Scalable Variational Integrators for Constrained Mechanical Systems in Generalized Coordinates

Elliot Johnson, *Student Member, IEEE,* Todd Murphey, *Member, IEEE,*

***Abstract*—We use a tree-based structure to represent mechanical systems comprising interconnected rigid bodies. The tree description provides a way to design a generic, variational integrator that works for arbitrary systems while staying in generalized coordinates. Variational integrators have the advantage of preserving constraints and energetic quantities for all time. Moreover, by taking advantage of caching, performance scales very well. A description of the associated software is included, along with a closed-kinematic-chain example illustrating the ease of specifying systems. Several other examples, including a comparison with the Open Dynamics Engine (ODE) software, are used to illustrate the scalability of the technique.**

I. INTRODUCTION

There are many different algorithms for simulating mechanical systems. The most popular methods in simulation software are those based on the Newton-Euler force balance approach[33]. Systems are usually represented as collections of free bodies with mechanical structure imposed by constraints. This is the approach used in the popular Open Dynamics Engine (ODE) [28]. Others like OpenHRP [2] use the Newton-Euler method but represent the system in generalized coordinates [9]. Some packages, like Autolev [26] generate the full equations of motion for a system instead of algorithmically evaluating the dynamics. The different methods have tradeoffs in computational complexity, accuracy, and ease-of-representation. In this paper we present another algorithm that results in scalable simulations in generalized coordinates that use variational integrators to guarantee desirable energy, momentum, and constraint-satisfying behavior.

The constrained free-body approach to dynamics is the industry standard in computer graphics, video games, and CAD software. The algorithms are fast and scalable, typically using special implementation techniques like the LCP formulation [3] and sparse matrix methods [4]. They are flexible because forces are explicitly added to bodies at each time step, making it straightforward to include anything from friction to springs to motors. The theory is also accessible because it is based directly on intuitive force balance methods.

Simulations based on generalized coordinates, on the other hand, are almost always preferred in controls analysis. We want to think of a pendulum as an angle, not a body in $\mathrm{SE}(3)$ constrained in 5 degrees of freedom. This is important for analyzing important system properties like stability and controllability. However, models in generalized coordinates are either derived manually (which is irritating, error-prone, and not scalable) or use algebraic software to automatically derive symbolic equations of motion (which is less irritating, less error-prone, only slightly more scalable). Nonetheless, when a system is simple enough to work with generalized coordinates, the benefits are usually worth the extra effort and slower performance.

Equations of motion in generalized coordinates scale poorly because they repeat a great deal of information. For example, consider a simulation of a simplified human body, an example we will consider shortly. The equations for the femur, knee, shank, ankle, foot, and toes all contain their own copy of the hip kinematics. The repetition reflects the dependencies that arise in generalized coordinates. Scalability issues are compounded by taking derivatives, where repeated application of the product rule causes the equations to grow quickly. The constrained free-body approach avoids this repetition because each body is represented independently. The structural relationships are enforced by constraint forces instead.

Featherstone [11] showed that using a tree[1] representation and recursive equations for a system eliminates much of the repetition and explicitly reuses information. In this setting there are $O(n)$ algorithms (where n is the number of rigid bodies in the system) available for simulating the (continuous) forward dynamics by using these representations.

We advocate the same ideas but use a different tree representation based on homogeneous coordinate transformations. While our method does not yield $O(n)$ dynamics, the resulting algorithm is concise, transparent, and extendable. No modifications are required to handle branching, closed kinematic chains, or holonomic constraints. Other tree-based methods[34] also handle closed chains, but modify the dynamics to use iterative algorithms that require solving the inverse kinematics. Additionally, this method can be extended to obtain the derivatives of the dynamics (i.e. linearizations) that are needed for many optimal control techniques [14]. We do not, however, discuss derivatives in this paper beyond those needed for dynamic simulation.

Another benefit, which we focus on in this paper, is that this approach allows us to develop variational integrators for arbitrary systems in generalized coordinates. Variational integrators are a class of symplectic integrators [13] that have been developed by the discrete mechanics community. They preserve (or nearly preserve, depending on the particular integrator) fundamental properties of mechanical systems like conservation of energy, conservation of momentum, and the symplectic form [20]. They also deal with holonomic constraints and non-smooth phenomenon like collisions well.

These properties are often *not* preserved in force-balance simulations (or even continuous Euler-Lagrange mechanics).

[1]More generally, we may represent a system as a graph with tree descriptions as the subset of directed acyclic graphs [7]. However, the applications presented in this paper do not derive any immediate benefits to considering trees from a graph theory perspective, so we do not advocate this perspective here.

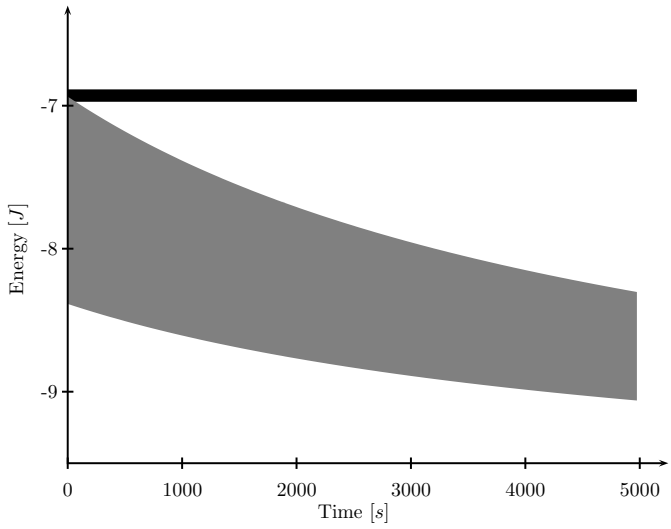


Fig. 1: This plot shows the total energy of a single link planar pendulum simulated for 5000 seconds with a time step of $0.01s$. The trajectories appear as regions rather than lines because the variations change rapidly compared to the long time scale. The variational integrator simulation is shown in black. The `ODE` simulation is shown in gray.

For example, Fig. 1 compares the total energy of a single pendulum simulated using a variational integrator and ODE. `ODE` uses a force-balance method to compute the continuous dynamics of a system. The variational integrator does not perfectly conserve energy at each time step, but maintains the correct average energy. The `ODE` solution slowly dissipates energy over time, slowly leading to an implausible simulation.

While energy conservation does not guarantee an accurate solution, it is still desirable that a simulation respects known analytical properties of a system like conservation of energy and momentum or naturally maintaining constraints. When these properties are not satisfied, the resulting trajectory is certainly incorrect, and this is the sense in which we consider a simulation to be implausible.

Variational integrators also handle holonomic constraints (expressed as $h(q) = 0$ for valid configurations q) better than continuous methods (Force-balance or Euler-Lagrange). Holonomic constraints restrict the relative positions and orientations between bodies. A pin joint is an example of a holonomic constraint. (Non-holonomic constraints, on the other hand, restrict relative velocities between bodies rather than relative positions.) In continuous dynamics, holonomic constraints generally cannot be implemented directly. Instead, they are replaced with equivalent higher-order constraints. The standard approach is to ensure the simulation's initial conditions satisfy the constraints and then require everything to accelerate in a consistent way that (ideally) keeps the constraints satisfied:

$$
\begin{aligned}
& h(q(t)) = 0 \ \forall \ t \\
& \qquad h(q(t_0)) = 0 \\
\Rightarrow \quad & \tfrac{\partial h}{\partial q}(q(t_0))\dot{q}(t_0) = 0 \\
& \tfrac{\partial h}{\partial q}(q(t))\ddot{q}(t) + \tfrac{\partial^2 h}{\partial q \partial q}(q(t)) \cdot (\dot{q}(t), \dot{q}(t)) = 0 \ \forall \ t
\end{aligned}
$$

Simulations using this approach tend to violate the constraints as error is introduced through numeric integration.

To fix this, restoring forces resembling damped springs are added to each constraint. This adds "magic" parameters (e.g. the Error Reduction Parameter in `ODE`) that must be tuned for each simulation (often at the risk of introducing unstable dynamics for bad choices of parameters) and introduces artificial energy loss. For highly constrained systems, the losses can dominate the resulting dynamics (See the scissor lift example in Sec. VI-C). Variational integrators work with holonomic constraints directly, avoiding these problems completely.

We begin with a brief overview of homogeneous representations for rigid body transformations in Sec. II. In Sec. III we introduce our tree representations for mechanical systems. Section III-B and Sec. III-C show how to compute positions, velocities, and their derivatives from the tree. Discrete mechanics are introduced in Sec. IV and then used to implement a variational integrator based on the tree descriptions in Sec. IV-A. Section IV-B briefly describes how continuous dynamics can also be derived in this framework. Section IV-C demonstrates how potential energies are calculated for the system. Section IV-D and Sec. IV-E extend the variational integrator to include constraints and non-conservative forcing. Section V introduces a software package called `trep` (available at *http://robotics.colorado.edu/trep/*) that implements the presented methods. Finally, Sec. VI presents examples that demonstrate how these methods scale and compare to other simulations.

II. Homogeneous Representation

The tree description makes extensive use of homogeneous representations for coordinate frames and rigid body transformations, so a brief overview is helpful. For an in-depth discussion on homogeneous representations, see [21].

Homogeneous representations provide a convenient and uniform way to represent rigid body transformations. A transformation is represented as a matrix in $\mathbb{R}^{4\times 4}$ comprising a rotation matrix R and the translational components p:

$$
g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \qquad R \in SO(3),\, p \in \mathbb{R}^3 \tag{1}
$$

In this setting rigid body transformations can also be thought of as defining a coordinate frame relative to another coordinate frame. The p vector defines the position of the frame's origin and R defines the orientation of the coordinate axes.

Homogeneous transformations are composed using standard matrix multiplication. If g_{ab} defines the coordinate transformation from coordinate frame b to coordinate frame a, and g_{bc} defines the transformation from frame c to frame a, the transformation from frame c to frame a is

$$
g_{ac} = g_{ab} \cdot g_{bc}
$$

Points and vectors are represented as elements in $\mathbb{R}^4$. Points are extended from $\mathbb{R}^3$ with an extra 1 component and vectors are extended with a 0:

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad v = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

Points and vectors are also transformed with standard matrix multiplication.

$$p_a = g_{ab} \cdot p_b \qquad v_a = g_{ab} \cdot v_b$$

We will often make use of a Lie algebra isomorphism [9] casually called the "hat" operator and its inverse, the "unhat" operator. These simply change the way we represent special quantities that show up when working in homogeneous coordinates, namely body velocities. The "hat" operator takes a $\mathbb{R}^6$ vector to a $\mathbb{R}^{4x4}$ matrix:

$$\begin{bmatrix} x \\ y \\ z \\ a \\ b \\ c \end{bmatrix}^{\wedge} \mapsto \begin{bmatrix} 0 & -c & b & x \\ c & 0 & -a & y \\ -b & a & 0 & z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The "unhat" operator $(\cdot)^{\vee}$ simply inverts this operation. Often we write the hat directly above the variable (e.g. $v^{\wedge} = \hat{v}$). For more detail, see [21].

Homogeneous representations are useful because they are generic; the same form is used for translations, rotations, and compositions thereof. We use this to our advantage by having a single way to represent the various coordinate frames that describe a mechanical system. The resulting simplicity is a major advantage of tree descriptions of mechanical systems.

III. Tree Representations

The main idea for a tree representation of a mechanical system is to attach coordinate frames throughout the system and relate them to one another with simple rigid body transformations. We organize the coordinate frames into a tree so that each frame has a single parent and zero or more children. The root of the tree is the stationary world frame s. Frames are defined in space by rigid body transformations from their parent frame.

In this work, we use only six basic transformations, represented as elements of $\mathrm{SE}(3)$: translations along and rotations about the parent's X, Y, and Z. If a frame is fixed relative to the parent, the transformation is parameterized by a constant. If the frame can move, it is parameterized by a unique configuration variable. The set of all configuration variables forms the generalized coordinates for the system.

