A Versatile Quaternion-based Constrained Rigid Body Dynamics

GUIREC MALOISEL*, Disney Research, Switzerland
RUBEN GRANDIA, Disney Research, Switzerland
CHRISTIAN SCHUMACHER, Disney Research, Switzerland
ESPEN KNOOP, Disney Research, Switzerland
MORITZ BÄCHER*, Disney Research, Switzerland

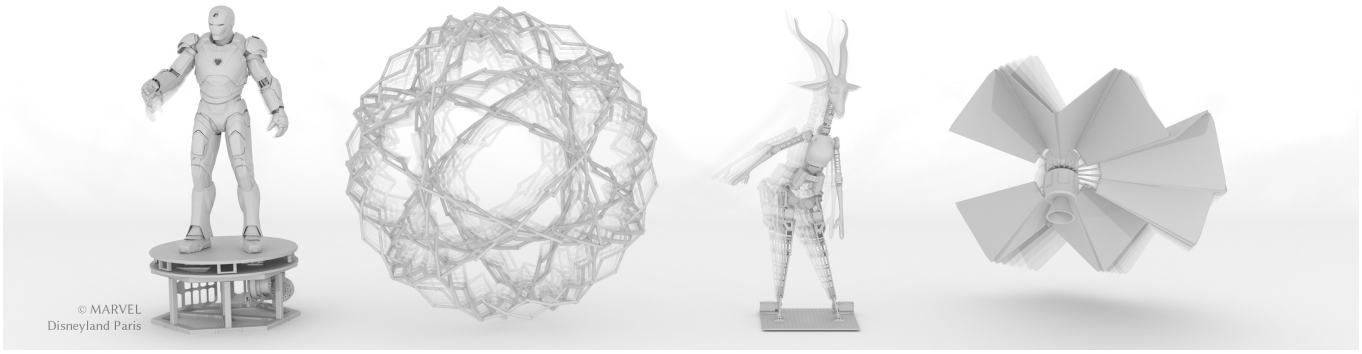

Fig. 1. Our constrained rigid body dynamics can simulate complex kinematic structures, as illustrated here with (from left to right) an Iron Man Audio-Animatronics® figure, a Hoberman sphere, a Gazelle Audio-Animatronics® figure, and the solar panel deployment mechanism of a satellite. Our formulation guarantees constraint satisfaction and handles systems with kinematic loops, redundant constraints, overactuation, and passive degrees of freedom.

We present a constrained Rigid Body Dynamics (RBD) that guarantees satisfaction of kinematic constraints, enabling direct simulation of complex mechanical systems with arbitrary kinematic structures. To ensure constraint satisfaction, we use an implicit integration scheme. For this purpose, we derive compatible dynamic equations expressed through the quaternion time derivative, adopting an additive approach to quaternion updates instead of a multiplicative one, while enforcing quaternion unit-length as a constraint. We support all joints between rigid bodies that restrict subsets of the three translational or three rotational degrees of freedom, including position- and force-based actuation. Their constraints are formulated such that Lagrange multipliers are interpretable as joint forces and torques. We discuss a unified solution strategy for systems with redundant constraints, overactuation, and passive degrees of freedom, by eliminating redundant constraints and navigating the subspaces spanned by multipliers. As our method uses a standard additive update, we can interface with unconditionally-stable implicit integrators. Moreover, the simulation can readily be made differentiable as we show with examples.

*Both authors contributed equally to this research.

Authors' Contact Information: Guirec Maloisel, guirec.maloisel@disney.com, Disney Research, Switzerland; Ruben Grandia, ruben.grandia@disney.com, Disney Research, Switzerland; Christian Schumacher, christian.schumacher@disney.com, Disney Research, Switzerland; Espen Knoop, espen.knoop@disney.com, Disney Research, Switzerland; Moritz Bächer, moritz.baecher@disney.com, Disney Research, Switzerland.

ACM 1557-7368/2025/8-ART
https://doi.org/10.1145/3730872

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

Additional Key Words and Phrases: rigid body dynamics, kinematic constraints, differentiable simulation

ACM Reference Format:
Guirec Maloisel, Ruben Grandia, Christian Schumacher, Espen Knoop, and Moritz Bächer. 2025. A Versatile Quaternion-based Constrained Rigid Body Dynamics. *ACM Trans. Graph.* 44, 4 (August 2025), 17 pages. https://doi.org/10.1145/3730872

1 Introduction

Rigid bodies are omnipresent in virtual environments as well as in mechanical systems in the real world. We are interested in simulating the dynamics of systems with arbitrary kinematic structures, including loops and passive degrees of freedom, which requires a constraint-based formulation. For the accurate simulation of such systems, it is imperative that constraints remain preserved — in particular in the vicinity of singularities where even small constraint violations can cause entirely wrong simulations.

Commonly used explicit and semi-explicit time-stepping schemes do, in general, not offer guarantees on constraint satisfaction for arbitrary systems and steps sizes. In contrast, implicit integration schemes allow the inclusion of kinematic constraints at the next time step and hence solving for them up to a specified tolerance.

For implicit schemes, rotational motion presents a challenge. Quaternions are a common representation choice, being compact and singularity-free, however their use within such an implicit scheme requires care when handling the unit-length constraint. While this can be handled using implicit Lie-group integrators, their specialized multiplicative algebra complicates the use of these formulations in downstream applications.

Our proposed rigid body dynamics instead writes the dynamical equations in terms of the quaternion time derivative, formulating an additive rather than multiplicative quaternion update, and incorporates the quaternion unit length as a constraint. This enables interfacing with implicit integration schemes for differential-algebraic equations (DAEs) and makes the simulation readily differentiable.

We handle systems with position- or force-based actuation and configurations with redundant kinematic constraints and overactuation, which frequently arise in real-world systems (see Fig. 1). In the proposed method, we explicitly handle the non-uniqueness of constraint forces in such redundant and overactuated configurations and provide the user with control over the desired solution.

We begin by deriving the proposed equations of motion for a single rigid body directly from the Euler-Lagrange equations (Sec. 3). We then express atomic translational and rotational constraints, providing a unified way of formulating passive, position- and force-actuated joints as an extension of the dot-product formulation for kinematic constraints (Sec. 4). Next, we discuss the implicit integration of the resulting constrained rigid body dynamics (Secs. 5 and 6), and the differentiability of our simulator (Sec. 7). We evaluate our method on a rich set of examples, including targeted studies of different integration schemes, and compare with a common semi-implicit approach (Sec. 8).

Succinctly, we contribute

- a quaternion-based RBD that enforces unit-length constraints without the need for Lagrange multipliers.
- a unified treatment of passive, position- or force-actuated joints, facilitating analysis of constraint forces and torques by making corresponding Lagrange multipliers directly interpretable.
- a solution strategy that combines implicit time integration and a Newton-type solver to handle fully-, over-, or underactuated systems as well as redundancy in the kinematic constraints and the corresponding force subspaces.
- a demonstration of the differentiability of the simulator, facilitated by the additive orientation update.

2 Related Work

We first discuss related work on the integration of the equations of motion of a single, unconstrained body, followed by a discussion of incorporating constraints.

Rigid Body Dynamics. Motion equations for a single body differ in how they represent its orientation and angular velocity. Unit quaternions are free of singularities, but only represent rotations if they are kept unit length. Renormalization [Witkin and Baraff 1997] introduces inaccuracies due to the ad-hoc projection operation. The use of non-unit-length quaternions [Rucker 2018] has been shown for single bodies, but has not been extended to incorporate constraints. Variational integrators inherently maintain unit length by using an exponential map [Grassia 1998; Kobilarov et al. 2009; Simo and Wong 1991; Wieloch and Arnold 2021], however their multiplicative update hinders downstream use (e.g., differentiability).

Similar to previous work [Betsch and Siebert 2009; Möller and Glocker 2012; Nielsen and Krenk 2012; Nikravesh et al. 1985b; Udwadia and Schutte 2010; Xu et al. 2020], we derive quaternion-based motion equations from the Lagrangian or Hamiltonian, rely on a unit-length constraint, and use the time derivative of the quaternion to represent the angular velocity. However, our formulation avoids the introduction of a 4D augmented inertia matrix that does not have a physical meaning [Betsch and Siebert 2009; Xu et al. 2020] and enforces the unit-length constraint *without* the need for a Lagrange multiplier, resulting in 3 instead of 4 angular motion equations and as many equations as unknowns.

Kinematic Constraints. The incorporation of kinematic constraints, a.k.a. geometric or bilateral constraints, into RBDs has received attention for decades [Bender et al. 2014]. Our constraint formulation is similar to previous work [Bächer et al. 2015; Barzel and Barr 1988; Coros et al. 2013; Haug 1989; Maloisel et al. 2021, 2023; Schumacher et al. 2021; Thomaszewski et al. 2014]. However, we make sure that the corresponding Lagrange multipliers can be interpreted as either forces or torques in joint coordinates, avoiding an unintuitive scaling due to the use of the imaginary part of the quaternion [Tasora and Righettini 1999]. This in turn enables the formulation of more general force-based actuators that are consistent with previously introduced position-based actuators [Maloisel et al. 2023]. Furthermore, we propose an explicit treatment of the extraneous kinematic solutions in rotational constraints ("joint flips") that can emerge for large simulation steps.

Constrained Multibody Dynamics. Rigid body systems with many loop-closure constraints are stiff systems and hence challenging to stably integrate. Methods for articulated [Geilinger et al. 2020] or hybrid systems with only a few kinematic loops [Tomcin et al. 2014; Wang et al. 2019] are therefore more common, where minimal coordinate formulations are frequently used. Another body of work focuses on the resolution of unilateral constraints or frictional contact [Erleben 2017; Ferguson et al. 2021; Kaufman et al. 2005, 2008; Smith et al. 2012], or relaxes the problem by assuming some [Hoshyari et al. 2019] or all bodies [Li et al. 2020; Tournier et al. 2015] to be flexible or at least close-to-rigid [Lan et al. 2022]. This flexibility not only reduces the stiffness in the system, but also regularizes subspaces in constraint forces and torques.

Simulation software in graphics and robotics (PhysX, ODE, Mujoco) [Erez et al. 2015], support both bilateral and unilateral constraints and primarily use semi-implicit integration, where velocities are implicitly solved and positions are explicitly integrated thereafter. This relies on stabilized velocity constraints [Ascher et al. 1995; Baumgarte 1972], which require tuning stabilization parameters and cannot guarantee position constraint satisfaction. Instead, we implicitly integrate the resulting DAEs and directly solve for position constraints.

3 A Quaternion-based Rigid Body Dynamics

We derive our constrained dynamics directly from the Euler-Lagrange equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T}{\partial \dot{\mathbf{s}}}\right)^T - \left(\frac{\partial T}{\partial \mathbf{s}}\right)^T = \mathbf{C}_\mathbf{s}^T \boldsymbol{\lambda} + \mathbf{f}_{\text{gen}} \tag{1}$$

with the goal to solve for the time-varying pose of the rigid bodies, $\mathbf{s}$, corresponding velocities, $\dot{\mathbf{s}}$, and Lagrange multipliers, $\boldsymbol{\lambda}$, so that a mechanical system is in dynamic equilibrium at all times t. Because

we assume non-conservative *and* conservative forces to be part of the generalized force term, $\mathbf{f}_{\text{gen}}$, we set the Lagrangian to the kinetic energy T, omitting a potential energy term for conservative forces. To enforce a set of equality constraints, $\mathbf{C} = \mathbf{0}$, we choose multipliers $\boldsymbol{\lambda}$ so that the generalized forces, $\mathbf{C}_\mathbf{s}^T\boldsymbol{\lambda}$, do not do any work on the system, where $\mathbf{C}_\mathbf{s}$ is the Jacobian of the constraints or their derivatives with respect to the pose.

For the remainder of this section, we will focus on a single rigid body, deferring a discussion of the constrained motion of several rigid bodies to the next section.

We represent the state of a rigid body with the position of its center of mass, $\mathbf{c}$, in global coordinates and its orientation with a unit quaternion $\mathbf{q}$

$$\mathbf{s} = \begin{bmatrix}\mathbf{c}\\ \mathbf{q}\end{bmatrix}, \quad \text{together with corresponding velocities} \quad \dot{\mathbf{s}} = \begin{bmatrix}\mathbf{v}\\ \mathbf{w}\end{bmatrix}. \tag{2}$$

While it is common practice to represent the linear velocity of rigid bodies with $\mathbf{v} = \dot{\mathbf{c}}$, it is far less common to use the time derivative of the quaternion, $\mathbf{w} = \dot{\mathbf{q}}$, instead of the angular velocity $\boldsymbol{\omega}$, for angular motion. Instead of working with a 13-dimensional set of state variables ($\mathbf{c}$, $\mathbf{q}$, $\mathbf{v}$, $\boldsymbol{\omega}$), we work with a 14-dimensional set ($\mathbf{c}$, $\mathbf{q}$, $\mathbf{v}$, $\mathbf{w}$) and enforce the unit-length constraint, $C = \mathbf{q}^T\mathbf{q} - 1$, explicitly.

To derive the equations of motion of the body, we define its kinetic energy $T = T_{\text{lin}} + T_{\text{ang}}$ that is due to its linear and angular motion

$$T_{\text{lin}}(\dot{\mathbf{c}}) = \frac{1}{2}\dot{\mathbf{c}}^T M\dot{\mathbf{c}} \qquad T_{\text{ang}}(\mathbf{q},\boldsymbol{\omega}) = \frac{1}{2}\boldsymbol{\omega}^T\mathbf{R}(\mathbf{q})\mathbf{J}_{\text{rb}}\mathbf{R}(\mathbf{q})^T\boldsymbol{\omega}, \tag{3}$$

where $\mathbf{J}_{\text{rb}}$ is the constant moment of inertia, rotated to global coordinates using the rotation matrix $\mathbf{R}(\mathbf{q})$. Matrix $\mathbf{R}(\mathbf{q})$ can be expressed with two matrices, $\mathbf{G}(\mathbf{q})$ and $\mathbf{H}(\mathbf{q})$, whose entries *linearly* depend on the coordinates of the unit quaternion (see Tab. 1, entries and row 6). These matrices have a set of remarkable properties as previously discovered (see, e.g., [Nikravesh et al. 1985a,b]) and summarized in Tab. 1.

For the linear motion of a body, the Euler-Lagrange equations result in Newton's second law

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T_{\text{lin}}}{\partial\dot{\mathbf{c}}}\right)^T - \left(\frac{\partial T_{\text{lin}}}{\partial\mathbf{c}}\right)^T = M\ddot{\mathbf{c}} = M\dot{\mathbf{v}} = \mathbf{f}. \tag{4}$$

with external forces $\mathbf{f} \in \mathbb{R}^3$ that act on the body's center of mass.

The Euler-Lagrange equations for the angular motion of the body,

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial T_{\text{ang}}}{\partial\dot{\mathbf{q}}}\right)^T - \left(\frac{\partial T_{\text{ang}}}{\partial\mathbf{q}}\right)^T = C_\mathbf{q}^T\lambda + 2\mathbf{G}(\mathbf{q})^T\boldsymbol{\tau}, \tag{5}$$

have a less trivial right-hand side, because the derivative of the unit-length constraint is non-zero and the external torque around the center of mass, $\boldsymbol{\tau}$, needs to be transformed to the 4D space by applying the transformation $2\mathbf{G}(\mathbf{q})^T$. The particular form of this transformation can be traced back to the relationship between the angular velocity and the time derivative of the quaternion (property 10 in Tab. 1).

