Multibody System Dynamics (2023) 61:381–414
https://doi.org/10.1007/s11044-023-09949-x

RESEARCH

Variational integrators and graph-based solvers for multibody dynamics in maximal coordinates

Jan Brüdigam[1] · Stefan Sosnowski[1] · Zachary Manchester[2] · Sandra Hirche[1]

Received: 12 February 2023 / Accepted: 20 October 2023 / Published online: 3 November 2023

Abstract
Multibody dynamics simulators are an important tool in many fields, including learning and control in robotics. However, many existing dynamics simulators suffer from inaccuracies when dealing with constrained mechanical systems due to unsuitable integrators with bad energy behavior and problematic constraint violations, for example in contact interactions. Variational integrators are numerical discretization methods that can reduce physical inaccuracies when simulating mechanical systems, and formulating the dynamics in maximal coordinates allows for easy and numerically robust incorporation of constraints such as kinematic loops or contacts. Therefore, this article derives a variational integrator for mechanical systems with equality and inequality constraints in maximal coordinates. Additionally, efficient graph-based sparsity-exploiting algorithms for solving the integrator are provided and implemented as an open-source simulator. The evaluation of the simulator shows improved physical accuracy due to the variational integrator and the advantages of the sparse solvers. Comparisons to minimal-coordinate algorithms show improved numerical robustness, and application examples of a walking robot and an exoskeleton with explicit constraints demonstrate the necessity and capabilities of maximal coordinates.

Keywords Maximal coordinates · Multibody dynamics · Variational integrators · Simulation

✉ J. Brüdigam
jan.bruedigam@tum.de

S. Sosnowski
sosnowski@tum.de

Z. Manchester
zacm@cmu.edu

S. Hirche
hirche@tum.de

[1] School of Computation, Information and Technology, Technical University of Munich, Barer Str. 21, Munich, 80333, Germany

[2] The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, 15213, PA, USA

1 Introduction

Simulators for mechanical systems are widely used, for example, in testing and verification [1, 2], in model-based control strategies [3, 4], or in learning-based methods [5, 6]. However, many common simulators have numerical difficulties with more complex mechanical systems involving constraints [7]. Such constraints can represent joints connecting rigid bodies, which may form kinematic loops, for example, in exoskeletons. Constraints can also be used to confine the movement of bodies, for example, to model joint limits in robotic arms or to describe contact with other bodies or the environment in walking and grasping. Exactly enforcing such constraints can cause numerical issues, for example, due to the stiff nature of contact interactions. To alleviate these numerical issues, simulators often allow small constraint violations by representing all constraints as spring-damper elements, as in MuJoCo [8] and Brax [9], or by accepting interpenetration of bodies, as in Drake [10] and Bullet [11]. Small violations can sometimes be acceptable, for example, contact interpenetration in the order of micrometers for meter-scale walking robots. But millimeter or centimeter violations, for example in MuJoCo, can be considered too large. Employing these methods and accepting larger constraint violations for stable simulations contributes to the sim-to-real gap, which is a major issue in robotics [12].

Therefore, this article addresses physically accurate simulations of mechanical systems. Specifically, we focus on good modeling of the energy behavior of such systems, as this quantity is important for the stability of dynamical systems from a control theory perspective. Moreover, we treat the correct enforcement of constraints, since allowing for constraint violations can lead to problematic results. For example, allowing softness in contact interactions can lead to wrong contact points, which can cause a robot to fail to properly grasp an object when the simulation results are directly transferred to a real-world application.

Most simulators, including the ones listed above except Brax, use minimal coordinates (generalized coordinates) as a mechanism's state representation. Given a system with m degrees of freedom and a set of constraints that removes c degrees of freedom, the system is parameterized by $n = m - c$ independent coordinates. In minimal coordinates, the constraints are implicitly incorporated into the dynamics equations, and only the smallest possible number of variables is retained, which leads to dimensionally small problem sizes. However, eliminating constraints requires specialized treatment of each constraint type and is not possible for nonholonomic constraints such as contacts. Furthermore, the algorithms typically used for minimal coordinates can have numerical issues such as ill-conditioning [13, 14], which may be increased by their recursive nature, although a rigorous study of the numerical effects and stability of minimal-coordinate dynamics algorithms remains to be done. We provide some empirical evidence for these numerical issues in Sect. 5.

It is also possible to explicitly retain all or part of the constraints on a mechanical system by introducing constraint forces to adhere to these constraints. With this approach, redundant coordinates are obtained since the constraints make part of the coordinates mathematically dependent. There are many ways to parameterize mechanical systems in redundant coordinates. Natural coordinates use three points in Cartesian coordinates to parameterize the configuration of each body [15]. Alternatively, the position of the center of mass of a body and a director frame for the orientation can be used [16, 17]. These methods have in common that they aim at circumventing the inherent intricacies of parameterizing rotations. In contrast, we use so-called maximal coordinates [18], which represent each body in a mechanism with its three degrees of freedom for the position of the center of mass and three degrees of freedom for the orientation by purposefully building on the group structure of rotations. All kinematic relations, such as joints or contacts, are formulated as explicit constraints. The use of maximal coordinates allows for very modular modeling of dynamics, which makes it possible to describe many different kinds of constraints in a unified and non-specialized manner. Additionally, resorting to maximal and redundant coordinates enables the use of different algorithms than for minimal coordinates, specifically direct matrix methods, which generally have well-investigated numerical properties and enable the formulation of stable and well-conditioned algorithms [19]. The core idea of direct matrix methods is to modify the underlying matrix before, during, and after it is used in an algorithm to improve the stability and accuracy of an algorithm. Examples of such modifications are scaling certain matrix entries or iterative solution refinement. The few existing works on simulation, specifically in maximal coordinates, have explored algorithms for continuous-time forward dynamics [18] and discrete-time algorithms for mechanisms without kinematic loops or contacts [20]. Besides their application in simulation, maximal coordinates have shown promising results in control applications [21, 22]. The simulator Brax is also based on maximal coordinates, but suffers from the aforementioned issue of soft constraint handling.

Besides the numerical issues related to constraints, all of the packaged simulators above use integrators for the dynamics that are not well suited for mechanical systems due to the unrealistic energy behavior of certain explicit or implicit Runge–Kutta methods. Deriving a perfect integrator is generally not trivial [23, 24], but certain classes of integrators exhibit excellent properties, among them symmetric and symplectic methods [25]. An elegant way to derive symplectic integrators for mechanical systems is through a variational perspective, leading to variational integrators. These integrators are derived by discretizing the derivation of the mechanism's differential equations instead of the differential equations themselves. Thus, certain properties such as energy and momentum conservation are maintained [26]. Compared to classical discretizations such as explicit Runge–Kutta methods, larger time steps can be taken due to the increased physical accuracy, and constraint drift is avoided entirely. Computationally efficient variational integrators have been theoretically investigated in minimal coordinates [27–29]. However, the implementations are limited to simple mechanical systems with few joints types and no contact interactions, potentially due to numerical difficulties. Variational and other structure-preserving integrators have also been investigated in redundant coordinates. Here, much effort has been put into variational integrators for systems with explicit constraints [17, 26, 30]. Another investigated approach is nullspace methods which eliminate constraints while remaining structure-preserving [31, 32]. Redundant coordinates are also well suited for structure-preserving integration of flexible multibody systems [33–35]. These presented works deal with advancing the theoretical properties of structure-preserving integrators. In this article, we derive a unified and modular integrator and develop methods for the numerically efficient implementation of such an integrator.

More specifically, the contribution of this article to maximal-coordinate simulators is as follows. Firstly, we derive a variational integrator in maximal coordinates for physically accurate simulations, including good energy behavior and rigid, non-drifting constraints. While variational integrators are generally not new, we derive this integrator in a unified framework for typical dynamics components, including rigid bodies, joints, contacts, friction, actuators and external forces, springs, and dampers. Secondly, we provide an efficient graph-based solver algorithm for the system of nonlinear equations that form the integrator. The solver exploits the sparsity of the nonlinear system of equations to account for the increased number of variables in maximal coordinates. It achieves linear computational complexity in the number of links and joints for mechanisms without kinematic loops and reduces the complexity for mechanisms with kinematic loops. For environment contacts and friction, the solver also achieves linear computational complexity in the number of links and contact points, while reducing the complexity for inter-mechanism contacts. Besides these theoretical contributions for efficient maximal-coordinate simulators based on variational integrators, we also provide an open-source implementation of such a simulator (see Sect. 5), which achieves competitive timing results compared to state-of-the-art simulators.

This article is structured as follows. In Sect. 2, the desired simulator components are mathematically formalized. Based on these components, the variational integrator is derived in Sect. 3, resulting in a system of nonlinear equations. The solver for this system of equations is presented in Sect. 4. An evaluation of the theoretical and numerical properties and application examples are given in Sect. 5, and conclusions are drawn in Sect. 6. The appendices provide background information, including our quaternion notation in Appendix A.

2 Dynamics components

This section formulates the dynamics components for which the variational integrator is derived in Sect. 3. Unit quaternions are used for rotations and orientations due to their computational efficiency (see Appendix A for our quaternion notation). However, other representations, such as rotation matrices, of which the group of unit quaternions is a double cover [36], could be used as well. Figure 1 shows a mechanism with the treated components.

Rigid body In maximal coordinates, each of the n_b rigid bodies in a mechanism has a position $\boldsymbol{x} \in \mathbb{R}^3$ and orientation $\boldsymbol{q} \in \mathbb{H}$ ($\boldsymbol{q}^\mathsf{T}\boldsymbol{q} = 1$), as well as a translational velocity $\boldsymbol{v} \in \mathbb{R}^3$ and angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$. The configuration of a body is denoted as $\boldsymbol{z} = [\boldsymbol{x}^\mathsf{T}\ \boldsymbol{q}^\mathsf{T}]^\mathsf{T}$, and the velocity is denoted as $\dot{\boldsymbol{z}} = [\boldsymbol{v}^\mathsf{T}\ \boldsymbol{\omega}^\mathsf{T}]^\mathsf{T}$. Each body has a mass $m \in \mathbb{R}$ and a symmetric moment of inertia matrix $\boldsymbol{J} \in \mathbb{R}^{3\times 3}$. All quantities refer to the center of mass of a body. Given $\boldsymbol{M} = m\boldsymbol{I}_{3\times 3}$, the kinetic energy of each body is $\mathcal{T}(\dot{\boldsymbol{z}}) = \frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{M}\boldsymbol{v} + \frac{1}{2}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{J}\boldsymbol{\omega}$.

Conservative forces, potentials, and springs Conservative forces in the dynamics are derived from potential functions $\mathcal{V}(\boldsymbol{z}_\mathrm{a}, \boldsymbol{z}_\mathrm{b}, \boldsymbol{z}_\mathrm{c}, \ldots) \in \mathbb{R}$ involving one or multiple bodies. Such potentials can represent, for example, gravity or springs.

Non-conservative forces, actuators, and dampers Non-conservative forces, including actuators and dampers, are directly added to the dynamics as external forces $\mathbf{f} \in \mathbb{R}^3$ or torques $\boldsymbol{\tau} \in \mathbb{R}^3$. Forces are described in the global frame and torques in the body frame. The wrench on a body is denoted as $\mathbf{w} = [\mathbf{f}^\mathsf{T}\ \boldsymbol{\tau}^\mathsf{T}]^\mathsf{T}$. These wrenches are added for each body individually. Actuators fixed at joints are formulated by expressing their wrench in the frames of the connected bodies.

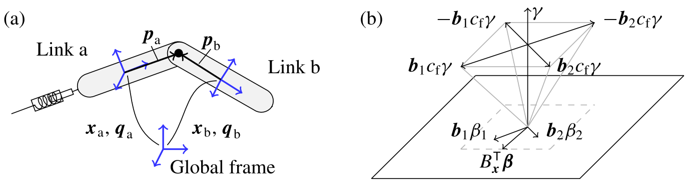

(a)
Link a
$\boldsymbol{p}_\mathrm{a}$
$\boldsymbol{p}_\mathrm{b}$
Link b
$\boldsymbol{x}_\mathrm{a}, \boldsymbol{q}_\mathrm{a}$
$\boldsymbol{x}_\mathrm{b}, \boldsymbol{q}_\mathrm{b}$
Global frame
(b)
$-\boldsymbol{b}_1 c_\mathrm{f}\gamma$
γ
$-\boldsymbol{b}_2 c_\mathrm{f}\gamma$
$\boldsymbol{b}_1 c_\mathrm{f}\gamma$
$\boldsymbol{b}_2 c_\mathrm{f}\gamma$
$\boldsymbol{b}_1\beta_1$
$\boldsymbol{b}_2\beta_2$
$B_x^\mathsf{T}\boldsymbol{\beta}$

Fig. 1 Exemplary depiction of the simulator components. (a) A two-link mechanism with joints, a spring-damper, and a contact subject to friction. (b) A four-sided linearized friction cone

As an example, for an actuator with wrench $\mathbf{w}_{\mathrm{act}} = [\mathbf{f}_{\mathrm{act}}^{\mathsf{T}}\ \boldsymbol{\tau}_{\mathrm{act}}^{\mathsf{T}}]^{\mathsf{T}}$ at a joint between bodies a (parent) and b (child), the resulting wrenches for the bodies are

$$\mathbf{w}_{\mathrm{a}} = \begin{bmatrix} {}^{\mathrm{N}}\mathbf{f}_{\mathrm{a}} \\ {}^{\mathrm{A}}\boldsymbol{\tau}_{\mathrm{a}} \end{bmatrix} = - \begin{bmatrix} {}^{\mathrm{N}}\mathbf{f}_{\mathrm{act}} \\ {}^{\mathrm{A}}\boldsymbol{\tau}_{\mathrm{act}} + {}^{\mathrm{A}}\boldsymbol{p}_{\mathrm{a}} \times {}^{\mathrm{A}}\mathbf{f}_{\mathrm{act}} \end{bmatrix}, \tag{1a}$$

$$\mathbf{w}_{\mathrm{b}} = \begin{bmatrix} {}^{\mathrm{N}}\mathbf{f}_{\mathrm{b}} \\ {}^{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{b}} \end{bmatrix} = \begin{bmatrix} {}^{\mathrm{N}}\mathbf{f}_{\mathrm{act}} \\ {}^{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{act}} + {}^{\mathrm{B}}\boldsymbol{p}_{\mathrm{b}} \times {}^{\mathrm{B}}\mathbf{f}_{\mathrm{act}} \end{bmatrix}, \tag{1b}$$

where N is the global frame, A and B are reference frames of bodies a and b, respectively, and $\boldsymbol{p}$ is a vector from center of mass to actuator.

Joints and equality constraints The joints of a mechanism consisting of one or multiple rigid bodies are represented by differentiable equality constraints $\boldsymbol{g}(z_{\mathrm{a}}, z_{\mathrm{b}}, z_{\mathrm{c}}, \ldots) = \mathbf{0} \in \mathbb{R}^{n_{\mathrm{e}}}$, where n_{e} is the number of equality constraints on the mechanism. Typically, equality constraints are formulated for all i kinematically connected pairs of rigid bodies independently and subsequently stacked into one constraint function $\boldsymbol{g} = [\boldsymbol{g}_1^{\mathsf{T}} \cdots \boldsymbol{g}_i^{\mathsf{T}}]^{\mathsf{T}}$. In maximal coordinates, two generic equality constraint functions, one for the translational and one for the rotational movement of the two connected bodies, can be combined to create most of the common joints encountered in mechanisms. This insight greatly simplifies analytic gradient calculations for fast computations and gives direct access to minimal coordinates as well (see Appendix B for a list of possible joints and the recovery of minimal coordinates).

Contacts and inequality constraints Rigid contacts between multiple bodies and with the environment are represented by differentiable inequality constraints $\boldsymbol{\phi}(z_{\mathrm{a}}, z_{\mathrm{b}}, z_{\mathrm{c}}, \ldots) \geq \mathbf{0} \in \mathbb{R}^{n_{\mathrm{i}}}$, where n_{i} is the number of inequality constraints on the mechanism. As for joints, inequality constraints are typically formulated independently for all j pairs of rigid bodies from their signed distance function and subsequently stacked into one constraint function $\boldsymbol{\phi} = [\phi_1^{\mathsf{T}} \cdots \phi_j^{\mathsf{T}}]^{\mathsf{T}}$.

As an example, ground contact of a single point contact on a body in maximal coordinates always has the form $\phi(z) = \boldsymbol{e}_{\mathrm{z}}^{\mathsf{T}}(\boldsymbol{x} + \boldsymbol{q} \cdot \boldsymbol{p} \cdot \boldsymbol{q}^{-1}) \geq 0$, where $\boldsymbol{e}_{\mathrm{z}}$ is the z-axis unit vector in the global frame and $\boldsymbol{p} \in \mathbb{R}^3$ points from center of mass to contact point in the body's frame. As before, this consistent structure allows for the calculation of analytic gradients to reduce computation time.

