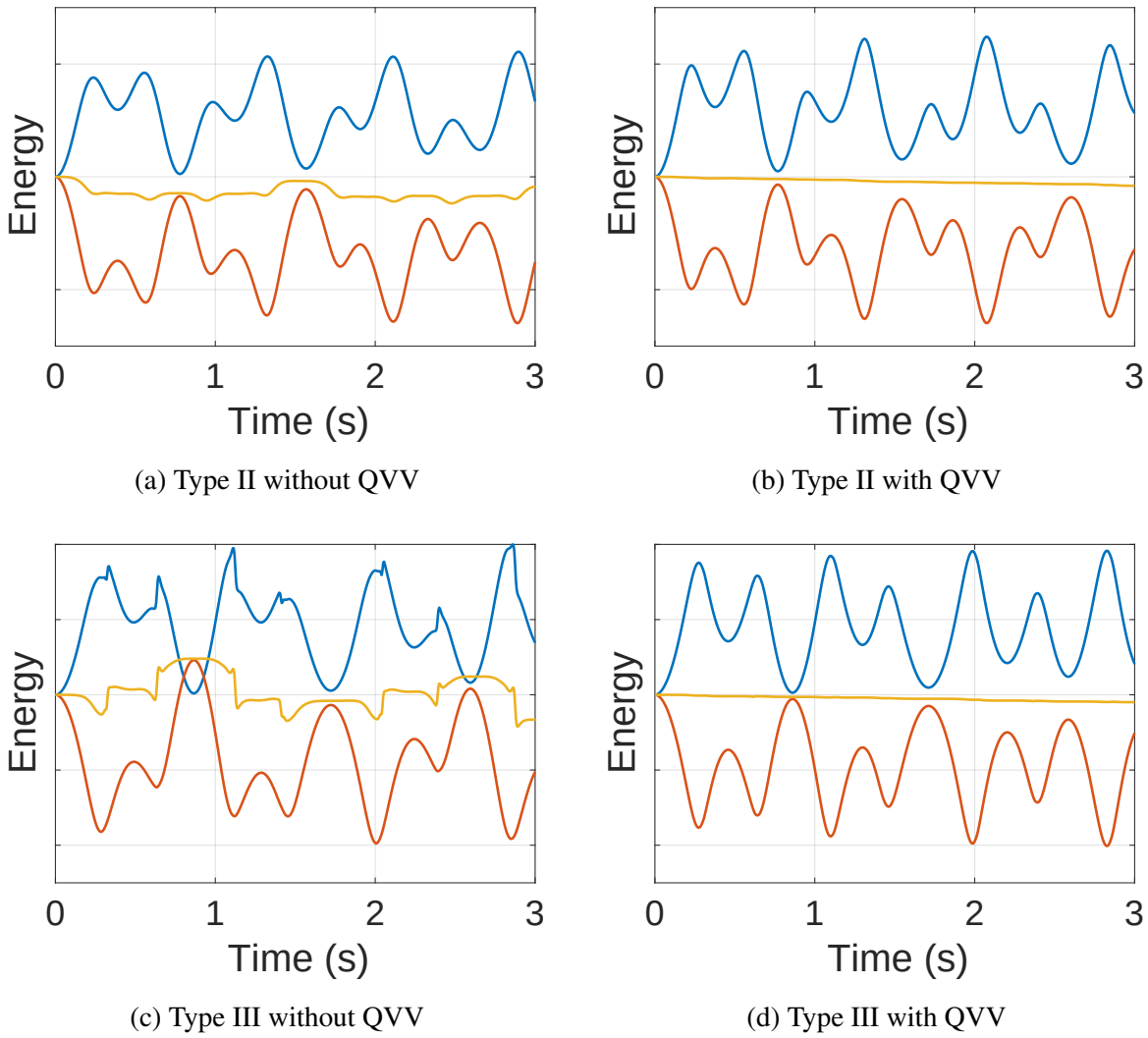


Figure 6.5: Double pendulums with cylinder wrapping. The same trained network is used for a range of input parameters. For comparison, the right-most double pendulum is simulated without a muscle.

We simulate a group of double pendulums with varying origin, insertion, radius, and the initial rotation of the wrapping surface, as shown in Fig. 6.5. 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.

6.2.4 Energy Behavior



(a) Type II without QVV

(b) Type II with QVV

(c) Type III without QVV

(d) Type III with QVV

Figure 6.6: (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.

To show the importance of the $\dot{\mathbf{J}}_{\alpha m}$ term that we derived, we take one of the simulations from Fig. 6.5, and remove $\dot{\mathbf{J}}_{\alpha m}$, and consequently the quadratic velocity vector (QVV) of the muscle mass points [68]. (We keep the QVV of the bones in the simulation.) As shown in Fig. 6.6c, even

with the SDIRK2 time integrator, the energy oscillates wildly. On the other hand, as shown in Fig. 6.6d, the energy stays stable once we put the QVV of the muscle back in.

Similarly, in Fig. 6.6a–6.6b, 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.

6.2.5 Simulation Stability

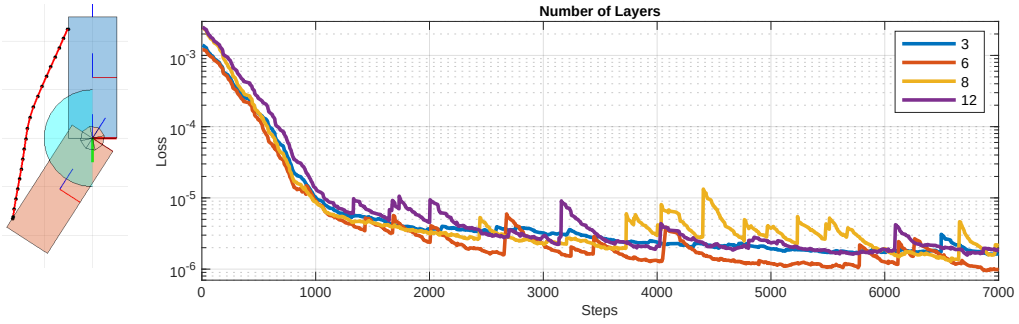
The effect of muscle inertia is stronger when a relatively light bone is actuated by a relatively large muscle mass located away from the joint. In Fig. 1.1b, we show an example of such a case with the flexor digitorum profundus and superficialias muscles (FDP & FDS), which originate near the elbow and insert into the distal and middle phalanges, respectively. For our simulation, we modeled the bones and joints using open source data [83], 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 $\mathrm{g\,cm^{-3}}$ 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 [84].) 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.

6.2.6 Sampling & Network Architectures

This section provides additional experiments with different sampling thresholds and network architectures for the neural network for Type III muscles. Specifically, we closely examine the behavior of the network around the Jacobian discontinuity.

To train networks that behave smoothly near the Jacobian discontinuity and to encourage better convergence, we exclude a small percentage of data points around the sharpest features from our sampling range. This sampling threshold needs to be chosen carefully. If we throw away too many samples, the resulting simulation produces noticeable artifacts, such as the one shown in Fig. 6.7a, where the threshold was set to 20% (l/L in §5.3.3.1). As we lower the threshold to 10%, the artifacts become minor but remain visible. We have empirically determined that a threshold of 1% can retain the network's convergence while keeping the artifacts negligible.



(a) (b)

Figure 6.7: (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. 6.7b, 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.

In order to better understand what the network is doing around the Jacobian discontinuity, we first set up a Type III muscle around an event that can cause Jacobian discontinuity in Fig. 6.8. We

fixed one end of the muscle, $\mathbf{x}_{\text{ori}}$, and moved the other end of the muscle, $\mathbf{x}_{\text{ins}}$, along the x-axis. As the position of $\mathbf{x}_{\text{ins}}$ moves from right to left, the muscle loses contact with the wrapping surface and becomes straight.