Before taking derivatives, it is convenient to use the properties in Tab. 1 to derive two variants of the kinetic energy

$$T_{\text{ang}}(\mathbf{q},\dot{\mathbf{q}}) \overset{6,10,3}{=} 2\dot{\mathbf{q}}^T\mathbf{H}(\mathbf{q})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}} \overset{2}{=} 2\mathbf{q}^T\mathbf{H}(\dot{\mathbf{q}})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\dot{\mathbf{q}})\mathbf{q} \tag{6}$$

that only depend on the quaternion and its time derivative. The numbers above the equality signs indicate which properties we used and in which order we applied them. The first variant makes it straightforward to take the derivatives of T_{ang} with respect to $\dot{\mathbf{q}}$ and time t

$$4\mathbf{H}(\mathbf{q})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \cancel{4\mathbf{H}(\mathbf{q})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\dot{\mathbf{q}})\dot{\mathbf{q}}} + 4\mathbf{H}(\dot{\mathbf{q}})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}} \tag{7}$$

where the second term is zero due to property 5. The second variant makes it trivial to take the derivative with respect to $\mathbf{q}$

$$4\mathbf{H}(\dot{\mathbf{q}})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\dot{\mathbf{q}})\mathbf{q} \overset{2}{=} -4\mathbf{H}(\dot{\mathbf{q}})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}}. \tag{8}$$

The 4D Euler-Lagrange equations for angular motion are therefore

$$4\mathbf{H}(\mathbf{q})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + 8\mathbf{H}(\dot{\mathbf{q}})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}} = 2\mathbf{q}\lambda + 2\mathbf{G}(\mathbf{q})^T\boldsymbol{\tau}. \tag{9}$$

In contrast to related work [Betsch and Siebert 2009; Nielsen and Krenk 2012; Xu et al. 2020], we project the equations to the 3D space by multiplying either side with $\frac{1}{2}\mathbf{G}(\mathbf{q})$, applying properties 1 and 3 to simplify the right-hand side

$$2\mathbf{G}(\mathbf{q})\mathbf{H}(\mathbf{q})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + 4\mathbf{G}(\mathbf{q})\mathbf{H}(\dot{\mathbf{q}})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\dot{\mathbf{q}} = \boldsymbol{\tau}. \tag{10}$$

Note how the generalized force due to the unit-length constraint cancels. The Lagrange multiplier can therefore be ignored, resulting

Table 1. **Properties of Matrices $\mathbf{G}(\mathbf{q})$, $\mathbf{H}(\mathbf{q})$, and $\mathbf{R}(\mathbf{q})$.** We number the properties (left column) and indicate if we use them in derivations by adding the property number(s) above the equal sign. We first summarize the entries and properties (1-5) of the individual matrices $\mathbf{G}(\mathbf{q})$ and $\mathbf{H}(\mathbf{q})$ (top), then list properties that use a combination of them to express the rotation matrix (6), its time derivative (7,8), and relate the angular velocity to the time derivative of the quaternion and vice versa (9, 10). The properties can be verified by substituting $\mathbf{q} = [q_x\ q_y\ q_z\ q_w]^T$, with the real part q_w, and by using the unit-length property, $\mathbf{q}^T\mathbf{q} = 1$. $\mathbf{I}_3$ and $\mathbf{I}_4$ are 3D and 4D identity matrices and $[\mathbf{a}]_\times$ is the skew-symmetric matrix form of the cross product of $\mathbf{a}$ with another vector.

	$\mathbf{G}(\mathbf{q})$	$\mathbf{H}(\mathbf{q})$
Entries	$\begin{bmatrix} q_w & -q_z & q_y & -q_x \\ q_z & q_w & -q_x & -q_y \\ -q_y & q_x & q_w & -q_z \end{bmatrix}$	$\begin{bmatrix} q_w & q_z & -q_y & -q_x \\ -q_z & q_w & q_x & -q_y \\ q_y & -q_x & q_w & -q_z \end{bmatrix}$
1	$\mathbf{G}(\mathbf{q})\mathbf{q} = \mathbf{0}$	$\mathbf{H}(\mathbf{q})\mathbf{q} = \mathbf{0}$
2	for $\mathbf{v} \in \mathbb{R}^4$: $\mathbf{G}(\mathbf{q})\mathbf{v} = -\mathbf{G}(\mathbf{v})\mathbf{q}$	for $\mathbf{v} \in \mathbb{R}^4$: $\mathbf{H}(\mathbf{q})\mathbf{v} = -\mathbf{H}(\mathbf{v})\mathbf{q}$
3	$\mathbf{G}(\mathbf{q})\mathbf{G}(\mathbf{q})^T = \mathbf{I}_3$	$\mathbf{H}(\mathbf{q})\mathbf{H}(\mathbf{q})^T = \mathbf{I}_3$
4	$\mathbf{G}(\mathbf{q})^T\mathbf{G}(\mathbf{q}) = -\mathbf{q}\mathbf{q}^T + \mathbf{I}_4$	$\mathbf{H}(\mathbf{q})^T\mathbf{H}(\mathbf{q}) = -\mathbf{q}\mathbf{q}^T + \mathbf{I}_4$
5	$\mathbf{G}(\dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{0}$	$\mathbf{H}(\dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{0}$

6	$\mathbf{R}(\mathbf{q}) = \mathbf{G}(\mathbf{q})\mathbf{H}(\mathbf{q})^T$
Entries	$\begin{bmatrix} q_w^2+q_x^2-q_y^2-q_z^2 & 2q_xq_y-2q_wq_z & 2q_wq_y+2q_xq_z \\ 2q_wq_z+2q_xq_y & q_w^2-q_x^2+q_y^2-q_z^2 & 2q_yq_z-2q_wq_x \\ 2q_xq_z-2q_wq_y & 2q_wq_x+2q_yq_z & q_w^2-q_x^2-q_y^2+q_z^2 \end{bmatrix}$
7	$\mathbf{G}(\mathbf{q})\mathbf{H}(\dot{\mathbf{q}})^T = \mathbf{G}(\dot{\mathbf{q}})\mathbf{H}(\mathbf{q})^T$
8	$\dot{\mathbf{R}} \overset{7}{=} 2\mathbf{G}(\mathbf{q})\mathbf{H}(\dot{\mathbf{q}})^T \overset{7}{=} 2\mathbf{G}(\dot{\mathbf{q}})\mathbf{H}(\mathbf{q})^T$
9	$[\boldsymbol{\omega}]_\times = \dot{\mathbf{R}}\mathbf{R}^T \overset{6,8}{=} 2\mathbf{G}(\dot{\mathbf{q}})\mathbf{H}(\mathbf{q})^T\mathbf{H}(\mathbf{q})\mathbf{G}(\mathbf{q})^T \overset{4,1}{=} 2\mathbf{G}(\dot{\mathbf{q}})\mathbf{G}(\mathbf{q})^T$
10	$\boldsymbol{\omega} \overset{9}{=} 2\mathbf{G}(\mathbf{q})\dot{\mathbf{q}} \quad\Leftrightarrow\quad \dot{\mathbf{q}} \overset{3}{=} \frac{1}{2}\mathbf{G}(\mathbf{q})^T\boldsymbol{\omega}$

in the first-order system with 14 equations

$$\begin{bmatrix} \mathbf{F1} \\ \mathbf{F2} \\ \hline \mathbf{F3} \\ \mathbf{F4} \\ \hline \mathbf{F5} \end{bmatrix} = \left[\begin{array}{ccc} \dot{\mathbf{c}} & - & \mathbf{v} \\ \dot{\mathbf{q}} & - & \mathbf{w} \\ \hline M\dot{\mathbf{v}} & - & \mathbf{f} \\ 2\mathbf{R}(\mathbf{q})\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\dot{\mathbf{w}} & + & 4\mathbf{G}(\mathbf{q})\mathbf{H}(\mathbf{w})^T\mathbf{J}_{\text{rb}}\mathbf{H}(\mathbf{q})\mathbf{w} - \boldsymbol{\tau} \\ \hline & & \mathbf{q}^T\mathbf{q} - 1 \end{array}\right] = \mathbf{0}$$

in 14 unknown state variables $\mathbf{c}$, $\mathbf{q}$, $\mathbf{v}$, and $\mathbf{w}$ that is in equilibrium if equations **F1**−**F5** evaluate to zero. Because the last equation does not depend on the time derivative of a state variable, the system consists of a set of DAEs and not only ordinary differential equations (ODEs). We will discuss numerical solution strategies after incorporating kinematic constraints into our RBD.

4 Incorporating Constraints

Our kinematic constraints for mechanical joints and position-based actuators are similar to related work that focuses on kinematics [Maloisel et al. 2023; Schumacher et al. 2021]. However, because we use them in the context of a dynamics simulation, we make adjustments so that Lagrange multipliers consistently represent forces and torques in joint coordinates of one of the two bodies. Moreover, we show that we can turn every position-based actuator [Maloisel et al. 2023] into a corresponding force-based actuator by replacing Lagrange multipliers with parameters in constraint forces and torques.

Constraints. To restrict the motion between a follower body F and a base body B, we add a constraint

$$C(\mathbf{c}^F, \mathbf{q}^F, \mathbf{c}^B, \mathbf{q}^B) = 0 \tag{11}$$

for every translation or rotation along or about an axis without mobility to the equations of motion **F** of the individual bodies. At the same time, we add a Lagrange multiplier λ to the unknown state $\mathbf{y}$, keeping the number of equations and unknowns equal by construction as illustrated here for a two-body system

$$\mathbf{F}(\dot{\mathbf{y}}, \mathbf{y}, t) = \left[\begin{array}{ccc} \mathbf{F1}^F & & \\ \mathbf{F2}^F & & \\ \hline \mathbf{F3}^F & - & \mathbf{f}^F \\ \mathbf{F4}^F & - & \boldsymbol{\tau}^F \\ \hline \mathbf{F5}^F & & \\ \hline \mathbf{F1}^B & & \\ \mathbf{F2}^B & & \\ \hline \mathbf{F3}^B & - & \mathbf{f}^B \\ \mathbf{F4}^B & - & \boldsymbol{\tau}^B \\ \hline \mathbf{F5}^B & & \\ \hline & & C \end{array}\right] = \mathbf{0} \qquad \mathbf{y} = \begin{bmatrix} \mathbf{c}^F \\ \mathbf{q}^F \\ \hline \mathbf{v}^F \\ \mathbf{w}^F \\ \hline \mathbf{c}^B \\ \mathbf{q}^B \\ \hline \mathbf{v}^B \\ \mathbf{w}^B \\ \hline \lambda \end{bmatrix}. \tag{12}$$

In addition and to ensure that forces and torques are properly transferred between the two bodies, we subtract the constraint forces

$$\mathbf{f}^F = C_{\mathbf{c}^F}^T \lambda \qquad \mathbf{f}^B = C_{\mathbf{c}^B}^T \lambda \tag{13}$$

Table 2. **Properties of Constraint Derivatives.** The property pairs $(1, 4)$ and $(2, 3)$ are equivalent. Properties 1 and 2 are useful in the derivation of constraint torques for translational constraints, and properties 3 and 4 for derivation of torques for rotational constraints. Note that $\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a})$ and $(\mathbf{a} \times \mathbf{b})\lambda = (\mathbf{a}\lambda) \times \mathbf{b} = \mathbf{a} \times (\mathbf{b}\lambda)$.

	$C(\mathbf{q})$	$\frac{1}{2}\mathbf{G}(\mathbf{q})C_{\mathbf{q}}^T\lambda$
1	$\mathbf{a} \cdot (\mathbf{R}(\mathbf{q})\mathbf{b})$	$(\mathbf{R}(\mathbf{q})\mathbf{b}) \times (\mathbf{a}\lambda)$
2	$\mathbf{a} \cdot (\mathbf{R}(\mathbf{q})^T\mathbf{b})$	$\mathbf{b} \times (-\mathbf{R}(\mathbf{q})\mathbf{a}\lambda)$
3	$(\mathbf{R}(\mathbf{q})\mathbf{a}) \cdot \mathbf{b}$	$(\mathbf{R}(\mathbf{q})\mathbf{a} \times \mathbf{b})\,\lambda$
4	$\mathbf{a} \cdot (\mathbf{R}(\mathbf{q})\mathbf{b})$	$-(\mathbf{a} \times \mathbf{R}(\mathbf{q})\mathbf{b})\,\lambda$

from the equations **F3** and constraint torques

$$\boldsymbol{\tau}^F = \frac{1}{2}\mathbf{G}(\mathbf{q}^F)C_{\mathbf{q}^F}^T\lambda \qquad \boldsymbol{\tau}^B = \frac{1}{2}\mathbf{G}(\mathbf{q}^B)C_{\mathbf{q}^B}^T\lambda \tag{14}$$

from equations **F4** of the two bodies.

In the remainder of this section, we will introduce a single translational and a single rotational constraint that enable the formulation of all joint and actuator types with up to three translational and three rotational degrees of freedom. To derive analytical and interpretable expressions for the constraint forces and torques, we make use of the properties in Tab. 2. For the sake of illustration, we use a prismatic and a revolute joint (Figs. 2 and 3).

To formulate constraints, we assume the initial location of a joint, $\mathbf{x}$, as well as its axes, $\mathbf{A} = [\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z]$, to be defined in global coordinates (Figs. 2 and 3, Initialization). By setting the orientations of all bodies to the identity at time $t = 0$, the joint axes in local body coordinates equal the global axes, $\mathbf{A}_{\text{rb}}^F = \mathbf{A}_{\text{rb}}^B = \mathbf{A}$, and the joint locations in body coordinates are $\mathbf{x}_{\text{rb}}^F = \mathbf{x} - \mathbf{c}^F$ and $\mathbf{x}_{\text{rb}}^B = \mathbf{x} - \mathbf{c}^B$.

Translational Constraints. A prismatic joint with a translational degree of freedom along axis $\mathbf{a}_x$ restricts translations in the two orthogonal directions $\mathbf{a}_y$ and $\mathbf{a}_z$, while a revolute joint restricts translations along all three axes. A single constraint that enables the formulation of all joint types relies on the difference vector, $\mathbf{d}$, between the two global joint locations, transformed to coordinates

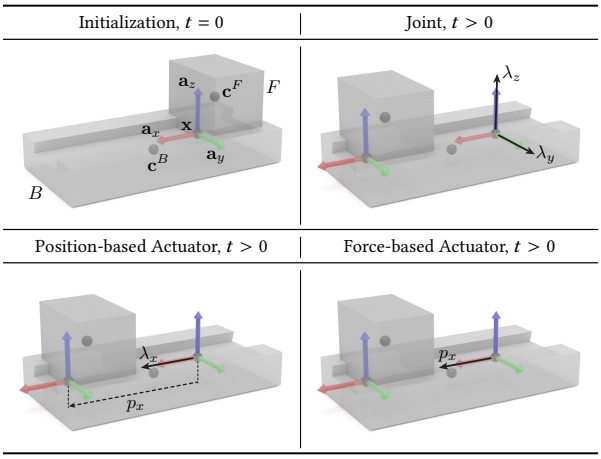

Fig. 2. **Prismatic Joint and Actuators.**

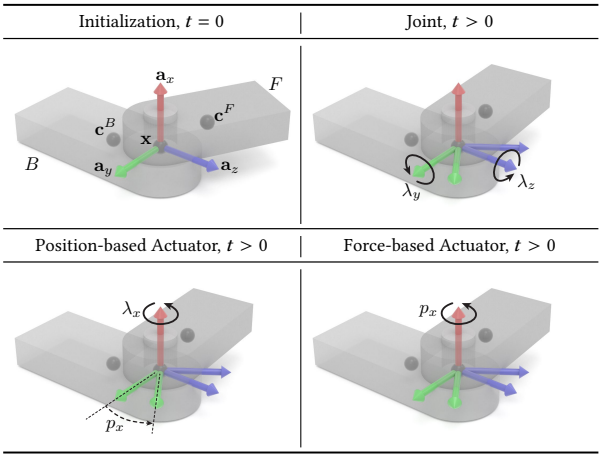

Fig. 3. **Revolute Joint and Actuators.**

of the base body, and restricts motion along the generic axis $\mathbf{a}$

$$C = \mathbf{a} \cdot (\mathbf{R}(\mathbf{q}^B)^T \mathbf{d}) \tag{15}$$

$$\mathbf{d} = \mathbf{R}(\mathbf{q}^F)\mathbf{x}^F_{\text{rb}} + \mathbf{c}^F - (\mathbf{R}(\mathbf{q}^B)\mathbf{x}^B_{\text{rb}} + \mathbf{c}^B). \tag{16}$$

If a translation is restricted along an axis, we expect forces to be transferred along this axis. Moreover, we expect constraint forces to be equal and opposite, not doing any work on the system. Derivations of analytical expressions for the two constraint forces confirm that this is the case

$$\mathbf{f}^F = \mathbf{R}(\mathbf{q}^B)\mathbf{a}\lambda \quad \mathbf{f}^B = -\mathbf{f}^F. \tag{17}$$

In addition, we observe that the Lagrange multiplier can be interpreted as force along $\mathbf{a}$ in joint coordinates on body B, explaining why we transform the difference vector to local coordinates of B.