Static and sliding friction In the dynamics, static and sliding friction are added as external forces on the bodies. These forces are calculated by solving an optimization problem derived from the maximum dissipation principle [37]. We are using a linearized friction cone [38], although a nonlinear friction cone could be used as well. The maximum dissipation principle states that the energy dissipation rate of the bodies in contact is maximized by a friction force $\boldsymbol{\beta} \in \mathbb{R}^{n_{\mathrm{c}} n_{\mathrm{f}}}$, where n_{c} is the number of contacts and n_{f} is the even number of basis vectors of the friction cone. The basis vectors $\boldsymbol{b}_i \in \mathbb{R}^3$ of the linearized friction cone are depicted in Fig. 1 (b). A detailed derivation of the optimization problem is stated in Appendix C. The optimization problem resulting from the maximum dissipation principle for the bodies of a mechanism is

$$\min_{\boldsymbol{\beta}} \quad \begin{bmatrix} \dot{z}_{\mathrm{a}}^{\mathsf{T}} & \dot{z}_{\mathrm{b}}^{\mathsf{T}} & \dot{z}_{\mathrm{c}}^{\mathsf{T}} & \cdots \end{bmatrix} \boldsymbol{B}\,(z_{\mathrm{a}}, z_{\mathrm{b}}, z_{\mathrm{c}}, \ldots)^{\mathsf{T}}\, \boldsymbol{\beta}, \tag{2a}$$

$$\text{s.t.} \quad \boldsymbol{E}^{\mathsf{T}}\boldsymbol{\beta} \leq \boldsymbol{C}_{\mathrm{f}}\boldsymbol{\gamma}, \tag{2b}$$

$$\boldsymbol{\beta} \geq \mathbf{0}, \tag{2c}$$

where $\boldsymbol{B} \in \mathbb{R}^{6n_\mathrm{b} \times n_\mathrm{c} n_\mathrm{f}}$ maps the n_f-dimensional friction forces at each of the n_c contact point to a six-dimensional wrench on the respective bodies. The constraint $\boldsymbol{E}^\mathsf{T}\boldsymbol{\beta} \leq \boldsymbol{C}_\mathrm{f}\boldsymbol{\gamma}$ describes the limit on the friction forces with $\boldsymbol{E} = \operatorname{diag}(\mathbf{1}, \dots, \mathbf{1}) \in \mathbb{R}^{n_\mathrm{c} n_\mathrm{f} \times n_\mathrm{c}}$, $\mathbf{1} = [1 \ \cdots \ 1]^\mathsf{T} \in \mathbb{R}^{n_\mathrm{f}}$, normal forces $\boldsymbol{\gamma} \in \mathbb{R}^{n_\mathrm{c}}$, and the friction coefficient matrix $\boldsymbol{C}_\mathrm{f} = \operatorname{diag}(c_{\mathrm{f},1}, \dots, c_{\mathrm{f},n_\mathrm{c}}) \in \mathbb{R}^{n_\mathrm{c} \times n_\mathrm{c}}$ containing the friction coefficients c_f for each contact point. The friction wrenches for each body resulting from the optimization are added as external wrenches to the dynamics.

As an example, the optimization problem for ground contact of a single point contact on a body in maximal coordinates always has the form

$$\min_{\boldsymbol{\beta}} \ \dot{\boldsymbol{z}}^\mathsf{T}\boldsymbol{B}^\mathsf{T}\boldsymbol{\beta}, \tag{3a}$$

$$\text{s.t.} \quad \mathbf{1}^\mathsf{T}\boldsymbol{\beta} \leq c_\mathrm{f}\gamma, \tag{3b}$$

$$\boldsymbol{\beta} \geq \mathbf{0}, \tag{3c}$$

with $\boldsymbol{B}^\mathsf{T} = [\boldsymbol{B}_{\boldsymbol{x}} \ \boldsymbol{B}_{\boldsymbol{q}}]^\mathsf{T}$ consisting of

$$\boldsymbol{B}_{\boldsymbol{x}}^\mathsf{T} = \begin{bmatrix} \boldsymbol{b}_1 & -\boldsymbol{b}_1 & \cdots & \boldsymbol{b}_{\frac{n_\mathrm{f}}{2}} & -\boldsymbol{b}_{\frac{n_\mathrm{f}}{2}} \end{bmatrix} \in \mathbb{R}^{3 \times n_\mathrm{f}}, \tag{4a}$$

$$\boldsymbol{B}_{\boldsymbol{q}}^\mathsf{T} = \boldsymbol{p}^\times \boldsymbol{Q}(\boldsymbol{q}) \boldsymbol{B}_{\boldsymbol{x}}^\mathsf{T} \in \mathbb{R}^{3 \times n_\mathrm{f}}, \tag{4b}$$

where the vector from the body's center of mass to the contact point is denoted $\boldsymbol{p}$, the $^\times$ operator creates the skew-symmetric matrix from this vector, and $\boldsymbol{Q}(\boldsymbol{q})$ is the rotation matrix for quaternion $\boldsymbol{q}$.

3 Mathematical integrator

A first-order variational (symplectic) integrator [26] is derived for the simulator components described in the previous section. This integrator discretizes the rigid-body dynamics while maintaining energy and momentum conservation properties as well as constraint satisfaction. Higher-order variational integrators are possible [39, 40], and we restrict the derivation to the first order for clarity. First, the derivation for unconstrained dynamics is provided. Afterward, equality and inequality constraints are added to the integrator, and finally, friction dynamics are incorporated.

3.1 Unconstrained dynamics

The derivation of the integrator for unconstrained dynamics is split into translational and rotational components for clarity. Note that the derivation also holds for coupled translational and rotational dynamics. Variational integrators are based on the principle of least action, which states that a mechanical system takes the path of least action when going from a fixed starting point to a fixed ending point. External forces and torques are incorporated with the Lagrange-d'Alembert principle. Action has the dimensions [Energy] × [Time] and the unconstrained action integral S_0 with is defined as

$$S_0(\boldsymbol{z}, \dot{\boldsymbol{z}}) = \int_{t_0}^{t_N} \mathcal{L}(\boldsymbol{z}, \dot{\boldsymbol{z}}) \,\mathrm{d}t + \int_{t_0}^{t_N} \begin{bmatrix} \mathbf{f}^\mathsf{T} & 2\boldsymbol{L}(\boldsymbol{q})\boldsymbol{V}^\mathsf{T}\boldsymbol{\tau} \end{bmatrix}^\mathsf{T} \boldsymbol{z} \,\mathrm{d}t, \tag{5}$$

where $\mathcal{L} = \mathcal{T} - \mathcal{V}$ is the Lagrangian with kinetic energy $\mathcal{T}$ and potential energy $\mathcal{V}$. A brief explanation of the external force and torque components in (5), as well as our quaternion notation, are given in Appendix A. Further details on virtual work for quaternions can be found in [41, 42].

Translational component The translational component of the action integral (5) is

$$S_{0,\mathrm{T}}(\boldsymbol{x}, \boldsymbol{v}) = \int_{t_0}^{t_N} \mathcal{L}_\mathrm{T}(\boldsymbol{x}, \boldsymbol{v}) \, \mathrm{d}t + \int_{t_0}^{t_N} \mathbf{f}^\mathsf{T} \boldsymbol{x} \, \mathrm{d}t. \tag{6}$$

For numerical integration, (6) is discretized. A first-order discretization of the integral and a first-order approximation of the velocity,

$$\boldsymbol{v}_k = \frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t}, \tag{7}$$

with step size Δt is used to obtain the discrete action sum

$$S_{\mathrm{d},0,\mathrm{T}}(\boldsymbol{x}_k, \boldsymbol{v}_k) = \sum_{k=0}^{N-1} \left(\mathcal{L}_\mathrm{T}(\boldsymbol{x}_k, \boldsymbol{v}_k) + \mathbf{f}_k^\mathsf{T} \boldsymbol{x}_k \right) \Delta t. \tag{8}$$

For a first-order integrator, the principle of least action must be fulfilled for trajectories consisting of three knot points, i.e., from 0 to $N = 2$. Since $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$ are fixed start and end points of the trajectory, only $\boldsymbol{x}_1$ can vary. Therefore, the action sum is minimized with respect to the position $\boldsymbol{x}_1$:

$$\nabla_{\boldsymbol{x}_1} S_{\mathrm{d},0,\mathrm{T}} = -\boldsymbol{d}_{0,\mathrm{T}} \Delta t = \boldsymbol{0}, \tag{9}$$

where $\boldsymbol{d}_{0,\mathrm{T}}$ are the resulting implicit discretized translational dynamics. In other words, if Equation (9) is fulfilled, we have found the discrete approximation of the physically correct trajectory consisting of the knot points $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$. Note that the derivative with respect to $\boldsymbol{x}_1$ is taken for each body in a mechanism.

An example for a single body with potential function $\mathcal{V}(\boldsymbol{x})$ yields

$$\begin{aligned} \boldsymbol{d}_{0,\mathrm{T}}(\boldsymbol{v}_1) &= -\nabla_{\boldsymbol{x}_1} \left(\mathcal{L}_\mathrm{T}(\boldsymbol{x}_0, \boldsymbol{v}_0) + \mathbf{f}_0^\mathsf{T} \boldsymbol{x}_0 + \mathcal{L}_\mathrm{T}(\boldsymbol{x}_1, \boldsymbol{v}_1) + \mathbf{f}_1^\mathsf{T} \boldsymbol{x}_1 \right) \\ &= -\left(\boldsymbol{M} \frac{\boldsymbol{x}_1 - \boldsymbol{x}_0}{\Delta t} - \boldsymbol{M} \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{\Delta t} - \nabla_{\boldsymbol{x}_1} \mathcal{V}(\boldsymbol{x}_1) + \mathbf{f}_1 \right) \\ &= \boldsymbol{M} \frac{\boldsymbol{v}_1 - \boldsymbol{v}_0}{\Delta t} + \nabla_{\boldsymbol{x}_1} \mathcal{V}(\boldsymbol{x}_1) - \mathbf{f}_1 = \boldsymbol{0}, \end{aligned} \tag{10}$$

which resembles the discretized version of Newton's second law, $\boldsymbol{M}\dot{\boldsymbol{v}} - \mathbf{f} = \boldsymbol{0}$.

The physically accurate dynamics are obtained by varying $\boldsymbol{x}_1$ with fixed $\boldsymbol{x}_0$ and $\boldsymbol{x}_2$. However, when integrating dynamics forward in time, we start from a known initial state $\boldsymbol{x}_0$ and $\boldsymbol{v}_0$. From (7), the position at the next time step, $\boldsymbol{x}_1$, is calculated as

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{v}_0 \Delta t. \tag{11}$$

Then, the implicit dynamics equations (9) are solved to obtain the velocity $\boldsymbol{v}_1$. This resulting integration scheme, consisting of (11) and (9), is the symplectic Euler method [25].

Rotational component The rotational component of the integrator can be derived similarly to the translation case. A description for an unconstrained floating single rigid body is presented in [43], and we later extend the derivation to constrained multibody systems.

The action integral for the rotational component is

$$S_{0,\mathrm{R}}(\boldsymbol{q},\boldsymbol{\omega}) = \int_{t_0}^{t_N} \mathcal{L}_\mathrm{R}(\boldsymbol{q},\boldsymbol{\omega})\,\mathrm{d}t + \int_{t_0}^{t_N} 2\boldsymbol{\tau}^\mathsf{T}\boldsymbol{V}\boldsymbol{L}(\boldsymbol{q})^\mathsf{T}\boldsymbol{q}\,\mathrm{d}t. \tag{12}$$

To maintain unit norm in the quaternion update (see Appendix A), the discrete quaternion angular velocity is defined as

$$\bar{\boldsymbol{\omega}}_k = \begin{bmatrix} \sqrt{\left(\frac{2}{\Delta t}\right)^2 - \boldsymbol{\omega}_k^\mathsf{T}\boldsymbol{\omega}_k} \\ \boldsymbol{\omega}_k \end{bmatrix} = \frac{2}{\Delta t}\boldsymbol{L}(\boldsymbol{q}_k)^\mathsf{T}\boldsymbol{q}_{k+1}. \tag{13}$$

As before, the action integral (12) is discretized to obtain the action sum

$$S_{\mathrm{d},0,\mathrm{R}}(\boldsymbol{q}_k,\boldsymbol{\omega}_k) = \sum_{k=0}^{N-1}\left(\mathcal{L}_\mathrm{R}(\boldsymbol{q}_k,\boldsymbol{\omega}_k) + 2\boldsymbol{\tau}_k^\mathsf{T}\boldsymbol{V}\boldsymbol{L}(\boldsymbol{q}_k)^\mathsf{T}\boldsymbol{q}_k\right)\Delta t. \tag{14}$$

The principle of least action is fulfilled by minimizing the discrete action sum from 0 to $N=2$ over the orientation $\boldsymbol{q}_1$:

$$\nabla^\mathrm{r}_{\boldsymbol{q}_1} S_{\mathrm{d},0,\mathrm{R}} = -\boldsymbol{d}_{0,\mathrm{R}}\Delta t = \boldsymbol{0}, \tag{15}$$

where $\boldsymbol{d}_{0,\mathrm{R}}$ are the implicit discretized rotational dynamics. Note that we have used the rotational gradient ∇^r (see Appendix A) and that the derivative with respect to $\boldsymbol{q}_1$ is taken for each body in a mechanism.

An example for a single body with potential function $\mathcal{V}(\boldsymbol{q})$ yields

$$\begin{aligned}\boldsymbol{d}_{0,\mathrm{R}}(\boldsymbol{\omega}_1) = \boldsymbol{J}\boldsymbol{\omega}_1\sqrt{\tfrac{4}{\Delta t^2} - \boldsymbol{\omega}_1^\mathsf{T}\boldsymbol{\omega}_1} + \boldsymbol{\omega}_1^\times\boldsymbol{J}\boldsymbol{\omega}_1 - \\ \boldsymbol{J}\boldsymbol{\omega}_0\sqrt{\tfrac{4}{\Delta t^2} - \boldsymbol{\omega}_0^\mathsf{T}\boldsymbol{\omega}_0} + \boldsymbol{\omega}_0^\times\boldsymbol{J}\boldsymbol{\omega}_0 + \nabla^\mathrm{r}_{\boldsymbol{q}_1}\mathcal{V}(\boldsymbol{q}_1) - 2\boldsymbol{\tau}_2 = \boldsymbol{0},\end{aligned} \tag{16}$$

which—analogous to the translational case—bears resemblance to Euler's equations for rotations $\boldsymbol{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega}^\times\boldsymbol{J}\boldsymbol{\omega} - \boldsymbol{\tau} = \boldsymbol{0}$.

An integration step given $\boldsymbol{q}_0$ and $\boldsymbol{\omega}_0$ is performed by first calculating

$$\boldsymbol{q}_1 = \frac{\Delta t}{2}\boldsymbol{L}(\boldsymbol{q}_0)\bar{\boldsymbol{\omega}}_0, \tag{17}$$

and subsequently solving (15) for the angular velocity $\boldsymbol{\omega}_1$.

3.2 Equality constrained dynamics

Equality constraint functions $\boldsymbol{g}(z)$ with Lagrange multiplier $\boldsymbol{\lambda} \in \mathbb{R}^{n_\mathrm{e}}$ are added to the integrator by appending them to the action integral:

$$S(z,\dot{z},\boldsymbol{\lambda}) = S_0(z,\dot{z}) + \int_{t_0}^{t_N}\boldsymbol{\lambda}^\mathsf{T}\boldsymbol{g}(z)\,\mathrm{d}t. \tag{18}$$

Accordingly, the discrete action sum changes to

$$S_{\mathrm{d}}(z_k, \dot{z}_k, \boldsymbol{\lambda}_k) = S_{\mathrm{d},0}(z_k, \dot{z}_k) + \sum_{k=0}^{N-1} \boldsymbol{\lambda}_k^{\mathsf{T}} \boldsymbol{g}(z_k)\Delta t. \tag{19}$$

Taking the gradient of (19) with respect to $\boldsymbol{x}_1$ and $\boldsymbol{q}_1$ yields the constrained implicit discretized dynamics

$$\boldsymbol{d}(\dot{z}_1, \boldsymbol{\lambda}_1) = \boldsymbol{d}_0(\dot{z}_1) - \boldsymbol{G}(z_1)^{\mathsf{T}}\boldsymbol{\lambda}_1 = \mathbf{0}, \tag{20a}$$

$$\boldsymbol{g}\left(z_2\left(\dot{z}_1\right)\right) = \mathbf{0}, \tag{20b}$$

where

$$\boldsymbol{G}(z) = \begin{bmatrix} \frac{\partial \boldsymbol{g}(z)}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{g}(z)}{\partial^{\mathrm{r}} \boldsymbol{q}} \end{bmatrix}. \tag{21}$$

Physically, the constraint forces $\boldsymbol{G}(z_1)^{\mathsf{T}}\boldsymbol{\lambda}_1$ act on the rigid bodies to guarantee satisfaction of constraints $\boldsymbol{g} = \mathbf{0}$. Mathematically, $\boldsymbol{\lambda}$ serves a similar purpose as Lagrange multipliers in constrained optimization.