Fig. 2 is an example tree description for a planar human. Note that coordinate frames do not necessarily have to have an associated mass so we can define as many frames as needed for the particular application. Consequently, a tree representation is not unique. There are an infinite number of representations for a given system.

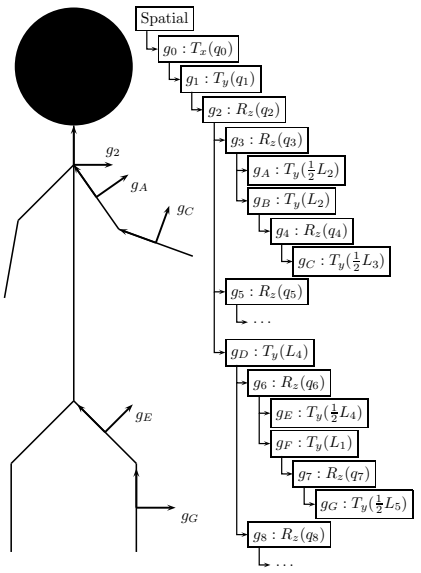


Fig. 2: A planar human is represented with a tree structure. Note that although we are restricted to simple transformations, we can represent complex mechanical systems by composing multiple transformations.

Fig. 2 also establishes the naming convention used throughout the paper. Transformations that are parameterized by constants are indexed with letters (e.g., $g_A(1.0)$). Variable transformations are indexed by numbers and are parameterized by the corresponding configuration variable (e.g., $g_1(q_1)$).

We impose the following requirements on the tree description:

- R1. Frames are related to their parents through six basic transformations: Translations along the parent's X, Y, and Z axes and rotations about the parent's X, Y, and Z axes.
- R2. Each configuration variable is used in only one transformation and each transformation only depends on one parameter or configuration variable.

Together, these two requirements establish a one-to-one mapping between configuration variables and variable transformations. They are not absolutely necessary for this approach, but they significantly simplify the discussion, notation, and equations without significantly restricting the systems that can be represented.

The basic transformations defined by R1 are parameterized by single real-valued numbers: configuration variables for moving joints and constant numbers for fixed joints. This requirement leads to simplifications and leads to uniform generalized coordinates consisting only of real-valued numbers.

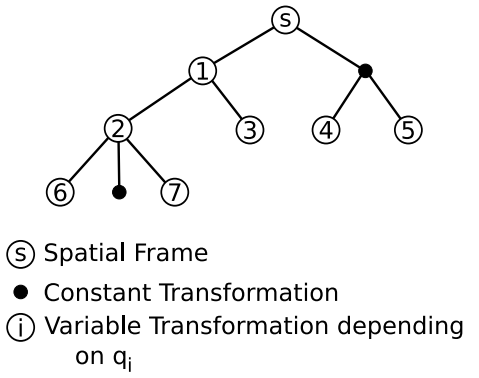
The practical consequence of R1 is disallowing direct SO(3) parameterizations for free rotations. Instead, we represent free rotation with multiple single-axis rotations (e.g. Euler angles). This requirement can be removed if one is willing to handle

the extra book-keeping involved.[2]

R2 prevents using the same configuration variable to drive multiple transformations, which is useful for modeling systems with parallel linkages. Parallel linkages and closed chains are instead handled by connecting open chains with holonomic constraints.

Table I defines the notation used throughout the paper. We will typically drop the explicit dependence on q and q_k for visual clarity (e.g. $g_{s,k}$ rather than $g_{s,k}(q)$).

TABLE I: Notation used for the tree representation.



$q_i(t) \in \mathbb{R}$	Configuration variable of the i-th frame.
$\dot{q}_i(t) \in \mathbb{R} = \frac{d}{dt} q_i(t)$	Time derivative of the i-th configuration variable.
$q(t) \in \mathbb{R}^n$	Configuration vector comprising q_0, q_1, ... , q_n.
m_i	Mass of the i-th frame.
$M_i \in \mathbb{R}^{6x6}$	Inertia Tensor of the i-th mass in body-fixed coordinates.
$g_i(q_i) \in \mathrm{SE}(3)$	Transformation of the i-th frame to its parent frame.
$g_{s,i}(q) \in \mathrm{SE}(3)$	Transformation from the i-th frame to the spatial reference frame s.
$g_{j,i}(q) \in \mathrm{SE}(3)$	Transformation the i-th frame to the j-th frame.
$v^b_{s,i} \in T_e SE(3)$	Body velocity (i.e., an element of the Lie algebra) of the i-th frame relative to the spatial reference frame s.
$p_{s,i}(q) \in \mathbb{R}^3$	Position of the i-th frame relative to the spatial reference frame. The position is obtained by extracting the upper right components in (1) from $g_{s,i}$.
$\mathrm{anc}(i)$	The ancestors of the the i-th frame are the frames passed while moving up the tree from the i-th frame to the spatial frame. For example, frame 6 (above) has ancestors $\{2, 1, s\}$.
$\mathrm{par}(i)$	Immediate parent frame of the i-th frame. For example, the parent of frame 6 (above) is frame 2.
$g'_k = \frac{\partial}{\partial q_k} g_k$	Notation for the derivative of a transformation of the i-th frame to its parent frame.
$\dot{g}_k = \frac{\partial}{\partial t} g_k$	Notation for the time derivative of a transformation of the i-th frame to its parent frame.

A tree representation of a mechanical system provides a clean way to organize and describe the system's structure. From this description, we can find positions, velocities, and derivatives of the coordinate frames in the system. These provide a foundation for doing operations like dynamic simulation, trajectory exploration, and analysis.

The position, velocities and their derivatives are found in the following sections as recursive, piecewise equations. The piecewise equations in this paper have a specific ordering. For cases that can be simultaneously satisfied, the top-most case always takes precedence.

A. Lower and Higher Pairs

The six basic transformations can be combined to create the canonical lower and higher pair joints [30]. The six lower pairs are revolute joints, prismatic joints, cylindrical joints, helical joints, spherical joints, and plane pairs. A revolute joint is simply one of the three rotation transformations. A prismatic joint is one of the three translation transformations. A cylindrical joint is a rotation transformation followed by a translation along the same axis (e.g. $R_x(q_1)T_x(q_2)$). A helical joint is modeled with a cylindrical joint with an added constraint (see Sec. IV-D5). A spherical joint is modeled with three successive rotations about different axes (e.g. $R_z(q_1)R_y(q_2)R_x(q_3)$). Finally, a plane pair is modeled with two translational transformations followed by a rotational transformation (e.g. $T_x(q_1)T_y(q_2)R_z(q_3)$).

The two common higher pairs are gear and cam pairs. A simple gear pair can be modeled with two rotations and a constraint on the configuration variables. A simple cam pair can be modeled with a rotation and a translation followed by a rotation along with a constraint.

More complicated mechanisms are modeled using the same approach. The system is designed with enough transformations to get the necessary degrees of freedom, and then constraints are added to create the proper kinematic relationships.

We could also model joints by adding new types of transformations. In particular, it is possible to add a transformation where the user specifies a twist that generates a parameterized element of $\mathrm{SE}(3)$[21]. This would allow direct modeling of screw joints for example. The necessary derivatives could still be automatically calculated and the rest of the algorithm would remain unchanged. We have instead chosen to limit ourselves to a basic set in order to facilitate automation and a simple implementation.

B. Frame Positions

The rigid body transformation from any frame to the spatial reference frame is a straightforward calculation, given the tree representation. For the spatial reference frame, it is the identity transformation, I. Otherwise, for frame k, it is the parent frame's transformation $g_{s,\mathrm{par}(k)}$ composed with the local transformation g_k. This is expressed as a piecewise expression for $g_{s,k}(q)$:

$$g_{s,k}(q) = \begin{cases} I & k = s \\ g_{s,\mathrm{par}(k)} g_k & k \neq s \end{cases} \tag{2}$$

[2]We emphasize that both requirements can be removed and more general, yet more complicated, equations derived. The expanded equations are not particularly complex themselves, but they require more specific notation and extra bookkeeping, so they are avoided here for clarity. Using SO(3) parameterizations, for example, requires adding constraints on the nine matrix elements to ensure the SO(3) matrix remains orthogonal [21].

Eq. (2) evaluates the transformation from any frame to the spatial frame using only the local transformations and itself recursively. The recursion is guaranteed to terminate on the $k = s$ condition because the tree is by definition acyclic. Each recursive call gets closer to the root node. The same is true for the remaining recursive equations that are derived from the tree.

The compact form of (2) is useful on its own, but it is particularly nice because we can find derivatives and end up with similarly simple equations. The derivative of (2) with respect to a configuration variable q_i is found with the standard derivative:

$$\begin{aligned}\tfrac{\partial}{\partial q_i} g_{s,k}(q) &= \begin{cases} \frac{\partial}{\partial q_i} I & k = s \\ \frac{\partial}{\partial q_i}\left(g_{s,\mathrm{par}(k)} g_k\right) & k \neq s \end{cases} \\ &= \begin{cases} 0 & k = s \\ \frac{\partial}{\partial q_i}\left(g_{s,\mathrm{par}(k)}\right) g_k & \\ \quad + g_{s,\mathrm{par}(k)} \frac{\partial}{\partial q_i} g_k & k \neq s \end{cases}\end{aligned}$$

The two requirements R1 and R2 help to further simplify this expression. R1 guarantees $\frac{\partial}{\partial q_i} g_k = 0$ when $i \neq k$. R2 implies $\frac{\partial}{\partial q_i} g_{s,\mathrm{par}(i)} = 0$ since $g_i(q)$ is the only transformation that depends on q_i. Furthermore, we know that if g_i is not an ancestor of g_k, the derivative is always zero:

$$\tfrac{\partial}{\partial q_i} g_{s,k}(q) = \begin{cases} 0 & k = s \\ & i \notin \mathrm{anc}(k) \\ \frac{\partial}{\partial q_i}\left(g_{s,\mathrm{par}(k)}\right) g_k & i \neq k \\ g_{s,\mathrm{par}(k)} g_k' & i = k \end{cases} \tag{3}$$

where $g_k' = \frac{\partial}{\partial q_k} g_k(q_k)$. Using (2) and (3), we can numerically compute the *exact* derivative of any coordinate frame with respect to any configuration variable in the system. Note that the derivative expression remains relatively simple and compact. The mixed partial derivative with respect to q_i and q_j is calculated using the same procedure.

$$\tfrac{\partial^2}{\partial q_j \partial q_i} g_{s,k}(q) = \begin{cases} & k = s \\ 0 & i \notin \mathrm{anc}(k) \\ & j \notin \mathrm{anc}(k) \\ g_{s,\mathrm{par}(k)} g_k'' & i = k = j \\ \frac{\partial}{\partial q_j}[g_{s,\mathrm{par}(k)}] g_k' & i = k \neq j \\ \frac{\partial}{\partial q_i}[g_{s,\mathrm{par}(k)}] g_k' & i \neq k = j \\ \frac{\partial^2}{\partial q_j \partial q_i}[g_{s,\mathrm{par}(k)}] g_k & i \neq k \neq j \end{cases} \tag{4}$$

Eq. (4) uses itself recursively along with (2) and (3) to evaluate second derivatives of the rigid body transformations. For the applications discussed in this paper, these are the only derivatives needed. However, we emphasize that this procedure can be continued to get higher derivatives as needed. One simply takes normal derivative and uses R1 and R2 to separate and simplify the various cases.

C. Frame Body Velocities

The body velocity is the relative motion of a coordinate frame with respect to the stationary world frame, but expressed in the frame's local coordinates. It is calculated as the velocity of the parent frame, transformed[3] into local coordinates, plus the velocity of the frame with respect to the parent, $g_k^{-1}\dot{g}_k$:

$$\hat{v}^b_{s,k}(q,\dot{q}) = \begin{cases} 0 & k = s \\ g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\dot{g}_k & k \neq s \end{cases}$$

where $\dot{g}_k = \frac{\partial}{\partial t} g_k$. For the transformations defined in R1, the local velocity term $g_k^{-1}\dot{g}_k$ reduces to a special form using a twist $\hat{\xi}$ [21]. The twist is constant (i.e. it does not depend on the configuration) for each type of transformation:

$$\hat{v}^b_{s,k}(q,\dot{q}) = \begin{cases} 0 & k = s \\ g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k + \hat{\xi}_k \dot{q}_k & k \neq s \end{cases} \tag{5}$$

Taking the same approach used earlier, we find derivative expressions.