Because the two forces act at the joint location on the follower, they induce two torques that are not equal and opposite due to the difference in the two moment arms

$$\boldsymbol{\tau}^F \overset{1}{=} (\mathbf{R}(\mathbf{q}^F)\mathbf{x}^F_{\text{rb}}) \times \mathbf{f}^F \quad \boldsymbol{\tau}^B \overset{2}{=} (\mathbf{R}(\mathbf{q}^F)\mathbf{x}^F_{\text{rb}} + \mathbf{c}^F - \mathbf{c}^B) \times \mathbf{f}^B. \tag{18}$$

In their derivation, we made use of properties 1 and 2 in Tab. 2.

For a prismatic joint with axis $\mathbf{a}_x$, we add two constraints with $\mathbf{a}$ set to $\mathbf{a}_y$ and $\mathbf{a}_z$, and for a revolute joint we add a constraint for every axis.

Rotational Constraints. To restrict rotational motion, it is common to ask an axis $\mathbf{a}$ on the follower to remain orthogonal to another axis $\mathbf{b}$ on the base with a dot-product constraint

$$C = (\mathbf{R}(\mathbf{q}^F)\mathbf{a}) \cdot (\mathbf{R}(\mathbf{q}^B)\mathbf{b}). \tag{19}$$

Because the constraint does not depend on the two centers of mass, only the two constraint torques are non-zero and are, as expected, equal and opposite

$$\boldsymbol{\tau}^F \overset{3}{=} (\mathbf{R}(\mathbf{q}^F)\mathbf{a} \times \mathbf{R}(\mathbf{q}^B)\mathbf{b})\lambda \qquad \boldsymbol{\tau}^B \overset{4}{=} -\boldsymbol{\tau}^F, \tag{20}$$

where we made use of properties 3 and 4 in Tab. 2. By taking a closer look at the analytical expression, we observe that the Lagrange multiplier can be interpreted as *torque about the axis that is perpendicular to both* $\mathbf{R}(\mathbf{q}^F)\mathbf{a}$ *and* $\mathbf{R}(\mathbf{q}^B)\mathbf{b}$. For consistency with our translational constraints, we choose the two axes and multiply the constraint with -1 if necessary so that λ represents a positive torque about the axis $\mathbf{R}(\mathbf{q}^B)(\mathbf{a} \times \mathbf{b})$ on the *base body*.

For a prismatic joint, we introduce constraints between all three pairs of axes, while we set the pair $(\mathbf{a}, \mathbf{b})$ to $(\mathbf{a}_x, \mathbf{a}_z)$ for a first and to $(\mathbf{a}_x, \mathbf{a}_y)$ for a second constraint for our revolute joint, multiplying the first one with -1.

Position-based Actuators. To turn a passive joint into a position-based actuator, we need to complement the set of passive constraints with active constraints along or about the "free" axes [Maloisel et al. 2023], controlling the degree of freedom with a position-based variable p. Once p is set, the relative motion between the two bodies is fully constrained and forces and torques are transferred in all directions.

To control the position along a generic axis $\mathbf{a}$ with p, we simply subtract p from our passive translational constraint

$$C = \mathbf{a} \cdot (\mathbf{R}(\mathbf{q}^B)^T \mathbf{d}) - p. \tag{21}$$

Due to the linear dependence on p, the constraint forces and torques are the same for an active and a passive version of this constraint.

With an active rotational constraint, we aim at controlling the rotation about the axis $\mathbf{a} \times \mathbf{b}$ by angle p. To do so, we can either rotate $\mathbf{a}$ on the follower

$$C = (\mathbf{R}(\mathbf{q}^F)\mathbf{R}[p, \mathbf{a} \times \mathbf{b}]^T \mathbf{a}) \cdot (\mathbf{R}(\mathbf{q}^B)\mathbf{b}) \tag{22}$$

or $\mathbf{b}$ on the base body

$$C = (\mathbf{R}(\mathbf{q}^F)\mathbf{a}) \cdot (\mathbf{R}(\mathbf{q}^B)\mathbf{R}[p, \mathbf{a} \times \mathbf{b}]\mathbf{b}). \tag{23}$$

Because we want p to represent a positive rotation about $\mathbf{a} \times \mathbf{b}$ *on the base body*, we transpose the rotation $\mathbf{R}[p, \mathbf{a} \times \mathbf{b}]$ in the first constraint. The variant we use depends on the joint type. Because a universal joint is technically a three body joint with a star-shaped mid-body, we need both variants to turn it into an actuator.

In contrast to active translational constraints, the constraint torques depend on parameter p and can easily be derived with properties 3 and 4 in Tab. 2.

To turn our prismatic joint into an actuator, we add a translational constraint with $\mathbf{a} = \mathbf{a}_x$ and $p = p_x$. Similarly, we add a rotational constraint with $(\mathbf{a}, \mathbf{b}) = (\mathbf{a}_y, \mathbf{a}_z)$ and $p = p_x$ for a revolute actuator, where p_x represents a rotation about $\mathbf{a}_x = \mathbf{a}_y \times \mathbf{a}_z$.

Force-based Actuators. To turn a passive joint into a force-based actuator, we add constraint forces and torques with the Lagrange multiplier replaced with a parameter p, instead of adding the corresponding constraint. For translations, we simply substitute p for λ in Eqs. 17 and 18. To actively control rotations with a torque p about axis $\mathbf{a} \times \mathbf{b}$, we use the two constraint torques

$$\boldsymbol{\tau}_F = \mathbf{R}(\mathbf{q}^B)(\mathbf{a} \times \mathbf{b})p \qquad \boldsymbol{\tau}^B \overset{4}{=} -\boldsymbol{\tau}^F, \tag{24}$$

For our prismatic example, we add forces and torques with $\mathbf{a}$ set to $\mathbf{a}_x$ and p_x representing the actuation force along $\mathbf{a}_x$. For our revolute joint, we set the pair $(\mathbf{a}, \mathbf{b})$ to $(\mathbf{a}_y, \mathbf{a}_z)$, with p_x representing an actuation torque about the axis $\mathbf{a}_x = \mathbf{a}_y \times \mathbf{a}_z$.

Table 3. **Constraints for Common Joints**

Name	Translation Constraints	Rotation Constraints
Fixed	fixed	fixed
Prismatic	1 DoF ($\mathbf{a}_x$)	fixed
Revolute	fixed	1 Dof ($\mathbf{a}_x$)
Rectangular	2 DoF ($\mathbf{a}_x, \mathbf{a}_y$)	fixed
Cylindrical	1 DoF ($\mathbf{a}_x$)	1 DoF ($\mathbf{a}_x$)
Pin in Slot	1 DoF ($\mathbf{a}_x$)	1 DoF ($\mathbf{a}_y$)
Universal	fixed	2 DoF($\mathbf{a}_x, \mathbf{a}_y$)
Cartesian	free	fixed
Planar	2 DoF ($\mathbf{a}_x, \mathbf{a}_y$)	1 DoF ($\mathbf{a}_z$)
Spherical	fixed	free
Slot	1 DoF ($\mathbf{a}_x$)	free
Free	free	free

Combining Constraints. By adding a constraint for each restricted translation or rotation, we can model the degrees of freedom of many mechanical joints (see Tab. 3), turning them either into position- or force-based actuators if needed. We refer the reader to Tab. 7 in our appendix for implementation-ready formulas for the translational and rotational constraints that we list in Tab. 3. While we haven't explored it, we can use our two basic constraints to partially actuate a mechanical joint, or to mix position- and force-based actuation along or about axes.

Unary Constraints. By setting the center of mass of the base body to zero and its orientation to the identity, we can turn each constraint into one that constrains a follower body to the world. We use such constraints for our spinning top example, where we constrain the tip to stay at the same location with a unary constraint to the world.

Joint Flips. For joints and actuators with more than one orthogonality constraint, extraneous solutions with flipped axes exist. Solvers can converge to a "flipped" solution if large steps are taken. For instance, for a revolute joint, both solutions $\mathbf{R}(\mathbf{q}^F)\mathbf{a}_x = \pm\mathbf{R}(\mathbf{q}^B)\mathbf{a}_x$ satisfy the two constraints.

To prevent flips, we formulate what we refer to as *consistency constraints* (see const. **C** columns in Tab. 7), which evaluate to 1 for the correct solution, and 0 or -1 for extraneous solutions. For example, for a revolute joint with axis $\mathbf{a}_x$, we introduce one consistency constraint by setting $\mathbf{a} = \mathbf{b} = \mathbf{a}_x$ in Eq. 19. When solving the equations of motion, during line search, we backtrack if any consistency constraint is below a positive threshold.

5 Implicitly Integrating Constrained Dynamics

Analogously to the two-body case (Eq. 12), we form the continuous equations of motion for a multibody system, $\mathbf{F}(\dot{\mathbf{y}}, \mathbf{y}, t) = \mathbf{0}$, by first adding the equations of motion for the individual bodies, followed by adding constraints $\mathbf{C}$. For every constraint, we add a Lagrange multiplier to $\mathbf{y}$ and subtract corresponding constraint forces and torques from the respective body equations.

$\mathbf{F} = \mathbf{0}$ constitutes a system of DAEs of index 3 and Hessenberg form [Brenan et al. 1995], which we integrate forward in time with a fully implicit scheme, so that kinematic constraints can be enforced up to a numerical tolerance. More specifically, we pick an unconditionally stable scheme, which is valuable when solving optimization problems with a simulator in the loop.

Compared to ODEs, the integration of DAEs, even with implicit schemes, poses additional challenges. Two families of implicit methods have seen common use, and have proven convergence results [Brenan et al. 1995].

5.1 Backward Differentiation Formula (BDF)

The first and simplest applies the Backward Differentiation Formula (BDF) with a *fixed* time step Δt. Given a linear combination of previous states, $\mathbf{y}_p := \alpha_1\mathbf{y}^{(k-1)} + \ldots + \alpha_m\mathbf{y}^{(k-m)}$, the next state $\mathbf{y}_n := \mathbf{y}^{(k)}$ at time $t_n := k\Delta t$ is computed as the solution of the system

$$\mathbf{F}\left(\frac{\mathbf{y}_n - \mathbf{y}_p}{\beta\Delta t}, \mathbf{y}_n, t_n\right) = \mathbf{0}, \tag{25}$$

which is solved with an iterative Newton-type solver, as detailed in Sec. 6. Coefficients $\alpha_1, \ldots, \alpha_m, \beta$ are tabulated for a BDF of a particular order m.

One advantage of discretizing with a BDF formula is that the cost of solving Eq. 25 is largely independent of the order m, as it only affects the precomputation of $\mathbf{y}_p$. Note, however, that formulas of order 3 or higher are not unconditionally stable. The usage of a *fixed* time step Δt is a requirement for convergence [Brenan et al. 1995], preventing the use of adaptive time-stepping schemes, or requiring additional correction [Arévalo and Lötstedt 1995]. In our experiments, we therefore use BDF1 ($\beta = 1, \alpha_1 = 1$) and BDF2 ($\beta = \frac{2}{3}, \alpha_1 = \frac{4}{3}, \alpha_2 = -\frac{1}{3}$) and find the latter a good trade-off between computational cost, accuracy, and stability. The fixed time step also facilitates downstream processing of the simulation results, including differentiating through a simulation sequence for sensitivity analysis or optimization.

5.2 Implicit Runge-Kutta Methods

If a higher-order method or adaptive time-stepping are desired, an implicit s-stage Runge-Kutta method [Brenan et al. 1995] can be applied, solving

$$\begin{cases} \mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i \\ \mathbf{F}\left(\mathbf{k}_i, \mathbf{y}^{(k-1)} + \Delta t \sum_{j=1}^{s} a_{ij}\mathbf{k}_j, t_{k-1} + c_i\Delta t\right) = \mathbf{0}, i = 1 \ldots s \end{cases} \tag{26}$$

for $\mathbf{k}_1, \ldots, \mathbf{k}_s$ and $\mathbf{y}^{(k)}$ at $t_k = t_{k-1} + \Delta t$, given the previous state $\mathbf{y}^{(k-1)}$ at t_{k-1}. Coefficients a_{ij}, b_i, and c_i are scheme-specific.

Assuming a *diagonally implicit Runge-Kutta* (DIRK) method, that is, $a_{ij} = 0$ for $i < j$, intermediate solutions $\mathbf{k}_i$ can be computed sequentially, applying our algebraic solver s times to solve

$$\mathbf{F}\left(\frac{\mathbf{y}_i - \left(\mathbf{y}^{(k-1)} + \Delta t \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j\right)}{a_{ii}\Delta t}, \mathbf{y}_i, t_{k-1} + c_i\Delta t_i\right) = \mathbf{0} \tag{27}$$

for the intermediate state $\mathbf{y}_i$, from which we then recover $\mathbf{k}_i$ as

$$\mathbf{k_i} = \frac{\mathbf{y}_i - \left(\mathbf{y}^{(k-1)} + \Delta t \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j\right)}{a_{ii}\Delta t}. \tag{28}$$

Note that the discretized equations Eq. 27 and Eq. 25 are the same if we set the unknown state $\mathbf{y}_n$ in Eq. 27 to $\mathbf{y}_i$ and the aggregate previous state $\mathbf{y}_p$ to $\mathbf{y}^{(k-1)} + \Delta t \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j$. We can therefore use the same solution strategy to solve them.

DIRK methods do not generally fulfill algebraic constraints at the new state $\mathbf{y}^{(k)}$, unless the method is *stiffly-accurate*, which requires that $c_s = 1$ and $a_{si} = b_i$ for $i = 1, \ldots, s$.

For systems where energy conservation is of secondary importance, using BDF2 integration with a smaller time step is often preferable over a DIRK method, as the higher order of convergence does not offset the additional computational cost. In addition, it has been observed that implicit Runge-Kutta methods can yield a strictly lower order of convergence when used for DAE integration as opposed to ODE integration. In our results, the use of Runge-Kutta methods is therefore limited to validations, and to improve the start-up of BDF2 by integrating the first step using a *stiffly-accurate two-stage SDIRK* method [Nishikawa 2019] (see Appendix A for scheme coefficients).

6 Solving the Algebraic Equations

To step our simulator in time, we solve the discretized equations iteratively up to a numerical tolerance. Our solver largely follows a Newton scheme, refining a solution in steps by evaluating the equations $\mathbf{F}$ and their Jacobian $\mathbf{F}_{\mathbf{y}_n}$ at the current iterate $\mathbf{y}_n$, and computing an update $\Delta\mathbf{y}_n$ by solving

$$\mathbf{F}_{\mathbf{y}_n}\,\Delta\mathbf{y}_n = -\mathbf{F}. \tag{29}$$

For robustness, we perform a line search to select the next iterate along the direction $\Delta\mathbf{y}_n$, ensuring sufficient progress and preventing violations in consistency constraints (see Sec. 4, Tab. 7).

