Linear-Time Variational Integrators in Maximal Coordinates

Jan Brüdigam and Zachary Manchester

Stanford University, Stanford CA 94305, USA,
{bruedigam, zacm}@stanford.edu

Abstract. Most dynamic simulation tools parameterize the configuration of multi-body robotic systems using minimal coordinates, also called generalized or joint coordinates. However, maximal-coordinate approaches have several advantages over minimal-coordinate parameterizations, including native handling of closed kinematic loops and nonholonomic constraints. This paper describes a linear-time variational integrator that is formulated in maximal coordinates. Due to its variational formulation, the algorithm does not suffer from constraint drift and has favorable energy and momentum conservation properties. A sparse matrix factorization technique allows the dynamics of a loop-free articulated mechanism with n links to be computed in $O(n)$ (linear) time. Additional constraints that introduce loops can also be handled by the algorithm without incurring much computational overhead. Experimental results show that our approach offers speed competitive with state-of-the-art minimal-coordinate algorithms while outperforming them in several scenarios, especially when dealing with closed loops and configuration singularities.

Keywords: Discrete Mechanics, Variational Integrators, Dynamics, Maximal Coordinates, Computer Animation & Simulation

1 Introduction

In many fields, the predominant method to describe rigid body dynamics is with *minimal coordinates* (also called joint or generalized coordinates). In this approach, each degree of freedom of a system is represented by a single coordinate describing, for example, an angle or a displacement. The main reasons for this choice of coordinates are computational performance—since only the minimal required set of variables are tracked—the absence of explicit constraints to enforce joint behavior, and the prevention of constraint drift, i.e. links drifting apart due to numerical integration errors [2].

In contrast, a different way of describing configurations of rigid bodies is with *maximal* coordinates, in which all six degrees of freedom of each link are modeled and the reduction of degrees of freedom of a system is achieved by imposing constraints via Lagrange multipliers. While linear-time computational performance for loop-free structures has been proved for both minimal and maximal

coordinates more than 20 years ago [8,1], constraint drift when using maximal coordinates is still an issue and requires stabilization schemes [2,14].

A separate and more recent area of research regarding rigid body dynamics is concerned with *variational integrators* [16]. These integrators have a number of advantages over classical Runge-Kutta methods when numerically integrating the differential equations used to describe mechanical systems. Instead of deriving the equations of motion of a dynamical system in continuous time and then discretizing them, variational approaches discretize the derivation of the equations of motion itself, and thereby retain many of the properties of the real system such as energy and momentum conservation [16,9].

In the last few years, algorithms for calculating rigid body dynamics in linear time with variational integrators in minimal coordinates have been presented [13,7]. While these algorithms show promising results, mechanical structures with closed kinematic loops or nonholonomic (i.e. velocity dependent), constraints have not been explicitly treated. Additionally, closed-loop structures and nonholonomic settings require explicit constraints, which diminish the advantages of minimal coordinates since these constraints can no longer be eliminated in advance and have to be handled similarly to the maximal-coordinate approach.

To address the accurate handling of constrained mechanical systems, this paper presents:

- A variational integrator for the equations of motion of rigid bodies in maximal coordinates that guarantees constraint satisfaction.
- An algorithm to calculate the dynamics of loop-free structures with this integrator in linear-time.
- The capability to robustly extend this algorithm by loop-closure constraints with limited increase in computation time.

This paper is structured as follows: After introducing some notation in Sec. 2, we derive a variational integrator in maximal coordinates for the equations of motion of rigid bodies in Sec. 3. Subsequently, in Sec. 4 we state the algorithms to integrate the equations of motion in linear-time and show experimental results in Sec. 5.

2 Background

We will give a brief review of quaternions (see [20] for details), and provide an example of two rigid bodies and how their connection with a joint can be described by a constraint equation.

2.1 Quaternions

Quaternions are a natural choice to represent the orientation of a rigid body as they are globally non-singular as opposed to three-parameter representations, but still have a concise set of only four parameters, unlike, for example, rotation matrices with nine parameters.

Quaternions have four components, commonly written as a stacked vector,

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ q_{v_1} \\ q_{v_2} \\ q_{v_3} \end{bmatrix} = \begin{bmatrix} q_w \\ \boldsymbol{q}_v \end{bmatrix}, \tag{1}$$

where q_w and $\boldsymbol{q}_v$ are called the scalar and vector components, respectively. In this paper we will only be using unit quaternions, i.e. $\boldsymbol{q}^T\boldsymbol{q} = 1$, to represent orientations, and we are using 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 (1) allows for a simple formulation of the basic operations *inverse* and *multiplication*:

$$\text{Inverse:} \qquad \boldsymbol{q}^\dagger = \begin{bmatrix} q_w \\ -\boldsymbol{q}_v \end{bmatrix} \tag{2}$$

$$\text{Multiplication:} \quad \boldsymbol{q} \otimes \boldsymbol{p} = \begin{bmatrix} q_w p_w - \boldsymbol{q}_v^{\mathrm{T}} \boldsymbol{p}_v \\ q_w \boldsymbol{p}_v + p_w \boldsymbol{q}_v + \boldsymbol{q}_v \times \boldsymbol{p}_v \end{bmatrix} \tag{3}$$

The $\times$ operator indicates the standard cross product of two vectors. Three other common operations include 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:

$$\text{Expand vector:} \qquad \hat{\boldsymbol{x}} = \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \tag{4}$$

$$\text{Retrieve vector:} \qquad \boldsymbol{q}^{\mathrm{v}} = \boldsymbol{q}_v \tag{5}$$

$$\text{Skew-symmetric matrix:} \quad \boldsymbol{x}^\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \rightarrow \boldsymbol{x}_1 \times \boldsymbol{x}_2 = \boldsymbol{x}_1^\times \boldsymbol{x}_2 \tag{6}$$

To simplify calculations with quaternions, we introduce the following matrices:

$$T = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & -I_3 \end{bmatrix} \qquad \in \mathbb{R}^{4\times 4}, \tag{7}$$

$$L(\boldsymbol{q}) = \begin{bmatrix} q_w & -\boldsymbol{q}_v^{\mathrm{T}} \\ \boldsymbol{q}_v & q_w I_3 + \boldsymbol{q}_v^\times \end{bmatrix} \quad \in \mathbb{R}^{4\times 4}, \tag{8}$$

$$R(\boldsymbol{q}) = \begin{bmatrix} q_w & -\boldsymbol{q}_v^{\mathrm{T}} \\ \boldsymbol{q}_v & q_w I_3 - \boldsymbol{q}_v^\times \end{bmatrix} \quad \in \mathbb{R}^{4\times 4}, \tag{9}$$

$$V = \begin{bmatrix} \mathbf{0} & I_3 \end{bmatrix} \qquad \in \mathbb{R}^{3\times 4}, \tag{10}$$

with the identity matrix $I_3 \in \mathbb{R}^{3\times3}$ to perform all required quaternion operations as matrix-vector products for which the standard rules of linear algebra hold:

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = L(\boldsymbol{q}_1)\boldsymbol{q}_2 = R(\boldsymbol{q}_2)\boldsymbol{q}_1 \tag{11}$$

$$\boldsymbol{q}^\dagger = T\boldsymbol{q} \tag{12}$$

$$\boldsymbol{q}^\mathrm{v} = V\boldsymbol{q} \tag{13}$$

$$\hat{\boldsymbol{x}} = V^\mathrm{T}\boldsymbol{x} \tag{14}$$

Now, we can write the rotation of a vector $\boldsymbol{x}$ as a matrix-vector product,

$$\left(\boldsymbol{q} \otimes \hat{\boldsymbol{x}} \otimes \boldsymbol{q}^\dagger\right)^\mathrm{v} = VR(\boldsymbol{q})^\mathrm{T}L(\boldsymbol{q})V^\mathrm{T}\boldsymbol{x} = VL(\boldsymbol{q})R(\boldsymbol{q})^\mathrm{T}V^\mathrm{T}\boldsymbol{x}, \tag{15}$$

and do the same for the angular velocity:

$$\boldsymbol{\omega} = 2\left(\boldsymbol{q}^\dagger \otimes \dot{\boldsymbol{q}}\right)^\mathrm{v} = 2VL(\boldsymbol{q})^\mathrm{T}\dot{\boldsymbol{q}}. \tag{16}$$

2.2 Articulated Rigid Body

A single rigid body has a position $\boldsymbol{x}$, velocity $\boldsymbol{v} = \dot{\boldsymbol{x}}$, and mass m, as well as an orientation $\boldsymbol{q}$, angular velocity $\boldsymbol{\omega} = 2VL(\boldsymbol{q})^\mathrm{T}\dot{\boldsymbol{q}}$, and a moment of inertia matrix J. All quantities refer to the center of mass of a body.

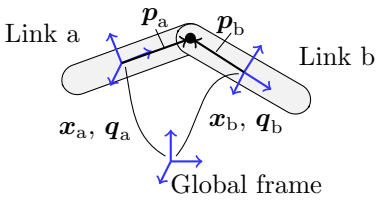

Fig. 1: Two links connected by a joint.

For an articulated mechanism consisting of multiple rigid bodies, we can formulate constraints $\boldsymbol{g}$ to represent, for instance, joints. As an example, we give the constraints for the ball-and-socket joint connecting bodies a and b in Fig. 1 (see [19] for a complete list of common joint types):

$$\boldsymbol{g} = \boldsymbol{x}_\mathrm{a} + VR(\boldsymbol{q}_\mathrm{a})^\mathrm{T}L(\boldsymbol{q}_\mathrm{a})V^\mathrm{T}\boldsymbol{p}_\mathrm{a} \quad - \quad \left(\boldsymbol{x}_\mathrm{b} + VR(\boldsymbol{q}_\mathrm{b})^\mathrm{T}L(\boldsymbol{q}_\mathrm{b})V^\mathrm{T}\boldsymbol{p}_\mathrm{b}\right), \tag{17}$$

where $\boldsymbol{p}$ is a vector from the center of mass of a body to the joint. Our algorithm enforces the constraint $\boldsymbol{g} = \boldsymbol{0}$ at the position level to guarantee constraint satisfaction, whereas other maximal-coordinate methods only enforce $\boldsymbol{g}$ at the acceleration level which results in constraint-drift [1].

3 Variational Integrator

We derive a first-order variational integrator to discretize rigid body dynamics while maintaining realistic energy and momentum conservation behavior. While this derivation is generally not new and has been treated rigorously [16,10,15], we provide a derivation in maximal coordinates with unified notation.

Variational integrators are based on the *principle of least action* which states that a mechanical system always takes the path of least action when going from a

fixed start point to a fixed end point. Action has the dimensions [Energy]×[Time] and the unforced action integral S_0 is defined as

$$S_0 = \int_{t_1}^{t_N} \mathcal{L}(\mathbf{x}(t), \dot{\mathbf{x}}(t))\ \mathrm{d}t, \tag{18}$$

where $\mathbf{x}$ is a system's trajectory in arbitrary coordinates and $\mathcal{L} = \mathcal{T} - \mathcal{V}$ is the Lagrangian of the system with kinetic energy $\mathcal{T}$ and potential energy $\mathcal{V}$. For the translational case and assuming only gravitational potential, we have

$$\mathcal{T}_\mathrm{T} = \frac{1}{2}\boldsymbol{v}^\mathrm{T} M \boldsymbol{v} \tag{19}$$

$$\mathcal{V}_\mathrm{T} = g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x}, \tag{20}$$

with position $\boldsymbol{x}$, velocity $\boldsymbol{v}$, mass matrix $M = mI_3$, gravitational acceleration g, and the z-axis vector $\boldsymbol{e}_\mathrm{z}$. In the rotational setting we assume zero potential, so $\mathcal{V}_\mathrm{R} = 0$, and the kinetic energy is

$$\mathcal{T}_\mathrm{R} = \frac{1}{2}\boldsymbol{\omega}^\mathrm{T} J \boldsymbol{\omega}. \tag{21}$$

3.1 Translational Component

Inserting the Lagrangian for the translational case into (18) and appending integrals for external forces $\boldsymbol{F}$ and constraints $\boldsymbol{g}$ with virtual constraint forces $\boldsymbol{\lambda}$ yields the forced and constrained action integral

$$S_\mathrm{T}(\boldsymbol{x}) = \int_{t_0}^{t_N} \left(\frac{1}{2}\boldsymbol{v}^\mathrm{T} M \boldsymbol{v} - g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x})\right)\ \mathrm{d}t + \int_{t_0}^{t_N} \boldsymbol{F}^\mathrm{T}\boldsymbol{x}\ \mathrm{d}t + \int_{t_0}^{t_N} \boldsymbol{\lambda}^\mathrm{T}\boldsymbol{g}(\boldsymbol{x})\ \mathrm{d}t. \tag{22}$$

The virtual constraint forces $\boldsymbol{\lambda}$ act on a rigid body to guarantee satisfaction of constraints $\boldsymbol{g}(\boldsymbol{x})$. They are called virtual as they should be workless and not change the energy of a body.

For numerical integration, we need to discretize (22). We use a first-order discretization of the integral and a first-order approximation of the velocity,

$$\boldsymbol{v}_k = \frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t}, \tag{23}$$

to obtain the forced and constrained discrete action sum:

$$S_{\mathrm{d,T}}(\boldsymbol{x}) = \sum_{k=1}^{N-1} \left(\frac{1}{2}\boldsymbol{v}_k^\mathrm{T} M \boldsymbol{v}_k - g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x}_k + \boldsymbol{F}_k^\mathrm{T}\boldsymbol{x}_k + \boldsymbol{\lambda}_k^\mathrm{T}\boldsymbol{g}(\boldsymbol{x}_k)\right)\Delta t \tag{24}$$

With our discrete action sum we can now analyze a very short trajectory from, t_1 to t_3, i.e. $k = 1$ to $k = 2 = N - 1$, that satisfies the principle of least action. Since $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ are fixed start and end points, only $\boldsymbol{x}_2$ can vary. We therefore minimize the discrete action sum with respect to $\boldsymbol{x}_2$:

$$\nabla_{\boldsymbol{x}_2} S_{\mathrm{d,T}}(\boldsymbol{x}_2) = \left(\frac{1}{\Delta t} M \boldsymbol{v}_1 - \frac{1}{\Delta t} M \boldsymbol{v}_2 - gM\boldsymbol{e}_\mathrm{z} + \boldsymbol{F}_2 + G_{\boldsymbol{x}}(\boldsymbol{x}_2)^\mathrm{T}\boldsymbol{\lambda}_2\right)\Delta t = \mathbf{0}, \tag{25}$$

where $G_{\boldsymbol{x}}$ is the Jacobian of constraints $\boldsymbol{g}$ with respect to $\boldsymbol{x}$. In words, if equation (25) is fulfilled, we have found the physically correct trajectory $\boldsymbol{x}(t)$ with fixed start point $\boldsymbol{x}(t_1)$ and fixed end point $\boldsymbol{x}(t_3)$.