In the following experiments, we closely examine the behavior of various neural networks around the point where the muscle loses contact with the wrapping surface. For each network architecture, we plotted the x component of the position of the muscle mass points in the middle, $\mathbf{x}_\alpha$ (where $\alpha = 0.5$), as well as the x component of the Jacobian, $\mathbf{J}_{\alpha x_{ins}}$, against the x component of the position of the insertion point $\mathbf{x}_{\text{ins}}$. Those networks with the same sampling threshold are gathered in the same plot. We also include the results from the existing library in the plot for comparison purposes. The plots showed us one dimension of the exact Jacobians, which the networks learned from the samples. Thus we can get a better idea about how the networks are approximating around the sharp features and use these plots as a reference when we choose the right network to use in the simulation.

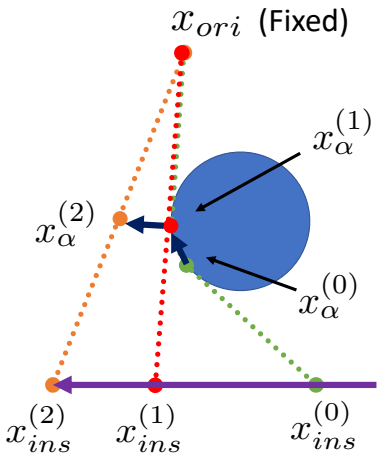
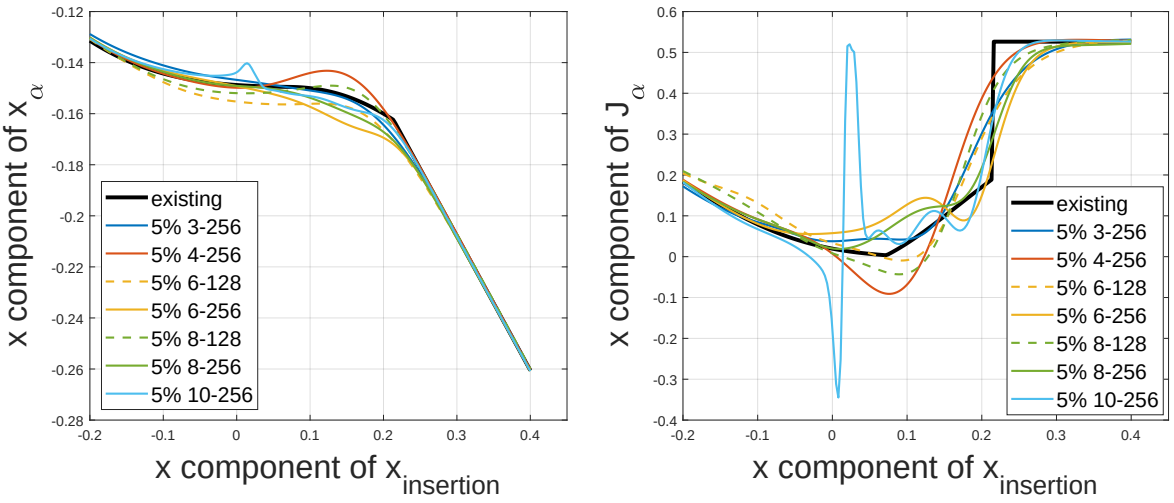


Figure 6.8: Here is the scene where we conducted experiments with various network structures and sampling thresholds. The origin point $\mathbf{x}_{\text{ori}}$ and the wrapping surface are fixed. We move the insertion point $\mathbf{x}_{\text{ins}}$ from right to left along the x-axis, which is represented by the purple arrow. The muscle point ($\mathbf{x}_\alpha$, where $\alpha = 0.5$) is moving with the insertion point, and its status is changing from moving on the surface to barely touching the surface and finally detaching from the surface. The dotted lines in green, red, and orange color represent three different stages of the muscle. We can learn how networks behave around the sharp feature by plotting the change of $\mathbf{x}_\alpha$ against the change of $\mathbf{x}_{\text{ins}}$ in the x-axis.

After setting up the scene, we ran experiments to determine the most suitable network architecture and the sampling threshold. Since our problem is a straightforward regression problem, we limit our neural network to multi-layer perceptrons. We then trained the networks with layers 1, 2, 3, 4, 6, 8, 10, and 12. We also tested 128 and 256 neurons per layer for each network depth. We used different training data for each network structure with thresholds 1%, 3%, 5%, and 10%. Here, we only used 1% and 5% to make the comparisons.

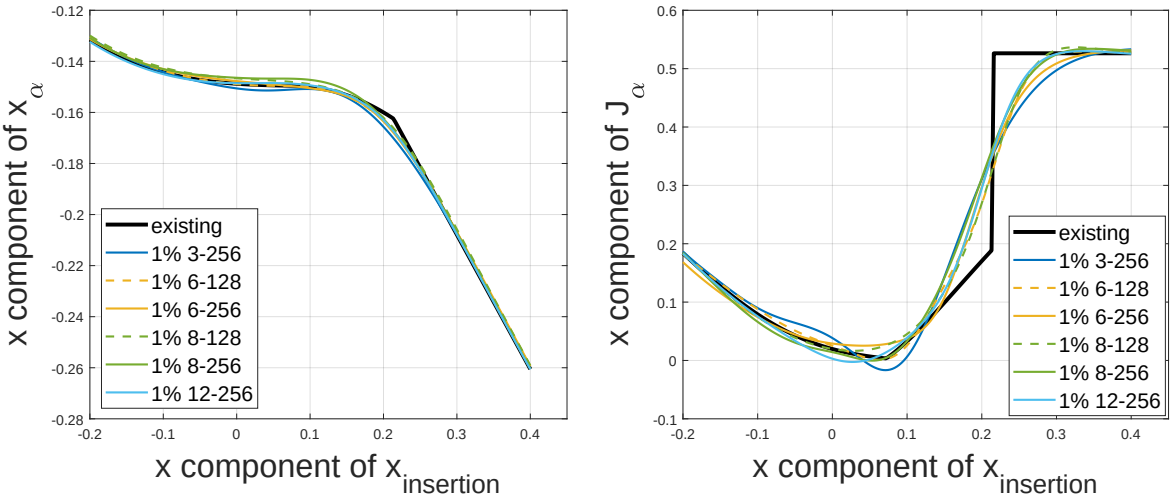


(a) (b)

Figure 6.9: Threshold 5% Comparison. (a) Plot of the x-component of the muscle point as a function of the x-component of the insertion point. (b) Plot of the x-component of Jacobian of the muscle point as a function of the x-component of the insertion point. Networks with the same number of layers are plotted in the same color. Solid lines represent the networks with 256 neurons in each layer, and dotted lines represent the networks with 128 neurons. The black line results from the existing library and the other neural networks' results should closely approximate it.

As Fig. 6.9a shows, none of the networks approximates the black line sufficiently well. All of them produced visible artifacts during simulation. Starting from the model with six layers and 256 neurons, it can be seen from Fig. 6.9b that the networks beyond this size started to overfit, which is indicated by the fluctuations in the solid yellow line. The extent to which the models overfit becomes more severe as the size of the networks becomes larger than six. The model with eight

layers and 256 neurons (shown in solid green line) and the model with ten layers and 256 neurons (shown in solid blue line) display more significant fluctuations and generate unexpected movements in the energy plots.



