Linear-Time Variational Integrators in Maximal Coordinates

Jan Brüdigam and Zachary Manchester

Stanford University, Stanford CA 94305, USA,
{bruedigam, zacmanchester}@stanford.edu,
WWW home page: https://rexlab.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 existing minimal-coordinate algorithms while outperforming them in several scenarios, especially when dealing with closed loops and configuration singularities.

Keywords: Maximal Coordinates, Dynamics, Variational Integrators, Discrete Mechanics, 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 the whole 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 [6,1], constraint drift when using maximal coordinates is still an issue and requires stabilization schemes [2,11].

A separate and more recent area of research regarding rigid body dynamics is concerned with *variational integrators* [13]. 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 [13,7].

In the last few years, algorithms for calculating rigid body dynamics in linear time with variational integrators in minimal coordinates have been presented [10,5]. 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 non-holonomic settings require explicit constraints, which diminish the advantages of minimal coordinates since these constraints can no longer be eliminated 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 extend this algorithm by loop-closure constraints with limited increase in computation time.

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

2 Background

We will give a brief review of quaternions (see [17] for details) and provide an example of two rigid bodies and how their connection with a joint can be described. Additionally, we state the well-known factorization and back-substitution algorithms for an LDU decomposition [cite some book] which we will need in order to develop our linear-time algorithm.

2.1 Quaternions

Quaternions are a natural choice to represent the orientation of a rigid body as they do not encounter the gimbal lock singularities present in Euler angles or other 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 $\boldsymbol{q}$ always maps vectors from the local to the global frame, whereas its inverse always maps from the global to the local frame.

This notation 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:

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

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

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

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

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

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

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

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

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

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

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

and do the same for the angular velocity

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

The newly introduced matrices also allow us to state the *rotational gradient* ∇^r of a function $f(\boldsymbol{q}) \in \mathbb{R}$ as (see appendix for detailed derivation)

$$\nabla^\text{r}_{\boldsymbol{q}} f(\boldsymbol{q}) = VL(\boldsymbol{q})^\text{T}\nabla_{\boldsymbol{q}} f(\boldsymbol{q}) \quad \in \mathbb{R}^3, \tag{17}$$

where $\nabla_{\boldsymbol{q}}$ is the regular gradient with respect to $\boldsymbol{q}$. The rotational gradient is necessary to give three-dimensional derivatives of functions despite the four parameters of a quaternion.

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 $\boldsymbol{\omega} = 2L(\boldsymbol{q})^\text{T}\dot{\boldsymbol{q}}$, and a moment of inertia matrix J. All quantities refer to the center of mass of a body.

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 a ball-and-socket joint connecting bodies a and b

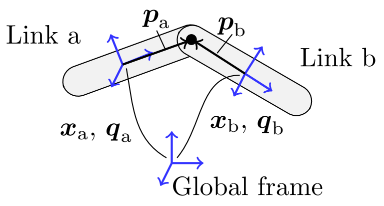
Link a
$\boldsymbol{p}_\text{a}$
$\boldsymbol{p}_\text{b}$
Link b
$\boldsymbol{x}_\text{a}, \boldsymbol{q}_\text{a}$
$\boldsymbol{x}_\text{b}, \boldsymbol{q}_\text{b}$
Global frame

Fig. 1: Two links connected by a joint.

$$\boldsymbol{g} = \boldsymbol{x}_\text{a} + VR(\boldsymbol{q}_\text{a})^\text{T}L(\boldsymbol{q}_\text{a})V^\text{T}\boldsymbol{p}_\text{a} \quad - \quad \left(\boldsymbol{x}_\text{b} + VR(\boldsymbol{q}_\text{b})^\text{T}L(\boldsymbol{q}_\text{b})V^\text{T}\boldsymbol{p}_\text{b}\right), \tag{18}$$

where $\boldsymbol{p}$ is a vector from the center of mass of a body to the joint connection point. For $\boldsymbol{g} = \boldsymbol{0}$, the two bodies are properly connected. Figure 1 shows a sketch of two bodies connected by such a joint. See [16] for more details on joint kinematics.

2.3 LDU Decomposition

An LDU decomposition is a numerically robust method to factorize and solve square linear systems of equations $F\boldsymbol{s} = \boldsymbol{f}$ with non-symmetric F. 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.