Rearranging (25) yields the *discretized translational equations of motion* for a single rigid body with forcing and constraints:

$$\boldsymbol{d}_{\mathrm{T}}(\boldsymbol{v}_2, \boldsymbol{\lambda}_2) = M\left(\frac{\boldsymbol{v}_2 - \boldsymbol{v}_1}{\Delta t} + g\boldsymbol{e}_z\right) - \boldsymbol{F}_2 - G_{\boldsymbol{x}}(\boldsymbol{x}_2)^{\mathrm{T}}\boldsymbol{\lambda}_2 = \boldsymbol{0} \tag{26}$$

To derive the physically accurate equations of motion we have varied $\boldsymbol{x}_2$ and assumed fixed start and end points $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$. However, when we want to integrate our dynamics forward in time, we are actually trying to find $\boldsymbol{x}_3$ given $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. The new position $\boldsymbol{x}_3$ can be derived from (23) as

$$\boldsymbol{x}_3 = \boldsymbol{x}_2 + \boldsymbol{v}_2 \Delta t, \tag{27}$$

which means, for simulations, we will have to solve (26) implicitly for $\boldsymbol{v}_2$ while also finding appropriate constraint forces $\boldsymbol{\lambda}_2$.

Higher-order variational integrators can be derived by using higher-order quadrature rules in the discrete action sum to better approximate the true action integral [22,17]. While our algorithm is compatible with higher-order integrators, for clarity we are restricting the discussion in this paper to first-order integrators.

3.2 Rotational Component

The rotational component of the integrator can be derived similarly to the translation case. Details on the derivation can be found in the supplementary material and the literature [15]. We will only state important equations and results here.

The forced and constrained action integral for rotations is

$$S_{\mathrm{R}}(\boldsymbol{q}) = \int_{t_1}^{t_N} \frac{1}{2}\boldsymbol{\omega}^{\mathrm{T}} J \boldsymbol{\omega} \, \mathrm{d}t + \int_{t_1}^{t_N} 2\boldsymbol{\tau}^{\mathrm{T}} V L(\boldsymbol{q})^{\mathrm{T}} \boldsymbol{q} \, \mathrm{d}t + \int_{t_1}^{t_N} \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{q}) \, \mathrm{d}t, \tag{28}$$

with torque $\boldsymbol{\tau}$, which leads to the discrete forced and constrained action sum

$$S_{\mathrm{d,R}}(\boldsymbol{q}) = \sum_{k=1}^{N-1} \left(\frac{1}{2}\boldsymbol{\omega}_k^{\mathrm{T}} J \boldsymbol{\omega}_k + 2\boldsymbol{\tau}_k^{\mathrm{T}} V L(\boldsymbol{q}_k)^{\mathrm{T}} \boldsymbol{q}_k + \boldsymbol{\lambda}_k^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{q}_k)\right) \Delta t. \tag{29}$$

In this discrete action sum we have approximated $\dot{\boldsymbol{q}}$ as

$$\dot{\boldsymbol{q}} = \frac{\boldsymbol{q}_{k+1} - \boldsymbol{q}_k}{\Delta t}. \tag{30}$$

To derive the rotational equations of motion, similarly to (25), we again minimize the discrete action sum from $k = 1$ to $k = 2$, this time while varying the rotation $\boldsymbol{q}_2$:

$$\nabla^{\mathrm{r}}_{\boldsymbol{q}_2} S_{\mathrm{d,R}}(\boldsymbol{q}) = V L(\boldsymbol{q}_2)^{\mathrm{T}} \nabla_{\boldsymbol{q}_2} S_{\mathrm{d,R}}(\boldsymbol{q}) = \boldsymbol{0} \tag{31}$$

Note the *rotational gradient* ∇^{r} and see the supplementary material for details.

Introducing an implicit unit norm constraint on the quaternions [15] to avoid explicit constraints yields an update rule for the orientation similar to (27):

$$\boldsymbol{q}_3 = \frac{\Delta t}{2} L(\boldsymbol{q}_2) \begin{bmatrix} \sqrt{\left(\frac{2}{\Delta t}\right)^2 - \boldsymbol{\omega}_2^{\mathrm{T}} \boldsymbol{\omega}_2} \\ \boldsymbol{\omega}_2 \end{bmatrix} \tag{32}$$

With this implicit constraint we can state the *discretized rotational equations of motion*:

$$\begin{aligned} \boldsymbol{d}_{\mathrm{R}}(\boldsymbol{\omega}_2, \boldsymbol{\lambda}_2) = {} & J\boldsymbol{\omega}_2 \sqrt{\tfrac{4}{\Delta t^2} - \boldsymbol{\omega}_2^{\mathrm{T}} \boldsymbol{\omega}_2} + \boldsymbol{\omega}_2^{\times} J \boldsymbol{\omega}_2 \\ & - J\boldsymbol{\omega}_1 \sqrt{\tfrac{4}{\Delta t^2} - \boldsymbol{\omega}_1^{\mathrm{T}} \boldsymbol{\omega}_1} + \boldsymbol{\omega}_1^{\times} J \boldsymbol{\omega}_1 - 2\boldsymbol{\tau}_2 - G_{\boldsymbol{q}_2}(\boldsymbol{q}_2)^{\mathrm{T}} \boldsymbol{\lambda}_2 = \boldsymbol{0}. \end{aligned} \tag{33}$$

4 Linear-Time Sparse Solver

We can use the discretized equations of motion from Sec. 3 to simulate the rigid body dynamics forward in time. For a single rigid body, we stack our equations of motion $\boldsymbol{d} = [\boldsymbol{d}_{\mathrm{T}}^{\mathrm{T}} \;\; \boldsymbol{d}_{\mathrm{R}}^{\mathrm{T}}]^{\mathrm{T}}$ and the constraints $\boldsymbol{g}$ into a function

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{g} \end{bmatrix}. \tag{34}$$

To treat multiple rigid bodies, connected or not, their equations are simply stacked into $\boldsymbol{f}$. Hence, from now on, $\boldsymbol{f}$ contains all equations of motion and constraints for all bodies in a simulation. Our goal is then to find the solution to the non-linear equation

$$\boldsymbol{f}(\boldsymbol{s}) = \boldsymbol{0}, \tag{35}$$

where the solution, $\boldsymbol{s}$, consists of $\boldsymbol{v}_2$, $\boldsymbol{\omega}_2$, and $\boldsymbol{\lambda}_2$. Once we have found these values, $\boldsymbol{v}_2$ and $\boldsymbol{\omega}_2$ can be used to update the mechanism's position and orientation according to (27) and (32).