$\frac{\partial}{\partial q_i}\hat{v}^b_{s,k}(q,\dot{q}) =$

$$\begin{cases} 0 & k = s \\ & i \notin \mathrm{anc}(k) \\ g_k^{-1\prime}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & i = k \\ g_k^{-1}\frac{\partial}{\partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & i \neq k \end{cases} \tag{6}$$

Eq. (6) is evaluated recursively with itself, the local rigid body transformations, and (5).

$\frac{\partial^2}{\partial q_j \partial q_i}\hat{v}^b_{s,k}(q,\dot{q}) =$

$$\begin{cases} & k = s \\ 0 & i \notin \mathrm{anc}(k) \\ & j \notin \mathrm{anc}(k) \\ g_k^{-1\prime\prime}\hat{v}^b_{s,\mathrm{par}(k)} g_k + 2 g_k^{-1\prime}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & \\ \quad + g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k'' & i = k = j \\ g_k^{-1\prime}\frac{\partial}{\partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\frac{\partial}{\partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & i \neq k = j \\ g_k^{-1\prime}\frac{\partial}{\partial q_j}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\frac{\partial}{\partial q_j}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & i = k \neq j \\ g_k^{-1}\frac{\partial^2}{\partial q_j \partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & i \neq k \neq j \end{cases} \tag{7}$$

Eq. (7) is evaluated using itself, local rigid body transformations, (5), and (6). We are also interested in derivatives of body velocities with respect to the configuration variable time derivatives.

$$\tfrac{\partial}{\partial \dot{q}_i}\hat{v}^b_{s,k}(q,\dot{q}) = \begin{cases} 0 & k = s, \\ & i \notin \mathrm{anc}(k) \\ \hat{\xi}_k & i = k \\ g_k^{-1}\frac{\partial}{\partial \dot{q}_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & i \neq k \end{cases} \tag{8}$$

$$\tfrac{\partial^2}{\partial \dot{q}_j \partial \dot{q}_i}\hat{v}^b_{s,k}(q,\dot{q}) = 0 \tag{9}$$

We can also find mixed partial derivatives with respect to configuration variables q_i and their time derivatives $\dot{q}_i$.

[3]Note that the similarity transform $g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k$ could be replaced by an "Adjoint" $Ad_{g_k} v^b_{par(k)}$ transformation [21]. Indeed, all the following equations can be modified to use their intrinsic counterparts. However, we focus here on as transparent an approach as possible and do not use any differential geometric formality.

$\frac{\partial^2}{\partial \dot{q}_j \partial q_i} \hat{v}^b_{s,k}(q,\dot{q}) =$

$$\begin{cases} 0 & \begin{array}{l} k = s \\ k = j, \\ i \notin \mathrm{anc}(k) \\ j \notin \mathrm{anc}(k) \end{array} \\ g_k^{-1\prime} \frac{\partial}{\partial \dot{q}_j} \hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1} \frac{\partial}{\partial \dot{q}_j} \hat{v}^b_{s,\mathrm{par}(k)} g_k' & k = i \\ g_k^{-1} \frac{\partial^2}{\partial \dot{q}_j \partial q_i} \hat{v}^b_{s,\mathrm{par}(k)} g_k & k \neq i \end{cases} \quad (10)$$

Eq. (2) through (10) demonstrate how we calculate the forward kinematics and derivatives from a tree description. As will be discussed later, these equations provide all the necessary values for simulating the system dynamics. Other applications, such as trajectory exploration in optimal control or nonlinear controllability analysis, may require higher derivatives. That these derivatives can be found by continuing this procedure (and that they remain simple themselves) is a major advantage of this method.

D. Primitive Transformations

Eq. (2) through (10) include terms that we have not explicitly shown how to calculate (i.e. g_k, g_k', g_k'', g_k^{-1}, $g_k^{-1\prime}$, $g_k^{-1\prime\prime}$, and ξ_k). These are found manually for each of the primitive transforms (defined in R1). For example, the values for a rotation about the Z axis are:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z'(\theta) = \begin{bmatrix} -\sin\theta & -\cos\theta & 0 & 0 \\ \cos\theta & -\sin\theta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R_z''(\theta) = \begin{bmatrix} -\cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & -\cos\theta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Higher derivatives for the six transformations in R1 can easily be parameterized. The derivatives of translations are zero for the second derivatives and higher. The derivatives of rotations are cyclic (e.g. $R^{(4)} = R$). We can therefore find the expression for the n^{th} derivative arbitrarily.

For the six transformations given in R1, the inverse is always the same transformation but by the opposite amount ($g^{-1}(x) = g(-x)$):

$$\begin{aligned} R_x^{-1}(\theta) &= R_x(-\theta) \\ R_x^{-1\prime}(\theta) &= -R_x'(-\theta) \\ R_x^{-1\prime\prime}(\theta) &= R_x''(-\theta) \end{aligned}$$

The twist ξ_{R_z} is a bit more work.

$$\begin{aligned} \hat{\xi}_{R_z}\dot{\theta} &= R_z^{-1}(\theta)\dot{R}_z(\theta) \\ &= R_z^{-1}(\theta)R_z'(\theta)\dot{\theta} \\ \hat{\xi}_{R_z} &= R_z^{-1}(\theta)R_z'(\theta) \\ &= \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \Rightarrow \xi_{R_z} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T \end{aligned}$$

Later equations also make use of the position of a coordinate frame's origin relative to the spatial world frame. The position $p_{s,k}$ of a frame is trivially obtained by extracting the upper right components from the corresponding $g_{s,k}$ transformation in (2). Derivatives of $p_{s,k}$ are similarly extracted from the corresponding derivative of $g_{s,k}$ in (3) and (4).

E. Performance

There are two notes on the performance of evaluating the above equations. Values are frequently reused in these calculations. For example, $g_{s,k}(q)$ may be evaluated once for itself, once for each of its descendants' positions, and then again for derivative values. However, once it is evaluated, it is constant until a new configuration is written to the tree. We can therefore save the first result and reuse it until a new configuration is written. This avoids recursing all the way to the base of the tree in every calculation, essentially flattening (2) through (10). This technique is called caching and significantly improves performance.

The second and more technical note is that (2) through (10) involve only 4x4 matrix operations (multiplication and addition). Modern computers typically have special hardware support (SIMD instructions, graphics accelerators) for the same 4×4 matrix operations because of their importance in computer graphics and multimedia applications. Current implementations are typically limited to 32- or 64-bit precision, but future hardware is likely to support full 80-bit precision that can be used to greatly improve performance without sacrificing accuracy or modifying the tree description and algorithms.

With the caching optimization in particular, the tree equations scale to large systems well enough that we can use a tree description as a basis for fast and accurate dynamic simulation.

IV. Discrete Mechanics

It is useful to consider continuous Lagrangian mechanics before introducing discrete mechanics and variational integrators. Lagrangian mechanics use Hamilton's Least Action Principle to derive the equations of motion for a system from an abstract quantity called the Lagrangian. The Lagrangian $L(q,\dot{q})$ is defined as the system's kinetic energy minus its potential energy.

$$L(q,\dot{q}) = \sum_k T_k(q,\dot{q}) - \sum_k V_k(q)$$

where q is the state configuration vector and $\dot{q}$ is its time derivative. For a tree description, q is the vector of all variables used to parameterize coordinate transformations. Common potentials include gravity and springs. Examples of both are discussed later.

The kinetic energy, $T_k(q, \dot{q})$, takes on a particularly nice form if we define a coordinate frame at each center of mass and align the axes with the body's principal axes[8][24]. In this case, the inertia matrix is a constant 6x6 diagonal matrix and the kinetic energy is written as $T_k(q, \dot{q}) = \frac{1}{2} v_{s,k}^{bT} M_k v_{s,k}^b$ (recalling that v^b is the body velocity) where the inertia matrix is

$$M_k = \begin{bmatrix} m_k & 0 & 0 & 0 & 0 & 0 \\ 0 & m_k & 0 & 0 & 0 & 0 \\ 0 & 0 & m_k & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx,k} & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{yy,k} & 0 \\ 0 & 0 & 0 & 0 & 0 & I_{zz,k} \end{bmatrix}$$

The resulting Lagrangian is:

$$L(q, \dot{q}) = \sum_k \tfrac{1}{2} v_{s,k}^{bT} M_k v_{s,k}^b - \sum_k V_k(q) \tag{11}$$

Note that given $V_k(q)$, (11) can be numerically evaluated from a tree description using (5). We continue to assume that $V_k(q)$ and derivatives are known. Their actual computation is discussed in Sec. IV-C. The Lagrangian (11) has an associated quantity called the Action which is the integral of the Lagrangian along a trajectory.

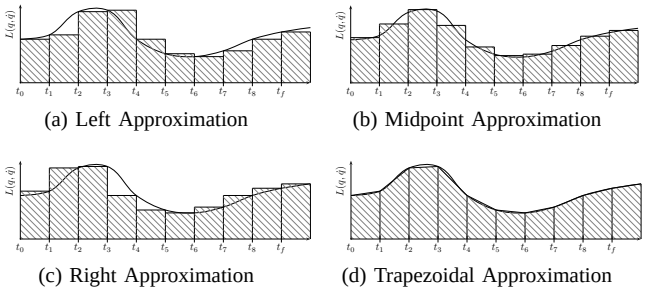
$$S(q([t_0, t_f])) = \int_{t_0}^{t_f} L(q(\tau), \dot{q}(\tau)) \mathrm{d}\tau \tag{12}$$

The Least Action principle states that the system will naturally follow the trajectory that minimizes[4] the action. A variational principle is used to minimize (12) to get the Euler-Lagrange equations [19]:

$$\frac{\partial}{\partial t} \frac{\partial L}{\partial \dot{q}}(q, \dot{q}) - \frac{\partial L}{\partial q}(q, \dot{q}) = 0 \tag{13}$$

This is a second order differential equation that can be integrated to simulate the system and obtain a trajectory $q(t)$ from a set of initial conditions $q(t_0)$ and $\dot{q}(t_0)$. However, numeric integration introduces error in the simulated trajectory. Since the dynamics are treated as generic ordinary differential equations, the error is introduced in ways that do not preserve important mechanical properties like conservation of energy and momentum. Additional features like constraints also perform poorly and exhibit unrealistic behavior like objects 'sinking' into hard surfaces [27].

Recent research has found that explicitly including the discrete approximations at a more fundamental level of the dynamics derivation leads to integration schemes that respect the fundamental symmetries in dynamics.[5] This approach is called discrete mechanics and the resulting integrators are known as Variational Integrators [16]. Variational integrators conserve (or nearly conserve, depending on the particular integrator) fundamental quantities like momentum and energy [17]. They are also well suited for problems involving holonomic constraints, impacts, and non-smooth phenomenon [12].



(a) Left Approximation (b) Midpoint Approximation

(c) Right Approximation (d) Trapezoidal Approximation

Fig. 3: The discrete Lagrangian approximates segments of the continuous action integral. The area of each shaded region represents a value of the discrete Lagrangian.

In discrete mechanics, we find a sequence $\{(t_0, q_0), (t_1, q_1), \ldots, (t_n, q_n)\}$ that approximates the continuous trajectory of a mechanical system ($q_k \approx q(t_k)$). We assume a constant time step ($t_{k+1} - t_k = \Delta t \ \forall \ k$) for simplicity, but in general the time step can be varied to use adaptive time stepping algorithms. For example, [20] describes a method that adapts the time step to maintain perfect energy conservation.