Algorithm 1 Dense In-Place LDU Factorization

```
1: for $n = 1 : \text{length}(\boldsymbol{s})$ do
2:     for $i = 1 : n - 1$ do                                  ▷ L and U factorization
3:         for $j = 1 : n-$ do
4:             $F_{n,i} \leftarrow F_{n,i} - F_{n,j} F_{j,j} F_{j,i}$          ▷ L
5:             $F_{i,n} \leftarrow F_{i,n} - F_{i,j} F_{j,j} F_{j,n}$          ▷ U
6:         $F_{n,i} \leftarrow F_{n,i} F_{i,i}^{-1}$                              ▷ L
7:         $F_{i,n} \leftarrow F_{i,i}^{-1} F_{i,n}$                              ▷ U
8:     for $j = 1 : n - 1$ do                                  ▷ D factorization
9:         $F_{n,n} \leftarrow F_{n,n} - F_{n,j} F_{j,j} F_{j,n}$              ▷ 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" (hence the name), with their product being LDU $= F$. The structures of matrices L, D, and U allow for efficient back-substitution (complexity $O(n^2)$) of $\boldsymbol{s} = \text{U}^{-1}\text{D}^{-1}\text{L}^{-1}\boldsymbol{f}$ as described in Algorithm 2.

Algorithm 2 Dense LDU Back-Substitution