Note that there are always as many equations as unknowns by construction: For every rigid body, we add 14 equations in 14 unknowns, and for every constraint, we add as many constraint equations as unknown Lagrange multipliers. For most systems, however, the Jacobian is poorly conditioned or rank-deficient. The latter occurs whenever kinematic constraints are redundant.

In the remainder of this section, we present our adaptations to this general iterative procedure, so that the discretized equations can be solved accurately for all classes of systems, including under-, fully- and overactuated systems, and/or systems with redundant kinematic constraints.

We use $\mathbf{s}_n, \dot{\mathbf{s}}_n, \boldsymbol{\lambda}_n$ to denote the segments of $\mathbf{y}_n$ corresponding to the collection of rigid body poses, velocities, and Lagrange multipliers, respectively.

6.1 System Conditioning for Small Time Steps

Even in the case of a full-rank Jacobian, the condition number of $\mathbf{F}_{\mathbf{y}_n}$ grows quickly, with $O(1/\Delta t^4)$, when the time step is decreased. This can be remedied by scaling the rows and columns of the Jacobian [Bottasso et al. 2008]. We therefore define scaled versions of the variables, $\mathbf{y}'_n := \mathbf{D}_\mathbf{y}\mathbf{y}_n$, and of the equations, $\mathbf{F}' := \mathbf{D}_\mathbf{F}\mathbf{F}$, where $\mathbf{D}_\mathbf{y}, \mathbf{D}_\mathbf{F}$ are diagonal matrices. The equivalent Newton step in the scaled variables is then

$$\mathbf{D}_\mathbf{F}\,\mathbf{F}_{\mathbf{y}_n}\mathbf{D}_\mathbf{y}^{-1}\,\Delta\mathbf{y}'_n = -\mathbf{F}'. \tag{30}$$

By an appropriate choice of the scaling, the dominant terms in the condition number of the scaled Jacobian $\mathbf{D}_\mathbf{F}\,\mathbf{F}_{\mathbf{y}_n}\mathbf{D}_\mathbf{y}^{-1}$ can be made independent of Δt. We set entries of $\mathbf{D}_\mathbf{y}, \mathbf{D}_\mathbf{F}$ to

- 1 if they correspond to position or orientation variables, or to unit-length or kinematic constraint equations.
- Δt if they correspond to linear and angular velocity variables, or to velocity equations.
- Δt^2 if they correspond to Lagrange multipliers, or to equilibrium equations (since forces scale like accelerations).

The next iterate in the original variables is then recovered after the Newton step by applying the inverse scaling. In all of the following, we work with scaled variables, equations and Jacobians, but drop the additional notation for simplicity.

6.2 Overconstrained Systems

Systems with kinematic loops commonly display redundancy in their constraints — for example a regular planar four-bar linkage with all-revolute joints will have 3 constraints too many.

Redundant constraints manifest as redundant Lagrange multipliers, resulting in a subspace of solutions in terms of generalized forces transmitted at joints or actuators. Picture a door pivoting about two hinges on the *same* axis: The weight of the door could be carried equally by both hinges, or a single hinge, or any intermediate distribution.

Redundant Constraints Elimination. When redundant kinematic constraints are present, the Jacobian, $\mathbf{F}_{\mathbf{y}_n}$, becomes rank-deficient. The system Eq. 29 becomes both overconstrained (due to redundant constraints) and underconstrained (due to the subspace in the Lagrange multipliers), and can therefore not be directly solved for a least-squares (via ${\mathbf{F}_{\mathbf{y}_n}}^T\mathbf{F}_{\mathbf{y}_n}$) or least-norm (via $\mathbf{F}_{\mathbf{y}_n}{\mathbf{F}_{\mathbf{y}_n}}^T$) solution. An expensive rank-revealing solver, e.g. QR or singular value decomposition, would be required instead.

To circumvent this, we utilize the technique introduced in [Maloisel et al. 2023] to pick a subset of kinematic constraints that can be eliminated while preserving the kinematics of the system. The key steps are reproduced in Appendix B for completeness. By also eliminating the corresponding multipliers, we obtain a reduced, full-rank system, which permits a more efficient solver — here we use a sparse LU decomposition. A more expensive singular value decomposition is required only for the redundancy analysis as a precomputation.

For some systems, the set of removable constraints may vary over time, in particular close to a singularity. We remedy this by checking, at the end of each simulation step of the reduced system, that the eliminated constraints are still satisfied within a tolerance. If they are not, a new redundancy analysis is performed at the current state. In practice, this occurs rarely and does not significantly affect performance. For increased robustness, the number of constraints to eliminate is stored after the first analysis, as numerical rank computations may be unreliable near singular configurations.

Overactuated Systems. Overactuated systems contain more actuators than the degrees of freedom of the equivalent system with all passive joints. This can be used, for example, to distribute a load across multiple actuators. As a consequence, when performing

position control, actuators cannot be stepped individually without violating the constraints — rather, they must move in concert. To handle this case, we follow [Maloisel et al. 2023], essentially treating redundant actuators similarly to redundant constraints.

In the case of redundant force-based actuators, no special treatment is required: the force inputs for these actuators only appear on the right-hand side of the system of Eq. 29, and thus do not affect its rank. When analyzing kinematic constraints, these actuators are treated like their corresponding passive joints.

Weighted Least-Norm Multipliers. As discussed, overconstrained systems exhibit a redundancy in their Lagrange multipliers. Using a reduced system removes the redundancy by allowing only a subset of multipliers to be non-zero, but results in an arbitrary solution, with discontinuities if the set of reduced constraints needs to be updated. We note that only the equilibrium equations **E**, corresponding to rigid body equations **F3** − **F4** with added constraint forces, depend, linearly, on multipliers $\boldsymbol{\lambda}_n$; thus the more general solution subspace is characterized by

$$\mathbf{E}_{\boldsymbol{\lambda}_n}\,\boldsymbol{\lambda}_n = -\mathbf{E} \tag{31}$$

where $\mathbf{E}, \mathbf{E}_{\boldsymbol{\lambda}_n}$ are evaluated for $\boldsymbol{\lambda}_n = 0$ and $\mathbf{s}_n, \dot{\mathbf{s}}_n$ set as per the solution for the reduced system. To select a more meaningful and time-consistent solution within this subspace, we allow the user to specify a fixed weight $w_i > 0$ for each force or torque component λ_i. As a post-processing step, we then solve for $\boldsymbol{\lambda}_n$ by minimizing the weighted norm $\frac{1}{2}\sum_i w_i\lambda_i^2$, which admits a closed form

$$\boldsymbol{\lambda}_n = -\mathbf{W}^{-1}{\mathbf{E}_{\boldsymbol{\lambda}_n}}^T\left(\mathbf{E}_{\boldsymbol{\lambda}_n}\,\mathbf{W}^{-1}{\mathbf{E}_{\boldsymbol{\lambda}_n}}^T\right)^+\mathbf{E} \tag{32}$$

where $\mathbf{W} := \mathrm{diag}(\dots w_i \dots)$, and the + exponent denotes the Moore-Penrose pseudo-inverse. For systems without unconstrained degrees of freedom, this becomes a regular inverse, and the system can be solved efficiently with a sparse Cholesky (LLT) decomposition. Otherwise, a sparse QR solver can be employed. If only trajectories of the system (i.e. values of $\mathbf{s}$ and $\dot{\mathbf{s}}$) are of interest to the user, this step can also be omitted, keeping the reduced multipliers only, as future time steps are not influenced by Lagrange multipliers.

Since the Lagrange multipliers in our formulation have a simple interpretation, the user can intuitively set weights w_i. For instance, if minimal actuator torques are desired, one might set $w_i = 10^5$ for multipliers corresponding to position-controlled actuator torques, and $w_i = 1$ for other forces and torques. Alternatively, weights could be chosen based on the relative compliance (inverse stiffness) of each joint to model the load distribution effect that compliance has in real mechanisms.

6.3 Fully-Constrained Systems

The solver strategy described thus far is applicable for the general case of systems with potential unconstrained degrees of freedom. However, many real-world mechanisms are fully-constrained systems, that is, for which the pose of all bodies is fully determined by the kinematic constraints $\mathbf{C}(\mathbf{s})$. Note that this implies only position-based actuators.

For such systems, a more efficient two-step solver strategy can be applied, exploiting the fact that position and force variables can be decoupled. Specifically, we first solve forward kinematics, computing $\mathbf{s}_n$ in an iterative Gauss-Newton process with update step

$${\mathbf{K}_{\mathbf{s}_n}}^T\mathbf{K}_{\mathbf{s}_n}\,\Delta\mathbf{s}_n = -{\mathbf{K}_{\mathbf{s}_n}}^T\mathbf{K} \tag{33}$$

where $\mathbf{K}(\mathbf{s}_n)$ concatenates unit-quaternion constraints **F5** and kinematic constraints **C**. Note that in the presence of redundant kinematic constraints, the kinematic Jacobian, $\mathbf{K}_{\mathbf{s}_n}$, as opposed to the full Jacobian $\mathbf{F}_{\mathbf{y}_n}$, is rank-deficient along its rows only and therefore ${\mathbf{K}_{\mathbf{s}_n}}^T\mathbf{K}_{\mathbf{s}_n}$ is always full-rank for fully-constrained systems, so the redundancy analysis step can be omitted. The scaling process of Sec. 6.1 is also not required, as Δt does not appear. The system (Eq. 33) can be solved efficiently with a sparse Cholesky (LLT) decomposition.

After solving the forward kinematics, velocities can be evaluated directly as per equations **F1** − **F2**. Finally, we can compute the Lagrange multipliers, $\boldsymbol{\lambda}_n$, in a single system solve as per Eq. 32, or more simply with $\boldsymbol{\lambda}_n = -{\mathbf{E}_{\boldsymbol{\lambda}_n}}^{-1}\mathbf{E}$ if no redundancy is present.

Table 4. **Classification of Examples.**

	No redundant constraints	**Redundant constraints**	
		Not overactuated	**Overactuated**
Fully actuated	-	Iron Man	Gazelle
Passive DoFs	T-Handle Spinning Top Pendulum	Hoberman Sphere	Satellite

7 Computing Derivatives

Differentiability is an important property for a simulator, with use in applications such as sensitivity analysis, optimal control, design optimization or system identification. Our choice of an implicit integrator simplifies the computation of simulation derivatives with respect to arbitrary parameters, since no stabilization, or adaptive time-stepping, that would complicate differentiation, are required. Because we rely on additive instead of multiplicative quaternion updates, symbolic differentiation is directly applicable to all building blocks.

We can compute derivatives for all cases discussed herein, including overactuation and redundant constraint subspaces. See Appendix C for details, including how we compute derivatives using the implicit function theorem, and the use of the adjoint method for evaluation speed. Notably, we expand on the standard adjoint method to handle the case of weighted least-norm Lagrange multipliers.

8 Results

We validate our approach on a diverse set of examples. To illustrate the generality of our approach, it is useful to categorize the examples as shown in Tab. 4. The most straightforward case is a fully actuated system with no redundancy in the constraints. However, in our experience, even simple real-world examples will tend to fall into one of the other categories. Our method handles all of these categories "out of the box", without requiring any special considerations. Note in particular that systems can simultaneously be overactuated and have unconstrained degrees of freedom. While our method is neither energy- nor momentum-preserving, we show that the energy and

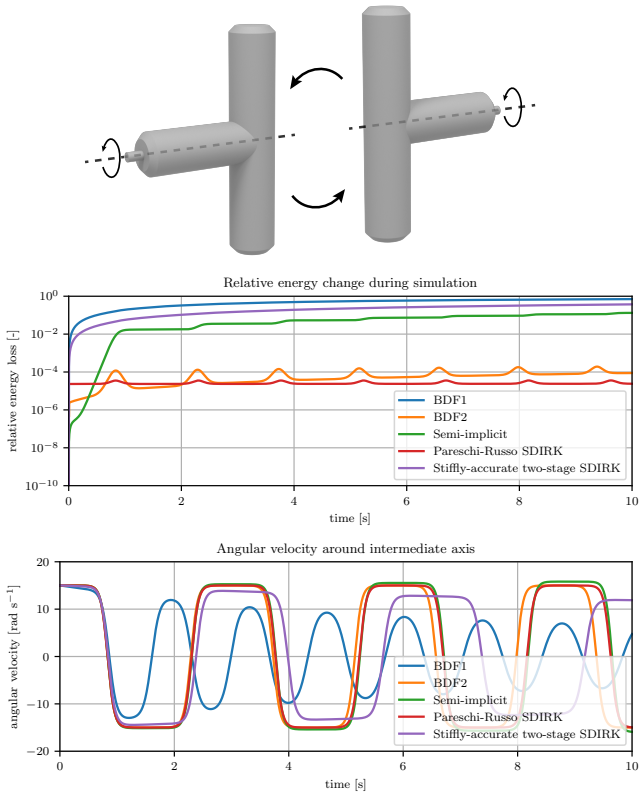

Fig. 4. **T-Handle.** Evaluation of velocity and energy preservation of different integration schemes for a rigid body spinning freely in space, periodically passing through a rotational instability due to the intermediate axis theorem.

momentum loss is reasonably small and decreases with decreasing time step, confirming the validity of our approach.

Key specifications for all our examples are summarized in Tab. 5. Unless explicitly noted, we use a BDF2 integration scheme. For the first time step, where a single previous state is known, we use a stiffly-accurate second-order implicit Runge-Kutta step, as is common practice [Nishikawa 2019]; see also Appendix A. The semi-implicit scheme, which is used as a baseline in several examples, is described in Appendix D.

Our implementation is C++-based, relying on Eigen and MKL Pardiso for linear algebra. For our line search, we use Armijo's condition. Solver tolerances are set to 10^{-9} or 10^{-10} for all examples. To set least-norm multiplier weights, unless specified otherwise, we use the first strategy described at the end of Sec. 6.2, with a large weight for actuator torques.

T-Handle. We explore the behavior of different time integration schemes on a single rigid body using a *T-handle* spinning in zero gravity. The simulation is initialized with a non-zero initial angular velocity around the second principal axis in order to trigger the rotational instability described by the *intermediate axis theorem*, resulting in a continuous rotation where the object will flip in regular intervals (see accompanying video).

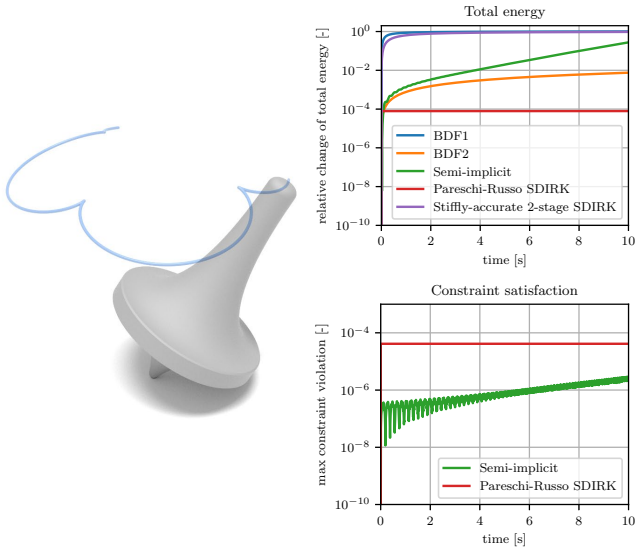

Fig. 5. **Spinning Top.** Left: our simulated spinning top exhibits a characteristic precession and nutation pattern. Top right: the relative energy change varies significantly between different integration methods. Bottom right: the semi-implicit and Pareschi-Russo SDIRK integration methods do not fulfill the mechanical constraints exactly (all others have constraint norms $< 10^{-10}$).