(a) (b)

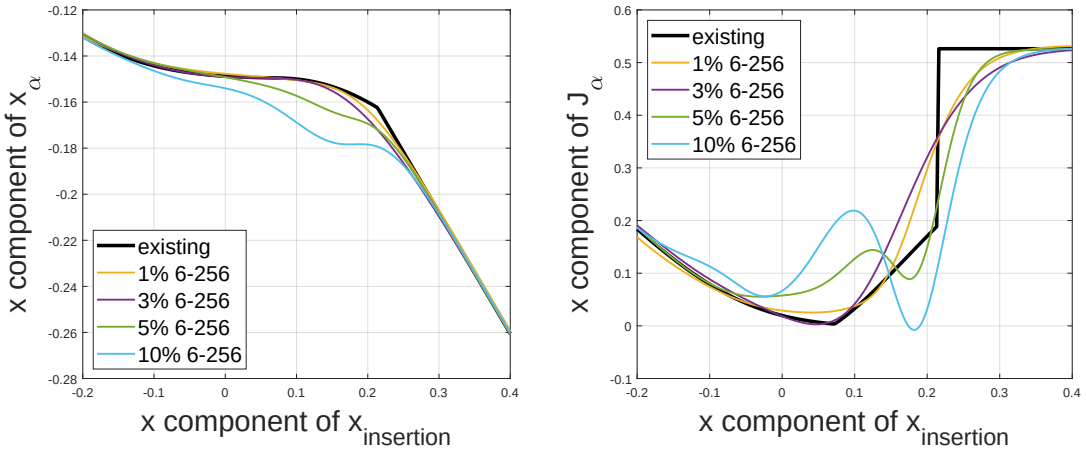
Figure 6.10: Threshold 1% Comparison. (a) Plot of the x-component of the middle muscle point as a function of the x component of the insertion point. The 6-256 and 12-256 models produced good approximations of the black line and resulted in visually good simulations. Other models were not good enough due to the presence of artifacts. (b) The same plot of the Jacobian. Compared to the plot with threshold 5%, the Jacobian fluctuation disappeared because more data points exist around the discontinuity and a larger model no longer overfits.

As Fig. 6.10a shows, unlike the plots with threshold 5%, most networks can approximate the black line well in terms of position accuracy. As a result, we prefer networks with a threshold of 1% over 5%. We further compared both Fig. 6.10a and Fig. 6.10b and observed that all networks between the 6-layer-256-neuron and 12-layer-256-neuron models gave good approximations. We chose the 6-layer-256-neuron model for its simplicity.

Finally, we want to compare the influence of different sampling thresholds. We trained the network consisting of 6 layers and 256 neurons per layer, using different training datasets obtained by applying different sampling thresholds. We chose the 6-layer model because prior experiments

showed that it generally works well with different thresholds.

The results of different sampling thresholds can be seen from Fig. 6.11a; each line corresponds to the behavior of a separately trained model. When trained with a 1% threshold, the model best approximates the black line. Moreover, as the sampling threshold gets larger than 1%, the lines corresponding to the trained models deviate further from the existing library.



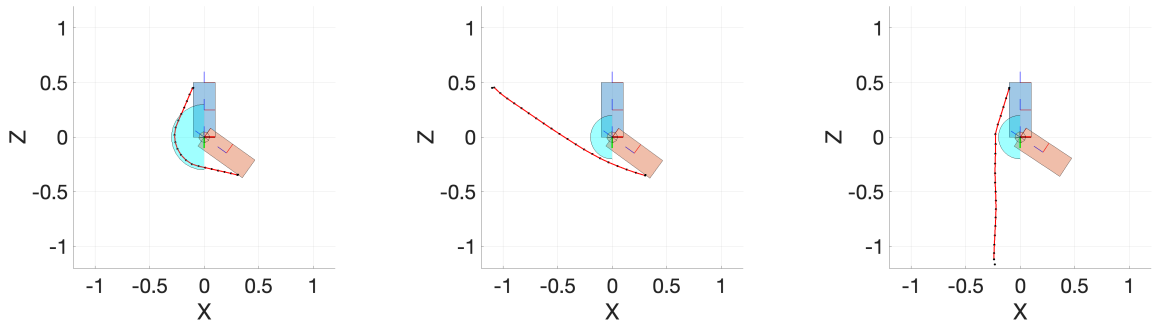
(a) (b)

Figure 6.11: Here we compared different sampling thresholds with the same network structure. (a) Plot of the x-component of the middle muscle point as a function of the x component of the insertion point. (b) The same plot of the Jacobian.

6.2.7 Failure Cases

If we use values outside the training range, the network is not able to generate good results. Fig. 6.12 shows three such examples. In Fig. 6.12a, the runtime radius is outside of the training range. The resulting muscle path visibly penetrates the wrapping surface. In Fig. 6.12b, the origin point is outside of the training range. The resulting muscle path is visibly curved. In Fig. 6.12c, the insertion point is outside of the training range. The resulting muscle path is visibly curved and irregular.

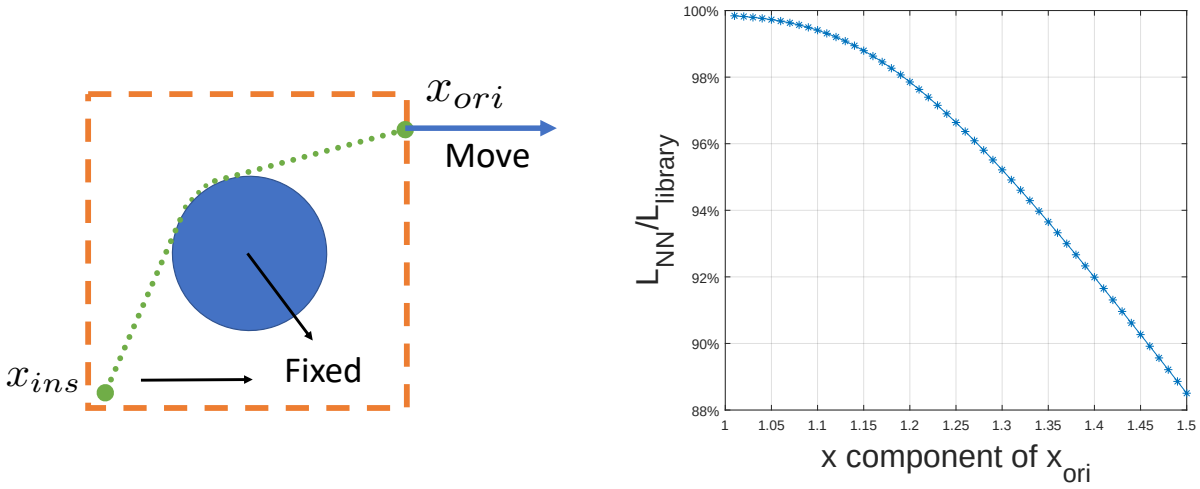
To further analyze the failure cases, we computed the total length of the muscle line by both the



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

Figure 6.12: 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.

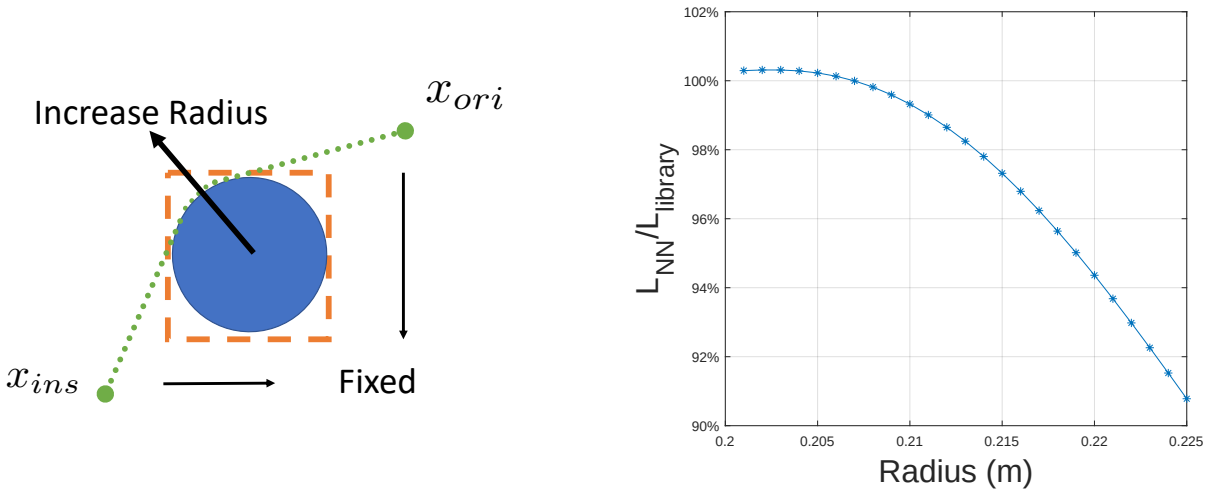
network and the existing library. In Fig. 6.13a, we set up a scene with a fixed radius and a fixed insertion point and move the origin point away from the training range. The result showed that as the origin moves further from the training range, the resulting muscle line will be more different from the existing library, leading to much worse visual artifacts. In Fig. 6.14a, we set up another scene with a fixed origin point and a fixed insertion point, increasing the radius and making it out of the training range. The result showed that as the radius gets more extensive than the training range, the resulting muscle line will be more different from the existing library.