```
1: $\boldsymbol{s} \leftarrow \boldsymbol{f}$
2: for $n = 1 : \text{length}(\boldsymbol{s})$ do                      ▷ L back-substitution
3:     for $j = 1 : n - 1$ do
4:         $\boldsymbol{s}_n \leftarrow \boldsymbol{s}_n - F_{n,j} \boldsymbol{s}_j$                         ▷ L
5: for $n = \text{length}(\boldsymbol{s}) : -1 : 1$ do            ▷ D and U back-substitution
6:     $\boldsymbol{s}_n \leftarrow F_{n,n}^{-1} \boldsymbol{s}_n$                                    ▷ D
7:     for $j = n + 1 : \text{length}(\boldsymbol{s})$ do                         ▷ U
8:         $\boldsymbol{s}_n \leftarrow \boldsymbol{s}_n - F_{n,j} \boldsymbol{s}_j$
```

3 Variational Integrator

We now derive a first-order variational integrator from the *principle of least action* (also called Hamilton's principle). This integrator allows us to discretize rigid body dynamics while maintaining properties such as realistic energy and momentum conservation behavior. While the derivation is generally not new and has been treated rigorously [13,8,18,12], we provide a derivation in maximal coordinates with unified notation. For clarity, we only treat the translational component of the integrator in more detail as the rotational derivation is obtained in similar fashion.

The fundamental proposition of the principle of least action is 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] $\times$ [Time] and the unforced action integral S_0 is defined as

$$S_0 = \int_{t_1}^{t_N} \mathcal{L}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t))\ \mathrm{d}t. \tag{19}$$

where $\mathcal{L}$ is the Lagrangian of the system. The Lagrangian is defined as

$$\mathcal{L} = \mathcal{T} - \mathcal{V}, \tag{20}$$

with kinetic energy $\mathcal{T}$ and potential energy $\mathcal{V}$. For the translational setting the kinetic energy is

$$\mathcal{T}_\mathrm{T} = \frac{1}{2}\boldsymbol{v}^\mathrm{T} M \boldsymbol{v}, \tag{21}$$

and for this paper we assume only gravitational potential energy, yielding

$$\mathcal{V}_\mathrm{T} = g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x} \tag{22}$$

with position $\boldsymbol{x}$, velocity $\boldsymbol{v}$, mass matrix $M = mI_3$, gravitational acceleration g, and the z-axis vector $\boldsymbol{e}_\mathrm{z}$.

Inserting the Lagrangian for the translational case into (19) allows us to approximate the unforced action integral $S_{0,\mathrm{T}}$ by the following unforced discrete action sum $S_{\mathrm{d},0,\mathrm{T}}$:

$$S_{0,\mathrm{T}}(\boldsymbol{x}) = \int_{t_1}^{t_N} \left(\frac{1}{2}\boldsymbol{v}^\mathrm{T} M \boldsymbol{v} - g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x}\right) \mathrm{d}t = \sum_{k=1}^{N-1} \int_{t_k}^{t_{k+1}} \left(\frac{1}{2}\boldsymbol{v}^\mathrm{T} M \boldsymbol{v} - g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x}\right) \mathrm{d}t \tag{23}$$

$$\approx \sum_{k=1}^{N-1} \left(\frac{1}{2}\boldsymbol{v}_k^\mathrm{T} M \boldsymbol{v}_k - g\boldsymbol{e}_\mathrm{z}^\mathrm{T} M \boldsymbol{x}_k\right) \Delta t = S_{\mathrm{d},0,\mathrm{T}}(\boldsymbol{x}) \tag{24}$$

Here, we have used a first-order discretization of the integral and the first-order approximation of the velocity

$$\boldsymbol{v}_k = \frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t}. \tag{25}$$

External forces $\boldsymbol{F}$ and constraints $\boldsymbol{g}$ with virtual constraint forces $\boldsymbol{\lambda}$ can be added to the action integral as follows,