To derive a variational integrator, we define a discrete Lagrangian that approximates the action integral over a short interval.

$$L_d(q_k, q_{k+1}) \approx \int_{t_k}^{t_{k+1}} L(q(\tau), \dot{q}(\tau)) \mathrm{d}\tau$$

Figure 3 shows several choices of approximations to determine the discrete Lagrangian. The order of accuracy for the approximation is directly related to the order of accuracy for the resulting trajectory [20].

The discrete Lagrangian replaces the system's action integral with an action sum.

$$S(q([t_0, t_f])) = \int_{t_0}^{t_f} L(q(\tau), \dot{q}(\tau)) \mathrm{d}\tau \approx \sum_{k=0}^{n-1} L_d(q_k, q_{k+1}) \tag{14}$$

The action sum is minimized with a variational principle to get an implicit difference equation, analogous to the Euler-Lagrange equations in (13), called the Discrete Euler-Lagrange (DEL) equation.

$$D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k) = 0 \tag{15}$$

where $D_n f(\ldots)$ is the derivative of $f(\ldots)$ with respect to its n-th argument. This is known as the *slot derivative*. Note that the derivation of (15) is analogous to the approach used to derive the continuous dynamics equation (13).

Whereas the continuous Euler-Lagrange equation is an ODE that is integrated to find the trajectory of the system,

[4]To be rigorous, the Least Action Principle states that the action should be extremized, not minimized. While in practice it is almost always minimized (hence the name Least Action Principle), the distinction should be remembered.

[5]There are also specially designed numeric integrators for continuous dynamics that preserve the same properties (e.g. the Newmark scheme). It has been shown that these special integrators can often be derived from a variational integrator by choosing a particular discrete Lagrangian [31].

the discrete Euler-Lagrange equation (15) presents a root-finding problem to get the next configuration. Given two initial configurations q_0 and q_1, we solve

$$f(q_{k+1}) = D_1 L_d\left(q_k, q_{k+1}\right) + D_2 L_d\left(q_{k-1}, q_k\right) = 0 \quad (16)$$

to determine q_2. We then move forward and iterate to find $q_3, q_4, \ldots, q_N$. The resulting sequence is the discrete trajectory. This procedure is illustrated in Fig. 4.

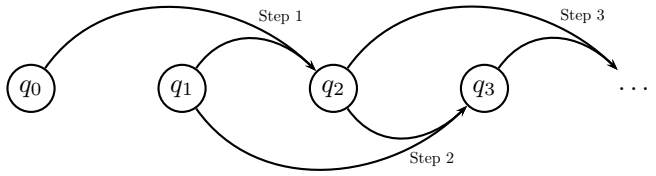


Fig. 4: A root finder solves the discrete Euler Lagrange equation to determine the next configuration from the previous and current configurations. This process is iterated to find the entire trajectory.

A. Creating a Variational Integrator

There are generally two common approaches to implement a variational integrator.[6] In the first, one explicitly finds the equations of motion (the discrete Euler-Lagrange equation) manually or with symbolic algebra software. For large systems the complexity essentially requires a symbolic algebra package such as *Mathematica*, but such tools only make the task possible in a formal sense. Realistically, the equations become so large and complex that they are impractical to manipulate.

Alternatively, the system can be described using a special choice of coordinates that result in special Lagrangian forms [15]. The most common examples are treating everything as a point mass ($L(q) = \dot{q}^T M q + V(q)$) [18] or treating each body as being free in space and imposing the mechanical structure through constraints [6]. Integrators based on these forms have excellent performance because of their simplicity, but lose the benefits and convenience of generalized coordinates.

With the tree description, *we can achieve comparable performance and still work in generalized coordinates*. The integrator works for arbitrary systems, not dependent on symbolic algebra software, and by taking advantage of caching, performance scales very well.

We begin by considering (16). At each time step, we must solve $f(q_{k+1}) = 0$. The Newton-Raphson root finding algorithm [25] performs very well for this problem. The Newton-Raphson root solver uses a linear model function to iteratively improve the estimated root until a satisfactory solution is found:

$$\begin{array}{l}\text{Seed } q_{k+1} = q_k \\ \textbf{while } f(q_{k+1}) \neq 0 \textbf{ do} \\ \quad z = -Df^{-1}(q_{k+1}) \cdot f(q_{k+1}) \\ \quad q_{k+1} = q_{k+1} + z \\ \textbf{return } \ q_{k+1}\end{array} \quad (17)$$

[6]Though we focus on variational integrators, this discussion largely applies to continuous Lagrangian mechanics.

This algorithm requires that the derivative $Df(\cdot)$ is available:

$$Df(q_{k+1}) = D_2 D_1 L_d\left(q_k, q_{k+1}\right) \quad (18)$$

We must now choose a discrete Lagrangian to implement (16) and (18). A common choice is the generalized midpoint approximation [32].

$$L_d(q_k, q_{k+1}) = L\left((1-\alpha)q_k + \alpha q_{k+1}, \tfrac{q_{k+1}-q_k}{\Delta t}\right)\Delta t \quad (19)$$

where $\alpha \in [0,1]$ is an algorithm parameter and $\alpha = \frac{1}{2}$ leads to second order accuracy [32]. Figures 3a, 3b, and 3c correspond to (19) with $\alpha = 0$, $\alpha = \frac{1}{2}$, and $\alpha = 1$, respectively.

We find derivatives of (19) using the chain rule:

$$\begin{array}{l}D_1 L_d(q_k, q_{k+1}) = \\ \quad \frac{\partial}{\partial q} L\left((1-\alpha)q_k + \alpha q_{k+1}, \frac{q_{k+1}-q_k}{\Delta t}\right)(1-\alpha)\Delta t - \\ \quad \frac{\partial}{\partial \dot{q}} L\left((1-\alpha)q_k + \alpha q_{k+1}, \frac{q_{k+1}-q_k}{\Delta t}\right)\end{array} \quad (20)$$

$$\begin{array}{l}D_2 L_d(q_{k-1}, q_k) = \\ \quad \frac{\partial}{\partial q} L\left((1-\alpha)q_{k-1} + \alpha q_k, \frac{q_k-q_{k-1}}{\Delta t}\right)\alpha\Delta t + \\ \quad \frac{\partial}{\partial \dot{q}} L\left((1-\alpha)q_{k-1} + \alpha q_k, \frac{q_k-q_{k-1}}{\Delta t}\right)\end{array} \quad (21)$$

$$\begin{array}{l}D_2 D_1 L_d(q_k, q_{k+1}) = \\ \quad \frac{\partial^2}{\partial q \partial q} L\left((1-\alpha)q_k + \alpha q_{k+1}, \frac{q_{k+1}-q_k}{\Delta t}\right)(1-\alpha)\alpha\Delta t + \\ \quad \frac{\partial^2}{\partial \dot{q} \partial q} L\left((1-\alpha)q_k + \alpha q_{k+1}, \frac{q_{k+1}-q_k}{\Delta t}\right)(1-\alpha) - \\ \quad \frac{\partial^2}{\partial q \partial \dot{q}} L\left((1-\alpha)q_k + \alpha q_{k+1}, \frac{q_{k+1}-q_k}{\Delta t}\right)\alpha - \\ \quad \frac{\partial^2}{\partial \dot{q} \partial \dot{q}} L\left((1-\alpha)q_k + \alpha q_{k+1}, \frac{q_{k+1}-q_k}{\Delta t}\right)\frac{1}{\Delta t}\end{array} \quad (22)$$

Eq. (20), (21), and (22) allow us to calculate (16) and (18) in terms of the continuous Lagrangian and its derivatives.

We continue by finding the necessary derivatives of the continuous Lagrangian (11):

$$\begin{aligned}\frac{\partial L}{\partial q_i} &= \frac{\partial}{\partial q_i}\left[\sum_k \tfrac{1}{2} v_{s,k}^{bT} M_k v_{s,k}^b - \sum_k V_k(q)\right] \\ &= \sum_k \left[\tfrac{1}{2}\frac{\partial v_{s,k}^b}{\partial q_i}^T M_k v_{s,k}^b + \tfrac{1}{2} v_{s,k}^{bT} M_k \frac{\partial v_{s,k}^b}{\partial q_i}\right] - \sum_k \frac{\partial V_k}{\partial q_i}(q) \\ &= \sum_k v_{s,k}^{bT} M_k \frac{\partial v_{s,k}^b}{\partial q_i} - \sum_k \frac{\partial V_k}{\partial q_i}(q)\end{aligned} \quad (23)$$

Eq. (23) is evaluated from the tree structure using (5) and (6). The remaining derivatives are found similarly.

$$\begin{array}{l}\frac{\partial^2 L}{\partial q_i \partial q_j} = \\ \sum_k \left[\frac{\partial v_{s,k}^{bT}}{\partial q_j} M_k \frac{\partial v_{s,k}^b}{\partial q_i} + v_{s,k}^{bT} M_k \frac{\partial^2 v_{s,k}^b}{\partial q_i \partial q_j}\right] - \sum_k \frac{\partial^2 V_k}{\partial q_i \partial q_j}(q)\end{array} \quad (24)$$

Eq. (24) is evaluated from the tree description using (5), (6), and (7).

$$\frac{\partial L}{\partial \dot{q}_i} = \sum_k v_{s,k}^{bT} M_k \frac{\partial v_{s,k}^b}{\partial \dot{q}_i} \quad (25)$$

Eq. (25) is evaluated from the tree using (5) and (8).

$$\frac{\partial^2 L}{\partial q_i \partial \dot{q}_j} = \sum_k \left[\frac{\partial v_{s,k}^{bT}}{\partial \dot{q}_j} M_k \frac{\partial v_{s,k}^b}{\partial q_i} + v_{s,k}^{bT} M_k \frac{\partial^2 v_{s,k}^b}{\partial q_i \partial \dot{q}_j} \right] \quad (26)$$

Eq. (26) is evaluated from the tree structure using (5), (6), (8), and (10).

$$\frac{\partial^2 L}{\partial \dot{q}_i \partial \dot{q}_j} = \sum_k \left[\frac{\partial v_{s,k}^{bT}}{\partial \dot{q}_j} M_k \frac{\partial v_{s,k}^b}{\partial \dot{q}_i} + v_{s,k}^{bT} M_k \frac{\partial^2 v_{s,k}^b}{\partial \dot{q}_i \partial \dot{q}_j} \right] \quad (27)$$

Eq. (27) is evaluated from the tree using (5), (8), and (9). Once (23) - (27) can be evaluated, we can completely evaluate (20), (21), and (22) and, therefore, implement a variational integrator for an arbitrary tree description. The user only needs to provide a tree description and two initial configurations to simulate the system.

B. Continuous Lagrangian Dynamics

We note that this approach also works for the continuous dynamics in generalized coordinates. The Euler-Lagrange equation (13) is expanded using the chain rule:

$$\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}} \ddot{q} + \frac{\partial^2 L}{\partial q \partial \dot{q}} \dot{q} - \frac{\partial L}{\partial q} = 0 \quad (28)$$

where the Lagrangian's dependence on q and $\dot{q}$ has been dropped. This is similar to the standard form, $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + V(q) = 0$, but left in terms of the Lagrangian. If the operator $\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}}$ is invertible[7], (28) can be solved for $\ddot{q}$:

$$\ddot{q} = \left(\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}} \right)^{-1} \left(\frac{\partial L}{\partial q} - \frac{\partial^2 L}{\partial q \partial \dot{q}} \dot{q} \right) \quad (29)$$

We can evaluate the above using (23), (26), and (27). A standard numeric integration package such as MATLAB integrates (29) to simulate the system. Again, this avoids explicitly calculating the equations of motion, which tend to be intractably large for complex systems.

Comparing (29) to the discrete algorithm (17) highlights an similarity between the two methods. Both algorithms require large matrix inversions to solve the dynamics. In both cases, we can detect a singular matrix and abort the simulation.

C. Potential Energies

Potential energies are included in the simulation through the generalized terms $V(q)$ and their derivatives. Each type of potential energy has a different form for $V(q)$. These are implemented manually, but in a way that uses the tree calculations and makes them applicable to arbitrary systems. This technique provides a great deal of flexibility for including potentials.