(a) Scene

(b) Origin

Figure 6.13: (a) The scene with a fixed radius value of 0.15 and a fixed insertion $(-0.5, 0.0, 0.0)$ in the coordinate space of the wrapping surface. The initial position of the origin is $(1.0, 0.0, 0.0)$ in the coordinate space of the wrapping surface. We move the origin point along its x-direction as the arrow points. We computed the total length L of the muscle line using the network and the existing library. The orange dotted box represents the network training range: origin between $(-1.0, -1.0, -1.0)$ and $(1.0, 1.0, 1.0)$ in the coordinate space of the wrapping surface. (b) We plot the percentage difference between the resulting length of the muscle line from the network and from the existing library as we move the origin further away from the training range. Note that the difference increases as the origin moves further away from the training range.



(a) Scene (b) Radius

Figure 6.14: (a) The scene with a fixed origin of $(1.0, 0.0, 0.0)$ and a fixed insertion $(-0.5, 0.0, 0.0)$ in the coordinate space of the wrapping surface. The initial radius of the cylinder is 0.2. Then, we increase the radius and make it out of the training range (represented by the dotted orange box). Similarly, we computed the total length L of the muscle line using the network and the existing library. The network training range of the radius is between 0.01 and 0.2. (b) We plot the percentage difference between the resulting length of the muscle line from the network and from the existing library as we increase the radius and make it further away from the training range. Note that the difference increases as the radius moves further away from the training range.

6.2.8 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 s clip in Fig. 6.15. 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 subject runs on a treadmill at 19.1 $\mathrm{km/h}$, 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 s trials using OpenSim and our simulator. We overlay the swing phases on top of each other and plot the results in Fig. 6.16. 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 [85]. 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. 6.16b shows the closeup of the final dip. Comparing the blue and red plots, we confirm that our simulator generates results that *gracefully*

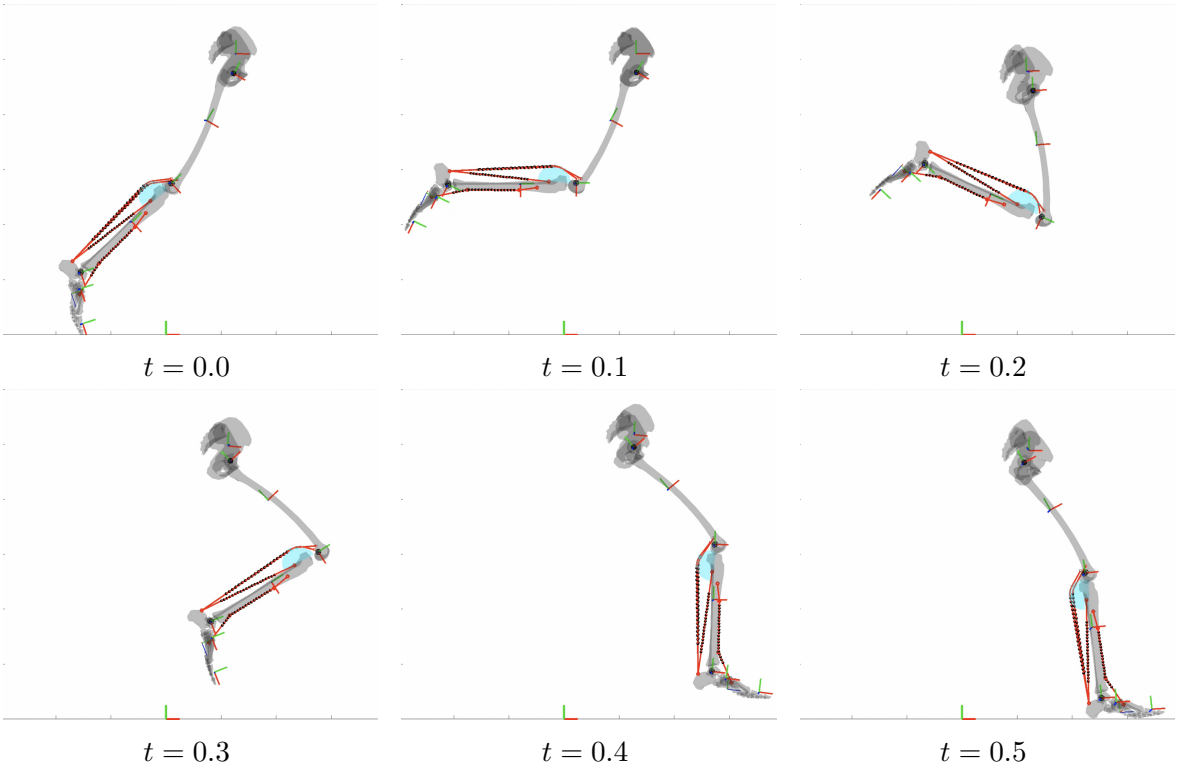


Figure 6.15: The swing phase of a 19.1 km/h treadmill run, showing only the right leg. The four muscles (and their types) are: gastrocnemius lateral (Type III), gastrocnemius medial (Type III), soleus (Type I), and tibialis anterior (Type II).

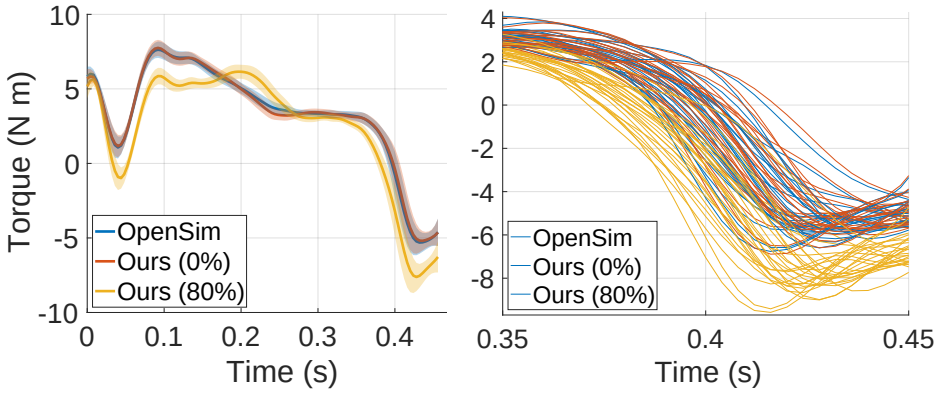


Figure 6.16: (Left) Ankle torque computed by inverse dynamics, showing the mean and the standard deviation. Blue plot is generated by OpenSim, which does not support inertial muscles. Red plot is generated by our simulator with the muscles accounting for 0% of the total mass. Yellow plot is generated 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 degrades to OpenSim's results as the inertia of the muscles is decreased to zero.

degrades to OpenSim's results, as the inertia of the muscles is 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.

6.2.9 Graceful Degradation

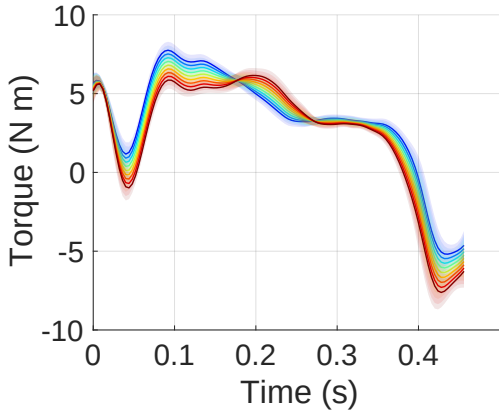


Figure 6.17: The colors show how much of the segment mass has been put into the muscles: from dark blue (0%) to dark red (80%).