The integration step starting from z_0 and $\dot{z}_0$ is calculated as follows. First, z_1 is calculated from the update rules (11) and (17). Then, the nonlinear system of equations (20a)–(20b) is solved for $\dot{z}_1$ and $\boldsymbol{\lambda}_1$. Note that the constraints are fulfilled for z_2, which depends on $\dot{z}_1$ through update rules (11) and (17). Therefore, the resulting velocity $\dot{z}_1$ always ensures constraint satisfaction for the next position z_2.

3.3 Inequality constrained dynamics

Inequality constraint functions $\boldsymbol{\phi}(z)$ with Lagrange multipliers $\boldsymbol{\gamma} \in \mathbb{R}^{n_{\mathrm{i}}}$ are added to the integrator in a similar fashion. Physically, the multipliers $\boldsymbol{\gamma}$ are the magnitudes of the normal forces at the contacts. To add the constraints to the dynamics, they are discretized and formulated as a nonlinear complementarity problem (NCP)

$$\boldsymbol{\phi}(z_k) \geq \mathbf{0}, \tag{22a}$$

$$\boldsymbol{\gamma}_k \geq \mathbf{0}, \tag{22b}$$

$$\boldsymbol{\phi}(z_k)^{\mathsf{T}}\boldsymbol{\gamma}_k = \mathbf{0}, \tag{22c}$$

with element-wise $\geq$, for which we use the standard shorthand notation

$$\mathbf{0} \leq \boldsymbol{\phi}(z_k) \perp \boldsymbol{\gamma}_k \geq \mathbf{0}. \tag{23}$$

The resulting dynamics are

$$\boldsymbol{d}(\dot{z}_1, \boldsymbol{\gamma}_1) = \boldsymbol{d}_0(\dot{z}_1) - \boldsymbol{N}(z_1)^{\mathsf{T}}\boldsymbol{\gamma}_1 = \mathbf{0}, \tag{24a}$$

$$\mathbf{0} \leq \boldsymbol{\phi}(z_2(\dot{z}_1)) \perp \boldsymbol{\gamma}_1 \geq \mathbf{0}, \tag{24b}$$

where

$$\boldsymbol{N}(z) = \begin{bmatrix} \frac{\partial \boldsymbol{\phi}(z)}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{\phi}(z)}{\partial^{\mathrm{r}} \boldsymbol{q}} \end{bmatrix}. \tag{25}$$

The integration step starting from z_0 and $\dot{z}_0$ is again performed by first calculating z_1 from (11) and (17), and then solving (24a)–(24b) for $\dot{z}_1$ and $\boldsymbol{\gamma}_1$.

3.4 Friction dynamics

The friction dynamics are included in the variational integrator by discretizing the maximum dissipation principle (2a)–(2c):

$$\min_{\boldsymbol{\beta}} \ \dot{z}_k^\top \boldsymbol{B}(z_k)^\top \boldsymbol{\beta}_k, \tag{26a}$$

$$\text{s.t.} \quad \boldsymbol{E}^\top \boldsymbol{\beta}_k \leq \boldsymbol{C}_\mathrm{f} \boldsymbol{\gamma}_k, \tag{26b}$$

$$\boldsymbol{\beta}_k \geq \mathbf{0}. \tag{26c}$$

Note that the normal force multipliers $\boldsymbol{\gamma}$ result from contact inequality constraints in the form of (22a)–(22c).

As with the contact constraint, we formulate (26a)–(26c) as an NCP with Lagrange multipliers $\boldsymbol{\psi}_k \in \mathrm{R}^{n_\mathrm{c}}$—the tangential velocities at the contact points—and $\boldsymbol{\eta}_k \in \mathbb{R}^{n_\mathrm{c} n_\mathrm{f}}$:

$$\boldsymbol{B}(z_k)\dot{z}_k + \boldsymbol{E}\boldsymbol{\psi}_k - \boldsymbol{\eta}_k = \mathbf{0}, \tag{27a}$$

$$\mathbf{0} \leq \boldsymbol{C}_\mathrm{f}\boldsymbol{\gamma}_k - \boldsymbol{E}^\top \boldsymbol{\beta}_k \perp \boldsymbol{\psi}_k \geq \mathbf{0}, \tag{27b}$$

$$\mathbf{0} \leq \boldsymbol{\beta}_k \perp \boldsymbol{\eta}_k \geq \mathbf{0}. \tag{27c}$$

Given a mechanism with n_c contact inequality constraints with friction, the resulting dynamics are

$$\boldsymbol{d}(\dot{z}_1, \boldsymbol{\gamma}_1, \boldsymbol{\beta}_1, \boldsymbol{\psi}_1, \boldsymbol{\eta}_1) = \boldsymbol{d}_0(\dot{z}_1) - \boldsymbol{N}(z_1)^\top \boldsymbol{\gamma}_1 - \boldsymbol{B}(z_1)^\top \boldsymbol{\beta}_1 = \mathbf{0}, \tag{28a}$$

$$\mathbf{0} \leq \boldsymbol{\phi}(z_2(\dot{z}_1)) \perp \boldsymbol{\gamma}_1 \geq \mathbf{0}, \tag{28b}$$

$$\boldsymbol{B}(z_1)\dot{z}_1 + \boldsymbol{E}\boldsymbol{\psi}_1 - \boldsymbol{\eta}_1 = \mathbf{0}, \tag{28c}$$

$$\mathbf{0} \leq \boldsymbol{C}_\mathrm{f}\boldsymbol{\gamma}_1 - \boldsymbol{E}^\top \boldsymbol{\beta}_1 \perp \boldsymbol{\psi}_1 \geq \mathbf{0}, \tag{28d}$$

$$\mathbf{0} \leq \boldsymbol{\beta}_1 \perp \boldsymbol{\eta}_1 \geq \mathbf{0}, \tag{28e}$$

and the integration step starting from z_0 and $\dot{z}_0$ is performed by first calculating z_1 from (11) and (17), and then solving (28a)–(28e) for $\dot{z}_1$, $\boldsymbol{\gamma}_1$, $\boldsymbol{\beta}_1$, $\boldsymbol{\psi}_1$, and $\boldsymbol{\eta}_1$.

3.5 Complete integrator

Putting all components together, the simulation of one time step given z_0 and $\dot{z}_0$ is performed by first calculating z_1 from (11) and (17). Subsequently, the constrained implicit dynamics must be solved:

$$\boldsymbol{d}(\dot{z}_1, \boldsymbol{\lambda}_1, \boldsymbol{\gamma}_1, \boldsymbol{\beta}_1, \boldsymbol{\psi}_1, \boldsymbol{\eta}_1) = \boldsymbol{d}_0(\dot{z}_1) - \boldsymbol{G}^\top \boldsymbol{\lambda}_1 - \boldsymbol{N}^\top \boldsymbol{\gamma}_1 - \boldsymbol{B}^\top \boldsymbol{\beta}_1 = \mathbf{0}, \tag{29a}$$

$$\boldsymbol{g}\left(z_2(\dot{z}_1)\right) = \mathbf{0}, \tag{29b}$$

$$\mathbf{0} \leq \boldsymbol{\phi}\left(z_2(\dot{z}_1)\right) \perp \boldsymbol{\gamma}_1 \geq \mathbf{0}, \tag{29c}$$

$$\boldsymbol{B}\dot{z}_1 + \boldsymbol{E}\boldsymbol{\psi}_1 - \boldsymbol{\eta}_1 = \mathbf{0}, \tag{29d}$$

$$\mathbf{0} \leq \boldsymbol{C}_\mathrm{f}\boldsymbol{\gamma}_1 - \boldsymbol{E}^\top \boldsymbol{\beta}_1 \perp \boldsymbol{\psi}_1 \geq \mathbf{0}, \tag{29e}$$

$$\mathbf{0} \leq \boldsymbol{\beta}_1 \perp \boldsymbol{\eta}_1 \geq \mathbf{0}. \tag{29f}$$

Note that (29a)–(29f) contains the equations and constraints for all bodies of a mechanism.

If the initial velocity $\dot{z}_0$ is unknown, and a non-constraint-fulfilling one is chosen, an error occurs at the first time step. The magnitude of the error depends on the magnitude of the initial constraint violation. The discrete Legendre transform can be used to determine a constraint-fulfilling initial velocity. We refer to the extensive explanation in [30] for details on initializing a simulation with the Legendre transform.

4 Numerical solver

The variational integrator (29a)–(29f) can be summarized as a system of nonlinear equations with inequality constraints:

$$\boldsymbol{f}(\boldsymbol{s}) = \boldsymbol{0}, \tag{30a}$$

$$\boldsymbol{h}(\boldsymbol{s}) \geq \boldsymbol{0}, \tag{30b}$$

with solution vector $\boldsymbol{s} = [\dot{\boldsymbol{z}}_1^\mathsf{T}\ \boldsymbol{\lambda}_1^\mathsf{T}\ \boldsymbol{\gamma}_1^\mathsf{T}\ \boldsymbol{\beta}_1^\mathsf{T}\ \boldsymbol{\psi}_1^\mathsf{T}\ \boldsymbol{\eta}_1^\mathsf{T}]^\mathsf{T}$. In this section, the algorithms derived for solving the system (30a)–(30b) are applicable to the class of Newton-based root-finding methods. Certain methods of this class, for example, interior-point methods [44], introduce slack variables $\boldsymbol{\sigma}$ and additional constraints $\boldsymbol{h}(\boldsymbol{s}) = \boldsymbol{\sigma}$ to facilitate the numerical treatment of the inequality constraints. Since the slack variables match the inequality constraints up to the desired solution tolerance, the theoretical properties of the integrator still hold. The solution vector $\boldsymbol{s}$ and the system $\boldsymbol{f}(\boldsymbol{s})$ would be extended by these slack variables and constraints, but the computational complexity does not change if each inequality constraint and its associated slack constraint are considered as a single node in a graph. Moreover, the graph-based argument is not limited to mechanical systems and can be implemented for any graph-based system, but we will restrict the discussion to mechanical systems.

At the core, Newton-based methods iteratively produce solution approximations for (30a)–(30b) with the procedure

$$\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)} - \boldsymbol{F}(\boldsymbol{s}^{(i)})^{-1}\boldsymbol{f}(\boldsymbol{s}^{(i)}), \tag{31}$$

where

$$\boldsymbol{F}(\boldsymbol{s}) = \frac{\partial \boldsymbol{f}(\boldsymbol{s})}{\partial \boldsymbol{s}}. \tag{32}$$

Numerically, (31) is formulated as a linear system of equations

$$\boldsymbol{F}(\boldsymbol{s}^{(i)})\Delta\boldsymbol{s}^{(i)} = -\boldsymbol{f}(\boldsymbol{s}^{(i)}), \tag{33}$$

where the result $\Delta\boldsymbol{s}^{(i)}$ is used to obtain $\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)} + \Delta\boldsymbol{s}^{(i)}$.

Linear systems of the form (33) are solved with decomposition and backsubstitution with overall cubic computational complexity $\mathcal{O}(n^3)$. For general integrators, including the variational integrator derived in the previous section, (33) is neither symmetric nor block symmetric. Therefore, the LDU decomposition [45] for asymmetric systems is chosen as the foundation of the algorithms. While (33) is asymmetric, its sparsity pattern, i.e., the zero and non-zero entries, is block-symmetric, and the following algorithms exploit this sparsity to improve complexity.

4.1 Linear-complexity algorithm

For mechanisms without kinematic loops, the LDU decomposition can be modified to obtain decomposition and backsubstitution with linear computational complexity $\mathcal{O}(n)$, where n is the number of nodes in the corresponding graph. This modification is achieved by taking into account the graph representing the components of a mechanism and its associated $\boldsymbol{F}$ matrix. Consider, for example, the mechanism and graph in Fig. 2.

According to [46], the $\boldsymbol{F}$ matrix corresponding to an acyclic graph, i.e., a mechanism without kinematic loops, can be decomposed with linear complexity by traversing the graph from leaves to root.

A depth-first search (DFS) starting from the (arbitrary) root is performed to find the correct processing order. The found nodes are stored in a list with the root as the last element and the last-found node as the first element. This list is then used in the modified LDU decomposition (Algorithm 1) and backsubstitution (Algorithm 2). Note that in the algorithms, vector indices i stand for the respective rows of node i, and matrix indices i, j stand for the respective rows of node i and columns of node j.

The decomposition in Algorithm 1 processes the matrix $\boldsymbol{F}$ according to the graph structure from leaves to root. Additionally, for each node, computations are only performed for

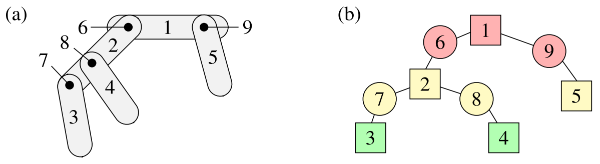


Fig. 2 (a) A mechanism with five links and four joints. (b) A graph representing the mechanism and its matrix. Squares represent links, circles represent joints. Coloring represents different levels of the tree-shaped graph

Algorithm 1 Sparse in-place LDU decomposition, complexity $\mathcal{O}(n)$

```
1: for $i \in$ list do                                        ▷ list from DFS
2:     for $c \in$ children($i$) do                           ▷ children from DFS
3:         $\boldsymbol{F}_{i,c} \leftarrow \boldsymbol{F}_{i,c}\boldsymbol{F}_{c,c}^{-1}$
4:         $\boldsymbol{F}_{c,i} \leftarrow \boldsymbol{F}_{c,c}^{-1}\boldsymbol{F}_{c,i}$
5:         $\boldsymbol{F}_{i,i} \leftarrow \boldsymbol{F}_{i,i} - \boldsymbol{F}_{i,c}\boldsymbol{F}_{c,c}\boldsymbol{F}_{c,i}$
6:     end for
7: end for
```

Algorithm 2 Sparse in-place LDU backsubstitution, complexity $\mathcal{O}(n)$

```
1: for $i \in$ list do
2:     $\Delta\boldsymbol{s}_i \leftarrow -\boldsymbol{f}_i$
3:     for $c \in$ children($i$) do
4:         $\Delta\boldsymbol{s}_i \leftarrow \Delta\boldsymbol{s}_i - \boldsymbol{F}_{i,c}\Delta\boldsymbol{s}_c$
5:     end for
6: end for
7: for $i \in$ reverse(list) do
8:     $\Delta\boldsymbol{s}_i \leftarrow \boldsymbol{F}_{i,i}^{-1}\Delta\boldsymbol{s}_i$
9:     $\Delta\boldsymbol{s}_i \leftarrow \Delta\boldsymbol{s}_i - \boldsymbol{F}_{c,\text{parent}(i)}\Delta\boldsymbol{s}_{\text{parent}(i)}$      ▷ ignore if parent($i$) = ∅
10: end for
```

connected components, since computations for disconnected components are zero in the LDU decomposition. The linear complexity is a direct result.

Decomposition complexity: In an acyclic graph with n nodes, each node has at most one parent, so there are $\mathcal{O}(n)$ children (and $\mathcal{O}(n)$ parents). Therefore, a total of $\mathcal{O}(n)$ evaluations of the for-loop on line 2 of Algorithm 1 are required. The result is a linear complexity $\mathcal{O}(n)$.

Backsubstitution complexity: In an acyclic graph with n nodes, there are $\mathcal{O}(n)$ children and $\mathcal{O}(n)$ parents. Therefore, a total of $\mathcal{O}(n)$ evaluations of the for-loops on lines 3 and 9 of Algorithm 2 are required. The result is a linear complexity $\mathcal{O}(n)$.

Articulated mechanism example The comparison of dense and sparse LDU decomposition for the example in Fig. 2 is shown in Fig. 3.

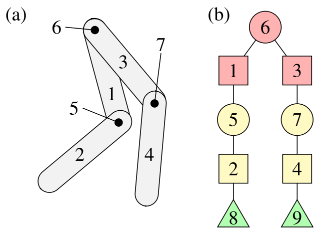
The matrices in Fig. 3 have off-diagonal entries only at the intersection of directly connected nodes. For the example mechanism in Fig. 2, the diagonal entries $\boldsymbol{D}_1$ to $\boldsymbol{D}_5$ are the derivatives of the dynamics $\boldsymbol{d}$ of each body, and the off-diagonal entries $\boldsymbol{c}_{ij}$ are the equality constraint derivatives representing joints between two bodies. Note that when processing the matrix in the wrong order, a so-called fill-in is created. This fill-in occurs at the off-diagonals of nodes indirectly connected through a node that is processed before it becomes a leaf. Since fill-in must also be processed in the decomposition and backsubstitution, linear complexity is no longer achieved.