The common potential energies encountered in mechanical systems are gravity and springs. We demonstrate a basic gravity model and linear spring.

[7]This is the system's inertia tensor $M(q)$ expressed in generalized coordinates.

1) Gravity: We commonly use the simple model of gravity for mechanical systems:

$$F = m\vec{g}$$

where $\vec{g}$ is the gravity vector, typically $[0 \ 0 \ -9.81]^T$. The potential created by this force as applied to a mass at point $p_{s,k}(q)$ is:

$$V(q) = -m_k \vec{g} \cdot p_{s,k} \quad (30)$$

The two derivatives are straightforward:

$$\frac{\partial V}{\partial q_i}(q) = -m_k \vec{g} \cdot \frac{\partial}{\partial q_i} p_{s,k} \quad (31)$$

$$\frac{\partial^2 V}{\partial q_i \partial q_j}(q) = -m_k \vec{g} \cdot \frac{\partial^2}{\partial q_i \partial q_j} p_{s,k} \quad (32)$$

Eq. (30), (31), and (32) are evaluated using (2), (3), and (4) from a tree description. Typically, we would automatically add a gravity potential for every mass in the system. For more exotic simulations, however, we might selectively add this gravity model for some masses and a different model for others. In this way the approach is very flexible.

This same approach can be taken for the nonlinear gravity model ($F = -G\frac{m_1 m_2}{r^2}\hat{r}$) commonly used in celestial mechanics.

2) Springs: Suppose we have a linear spring with spring constant k and natural length x_0 connecting two points p_1 and p_2. The potential energy for the spring is

$$V(q) = \tfrac{1}{2} k (x - x_0)^2$$

where x is the distance between p_1 and p_2. The derivatives are found manually:

$$\frac{\partial V}{\partial q_i}(q) = k\,(x - x_0)\,\frac{\partial x}{\partial q_i}$$

$$\frac{\partial^2 V}{\partial q_i \partial q_j} = k \frac{\partial x}{\partial q_j} \frac{\partial x}{\partial q_i} + k\,(x - x_0)\,\frac{\partial^2 x}{\partial q_i \partial q_j}$$

The spring length x is

$$x(q) = (\vec{v}^T \vec{v})^{\frac{1}{2}}$$

where $\vec{v} = p_1 - p_2$. The derivatives are found:

$$\frac{\partial x}{\partial q_i}(q) = x^{-1} \vec{v}^T \frac{\partial \vec{v}}{\partial q_i}$$

$$\begin{aligned} \frac{\partial^2 x}{\partial q_i \partial q_j}(q) = \quad & -\tfrac{1}{2} x^{-2} \frac{\partial x}{\partial q_j} \vec{v}^T \frac{\partial \vec{v}}{\partial q_i} + \\ & x^{-1} \frac{\partial \vec{v}}{\partial q_j}^T \frac{\partial \vec{v}}{\partial q_i} + \\ & x^{-1} \vec{v}^T \frac{\partial^2 \vec{v}}{\partial q_i \partial q_j} \end{aligned}$$

Finally, the spring vector $\vec{v}$ and its derivatives are found:

$$\vec{v}(q) = p_1 - p_2 \quad (33)$$

$$\frac{\partial \vec{v}}{\partial q_i}(q) = \frac{\partial p_1}{\partial q_i} - \frac{\partial p_2}{\partial q_i} \quad (34)$$

$$\frac{\partial^2 \vec{v}}{\partial q_i \partial q_j}(q) = \frac{\partial^2 p_1}{\partial q_i \partial q_j} - \frac{\partial^2 p_2}{\partial q_i \partial q_j} \quad (35)$$

Eq. (33) through (35) are evaluated with (2), (3), and (4) from the tree representation. This same approach is taken to include nonlinear springs, torsional springs, etc.

D. Constraints

Both continuous and discrete Lagrangian mechanics can include constraints in the system. The tree representation does not change how constraints are included, but can help in calculating the necessary values. Similarly, the constraints do not change the tree representation at all. The constraints depend on the values provided by tree, but the tree does not depend on the constraints.

In discrete mechanics, we typically only deal with holonomic constraints and therefore do not discuss non-holonomic constraints here. However, [5] shows how to do this, and that technique could be implemented using the method presented in the present work. A holonomic constraint restricts the system to a sub-manifold of the configuration. Holonomic constraints are defined as $h(q) = 0 \in \mathbb{R}$ for valid configurations. A system may be subject to many holonomic constraints at once. These are grouped together as a vector of the individual constraints:

$$h(q) = \begin{bmatrix} h_1(q) \\ h_2(q) \\ \vdots \\ h_m(q) \end{bmatrix} \in \mathbb{R}^m$$

1) Continuous Constrained Dynamics: The constrained Euler-Lagrange equations are derived by minimizing the action $S(q(\cdot))$ subject to the constraint $h(q(\tau)) = 0 \ \forall \ \tau \in [t_0, t_f]$. The derivation [19] yields the constrained Euler Lagrange equations:

$$\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}} \ddot{q} + \frac{\partial^2 L}{\partial q \partial \dot{q}} \dot{q} - \frac{\partial L}{\partial q} = \frac{\partial h}{\partial q}^T (q) \lambda \tag{36a}$$

$$\frac{\partial^2 h}{\partial q \partial q}(q) \cdot (\dot{q}, \dot{q}) + \frac{\partial h}{\partial q}(q) \ddot{q} = 0 \tag{36b}$$

Note that the original constraint $h(q)$ doesn't directly appear in (36). Instead, the holonomic constraints are enforced in (36b) as constraints on the acceleration. This often leads to errors that slowly creep into the simulation during numeric integration. As the error grows, the constraints are increasingly violated.

Special techniques, such as projecting the system into the constraint sub-manifold or introducing damped-spring restoring forces, are used to fix this but these tend to add or remove energy from the system and introduce simulation parameters that have to be adjusted for individual scenarios. Bad choices for these parameters can introduce unstable dynamics.

2) Discrete Constraint Dynamics: The constrained variational integrator is derived by minimizing the discrete action sum (14) subject to $h(q_k) = 0 \ \forall \ k = 0 \dots N$. This leads to the constrained DEL equations [20]:

$$D_2 L_d(q_{k-1}, q_k) + D_1 L_d(q_k, q_{k+1}) = Dh^T(q_k) \lambda_k$$
$$h(q_{k+1}) = 0$$

We now have $n + m$ non-linear equations to solve in terms of q_{k+1} and λ_k. We define a new equation for the root solver:

$$f\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} D_2 L_d(q_{k-1}, q_k) + D_1 L_d(q_k, q_{k+1}) - Dh^T(q_k)\lambda_k \\ h(q_{k+1}) \end{bmatrix}$$

and find the necessary derivative:

$$Df\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} D_2 D_1 L_d(q_k, q_{k+1}) & -Dh^T(q_k) \\ Dh(q_{k+1}) & 0 \end{bmatrix}$$

The discrete integrator enforces $h(q_k) = 0$ directly at every time step. This eliminates the aforementioned error creep and results in trajectories that always satisfy the constraint.

From the above, constraints are included in the simulation by providing $h(q)$ and $Dh(q)$ for each type of constraint.[8] In the following sections, we present several constraints as examples.

3) Wire Constraint: A wire constraint holds two points at a fixed distance apart, as if they are connected by a stiff wire.

Suppose we have two points $p_1(q), p_2(q) \in \mathbb{R}^3$ to be a fixed distance $L \in \mathbb{R}$ apart. The constraint $h(q)$ is

$$\begin{aligned} h(q) &= 0 \\ &= ||p_1 - p_2||^2 - L^2 \\ &= (p_1 - p_2)^T (p_1 - p_2) - L^2 \end{aligned} \tag{37}$$

The above is evaluated using (2) from a tree representation. The derivative is found manually

$$\begin{aligned} \frac{\partial h}{\partial q_i}(q) &= \frac{\partial}{\partial q_i}\left((p_1 - p_2)^T (p_1 - p_2) - L^2\right) \\ &= 2 (p_1 - p_2)^T \left(\frac{\partial p_1}{\partial q_i} - \frac{\partial p_2}{\partial q_i}\right) \end{aligned} \tag{38}$$

and is evaluated using (2) and (3).

Again, by implementing (37) and (38) and providing a way to define the constraint, the simulator can use the wire constraint in arbitrary systems.

4) Point Constraint : Another common holonomic constraint is for two points to be coincident. This can be used, for example, to create a pin or spherical joint holding two parts together.

A natural approach is to use the wire constraint from above with $L = 0$. However, this introduces singularities because the constraint force direction (determined by (38)) is zero (or numerically near zero) when the constraint is satisfied.

Instead, we declare multiple constraints, each with a fixed direction. In this case, each constraint requires the distance between two points along an axis to be zero:

$$h(q) = \hat{n} \cdot (p_1 - p_2)$$

where $\hat{n}$ is the direction of the constraint. Terms p_1 and p_2 are the two points to be constrained and are calculated from (2).

The derivative

$$\frac{\partial h}{\partial q_i}(q) = \hat{n} \cdot \left(\frac{\partial p_1}{\partial q_i} - \frac{\partial p_2}{\partial q_i}\right)$$

is evaluated using (2) and (3).

To connect two points, we define multiple constraints with orthogonal directions. For example, to connect two points in $\mathbb{R}^3$, we define three constraints with linearly independent directions (e.g. $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$). We could also redefine the constraint so that $\hat{n}$ is specified relative to one of the coordinate frames associated with p_1 or p_2.

[8] The continuous case also requires $D^2 h(q)$, but we focus on the discrete case.

5) Screw Constraint : One limitation of the proposed six primitive transformations is they cannot naturally represent screw joints. However, we can build a screw joint by using constraints.

A screw motion is a rotation about an axis by an angle θ followed by a translation along the same axis by a distance d. The ratio of the distance to the angle is called the pitch $p = d/\theta$ [21].

We define the constraint equation:

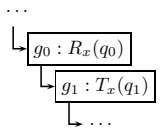
$$h(q) = pq_\theta - q_d \tag{39}$$

where q_θ is a configuration variable that is parameterizing a rotation and q_d is a configuration variable that is parameterizing a translation.

The derivative is:

$$\frac{\partial h}{\partial q_i}(q) = \begin{cases} p & i = \theta \\ -1 & i = d \\ 0 & d \neq i \neq \theta \end{cases} \tag{40}$$

To set up a screw joint, we create a frame with a rotation about an axis and add a child that translates along the same axis:



A screw constraint is created using q_0 for q_θ and q_1 for q_d. The resulting system will model a screw joint.

E. Forcing

Another common extension to Lagrangian mechanics is external forcing. Forcing is used to include dissipation (e.g. friction), control inputs (e.g. motor torque), and other effects. The Lagrange d'Alembert principle is used to introduce external forcing to the continuous Euler-Lagrange equation [19]. A forcing term is added to the action integral:

$$\delta \int_{t_0}^{t_f} L(q(\tau), \dot{q}(\tau))\ \mathrm{d}\tau + \int_{t_0}^{t_f} f_c(q(\tau), \dot{q}(\tau), \tau) \cdot \delta q\ \mathrm{d}\tau = 0$$

where $f_c(q, \dot{q}, t)$ is the total external forcing expressed in the system's generalized coordinates. This derivation leads to the forced Euler-Lagrange equation:

$$\frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{q}}(q, \dot{q}) - \frac{\partial L}{\partial q}(q, \dot{q}) = f_c(q, \dot{q}, t)$$

In discrete mechanics, we approximate the continuous force $f_c(q, \dot{q}, t)$ with left and right discrete forces $f_d^-(q_k, q_{k+1}, t_k, t_{k+1})$ and $f_d^+(q_k, q_{k+1}, t_k, t_{k+1})$ such that:

$$f_d^-\left(q_k, q_{k+1}, t_k, t_{k+1}\right) \cdot \delta q_k + f_d^+\left(q_k, q_{k+1}, t_k, t_{k+1}\right) \cdot \delta q_{k+1} \approx \int_{t_k}^{t_{k+1}} f_c\left(q(\tau), \dot{q}(\tau), \tau\right) \cdot \delta q\ \mathrm{d}\tau$$

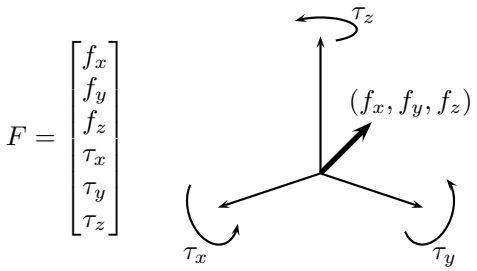


Fig. 5: The wrench F combines a force applied to the origin of a coordinate frame and torques applied about each axis.