In §6.2.8, we showed that the inverse dynamics output computed by our simulator matches the output computed with OpenSim [18], 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 Fig. 6.17, 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.

6.2.10 Spline Joint Knee with Hill-Type Muscles

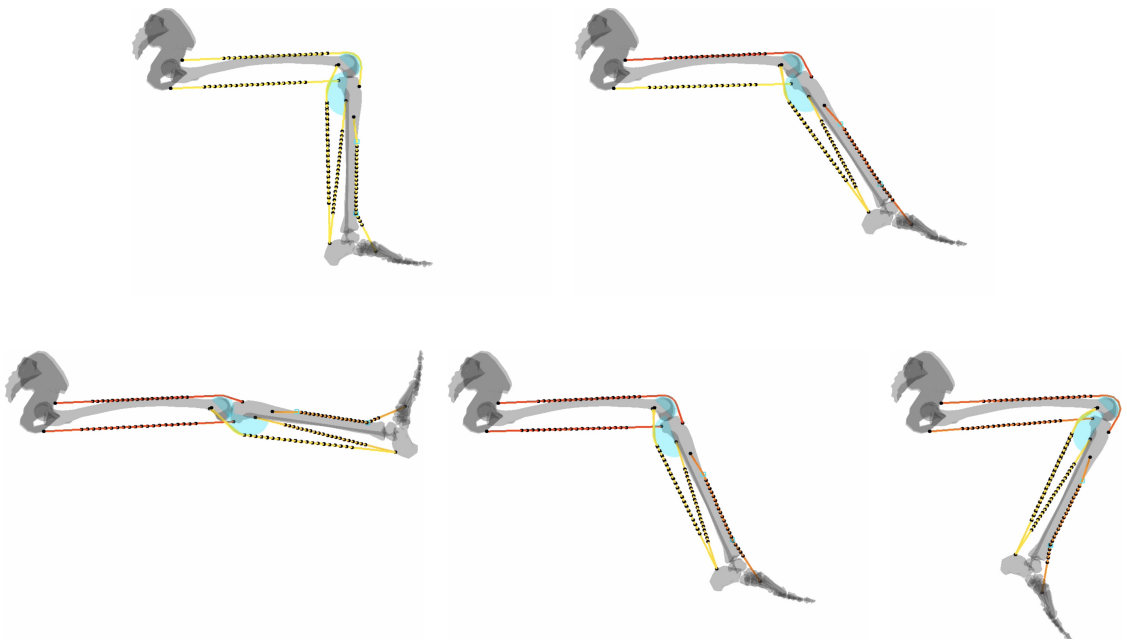


Figure 6.18: For §6.2.10, we add a spline joint knee and Hill-type muscles to the model used in §6.2.8. We manually excite the rectus femoris and the semimembranosus muscles. The excitation levels of the soleus and the tibialis anterior muscles are computed automatically with a proportional controller.

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 [71] and add the semimembranosus (Type I) and the rectus femoris (Type III). As shown in Fig. 6.19, we manually model a spline

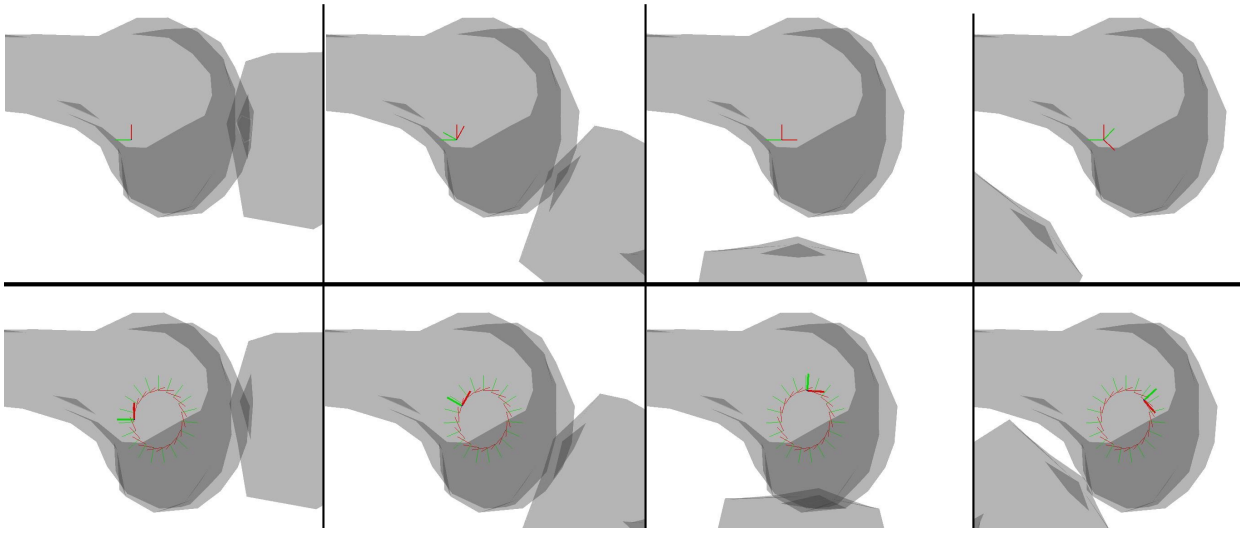


Figure 6.19: Our approach supports complex joint types. Top row: Knee with a revolute joint—the tibia separates from the femur. Bottom row: Knee with a spline joint—the tibia stays close to the femur.

joint to better model the motion of the tibia with respect to the femur. (OpenSim uses a similar technique called a "mobilizer" [86].) We also use Hill-type muscles [87] to drive the knee and ankle joints, as opposed to using mocap as in §6.2.8. 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 [88]. 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. 6.18.

6.2.11 Differentiable Reaching with Adjoint Method

For the final result, we use the adjoint method [74, 75, 76] to compute the simulation derivatives to optimize for a reaching task using an arm model [89] with manually placed muscles, shown in Fig. 6.20. 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 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.

6.3 Summary

In this chapter, we showcased some applications of the REDMAX framework to simulate articulated rigid body. We further showcased various experiments which compared some simulation results generated by our framework and analytical results by the published results, and verified that our general framework is in agreement with some previous works. Finally, we have shown that our framework is compatible with different existing techniques and applications, including higher-order integrators, inverse dynamics, Hill-type muscle models, and differentiability.