$$S_\mathrm{T}(\boldsymbol{x}) = \int_{t_0}^{t_N} \left(\frac{1}{2}\boldsymbol{v}^\mathrm{T} M \boldsymbol{v} - V(\boldsymbol{x})\right)\ \mathrm{d}t + \int_{t_0}^{t_N} \boldsymbol{F}^\mathrm{T}\boldsymbol{x}\ \mathrm{d}t + \int_{t_0}^{t_N} \boldsymbol{\lambda}^\mathrm{T}\boldsymbol{g}(\boldsymbol{x})\ \mathrm{d}t, \tag{26}$$

to form the forced and constrained discrete action sum:

$$S_{\mathrm{d,T}}(\boldsymbol{x}) = \sum_{k=1}^{N-1} \left(\frac{1}{2} \boldsymbol{v}_k^{\mathrm{T}} M \boldsymbol{v}_k - g \boldsymbol{e}_{\mathrm{z}}^{\mathrm{T}} M \boldsymbol{x}_k + \boldsymbol{F}_k^{\mathrm{T}} \boldsymbol{x}_k + \boldsymbol{\lambda}_k^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}_k) \right) \Delta t \tag{27}$$

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.

With our discrete action sum we can now analyze a very short trajectory with only three time steps, t_1 to t_3, that satisfies the principle of least action. Since $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ are fixed end points, only $\boldsymbol{x}_2$ can vary. We therefore minimize the discrete action sum with respect to $\boldsymbol{x}_2$:

$$\begin{aligned} \nabla_{\boldsymbol{x}_2} S_{\mathrm{d,T}}(\boldsymbol{x}_2) = \nabla_{\boldsymbol{x}_2} \Bigg(& \frac{1}{2} \frac{(\boldsymbol{x}_2 - \boldsymbol{x}_1)^{\mathrm{T}}}{\Delta t} M \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{\Delta t} + \frac{1}{2} \frac{(\boldsymbol{x}_3 - \boldsymbol{x}_2)^{\mathrm{T}}}{\Delta t} M \frac{\boldsymbol{x}_3 - \boldsymbol{x}_2}{\Delta t} \\ & - g \boldsymbol{e}_{\mathrm{z}}^{\mathrm{T}} M \boldsymbol{x}_2 + \boldsymbol{F}_2^{\mathrm{T}} \boldsymbol{x}_2 + \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}_2) \Bigg) \Delta t \\ = & \left(\frac{1}{\Delta t} M \boldsymbol{v}_1 - \frac{1}{\Delta t} M \boldsymbol{v}_2 - g M \boldsymbol{e}_{\mathrm{z}} + \boldsymbol{F}_2 + G_{\boldsymbol{x}}(\boldsymbol{x}_2)^{\mathrm{T}} \boldsymbol{\lambda}_2 \right) \Delta t = \boldsymbol{0}, \end{aligned} \tag{28}$$

where $G_{\boldsymbol{x}}$ is the Jacobian of constraints $\boldsymbol{g}$ with respect to $\boldsymbol{x}$. In words, if equation (28) 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 (28) yields the *discretized translational equations of motion* for a single rigid body with forcing and constraints:

$$\boldsymbol{d}_{\mathrm{T}}(\boldsymbol{v}_2, \boldsymbol{\lambda}_2) = M \left(\frac{\boldsymbol{v}_2 - \boldsymbol{v}_1}{\Delta t} + g \boldsymbol{e}_{\mathrm{z}} \right) - \boldsymbol{F}_2 - G_{\boldsymbol{x}}(\boldsymbol{x}_2)^{\mathrm{T}} \boldsymbol{\lambda}_2 = \boldsymbol{0} \tag{29}$$

As an aside, one can note the similarity between (29) and Newton's second law $M\boldsymbol{a} - \boldsymbol{F} = \boldsymbol{0}$.

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 [19,14]. However, since our goal in this paper is to demonstrate general capabilities of variational integrators in maximal coordinates, we will only use the first-order integrator.

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 (25) as

$$\boldsymbol{x}_3 = \boldsymbol{x}_2 + \boldsymbol{v}_2 \Delta t, \tag{30}$$

which means we have to solve (29) implicitly for $\boldsymbol{v}_2$ while also finding appropriate constraint forces $\boldsymbol{\lambda}_2$.

The rotational case can be derived similarly to the translation case. More details on the derivation and some intricacies can be found in the appendix and the literature [12]. We will only state the update rule for $\boldsymbol{q}_3$