The discrete forcing can be determined from a number of approximations. For example, we may choose an approximation analogous the discrete Lagrangian:

$$f_d^{\alpha-}\left(q_k, q_{k+1}, t_k, t_{k+1}\right) = \frac{1}{2} f_c\left((1-\alpha)q_k + \alpha q_{k+1}, \tfrac{q_{k+1}-q_k}{\Delta t}, (1-\alpha)t_k + \alpha t_{k+1}\right)\Delta t$$

$$f_d^{\alpha+}\left(q_k, q_{k+1}, t_k, t_{k+1}\right) = \frac{1}{2} f_c\left((1-\alpha)q_k + \alpha q_{k+1}, \tfrac{q_{k+1}-q_k}{\Delta t}, (1-\alpha)t_k + \alpha t_{k+1}\right)\Delta t$$

The discrete analog to the Lagrange d'Alembert principle is:

$$\delta \sum_{k=0}^{N-1} L_d\left(q_k, q_{k+1}\right) + \sum_{k=0}^{N-1} \left(f_d^-\left(q_k, q_{k+1}, t_k, t_{k+1}\right) \cdot \partial q_k + f_d^+\left(q_k, q_{k+1}, t_k, t_{k+1}\right) \cdot \partial q_{k+1}\right) = 0 \tag{42}$$

Solving (42) leads to the forced discrete Euler-Lagrange equation:

$$D_2 L_d\left(q_{k-1}, q_k\right) + f_d^+\left(q_{k-1}, q_k, t_{k-1}, t_k\right) + D_1 L_d\left(q_k, q_{k+1}\right) + f_d^-\left(q_k, q_{k+1}, t_k, t_{k+1}\right) = 0 \tag{43}$$

We again use (43) to solve for q_{k+1} from a given previous and current configuration, q_{k-1} and q_k.

1) Transforming Forces: The above includes forces that are expressed in generalized coordinates. However, we often describe forces using linear vectors and torques relative to a body frame in the system. These forces must be transformed into generalized coordinates.

A wrench, F, combines a linear force and three torques into a single vector in $\mathbb{R}^6$. The linear force is applied to the origin of a coordinate frame. The three torques are applied about each axis of the frame. See Fig. 5.

The wrench F is transformed into generalized coordinates by the body Jacobian [21] for the coordinate frame, $g_{s,i}(q)$:

$$f_c = \left[J_{s,i}^b\right]^T F$$

where f is the equivalent force in generalized coordinates.

The Jacobian can be calculated from values provided by the tree representation (specifically, (2) and (3)):

$$J_{s,i}^b = \left[\left(g_{s,i}^{-1}\frac{\partial g_{s,i}}{\partial q_0}\right)^\vee \quad \left(g_{s,i}^{-1}\frac{\partial g_{s,i}}{\partial q_1}\right)^\vee \quad \cdots \quad \left(g_{s,i}^{-1}\frac{\partial g_{s,i}}{\partial q_n}\right)^\vee\right]$$

A similar approach can be used to include forces applied to a frame but specified in spatial coordinates by using the spatial Jacobian.

V. IMPLEMENTATION: `trep`

We have been developing a simulation package called `trep`[9] based on Sections III and IV. The `trep` package allows a user to create a tree representation of a mechanical system and provides cached implementations of (2) through (10). A variational integrator is implemented on top of the tree representation to simulate arbitrary mechanical systems in generalized coordinates. Additionally, `trep` can calculate the continuous mechanics (i.e. $\ddot{q}, \lambda$) but does not provide numeric integration facilities.

The `trep` package is implemented as a Python package with a C-back-end for performance critical sections. This arrangement makes it particularly convenient to use without sacrificing speed. In this section, we consider several important aspects of the implementation and give an example to illustrate the relative ease of specifying systems in the implementation.

A. Variational Integrator

The `trep` package implements a forced, constrained variational integrator using the methods described in this paper. The combined integration equations are:

$$f\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} D_2L_d(q_{k-1},q_k) + f_d^+(q_{k-1},q_k,t_{k-1},t_k) + \\ \quad D_1L_d(q_k,q_{k+1}) + f_d^-(q_k,q_{k+1},t_k,t_{k+1}) - Dh^T(q_k)\lambda_k \\ h(q_{k+1}) \end{bmatrix} \tag{44}$$

We can improve the performance of (44) by noticing that several terms are constant with respect to parameters q_{k+1} and λ_k. We define the terms

$$p_k = D_2L_d(q_{k-1},q_k) + f_d^+(q_{k-1},q_k,t_{k-1},t_k)$$
$$\pi_k = Dh(q_k)$$

In the absence of forcing, p_k is the momentum quantity conserved by the integrator [32]. In general, however, it is defined only for computation convenience. The integrator equation becomes:

$$f\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} p_k + D_1L_d(q_k,q_{k+1}) + f_d^-(q_k,q_{k+1},t_k,t_{k+1}) - \pi_k^T\lambda_k \\ h\left(q_{k+1}\right) \end{bmatrix} \tag{45}$$

The derivative is:

$$Df\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} D_2D_1L_d(q_k,q_{k+1}) + D_2f_d^-(q_k,q_{k+1},t_k,t_{k+1}) & -\pi_k^T \\ \pi_{k+1}\left(= Dh(q_{k+1})\right) & 0 \end{bmatrix} \tag{46}$$

[9]The name `trep` is derived from "tree representation"

This avoids calculating $D_2L_d(\cdot)$ during each root solver iteration. Note that introducing the p_k term removes the explicit dependence on q_{k-1} and the integrator becomes a one step mapping $(q_k, p_k) \to (q_{k+1}, p_{k+1})$.

We now require an initialization procedure for the integrator:

Require: q_0, q_1
  Set tree: $q = (1-\alpha)q_0 + \alpha q_1$, $\dot{q} = \frac{q_1 - q_0}{\Delta t}$
  $p_1 = D_2L_d(q_0,q_1) + f_d^+(q_0,q_1)$
  Set tree: $q = q_1$, $\dot{q} = \frac{q_1 - q_0}{\Delta t}$
  $\pi_1 = Dh(q_1)$
  $\lambda_0 = 0$
  return $q_1, p_1, \pi_1, \lambda_0$

Recall that the root finding algorithm is:

  Seed $q_{k+1} = q_k$
  Seed $\lambda_k = \lambda_{k-1}$
  while $|f(q_{k+1},\lambda_k)| < tolerance$ **do**
    $z = -Df^{-1}(q_{k+1},\lambda_k) \cdot f(q_{k+1},\lambda_k)$
    $\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix} = \begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix} + z$
  return q_{k+1}, λ_k

If this is implemented naively, each root solver iteration writes four configurations to the tree. First, we set $q = (1-\alpha)q_k + q_{k+1}$ to find the top part of (45), then $q = q_{k+1}$ to find the bottom part of (45), and then repeat for (46). Each write erases the cached values in the tree, so avoiding writes reduces computation and improves performance.

By expanding the algorithm and rearranging the order of evaluation, we can reduce the number of writes to two per iteration. Additionally, once the solution is found, we calculate p_{k+1} with the tree already in the correct cached state. The optimized simulation algorithm is as follows:

Require: q_k, p_k, π_k, λ_{k-1}
  Seed $q_{k+1} = q_k$
  Seed $\lambda_k = \lambda_{k-1}$
  Set tree: $q = q_{k+1}$
  $f_2 = h(q_{k+1})$
  $Df_{2,1} = \pi_{k+1} = Dh(q_{k+1})$
  Set tree: $q = (1-\alpha)q_k + \alpha q_{k+1}$, $\dot{q} = \frac{q_{k+1} - q_k}{\Delta t}$
  $f_1 = p_k + D_1L_d(q_k,q_{k+1}) + f_d^-(q_k,q_{k+1}) - \pi_k^T\lambda_k$
  $Df_{1,2} = -\pi_k^T$
  $Df_{2,2} = 0$
  while $|f| < tolerance$ **do**
    $Df_{1,1} = D_2D_1L_d(q_k,q_{k+1}) + D_2f_d^-(q_k,q_{k+1})$
    $z = -Df^{-1} \cdot f$
    $\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix} = \begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix} + z$
    Set tree: $q = q_{k+1}$
    $f_2 = h(q_{k+1})$
    $Df_{2,1} = \pi_{k+1} = Dh(q_{k+1})$
    Set tree: $q = (1-\alpha)q_k + \alpha q_{k+1}$, $\dot{q} = \frac{q_{k+1} - q_k}{\Delta t}$
    $f_1 = p_k + D_1L_d(q_k,q_{k+1}) + f_d^-(q_k,q_{k+1}) - \pi_k\lambda_k$
  $p_{k+1} = D_2L_d(q_k,q_{k+1}) + f_d^+(q_k,q_{k+1})$
  Set tree: $q = q_{k+1}$, $\dot{q} = \frac{q_{k+1} - q_k}{\Delta t}$
  return $\lambda_k, q_{k+1}, p_{k+1}, \pi_{k+1}$

By carefully arranging the order in which components are evaluated, this algorithm minimizes writes to the tree configuration to maximize the use of cached values.

B. System Specification

One advantage of uniformly representing systems using trees is that we can specify systems using compact forms. S-expressions, the syntactic form used in the LISP programming language, is well suited for this task. In this section, we briefly describe the s-expression syntax used in `trep` to specify new systems.[10] These expressions allow one to quickly create new simulations without having to write any new code, thus improving reliability.

As an example, Listing 1 is the s-expression used to describe the device shown in Fig. 6. The first three terms ((`ry` "J" ...), (`tx` -1.5 ...), (`tx` 1.5 ...)) define the three "arms" for the device. The last four terms (`point-constraint...`) connect the arms form the closed kinematic chains. We emphasize that Listing 1 is actual input to `trep`, not pseudo-code.

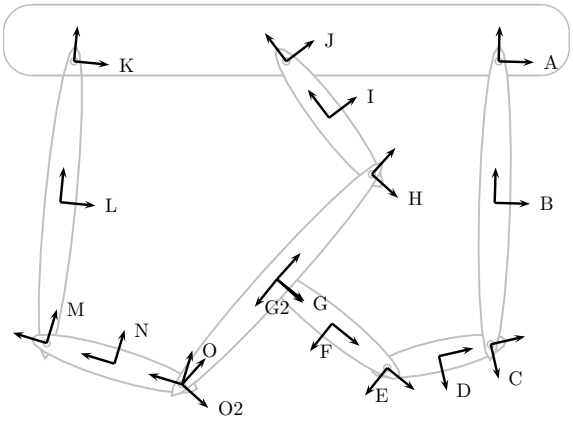


Fig. 6: A mechanical system with many closed kinematic chains.

The syntax is described using regular definitions [1]. Symbols beginning with $ are non-terminal expressions. **Bold** tokens are literal s-expression symbols while **bold** parentheses are the s-expression parentheses. Expressions are grouped with {curly brackets}. | is the logical "or" operator. + requires the preceding expression to occur one or more times. ∗ requires the preceding expression can appear zero or more times. A trailing question mark indicates an optional expression that occurs zero or one time.

Table II shows the syntax for defining mechanical systems.