We simulate the behavior of the T-handle using BDF1 and BDF2, semi-implicit integration, as well as two different singly diagonally implicit Runge-Kutta (SDIRK) methods (an L-stable *two-stage Pareschi-Russo SDIRK* [Pareschi and Russo 2005] and an L-stable *stiffly-accurate two-stage SDIRK* scheme [Alexander 1977], see Appendix A for coefficients).

The simulations use a time step of $\Delta t = 1$ms and are initialized with an angular velocity of $15\,\mathrm{rad\,s^{-1}}$ around the second (intermediate) principal axis of the body and initial angular velocities of $0.1\,\mathrm{rad\,s^{-1}}$ around the first and third principal axes to provide an initial disturbance.

Fig. 4 shows the evolution of energy and angular velocity for the different time integration methods. The stiffly-accurate two-stage SDIRK method preserves energy better than the BDF1 method, but significantly worse than the BDF2 method, while the Pareschi-Russo SDIRK method, after an initial energy loss during start-up, outperforms the other implicit methods.

Spinning Top. We validate the basic constrained rotational dynamics of our simulator on a spinning top with a fixed base. The spinning top is a single, rotationally symmetric body whose motion is constrained by a spherical joint connecting the tip of the body to world, thus fixing the translation at this point. When tilted at an angle and given a sufficiently high initial angular velocity, and in the absence of friction, such a spinning top will follow a characteristic precession and nutation pattern (Fig. 5).

Since the spinning top is a passive system without external influence besides gravity, this example allows us to investigate the

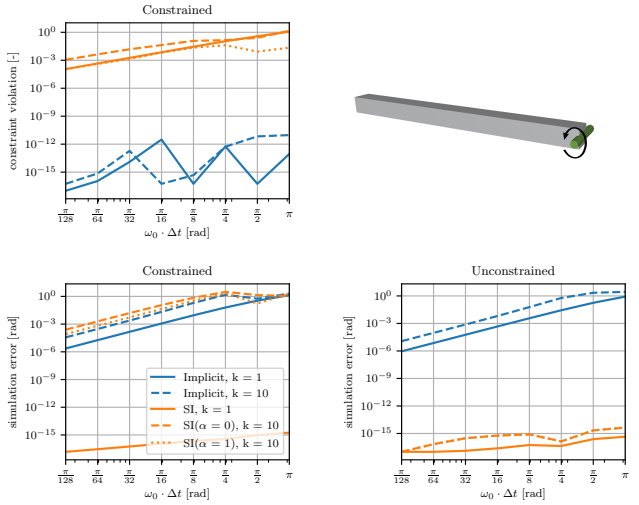

Fig. 6. **Pendulum.** We study the effect of large angular velocities with a freely-rotating pendulum (top right). $\Delta t = 0.01$ s for all experiments. Our implicit scheme ensures constraint satisfaction, while a semi-implicit (SI) scheme does not.

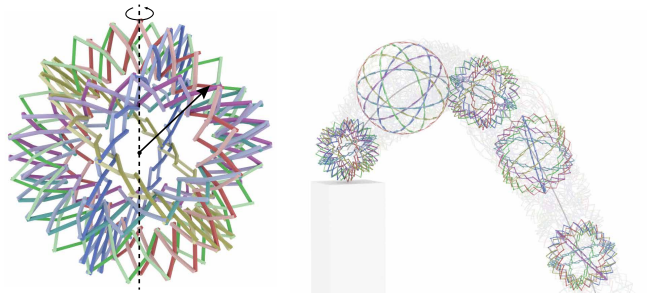

Fig. 7. **Hoberman Sphere.** Left: we simulate a spinning Hoberman sphere thrown into the air. Right: the spin of the sphere forces it into an oscillating pattern of expansion and contraction. See video for the full sequence.

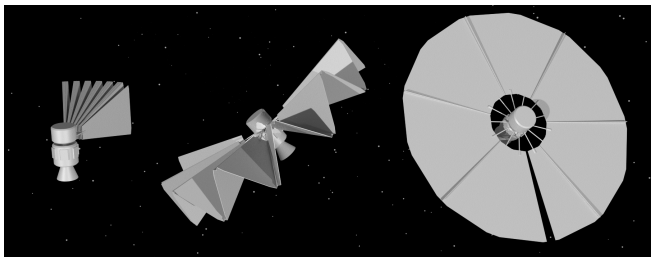

Fig. 8. **Satellite.** We simulate the overactuated deployment mechanism of a solar panel on a satellite floating in space.

energy preservation properties of our method for constrained systems and compare to other integration schemes, see Fig. 5: With a BDF2 time integration scheme we achieve a reasonably small loss of energy throughout the simulation and ensure constraint satisfaction. While Pareschi-Russo SDIRK performs better in terms of energy preservation, the constraint satisfaction is poor.

Pendulum. To evaluate the performance of our method for large angular velocities, we consider a gravity-free freely-rotating pendulum where we prescribe an initial velocity and then simulate for a few time steps. The ground truth solution should maintain this velocity and satisfy the joint constraint.

See Fig. 6. The top and bottom left plots show the constraint violation and the simulation error of the pendulum angle against the angular velocity relative to the time step, i.e., how far a body rotates in a single step. We show the result after 1 and 10 time steps, and for our implicit integration scheme as well as a semi-implicit scheme with and without Baumgarte stabilization. While the semi-implicit scheme has virtually zero simulation error at the first time step, the error grows over time due to the effect of the constraint. However, constraint satisfaction with the semi-implicit scheme is poor.

Our implicit scheme, in contrast, ensures constraint satisfaction for any initial velocity, with comparable simulation error to the semi-implicit scheme after the first time step.

We repeat the experiment on an unconstrained pendulum (i.e., a rigid body floating freely in space), see bottom right plot. In this case, the semi-implicit scheme is near-perfect, as one would expect, while the performance of our implicit scheme remains as before.

Hoberman Sphere. Next, we consider a *Hoberman sphere* — a combination of radial scissor linkages [Patel and Ananthasuresh 2007] forming a ball with a single degree of freedom — being thrown into the air with a spin. The spin causes the sphere to expand, and while a real mechanism would run into collisions, our idealized mechanism passes through the kinematic singularity at full expansion due to its momentum, contracting until the centripetal forces again cause the cycle to repeat. The system is fully passive and highly overconstrained, only influenced by its initial condition and gravitational forces. The presence of a kinematic singularity, even regularized by the system dynamics, is a stress test for our constraint elimination scheme. We handle this robustly by continuously checking the validity of our constraint elimination and recomputing if necessary.

The semi-implicit approach struggles in this setting and does not simulate the expansion and contraction: imperfect constraint satisfaction causes misalignment of the many parallel axes, creating additional internal forces which effectively lock the internal degree of freedom of the mechanism. The global trajectory does remain consistent though.

We refer to the accompanying video for a visualization of the full simulation.

Satellite. Here, we simulate the deployment sequence of a solar panel on a satellite in space, Fig. 8. The satellite mechanism is overactuated, with 7 actuators working in concert in order to minimize internal loads in the structure. This example is both overconstrained and has passive degrees of freedom, as it is flying through space, a combination of properties that our method handles natively.

Iron Man. We simulate the rigid body dynamics of a complex Audio-Animatronics® Iron Man figure executing a motion. This is a complex example due to the size of the problem (see Tab. 5), the

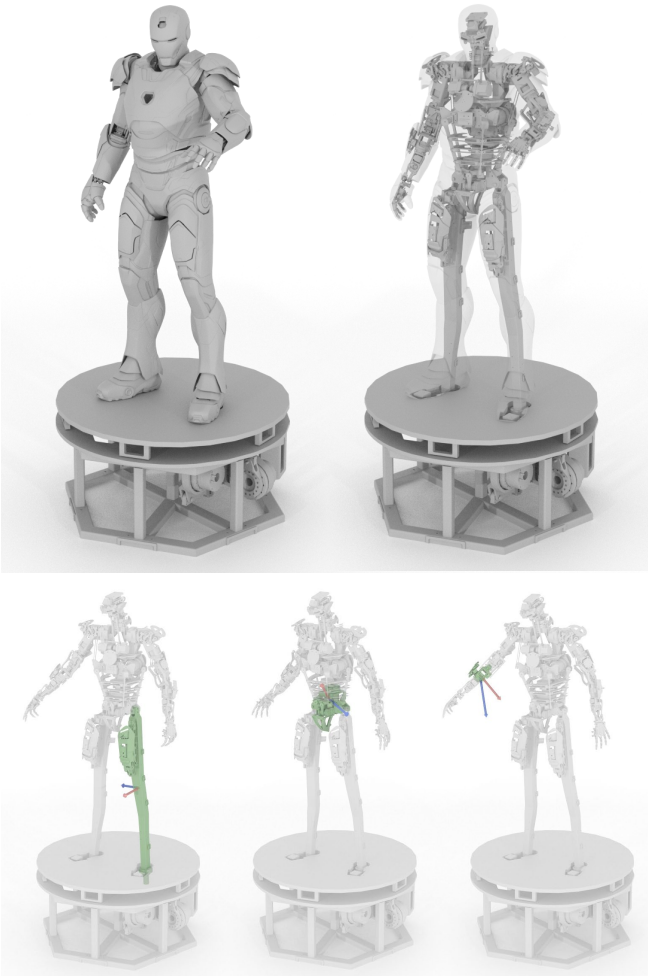

Fig. 9. **Iron Man.** Top: we simulate an Audio-Animatronics® figure with a complex internal structure and intricate mechanisms. Bottom: forces and torques acting on single rigid bodies during the motion. © MARVEL, Disneyland Paris

large range of rigid body masses (0.3 g to 49.2 kg), and the kinematic loops — in particular in the hands. This example is fully actuated and overconstrained, and the base is grounded so there are no passive system dynamics.

Fig. 9 shows illustrative snapshots of rigid body forces and torques at instances in time; see also the accompanying video.

Gazelle. We consider for this demonstrator an Audio-Animatronics® character with a fixed base, whose legs form a kinematic loop with the ground. Several linkages form smaller loops within this large loop, resulting in a complex kinematic structure (7 loops in total). Furthermore, the legs are overactuated: 8 actuators in total control the 6 degrees of freedom of the pelvis. This provides benefits with respect to the distribution of efforts and the regularization of otherwise singular configurations. However, it complicates the simulation, as actuators cannot be considered in isolation, since

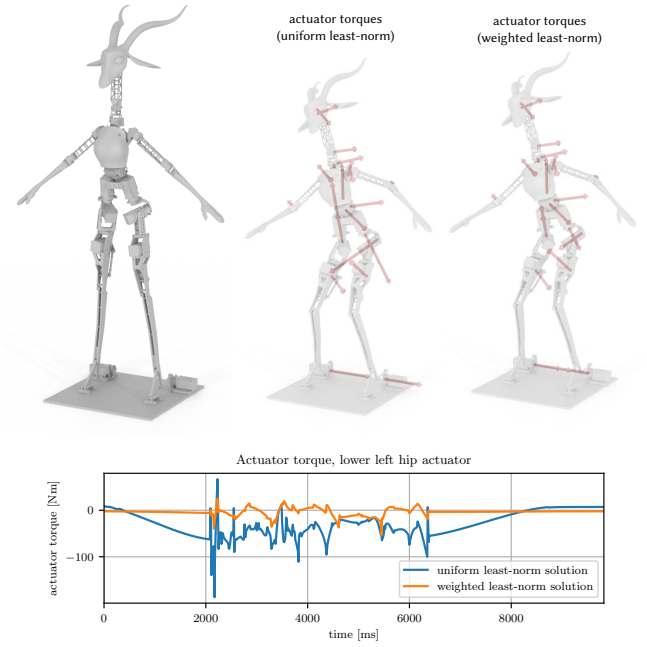

Fig. 10. **Gazelle: Overactuation Subspace** Top: we simulate a dancing motion on this Audio-Animatronics® figure. The overactuation in the leg mechanism allows us to choose between different weights for the least-norm solution (center; right). Bottom: actuator torques for an individual hip actuator. For this actuator, the uniform least-norm solution has significantly higher torques than the weighted least-norm solution.

arbitrary motion inputs are now generally infeasible, and some kinematic constraints become redundant. Our simulator natively handles this situation.

We simulate a dancing animation on this character with position-controlled actuators, using an inverse kinematics tool [Schumacher et al. 2021] for motion retargeting to ensure that the set of actuator positions are always feasible. Due to the redundancy in the constraints, even though there is a single valid *kinematic* trajectory, there are several solutions to the *dynamics* problem in terms of forces and torques at the joints. Using the strategy exposed in Sec. 6.2, we can pick a solution prioritizing actuator torque reduction. In comparison, using a naive least-norm solution results in 63% higher root-mean-square torques, averaged over the 8 actuators in the legs. See Fig. 10.

To validate that both actuator torque sequences result in the same kinematic motion, we recreate the simulation with force-controlled actuators, using the torque sequences computed by the first simulation as feedforward input. For stability, we add feedback torques provided by a simple PD controller, computed based on the reference actuator positions. We were able to validate that the resulting motion matched the reference motion, with average and peak actuator position errors of 0.03° and 1.44°, respectively.

For the same motion we evaluate the semi-implicit scheme for different stabilization parameters and at two step sizes (Fig. 11, left). The semi-implicit scheme shows significantly higher constraint

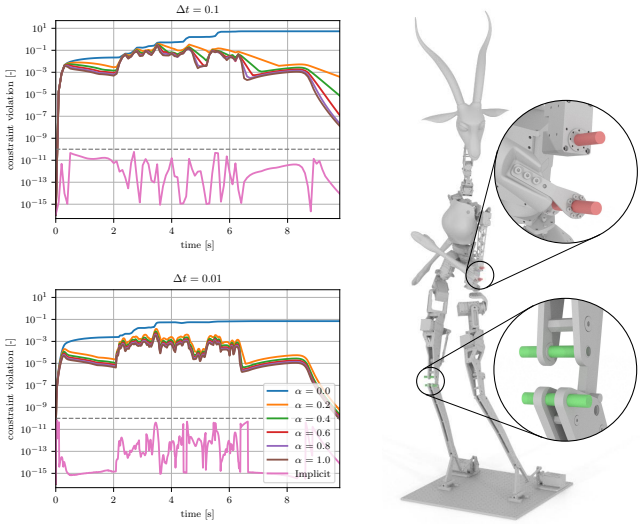

Fig. 11. **Gazelle: Effect of Step Size.** Left: we simulate the gazelle dancing motion at two step sizes with the semi-implicit scheme with different stabilization parameters and compare the result to the proposed implicit scheme. The reported violation is the maximum absolute value of the constraint vector. For the implicit scheme, the tolerance is set to 10^{-10}. Right: the simulation state at $t = 4.7$ s for the semi-implicit scheme with $\Delta t = 0.1, \alpha = 1.0$. The constraint violation is clearly visible for the position-based actuator at the elbow and the passive degree of freedom in the knee.

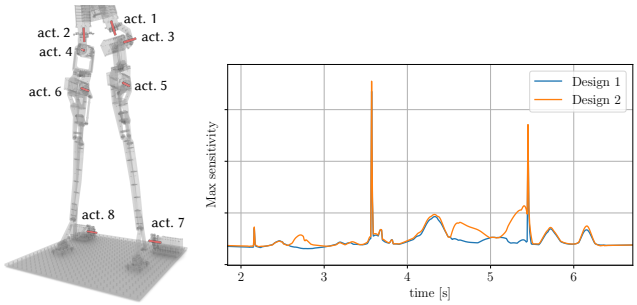

Fig. 12. **Sensitivity Analysis on the Gazelle Dancing Motion.** Plot of the worst-case sensitivity of rigid body poses to lower-body actuator torque inputs (i.e. maximal absolute coefficient of the sensitivity matrix), for two possible alternative designs of the Gazelle character. In design 1, actuators 1 and 8 are replaced with passive joints; in design 2, actuators 5 and 6. Design 2 features large sensitivity peaks at several points in the animation, indicating likely instabilities on the physical design.