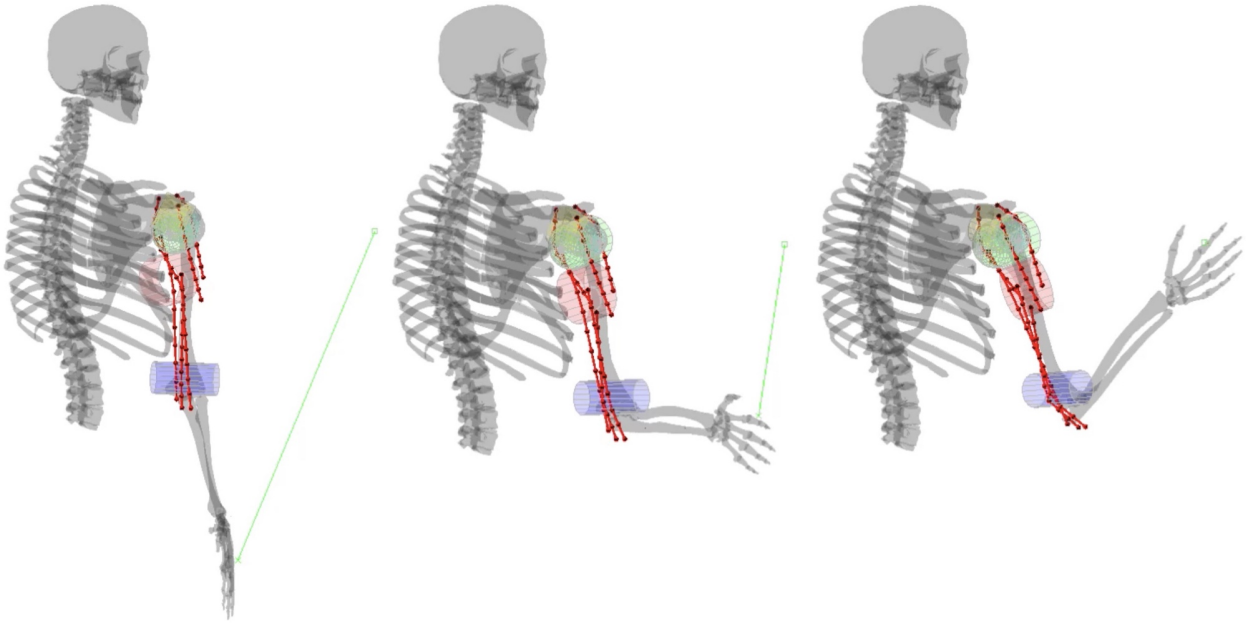


Figure 6.20: Reaching task. Our inertial muscles work flawlessly with the adjoint method for computing the simulation derivatives.

7. CONCLUSIONS AND FUTURE WORK*

7.1 Summary

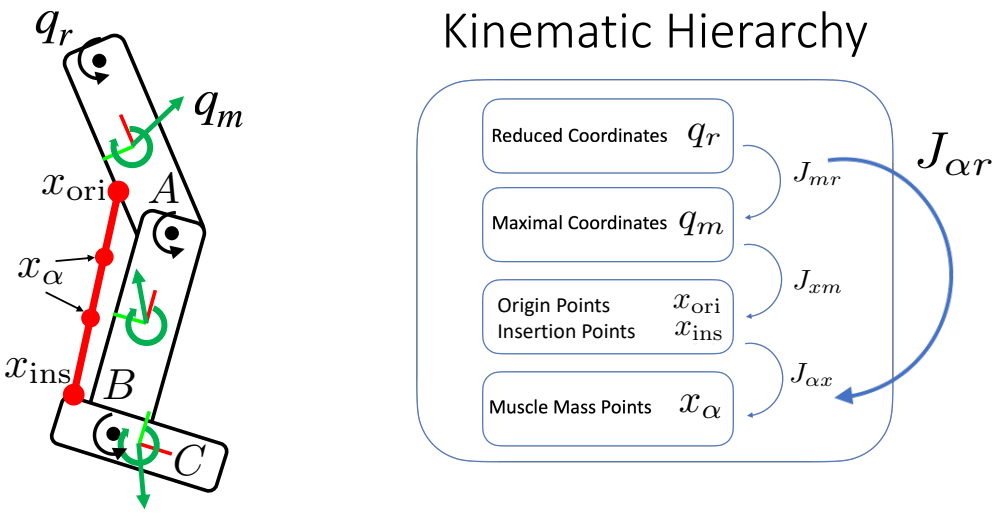


Figure 7.1: We have presented all the Jacobian mappings involved in our kinematic hierarchy, from reduced coordinates to maximal coordinates, then to the 3D coordinates of the origin/insertion points, and finally to the 3D coordinates of the muscle mass points. Finding those mappings for different stages in our hierarchy is essential to achieve efficient inertial muscle simulation.

In this thesis, we presented an approach to account for the inertia of the muscles in a musculoskeletal simulation. Fig. 7.1 shows how our framework works. We use a chain of Jacobians to map between the 3D world coordinates of the muscle mass points to the reduced coordinates of the articulated rigid body system and then drive the dynamic system using only the joint angles of the skeleton. We discussed different stages of our pipeline in each chapter.

*Part of this chapter is reprinted with permission from Y. Wang, J. Verheul, S.-H. Yeo, N. K. Kalantari, and S. Sueda, "Differentiable simulation of inertial musculotendons," *ACM Transactions on Graphics*, vol. 41, Nov. 2022.

In Chapters 3 and 4, we introduced REDMAX, where we derived the Jacobian that maps between maximal and reduced coordinates of the articulated rigid body, and highlighted the flexibility of REDMAX with different results. In Chapter 5, we introduced our inertia muscle, where we can handle a wide variety of musculotendon paths, including (I) straight, (II) polyline, and (III) curved paths over wrapping surfaces. We use the Eulerian-on-Lagrangian framework for Type II muscles, and for Type III muscles, we use neural networks. We showed that 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.

7.2 Main Challenges and Lessons Learned

In order to produce our proposed method and analyze its efficacy, we experienced many difficulties during our research effort. Some of these difficulties deal with the overall research methodology, and others pertain to technical details that seem subtle but are essential to creating a clear and well-supported exposition. We will review these difficulties in this section because they provide valuable lessons on delivering high-quality research on our subject.

The main difficulty in our research on this subject has been the number of iterations it took before arriving at what some regard as the primary technical novelty in our work, i.e., the neural network approach. We changed our method twice before the current neural network approach was considered because we faced weird energy behavior. It finally became clear that the discontinuous Jacobian is an inherent property in our musculoskeletal model and that we require an alternative approach to mitigate its effects on simulation stability.

After we adopted the current neural network method, we experienced further difficulties in neural network training. While finding the mathematical formulation to incorporate neural Jacobian derivatives was a straightforward extension of our earlier research, it was necessary to find expertise outside the original research personnel to complete the training process. The biggest challenge was that model often failed to converge, which was eventually addressed by selectively removing

sample points. Additionally, the large number of parameters that had to be tuned prolonged the training process.

Finally, after months of experimentation, our neural models managed to capture the dynamics of musculoskeletal systems that include wrapping surfaces. There are many challenges along the way. One challenge was creating good energy plots and making the correct comparisons. We added additional training runs to explore further how our choice of models affects the simulation. While our method is an incremental step towards a fully comprehensive simulation framework, we still seek to demonstrate the advantage of our system over traditional ones where possible.

7.3 Future Work

In this work, we use the centerline to account for the muscle mass, which is still an approximation, but this is a prudent choice since using a complete, volumetric mesh is impractical for these experiments, at least currently. It may seem possible to tweak the FEM simulation parameters to produce the desired output. However, we believe that using FEM for these target applications is highly challenging, if not impossible, considering the high number of parameters and the computational complexity required by the volume model. Therefore, it would be a challenge to produce results with FEM that can gracefully degrade to OpenSim results the way our method can.

Future work may address these difficulties with volumetric FEM. 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 [90], moving muscle points [18]). 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.

Our approach still underestimates muscle inertia for muscles with long tendons 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 [44, 17]. In theory, our neural network approach can be used for any path. However, some

wrapping surfaces require many parameters, making training more difficult and slower. For example, training a double cylinder 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 [20] 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 is possible to mix and match inertial and non-inertial musculotendons, depending on the application. 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 [75, 76]. In our current implementation, as with most other musculoskeletal simulators [88], 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.

REFERENCES

[1] T. Komura, Y. Shinagawa, and T. L. Kunii, "An inverse kinematics method based on muscle dynamics," in *Proceedings. Computer Graphics International 2001*, pp. 15–22, IEEE, 2001.

[2] S.-H. Lee and D. Terzopoulos, "Heads up! biomechanical modeling and neuromuscular control of the neck," *ACM Trans. Graph.*, vol. 25, p. 1188–1198, Jul. 2006.

[3] S. Sueda, A. Kaufman, and D. K. Pai, "Musculotendon simulation for hand animation," *ACM Trans. Graph.*, vol. 27, p. 1–8, Aug. 2008.

[4] A. Murai, K. Kurosaki, K. Yamane, and Y. Nakamura, "Musculoskeletal-see-through mirror: Computational modeling and algorithm for whole-body muscle activity visualization in real time," *Progress in biophysics and molecular biology*, vol. 103, no. 2-3, pp. 310–317, 2010.