4.1 Newton's Method and Dense LDU Decomposition

Newton's method is used to solve the implicit equation (35):

$$\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)} - \left(F(\boldsymbol{s}^{(i)})\right)^{-1} \boldsymbol{f}(\boldsymbol{s}^{(i)}) \tag{36}$$

$$= \boldsymbol{s}^{(i)} - \Delta \boldsymbol{s}^{(i)}, \tag{37}$$

where $\boldsymbol{s}^{(i)}$ is the approximation of the solution at step i and F is the Jacobian of $\boldsymbol{f}$ with respect to $\boldsymbol{s}^{(i)} = [\boldsymbol{v}_2^{\mathrm{T}} \;\; \boldsymbol{\omega}_2^{\mathrm{T}} \;\; \boldsymbol{\lambda}_2^{\mathrm{T}}]^{\mathrm{T}}$:

$$F = \frac{\partial \boldsymbol{f}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial (\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)} = \begin{bmatrix} \dfrac{\partial \boldsymbol{d}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial (\boldsymbol{v}_2, \boldsymbol{\omega}_2)} & \dfrac{\partial \boldsymbol{d}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial \boldsymbol{\lambda}_2} \\ \dfrac{\partial \boldsymbol{g}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial (\boldsymbol{v}_2, \boldsymbol{\omega}_2)} & \dfrac{\partial \boldsymbol{g}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial \boldsymbol{\lambda}_2} \end{bmatrix} = \begin{bmatrix} D_{\boldsymbol{v},\boldsymbol{\omega}} & G_{\boldsymbol{x},\boldsymbol{q}}^{\mathrm{T}} \\ G_{\boldsymbol{v},\boldsymbol{\omega}} & 0 \end{bmatrix} \tag{38}$$

For computational efficiency, instead of directly inverting the Jacobian in (36), one solves the linear system

$$F(\boldsymbol{s}^{(i)})\Delta\boldsymbol{s}^{(i)} = \boldsymbol{f}(\boldsymbol{s}^{(i)}). \tag{39}$$

For a dense matrix F with few non-zeros, a linear system of equations like (39) is typically solved by using a matrix decomposition. Foreshadowing the linear-time algortihm, we review the LDU decomposition [12].

The LDU decomposition, which is essentially a scaled version of Gaussian elimination, provides a numerically robust method to factorize and solve non-symmetric linear systems of equations such as (39). The factorization part of an LDU decomposition processes a matrix diagonally from top-left to bottom-right. Algorithm 1 describes an in-place factorization (i.e. replacing current entries of the matrix with new factorized values), where $F_{i,j}$ denotes the entry of F in row i and column j.

Algorithm 1 Dense In-Place LDU Factorization

for $n = 1 : \text{length}(\Delta\boldsymbol{s})$ **do**
  for $i = 1 : n-1$ **do** ▷ L and U factorization
    for $j = 1 : i-1$ **do**
      $F_{n,i} \leftarrow F_{n,i} - F_{n,j}F_{j,j}F_{j,i}$ ▷ L
      $F_{i,n} \leftarrow F_{i,n} - F_{i,j}F_{j,j}F_{j,n}$ ▷ U
    $F_{n,i} \leftarrow F_{n,i}F_{i,i}^{-1}$ ▷ L
    $F_{i,n} \leftarrow F_{i,i}^{-1}F_{i,n}$ ▷ U
  for $j = 1 : n-1$ **do** ▷ D factorization
    $F_{n,n} \leftarrow F_{n,n} - F_{n,j}F_{j,j}F_{j,n}$ ▷ D

With this factorization of complexity $O(n^3)$, we have calculated a lower-triangular matrix "L", a diagonal matrix "D", and an upper-triangular matrix "U", with their product being LDU $= F$. The structures of matrices L, D, and U allow for back-substitution $\Delta\boldsymbol{s} = \mathrm{U}^{-1}\mathrm{D}^{-1}\mathrm{L}^{-1}\boldsymbol{f}$ with complexity $O(n^2)$ as described in Algorithm 2.

Algorithm 2 Dense LDU Back-Substitution

$\Delta\boldsymbol{s} \leftarrow \boldsymbol{f}$
for $n = 1 : \text{length}(\Delta\boldsymbol{s})$ **do** ▷ L back-substitution
  for $j = 1 : n-1$ **do**
    $\Delta\boldsymbol{s}_n \leftarrow \Delta\boldsymbol{s}_n - F_{n,j}\Delta\boldsymbol{s}_j$ ▷ L
for $n = \text{length}(\Delta\boldsymbol{s}) : 1$ **do** ▷ D and U back-substitution
  $\Delta\boldsymbol{s}_n \leftarrow F_{n,n}^{-1}\Delta\boldsymbol{s}_n$ ▷ D
  for $j = n+1 : \text{length}(\Delta\boldsymbol{s})$ **do** ▷ U
    $\Delta\boldsymbol{s}_n \leftarrow \Delta\boldsymbol{s}_n - F_{n,j}\Delta\boldsymbol{s}_j$

However, the Jacobian F in our problem is sparse and has structural properties that enable factorization and back-substitution of (39) with $O(n)$ complexity.

4.2 Sparse Factorization and Back-Substitution

While a linear-time algorithm to solve classical continuous-time dynamics in maximal coordinates has been known for over two decades [1], it cannot be applied to variational integrators. The main difference between these two settings lies in the system (39). While the continuous-time system is symmetric, allowing the use of a modified $\mathrm{LDL}^{\mathrm{T}}$ decomposition, (38) is not symmetric (not even block-symmetric) since $D_{\boldsymbol{v},\boldsymbol{\omega}}$ is not symmetric and generally $G_{\boldsymbol{x},\boldsymbol{q}} \neq G_{\boldsymbol{v},\boldsymbol{\omega}}$. These circumstances prevent us from applying the continuous-time method.

For the following analysis, we will treat F from (38) as a block matrix. Each block on the diagonal of F represents a body or a constraint of the underlying mechanism and will be called a node. Each off-diagonal block represents a connection between two nodes, i.e. between a body and a constraint. The i-th node has its diagonal block denoted by D_i, where $D_i = D_{\boldsymbol{v},\boldsymbol{\omega}}$ for a body and $D_i = 0$ for a constraint. An off-diagonal block connecting constraint i and body j is denoted by c_{ij} for blocks of $G_{\boldsymbol{x},\boldsymbol{q}}$ or c'_{ij} for blocks of $G_{\boldsymbol{v},\boldsymbol{\omega}}$.

While the block entries of F are not symmetric, the *sparsity pattern* of zero and non-zero blocks is symmetric due to matching patterns of $G_{\boldsymbol{x},\boldsymbol{q}}$ and $G_{\boldsymbol{v},\boldsymbol{\omega}}$. This means that c_{ij} and c'_{ij} are both either zero or both non-zero. We can give a graph representing this pattern that is undirected due to the symmetry and acyclic for loop-free mechanisms. An important insight is that the graph of the pattern of F and the graph of the underlying mechanical structure are identical. An exemplary mechanism and its graph are given in Fig. 2.

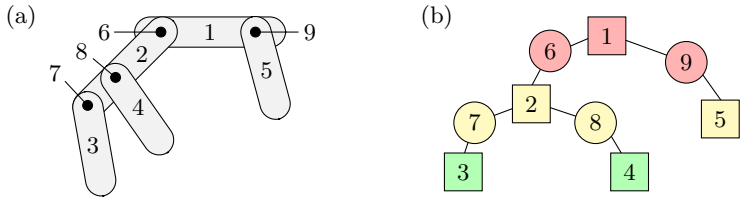

Fig. 2: (a) A mechanism with five links and four joints. (b) A graph representing the mechanism and its matrix. Squares represent links, circles represent joints. Coloring represents different levels of the tree-shaped graph.