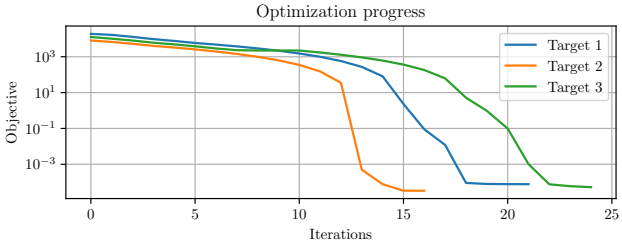

Fig. 13. **Trajectory Optimization.** Progress of our trajectory optimization for the Hoberman sphere throw sequence, for 3 different target states. Using simulation derivatives, we can consistently reduce the objective (error with respect to the target) under a low tolerance in a few iterations.

violation and has a constraint violation that scales with the time step. For a time step of $\Delta t = 0.1$s, which one might use in an optimization context, the violation is visually noticeable (see Fig. 11, right). In contrast, the proposed implicit methods satisfies the joint constraints up to the specified tolerance independent of the time step.

Note that as this is a position-controlled system without passive degrees of freedom, the simulation error will not accumulate over time, meaning that large time steps may be used.

In the accompanying video, we further showcase the usefulness of a general actuator model that allows for any combination of actuated degrees of freedom, for example when modifying an animation that only affects part of a model. By extracting the upper body of the Gazelle model and assigning a position-based actuator with six actuated degrees of freedom to the torso component, the upper body can be driven by the solution of an initial full-body simulation. A user can then iterate on the upper body motion and receive simulation feedback without having to run a simulation for the full model, while still accurately capturing the influence of the lower body dynamics on the upper body.

Sensitivity Analysis. We consider two alternative non-overactuated designs for our Gazelle character, in which 2 out of 8 actuators in the lower body are replaced with passive joints for cost and weight savings. While the kinematic sequences are identical, the choice of which actuators to remove affects the closeness to kinematic singularities and thus the motion stability on the physical system. Leveraging the differentiability of our simulator, we can guide design choices by computing, for our dancing motion, the sensitivity over time of rigid body poses to force-control inputs. A worst-case sensitivity comparison, displayed in Fig 12, shows clear superiority of one option in terms of robustness to perturbations.

Optimization Problem. We demonstrate how the differentiability of our simulator enables it to be used in the context of gradient-based optimization problems. Specifically, we consider a throw of the Hoberman sphere, and solve for the initial velocities and "opening" angle required such that a prescribed target pose is matched at a prescribed time. The converged optimization is able to closely match the target state. We solve the optimization problem using a standard quasi-Newton (BFGS) implementation [Nocedal and Wright 1999], using the adjoint method to differentiate a target-matching objective with respect to the initial state. Fig. 13 shows the optimization progress over time, and we refer to the accompanying video for the resultant trajectories at different stages of the optimization.

Performance. See Tab. 6 for performance statistics on the different examples. Smaller time steps are required when systems have fast and passive dynamics as demonstrated by the Spinning Top and Pendulum examples. We also report the performance difference

Table 5. **Key Specifications for Mechanisms.**

	comp.	const.	const. eqs.	cont. vars.	st. vars.
T-Handle	1	-	-	-	14
Spinning top	1	1	3	-	17
Pendulum	1	1	5	-	19
Hoberman sphere	240	420	1980	-	5340
Satellite	26	37	192	7	556
Iron Man	151	198	982	27	3096
Gazelle	38	45	236	22	768

comp.: number of components; **consts.**: number of constraints; **const. eqs.**: number of constraint equations; **cont. vars.**: number of control variables; **st. vars.**: number of state variables.

Table 6. **Key Time Performance Statistics.**

	act.	Δt	seq. len.	solver	step
T-Handle	passive	1 ms	10 s	general	0.037 ms
Spinning top	passive	0.1 ms	10 s	general	0.045 ms
Hoberman	passive	10 ms	4 s	general	1138.03 ms
Satellite	pos. control	33 ms	10 s	general	7.75 ms
Iron Man	pos. control	33 ms	138.5 s	general fully-constr.	26.89 ms 4.19 ms
Gazelle	pos. control	10 ms	9.82 s	general fully-constr.	3.55 ms 0.84 ms
	force control			general	6.11 ms

act.: type of actuation (passive, position control, or force control); Δt: time step of the simulation; **seq. len.**: length of the simulation sequence; **solver**: type of solver used (general or fast solver for fully-constrained systems); **step**: average time to compute a single simulation step. Computations were performed on a machine with an AMD Ryzen Threadripper Pro 3955WX (16 cores, 3.9 GHz) and 64 GB of RAM.

between the general and the fully-constrained solvers, and between force control and position control on the Gazelle figure.

9 Conclusions

We have devised an implicitly-integrated constrained rigid body dynamics that is quaternion-based, together with a modular set of constraints that facilitates the implementation of common joints and actuators. Our implicit integration guarantees that constraints are satisfied to numerical tolerances, and our solver strategy robustly simulates systems with either less or more constraints than unknown states or overactuation.

In comparison to simulators that keep quaternions at unit length with specialized non-additive gradient updates [Simo and Wong 1991], our final system of discretized equations is well-suited for symbolic differentiation and numerical optimization with standard additive updates. Moreover, our simulator remains fully differentiable, even if subspaces are present in constraint forces or torques, or corresponding Lagrange multipliers.

9.1 Limitations and Future Work

While the time complexity of our simulation enables the use of small time steps, and therefore to achieve the desired accuracy even for longer motion sequences, our time integration is not energy- or momentum-preserving.

Moreover, while our simulator interfaces with generic spatial mechanisms, we leave it as future work to model frictional contact at the joint level, between pairs of bodies, or a body and the environment. Simulators that model joints with frictional contact can account for joint-level sim-to-real gaps [Ferguson et al. 2021], at the cost of significantly higher time complexity. Other options for a practical implementation include using implicit penalties, or solving the dual problem separately for contact forces at each time step; the computed forces would then be fed to our method as external forces.

Even for complex mechanical systems with hundreds of bodies and constraints, and despite the use of implicit integration, we can solve for the system state at the next time step in a few milliseconds, opening exciting future avenues in the exploration of real-time, closed-loop control applications.

We have yet to explore the full potential of the differentiability of our simulator, with rich applications in optimal control and design, and the identification of optimal simulation parameters to close sim-to-real gaps.

References

Roger Alexander. 1977. Diagonally Implicit Runge-Kutta Methods for Stiff O.D.E.'s. *SIAM J. Numer. Anal.* 14, 6 (1977), 1006–1021.

Carmen Arévalo and Per Lötstedt. 1995. Improving the accuracy of BDF methods for index 3 differential-algebraic equations. *BIT Numerical Mathematics* 35 (1995), 297–308.

Uri M Ascher, Hongsheng Chin, Linda R Petzold, and Sebastian Reich. 1995. Stabilization of constrained mechanical systems with daes and invariant manifolds. *Journal of Structural Mechanics* 23, 2 (1995), 135–157.

Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. 2015. LinkEdit: interactive linkage editing using symbolic kinematics. *ACM Trans. Graph.* 34, 4, Article 99 (2015).

Ronen Barzel and Alan H Barr. 1988. A modeling system based on dynamic constraints. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. 179–188.

Joachim Baumgarte. 1972. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering* 1, 1 (1972), 1–16.

Jan Bender, Kenny Erleben, and Jeff Trinkle. 2014. Interactive simulation of rigid body dynamics in computer graphics. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 246–270.

Peter Betsch and Ralf Siebert. 2009. Rigid body dynamics in terms of quaternions: Hamiltonian formulation and conserving numerical integration. *Internat. J. Numer. Methods Engrg.* 79, 4 (2009), 444–473.

Carlo L Bottasso, Daniel Dopico, and Lorenzo Trainelli. 2008. On the optimal scaling of index three DAEs in multibody dynamics. *Multibody System Dynamics* 19 (2008), 3–20.

Kathryn Eleda Brenan, Stephen L Campbell, and Linda Ruth Petzold. 1995. *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM.

Yang Cao, Shengtai Li, and Linda Petzold. 2002. Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software. *Journal of computational and applied mathematics* 149, 1 (2002), 171–191.

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational design of mechanical characters. *ACM Trans. Graph.* 32, 4, Article 83 (2013).

Tom Erez, Yuval Tassa, and Emanuel Todorov. 2015. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 4397–4404.

Kenny Erleben. 2017. Rigid body contact problems using proximal operators. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. Article 13.

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. 2021. Intersection-free Rigid Body Dynamics. *ACM Transactions on Graphics (SIGGRAPH)* 40, 4, Article 183 (2021).

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

F. Sebastin Grassia. 1998. Practical parameterization of rotations using the exponential map. *J. Graph. Tools* 3, 3 (1998), 20 pages.

Edward J Haug. 1989. *Computer aided kinematics and dynamics of mechanical systems*. Vol. 1. Allyn and Bacon Boston.
Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2019. Vibration-minimizing motion retargeting for robotic characters. *ACM Trans. Graph.* 38, 4, Article 102 (2019).
Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. 2005. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph.* 24, 3 (2005).
Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. 2008. Staggered projections for frictional contact in multibody systems. *ACM Trans. Graph.* 27, 5, Article 164 (2008).
Marin Kobilarov, Keenan Crane, and Mathieu Desbrun. 2009. Lie group integrators for animation and control of vehicles. *ACM Trans. Graph.* 28, 2, Article 16 (2009).
Lei Lan, Danny M. Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. 2022. Affine body dynamics: fast, stable and intersection-free simulation of stiff materials. *ACM Trans. Graph.* 41, 4, Article 67 (2022).
Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental potential contact: intersection- and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (2020).
Guirec Maloisel, Espen Knoop, Bernhard Thomaszewski, Moritz Bächer, and Stelian Coros. 2021. Singularity-Aware Design Optimization for Multi-Degree-of-Freedom Spatial Linkages. *IEEE Robotics and Automation Letters* 6, 4 (2021).
Guirec Maloisel, Christian Schumacher, Espen Knoop, Ruben Grandia, and Moritz Bächer. 2023. Optimal Design of Robotic Character Kinematics. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–15.
Michael Möller and Christoph Glocker. 2012. Rigid body dynamics with a scalable body, quaternions and perfect constraints. *Multibody System Dynamics* 27 (04 2012).
M.B. Nielsen and S. Krenk. 2012. Conservative integration of rigid body motion by quaternion parameters with implicit constraints. *Internat. J. Numer. Methods Engrg.* 92, 8 (2012), 734–752.
P. E. Nikravesh, O. K. Kwon, and R. A. Wehage. 1985a. Euler Parameters in Computational Kinematics and Dynamics. Part 2. *Journal of Mechanisms, Transmissions, and Automation in Design* 107, 3 (09 1985), 366–369.
P. E. Nikravesh, R. A. Wehage, and O. K. Kwon. 1985b. Euler Parameters in Computational Kinematics and Dynamics. Part 1. *Journal of Mechanisms, Transmissions, and Automation in Design* 107, 3 (09 1985), 358–365.
Hiroaki Nishikawa. 2019. On Large Start-Up Error of BDF2. *J. Comput. Phys.* 392 (05 2019). doi:10.1016/j.jcp.2019.04.070
Jorge Nocedal and Stephen J Wright. 1999. *Numerical optimization*. Springer.
Lorenzo Pareschi and Giovanni Russo. 2005. Implicit–Explicit Runge–Kutta Schemes and Applications to Hyperbolic Systems with Relaxation. *Journal of Scientific Computing* 25 (2005), 129–155.
Jiten Patel and G.K. Ananthasuresh. 2007. A kinematic theory for radially foldable planar linkages. *International Journal of Solids and Structures* 44, 18 (2007), 6279–6298.
Caleb Rucker. 2018. Integrating Rotations Using Nonunit Quaternions. *IEEE Robotics and Automation Letters* 3, 4 (2018), 2979–2986.
Christian Schumacher, Espen Knoop, and Moritz Bächer. 2021. A Versatile Inverse Kinematics Formulation for Retargeting Motions Onto Robots With Kinematic Loops. *IEEE Robotics and Automation Letters* 6, 2 (2021).
J. C. Simo and K. K. Wong. 1991. Unconditionally stable algorithms for rigid body dynamics that exactly preserve energy and momentum. *Internat. J. Numer. Methods Engrg.* 31, 1 (1991), 19–52.
Breannan Smith, Danny M. Kaufman, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2012. Reflections on simultaneous impact. *ACM Trans. Graph.* 31, 4, Article 106 (2012).
Alessandro Tasora and Paolo Righettini. 1999. Application of quaternion algebra to the efficient computation of jacobians for holonomic-rheonomic constraints. *EUROMECH 404, Lisboa* (1999).
Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. 2014. Computational design of linkage-based characters. *ACM Trans. Graph.* 33, 4, Article 64 (2014).
Robin Tomcin, Dominik Sibbing, and Leif Kobbelt. 2014. Efficient enforcement of hard articulation constraints in the presence of closed loops and contacts. *Computer Graphics Forum* 33, 2 (2014).
Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. 2015. Stable constrained dynamics. *ACM Trans. Graph.* 34, 4, Article 132 (2015).
Firdaus Udwadia and Aaron Schutte. 2010. An Alternative Derivation of the Quaternion Equations of Motion for Rigid-Body Rotational Dynamics. *Journal of Applied Mechanics-transactions of The Asme - J APPL MECH* 77 (07 2010).
Ying Wang, Nicholas J. Weidner, Margaret A. Baxter, Yura Hwang, Danny M. Kaufman, and Shinjiro Sueda. 2019. RedMax: efficient & flexible approach for articulated dynamics. *ACM Trans. Graph.* 38, 4 (2019).
Victoria Wieloch and Martin Arnold. 2021. BDF integrators for constrained mechanical systems on Lie groups. *J. Comput. Appl. Math.* 387 (2021), 112517.
Andrew Witkin and David Baraff. 1997. Physically Based Modeling: Principles and Practice Differential Equation Basics. *Course Note A SIGGRAPH* 97 (1997).
Xiaoming Xu, Jiahui Luo, and Zhigang Wu. 2020. The numerical influence of additional parameters of inertia representations for quaternion-based rigid body dynamics. *Multibody System Dynamics* 49 (07 2020).