Environment contact example A direct result of the linear-complexity property for acyclic graphs is the following. Mechanisms without kinematic loops and only environment contact, i.e., no contact between bodies, correspond to acyclic graphs and, therefore, have linear complexity in the number of bodies and contact points. Consider the exemplary mechanism, graph, and matrix in Fig. 4.

(a)
$$\begin{pmatrix}
\boldsymbol{D}_1 & 0 & 0 & 0 & 0 & \boldsymbol{c}'_{61} & 0 & 0 & \boldsymbol{c}'_{91}\\
0 & \boldsymbol{D}_2 & 0 & 0 & 0 & \boldsymbol{c}'_{62} & \boldsymbol{c}'_{72} & \boldsymbol{c}'_{82} & 0\\
0 & 0 & \boldsymbol{D}_3 & 0 & 0 & 0 & \boldsymbol{c}'_{73} & 0 & 0\\
0 & 0 & 0 & \boldsymbol{D}_4 & 0 & 0 & 0 & \boldsymbol{c}'_{84} & 0\\
0 & 0 & 0 & 0 & \boldsymbol{D}_5 & 0 & 0 & 0 & \boldsymbol{c}'_{95}\\
\boldsymbol{c}_{61} & \boldsymbol{c}_{62} & 0 & 0 & 0 & \boldsymbol{D}_6 & \bullet & \bullet & \bullet\\
0 & \boldsymbol{c}_{72} & \boldsymbol{c}_{73} & 0 & 0 & \bullet & \boldsymbol{D}_7 & \bullet & \bullet\\
0 & \boldsymbol{c}_{82} & 0 & \boldsymbol{c}_{84} & 0 & \bullet & \bullet & \boldsymbol{D}_8 & \bullet\\
\boldsymbol{c}_{91} & 0 & 0 & 0 & \boldsymbol{c}_{95} & \bullet & \bullet & \bullet & \boldsymbol{D}_9
\end{pmatrix}$$

(b)
$$\begin{pmatrix}
\boldsymbol{D}_5 & \boldsymbol{c}'_{95} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\boldsymbol{c}_{95} & \boldsymbol{D}_9 & 0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{c}_{91}\\
0 & 0 & \boldsymbol{D}_4 & \boldsymbol{c}'_{84} & 0 & 0 & 0 & 0 & 0\\
0 & 0 & \boldsymbol{c}_{84} & \boldsymbol{D}_8 & 0 & 0 & \boldsymbol{c}_{82} & 0 & 0\\
0 & 0 & 0 & 0 & \boldsymbol{D}_3 & \boldsymbol{c}'_{73} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \boldsymbol{c}_{73} & \boldsymbol{D}_7 & \boldsymbol{c}_{72} & 0 & 0\\
0 & 0 & 0 & \boldsymbol{c}'_{82} & 0 & \boldsymbol{c}'_{72} & \boldsymbol{D}_2 & \boldsymbol{c}'_{62} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{c}_{62} & \boldsymbol{D}_6 & \boldsymbol{c}_{61}\\
0 & \boldsymbol{c}'_{91} & 0 & 0 & 0 & 0 & 0 & \boldsymbol{c}'_{61} & \boldsymbol{D}_1
\end{pmatrix}$$

Fig. 3 Matrices for the mechanism in Fig. 2 with matching color scheme. Fill-in indicated with "•". (a) Unordered matrix with fill-in after LDU decomposition. (b) Rearranged matrix without fill-in after LDU decomposition



(c)
$$\begin{pmatrix}
\boldsymbol{D}_9 & \boldsymbol{c}_{94} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\boldsymbol{c}'_{94} & \boldsymbol{D}_4 & \boldsymbol{c}'_{74} & 0 & 0 & 0 & 0 & 0 & 0\\
0 & \boldsymbol{c}_{74} & \boldsymbol{D}_7 & \boldsymbol{c}_{73} & 0 & 0 & 0 & 0 & 0\\
0 & 0 & \boldsymbol{c}'_{73} & \boldsymbol{D}_3 & \boldsymbol{c}'_{63} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & \boldsymbol{c}_{63} & \boldsymbol{D}_6 & 0 & 0 & 0 & \boldsymbol{c}_{61}\\
0 & 0 & 0 & 0 & 0 & \boldsymbol{D}_8 & \boldsymbol{c}_{82} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & \boldsymbol{c}'_{82} & \boldsymbol{D}_2 & \boldsymbol{c}'_{52} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{c}_{52} & \boldsymbol{D}_5 & \boldsymbol{c}_{51}\\
0 & 0 & 0 & 0 & \boldsymbol{c}'_{61} & 0 & 0 & \boldsymbol{c}'_{51} & \boldsymbol{D}_1
\end{pmatrix}$$

Fig. 4 (a) A walking mechanism with four links, three joints, and two contact points. (b) A graph representing the mechanism. (c) The corresponding rearranged matrix without fill-in after decomposition

Since all environment contacts are leaves in the graph, adding contact points leads to linear scaling with the correct processing order. This result is especially interesting for bipedal or quadrupedal walking robots.

4.2 Reduced-fill-in algorithm

If the graph of a mechanism has cycles, fill-in can generally no longer be avoided entirely, since nodes of a cycle are never leaves. However, by processing the leaves attached to a cycle first, the amount of fill-in is reduced. As an example, the mechanism from Fig. 2 is modified to contain a kinematic loop, displayed in Fig. 5.

Note that for computational reasons, two joints are combined into node 6, since these two joints form the beginning of the kinematic loop. Such loop-openers are also found with the depth-first search, since they are simply the first and last joints in a detected loop. In the algorithms, a distinction is made between nodes that are part of a cycle and nodes that are not. This distinction is also determined by the depth-first search.

Since any square matrix can be represented by a (potentially cyclic) graph, the following algorithms are applicable to all such matrices. In the case of mechanical systems, they can

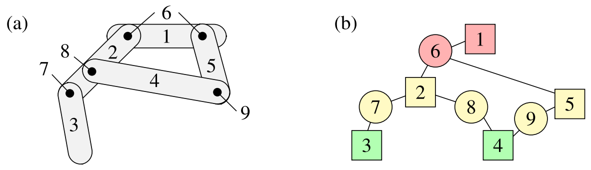


Fig. 5 (a) A mechanism with five links and four joints containing a kinematic loop. (b) A cyclic graph representing the mechanism and its matrix

Algorithm 3 Sparse in-place LDU decomposition, complexity $\mathcal{O}(n + kn^2)$

```
 1: for $i \in$ list do                                        ▷ list from DFS
 2:     for $c_1 \in$ acyclic_children($i$) do                  ▷ children not in a cycle
 3:         $\boldsymbol{F}_{i,c_1} \leftarrow \boldsymbol{F}_{i,c_1} \boldsymbol{F}^{-1}_{c_1,c_1}$
 4:         $\boldsymbol{F}_{c_1,i} \leftarrow \boldsymbol{F}^{-1}_{c_1,c_1} \boldsymbol{F}_{c_1,i}$
 5:         $\boldsymbol{F}_{i,i} \leftarrow \boldsymbol{F}_{i,i} - \boldsymbol{F}_{i,c_1} \boldsymbol{F}_{c_1,c_1} \boldsymbol{F}_{c_1,i}$
 6:     end for
 7:     for $c_1 \in$ cyclic_children($i$) do         ▷ successors of $i$ in a cycle started at $i$
 8:         for $c_2 \in$ cyclic_children($i$) do
 9:             if $c_1 == c_2$ then
10:                 break
11:             else if $c_2 \notin$ all_children($c_1$) then       ▷ acyclic and cyclic children
12:                 continue
13:             else
14:                 $\boldsymbol{F}_{i,c_1} \leftarrow \boldsymbol{F}_{i,c_1} - \boldsymbol{F}_{i,c_2} \boldsymbol{F}_{c_2,c_2} \boldsymbol{F}_{c_2,c_1}$
15:                 $\boldsymbol{F}_{c_1,i} \leftarrow \boldsymbol{F}_{c_1,i} - \boldsymbol{F}_{c_1,c_2} \boldsymbol{F}_{c_2,c_2} \boldsymbol{F}_{c_2,i}$
16:             end if
17:         end for
18:         $\boldsymbol{F}_{i,c_1} \leftarrow \boldsymbol{F}_{i,c_1} \boldsymbol{F}^{-1}_{c_1,c_1}$
19:         $\boldsymbol{F}_{c_1,i} \leftarrow \boldsymbol{F}^{-1}_{c_1,c_1} \boldsymbol{F}_{c_1,i}$
20:         $\boldsymbol{F}_{i,i} \leftarrow \boldsymbol{F}_{i,i} - \boldsymbol{F}_{i,c_1} \boldsymbol{F}_{c_1,c_1} \boldsymbol{F}_{c_1,i}$
21:     end for
22: end for
```

Algorithm 4 Sparse in-place LDU backsubstitution, complexity $\mathcal{O}(n + nk)$

```
1: for $i \in$ list do
2:     $\Delta \boldsymbol{s}_i \leftarrow -\boldsymbol{f}_i$
3:     for $c \in$ all_children$(i)$ do
4:         $\Delta \boldsymbol{s}_i \leftarrow \Delta \boldsymbol{s}_i - \boldsymbol{F}_{i,c} \Delta \boldsymbol{s}_c$
5:     end for
6: end for
7: for $i \in$ reverse(list) do
8:     $\Delta \boldsymbol{s}_i \leftarrow \boldsymbol{F}_{i,i}^{-1} \Delta \boldsymbol{s}_i$
9:     for $p \in$ all_parents$(i)$ do                ▷ direct and loop-opening parents
10:        $\Delta \boldsymbol{s}_i \leftarrow \Delta \boldsymbol{s}_i - \boldsymbol{F}_{c,p} \Delta \boldsymbol{s}_p$
11:    end for
12: end for
```

be used for all components defined in Sect. 2. The sparse decomposition for such systems is formulated in Algorithm 3 and the backsubstitution in Algorithm 4.

Note that all lists in Algorithms 3 and 4 are sorted in the same order as the depth-first search list.

Decomposition complexity: In a cyclic graph with n nodes and k cycles, there are $\mathcal{O}(n)$ acyclic children and $\mathcal{O}(n)$ acyclic parents. Additionally, there are $\mathcal{O}(n)$ cyclic children and $\mathcal{O}(n)$ loop-opening parents per cycle, since each cycle contains at most all nodes and each of these nodes has the same loop-opening parent. Therefore, a total of $\mathcal{O}(n)$ evaluations of the for-loop on line 2 of Algorithm 3 and a total of $\mathcal{O}(kn^2)$ evaluations of the for-loop on line 8 are required. Resulting is a complexity $\mathcal{O}(n + kn^2)$—quadratic in n and linear in k.

Backsubstitution complexity: In a cyclic graph with n nodes and k cycles, there are $\mathcal{O}(n)$ acyclic children, $\mathcal{O}(n)$ acyclic parents, $\mathcal{O}(n)$ cyclic children per cycle, and $\mathcal{O}(n)$ loop-opening parents per cycle. Therefore, a total of $\mathcal{O}(n + nk)$ evaluations of the for-loop on line 3 of Algorithm 4 and a total of $\mathcal{O}(n + nk)$ evaluations of the for-loop on line 9 are required. This results in complexity $\mathcal{O}(n + nk)$—linear in n and linear in k.

In the worst case of a fully connected graph, i.e., a fully dense matrix, each node has $\mathcal{O}(n)$ cyclic children and parents, resulting in a complexity of $\mathcal{O}(n + n^3)$. However, for real systems, the theoretical complexities are often too conservative. For a system without intersecting cycles, i.e., each node belongs to at most a single cycle, there are a total of $\mathcal{O}(n)$ cyclic children and parents and, therefore, the overall complexity is $\mathcal{O}(n + n^2)$. In the case of a constant cycle size, there is a total of $\mathcal{O}(k)$ cyclic children and parents, and a linear complexity $\mathcal{O}(n + k)$ is obtained. In a combined setting with non-intersecting cycles of fixed size, for example, a mechanism with disconnected identical legs made of kinematic loops, there are $\mathcal{O}(n)$ cycles with $\mathcal{O}(1)$ cyclic children and parents each, resulting in a linear complexity $\mathcal{O}(n)$. Improvements in the complexity are theoretically possible, but finding a processing order that creates the minimum fill-in is generally NP-hard [46].

The comparison of dense and sparse decomposition for the example in Fig. 4 with additional dampers at each joint is shown in Fig. 6.

The dampers depend on the relative velocity between the two connected bodies, leading to additional off-diagonal entries. As Fig. 6 shows, with a bad processing order, an almost fully dense matrix is obtained due to fill-in, while the correct processing order creates fill-in only at the off-diagonals of nodes that are part of cycles and the loop-openers.

(a) $$\begin{pmatrix} \boldsymbol{D}_1 & \boldsymbol{c}'_{12} & 0 & 0 & \boldsymbol{c}'_{51} & \boldsymbol{c}'_{61} & 0 & 0 & 0 \\ \boldsymbol{c}_{12} & \boldsymbol{D}_2 & \boldsymbol{c}'_{32} & \boldsymbol{c}'_{42} & \bullet & \boldsymbol{c}'_{62} & \boldsymbol{c}'_{72} & \boldsymbol{c}'_{82} & 0 \\ 0 & \boldsymbol{c}_{32} & \boldsymbol{D}_3 & \bullet & \bullet & \bullet & \boldsymbol{c}'_{73} & \bullet & 0 \\ 0 & \boldsymbol{c}_{42} & \bullet & \boldsymbol{D}_4 & \boldsymbol{c}'_{54} & \bullet & \bullet & \boldsymbol{c}'_{84} & \boldsymbol{c}'_{94} \\ \boldsymbol{c}_{51} & \bullet & \bullet & \boldsymbol{c}_{54} & \boldsymbol{D}_5 & \boldsymbol{c}'_{65} & \bullet & \bullet & \boldsymbol{c}'_{95} \\ \boldsymbol{c}_{61} & \boldsymbol{c}_{62} & \bullet & \bullet & \boldsymbol{c}_{65} & \boldsymbol{D}_6 & \bullet & \bullet & \bullet \\ 0 & \boldsymbol{c}_{72} & \boldsymbol{c}_{73} & \bullet & \bullet & \bullet & \boldsymbol{D}_7 & \bullet & \bullet \\ 0 & \boldsymbol{c}_{82} & \bullet & \boldsymbol{c}_{84} & \bullet & \bullet & \bullet & \boldsymbol{D}_8 & \bullet \\ 0 & 0 & 0 & \boldsymbol{c}_{94} & \boldsymbol{c}_{95} & \bullet & \bullet & \bullet & \boldsymbol{D}_9 \end{pmatrix}$$

(b) $$\begin{pmatrix} \boldsymbol{D}_5 & \boldsymbol{c}'_{95} & \boldsymbol{c}_{54} & 0 & 0 & 0 & 0 & \boldsymbol{c}'_{65} & \boldsymbol{c}_{51} \\ \boldsymbol{c}_{95} & \boldsymbol{D}_9 & \boldsymbol{c}_{94} & 0 & 0 & 0 & 0 & \bullet & 0 \\ \boldsymbol{c}'_{54} & \boldsymbol{c}'_{94} & \boldsymbol{D}_4 & \boldsymbol{c}'_{84} & 0 & 0 & \boldsymbol{c}_{42} & \bullet & 0 \\ 0 & 0 & \boldsymbol{c}_{84} & \boldsymbol{D}_8 & 0 & 0 & \boldsymbol{c}_{82} & \bullet & 0 \\ 0 & 0 & 0 & 0 & \boldsymbol{D}_3 & \boldsymbol{c}'_{73} & \boldsymbol{c}_{32} & 0 & 0 \\ 0 & 0 & 0 & 0 & \boldsymbol{c}_{73} & \boldsymbol{D}_7 & \boldsymbol{c}_{72} & 0 & 0 \\ 0 & 0 & \boldsymbol{c}'_{42} & \boldsymbol{c}'_{82} & \boldsymbol{c}'_{32} & \boldsymbol{c}'_{72} & \boldsymbol{D}_2 & \boldsymbol{c}'_{62} & \boldsymbol{c}_{21} \\ \boldsymbol{c}_{65} & \bullet & \bullet & \bullet & 0 & 0 & \boldsymbol{c}_{62} & \boldsymbol{D}_6 & \boldsymbol{c}_{61} \\ \boldsymbol{c}'_{51} & 0 & 0 & 0 & 0 & 0 & \boldsymbol{c}'_{21} & \boldsymbol{c}'_{61} & \boldsymbol{D}_1 \end{pmatrix}$$

Fig. 6 Matrices for the mechanism in Fig. 5. (a) Disordered, almost fully dense matrix due to fill-in after LDU decomposition. (b) Rearranged matrix with minimal fill-in after LDU decomposition

5 Evaluation

The evaluation of the simulator is comprised of four parts. First, the physical accuracy of the variational integrator is analyzed. Then, the runtime and computational complexity of the graph-based algorithms are investigated. Next, the numerical robustness of the simulator is tested. Lastly, two application examples for the simulator are given. Comparisons are made to different simulators and integrators. We compare to the dynamics simulator RigidBodyDynamics [47] as it is written in the same programming language as our integrator, Julia [48], and to the widely used simulator MuJoCo, which is representative for soft constraint handling. Variational integrator comparisons are made with two minimal-coordinate integrators [28, 29] and an integrator in redundant coordinates for constrained systems [17]. In order to solve the integrator equations (29a)–(29f), a basic interior-point method is used and described in Appendix D. Note that the focus of the implementation is on the theoretical properties and not runtime optimization. Nonetheless, even this basic implementation achieves reasonable timing results, benefitting from the easy and modular implementation in maximal coordinates. Code for the simulator[1] and the graph-based system solver[2] including all experiments and additional examples is available in Julia. All experiments are carried out on an Asus ZenBook with an i7-8565U CPU and 16 GB RAM.