[5] J. M. Wang, S. R. Hamner, S. L. Delp, and V. Koltun, "Optimizing locomotion controllers using biologically-based actuators and objectives," *ACM Trans. Graph.*, vol. 31, Jul. 2012.

[6] T. Geijtenbeek, M. van de Panne, and A. F. van der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Trans. Graph.*, vol. 32, Nov. 2013.

[7] S.-H. Lee, E. Sifakis, and D. Terzopoulos, "Comprehensive biomechanical modeling and simulation of the upper body," *ACM Trans. Graph.*, vol. 28, Sep. 2009.

[8] W. Si, S.-H. Lee, E. Sifakis, and D. Terzopoulos, "Realistic biomechanical simulation and control of human swimming," *ACM Trans. Graph.*, vol. 34, Dec. 2015.

[9] Y. Lee, M. S. Park, T. Kwon, and J. Lee, "Locomotion control for many-muscle humanoids," *ACM Trans. Graph.*, vol. 33, Nov. 2014.

[10] S. Lee, M. Park, K. Lee, and J. Lee, "Scalable muscle-actuated human simulation and control," *ACM Trans. Graph.*, vol. 38, July 2019.

[11] E. Marieb and K. Hoehn, *Human Anatomy & Physiology*. Benjamin Cummings, 8 ed., 2010.

[12] D. K. Pai, "Muscle mass in musculoskeletal models," *Journal of Biomechanics*, vol. 43, no. 11, pp. 2093–2098, 2010.

[13] M. Han, J. Hong, and F. Park, "Musculoskeletal dynamics simulation using shape-varying muscle mass models," *Multibody System Dynamics*, vol. 33, no. 4, pp. 367–388, 2015.

[14] J. Guo, H. Huang, Y. Yu, Z. Liang, J. Ambrósio, Z. Zhao, G. Ren, and Y. Ao, "Modeling muscle wrapping and mass flow using a mass-variable multibody formulation," *Multibody System Dynamics*, pp. 1–22, 2020.

[15] S. Sueda, G. L. Jones, D. I. W. Levin, and D. K. Pai, "Large-scale dynamic simulation of highly constrained strands," *ACM Trans. Graph.*, vol. 30, Jul. 2011.

[16] P. Sachdeva, S. Sueda, S. Bradley, M. Fain, and D. K. Pai, "Biomechanical simulation and control of hands and tendinous systems," *ACM Trans. Graph.*, vol. 34, Jul. 2015.

[17] B. A. Garner and M. G. Pandy, "The obstacle-set method for representing muscle paths in musculoskeletal models," *Computer methods in biomechanics and biomedical engineering*, vol. 3, no. 1, pp. 1–30, 2000.

[18] S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, C. T. John, E. Guendelman, and D. G. Thelen, "Opensim: open-source software to create and analyze dynamic simulations of movement," *IEEE T BIO-MED ENG*, vol. 54, no. 11, pp. 1940–1950, 2007.

[19] A. Scholz, M. Sherman, I. Stavness, S. Delp, and A. Kecskeméthy, "A fast multi-obstacle muscle wrapping method using natural geodesic variations," *Multibody System Dynamics*, vol. 36, no. 2, pp. 195–219, 2016.

[20] J. E. Lloyd, F. Roewer-Després, and I. Stavness, "Muscle path wrapping on arbitrary surfaces," *IEEE T BIO-MED ENG*, vol. 68, no. 2, pp. 628–638, 2020.

[21] D.-S. Bae and E. J. Haug, "A recursive formulation for constrained mechanical system dynamics: Part I. open loop systems," *J STRUCT MECH*, vol. 15, no. 3, pp. 359–382, 1987.

[22] D.-S. Bae and E. J. Haug, "A recursive formulation for constrained mechanical system dynamics: Part II. closed loop systems," *J STRUCT MECH*, vol. 15, no. 4, pp. 481–506, 1987.

[23] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *J DYN SYST-T ASME*, vol. 104, no. 3, pp. 205–211, 1982.

[24] A. Avello, J. M. Jiménez, E. Bayo, and J. G. de Jalón, "A simple and highly parallelizable method for real-time dynamic simulation based on velocity transformations," *Comput. Methods in Appl. Mech. Eng.*, vol. 107, no. 3, pp. 313–339, 1993.

[25] D. Negrut, R. Serban, and F. A. Potra, "A topology-based approach to exploiting sparsity in multibody dynamics: Joint formulation," *J STRUCT MECH*, vol. 25, no. 2, pp. 221–241, 1997.

[26] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Annual Conference Series (Proc. SIGGRAPH)*, pp. 43–54, 1998.

[27] T. Shinar, C. Schroeder, and R. Fedkiw, "Two-way coupling of rigid and deformable bodies," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, pp. 95–103, 2008.

[28] J. Kim and N. S. Pollard, "Fast simulation of skeleton-driven deformable body characters," *ACM Trans. Graph.*, vol. 30, pp. 121:1–121:19, Oct. 2011.

[29] S. Jain and C. K. Liu, "Controlling physics-based characters using soft contacts," *ACM Trans. Graph.*, vol. 30, pp. 163:1–163:10, Dec. 2011.

[30] L. Liu, K. Yin, B. Wang, and B. Guo, "Simulation and control of skeleton-driven soft body characters," *ACM Trans. Graph.*, vol. 32, pp. 215:1–215:8, Nov. 2013.

[31] E. Quigley, Y. Yu, J. Huang, W. Lin, and R. Fedkiw, "Real-time interactive tree animation," *IEEE TVCG*, vol. 24, pp. 1717–1727, May 2018.

[32] S. Hadap, "Oriented strands: Dynamics of stiff multi-body system," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, SCA '06, pp. 91–100, 2006.

[33] F. Hernandez, C. Garre, R. Casillas, and M. A. Otaduy, "Linear-time dynamics of characters with stiff joints," in *V Ibero-American Symposium on Computer Graphics (SIACG 2011)*, The Eurographics Association and Blackwell Publishing Ltd, June 2011.

[34] T. M. Wasfy and A. K. Noor, "Computational strategies for flexible multibody systems," *APPL MECH REV*, vol. 56, no. 6, pp. 553–613, 2003.

[35] F. Bertails, "Linear time super-helices," *Computer Graphics Forum (Proc. Eurographics)*, vol. 28, pp. 417–426, May 2009.

[36] K. Waters, "A muscle model for animation three-dimensional facial expression," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 17–24, 1987.

[37] D. Terzopoulos and K. Waters, "Physically-based facial modelling, analysis, and animation," *The Journal of Visualization and Computer Animation*, vol. 1, no. 2, pp. 73–80, 1990.

[38] K. Waters and D. Terzopoulos, "A physical model of facial tissue and muscle articulation," in *[1990] Proceedings of the First Conference on Visualization in Biomedical Computing*, pp. 77–78, IEEE Computer Society, 1990.

[39] F. Scheepers, R. E. Parent, W. E. Carlson, and S. F. May, "Anatomy-based modeling of the human musculature," in *Proc. SIGGRAPH 97*, Annual Conference Series, pp. 163–172, ACM, 1997.

[40] J. Wilhelms and A. V. Gelder, "Anatomically based modeling," in *Proc. SIGGRAPH 97*, Annual Conference Series, pp. 173–180, ACM, 1997.

[41] Autodesk, INC., "Maya."

[42] Ziva Dynamics, "Ziva VFX."

[43] M. Damsgaard, J. Rasmussen, S. T. Christensen, E. Surma, and M. De Zee, "Analysis of musculoskeletal systems in the anybody modeling system," *Simulation Modelling Practice and Theory*, vol. 14, no. 8, pp. 1100–1111, 2006.

[44] A. Seth, J. L. Hicks, T. K. Uchida, A. Habib, C. L. Dembia, J. J. Dunne, C. F. Ong, M. S. DeMers, A. Rajagopal, M. Millard, *et al.*, "OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement," *PLoS computational biology*, vol. 14, no. 7, p. e1006223, 2018.