We can factorize a matrix associated with the graph in Fig. 2 (b) efficiently by traversing the graph from the leaves to the root. Leaves are characterized by having only a single connection. The root of an acyclic graph can be chosen arbitrarily. Only if we formulate a procedure that follows the structure of the graph in this order during factorization, will we avoid *fill-ins*, i.e. we will not change any zero blocks [5]. Given the asymmetric entries but symmetric pattern of (38), we can modify the LDU decomposition to develop a linear-time algorithm.

If we reorder the rows and columns of a matrix, we can change the processing order of the LDU factorization. Reordering a matrix from leaves to root accord-

ing to its graph structure, therefore, creates a factorization without fill-ins. An ordering that creates fill-ins and one that does not are displayed in Fig. 3.

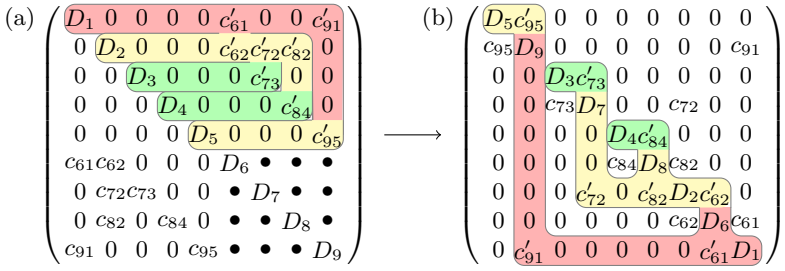

Fig. 3: Matrices for the mechanism in Fig. 2 with matching color scheme. Fill-ins indicated as "•". (a) Unordered matrix creating fill-ins during LDU factorization. (b) Rearranged matrix without fill-ins during LDU factorization.

When looking at the nodes on the diagonal of matrix (b) in Fig. 3 from top-left to bottom-right, one can clearly see how the nodes are ordered from leaves to root D_1 according to the graph in Fig. 2 (b). By contrast, in matrix (a) the non-leaf node D_1 is processed first. Note that generally there are multiple orderings to achieve zero fill-ins, for example in matrix (b), we could change the order of nodes D_3 and D_7 with D_4 and D_8, or chose an entirely different root.

We know that, for a loop-free structure, each link has exactly one parent. Therefore, for a structure with n links, we also have n constraints (or $n-1$ for floating-base systems). Because we are not creating fill-ins during the factorization, we can achieve $O(n)$ complexity by only evaluating the factorization for the $O(n)$ nodes. To find an appropriate matrix ordering, we have to perform a depth-first-search starting from the (arbitrary) root. We store the found nodes in a list with the root as the last element and the last-found node as the first element. This list then contains a valid ordering of nodes. The modified factorization procedure is formalized in Algorithm 3.

Algorithm 3 Sparse In-Place LDU Factorization

for $i \in$ list **do**
  for $c \in$ children(i) **do** ▷ Modified L and U factorization of Alg. 1
    $F_{i,c} \leftarrow F_{i,c}F_{c,c}^{-1}$
    $F_{c,i} \leftarrow F_{c,c}^{-1}F_{c,i}$
  for $c \in$ children(i) **do** ▷ Modified D factorization of Alg. 1
    $F_{i,i} \leftarrow F_{i,i} - F_{i,c}F_{c,c}F_{c,i}$

We will also exploit the fact that we did not create fill-ins during the factorization and that we have $O(n)$ nodes to develop a back-substitution procedure of $O(n)$ complexity. This modified procedure is described in Algorithm 4.

Algorithm 4 Sparse LDU Back-Substitution

for $i \in$ list **do**
  $\Delta \boldsymbol{s}_i \leftarrow \boldsymbol{f}_i$
  for $c \in$ children(i) **do** ▷ Modified L back-substitution of Alg. 2
    $\Delta \boldsymbol{s}_i \leftarrow \Delta \boldsymbol{s}_i - F_{i,c} \Delta \boldsymbol{s}_c$
for $i \in$ reverse(list) **do** ▷ Modified D and U back-substitution of Alg. 2
  $\Delta \boldsymbol{s}_i \leftarrow F_{i,i}^{-1} \Delta \boldsymbol{s}_i$
  $\Delta \boldsymbol{s}_i \leftarrow \Delta \boldsymbol{s}_i - F_{c,\mathrm{par}(i)} \Delta \boldsymbol{s}_{\mathrm{par}(i)}$ ▷ par(i): parent(i); if i has a parent

We perform sparse factorization and back-substitution iteratively until our Newton method converges, which typically takes three to four iterations with a backtracking line search to ensure the residual decreases. Once converged, we extract $\boldsymbol{v}_2$ and $\boldsymbol{\omega}_2$ from our solution $\boldsymbol{s}$ to update $\boldsymbol{x}_3$ and $\boldsymbol{q}_3$ according to (27) and (32). Subsequently, we use $\boldsymbol{s}$ as the initial guess for the next time step.

4.3 Extension to Closed-Loop Mechanisms

The presented algorithms achieve linear-time complexity for loop-free structures only. If we introduce constraints that form loops in the mechanism, we cannot avoid creating fill-ins during the factorization. Deriving an optimal factorization, for example with the minimal possible number of fill-ins, goes beyond the scope of this paper. Nonetheless, we can give a simple strategy to extend our sparse solver to detect and handle loop-closure constraints.

Mathematically, we treat closed-loop mechanisms just as loop-free ones. This is a considerable advantage for users as no adjustments are required. Algorithmically, loop-closure constraints can be found with the already deployed depth-first-search. We stack all constraints that enforce loop closure into a single node and append it to the end of our depth-first-search list. In this manner, we can process the entire system with the same algorithms as before except for the very last node. For this node, we will perform the standard dense LDU factorization and back-substitution of Algorithms (1) and (2). If there are only a few of these additional constraints, the majority of our algorithm can still perform in linear-time and only the processing of the last node containing all loop-closure constraints requires additional computational effort.

5 Experiments and Comparison

We have implemented our algorithm in the programming language Julia [3]. It has been recently shown that Julia can achieve performance comparable to state-of-the-art C++ dynamics implementations [11]. The code for our algorithm and all experiments is available at: https://github.com/RoboticExplorationLab. All experiments were performed on an ASUS ZenBook with an Intel i7 processor.

5.1 Linear Time, Energy Conservation, and No Constraint Drift

To validate the main properties of our algorithm—energy conservation, elimination of constraint drift, and $O(n)$ complexity—we ran three representative experiments demonstrating qualitative behavior in Fig. 4 where we also show typical convergence behavior.