5.1 Physical accuracy

The physical accuracy of the simulator is examined in four scenarios: constraint drift, energy conservation, energy dissipation, and contact violation. A comparison to commonly used alternative implementations is provided for reference. The results are displayed in Fig. 7.

For the constraint drift in Fig. 7 (a), a three-link mechanism forming a kinematic loop is used with link lengths $l_1 = 1$ m, $l_2 = \frac{\sqrt{2}}{2}$ m, $l_3 = 1$ m, masses $m_1 = 1$ kg, $m_2 = \frac{\sqrt{2}}{2}$ kg, $m_3 = 1$ kg, and a step size $\Delta t = 0.01$ s. In minimal coordinates, the loop-closure constraints must be explicitly enforced. In non-variational integrators, the dynamics and constraints are formulated on an acceleration level. Without constraint stabilization [49], i.e., a spring-damper connection, the links of the mechanism start to drift apart. The explicit 4th-order Runge–Kutta–Munte–Kaas integrator [25] exhibits constraint drift without such stabilization. The 1st-order variational integrator prevents constraint drift entirely and does not require constraint stabilization.

The energy conservation in Fig. 7 (b) is evaluated on a frictionless double pendulum with link lengths $l = 1$ m, masses $m = 1$ kg, and a step size $\Delta t = 0.01$ s. Unlike variational

[1]https://github.com/janbruedigam/ConstrainedDynamics.jl.

[2]https://github.com/janbruedigam/GraphBasedSystems.jl.

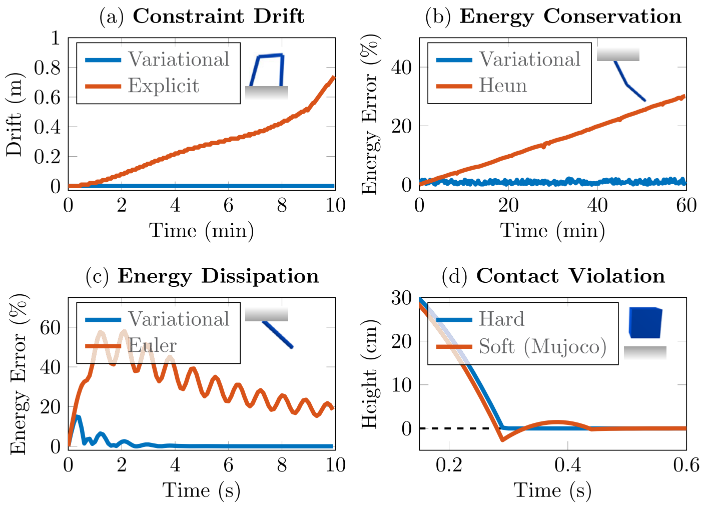


Fig. 7 Evaluation of the physical accuracy of the variational integrator (blue) with comparison (red). (a) Zero constraint drift with variational integrator, drift resulting from explicit 4th-order Runge–Kutta. (b) Bounded energy error with variational integrator for conservative system, increasing error with 2nd-order Runge–Kutta. (c) Small energy error with variational integrator for dissipative system, large error with 1st-order Runge–Kutta. (d) No contact violation for box drop with rigid contact formulation, violation with MuJoCo's soft formulation. (Color figure online)

integrators, strictly explicit or implicit Runge–Kutta integrators do not generally have tight bounds on the energy error for conservative mechanical systems, although certain methods, for example symmetric ones, do [25]. The explicit 2nd-order Runge–Kutta method (Heun's method) injects energy into the system, while the energy error stays bounded for the variational integrator.

Accurate energy dissipation is an important property, for example, in passivity-based control approaches [50]. The dissipation behavior in Fig. 7 (c) is evaluated on a damped pendulum with link length $l = 1$ m, mass $m = 1$ kg, joint damping $d = \frac{1}{2}\mathrm{N}\frac{\mathrm{s}}{\mathrm{m}}$ and a step size $\Delta t = 0.1$ s. Euler's method used for comparison shows poor dissipation behavior in drastically underdamping the pendulum, whereas the variational integrator demonstrates good dissipation performance after a small initial error.

Correct simulation of rigid contacts is crucial for transferring learned or optimized control policies from simulation to real systems. To compare rigid and soft contacts, a cube with edge length $l = 0.5$ m, mass $m = 1$ kg, and step size $\Delta t = 0.01$ s is dropped from a height (bottom-to-ground) $h = 0.4$ m. MuJoCo's default solver parameters (solimp $=$ $(0.9, 0.95, 0.001, 0.5, 2)$, solref $= (0.02, 1)$) and default Euler integrator are used and result in a ground violation of 2.7 cm. We also analyzed drops from other heights up to 1 m, which resulted in similar violations. In contrast, the rigid contact formulation with the variational integrator in maximal coordinates stops 43μm above the ground due to the interior-point formulation, which practically satisfies the constraint.

5.2 Computational complexity

The evaluation of the computational complexity serves two purposes. We show that the complexity of the graph-based algorithms holds in practice, and by using these algorithms,

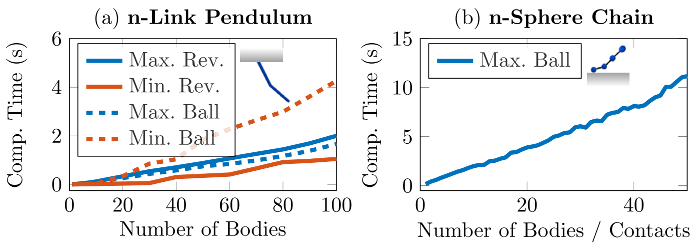


Fig. 8 Computation time comparison for simulating 1000 time steps with our simulator (blue) compared to RigidBodyDynamics (red). (a) n-link pendulum with revolute (solid) and spherical (dashed) joints. (b) n-sphere chain with contacts and spherical joints. (Color figure online)

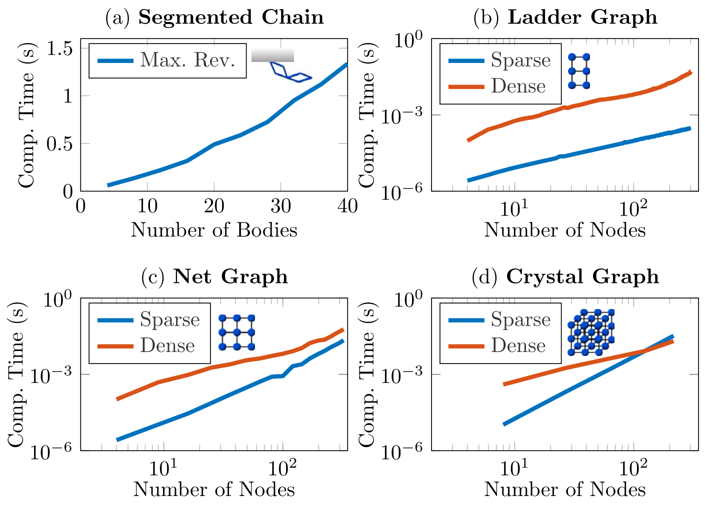


Fig. 9 Evaluation and comparison of our methods (blue) for cyclic structures compared to standard dense implementations (red). (a) Simulation of 1000 time steps of a chain consisting of 4-link segments. (b) Solving of linear system from ladder graph. (c) Solving of linear system from net graph. (d) Solving of linear system from crystal graph. (Color figure online)

maximal coordinates can achieve competitive timing results compared to minimal coordinates, despite their larger dimension. For all timings, the best result of 100 samples is used to diminish right-skewing computer noise. The linear complexity of the simulator and a comparison to minimal coordinates are shown in Fig. 8. The performance for systems with kinematic loops and comparisons to a dense solver are displayed in Fig. 9.

Figure 8 (a) compares the computation time of our simulator with the RigidBodyDynamics simulator for 1000 time steps of n-link pendulums with link length $l = 1$ m, mass $m = 1$ kg, and step size $\Delta t = 0.01$ s. The comparison is made for revolute and spherical (ball-and-socket) joints between the links. The main result is that despite the higher dimension of maximal coordinates, comparable computation times are achieved. Minimal coordinates are naturally faster for revolute joints, as they have the smallest number of degrees of freedom in minimal coordinates and the most constraints in maximal coordinates. On the other hand, spherical joints perform worse in minimal coordinates as these joints increase their state dimension and reduce the number of constraints in maximal coordinates. The linear complexity for both minimal and maximal coordinates becomes clearly visible.

In Fig. 8 (b), the linear complexity for contacts is demonstrated for a chain of spheres with radius $r = 0.25$ m, mass $m = 1$ kg, and a step size $\Delta t = 0.01$ s. The spheres are connected by spherical joints. A comparison to RigidBodyDynamics is not possible due to the limited support of contacts. Besides the complexity, this experiment demonstrates the numerical robustness of the maximal-coordinate approach for treating contacts. We also implemented the experiment in MuJoCo. While faster, MuJoCo with default solver parameters consistently has contact violations of 10 cm or more when simulating more than 20 spheres and fails to compute chains of more than 44 spheres. Therefore, a fair comparison is difficult.

The computation time for simulating 1000 time steps of a chain of 4-link segments is shown in Fig. 9 (a). The links have a length $l = 1$ m, mass $m = 1$ kg, and the step size is $\Delta t = 0.01$ s. While no longer fully linear, the increase in computation time for this system is modest due to the reduction of fill-in. A simulation of this system with the RigidBody-Dynamics simulator failed for more than one 4-link segment, even for smaller step sizes. Numerically, MuJoCo can simulate this system successfully, but with the default solver parameters, the explicit loop-closure constraints introduce high damping into the system, resulting in bad energy conservation behavior.

Comparisons of the sparse graph-based system solver with a dense one are shown in Figs. 9 (c)–(d). The ladder graph consists of cycles with four nodes each, and two nodes are shared by two cycles, except for the first and last two nodes. The net graph is a square net of nodes, where inner nodes are part of four cycles, edge nodes part of two, and corner nodes part of one. The crystal graph is a three-dimensional cubic structure of nodes, where inner nodes are part of twelve cycles, surface nodes part of eight, edge nodes part of five, and corner nodes part of three. The graphs with n nodes are represented by matrices with $6n$ entries. The sparse algorithms outperform the dense ones in most cases, often by more than two orders of magnitude. The crystal graph relates to a rather dense matrix, resulting in decreasing advantage of the sparse approach over the dense one and slightly better performance for $6^3 = 216$ nodes. This last example is an extreme case. The algorithms are aimed at robotic systems, which typically have significantly fewer nodes and cycles, and the sparse performance is convincing for such systems.

5.3 Numerical robustness

We investigate three different scenarios regarding the numerical robustness of the simulator. A comparison with two state-of-the-art variational integrators in minimal coordinates is made to demonstrate the ability to simulate systems with decreasing solution tolerance. The results are displayed in Fig. 10. Since constrained mechanical systems are known to become ill-conditioned with decreasing step sizes [51, 52], we show the ability to simulate systems for reasonable step sizes in Fig. 11. A double-four-bar linkage—a commonly-used benchmark problem that exhibits singular constraint configurations—is simulated, and the results are compared to another variational integrator for constrained systems in Fig. 12.

In Fig. 10, we compare our algorithm to state-of-the-art second- and third-order variational integrators with the same n-link pendulum we use for the timing comparison in Sect. 5.2. Only revolute joints and loop-free structures are tested as ball-and-socket joints, and loop-closures were, to the best of our knowledge, not part of the implementations of these integrators. To demonstrate the robustness of our algorithm, we run simulations with solution tolerances of 10^{-6}, 10^{-8}, and 10^{-10} for the Newton methods in all algorithms. All three algorithms display the theoretical linear computational complexity, and despite the

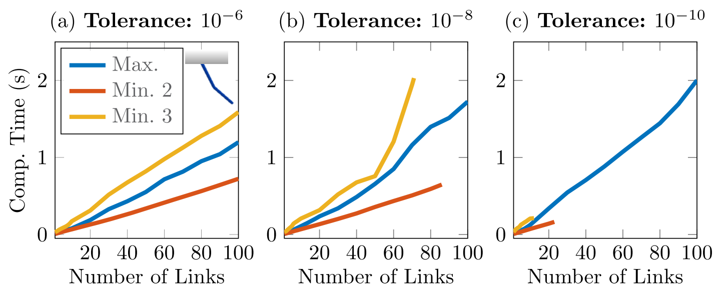


Fig. 10 Performance comparison simulating 1000 time steps of n-link pendulums with varying solution tolerances. Comparison of our 1st-order algorithm (blue) to a 2nd-order (red) and a 3rd-order variational integrator (yellow). (a) Tolerance of 10^{-6}. (b) Tolerance of 10^{-8}. (c) Tolerance of 10^{-10}. (Color figure online)

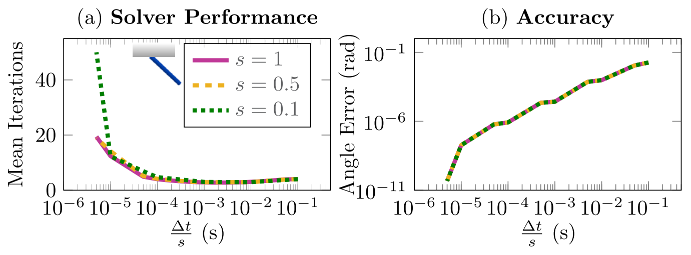


Fig. 11 Evaluation of the solver performance for pendulums of scale $s = 1$ (solid pink), $s = 0.5$ (dashed yellow), and $s = 0.1$ (dotted green) for scaled step sizes. (a) Mean iteration number. (b) Error of final angle. (Color figure online)

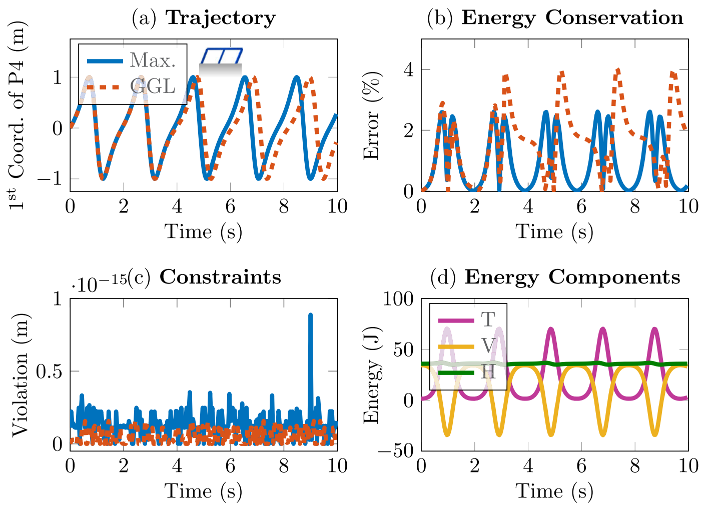


Fig. 12 Comparison of the double four-bar linkage benchmark system for our algorithm (blue) and the variational GGL method (red). (a) Simulation trajectory. (b) Energy error. (c) Constraint violation. (d) Energy components. (Color figure online)

higher dimensionality of the maximal-coordinate algorithm, we achieve comparable timing results. However, while our algorithm successfully simulates the n-link pendulums for all tolerances, the minimal-coordinate integrators fail for smaller tolerances and an increasing number of links, hinting at potential numerical issues in minimal coordinates.