[45] T. Komura, Y. Shinagawa, and T. L. Kunii, "A muscle-based feed-forward controller of the human body," in *Computer Graphics Forum*, vol. 16, pp. C165–C176, Wiley Online Library, 1997.

[46] T. Komura, Y. Shinagawa, and T. L. Kunii, "Creating and retargetting motion by the musculoskeletal human body model," *The visual computer*, vol. 16, no. 5, pp. 254–270, 2000.

[47] D. T. Chen and D. Zeltzer, "Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method," *SIGGRAPH Comput. Graph.*, vol. 26, p. 89–98, Jul. 1992.

[48] Q.-h. Zhu, Y. Chen, and A. Kaufman, "Real-time biomechanically-based muscle volume deformation using fem," in *Computer Graphics Forum*, vol. 17, pp. 275–284, Wiley Online Library, 1998.

[49] R. Lemos, M. Epstein, W. Herzog, and B. Wyvill, "Realistic skeletal muscle deformation using finite element analysis," in *Proceedings XIV Brazilian Symposium on Computer Graphics and Image Processing*, pp. 192–199, IEEE, 2001.

[50] V. Ng-Thow-Hing, *Anatomically-based models for physical and geometric reconstruction of humans and other animals*. PhD thesis, The University of Toronto, 2001.

[51] J. Teran, S. Blemker, V. Ng-Thow-Hing, and R. Fedkiw, "Finite volume methods for the simulation of skeletal muscle," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, (Goslar, DEU), p. 68–74, Eurographics Association, 2003.

[52] J. Teran, E. Sifakis, S. S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw, "Creating and simulating skeletal muscle from the visible human data set," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, p. 317–328, May 2005.

[53] E. Sifakis, I. Neverov, and R. Fedkiw, "Automatic determination of facial muscle activations from sparse motion capture marker data," *ACM Trans. Graph.*, vol. 24, p. 417–425, Jul. 2005.

[54] Y. Fan, J. Litven, and D. K. Pai, "Active volumetric musculoskeletal systems," *ACM Trans. Graph.*, vol. 33, Jul. 2014.

[55] S. Lee, R. Yu, J. Park, M. Aanjaneya, E. Sifakis, and J. Lee, "Dexterous manipulation and control with volumetric muscles," *ACM Trans. Graph.*, vol. 37, Jul. 2018.

[56] S. Min, J. Won, S. Lee, J. Park, and J. Lee, "Softcon: Simulation and control of soft-bodied animals with biomimetic actuators," *ACM Trans. Graph.*, vol. 38, Nov. 2019.

[57] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Comput. Methods in Appl. Mech. Eng.*, vol. 1, pp. 1–16, Jun 1972.

[58] M. B. Cline and D. K. Pai, "Post-stabilization for rigid body simulation with contact and constraints," in *IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 3744–3751, 2003.

[59] D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai, "Staggered projections for frictional contact in multibody systems," *ACM Trans. Graph.*, vol. 27, pp. 164:1–164:11, Dec 2008.

[60] J. M. Selig, "Lie groups and Lie algebras in robotics," in *Computational Noncommutative Algebra and Applications*, pp. 101–125, Springer, 2004.

[61] J. Kim, "Lie group formulation of articulated rigid body dynamics," tech. rep., Carnegie Mellon University, 12 2012.

[62] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[63] B. Mirtich, "Fast and accurate computation of polyhedral mass properties," *J. Graph. Tools*, vol. 1, no. 2, pp. 31–50, 1996.

[64] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[65] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *INT J ROBOT RES*, vol. 2, no. 1, pp. 13–30, 1983.

[66] F. C. Park, J. E. Bobrow, and S. R. Ploen, "A lie group formulation of robot dynamics," *INT J ROBOT RES*, vol. 14, no. 6, pp. 609–618, 1995.

[67] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Annual Conference Series (Proc. SIGGRAPH)*, pp. 137–146, 1996.

[68] A. A. Shabana, *Dynamics of Multibody Systems*. Cambridge University press, 2013.

[69] F. S. Grassia, "Practical parameterization of rotations using the exponential map," *J. Graph. Tools*, vol. 3, pp. 29–48, Mar. 1998.

[70] G. Gallego and A. Yezzi, "A compact formula for the derivative of a 3-D rotation in exponential coordinates," *J. Math. Imaging Vision*, vol. 51, no. 3, pp. 378–384, 2015.

[71] S.-H. Lee and D. Terzopoulos, "Spline joints for multibody dynamics," *ACM Trans. Graph.*, vol. 27, pp. 22:1–22:8, Aug. 2008.

[72] M.-J. Kim, M.-S. Kim, and S. Y. Shin, "A general construction scheme for unit quaternion curves with simple high order derivatives," in *Annual Conference Series (Proc. SIGGRAPH)*, (New York, NY, USA), pp. 369–376, ACM, 1995.

[73] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, vol. 31. Springer Science & Business Media, 2006.

[74] A. McNamara, A. Treuille, Z. Popović, and J. Stam, "Fluid control using the adjoint method," *ACM Trans. Graph.*, vol. 23, p. 449–456, aug 2004.

[75] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "Add: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Trans. Graph.*, vol. 39, Nov. 2020.

[76] J. Xu, T. Chen, L. Zlokapa, W. Matusik, S. Sueda, and P. Agrawal, "An end-to-end differentiable framework for contact-aware robot design," in *Robotics: Science and Systems*, 2021.

[77] Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, "REDMAX: Efficient & flexible approach for articulated dynamics," *ACM Trans. Graph.*, vol. 38, Jul. 2019.

[78] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[79] E. Sifakis and J. Barbic, "Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction," in *ACM SIGGRAPH 2012 Courses*, 2012.

[80] J. Leijnse, P. Quesada, and C. Spoor, "Kinematic evaluation of the finger's interphalangeal joints coupling mechanism—variability, flexion–extension differences, triggers, locking swan-neck deformities, anthropometric correlations," *Journal of Biomechanics*, vol. 43, no. 12, pp. 2381–2393, 2010.

[81] MATLAB, *version 9.9.0 (R2020b)*. Natick, Massachusetts: The MathWorks Inc., 2021.

[82] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[83] J. H. Lee, D. S. Asakawa, J. T. Dennerlein, and D. L. Jindrich, "Finger muscle attachments for an opensim upper-extremity model," *PloS one*, vol. 10, no. 4, p. e0121712, 2015.

[84] J. H. Kim, L. Aulck, M. C. Bartha, C. A. Harper, and P. W. Johnson, "Differences in typing forces, muscle activity, comfort, and typing performance among virtual, notebook, and desktop keyboards," *Applied Ergonomics*, vol. 45, no. 6, pp. 1406–1413, 2014.

[85] S. R. Ward, C. M. Eng, L. H. Smallwood, and R. L. Lieber, "Are current measurements of lower extremity muscle architecture accurate?," *Clinical orthopaedics and related research*, vol. 467, no. 4, pp. 1074–1082, 2009.

[86] A. Seth, M. Sherman, P. Eastman, and S. Delp, "Minimal formulation of joint motion for biomechanisms," *Nonlinear dynamics*, vol. 62, no. 1, pp. 291–303, 2010.

[87] F. E. Zajac, "Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control.," *Critical reviews in biomedical engineering*, vol. 17, no. 4, pp. 359–411, 1989.

[88] M. Millard, T. Uchida, A. Seth, and S. L. Delp, "Flexing computational muscle: modeling and simulation of musculotendon dynamics," *Journal of biomechanical engineering*, vol. 135, no. 2, p. 021005, 2013.

[89] E. K. Chadwick, D. Blana, R. F. Kirsch, and A. J. Van Den Bogert, "Real-time simulation of three-dimensional shoulder girdle and arm dynamics," *IEEE T BIO-MED ENG*, vol. 61, no. 7, pp. 1947–1956, 2014.

[90] H. Ryu, M. Kim, S. Lee, M. S. Park, K. Lee, and J. Lee, "Functionality-driven musculature retargeting," in *Computer Graphics Forum*, vol. 40, pp. 341–356, Wiley Online Library, 2021.