1) $system: A $system expression defines a tree representation of a mechanical system. It is simply a list of components to include in the system.

C. $system-component

$system-components are frames, constraints, potential energies, and forces. `trep` is designed to let users easily add new constraints, potentials, and forces that can be included in this group and supported by s-expressions.

[10]Note that the s-expressions are merely implemented by a Python script that builds systems through `trep`'s Python API. The same API can be used to build systems directly in code instead of using s-expressions

Listing 1: S-Expression definition for the system in Fig. 6

```
(mechanical-system (gravity 0 0 -9.81)
  (ry "J" (Name "J")
      (tz -0.5 (Name "I") (Mass 1))
      (tz -1.0
          (ry "H" (Name "H")
              (tz -1.0 (Name "G") (Mass 1))
              (tz -2.0 (Name "O2")))))
  (tx -1.5
      (ry "K" (Name "K")
          (tz -1.0 (Name "L") (Mass 1))
          (tz -2.0
              (ry "M" (Name "M")
                  (tz -0.5 (Name "N") (Mass 1))
                  (tz -1.0 (Name "O"))))))

  (tx 1.5
      (ry "A" (Name "A")
          (tz -1.0 (Name "B") (Mass 1))
          (tz -2.0
              (ry "C" (Name "C")
                  (tz -0.375 (Name "D") (Mass 1))
                  (tz -0.75
                      (ry "E" (Name "E")
                          (tz -0.5 (Name "F") (Mass 1))
                          (tz -1.0 (Name "G2"))))))))
 (point-constaint "G" "G2" (1 0 0))
 (point-constaint "G" "G2" (0 0 1))
 (point-constaint "O" "O2" (1 0 0))
 (point-constaint "O" "O2" (0 0 1)))
```

D. $gravity

$gravity expressions include a global linear gravity potential. The three numbers define the gravity force vector.

1) $frame: A $frame expression defines a coordinate frame. The parent of the frame is implied by the expression's location. Frames declared in a $system expression are children of the world frame. Frames declared as arguments to another frame are children of that frame.

$transform-type defines the transformation to the frame from its parent. The following $transform-param is the transformation's parameter. At least one type-param pair is required. If two or more are specified, the frame is expanded into multiple frames with each following frame being the child of the previous. For example, (`TX 0.1 TY 0.2 TZ 0.3` ...) is equivalent to (`TX 0.1 (TY 0.2 (TZ 0.3` ...`)))`. The remaining parameters will apply to the final frame.

A number of $frame-option expressions can be included to modify the frame.

Subsequent *frame* sub-expressions create new frames that are children of the current frame.

2) $transform-type: A $transform-type expression defines the transformation that relates a child frame to its parent. The values correspond to translations along and rotations about the X, Y, and Z axes.

3) $transform-param: A $transform-param expression defines the driving parameter for the transformation between a child frame and its parent.

A $number parameter indicates a fixed, constant transformation.

$system → **(SYSTEM** $system-component+**)**
$system-component → $gravity | $frame | $point-constraint | . . .
$gravity → **(GRAVITY** $number $number $number**)**
$frame → **(** {$transform-type $transform-param}+ $frame-option* $frame***)**
$transform-type → **TX** | **TY** | **TZ** | **RX** | **RY** | **RZ**
$transform-param → $number | $string | **(D** $string**)**
$frame-option → $mass | $name
$mass → **(MASS** $string**)**
$name → **(NAME** $string**)**
$point-constraint → **(POINT-CONSTRAINT** $string $string $number $number $number $name?**)**

TABLE II: Syntax for defining tree-form mechanical systems in `trep`.

A $string or **(D** $string**)** parameter makes the transformation dependent on a dynamic configuration variable. The $string becomes the name of the configuration variable.

4) $mass: A $mass expression defines a mass. The first number is the mass of the object. The second, third, and fourth numbers are the mass' I_{xx}, I_{yy}, and I_{zz} rotational inertias. If the rotational inertias are not specified, they default to zero to create a point mass.

5) $point-constraint: A $point-constraint expression creates the constraint described in Section IV-D4. The two strings specify the names of the frames that are to be joined. The three numbers define the direction vector for the constraint. An optional name can be specified.

E. Automatic Visualization

Because a tree description encodes so much structure of a mechanical system, automatically generating visual representations is trivial. This provides immediate feedback on a system's definition and simulation in a form that is accessible to users. Whereas bad simulations can be hard to identify from plots of configuration variables, they are almost immediately apparent when animated. Not only does this save time and effort, it also eliminates a subtle class of errors where a manually created visualization does not correctly represent the system being simulated or analyzed.

VI. Example Simulations

Several examples are provided to demonstrate the variety of systems we can simulate with `trep` and make comparisons where appropriate. Animated results of these simulations can be viewed at our website *http://robotics.colorado.edu/trep*.

A. N-link Pendulum

The N-link pendulum provides a practical way to numerically study how a simulator scales with system size. This is particularly true in generalized coordinates where a link is explicitly dependent on every link above it, making this a sort of worse-case scenario.

The N-link pendulum consists of N links in a plane. Each link has a mass (with rotational inertia) attached at the bottom. The simulation starts with the pendulum links aligned and the top link rotated by 45 degrees.

Fig. 7 plots simulation runtime against the number of links N in the pendulum. The simulations last 20 seconds and use a step-size of 0.01s. The dashed line represents simulations without caching in the tree calculations. The solid line represents simulations with caching.

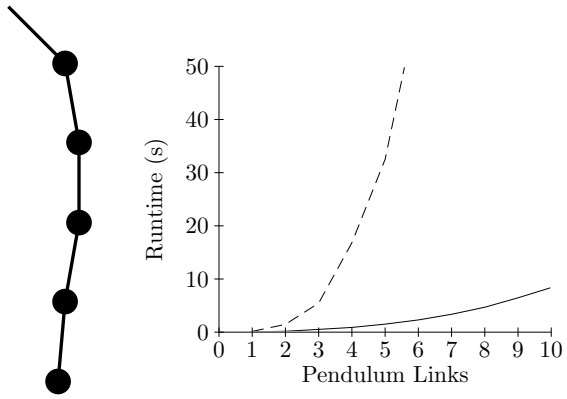


Fig. 7: Simulation runtime vs. number of pendulum links. The dashed line represents simulations run without caching while the solid line represents simulations with caching.

The non-caching times are on par with simulations based on symbolic equations of motion. The results show the value of caching using this method. The simulations are drastically faster and they grow at a slower rate as pendulum size increases.

B. Closed-Chain Device

Closed kinematic chains are typically considered as challenging features to handle in dynamics simulation. The mechanism shown in Fig. 6 is an example of a device with many closed kinematic chains [22].

While force-balance simulators can handle this system, they must introduce restoring forces to maintain the constraints over time. This causes artificial energy dissipation that can be seen on relatively short time scales.

This system was simulated in `trep` and the freely available Open Dynamics Engine (`ODE`) [28]. The system was simulated for 120 seconds with a time step of $0.01s$. The `ODE` runtime was $1.86s$. The `trep` runtime was $22.1s$. The total energy as a function of time is plotted in Fig. 8.

While `ODE` outperforms `trep` time-wise by a factor of 10, it introduces significant damping while the variational integrator maintains near constant energy. Additionally, the `trep` simulation is carried out in generalized coordinates while the `ODE` simulation consists of seven independent bodies connected by constraints. The speed advantage may not be useful when the resulting trajectory is fundamentally flawed.

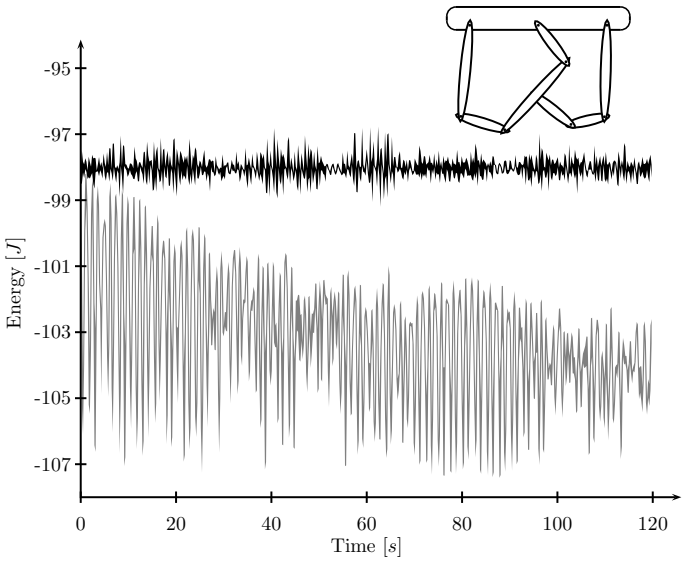


Fig. 8: A mechanical system with many closed kinematic chains. Total energy of the closed chain device simulated for 120 seconds with a time step of $0.01s$. The variational integrator simulation is shown in black. The ODE simulation is shown in gray.

C. Scissor Lift

Finally, we consider the mechanism shown in Fig. 9, commonly found in industrial lifts (and old cartoons). We consider the mechanism to be hanging rather than lifting to avoid introducing actuation.

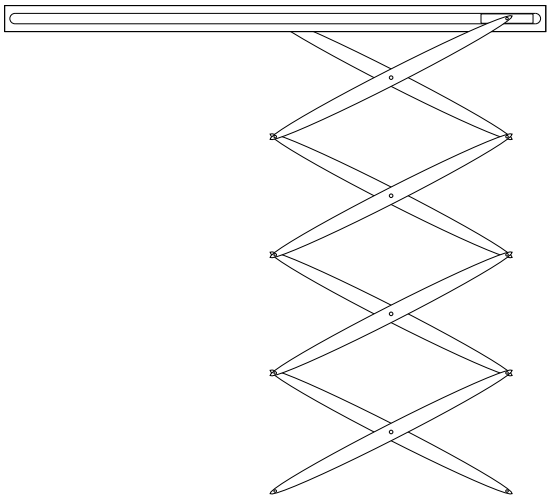


Fig. 9: The scissor lift has many bodies and closed kinematic chains but only one degree of freedom.

The scissor lift has many links and many closed kinematic chains, but can be reduced to a single degree of freedom (DOF). Additionally, its complexity is parameterized by varying the number of segments. We can write the Lagrangian for the equivalent one DOF system and use an accurate numeric integrator to generate a benchmark trajectory for comparison.

A schematic of the device is shown in Fig. 10. The top segment is pinned in the upper left. The upper right joint is pinned to a mass m_S that slides horizontally without friction. Each link has mass m_L at its center and rotational inertia I.

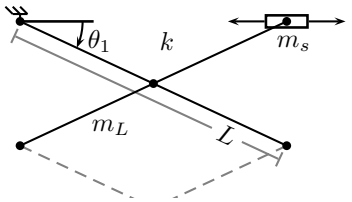


Fig. 10: A schematic for the top of the scissor lift. Each link has mass m_L at the center and no rotational inertia.

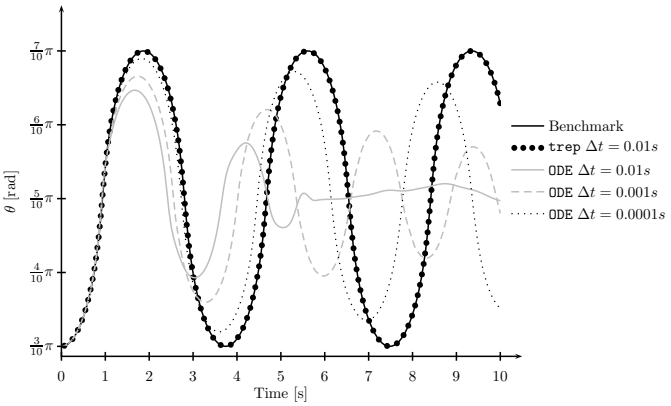


Fig. 11: Simulated trajectories for the scissor lift with 5 segments.