A Runge-Kutta Parameters

We provide the Runge-Kutta coefficients using a *Butcher tableau*, which takes the form

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}.$$

The coefficients for the *Pareschi-Russo DIRK* [Pareschi and Russo 2005] are

$$\begin{array}{c|cc} 1-\frac{\sqrt{2}}{2} & 1-\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \sqrt{2}-1 & 1-\frac{\sqrt{2}}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

The coefficients for the *stiffly-accurate two-stage SDIRK* [Alexander 1977], which we also use for the initial step when integrating with BDF2, are

$$\begin{array}{c|cc} 1-\frac{\sqrt{2}}{2} & 1-\frac{\sqrt{2}}{2} & 0 \\ 1 & \sqrt{2}-1 & 1-\frac{\sqrt{2}}{2} \\ \hline & \sqrt{2}-1 & 1-\frac{\sqrt{2}}{2} \end{array}.$$

B Redundant Constraints Analysis

We reproduce here the main steps of the redundancy analysis technique introduced in [Maloisel et al. 2023]. We analyze the forward kinematics Jacobian $\mathbf{K}_{\mathbf{s}_n}$ (introduced in Sec. 6.3) for the modified system in which position-control actuators are turned into corresponding passive joints. The Jacobian is evaluated on a reference state of the system, and its rows are normalized. Relying on a Singular Value Decomposition, we compute an orthonormal basis $\mathbf{Z}$ of the left-kernel of $\mathbf{K}_{\mathbf{s}_n}$, i.e. such that $\mathbf{Z}^T\mathbf{K}_{\mathbf{s}_n} = 0$. Each row k of these equations indicates one degree of redundancy

$$\sum_i z_{ik}\mathbf{k}_i = 0, \tag{34}$$

where $\mathbf{k}_i$ refers to row i of $\mathbf{K}_{\mathbf{s}_n}$. For any j such that $z_{jk} \neq 0$, Eq. 34 shows that $\mathbf{k}_j$ could be eliminated, as it is already in the span of the other constraints:

$$\mathbf{k}_j = -\sum_{i\neq j} \frac{z_{ik}}{z_{jk}}\mathbf{k}_i. \tag{35}$$

So that remaining rows of $\mathbf{K}_{\mathbf{s}_n}$ span the same subspace (otherwise we would introduce additional mobility in the system), we pick j yielding the smallest right-hand-side coefficients in Eq. 35 (i.e. so that $\mathbf{k}_j$ is the "least" orthogonal to remaining constraints)

$$j = \underset{j}{\operatorname{argmin}} \min_k \sum_{i\neq j} \frac{z_{ik}^2}{z_{jk}^2}. \tag{36}$$

Marking $\mathbf{k}_j$ as eliminated, we also eliminate the corresponding equation k from $\mathbf{Z}$, subtracting it from other equations

$$z_{:i} \leftarrow z_{:i} - \frac{z_{ji}}{z_{jk}} z_{:k}. \tag{37}$$

Table 7. **Constraints** We abbreviate transformed follower axis $\mathbf{a}_x$ with $\mathbf{r}^F_x = \mathbf{R}(\mathbf{q}^F)\mathbf{a}_x$ and transformed base axis $\mathbf{a}_x$ with $\mathbf{r}^B_x = \mathbf{R}(\mathbf{q}^B)\mathbf{a}_x$ (and similar for other axes). To abbreviate the two moment arms for translational constraints, we introduce $\mathbf{m}^F = \mathbf{R}(\mathbf{q}^F)\mathbf{x}^F_{\text{rb}}$ and $\mathbf{m}^B = \mathbf{R}(\mathbf{q}^F)\mathbf{x}^F_{\text{rb}} + \mathbf{c}^F - \mathbf{c}^B$. To form the base body torques, $\boldsymbol{\tau}^B$, for translational constraints, we replace $\mathbf{m}^F$ with $\mathbf{m}^B$ and multiply the individual force terms with -1 in the expressions for $\boldsymbol{\tau}^F$. In rotational constraints and corresponding torques for position-based actuators, we use abbreviations $\underline{\mathbf{r}}^F_y(p_x) = \mathbf{R}(\mathbf{q}^F)\mathbf{R}[p_x, \mathbf{a}_x]^T\mathbf{a}_y$ on the follower side and $\underline{\mathbf{r}}^B_y(p_x) = \mathbf{R}(\mathbf{q}^B)\mathbf{R}[p_x, \mathbf{a}_x]\mathbf{a}_y$ on the base side (and similar for other axes and parameters). To parameterize all rotational degrees of freedom (Rotational Constraints, free, position-based actuators), we use a quaternion $\mathbf{p}$ and the abbreviation $\underline{\mathbf{r}}^F_x(\mathbf{p}) = \mathbf{R}(\mathbf{q}^F)\mathbf{A}\mathbf{R}(\mathbf{p})\mathbf{e}_x$ where $\mathbf{e}_x$ is the first unit vector (and similar for unit vectors $\mathbf{e}_x$ and $\mathbf{e}_y$). Note that we use two different sets of Lagrange multipliers λ_x, λ_y and λ_z for the two constraint categories (translational vs. rotational). p in a translational constraint is either a position *or* a force parameter, and p in a rotational constraint is either an angle *or* torque parameter. For rotational constraints, we also provide the consistency constraints that we use to prevent flips. To obtain formulas for different axes for the 1 DoF or 2 DoF cases, the roles of (x, y, z) can be permuted to (y, z, x) or (z, x, y).

Translational Constraints

	joints			position-based actuators			force-based actuators		
	$\mathbf{C}$	$\mathbf{f}^F$	$\boldsymbol{\tau}^F$	$\mathbf{C}$	$\mathbf{f}^F$	$\boldsymbol{\tau}^F$	$\mathbf{C}$	$\mathbf{f}^F$	$\boldsymbol{\tau}^F$
fixed	$\mathbf{r}^B_x \cdot \mathbf{d}$ $\mathbf{r}^B_y \cdot \mathbf{d}$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x\lambda_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x\lambda_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$	$\mathbf{r}^B_x \cdot \mathbf{d}$ $\mathbf{r}^B_y \cdot \mathbf{d}$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x\lambda_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x\lambda_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$	$\mathbf{r}^B_x \cdot \mathbf{d}$ $\mathbf{r}^B_y \cdot \mathbf{d}$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x\lambda_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x\lambda_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$
1 DoF	$\mathbf{r}^B_y \cdot \mathbf{d}$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$	$\mathbf{r}^B_x \cdot \mathbf{d} - p_x$ $\mathbf{r}^B_y \cdot \mathbf{d}$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x\lambda_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x\lambda_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$	$\mathbf{r}^B_y \cdot \mathbf{d}$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x p_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x p_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$
2 DoF	$\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$	$\mathbf{r}^B_x \cdot \mathbf{d} - p_x$ $\mathbf{r}^B_y \cdot \mathbf{d} - p_y$ $\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x\lambda_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x\lambda_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$	$\mathbf{r}^B_z \cdot \mathbf{d}$	$+\mathbf{r}^B_x p_x$ $+\mathbf{r}^B_y p_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x p_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y p_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$
free				$\mathbf{r}^B_x \cdot \mathbf{d} - p_x$ $\mathbf{r}^B_y \cdot \mathbf{d} - p_y$ $\mathbf{r}^B_z \cdot \mathbf{d} - p_z$	$+\mathbf{r}^B_x\lambda_x$ $+\mathbf{r}^B_y\lambda_y$ $+\mathbf{r}^B_z\lambda_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x\lambda_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y\lambda_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z\lambda_z)$		$+\mathbf{r}^B_x p_x$ $+\mathbf{r}^B_y p_y$ $+\mathbf{r}^B_z p_z$	$+\mathbf{m}^F \times (\mathbf{r}^B_x p_x)$ $+\mathbf{m}^F \times (\mathbf{r}^B_y p_y)$ $+\mathbf{m}^F \times (\mathbf{r}^B_z p_z)$

Rotational Constraints

	joints			position-based actuators			force-based actuators		
	$\mathbf{C}$	$\boldsymbol{\tau}^F$	consist. $\mathbf{C}$	$\mathbf{C}$	$\boldsymbol{\tau}^F$	consist. $\mathbf{C}$	$\mathbf{C}$	$\boldsymbol{\tau}^F$	consist. $\mathbf{C}$
fixed	$\mathbf{r}^F_y \cdot \mathbf{r}^B_z$ $\mathbf{r}^F_z \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_x \cdot \mathbf{r}^B_y$	$+(\mathbf{r}^F_y \times \mathbf{r}^B_z)\lambda_x$ $+(\mathbf{r}^F_z \times \mathbf{r}^B_x)\lambda_y$ $+(\mathbf{r}^F_x \times \mathbf{r}^B_y)\lambda_z$	$\mathbf{r}^F_x \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_y \cdot \mathbf{r}^B_y$ $\mathbf{r}^F_z \cdot \mathbf{r}^B_z$	$\mathbf{r}^F_y \cdot \mathbf{r}^B_z$ $\mathbf{r}^F_z \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_x \cdot \mathbf{r}^B_y$	$+(\mathbf{r}^F_y \times \mathbf{r}^B_z)\lambda_x$ $+(\mathbf{r}^F_z \times \mathbf{r}^B_x)\lambda_y$ $+(\mathbf{r}^F_x \times \mathbf{r}^B_y)\lambda_z$	$\mathbf{r}^F_x \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_y \cdot \mathbf{r}^B_y$ $\mathbf{r}^F_z \cdot \mathbf{r}^B_z$	$\mathbf{r}^F_y \cdot \mathbf{r}^B_z$ $\mathbf{r}^F_z \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_x \cdot \mathbf{r}^B_y$	$+(\mathbf{r}^F_y \times \mathbf{r}^B_z)\lambda_x$ $+(\mathbf{r}^F_z \times \mathbf{r}^B_x)\lambda_y$ $+(\mathbf{r}^F_x \times \mathbf{r}^B_y)\lambda_z$	$\mathbf{r}^F_x \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_y \cdot \mathbf{r}^B_y$ $\mathbf{r}^F_z \cdot \mathbf{r}^B_z$
1 DoF	$-\mathbf{r}^F_x \cdot \mathbf{r}^B_z$ $\mathbf{r}^F_x \cdot \mathbf{r}^B_y$	$-(\mathbf{r}^F_x \times \mathbf{r}^B_z)\lambda_y$ $+(\mathbf{r}^F_x \times \mathbf{r}^B_y)\lambda_z$	$\mathbf{r}^F_x \cdot \mathbf{r}^B_x$	$\underline{\mathbf{r}}^F_y(p_x) \cdot \mathbf{r}^B_z$ $-\mathbf{r}^F_x \cdot \mathbf{r}^B_z$ $\mathbf{r}^F_x \cdot \mathbf{r}^B_y$	$+(\underline{\mathbf{r}}^F_y(p_x) \times \mathbf{r}^B_z)\lambda_x$ $-(\mathbf{r}^F_x \times \mathbf{r}^B_z)\lambda_y$ $+(\mathbf{r}^F_x \times \mathbf{r}^B_y)\lambda_z$	$\mathbf{r}^F_x \cdot \mathbf{r}^B_x$ $\underline{\mathbf{r}}^F_y(p_x) \cdot \mathbf{r}^B_y$ $\underline{\mathbf{r}}^F_z(p_x) \cdot \mathbf{r}^B_z$	$-\mathbf{r}^F_x \cdot \mathbf{r}^B_z$ $\mathbf{r}^F_x \cdot \mathbf{r}^B_y$	$+\mathbf{r}^B_x p_x$ $-(\mathbf{r}^F_x \times \mathbf{r}^B_z)\lambda_y$ $+(\mathbf{r}^F_x \times \mathbf{r}^B_y)\lambda_z$	$\mathbf{r}^F_x \cdot \mathbf{r}^B_x$
2 DoF	$-\mathbf{r}^F_y \cdot \mathbf{r}^B_x$	$-(\mathbf{r}^F_y \times \mathbf{r}^B_x)\lambda_z$		$\mathbf{r}^F_y \cdot \underline{\mathbf{r}}^B_z(p_x)$ $\underline{\mathbf{r}}^F_z(p_y) \cdot \mathbf{r}^B_x$ $-\mathbf{r}^F_y \cdot \mathbf{r}^B_x$	$+(\mathbf{r}^F_y \times \underline{\mathbf{r}}^B_z(p_x))\lambda_x$ $+(\underline{\mathbf{r}}^F_z(p_y) \times \mathbf{r}^B_x)\lambda_y$ $-(\mathbf{r}^F_y \times \mathbf{r}^B_x)\lambda_z$	$\underline{\mathbf{r}}^F_x(p_y) \cdot \mathbf{r}^B_x$ $\mathbf{r}^F_y \cdot \underline{\mathbf{r}}^B_y(p_x)$ $\underline{\mathbf{r}}^F_z(p_y) \cdot \underline{\mathbf{r}}^B_z(p_x)$	$-\mathbf{r}^F_y \cdot \mathbf{r}^B_x$	$\mathbf{r}^F_x p_x$ $+\mathbf{r}^B_y p_y$ $-(\mathbf{r}^F_y \times \mathbf{r}^B_x)\lambda_z$	
free				$\underline{\mathbf{r}}^F_y(\mathbf{p}) \cdot \mathbf{r}^B_z$ $\underline{\mathbf{r}}^F_z(\mathbf{p}) \cdot \mathbf{r}^B_x$ $\underline{\mathbf{r}}^F_x(\mathbf{p}) \cdot \mathbf{r}^B_y$	$+(\underline{\mathbf{r}}^F_y(\mathbf{p}) \times \mathbf{r}^B_z)\lambda_x$ $+(\underline{\mathbf{r}}^F_z(\mathbf{p}) \times \mathbf{r}^B_x)\lambda_y$ $+(\underline{\mathbf{r}}^F_x(\mathbf{p}) \times \mathbf{r}^B_y)\lambda_z$	$\underline{\mathbf{r}}^F_x(\mathbf{p}) \cdot \mathbf{r}^B_x$ $\underline{\mathbf{r}}^F_y(\mathbf{p}) \cdot \mathbf{r}^B_y$ $\underline{\mathbf{r}}^F_z(\mathbf{p}) \cdot \mathbf{r}^B_z$		$+\mathbf{r}^B_x p_x$ $+\mathbf{r}^B_y p_y$ $+\mathbf{r}^B_z p_z$	

We iterate the process until we have used all equations from $\mathbf{Z}$.

For overactuated systems, the process is then repeated after reintroducing constraints enforcing position-based actuation, this time considering only actuation constraints for elimination. It is necessary to treat passive and actuation constraints separately, because actuators are kinematically equivalent to fixed joints, and thus could be indifferently be eliminated by this process otherwise.

C Computing Derivatives

In this appendix, we detail how to compute simulation derivatives using the implicit function theorem, and how to use the adjoint method to speed up computations.

C.1 Implicit Function Theorem

Non-Overconstrained Systems. For mechanical systems without redundancies, we apply the implicit function theorem to Eq. 25 to obtain the sensitivity of state $\mathbf{y}_n$ at a given time step with respect to parameters $\mathbf{p}$ (e.g. initial state, control inputs, or design variables):

$$\frac{\partial \mathbf{y}_n}{\partial \mathbf{p}} = -\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}_n}\right)^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{p}}. \tag{38}$$

This is a partial derivative, expressing only the effect of $\mathbf{p}$ through a single simulation step. The full derivative $\frac{\mathrm{d}\mathbf{y}_n}{\mathrm{d}\mathbf{p}}$ is obtained by the chain rule

$$\frac{\mathrm{d}\mathbf{y}_n}{\mathrm{d}\mathbf{p}} = \frac{\partial \mathbf{y}_n}{\partial \mathbf{p}} + \frac{\partial \mathbf{y}_n}{\partial \mathbf{y}_p}\frac{\mathrm{d}\mathbf{y}_p}{\mathrm{d}\mathbf{p}}. \tag{39}$$

The single-step sensitivity $\frac{\partial \mathbf{y}_n}{\partial \mathbf{y}_p}$ w.r.t. the aggregate previous state can be computed by setting $\mathbf{p} = \mathbf{y}_p$ in Eq. 38, noting that only the effect of $\mathbf{s}_p, \dot{\mathbf{s}}_p$ needs to be considered, as $\boldsymbol{\lambda}_p$ does not affect $\mathbf{F}$. The derivative $\frac{\mathrm{d}\mathbf{y}_p}{\mathrm{d}\mathbf{p}}$ is obtained as a linear combination of $\frac{\mathrm{d}\mathbf{y}^{(k-1)}}{\mathrm{d}\mathbf{p}}, \frac{\mathrm{d}\mathbf{y}^{(k-2)}}{\mathrm{d}\mathbf{p}} \ldots$, with coefficients as per the chosen time integrator (BDF or Runge-Kutta).

Overconstrained Systems. For systems with redundant constraints, Eqs. 38 and 39 must be applied to the reduced system without redundant kinematic constraints and corresponding multipliers. Note that this already provides the sensitivities $\frac{\mathrm{d}\mathbf{s}_n}{\mathrm{d}\mathbf{p}}, \frac{\mathrm{d}\dot{\mathbf{s}}_n}{\mathrm{d}\mathbf{p}}$, as $\mathbf{s}_n, \dot{\mathbf{s}}_n$ are identical for the full and reduced systems.