Generally, large simulation step sizes are desirable for fast simulations. However, smaller step sizes are required for physically smaller systems to capture their high-frequency motions. Moreover, smaller step sizes may be necessary to achieve highly accurate results. By simulating a physically scaled pendulum with different step sizes, we determine the time-step range for which we can successfully simulate the systems. The pendulum of scale $s \in [1, 0.5, 0.1]$ has a mass $m = s$, length $l = s^2$, joint inertia $J = \frac{1}{3}ml^2$, and gravitational acceleration $g = 9.81$. Accordingly, the period of the pendulum is $T = 2\pi\sqrt{\frac{2}{3}\frac{l}{g}} = 2\pi\sqrt{\frac{2}{3}\frac{1}{g}}s$, i.e., proportional to the scale s. The pendulum is initialized at an angle and angular velocity $[\theta_0, \dot{\theta}_0] = [\frac{\pi}{2}, 0]$ and the simulation time is $\frac{T}{2}$, i.e., a half swing. The results in Fig. 11 are plotted for a step size scaled by s to account for the different frequencies of the systems. An increase in the average iteration number per time step for smaller step sizes can be seen in Fig. 11 (a), which is consistent with the increased ill-conditioning of the systems. Nonetheless, the simulation is successful and achieves accurate results even for small step sizes. The accuracy is measured as $\mathrm{norm}(-\frac{\pi}{2} - \theta_T)$, i.e., the error of the angle at the last time step.

We simulate the double four-bar linkage, a common benchmark system [7], to evaluate the performance of our simulator since the system encounters configuration singularities during simulation when the bars are in parallel. We compare our algorithm to the GGL method in [17] and use the same notation and parameters for the system. The simulation step size is $\Delta t = 0.01$ s. As Fig. 12 (a) and (b) show, our method simulates the system with a bounded error on the energy, whereas the GGL method incurs an increasing energy error, which also leads to an increasingly wrong trajectory. Regarding the constraint satisfaction shown in Fig. 12 (c), both algorithms achieve very low constraint violations, although GGL performs better, potentially because the GGL method enforces the constraints also on a velocity level. Figure 12 (d) shows the kinetic, potential, and total energy of the system as a reference for other benchmarks. We also ran the benchmark for a step size $\Delta t = 0.001$ s, where almost no difference exists between the two methods. A computation time comparison between uncompiled Matlab and compiled Julia code is difficult, and the GGL method is not designed for numerical efficiency, but it is still worth mentioning that our algorithm was more than 100 times faster than the GGL method.

5.4 Application examples

This article focuses on the theoretical foundation for building efficient and physically accurate maximal-coordinate simulators. However, an implementation with a basic interior-point method can already be used for real-world applications. As such application examples, we show that the control parameters for a quadrupedal robot can be learned with a simple sampling-based approach and how controller gains of an exoskeleton can be personalized to impaired patients in simulation to avoid injury. The quadrupedal robot and learning progress are displayed in Fig. 13. The exoskeleton and resulting torques for original and adapted controller gains are visualized in Fig. 14.

We use the Unitree A1 quadrupedal robot for sampling-based learning. The simulation step size is $\Delta t = 0.001$ s. The learning algorithm is stated in Appendix E. For each episode, a random set of control parameters in the vicinity of the current set of parameters is chosen,

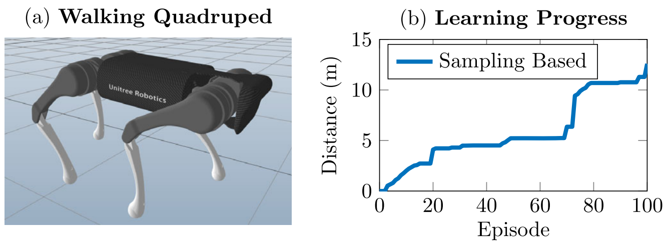


Fig. 13 (a) The trained walking quadruped. (b) Progress for learning to walk

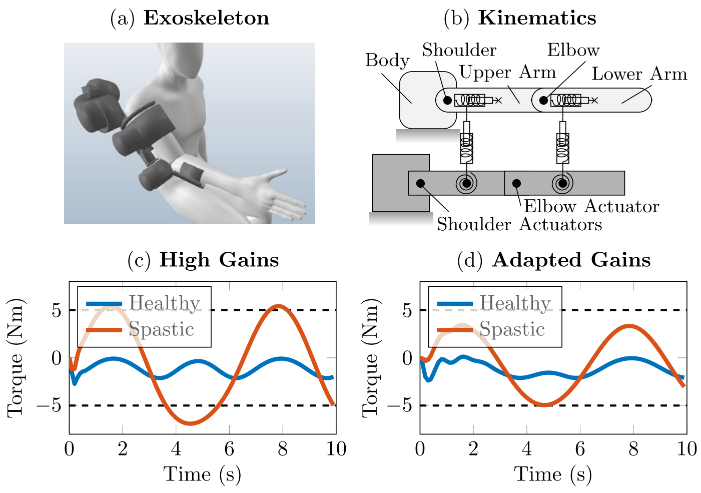


Fig. 14 (a) The donned exoskeleton in simulation. (b) Schematics of the exoskeleton kinematics (not to scale) showing the joints and kinematic loops. Offset between exoskeleton and body exaggerated. (c) Elbow torques during rehabilitation task for high gains. Within limits for healthy patient, exceeding limits for spastic patient. (d) Elbow torques for adapted gains within the limits for both patients. (Color figure online)

and a simulation rollout is performed for 10 seconds. In case of progress, i.e., the walking distance increased, this set of parameters is selected as the starting point for new sample draws. The robot learns to walk a distance of more than 12 m (average velocity of 1.2 $\frac{\mathrm{m}}{\mathrm{s}}$) in 100 episodes. Due to the rigid contacts of the simulation, a successful transfer of the learned control parameters to a real system is more likely than with an incorrectly soft contact model.

For the gain tuning, a 4-degrees-of-freedom (DoF) exoskeleton for an arm is used. Three DoF actuate the shoulder and upper arm, and one DoF the elbow and lower arm. The attachments of the upper and lower arm are modeled as two 3-DoF spring-damper joints with two translational and one rotational DoF. As a result, there are two connected kinematic loops in the mechanism. Commonly, attachments between the exoskeleton and the human body are modeled as pure spring-damper connections without joints to avoid such kinematic loops, for example, in [2]. However, such models are not necessarily correct. For the exoskeleton application, we assume a healthy and a spastic patient that perform a rehabilitation routine in the exoskeleton. The shoulder and elbow flexion/extension joints are supposed to follow sinusoidal trajectories (see Appendix E). A limit of 5 Nm is assumed to be a comfortable elbow torque for the patients. While the original gains stay within these limits for the healthy patient, they are exceeded for the spastic patient. By tuning the gains in simulation, an adapted set of gains adhering to the torque limits for both patients is found. With this approach, uncomfortable and potentially harmful tuning on a real patient can be avoided or at least reduced. The modeling accuracy, including proper kinematic loops, should provide closer estimates of the controller gains for the real system.

6 Conclusions

This article introduces a maximal-coordinate variational integrator and efficient graph-based solver for simulating mechanical systems with common components such as springs and dampers, actuated joints, and contacts with friction. Besides the theoretical formulation of the integrator and solver algorithms, an application-ready implementation of the simulator is provided as open-source code.

Building maximal-coordinate simulators on variational integrators is useful not just for the conservation properties and physical accuracy, but also for avoiding constraint drift in the naturally constrained formulation. The increased state dimension is treated with efficient numerical solver algorithms, which reduce the computational complexity in theory and render the simulator useable in practice for various applications. Additionally, it appears that the formulation in maximal coordinates increases the numerical robustness and allows the simulation of systems with contacts or kinematic loops that other simulators in minimal coordinates fail to compute.

Because of the simple modular formulation in maximal coordinates, additional components can be added to the integrator, such as a nonlinear friction model, joint limits, or physically correct elastic contacts. The graph-based nature of the solver algorithms also opens up the possibility of parallelizing computations on different branches of a graph.

Appendix A: Quaternions

We use quaternions as rotation representations since they are globally non-singular, as opposed to three-parameter representations, and numerically efficient, having only four parameters, unlike, for example, rotation matrices with nine parameters.

A.1 Notation

We write quaternions as a stacked vector,

$$\boldsymbol{q} = \begin{bmatrix} q_s \\ q_{v_1} \\ q_{v_2} \\ q_{v_3} \end{bmatrix} = \begin{bmatrix} q_s \\ \boldsymbol{q}_v \end{bmatrix} \in \mathbb{H}, \tag{A.1}$$

where q_s and $\boldsymbol{q}_v$ are the scalar and vector parts, respectively. We follow the Hamilton convention with a local-to-global rotation action. In this convention, a quaternion $\boldsymbol{q}$ maps vectors from the local to the global frame, whereas its inverse maps from the global to the local frame.

Notation (A.1) allows for a simple formulation of the basic operations conjugate, inverse, and multiplication:

$$\text{Conjugate:} \quad \boldsymbol{q}^{\mathsf{C}} = \begin{bmatrix} q_s \\ -\boldsymbol{q}_v \end{bmatrix}, \tag{A.2a}$$

$$\text{Inverse:} \quad \boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^{\mathsf{C}}}{\|\boldsymbol{q}\|}, \tag{A.2b}$$

$$\text{Multiplication:} \quad \boldsymbol{q} \cdot \boldsymbol{p} = \begin{bmatrix} q_s p_s - \boldsymbol{q}_v^{\mathsf{T}} \boldsymbol{p}_v \\ q_s \boldsymbol{p}_v + p_s \boldsymbol{q}_v + \boldsymbol{q}_v \times \boldsymbol{p}_v \end{bmatrix}. \tag{A.2c}$$

Note that, for unit quaternions, $\boldsymbol{q}^{\mathsf{C}} = \boldsymbol{q}^{-1}$, and that the $\times$ operator indicates the standard cross product of two vectors.

Three other common operations are expanding a vector $\boldsymbol{x} \in \mathbb{R}^3$ into a quaternion, retrieving the vector part from a quaternion, and constructing a skew-symmetric matrix from a vector $\boldsymbol{x} \in \mathbb{R}^3$ to form the cross product as a matrix-vector product:

$$\text{Expand vector:} \quad \boldsymbol{x}^{\wedge} = \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix}, \tag{A.3a}$$

$$\text{Retrieve vector:} \quad \boldsymbol{q}^{\vee} = \boldsymbol{q}_v, \tag{A.3b}$$

$$\text{Skew-symmetric matrix:} \quad \boldsymbol{x}^{\times} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}. \tag{A.3c}$$

The cross product of two vectors can then be written as $\boldsymbol{x}_1 \times \boldsymbol{x}_2 = \boldsymbol{x}_1^{\times} \boldsymbol{x}_2$.

To simplify calculations with quaternions, we introduce the following four matrices with the identity matrix $\boldsymbol{I}_{3\times3} \in \mathbb{R}^{3\times3}$:

$$\boldsymbol{T} = \begin{bmatrix} 1 & \boldsymbol{0}^{\mathsf{T}} \\ \boldsymbol{0} & -\boldsymbol{I}_{3\times3} \end{bmatrix} \in \mathbb{R}^{4\times4}, \tag{A.4a}$$

$$\boldsymbol{L}(\boldsymbol{q}) = \begin{bmatrix} q_s & -\boldsymbol{q}_v^{\mathsf{T}} \\ \boldsymbol{q}_v & q_s \boldsymbol{I}_{3\times3} + \boldsymbol{q}_v^{\times} \end{bmatrix} \in \mathbb{R}^{4\times4}, \tag{A.4b}$$

$$\boldsymbol{R}(\boldsymbol{q}) = \begin{bmatrix} q_s & -\boldsymbol{q}_v^{\mathsf{T}} \\ \boldsymbol{q}_v & q_s \boldsymbol{I}_{3\times3} - \boldsymbol{q}_v^{\times} \end{bmatrix} \in \mathbb{R}^{4\times4}, \tag{A.4c}$$

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{3\times3} \end{bmatrix} \in \mathbb{R}^{3\times4}. \tag{A.4d}$$

With these matrices all required quaternion operations can be written as matrix-vector products for which the standard rules of linear algebra hold:

$$\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 = \boldsymbol{L}(\boldsymbol{q}_1)\boldsymbol{q}_2 = \boldsymbol{R}(\boldsymbol{q}_2)\boldsymbol{q}_1, \tag{A.5a}$$

$$\boldsymbol{q}^{-1} = \boldsymbol{T}\boldsymbol{q}, \tag{A.5b}$$

$$\boldsymbol{q}^{\vee} = \boldsymbol{V}\boldsymbol{q}, \tag{A.5c}$$

$$\boldsymbol{x}^{\wedge} = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{x}. \tag{A.5d}$$

The rotation of a vector $\boldsymbol{x}$ can be expressed as

$$\left(\boldsymbol{q} \cdot \boldsymbol{x}^{\wedge} \cdot \boldsymbol{q}^{-1}\right)^{\vee} = \boldsymbol{Q}(\boldsymbol{q})\boldsymbol{x}, \tag{A.6}$$

where $\boldsymbol{Q}(\boldsymbol{q}) = \boldsymbol{V}\boldsymbol{R}(\boldsymbol{q})^{\mathsf{T}}\boldsymbol{L}(\boldsymbol{q})\boldsymbol{V}^{\mathsf{T}}$ is the rotation matrix formed from $\boldsymbol{q}$.

A.2 Derivatives

While quaternions have four parameters, rotations only have three degrees of freedom, and so we use specialized derivatives for quaternion functions following [20, 53].

The rotational gradient of a quaternion-dependent scalar function $f(\boldsymbol{q})$ is defined as

$$\nabla^{\mathrm{r}}_{\boldsymbol{q}} f(\boldsymbol{q}) = \boldsymbol{V}\boldsymbol{L}(\boldsymbol{q})^{\mathsf{T}}\nabla_{\boldsymbol{q}} f(\boldsymbol{q}), \tag{A.7}$$

and the rotational Jacobian of a vector-valued function $\boldsymbol{f}(\boldsymbol{q})$ as

$$\frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial^{\mathrm{r}}\boldsymbol{q}} = \frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial \boldsymbol{q}}\boldsymbol{L}(\boldsymbol{q})\boldsymbol{V}^{\mathsf{T}}. \tag{A.8}$$

A.3 Properties for dynamics descriptions

Using quaternions in the description of dynamical systems requires some attention.

Angular velocity In continuous time, the quaternion angular velocity is defined as

$$\bar{\boldsymbol{\omega}} = \begin{bmatrix} \bar{\omega}_s \\ \boldsymbol{\omega} \end{bmatrix} = 2\boldsymbol{L}(\boldsymbol{q})^{\mathsf{T}}\dot{\boldsymbol{q}}, \tag{A.9}$$

where $\bar{\omega}_s = 0$. However, using a first-order approximation of $\dot{\boldsymbol{q}}$,

$$\dot{\boldsymbol{q}}_k = \frac{\boldsymbol{q}_{k+1} - \boldsymbol{q}_k}{\Delta t}, \tag{A.10}$$

we obtain a discretized quaternion angular velocity $\bar{\boldsymbol{\omega}}_k$, generally with a scalar part $\bar{\omega}_{k,s} \neq 0$. Therefore, $\bar{\boldsymbol{\omega}}_k$ is defined so that given $\boldsymbol{q}_k$ and $\boldsymbol{\omega}_k$, $\boldsymbol{q}_{k+1}$ maintains unit norm.

Given the discretized angular velocity

$$\begin{aligned}\boldsymbol{\omega}_k &= \left(2\boldsymbol{L}(\boldsymbol{q}_k)^{\mathsf{T}}\frac{\boldsymbol{q}_{k+1} - \boldsymbol{q}_k}{\Delta t}\right)^{\vee} \\ &= \frac{2}{\Delta t}\left(\boldsymbol{L}(\boldsymbol{q}_k)^{\mathsf{T}}\boldsymbol{q}_{k+1}\right)^{\vee},\end{aligned} \tag{A.11}$$

we define the discretized quaternion angular velocity as

$$\bar{\boldsymbol{\omega}}_k = \frac{2}{\Delta t}\boldsymbol{L}(\boldsymbol{q}_k)^{\mathsf{T}}\boldsymbol{q}_{k+1}. \tag{A.12}$$

Since $\boldsymbol{L}(\boldsymbol{q}_k)^{\mathsf{T}}\boldsymbol{q}_{k+1}$ is an orientation and must have unit norm, the constraint on $\bar{\boldsymbol{\omega}}_k$ is

$$\left\|\frac{\Delta t}{2}\bar{\boldsymbol{\omega}}_k\right\|^2 = \left(\frac{\Delta t}{2}\right)^2 \bar{\omega}_{k,s}^2 + \left(\frac{\Delta t}{2}\right)^2 \boldsymbol{\omega}_k^{\mathsf{T}}\boldsymbol{\omega}_k = 1. \tag{A.13}$$

As a result,

$$\bar{\boldsymbol{\omega}}_k = \begin{bmatrix} \bar{\omega}_{k,s} \\ \boldsymbol{\omega}_k \end{bmatrix} = \begin{bmatrix} \sqrt{\left(\frac{2}{\Delta t}\right)^2 - \boldsymbol{\omega}_k^\mathsf{T}\boldsymbol{\omega}_k} \\ \boldsymbol{\omega}_k \end{bmatrix}. \tag{A.14}$$

Note that $\bar{\omega}_{k,s} = \frac{2}{\Delta t}$ for $\boldsymbol{\omega} = 0$. This difference to the continuous-time definition simplifies the integrator derivation and implementation.