$$\boldsymbol{q}_3 = \frac{\Delta t}{2} L(\boldsymbol{q}_2) \begin{bmatrix} \sqrt{\left(\frac{2}{\Delta t}\right)^2 - \boldsymbol{\omega}_2^{\mathrm{T}} \boldsymbol{\omega}_2} \\ \boldsymbol{\omega}_2 \end{bmatrix} \tag{31}$$

and the resulting *discretized rotational equations of motion*

$$\begin{aligned} \boldsymbol{d}_{\mathrm{R}}(\boldsymbol{\omega}_2, \boldsymbol{\lambda}_2) = J\boldsymbol{\omega}_2 \sqrt{\tfrac{4}{\Delta t^2} - \boldsymbol{\omega}_2^{\mathrm{T}} \boldsymbol{\omega}_2} + \boldsymbol{\omega}_2^{\times} J \boldsymbol{\omega}_2 \\ - J\boldsymbol{\omega}_1 \sqrt{\tfrac{4}{\Delta t^2} - \boldsymbol{\omega}_1^{\mathrm{T}} \boldsymbol{\omega}_1} + \boldsymbol{\omega}_1^{\times} J \boldsymbol{\omega}_1 - 2\boldsymbol{\tau}_2 - G_{\boldsymbol{q}_2}(\boldsymbol{q}_2)^{\mathrm{T}} \boldsymbol{\lambda}_2 = \mathbf{0}, \end{aligned} \tag{32}$$

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

4 Linear-Time Sparse Solver

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

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{g} \end{bmatrix}. \tag{33}$$

If we want to treat multiple rigid bodies, connected or not, we can stack their equations as well and write everything 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

$$\boldsymbol{f}(\boldsymbol{s}) = \mathbf{0}, \tag{34}$$

where the solution, $\boldsymbol{s}$, consists of $\boldsymbol{v}_2$, $\boldsymbol{\omega}_2$, and $\boldsymbol{\lambda}_2$. Once we have found these values, we can use $\boldsymbol{v}_2$ and $\boldsymbol{\omega}_2$ to update the mechanism's position and orientation according to (30) and (31).

4.1 Newton's method

Newton's method is used to solve the implicit equation (34):

$$\begin{aligned} \boldsymbol{s}^{(i+1)} &= \boldsymbol{s}^{(i)} - \left(F(\boldsymbol{s}^{(i)})\right)^{-1} \boldsymbol{f}(\boldsymbol{s}^{(i)}) && (35) \\ &= \boldsymbol{s}^{(i)} - \Delta \boldsymbol{s}^{(i)}, && (36) \end{aligned}$$

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

$$F = \frac{\partial \boldsymbol{f}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)} = \begin{bmatrix} \frac{\partial \boldsymbol{d}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial(\boldsymbol{v}_2, \boldsymbol{\omega}_2)} & \frac{\partial \boldsymbol{d}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial \boldsymbol{\lambda}_2} \\ \frac{\partial \boldsymbol{g}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial(\boldsymbol{v}_2, \boldsymbol{\omega}_2)} & \frac{\partial \boldsymbol{g}(\boldsymbol{v}_2, \boldsymbol{\omega}_2, \boldsymbol{\lambda}_2)}{\partial \boldsymbol{\lambda}_2} \end{bmatrix} = \begin{bmatrix} D_{\boldsymbol{v},\boldsymbol{\omega}} & G_{\boldsymbol{x},\boldsymbol{q}}^{\mathrm{T}} \\ G_{\boldsymbol{v},\boldsymbol{\omega}} & 0 \end{bmatrix} \tag{37}$$

For computational efficiency, instead of directly inverting the Jacobian in (35), one solves the linear system

$$F(\boldsymbol{s}^{(i)}) \Delta \boldsymbol{s}^{(i)} = \boldsymbol{f}(\boldsymbol{s}^{(i)}). \tag{38}$$

For a dense matrix F with few non-zeros, a linear system of equations like (38) is typically solved by using factorizations like the LDU decomposition, with a complexity of $O(n^3)$, followed by back-substitution, with a complexity of $O(n^2)$. Thankfully, the Jacobian F in our problem is very sparse and has structural properties that enable much faster solution of (38).