In applications where the sensitivities $\frac{\mathrm{d}\boldsymbol{\lambda}_n}{\mathrm{d}\mathbf{p}}$ are also required, one must differentiate through the weighted least-norm solution (Eq. 32). Setting

$$\mathbf{G} := \mathbf{E}_{\boldsymbol{\lambda}_n} \mathbf{W}^{-1} \mathbf{E}_{\boldsymbol{\lambda}_n}{}^T, \tag{40}$$

we obtain, using the analytical derivative of the pseudo-inverse and removing terms that are always zero,

$$\begin{aligned} \frac{\partial \boldsymbol{\lambda}_n}{\partial \mathbf{x}} &= -\mathbf{W}^{-1} \mathbf{E}_{\boldsymbol{\lambda}_n}{}^T \mathbf{G}^+ \mathbf{E}_{\mathbf{x}} \\ &\quad - \left(\mathbf{I} - \mathbf{W}^{-1} \mathbf{E}_{\boldsymbol{\lambda}_n}{}^T \mathbf{G}^+ \mathbf{E}_{\boldsymbol{\lambda}_n}\right) \mathbf{W}^{-1} \mathbf{E}_{\boldsymbol{\lambda}_n, \mathbf{x}}{}^T \mathbf{G}^+ \mathbf{E}, \end{aligned} \tag{41}$$

for $\mathbf{x} := \mathbf{p}, \mathbf{s}_n, \dot{\mathbf{s}}_n$, or $\dot{\mathbf{s}}_p$, and where $\mathbf{I}$ is the identity matrix. $\mathbf{E}_{\mathbf{x}}, \mathbf{E}_{\boldsymbol{\lambda}_n, \mathbf{x}}{}^T$ are evaluated with $\boldsymbol{\lambda}$ set to the multipliers computed for the reduced system, while $\mathbf{E}_{\boldsymbol{\lambda}_n}$ and $\mathbf{E}$ are evaluated with $\boldsymbol{\lambda} = \mathbf{0}$.

The pseudo-inverse $\mathbf{G}^+$ becomes an inverse if there are no unconstrained degrees of freedom. $\frac{\partial \boldsymbol{\lambda}_n}{\partial \mathbf{p}}$ is zero in many cases, but we include these derivatives for completeness. The second term in Eq. 41 is zero for $\mathbf{x} = \dot{\mathbf{s}}_n, \dot{\mathbf{s}}_p$. For $\mathbf{x} = \mathbf{s}_n$, this term involves a 3rd-order tensor times a vector $\mathbf{E}_{\boldsymbol{\lambda}_n, \mathbf{s}_n}{}^T \mathbf{G}^+ \mathbf{E}$ which can be computed by finding the second derivative $\frac{\partial^2 (\mathbf{E}^T \mathbf{v})}{\partial \boldsymbol{\lambda}_n \partial \mathbf{s}_n}$ for a fixed vector $\mathbf{v}$, and evaluating it for $\mathbf{v} = \mathbf{G}^+ \mathbf{E}$.

We can then obtain $\frac{\mathrm{d}\boldsymbol{\lambda}_n}{\mathrm{d}\mathbf{p}}$ through the chain rule:

$$\frac{\mathrm{d}\boldsymbol{\lambda}_n}{\mathrm{d}\mathbf{p}} = \frac{\partial \boldsymbol{\lambda}_n}{\partial \mathbf{p}} + \frac{\partial \boldsymbol{\lambda}_n}{\partial \mathbf{s}_n} \frac{\mathrm{d}\mathbf{s}_n}{\mathrm{d}\mathbf{p}} + \frac{\partial \boldsymbol{\lambda}_n}{\partial \dot{\mathbf{s}}_n} \frac{\mathrm{d}\dot{\mathbf{s}}_n}{\mathrm{d}\mathbf{p}} + \frac{\partial \boldsymbol{\lambda}_n}{\partial \dot{\mathbf{s}}_p} \frac{\mathrm{d}\dot{\mathbf{s}}_p}{\mathrm{d}\mathbf{p}} \tag{42}$$

C.2 Adjoint Method

When the full sensitivity matrix $\frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}\mathbf{p}}$ is not needed by itself, but only as part of a chain rule, we can use the *adjoint method* for more efficient computations [Cao et al. 2002]. In particular, to solve an optimization problem around a simulation sequence, we only need to compute the gradient of an objective O with respect to parameters $\mathbf{p}$

$$\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}} = \frac{\partial O}{\partial \mathbf{p}} + \sum_k \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}\mathbf{p}}{}^T \frac{\partial O}{\partial \mathbf{y}^{(k)}}. \tag{43}$$

For systems with non-redundant constraints, assuming temporarily that $\frac{\partial \mathbf{y}_\mathrm{n}}{\partial \mathbf{y}_\mathrm{p}} = \mathbf{0}$, we see from Eq. 38 that the term k in the sum is

$$-\frac{\partial \mathbf{F}}{\partial \mathbf{p}}^T \left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}_n}\right)^{-T} \frac{\partial O}{\partial \mathbf{y}_n}. \tag{44}$$

While evaluating Eq. 38 requires the solution of a linear system with as many right-hand sides as parameters $\mathbf{p}$, we can instead compute the *adjoint* vector $\bar{\mathbf{y}}_n := -\left(\frac{\partial \mathbf{F}}{\partial \mathbf{y}_n}\right)^{-T} \frac{\partial O}{\partial \mathbf{y}_n}$, solving the transposed system with a *single* right-hand side, and compute in a second step the product $\frac{\partial \mathbf{F}}{\partial \mathbf{p}}^T \bar{\mathbf{y}}_n$.

Taking into account dependencies across consecutive steps, this translates into Alg. 1 for systems with non-redundant constraints. The algorithm uses the adjoint method to update gradients of the objective with respect to states and parameters, from the end of the sequence to the beginning, assuming a fixed initial state $\mathbf{y}^{(\mathbf{0})}$, which may optionally depend on $\mathbf{p}$. The algorithm is written for a BDF1 discretization (i.e. $\mathbf{y}_p = \mathbf{y}^{(k-1)}$ if $\mathbf{y}_n = \mathbf{y}^{(k)}$); for higher-order discretizations, the update of line 4 in Alg. 1 must be distributed accordingly to $\mathbf{g}_\mathbf{y}^{(k-1)}, \mathbf{g}_\mathbf{y}^{(k-2)}, \ldots$, with the same weights as for computing the aggregate previous state $\mathbf{y}_p$.

Algorithm 1. Adjoint method for non-overconstrained systems.

1. Initialize $\mathbf{g}_\mathbf{p} := \frac{\partial O}{\partial \mathbf{p}}$ and $\mathbf{g}_\mathbf{y}^{(k)} := \frac{\partial O}{\partial \mathbf{y}^{(k)}}$ for $k = 0, \ldots, n$.
2. **For** $k = n$ to 1, do:
3. Compute the adjoint $\bar{\mathbf{y}}^{(k)} := -\frac{\partial \mathbf{F}^{(k)}}{\partial \mathbf{y}_n}{}^{-T} \mathbf{g}_k$
4. Update $\mathbf{g}_\mathbf{y}^{(k-1)}$ += $\frac{\partial \mathbf{F}^{(k)}}{\partial \mathbf{y}_p}{}^T \bar{\mathbf{y}}^{(k)}$
5. Update $\mathbf{g}_\mathbf{p}$ += $\frac{\partial \mathbf{F}^{(k)}}{\partial \mathbf{p}}{}^T \bar{\mathbf{y}}^{(k)}$
6. Update $\mathbf{g}_\mathbf{p}$ += $\frac{\partial \mathbf{y}^{(0)}}{\partial \mathbf{p}}{}^T \mathbf{g}_\mathbf{y}^{(0)}$
7. **Output** $\mathbf{g}_\mathbf{p}$ $\left(= \frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}}\right)$

See Alg. 2 for the overconstrained case for the same assumptions (fixed initial state, BDF1 discretization). While the standard adjoint method cannot be applied to the weighted least-norm multipliers, we can similarly ensure that only linear systems with a single right-hand side are solved in Eq. 41, as sensitivities to the multipliers are always multiplied with a column vector in the updates of lines 3–5 and 9 in Alg. 2. The adjoint vector computation on line 6 is done in the context of the reduced system without redundant constraints or multipliers.

D Semi-Implicit Euler

The semi-implicit method, a common choice in graphics and robotics [Erez et al. 2015], solves for velocities first and uses the updated velocities to integrate positions. It uses a state consisting of the body pose, $\mathbf{s} = \begin{bmatrix}\mathbf{c}^T & \mathbf{q}^T\end{bmatrix}^T$, and twist $\mathbf{u} = \begin{bmatrix}\mathbf{v}^T & \boldsymbol{\omega}^T\end{bmatrix}^T$.

The equilibrium equation written in terms of twist is

$$\mathbf{M}\dot{\mathbf{u}} + \mathbf{h} - \mathbf{C}_{\boldsymbol{\delta s}}^T \boldsymbol{\lambda} = \mathbf{0}, \text{ where} \tag{45}$$

$$\mathbf{M} = \begin{bmatrix} M \cdot \mathbf{I} & 0 \\ 0 & \mathbf{J}(\mathbf{q}) \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} -\mathbf{f} \\ \boldsymbol{\omega} \times (\mathbf{J}(\mathbf{q})\boldsymbol{\omega}) - \boldsymbol{\tau} \end{bmatrix}, \tag{46}$$

with the inertia expressed in global coordinates, $\mathbf{J}(\mathbf{q}) = \mathbf{R}(\mathbf{q}) \mathbf{J}_\mathrm{rb} \mathbf{R}(\mathbf{q})^T$.

The constraint derivative, $\mathbf{C}_{\boldsymbol{\delta s}} = [\mathbf{C}_{\boldsymbol{c}}, \mathbf{C}_{\boldsymbol{\phi}}]$, is the derivative of the constraints w.r.t. the center of mass position and the incremental body orientation parameterized by a rotation vector, $\boldsymbol{\phi}$. The orientation derivative is therefore related to the derivative w.r.t. the quaternion via

$$\mathbf{C}_{\boldsymbol{\phi}} = \mathbf{C}_\mathbf{q} \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{q}} = \frac{1}{2} \mathbf{C}_\mathbf{q} \mathbf{G}^T(\mathbf{q}), \tag{47}$$

Algorithm 2. Adjoint method for overconstrained systems.

1. Initialize $\mathbf{g}_\mathbf{p} := \frac{\partial O}{\partial \mathbf{p}}$ and $\begin{cases} \mathbf{g}_\mathbf{s}^{(k)} := \frac{\partial O}{\partial \mathbf{s}^{(k)}} \\ \mathbf{g}_{\dot{\mathbf{s}}}^{(k)} := \frac{\partial O}{\partial \dot{\mathbf{s}}^{(k)}} \\ \mathbf{g}_{\boldsymbol{\lambda}}^{(k)} := \frac{\partial O}{\partial \boldsymbol{\lambda}^{(k)}} \end{cases}$ for $k = 0, \ldots, n$.
2. **For** $k = n$ to 1, do:
3. Update $\mathbf{g}_\mathbf{s}^{(k)}$ += $\frac{\partial \boldsymbol{\lambda}^{(k)}}{\partial \mathbf{s}^{(k)}}^T \mathbf{g}_{\boldsymbol{\lambda}}^{(k)}$
4. Update $\mathbf{g}_{\dot{\mathbf{s}}}^{(k)}$ += $\frac{\partial \boldsymbol{\lambda}^{(k)}}{\partial \dot{\mathbf{s}}^{(k)}}^T \mathbf{g}_{\boldsymbol{\lambda}}^{(k)}$
5. Update $\mathbf{g}_{\dot{\mathbf{s}}}^{(k-1)}$ += $\frac{\partial \boldsymbol{\lambda}^{(k)}}{\partial \dot{\mathbf{s}}^{(k-1)}}^T \mathbf{g}_{\boldsymbol{\lambda}}^{(k)}$
6. Compute the adjoint $\bar{\mathbf{y}}^{(k)} := -\frac{\partial \mathbf{F}^{(k)}}{\partial \mathbf{y}_n}^{-T} \begin{bmatrix} \mathbf{g}_\mathbf{s}^{(k)} \\ \mathbf{g}_{\dot{\mathbf{s}}}^{(k)} \\ \mathbf{0} \end{bmatrix}$
7. Update $\mathbf{g}_\mathbf{s}^{(k-1)}$ += $\frac{\partial \mathbf{F}^{(k)}}{\partial \mathbf{s}_p}^T \bar{\mathbf{y}}^{(k)}$
8. Update $\mathbf{g}_{\dot{\mathbf{s}}}^{(k-1)}$ += $\frac{\partial \mathbf{F}^{(k)}}{\partial \dot{\mathbf{s}}_p}^T \bar{\mathbf{y}}^{(k)}$
9. Update $\mathbf{g}_\mathbf{p}$ += $\frac{\partial \mathbf{F}^{(k)}}{\partial \mathbf{p}}^T \bar{\mathbf{y}}^{(k)} + \frac{\partial \boldsymbol{\lambda}^{(k)}}{\partial \mathbf{p}}^T \mathbf{g}_{\boldsymbol{\lambda}}^{(k)}$
10. Update $\mathbf{g}_\mathbf{p}$ += $\frac{\partial \mathbf{y}^{(0)}}{\partial \mathbf{p}}^T \mathbf{g}_\mathbf{y}^{(0)}$
11. **Output** $\mathbf{g}_\mathbf{p}$ $\left(= \frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}}\right)$

analogous to the relationship between the angular velocity and the time derivative of the quaternion (property 10 in Tab. 1). The time derivative of the constraints can be written as

$$\dot{\mathbf{C}} = \mathbf{C}_{\boldsymbol{\delta s}}\mathbf{u} + \mathbf{C}_{\boldsymbol{\theta}}\dot{\boldsymbol{\theta}}, \tag{48}$$

which is linear in the body twist. The second term accounts for changes in the control parameters.

Since enforcing constraint velocities to be zero leads to numerical drift, the velocity constraint is stabilized through *Baumgarte-stabilization*, with coefficient $\alpha \in [0, 1]$

$$\dot{\mathbf{C}} + \frac{\alpha}{\Delta t}\mathbf{C} = \mathbf{0}. \tag{49}$$

Then, by substituting time derivatives with finite differences, and after multiplying the equilibrium equations by Δt, the following linear system is obtained

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}_{\boldsymbol{\delta s}}^T \\ \mathbf{C}_{\boldsymbol{\delta s}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{(k)} \\ -\Delta t \boldsymbol{\lambda}^{(k)} \end{bmatrix} = \begin{bmatrix} \mathbf{M}\mathbf{u}^{(k-1)} - \Delta t \mathbf{h} \\ -\mathbf{C}_{\boldsymbol{\theta}} \frac{(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^{(k-1)})}{\Delta t} - \frac{\alpha}{\Delta t}\mathbf{C} \end{bmatrix}, \tag{50}$$

which simultaneously enforces the velocity constraints and the discretized equilibrium equations. Note that dynamics and constraint terms in these equations are evaluated at time $k-1$. Solving this linear system yields the updated twist and constraint forces. Afterward, the body pose is integrated according to

$$\mathbf{c}^{(k)} = \mathbf{c}^{(k-1)} + \Delta t \mathbf{v}^{(k)}, \tag{51}$$

$$\mathbf{q}^{(k)} = \exp\left(\Delta t \boldsymbol{\omega}^{(k)}\right) \mathbf{q}^{(k-1)}, \tag{52}$$

where the exponential map is used to convert the rotation vector $\Delta t \boldsymbol{\omega}$ to the corresponding quaternion.