Virtual work In the Lagrange–d'Alembert principle, the virtual work for external forces and torques is

$$\delta W = \begin{bmatrix} \mathbf{f}(z) & \boldsymbol{\tau}(z) \end{bmatrix}^\mathsf{T} \delta z, \tag{A.15}$$

where δz represents variations of the trajectory. In our case, the force $\mathbf{f}(z) = \mathbf{f}$ is not state-dependent, and the torque $\boldsymbol{\tau}(z) = 2\boldsymbol{L}(\boldsymbol{q})\boldsymbol{V}^\mathsf{T}\boldsymbol{\tau}$ depends on a body's current orientation. In the integrator derivation, gradients represent variations δz.

Appendix B: Kinematic joints

Most of the common joints can be defined by composing a general translational constraint function and a general rotational constraint function. A visualization of the two constraints is given in Fig. B.1. This representation, based on two general constraint functions, also allows easy extraction of minimal coordinates.

B.4 Translational constraint

The general translational constraint function describes the relative distance of two points relative to the origins of the local frames of two bodies:

$$\boldsymbol{g}_\mathrm{T} = \boldsymbol{Q}(\boldsymbol{q}_\mathrm{a})^\mathsf{T} \left(\left(\boldsymbol{x}_\mathrm{b} + \boldsymbol{Q}(\boldsymbol{q}_\mathrm{b})\boldsymbol{p}_\mathrm{b}\right) - \left(\boldsymbol{x}_\mathrm{a} + \boldsymbol{Q}(\boldsymbol{q}_\mathrm{a})\boldsymbol{p}_\mathrm{a}\right)\right). \tag{B.1}$$

The vectors $\boldsymbol{p}_\mathrm{a}$ and $\boldsymbol{p}_\mathrm{b}$, pointing from the center of mass to the joint, are defined in the respective local frames, and the resulting relative distance is defined in body a's frame.

In case of only a single body directly connected to the global frame, i.e., $\boldsymbol{x}_\mathrm{a} = \mathbf{0}$ and $\boldsymbol{q}_\mathrm{a} = \mathbb{1}$ (identity quaternion), we obtain

$$\boldsymbol{g}_\mathrm{T} = \boldsymbol{x}_\mathrm{b} + \boldsymbol{Q}(\boldsymbol{q}_\mathrm{b})\boldsymbol{p}_\mathrm{b} - \boldsymbol{p}_\mathrm{a}. \tag{B.2}$$

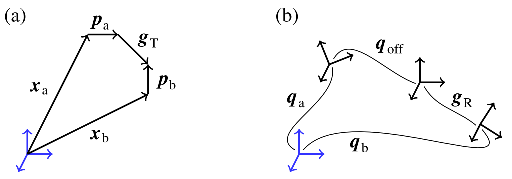


Fig. B.1 Visualization of the two general constraints. (a) The relationship of the translational constraint components. (b) The relationship of the rotational constraint components

B.5 Rotational constraint

The general rotational constraint function describes the relative distance of two local frames of two bodies, including a possible offset:

$$\boldsymbol{g}_{\mathrm{R}} = \boldsymbol{V}\boldsymbol{L}(\boldsymbol{q}_{\mathrm{off}})^{\mathrm{T}}\boldsymbol{L}(\boldsymbol{q}_{\mathrm{a}})^{\mathrm{T}}\boldsymbol{q}_{\mathrm{b}}. \tag{B.3}$$

The offset quaternion $\boldsymbol{q}_{\mathrm{off}}$ is defined in body a's frame, and the resulting relative rotation is defined in body a's frame as well.

In case of only a single body directly connected to the global frame, i.e., $\boldsymbol{q}_{\mathrm{a}} = \mathbb{1}$, we obtain

$$\boldsymbol{g}_{\mathrm{R}} = \boldsymbol{V}\boldsymbol{L}(\boldsymbol{q}_{\mathrm{off}})^{\mathrm{T}}\boldsymbol{q}_{\mathrm{b}}. \tag{B.4}$$

B.6 Composite constraints and minimal coordinates

To obtain actual joint constraints, the general constraint functions are multiplied with a selection matrix $\boldsymbol{D}$ indicating the desired constraints. Multiplying the constraint functions with the nullspace matrix $\boldsymbol{C}$ of $\boldsymbol{D}$ yields the corresponding minimal coordinates.

The selection matrix $\boldsymbol{D}$ is calculated by performing singular value decomposition on a skew-symmetric matrix formed from a vector $\boldsymbol{V}_3$:

$$\mathrm{svd}(\boldsymbol{V}_3^{\times}) = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}. \tag{B.5}$$

The matrix V contains both the original vector $\boldsymbol{V}_3$ (sign-adjustment might be necessary) and two perpendicular vectors $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$:

$$\boldsymbol{V} = \begin{bmatrix}\boldsymbol{V}_1 & \boldsymbol{V}_2 & \boldsymbol{V}_3\end{bmatrix} = \begin{bmatrix}\boldsymbol{V}_{1:2} & \boldsymbol{V}_3\end{bmatrix}. \tag{B.6}$$

Using the matrices $\boldsymbol{V}_{1:2}$ and $\boldsymbol{V}_3$, a number of constraint functions $\boldsymbol{D}^{\mathrm{T}}\boldsymbol{g}$ can be created, where the meaning depends on $\boldsymbol{D}$. The different options for $\boldsymbol{D}$ and the corresponding nullspace matrices $\boldsymbol{C}$ are stated in Table B.1.

Mechanical joints are created by using different selection and nullspace matrices for the translational and rotational constraints and stacking everything as

$$\boldsymbol{D}^{\mathrm{T}}\boldsymbol{g} = \begin{bmatrix}\boldsymbol{D}_{\mathrm{T}} & 0\\ 0 & \boldsymbol{D}_{\mathrm{R}}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\boldsymbol{g}_{\mathrm{T}}\\ \boldsymbol{g}_{\mathrm{R}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{D}_{\mathrm{T}}\boldsymbol{g}_{\mathrm{T}}\\ \boldsymbol{D}_{\mathrm{R}}\boldsymbol{g}_{\mathrm{R}}\end{bmatrix}, \tag{B.7a}$$

$$\boldsymbol{C}^{\mathrm{T}}\boldsymbol{g} = \begin{bmatrix}\boldsymbol{C}_{\mathrm{T}} & 0\\ 0 & \boldsymbol{C}_{\mathrm{R}}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\boldsymbol{g}_{\mathrm{T}}\\ \boldsymbol{g}_{\mathrm{R}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{C}_{\mathrm{T}}\boldsymbol{g}_{\mathrm{T}}\\ \boldsymbol{C}_{\mathrm{R}}\boldsymbol{g}_{\mathrm{R}}\end{bmatrix}. \tag{B.7b}$$

A list of mechanical joints that can be created with this composition is given in Table B.2.

Table B.1 Constraint selection and nullspace matrices

Selection matrix D	Nullspace matrix C	Description
$\boldsymbol{0}_{3\times 0} \in \mathbb{R}^{3\times 0}$	$\boldsymbol{V}$ or $\boldsymbol{I}_{3\times 3} \in \mathbb{R}^{3\times 3}$	No restrictions
$\boldsymbol{V}_3 \in \mathbb{R}^{3\times 1}$	$\boldsymbol{V}_{1:2} \in \mathbb{R}^{3\times 2}$	Restricted to plane $\boldsymbol{V}_{1:2}$
$\boldsymbol{V}_{1:2} \in \mathbb{R}^{3\times 2}$	$\boldsymbol{V}_3 \in \mathbb{R}^{3\times 1}$	Restricted to axis $\boldsymbol{V}_3$
$\boldsymbol{V}$ or $\boldsymbol{I}_{3\times 3} \in \mathbb{R}^{3\times 3}$	$\boldsymbol{0}_{3\times 0} \in \mathbb{R}^{3\times 0}$	Fully restricted

Table B.2 List of joints made of the two general constraint functions

Joint Name	D_T	D_R	C_T	C_R
Fixed	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{I}_{3\times3}$	$\mathbf{0}_{3\times0}$	$\mathbf{0}_{3\times0}$
Prismatic	$\boldsymbol{V}_{1:2}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{V}_3$	$\mathbf{0}_{3\times0}$
Planar, fixed orientation	$\boldsymbol{V}_3$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{V}_{1:2}$	$\mathbf{0}_{3\times0}$
Fixed orientation	$\mathbf{0}_{3\times0}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{I}_{3\times3}$	$\mathbf{0}_{3\times0}$
Revolute (Hinge)	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{V}_{1:2}$	$\mathbf{0}_{3\times0}$	$\boldsymbol{V}_3$
Cylindrical	$\boldsymbol{V}_{1:2}$	$\boldsymbol{V}_{1:2}$	$\boldsymbol{V}_3$	$\boldsymbol{V}_3$
Planar, rotation along axis	$\boldsymbol{V}_3$	$\boldsymbol{V}_{1:2}$	$\boldsymbol{V}_{1:2}$	$\boldsymbol{V}_3$
Rotation along axis, free movement	$\mathbf{0}_{3\times0}$	$\boldsymbol{V}_{1:2}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{V}_3$
Spherical (ball-and-socket)	$\boldsymbol{I}_{3\times3}$	$\mathbf{0}_{3\times0}$	$\mathbf{0}_{3\times0}$	$\boldsymbol{I}_{3\times3}$
Cylindrical, free orientation	$\boldsymbol{V}_{1:2}$	$\mathbf{0}_{3\times0}$	$\boldsymbol{V}_3$	$\boldsymbol{I}_{3\times3}$
Planar, free orientation	$\boldsymbol{V}_3$	$\mathbf{0}_{3\times0}$	$\boldsymbol{V}_{1:2}$	$\boldsymbol{I}_{3\times3}$
Floating (unconstrained)	$\mathbf{0}_{3\times0}$	$\mathbf{0}_{3\times0}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{I}_{3\times3}$

Joints with $\boldsymbol{D}_\mathrm{R} = \boldsymbol{V}_3$ do not appear to have any physical meaning. There are also joints that cannot be described by this composition, for example, helical joints (screws). Nonetheless, they can still be formulated as an equality constraint.

Appendix C: Friction dynamics

The friction forces in a mechanism are derived from the maximum dissipation principle [37] with a linearized friction cone [38]. The principle states that the energy dissipation rate of the bodies in contact is maximized through friction.

Since friction acts on moving bodies, the energy dissipation rate is the time derivative of the kinetic energy. As the kinetic energy for all bodies of a mechanism is the sum of the kinetic energies of all individual bodies,

$$\mathcal{T}(\dot{\boldsymbol{z}}) = \sum_{n=1}^{n_\mathrm{b}} \mathcal{T}(\dot{\boldsymbol{z}}_\mathrm{n}), \tag{C.1}$$

the energy dissipation is the sum of the dissipation of all individual bodies. The energy dissipation for the contact point on a single body is

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{T}(\dot{\boldsymbol{z}}) &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{2}\boldsymbol{v}^\mathsf{T}\boldsymbol{M}\boldsymbol{v} + \frac{1}{2}\boldsymbol{\omega}^\mathsf{T}\boldsymbol{J}\boldsymbol{\omega}\right) && \text{(C.2a)}\\
&= \boldsymbol{v}^\mathsf{T}\boldsymbol{M}\dot{\boldsymbol{v}} + \boldsymbol{\omega}^\mathsf{T}\boldsymbol{J}\dot{\boldsymbol{\omega}} && \text{(C.2b)}\\
&= \boldsymbol{v}^\mathsf{T}\mathbf{f} + \boldsymbol{\omega}^\mathsf{T}\boldsymbol{\tau} && \text{(C.2c)}\\
&= \boldsymbol{v}^\mathsf{T}\mathbf{f} + \boldsymbol{\omega}^\mathsf{T}\boldsymbol{p}^\times\boldsymbol{Q}(\boldsymbol{q}^{-1})\mathbf{f} \;\; (\boldsymbol{\tau} \text{ from } \mathbf{f} \text{ in body frame}) && \text{(C.2d)}\\
&= \boldsymbol{z}^\mathsf{T}\begin{bmatrix} \boldsymbol{I}_{3\times3} \\ \boldsymbol{p}^\times\boldsymbol{Q}(\boldsymbol{q}^{-1}) \end{bmatrix}\mathbf{f} && \text{(C.2e)}
\end{aligned}$$

$$= z^{\mathsf{T}} \begin{bmatrix} \boldsymbol{I}_{3\times3} \\ \boldsymbol{p}^{\times} \boldsymbol{Q}(\boldsymbol{q}^{-1}) \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_1 & -\boldsymbol{b}_1 & \cdots & \boldsymbol{b}_{\frac{n_\mathrm{f}}{2}} & -\boldsymbol{b}_{\frac{n_\mathrm{f}}{2}} \end{bmatrix}^{\mathsf{T}} \boldsymbol{\beta} \tag{C.2f}$$

$$= z^{\mathsf{T}} \boldsymbol{B}^{\mathsf{T}} \boldsymbol{\beta}, \tag{C.2g}$$

where $\mathbf{f}$ and $\boldsymbol{\tau}$ are the force and torque resulting from the friction at the contact point, $\boldsymbol{p}$ is the vector from the center of mass to the contact point in the body frame, and the $^{\times}$ operator creates the skew-symmetric matrix from a vector. The rotation matrix $\boldsymbol{Q}(\boldsymbol{q}^{-1})$ maps the force $\mathbf{f}$ from the global frame to the body frame in order to obtain the torque $\boldsymbol{\tau} = \boldsymbol{p}^{\times}\boldsymbol{Q}(\boldsymbol{q}^{-1})\mathbf{f}$ in the body frame. The force $\mathbf{f}$ at the contact point is the linearized friction force, consisting of the basis vectors $\boldsymbol{b}_i$ of the friction cone and the magnitude $\boldsymbol{\beta}$. The basis vectors $\boldsymbol{b}_i$ of a linearized friction cone are depicted in Fig. 1 (b).

The dissipation derivation (C.2a)–(C.2g) can be trivially extended to multiple bodies and contact points. Each contact point involves at most two bodies. Two bodies in contact share the same friction magnitude $\boldsymbol{\beta}$ but have different (flipped) basis vectors $\boldsymbol{b}_i$. Accordingly, the mapping matrix $\boldsymbol{B}$ differs for each body and contact point. Accordingly, the dissipation for multiple bodies and contact points takes on the same form

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{T}(\dot{z}) = z^{\mathsf{T}} \boldsymbol{B}^{\mathsf{T}} \boldsymbol{\beta}, \tag{C.3}$$

but now $\dot{z}$, z, and $\boldsymbol{\beta}$ are the stacked quantities for all bodies and magnitudes, and $\boldsymbol{B}$ consists of the individual $\boldsymbol{B}_{i,j}$ matrices of body i and contact point j.

Pairs of basis vectors point in opposite directions. Therefore, all elements of the friction magnitude must be positive. The magnitude is also limited by the friction coefficient c_f according to Coulomb friction, yielding the constraints

$$\mathbf{1}^{\mathsf{T}}\boldsymbol{\beta} \leq c_\mathrm{f}\gamma, \tag{C.4a}$$

$$\boldsymbol{\beta} \geq \mathbf{0}. \tag{C.4b}$$

Maximum dissipation can, therefore, be stated as a constrained optimization problem

$$\min_{\boldsymbol{\beta}} \; z^{\mathsf{T}} \boldsymbol{B}^{\mathsf{T}} \boldsymbol{\beta}, \tag{C.5a}$$

$$\text{s.t.} \quad \boldsymbol{E}^{\mathsf{T}}\boldsymbol{\beta} \leq \boldsymbol{C}_\mathrm{f}\boldsymbol{\gamma}, \tag{C.5b}$$

$$\boldsymbol{\beta} \geq \mathbf{0}, \tag{C.5c}$$

which yields the friction forces.

Appendix D: Simulator algorithms

The simulator computes the forward dynamics by solving the system of equations (29a)–(29f) at each time step. An interior-point method is implemented as a solver for this system. The implementation follows Algorithm 19.1 in [44], which provides more detailed explanations. Pseudo code for the implementation is given in Algorithm 5, and code is available in the open-source implementation (see Sect. 5).