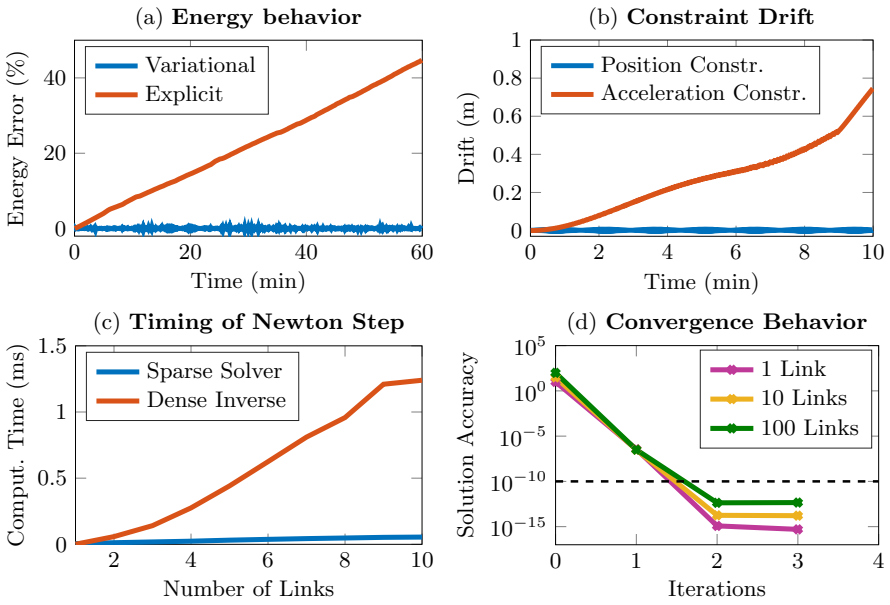

Fig. 4: Main algorithm properties. Our algorithm in blue, comparison in red. (a) Energy conservation with variational integrator compared to explicit integrator. (b) Drift with position constraints compared to drift with acceleration constraints. (c) Sparse $O(n)$ behavior compared to $O(n^3)$ dense matrix inversion. (d) Convergence behavior of Newton's method in our algorithm.

For plot (a) we measured the energy of a double pendulum over 60 minutes with a time step $\Delta t = 10$ms. We compare our first-order variational integrator to the explicit second-order Heun's method (a Runga Kutta method), using the *DifferentialEquations.jl* package [18]. The error increases substantially over time with the explicit method, a behavior typical for (lower-order) explicit integrators, whereas the energy stays constant over time with our variational approach.

To determine drift behavior, for plot (b) we simulated a three-link closed-loop mechanism with our maximal-coordinate approach and compared it to a state-of-the-art minimal-coordinate dynamics package, *RigidBodyDynamics.jl* [11]. The loop closure introduces constraints to the minimal-coordinate approach. As the minimal-coordinate approach only enforces acceleration constraints, without constraint stabilization, an increasing drift becomes visible. In contrast, our

algorithm directly enforces position constraints, which guarantees constraint satisfaction without requiring any additional constraint stabilization.

Plot (c) compares the computation time of sparse and dense single Newton steps for mechanisms with varying numbers of links. The rapid increase in computation time for a larger number of links when using dense inversion compared to our sparse approach highlights the necessity to exploit sparsity to achieve good performance.

We demonstrate quadratic convergence behavior, i.e. doubling accuracy at each iteration step, for simulating a single time step of pendulums with 1, 10, and 100 links in plot (d). The termination condition is a converged solution $\|\Delta \boldsymbol{s}^{(i+1)} - \Delta \boldsymbol{s}^{(i)}\| < \epsilon$ with a tolerance of $\epsilon = 10^{-10}$.

5.2 Performance Comparison

We compare the performance of our algorithm to three minimal-coordinate implementations: the *RigidBodyDynamics.jl* package with a state-of-the-art explicit Runge-Kutta-Munthe-Kaas integrator [9], a second-order variational integrator using a quasi-Newton method [13], and a third-order variational integrator using an exact Newton method [7]. Both variational integrators are implemented in C++. All experiments use the same initial conditions for all algorithms and the best timing result of 100 runs was taken.

Comparison with Explicit Integrator For the comparison with the explicit integrator, we simulated swinging pendulums—having snake robots [21] or finite-element-method soft robotics in mind [6]—with varying numbers of links for 10

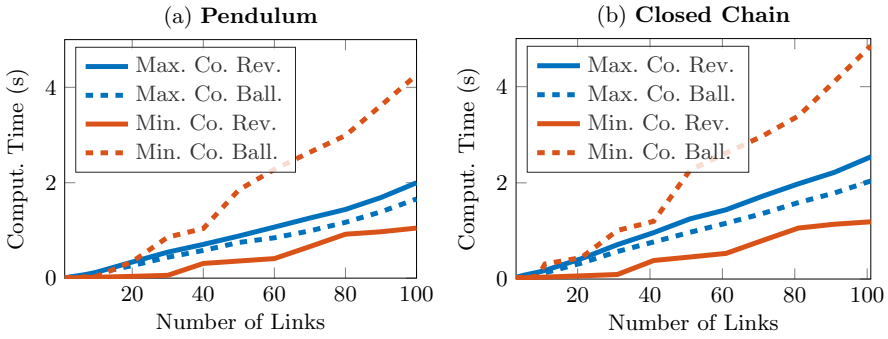

Fig. 5: Performance comparison simulating 1000 time steps of structures with different numbers of links. Our algorithm (maximal coordinates) in blue, explicit integrator (minimal coordinates) in red. Revolute joint dashed line, ball-and-socket joint solid line. (a) Pendulum. (b) Closed-loop mechanisms.

seconds and a time step $\Delta t = 10$ms. We show the computation time for revolute joints (1 degree of freedom) and ball-and-socket joints (3 degrees of freedom) in

Fig. 5. We also introduced a single loop closure and ran the same experimental setup for these closed-loop structures with results shown in Fig. 5.

In both the open and closed-loop experiments, the linear-time behavior is clearly visible. While the explicit integrator performs better for revolute joints, especially with few degrees of freedom, our algorithm achieves competitive results and outperforms the explicit integrator for ball-and-socket joints. Note that our method is faster at simulating ball-and-socket joints than revolute joints because a ball-and-socket joint constrains fewer degrees of freedom.

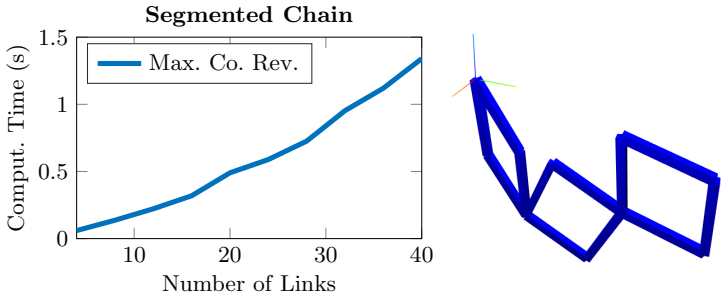

Fig. 6: Performance simulating 1000 time steps of different chains consisting of four-link segments (example of a chain consiting of three 4-link segments on the right). Timing results for our algorithm.

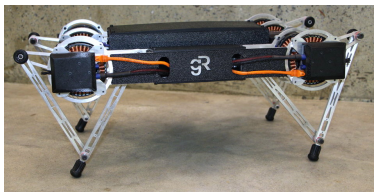

Fig. 7: Quadrupedal robot *Minitaur* [4] with closed-loop legs.

To highlight the robustness of our algorithm, we also simulated a chain of four-link closed-loop segments with revolute joints (see Fig. 6). Such closed-loop segments can be used as legs for quadrupedal robots, for example *Minitaur* [4], shown in Fig. 7. The segments have singular configurations when links overlap. Such kinematic singularities are generally difficult to handle with minimal-coordinate approaches. The results for our algorithm simulating the chains of different lengths for 10 seconds with $\Delta t = 10$ms are displayed in Fig. 6. The minimal-coordinate integrator was unable to reliably simulate chains of two or more four-link segments, and using smaller time steps did not lead to a significant improvement.