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 (38). While the continuous-time system is symmetric, allowing the use of a modified LDL$^{\mathrm{T}}$ factorization, (37) 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 (37) 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 constraints. 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 Figure 2.

We can factorize a matrix associated with the graph in Figure 2 (b) efficiently by traversing the graph from the leafs to the root. Leafs 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

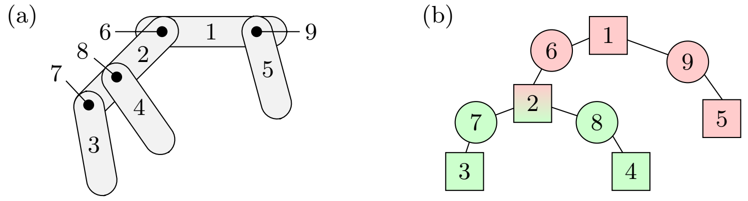

Fig. 2: (a) A mechanism with five links. (b) A graph representing the mechanism and its matrix. Squares represent links, circles represent joints. Coloring represents different levels of the tree-shaped graph.

graph in this order during factorization, we will not create *fill-ins*, i.e. we will not change any zero blocks [4]. Since the entries of our matrix F are not symmetric, we cannot use an LDL$^{\mathrm{T}}$ factorization and will instead modify the procedure of the LDU factorization described in section 2.3 which can handle non-symmetric matrices.

If we reorder the rows and columns of a matrix, we can change the processing order of the factorization. Reordering a matrix from leafs to root according 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 Figure 3.

$$
\text{(a)} \begin{pmatrix}
D_1 & 0 & 0 & 0 & 0 & c'_{61} & 0 & 0 & c'_{91} \\
0 & D_2 & 0 & 0 & 0 & c'_{62} & c'_{72} & c'_{82} & 0 \\
0 & 0 & D_3 & 0 & 0 & 0 & c'_{73} & 0 & 0 \\
0 & 0 & 0 & D_4 & 0 & 0 & 0 & c'_{84} & 0 \\
0 & 0 & 0 & 0 & D_5 & 0 & 0 & 0 & c'_{95} \\
c_{61} & c_{62} & 0 & 0 & 0 & D_6 & \bullet & \bullet & \bullet \\
0 & c_{72} & c_{73} & 0 & 0 & \bullet & D_7 & \bullet & \bullet \\
0 & c_{82} & 0 & c_{84} & 0 & \bullet & \bullet & D_8 & \bullet \\
c_{91} & 0 & 0 & 0 & c_{95} & \bullet & \bullet & \bullet & D_9
\end{pmatrix}
\longrightarrow
\text{(b)} \begin{pmatrix}
D_5 & c'_{95} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
c_{95} & D_9 & 0 & 0 & 0 & 0 & 0 & 0 & c_{91} \\
0 & 0 & D_3 & c'_{73} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & c_{73} & D_7 & 0 & 0 & c_{72} & 0 & 0 \\
0 & 0 & 0 & 0 & D_4 & c'_{84} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & c_{84} & D_8 & c_{82} & 0 & 0 \\
0 & 0 & 0 & c'_{72} & 0 & c'_{82} & D_2 & c'_{62} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & c_{62} & D_6 & c_{61} \\
0 & c'_{91} & 0 & 0 & 0 & 0 & 0 & c'_{61} & D_1
\end{pmatrix}
$$

Fig. 3: Matrices for the mechanism in Figure 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 following the colored paths in matrix (b) in Figure 3 from top-left to bottom-right, one can clearly see how we are visiting the diagonal blocks from the leafs to the root D_1. In contrast, matrix (a) starts the factorization with node D_1 which is not a leaf. Note that generally there are multiple orderings that 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.

So far we have achieved zero fill-ins purely by reordering a matrix without modifying the LDU factorization. However, we know that for a loop-free structure each link has exactly one predecessor (parent). Therefore, for a structure with n links, we also have n constraints (or $n-1$ for free-floating systems). Because we are not creating fill-ins during the factorization, we can achieve $O(n)$ (linear) 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 factorization procedure from Algorithm 1 is modified in Algorithm 3.

Algorithm 3 Sparse In-Place LDU Factorization