Algorithm 5 Dynamics simulator

```
1: function line_search($\Delta \boldsymbol{s}$, $\boldsymbol{s}^{(0)}$)
2:     $\Delta \boldsymbol{s} =$ feasible_step($\Delta \boldsymbol{s}$)          ▷ remain feasible, see [44]
3:     $\alpha = 1$                                          ▷ step length
4:     while true do
5:         $\boldsymbol{s}^{(1)} = \boldsymbol{s}^{(0)} + \alpha \cdot \Delta \boldsymbol{s}$                ▷ take a scaled step
6:         if norm($f(\boldsymbol{s}^{(1)})$) $<$ norm($f(\boldsymbol{s}^{(0)})$) then       ▷ successful
7:             return $\boldsymbol{s}^{(1)}$
8:         end if
9:         $\alpha = \frac{\alpha}{2}$                                    ▷ half step length
10:    end while
11: end function
12: function interior_point($\boldsymbol{s}_0$)
13:    while true do                          ▷ iterative solver (Newton-based)
14:        solve($\boldsymbol{F}(\boldsymbol{s}^{(0)})\Delta \boldsymbol{s} = -\boldsymbol{f}(\boldsymbol{s}^{(0)})$)         ▷ use algorithms in Sect. 4
15:        $\boldsymbol{s}^{(1)} =$ line_search($\Delta \boldsymbol{s}$, $\boldsymbol{s}^{(0)}$)
16:        if norm($\boldsymbol{f}(\boldsymbol{s}^{(1)})$) $<$ tolerance then                ▷ successful
17:            return $\boldsymbol{s}^{(1)}$
18:        end if
19:        update_barrier()               ▷ update barrier parameter, see [44]
20:        $\boldsymbol{s}^{(0)} = \boldsymbol{s}^{(1)}$
21:    end while
22: end function
23: function simulate($N$, $\boldsymbol{s}_0$)             ▷ simulate $N$ steps starting at $\boldsymbol{s}_0$
24:    for $k = 0 : N - 1$ do                  ▷ for each step solve (29a)–(29f)
25:        $\boldsymbol{s}_{k+1} =$ interior_point($\boldsymbol{s}_k$)           ▷ current state as initial guess
26:    end for
27: end function
```

Appendix E: Application details

Details on the application examples and pseudo code are given in this appendix. Code is available in the open-source implementation (see Sect. 5).

Walking quadruped The legs of the quadruped follow a sinusoidal trajectory with five parameters. The front right and back left leg follow the same trajectory, and the front left and back right leg follow the same trajectory offset by a period of π. The leg trajectories are tracked with proportional-derivative (PD) controllers. Pseudo code for controlling the gait of the quadrupedal robot is stated in Algorithm 6.

The idea of the sampling-based learning algorithm is to randomly pick the five gait parameters. If progress is made with these parameters, i.e., the robot walks further than before, the next sampling is biased in the successful parameter direction. If no progress is made, new parameters are picked randomly in the vicinity of the current parameters. The pseudocode of the sampling-based learning algorithm for the quadruped is stated in Algorithm 7.

Exoskeleton The exoskeleton tracks a sinusoidal rehabilitation trajectory in the shoulder flexion/extension joint and in the elbow flexion/extension joint. A PD-controller with vari-

Algorithm 6 Leg controller

```
1: function leg_controller($t$, params)                          ▷ params found through learning
2:     $b, a_1, a_2, d_1, d_2 =$ params
3:     $\theta_{d1} = a_1 \cdot \cos(20\pi \cdot t \cdot b + 0) + d_1$                  ▷ thigh joint angle 1
4:     $\theta_{d2} = a_1 \cdot \cos(20\pi \cdot t \cdot b + \pi) + d_1$                  ▷ thigh joint angle 2
5:     $\theta_{d3} = a_2 \cdot \cos(20\pi \cdot t \cdot b - \frac{\pi}{2}) + d_2$                  ▷ calf joint angle 1
6:     $\theta_{d4} = a_2 \cdot \cos(20\pi \cdot t \cdot b + \frac{\pi}{2}) + d_2$                  ▷ calf joint angle 2
7:     for front_right_leg and back_left_leg do
8:         $\tau_{hip} = 100 \cdot (0 - \theta_{hip}) - 5 \cdot \dot{\theta}_{hip}$                 ▷ keep hip joint stiff
9:         $\tau_{thigh} = 80 \cdot (\theta_{d1} - \theta_{thigh}) - 4 \cdot \dot{\theta}_{thigh}$          ▷ track desired angle
10:        $\tau_{calf} = 60 \cdot (\theta_{d3} - \theta_{calf}) - 3 \cdot \dot{\theta}_{calf}$            ▷ track desired angle
11:    end for
12:    for front_left_leg and back_right_leg do
13:        $\tau_{hip} = 100 \cdot (0 - \theta_{hip}) - 5 \cdot \dot{\theta}_{hip}$
14:        $\tau_{thigh} = 80 \cdot (\theta_{d2} - \theta_{thigh}) - 4 \cdot \dot{\theta}_{thigh}$
15:        $\tau_{calf} = 60 \cdot (\theta_{d4} - \theta_{calf}) - 3 \cdot \dot{\theta}_{calf}$
16:    end for
17: end function
```

Algorithm 7 Sampling-based learning algorithm

```
1: explore_factor = 0.1                                  ▷ scaling for random sample generation
2: bias = $\mathbf{0}_5$                                              ▷ bias direction for sampling
3: params$_0$ = params$_1$ = $[0.1, 0, 1, 0, -1.5]$               ▷ initial parameters for standing
4: distance$_0$ = distance$_1$ = 0                                         ▷ walked distance
5: for episode = 1 : 100 do
6:     if bias == $\mathbf{0}_5$ then                                ▷ no improvement in last walk
7:         params$_1$ = params$_1$ + explore_factor $\cdot$ rand(5)
8:     else                                                     ▷ improvement in last walk
9:         params$_1$ = params$_1$ + 0.002 $\cdot$ rand(5) + 0.01 $\cdot$ bias
10:    end if
11:    distance$_1$ = simulate(quadruped, params$_1$)              ▷ distance walked in 10 s
12:    if distance$_1$ > distance$_0$ then                                ▷ improvement made
13:        bias = normalize(params$_1$ − params$_0$)             ▷ calculate bias direction
14:        params$_0$ = params$_1$
15:        distance$_0$ = distance$_1$
16:        explore_factor = 0.1                                ▷ reset exploration factor
17:    else                                                     ▷ no improvement made
18:        bias = $\mathbf{0}_5$
19:        explore_factor = 0.9 $\cdot$ explore_factor            ▷ decrease exploration factor
20:    end if
21: end for
```

able scaling is used for tracking, and this scaling is manually tuned with the simulation to adhere to the torque limit. The pseudo-code of the tracking controller is stated in Algorithm 8.

Algorithm 8 Exoskeleton tracking controller

```
1: function exo_controller($t$, scale)                                            ▷ scale is tuned manually
2:     $\theta_{e,d} = -\frac{\pi}{4} + \frac{\pi}{4}\sin(t)$                                              ▷ elbow trajectory
3:     $\theta_{s,d} = \frac{\pi}{4} + \frac{\pi}{4}\sin(t)$                                              ▷ shoulder trajectory
4:     $\tau_e = \text{scale} \cdot 50 \cdot (\theta_{e,d} - \theta_e) + \text{scale} \cdot 5 \cdot (\dot{\theta}_{e,d} - \dot{\theta}_e)$     ▷ PD-controller
5:     $\tau_s = \text{scale} \cdot 100 \cdot (\theta_{s,d} - \theta_s) + \text{scale} \cdot 10 \cdot (\dot{\theta}_{s,d} - \dot{\theta}_s)$   ▷ PD-controller
6: end function
```

Acknowledgements The authors would like to thank Petar Bevanda for his help in preparing the manuscript, as well as Marko Galic and Jana Janeva for their help in the implementation.

Author contributions J.B. and Z.M. conceptualized the work. J.B. developed the methodology and software. J.B. wrote the main manuscript text. All authors reviewed and edited the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Competing interests The authors declare no competing interests.

References

1. Agarwal, P., Narayanan, M.S., Lee, L.-F., Mendel, F., Krovi, V.N.: Simulation-based design of exoskeletons using musculoskeletal analysis. In: Computers and Information in Engineering Conference, pp. 1357–1364. ASMEDC, Montreal (2010)
2. Kuhn, J., Hu, T., Schappler, M., Haddadin, S.: Dynamics simulation for an upper-limb human-exoskeleton assistance system in a latent-space controlled tool manipulation task. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), pp. 158–165. IEEE, Brisbane (2018)
3. Koenemann, J., Del Prete, A., Tassa, Y., Todorov, E., Stasse, O., Bennewitz, M., Mansard, N.: Whole-body model-predictive control applied to the HRP-2 humanoid. In: International Conference on Intelligent Robots and Systems (IROS), pp. 3346–3351. IEEE, Hamburg (2015)
4. Erez, T., Lowrey, K., Tassa, Y., Kumar, V., Kolev, S., Todorov, E.: An integrated system for real-time model predictive control of humanoid robots. In: International Conference on Humanoid Robots (Humanoids), pp. 292–299. IEEE, Atlanta (2013)
5. Andrychowicz, O.M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., Zaremba, W.: Learning dexterous in-hand manipulation. Int. J. Robot. Res. **39**(1), 3–20 (2020)
6. Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., Hutter, M.: Learning quadrupedal locomotion over challenging terrain. Sci. Robot. **5**(47), eabc5986 (2020)
7. González, M., Dopico, D., Lugrís, U., Cuadrado, J.: A benchmarking system for MBS simulation software: problem standardization and performance measurement. Multibody Syst. Dyn. **16**, 179–190 (2006)

8. Todorov, E., Erez, T., Tassa, Y.: MuJoCo: a physics engine for model-based control. In: International Conference on Intelligent Robots and Systems (IROS), pp. 5026–5033. IEEE, Vilamoura-Algarve (2012)
9. Freeman, C.D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., Bachem, O.: Brax – a differentiable physics engine for large scale rigid body simulation (2021). http://github.com/google/brax
10. Tedrake, R.: (2019). The Drake Development Team: Drake: model-based design and verification for robotics. https://drake.mit.edu
11. Coumans, E., Bai, Y.: PyBullet, a Python module for physics simulation for games, robotics and machine learning (2016). http://pybullet.org
12. Zhao, W., Queralta, J.P., Westerlund, T.: Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: Symposium Series on Computational Intelligence (SSCI), pp. 737–744. IEEE, Canberra (2020)
13. Featherstone, R.: Rigid Body Dynamics Algorithms. Springer, Boston (2008)
14. Featherstone, R.: An empirical study of the joint space inertia matrix. Int. J. Robot. Res. **23**(9), 859–871 (2004)
15. de Jalón, J.G.: Twenty-five years of natural coordinates. Multibody Syst. Dyn. **18**, 15–33 (2007)
16. Betsch, P., Steinmann, P.: Constrained integration of rigid body dynamics. Comput. Methods Appl. Mech. Eng. **191**(3–5), 467–488 (2001)
17. Kinon, P.L., Betsch, P., Schneider, S.: The ggl variational principle for constrained mechanical systems. Multibody Syst. Dyn. **57**(3–4), 211–236 (2023)
18. Baraff, D.: Linear-time dynamics using Lagrange multipliers. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques – SIGGRAPH '96, pp. 137–146. ACM Press, New Orleans (1996)
19. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia (2002)
20. Brüdigam, J., Manchester, Z.: Linear-time variational integrators in maximal coordinates. In: Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 194–209. Springer, Cham (2020)
21. Brüdigam, J., Manchester, Z.: Linear-quadratic optimal control in maximal coordinates. In: International Conference on Robotics and Automation (ICRA), pp. 9775–9781. IEEE, Xi'an (2021)
22. Shield, S., Patel, A.: Minor change, major gains II: are maximal coordinates the fastest choice for trajectory optimization? In: International Conference on Intelligent Robots and Systems (IROS), pp. 12963–12970. IEEE, Kyoto (2022)
23. Zhong, G., Marsden, J.E.: Lie-Poisson Hamilton-Jacobi theory and Lie-Poisson integrators. Phys. Lett. A **133**(3), 134–139 (1988)
24. Chartier, P., Faou, E., Murua, A.: An algebraic approach to invariant preserving integators: the case of quadratic and Hamiltonian invariants. Numer. Math. **103**, 575–590 (2006)
25. Hairer, E., Lubich, C., Wanner, G.: Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations, 2nd edn. Springer, Berlin (2006)
26. Marsden, J., West, M.: Discrete mechanics and variational integrators. Acta Numer. **10**, 357–514 (2001)
27. Johnson, E.R., Murphey, T.D.: Scalable variational integrators for constrained mechanical systems in generalized coordinates. IEEE Trans. Robot. **25**(6), 1249–1261 (2009)
28. Lee, J., Liu, C., Park, F., Srinivasa, S.: A linear-time variational integrator for multibody systems. In: Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 352–367. Springer, San Francisco (2016)
29. Fan, T., Schultz, J., Murphey, T.: Efficient computation of higher-order variational integrators in robotic simulation and trajectory optimization. In: Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 689–706. Springer, Mérida (2018)
30. Leyendecker, S., Marsden, J.E., Ortiz, M.: Variational integrators for constrained dynamical systems. Z. Angew. Math. Mech. **88**(9), 677–708 (2008)
31. Betsch, P.: The discrete null space method for the energy consistent integration of constrained mechanical systems: part I: holonomic constraints. Comput. Methods Appl. Mech. Eng. **194**(50–52), 5159–5190 (2005)
32. Betsch, P., Leyendecker, S.: The discrete null space method for the energy consistent integration of constrained mechanical systems. Part II: multibody dynamics. Int. J. Numer. Methods Eng. **67**(4), 499–552 (2006)
33. Betsch, P., Steinmann, P.: A dae approach to flexible multibody dynamics. Multibody Syst. Dyn. **8**, 365–389 (2002)
34. Leyendecker, S., Betsch, P., Steinmann, P.: The discrete null space method for the energy-consistent integration of constrained mechanical systems. Part III: flexible multibody dynamics. Multibody Syst. Dyn. **19**, 45–72 (2008)
35. Brugnoli, A., Alazard, D., Pommier-Budinger, V., Matignon, D.: Port-Hamiltonian flexible multibody dynamics. Multibody Syst. Dyn. **51**(3), 343–375 (2021)

36. Sola, J., Deray, J., Atchuthan, D.: A micro Lie theory for state estimation in robotics. arXiv preprint (2018). arXiv:1812.01537
37. Preclik, T., Eibl, S., Rüde, U.: The maximum dissipation principle in rigid-body dynamics with inelastic impacts. Comput. Mech. **62**(1), 81–96 (2018)
38. Stewart, D.E., Trinkle, J.C.: An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. Int. J. Numer. Methods Eng. **39**(15), 2673–2691 (1996)
39. Wenger, T., Ober-Blöbaum, S., Leyendecker, S.: Constrained Galerkin variational integrators and modified constrained symplectic Runge-Kutta methods. In: International Conference of Numerical Analysis and Applied Mathematics (ICNAAM), Rhodes, Greece (2017)
40. Wenger, T., Ober-Blöbaum, S., Leyendecker, S.: Construction and analysis of higher order variational integrators for dynamical systems with holonomic constraints. Adv. Comput. Math. **43**(5), 1163–1195 (2017)
41. Baruh, H.: Analytical Dynamics. WCB, McGraw-Hill, Boston (1999)
42. Shivarama, R., Fahrenthold, E.P.: Hamilton's equations with Euler parameters for rigid body dynamics modeling. J. Dyn. Syst. Meas. Control **126**(1), 124–130 (2004)
43. Manchester, Z., Peck, M.: Quaternion variational integrators for spacecraft dynamics. J. Guid. Control Dyn. **39**(1), 69–76 (2016)
44. Nocedal, J., Wright, S.: Numerical Optimization. Springer, New York (2006)
45. Kwak, J., Hong, S.: Linear Algebra, 2nd edn. Birkhäuser, Boston (2004)
46. Duff, I., Erisman, A., Reid, J.: Direct Methods for Sparse Matrices, 2nd edn. Oxford University Press, Oxford (2017)
47. Koolen, T., Deits, R.: Julia for robotics: simulation and real-time control in a high-level programming language. In: International Conference on Robotics and Automation (ICRA), pp. 604–611. IEEE, Montreal (2019)
48. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.: Julia: a fresh approach to numerical computing. SIAM Rev. **59**(1), 65–98 (2017)
49. Baumgarte, J.: Stabilization of constraints and integrals of motion in dynamical systems. Comput. Methods Appl. Mech. Eng. **1**(1), 1–16 (1972)
50. Music, S., Hirche, S.: Passive noninteracting control for human-robot team interaction. In: Conference on Decision and Control (CDC), pp. 421–427. IEEE, Miami Beach (2018)
51. Petzold, L., Lötstedt, P.: Numerical solution of nonlinear differential equations with algebraic constraints II: practical implications. SIAM J. Sci. Stat. Comput. **7**(3), 720–733 (1986)
52. Cardenal, J., Cuadrado, J., Morer, P., Bayo, E.: A multi-index variable time step method for the dynamic simulation of multibody systems. Int. J. Numer. Methods Eng. **44**(11), 1579–1598 (1999)
53. Jackson, B.E., Tracy, K., Manchester, Z.: Planning with attitude. IEEE Robot. Autom. Lett. **6**(3), 5658–5664 (2021)