Comparison with Variational Integrators As a final experiment, we compared our algorithm to state-of-the-art second- and third-order variational integrators with the same n-link pendulum we used for the comparison to the explicit integrator. Only revolute joints and loop-free structures were tested as ball-and-socket joints and loop-closure were, to the best of our knowledge, not part of the implementions of these integrators. To highlight the robustness of

our algorithm, we simulated with solution tolerances of 10^{-6}, 10^{-8}, and 10^{-10} for the Newton methods in all algorithms. The results are given in Fig. 8.

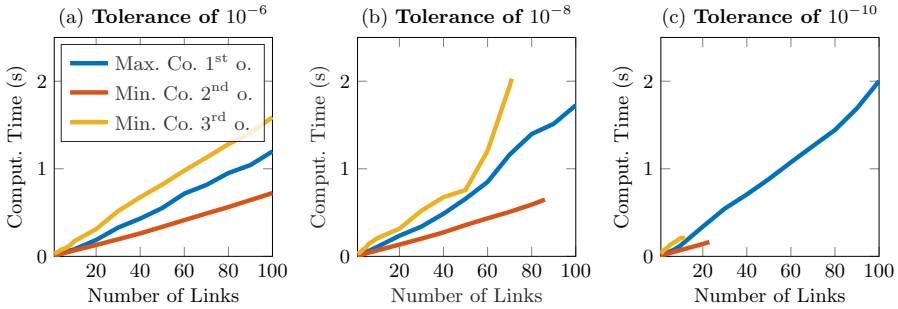

Fig. 8: Performance comparison simulating 1000 time steps of n-link pendulums with varying solution tolerances. Our (1^{st} order) algorithm in blue, 2^{nd} order variational integrator in red, 3^{rd} order variational integrator in yellow. (a) Tolerance of 10^{-6}. (b) Tolerance of 10^{-8}. (c) Tolerance of 10^{-10}.

While all variational integrators were able to simulate the pendulum successfully for a tolerance of 10^{-6}, a larger number of links and lower tolerances lead to failure to converge to a solution for the minimal-coordinate variational integrators (stopped after 100 Newton steps). By contrast, the LDU factorization used in the maximal-coordinate setting performs robustly and scales well.

6 Conclusion

We have presented a novel linear-time variational integrator in maximal coordinates to simulate open and closed-loop mechanical structures without constraint drift. To the best of our knowledge, no previous attempt has been made to develop a linear-time variational integrator in maximal coordinates.

Overall, the results of our experiments show that our algorithm is competitive in terms of computational performance to minimal-coordinate integrators, both explicit and variational. Especially for mechanical systems with a larger number of links and higher-degree-of-freedom joints our method shows better performance for both open and closed-loop structures. Thanks to the numerical stability and robustness of our specialized sparse LDU factorization, our method also achieves tighter numerical tolerances than comparable minimal-coordinate variational integrators. For structures with multiple closed kinematic loops and kinematic singularities, we perform robustly while a minimal-coordinate integrator frequently fails to find solutions.

In summary, these results make a strong case for the use of maximal-coordinate integrators, especially when dealing with complex, high-degree-of-freedom, closed-loop structures. Additionally, the modular structure of maximal-coordinate approaches opens up the possibility to parallelize computations in the future.

References

1. D. Baraff. Linear-Time Dynamics Using Lagrange Multipliers. In *ACM SIGGRAPH 96*, pages 137–146, 1996.
2. J. Baumgarte. Stabilization of Constraints and Integrals of Motion in Dynamical Systems. *Computer Methods in Appl. Mechanics and Engineering*, 1(1):1–16, 1972.
3. J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, 2017.
4. D. Blackman, J Nicholson, C Ordonez, B. Miller, and J. Clark. Gait Development on Minitaur, a Direct Drive Quadrupedal Robot. In *SPIE Defense+Security*, 2016.
5. I. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 2017.
6. C. Duriez. Control of Elastic Soft Robots based on Real-Time Finite Element Method. In *2013 International Conference on Robotics and Automation (ICRA)*, 2016.
7. T. Fan, J. Schultz, and T. Murphey. Efficient Computation of Higher-Order Variational Integrators in Robotic Simulation and Trajectory Optimization. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.
8. R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2008.
9. E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration*. Springer, 2006.
10. O. Junge, J. Marsden, and S. Ober-Blöbaum. Discrete Mechanics and Optimal Control. *IFAC Proceedings Volumes*, 38(1):538–543, 2005.
11. T. Koolen and R. Deits. Julia for Robotics: Simulation and Real-Time Control in a High-Level Programming Language. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 604–611, 2019.
12. J. Kwak and H. Sungpyo. *Linear Algebra*. Birkhäuser, 2004.
13. J. Lee, C. Liu, F. Park, and S. Srinivasa. A Linear-Time Variational Integrator for Multibody Systems. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.
14. M. Macklin, M. Müller, N. Chentanez, and T. Kim. Unified Particle Physics for Real-Time Applications. *ACM Transactions on Graphics*, 33(4):1–12, 2014.
15. Z. Manchester and M. Peck. Quaternion Variational Integrators for Spacecraft Dynamics. *Journal of Guidance, Control, and Dynamics*, 39(1):69–76, 2016.
16. J. Marsden and M. West. Discrete Mechanics and Variational Integrators. *Acta Numerica*, 10:357–514, 2001.
17. S. Ober-Blöbaum and N. Saake. Construction and Analysis of Higher Order Galerkin Variational Integrators. *Advances in Computational Mathematics*, 41:955–986, 2015.
18. C. Rackauckas and Q. Nie. DifferentialEquations.jl A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 5(1):15, 2017.
19. B. Siciliano and O. Khatib. *Springer Handbook of Robotics*. Springer, 2016.
20. J. Solà. Quaternion Kinematics for the Error-State Kalman Filter. *arXiv e-prints*, arXiv:1711.02508 [cs.RO], 2017.
21. M. Tesch, K. Lipkin, I. Brown, R. Hatton, A. Peck, J. Rembisz, and H. Choset. Parameterized and Scripted Gaits for Modular Snake Robots. *Advanced Robotics*, 23(9):1131–1158, 2009.
22. T. Wenger, S. Ober-Blöbaum, and S. Leyendecker. Construction and Analysis of Higher Order Variational Integrators for Dynamical Systems with Holonomic Constraints. *Advances in Computational Mathematics*, 43(5):1163–1195, 2017.