```
1: for $i \in$ list do
2:     for $c \in$ children($i$) do            ▷ Similar to L and U factorization of Alg. 1
3:         $F_{i,c} \leftarrow F_{i,c}F_{c,c}^{-1}$
4:         $F_{c,i} \leftarrow F_{c,c}^{-1}F_{c,i}$
5:     for $c \in$ children($i$) do            ▷ Similar to D factorization of Alg. 1
6:         $F_{i,i} \leftarrow F_{i,i} - F_{i,c}F_{c,c}F_{c,i}$
```

After the factorization of system (38), we need to perform back-substitution to find the solution. We will again exploit the fact that we did not create fill-ins during the factorization and that we only have $O(n)$ nodes to modify Algorithm 2 to be of $O(n)$ complexity. This procedure is depicted in Algorithm 4.

Algorithm 4 Sparse LDU Back-Substitution

```
1: for $i \in$ list do
2:     $\Delta \boldsymbol{s}_i \leftarrow \boldsymbol{f}_i$
3:     for $c \in$ children($i$) do            ▷ Similar to L back-substitution of Alg. 2
4:         $\Delta \boldsymbol{s}_i \leftarrow \Delta \boldsymbol{s}_i - F_{i,c}\boldsymbol{s}_c$
5: for $i \in$ reverse(list) do            ▷ Similar to D and U back-substitution of Alg. 2
6:     $\Delta \boldsymbol{s}_i \leftarrow F_{i,i}^{-1}\Delta \boldsymbol{s}_i$
7:     $\Delta \boldsymbol{s}_i \leftarrow \Delta \boldsymbol{s}_i - F_{c,\text{par}(i)}\Delta \boldsymbol{s}_{\text{par}(i)}$            ▷ par($i$): parent($i$); if $i$ has parent
```

We perform factorization and back-substitution iteratively until our Newton method converges, which typically takes three to four iterations with a simple line search ensuring decreasing function value. 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 (30) and (31). Subsequently, we use $\boldsymbol{v}_2$, $\boldsymbol{\omega}_2$, and $\boldsymbol{\lambda}_2$ as new initial guesses 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 on their side. 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. If we only have 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 [9]. The code for our algorithm and all experiments is available on the Robotic Exploration Lab's GitHub page https://github.com/RoboticExplorationLab. We ran all experiments 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 and show typical convergence behavior in Figure 4.

For plot (a) we measured the energy of a double pendulum over 60 minutes with a time step $\Delta t = 0.01$s. We compare our first order variational integrator to the explicit second order Heun's method (a Runga Kutta method), using Julia's *DifferentialEquations* package [15]. One can clearly see how the error of the explicit method increases substantially over time, 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 implemented in Julia *RigidBodyDynamics.jl* [9]. 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 in the

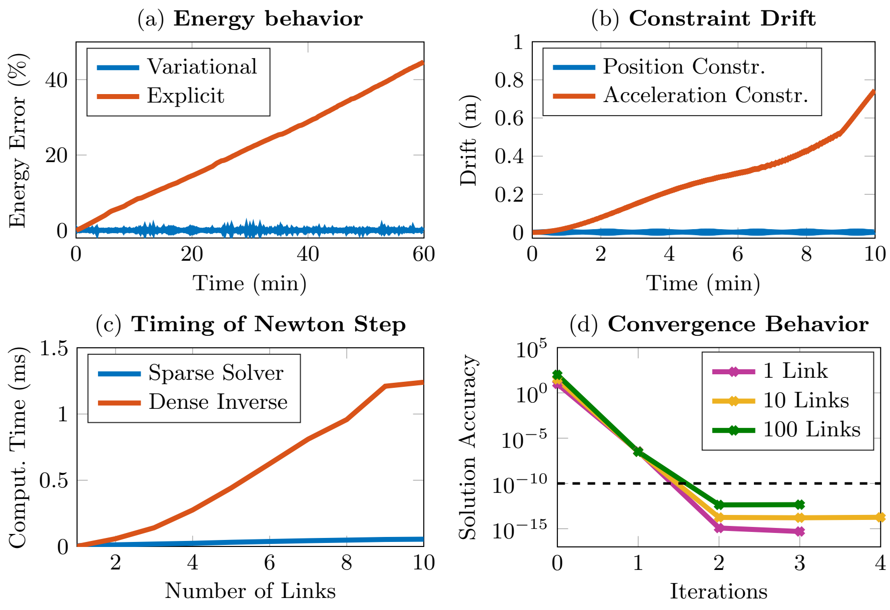

Fig. 4: Main algorithm properties. Our algorithm in blue, comparison in red. (a) Energy conservation with variational integrator compared to explicit integrator. (b) Drift with position constraints compared to drift with acceleration constraints. (c) Sparse $O(n)$ behavior compared to $O(n^3)$ dense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

Newton solver, which guarantees constraint satisfaction without requiring any additional constraint stabilization.

Plot (c) in Figure 4 compares the computation time of sparse and dense single Newton steps computed step 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 convergence behavior 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 an explicit Munthe-Kaas integrator, a second-order variational integrator using a quasi-Newton method [10], and a third-order variational integrator using an exact Newton method [5]. 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 with a different number of links for 10 seconds and a time step $\Delta t = 0.01$s. We show the computation time for revolute joints (1 degree of freedom) and ball-and-socket joints (3 degrees of freedom) in Figure 5. We also introduced a single loop closure and ran the same experimental setup for these closed-loop structures with results displayed in the same figure.

In both the open and closed-loop setting, the linear time behavior is clearly visible. While the explicit integrator performs better for revolute joints, especially with few degrees of freedom, we achieve competitive results and outperform this integrator for ball-and-socket joints. Note that we are faster at simulating ball-and-socket joints than revolute joints because a ball-and-socket joint constrains fewer degrees of freedom.

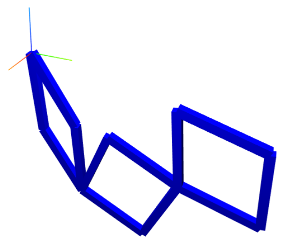