The Lagrangian for a lift is

$$L(\theta,\dot{\theta}) = \sum_{n=1}^{N} \left(m_L L^2 \left(\left(\tfrac{1}{2} - n\right)^2 \cos^2\theta_1 + \tfrac{1}{2}\sin^2\theta_1 + I \right) \dot{\theta}_1^2 + 2m_L g (n - \tfrac{1}{2}) L \sin\theta_1 \right) + \tfrac{1}{2} m_S L^2 \sin^2\theta_1 \theta_1^2$$

where N is the number of segments.

A benchmark solution was generated for a 5 segment lift from the above Lagrangian using Mathematica's `NDSolve[]` function. The system was simulated with `trep` for 10 seconds with a time step of $0.01s$ and took 1.74 seconds to compute. The system was also simulated in `ODE` with time steps of $0.01s$, $0.001s$, and $0.0001s$ and took 0.19, 1.85, and 18.47 seconds to compute, respectively. The simulated trajectories are plotted in Fig. 11.

The variational integrator tracks the benchmark solution almost perfectly. The `ODE` simulation, on the other hand, is clearly unsatisfactory. The $0.01s$ trajectory dissipates most of the energy immediately. The smaller time step trajectories are only slightly better. They dissipate energy more slowly but still depart from the true trajectory quickly. All three solutions result in an incorrect period of oscillation.

The variational integrator continues to perform well for long time scales. Fig. 12 shows the trajectory after 1000 seconds. There is a slight phase shift from accumulated error,

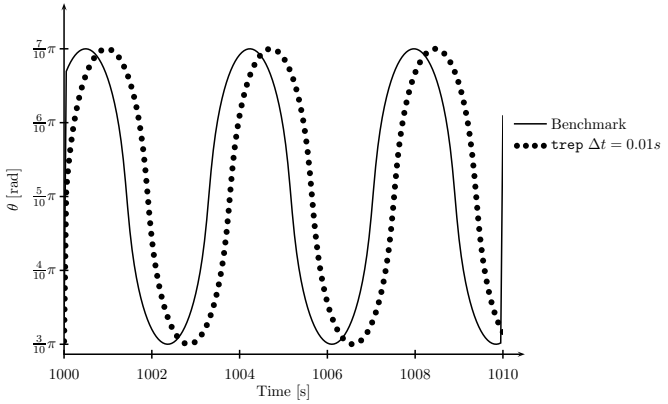


Fig. 12: Long-time simulated trajectories for the scissor lift with 5 segments.

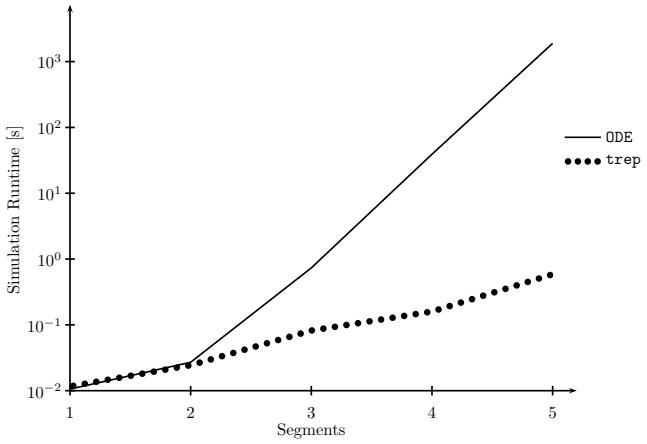


Fig. 13: Simulation runtime vs. Number of Segments in a scissor Lift. The simulation runtime was found by setting the time step so that the simulation time was $15s \pm 1\%$ when accumulated error exceeded 0.1. The y-axis is logarithmic.

but the amplitude, shape, and frequency are still close to the benchmark trajectory.

1) Scissor Lift Complexity Analysis: Traditional complexity analysis in mechanics considers the computational effort to calculate the state derivative depending on the number of bodies or degrees of freedom in the system. This type of complexity analysis is useful to compare similar algorithms, but becomes less meaningful as algorithms grow further apart. This is particularly true when comparing discrete mechanics, where the system is advanced by a root-finding problem, and continuous dynamics, where the system is advanced with numeric integration. There are additional fundamental differences like how constraints are handled.

We can consider comparing methods by running simulations with equivalent parameters and measuring the computational time taken as in Fig. 11. As we saw, however, this doesn't account for the validity of the result, and the amount of error in the simulation can vary widely.

Another approach is to include simulation error in our comparison. Given a benchmark trajectory $\theta_b(t)$ and a simulation result $\theta_s(t)$, the simulation error is defined as

$$e = \int_{t_0}^{t_f} \left(\theta_b(\tau) - \theta_s(\tau)\right)^2 \mathrm{d}\tau \tag{47}$$

First, we choose a desired error and simulation time. An N-segment lift is simulated until the error exceeds the desired error. If the simulation time is longer than the desired time, we increase the step size and re-run the simulation. If the simulation time is shorter, we decrease the step size. We iterate until the achieved simulation time is approximately equal to the desired time.

Figure 13 shows the results of this analysis when the desired time was 15 seconds and the desired error was 0.1. In this case `ODE` scales so poorly that the results must be plotted logarithmically. The results show that despite `ODE` having linear dynamics, it does not perform as well as the variational integrator because the integrator stepsize must be reduced to maintain an acceptable error.

Of course, the results of this analysis are limited to the scissor lift. There may be other examples that favor `ODE` over `trep`. The main point is that the result of traditional complexity analysis that `ODE` should always scale better can be misleading in practice because of error.

VII. Conclusion

Tree descriptions have enabled us to create a variational integrator for arbitrary mechanical systems in generalized coordinates. The recursive equations derived from the hierarchy lend themselves well to optimization (both fundamental optimizations like caching and incremental improvements like using vectorized hardware) that make the technique scale to large systems in generalized coordinates. The organization and structure used in the description has also proven to be convenient to work with (for example, to automatically generate visualizations).

The tree description is also appealing for its versatility. Though we emphasize variational integrators here, the same approach works for the continuous dynamics with the traditional Euler-Lagrange equation.

Similarly, variational integrators can be derived with special forms instead of the tree description. This can improve performance at the expense of generalized coordinates. Such integrators would still retain the benefits of good energy behavior and hack-less holonomic constraints.

It is unlikely that a variational integrator in generalized coordinates will ever outperform a constrained force-balance simulation,[11] but this technique at least makes them fast enough to be practical for many large systems. The additional benefits they offer, like good energy behavior and directly including holonomic constraints, are important enough for many applications to be worth the performance penalty.

There are two major areas to continue developing this work. The controls community has developed robust trajectory

[11]Here we mean outperform purely in terms of computational speed given similar time steps. A broader definition of performance that includes the accuracy of the trajectory may tip the balance in favor of variational integrators as seen in Sec. VI-C.

exploration algorithms that are very versatile [14], generalized coordinates are essential to keep the state size small and to develop meaningful cost/objective functions. As a result, these techniques have been limited to small systems or require highly optimized models that are hand-tailored to the problem. We believe that the methods in this paper will expand their practical application to complex mechanical systems. The tree description is also well suited to derive the higher derivatives and linearized dynamics needed for trajectory exploration.

Our simulator is also missing some important components. Some, like collisions and impacts [29], have been studied and developed for variational integrators [12]. These algorithms just need to be adapted to work with the tree representation. Others, like non-holonomic constraints [10], have been largely ignored in discrete mechanics (with a few exceptions [5]) but are needed to study phenomenon like slip-steered vehicles and contact mechanics. The tree framework may also be extended to explicitly include compliant/elastic meshes [23] that can be attached to coordinate frames.

VIII. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under CAREER award CMS-0546430. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

We would additionally like to acknowledge useful conversations with Prof. Magnus Egerstedt at the Georgia Institute of Technology.

REFERENCES

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Pinciples, Techniques, and Tools*. Addison-Wesley, 1988.

[2] AIST, Univ. of Tokyo, , and MSTC. Open architecture humanoid robotics platform, 2008. http://www.is.aist.go.jp/humanoid/openhrp/.

[3] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *SIGGRAPH*, 1994.

[4] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *SIGGRAPH*, 1996.

[5] P. Betsch. A unified approach to the energy-consistent numerical integration of nonholonomic mechanical systems and flexible multibody dynamics. *GAMM Mitteilungen*, 27:66–87, 2004.

[6] P. Betsch and S. Leyendecker. The discrete null space method for the energy consistent integration of constrained mechanical systems. part ii: Multibody dynamics. *Int. J. Numer. Meth. Engng*, 67(499-552), 2006.

[7] B. Bollobas. *Modern Graph Theory*. Springer, 1998.

[8] Roger W. Brockett, Ann Stokes, and Frank Park. A geometrical formulation of the dynamical equations describing kinematic chains. In *Proc. IEEE Conf. on Robotics and Automation*, pages 637–641, Atlanta, GA, 1993.

[9] F. Bullo and A.D. Lewis. *Geometric Control of Mechanical Systems*. Number 49 in Texts in Applied Mathematics. Springer-Verlag, 2004.

[10] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion*. The MIT Press, 2005.

[11] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, 1987.

[12] R.C. Fetecau, J.E. Marsden, M. Ortiz, and M. West. Nonsmooth Lagrangian mechanics and variational collision integrators. *SIAM Journal on Applied Dynamical Systems*, 2003.

[13] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration*. Springer Series in Computational Mathematics; 31. Springer-Verlag, 2004.

[14] J. Hauser and A. Saccon. A barrier function method for the optimization of trajectory functionals with constraints. In *Conference on Dynamics and Controls*, 2007.

[15] L. Kharevych, W. Yang, Y. Tong, E. Kanso, J. E. Marsden, P. Schroder, and M. Desbrun. Geometric, variational integrators for computer animation. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2006.

[16] A. Lew, J.E. Marsden, M. Ortiz, and M. West. An overview of variational integrators. *Finite Element Methods: 1970's and Beyond*, 2003.

[17] A. Lew, J.E. Marsden, M. Ortiz, and M. West. Variational time integrators. *International Journal for Numerical Methods in Engineering*, pages 153–212, 2004.

[18] S. Leyendecker, P. Betsch, and P. Steinmann. The discrete null space method for the energy consistent integration of constrained mechanical systems: Part iii: Flexible multibody dynamics. *Multibody System Dynamics*, 2007.

[19] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer, 1999.

[20] J. E. Marsen and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, pages 357–514, 2001.

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

[22] Y. Nakamura and K. Yamane. Dynamics computation of structure-varying kinematic chains and its application to human figures. *IEEE Transactions on Robotics and Automation*, 16(2), 2000.

[23] J. S. Pang, V. Kumar, and V. Song. Convergence of time-stepping method for initial and boundary-value frictional compliant contact problems. *SIAM J. Numerical Analysis*, 43(5):2200–2206, 2006.

[24] Frank C. Park and James E. Bobrow. A recursive algorithm for robot dynamics using lie groups. *International Conference on Robotics and Automation*, 1994.

[25] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C Second Edition*. Cambridge University Press, 1992.

[26] David Schaechter, Thomas Kane, David Levinson, and Paul Mitiguy. Autolev, 2008. http://www.autolev.com/.

[27] R. Smith. Dynamics Simulation: A whirlwind tour (current state, and new frontiers), 2004. http://ode.org/slides/parc/dynamics.pdf.

[28] R. Smith. Open Dynamics Engine, 2008. http://www.ode.org.

[29] D.E. Stewart and J.C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal of Numerical Methods in Engineering*, 39:2673–2691, 1996.
[30] Lung-Wen Tsai. *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*. Wiley-IEEE, 1999.
[31] M. West, C. Kane, J. E. Marsden, and M. Ortiz. Variational integrators, the Newmark scheme, and dissipative systems. 1999.
[32] Matthew West. Variational integrators. *California Institute of Technology Thesis*, 2004.
[33] A. Witkin, M. Gleicher, and W. Welch. Interactive dynamics. In *Computer Graphics*, 1990.
[34] Katsu Ymane. *Simulating and Generating Motions of Human Figures*. Springer-Verlag, 2004.