Supplementary Material for: Linear-Time Variational Integrators in Maximal Coordinates

Jan Brüdigam and Zachary Manchester

Stanford University, Stanford CA 94305, USA,
{bruedigam, zacm}@stanford.edu

Rotational Gradient

In (31) (Sec. 3.2) we have used the *rotational gradient* ∇^{r}. We want to explain in more detail the idea of how to derive this gradient an why it is necessary. We start by revisiting the gradient of a function $f\colon \mathbb{R}^n \to \mathbb{R}$. The gradient of this function is defined as

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}, \tag{40}$$

and tells us what change of function value we get for changing values of $\boldsymbol{x}$. Stated more formally, we add a small deviation $\boldsymbol{\epsilon}$ to $\boldsymbol{x}$ and observe the relative change

$$\frac{f(\boldsymbol{x}+\boldsymbol{\epsilon}) - f(\boldsymbol{x})}{\boldsymbol{\epsilon}}. \tag{41}$$

Taking the limit of this relative change component-wise as $\boldsymbol{\epsilon}$ goes to zero yields

$$\lim_{\boldsymbol{\epsilon}\to\mathbf{0}} \frac{f(\boldsymbol{x}+\boldsymbol{\epsilon}) - f(\boldsymbol{x})}{\boldsymbol{\epsilon}} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{42}$$

which is simply the gradient (40).

For rotations, a small deviation can be described as

$$\boldsymbol{d} = \begin{bmatrix} 1 \\ \boldsymbol{\epsilon} \end{bmatrix}, \tag{43}$$

since zero rotation would evaluate to the identity quaternion $[1 \quad \mathbf{0}^{\mathrm{T}}]$. "Adding" this small deviation $\boldsymbol{d}$ to a quaternion $\boldsymbol{q}$ amounts to multiplication, similar to "adding" rotations with rotation matrices:

$$\boldsymbol{q} \otimes \boldsymbol{d}. \tag{44}$$

We can rewrite (44) as

$$\boldsymbol{q} \otimes \boldsymbol{d} = \begin{bmatrix} q_w - \boldsymbol{q}_v^{\mathrm{T}}\boldsymbol{\epsilon} \\ \boldsymbol{q}_v + q_w\boldsymbol{\epsilon} + \boldsymbol{q}_v \times \boldsymbol{\epsilon} \end{bmatrix} = \boldsymbol{q} + L(\boldsymbol{q})V^{\mathrm{T}}\boldsymbol{\epsilon}. \tag{45}$$

Now, if we have function $f(\boldsymbol{q})$ with some rotation described by a quaternion $\boldsymbol{q}$, we can again take a look at the change $f(\boldsymbol{q} \otimes \boldsymbol{d}) - f(\boldsymbol{q})$ for a small deviation $\boldsymbol{d}$. Taking the limit as in (42) and using the linear approximation

$$f(\boldsymbol{q} + \Delta\boldsymbol{q}) = f(\boldsymbol{q}) + \Delta\boldsymbol{q}^{\mathrm{T}} \nabla_{\boldsymbol{q}} f(\boldsymbol{q}) \quad \text{(true for small } \Delta\boldsymbol{q}) \tag{46}$$

we obtain the rotational gradient $\nabla^{\mathrm{r}}_{\boldsymbol{q}}$ after simplification:

$$\lim_{\boldsymbol{\epsilon}\to\boldsymbol{0}} \frac{f(\boldsymbol{q} \otimes \boldsymbol{d}) - f(\boldsymbol{q})}{\boldsymbol{\epsilon}} = \lim_{\boldsymbol{\epsilon}\to\boldsymbol{0}} \frac{f\left(\boldsymbol{q} + L(\boldsymbol{q})V^{\mathrm{T}}\boldsymbol{\epsilon}\right) - f(\boldsymbol{q})}{\boldsymbol{\epsilon}} \tag{47}$$

$$= \lim_{\boldsymbol{\epsilon}\to\boldsymbol{0}} \frac{f(\boldsymbol{q}) + \boldsymbol{\epsilon}^{\mathrm{T}} V L(\boldsymbol{q})^{\mathrm{T}} \nabla_{\boldsymbol{q}} f(\boldsymbol{q}) - f(\boldsymbol{q})}{\boldsymbol{\epsilon}} \tag{48}$$

$$= V L(\boldsymbol{q})^{\mathrm{T}} \nabla_{\boldsymbol{q}} f(\boldsymbol{q}) := \nabla^{\mathrm{r}}_{\boldsymbol{q}} f(\boldsymbol{q}) \tag{49}$$

It is important to note that the rotational gradient $\nabla^{\mathrm{r}}_{\boldsymbol{q}} f(\boldsymbol{q}) \in \mathbb{R}^3$ gives us information in the actual three-dimensional space of rotations despite a quaternion having four parameters.

Rotational Equations of Motion

In Sec. 3.2 we stated results for the rotational equations of motion derived from a variational principle. Here, we want to offer more detail on the derivation. The two main differences to the translational case are: (a) using an implicit unit norm constraint on quaternions to avoid tracking of the unit norm explicitly in a constraint and (b) applying the rotational gradient appropriately.

The quaternion angular velocity $\widetilde{\boldsymbol{\omega}}$ is defined as (cf. (16))

$$\widetilde{\boldsymbol{\omega}} = 2L(\boldsymbol{q})^{\mathrm{T}} \dot{\boldsymbol{q}} = \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}. \tag{50}$$

However, when using the discrete approximation of $\dot{\boldsymbol{q}}$ in (30), we get a discrete angular velocity

$$\widetilde{\boldsymbol{\omega}}_k = \begin{bmatrix} \widetilde{\omega}_{k,w} \\ \boldsymbol{\omega}_k \end{bmatrix}, \tag{51}$$

where $\widetilde{\omega}_{k,w} \neq 0$.

We can update the orientation $\boldsymbol{q}_{k+1}$ at the next time step with this discrete angular velocity by discretizing and rearranging (50):

$$\boldsymbol{q}_{k+1} = \frac{\Delta t}{2} L(\boldsymbol{q}_k) \widetilde{\boldsymbol{\omega}}_k + \boldsymbol{q}_k. \tag{52}$$

This update, however, will not maintain unit norm without an additional constraint because of $\widetilde{\omega}_{k,w} \neq 0$. We will therefore derive an expression for $\widetilde{\omega}_{k,w}$

similar to [15], that ensures unit norm:

$$\left\|\boldsymbol{q}_{k+1}\right\| = 1 \tag{53}$$

$$\left\|\frac{\Delta t}{2} L(\boldsymbol{q}_k)\widetilde{\boldsymbol{\omega}}_k + \boldsymbol{q}_k\right\| = 1 \tag{54}$$

$$\Rightarrow \widetilde{\omega}_{k,w} = \sqrt{\left(\tfrac{2}{\Delta t}\right)^2 - \boldsymbol{\omega}_k^{\mathrm{T}}\boldsymbol{\omega}_k} - \frac{2}{\Delta t} \tag{55}$$

This result was then used for the update rule (32) and to calculate the equations of motion (33).

Applying the rotational gradient to calculate the equations of motion is done as described in the previous section. However, there is a slight subtlety with the torque term in (29). The correct derivative of this term is:

$$\nabla^{\mathrm{r}}_{\boldsymbol{q}_k} 2\boldsymbol{\tau}_k^{\mathrm{T}} V L(\boldsymbol{q}_k)^{\mathrm{T}}\boldsymbol{q}_k = 2V L(\boldsymbol{q}_k)^{\mathrm{T}} L(\boldsymbol{q}_k) V^{\mathrm{T}}\boldsymbol{\tau}_k \tag{56}$$

Note that in this case, we are only taking the derivative with respect to the explicit $\boldsymbol{q}_k$ and we do not take the derivative of $L(\boldsymbol{q}_k)^{\mathrm{T}}$. This special differentiation rule arises from the derivation of the generalized force corresponding to torque $\boldsymbol{\tau}$. More details can be found in [15].