Fig. 6: Chain consiting of 3 4-link segments.

To highlight the robustness of our algorithm, we additionally simulated a chain of four-link closed-loop segments with revolute joints (see Figure 6). These segments have structurally singular configurations for overlapping links. 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 $\Delta t = 0.01$s are displayed in Figure 7. Because the minimal-coordinate integrator had trouble

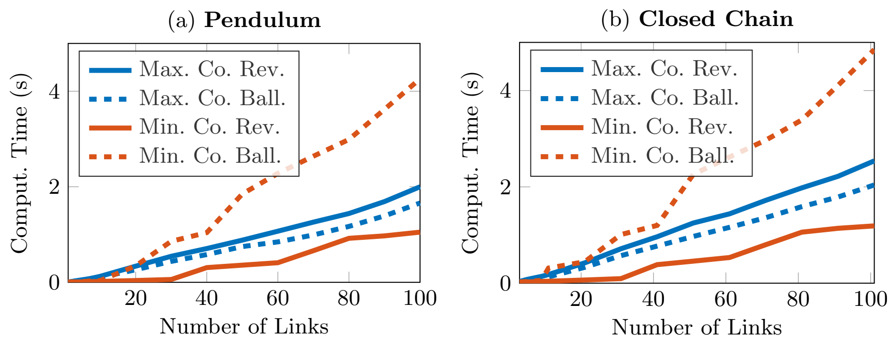

Fig. 5: Performance comparison simulating 1000 time steps of structures with different numbers of links. Our algorithm (maximal coordinates) in blue, explicit integrator (minimal coordinates) in red. Revolute joint dashed line, ball-and-socket joint solid line. (a) Pendulum. (b) Closed-loop mechanisms.

reliably simulating chains of two or more four-link segments, we cannot give timing results for this integrator. Using smaller time steps for a potentially more accurate simulation did not lead to a significant improvement.

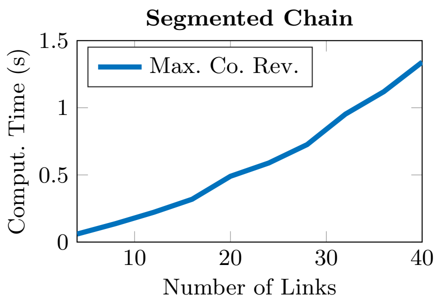

Fig. 7: Performance simulating 1000 time steps of different chains consisting of four-link segments (see Figure 6). Timing results for our algorithm.

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. We only simulated revolute joints and loop-free structures, 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 Figure 8.

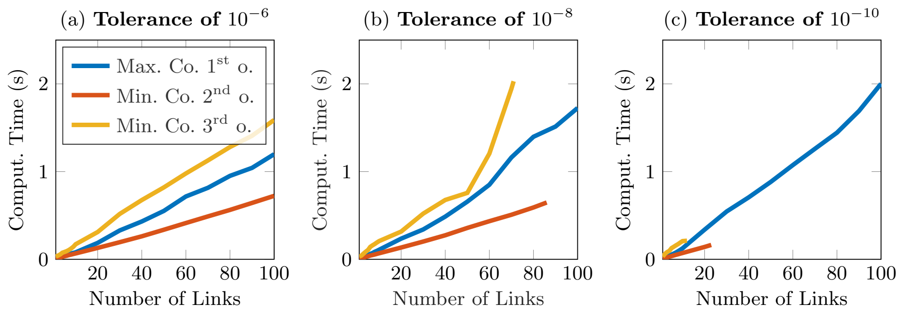

Fig. 8: Performance comparison simulating 1000 time steps of n-link pendulums with varying solution tolerances. Our (1^{st} order) algorithm in blue, 2^{nd} order variational integrator in red, 3^{rd} order variational integrator in yellow. (a) Tolerance of 10^{-6}. (b) Tolerance of 10^{-8}. (c) Tolerance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

While all variational integrators were able to simulate the pendulum successfully for a tolerance of 10^{-6}, a higher 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 robustness and scales well without any numerical adjustments. In terms of computational performance, we are competitive in being slightly slower than the first order integrator and slightly faster than the third order integrator.

6 Conclusion

We 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 variational integrator in maximal coordinates.

Overall, the results of our experiments show that we are competitive in terms of computational performance to minimal-coordinate integrators both explicit and variational. Especially for mechanical systems with a higher number of links and higher degree of freedom joints we show better performance for both open and closed-loop structures. We are also able to achieve higher accuracy in simulation than comparable variational integrators. For structures with multiple closed kinematic loops and frequently occurring singularities we perform robustly while a minimal-coordinate integrator fails to continuously find solutions.

In summary, these results make a strong case for the use of maximal-coordinate integrators, especially when dealing with more complex, close-loop structures.

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. I. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, 2017.
5. 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.
6. R. Featherstone. *Rigid Body Dynamics Algorithms.* Springer, 2008.
7. E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration.* Springer, 2006.
8. O. Junge, J. Marsden, and S. Ober-Blöbaum. Discrete Mechanics and Optimal Control. *IFAC Proceedings Volumes*, 38(1):538–543, 2005.
9. 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.
10. 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.
11. 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.
12. Z. Manchester and M. Peck. Quaternion Variational Integrators for Spacecraft Dynamics. *Journal of Guidance, Control, and Dynamics*, 39(1):69–76, 2016.
13. J. Marsden and M. West. Discrete Mechanics and Variational Integrators. *Acta Numerica*, 10:357–514, 2001.
14. S. Ober-Blöbaum and N. Saake. Construction and Analysis of Higher Order Galerkin Variational Integrators. *Advances in Computational Mathematics*, 41:955–986, 2015.
15. 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.
16. B. Siciliano and O. Khatib. *Springer Handbook of Robotics.* Springer, 2016.
17. J. Solà. Quaternion Kinematics for the Error-State Kalman Filter. *arXiv:1711.02508 [cs.RO]*, 2017.
18. T. Wenger, S. Ober-Blöbaum, and S. Leyendecker. Constrained Galerkin Variational Integrators and Modified Constrained Symplectic Runge-Kutta Methods. page 320013, 2017.
19. 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.