Scalable Variational Integrators for Constrained Mechanical Systems in Generalized Coordinates

Elliot R. Johnson, *Student Member, IEEE,* Todd D. Murphey, *Member, IEEE,*

***Abstract*—We present a technique to implement scalable variational integrators for generic mechanical systems in generalized coordinates. Systems are represented by a tree-based structure that provides efficient means for algorithmically calculating values (position, velocities, derivatives) needed for variational integration without resorting to explicit equations of motion. The variational integrator handles closed kinematic chains, holonomic constraints, dissipation, and external forcing without modification. By avoiding the full equations of motion, using recursive equations, and caching calculated values, this method scales to large systems while using generalized coordinates. An example of a closed-kinematic-chain system is included along with a comparison with the Open Dynamics Engine (`ODE`) to illustrate the scalability and desirable energetic properties of the technique.**

***Index Terms*—Dynamics, Animation and Simulation, Variational Integrators**

I. INTRODUCTION

Variational integrators are relatively new tools for modeling dynamic systems. In particular, they are an alternative to continuous ordinary differential equations for simulating mechanical systems. Variational integrators often yield superior results, but were thought to be impossible to scale to large systems without specialized coordinate choices. This paper, however, describes an implementation of scalable variational integrators for arbitrary mechanical systems in generalized coordinates. The approach relies on a tree-based representation of the system that efficiently calculates positions, velocities, and their derivatives (numerically and without approximation). The resulting integrators have excellent long-time behavior, excellent treatment of holonomic constraints, and excellent performance.

Variational integrators are a class of symplectic integrators [10] that preserve (or nearly preserve, depending on the particular integrator) fundamental properties like conservation of energy and momentum [19]. They have traditionally been used to study conservative systems (e.g. celestial mechanics), but modern variational integrators also include forcing and dissipation. In fact, the results often have better accuracy than classical methods. Variational integrators also deal with holonomic constraints and non-smooth phenomenon like collisions well.

While energy conservation does not guarantee an accurate solution, it is desirable that a simulation respects known properties of a system like conservation of energy and momentum. When these properties are not satisfied, the resulting trajectory is certainly incorrect.

Variational integrators 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 are not implemented directly; they are replaced with equivalent acceleration constraints. Errors introduced during numeric integration cause the system to violate the original constraint, resulting in unrealistic behavior like objects 'sinking' into hard surfaces [25]. Restoring forces–resembling damped springs–are added to correct these errors. These add "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. V).

Variational integrators work with holonomic constraints directly and completely avoid these problems.

The current approach to variational integration is to derive the full discrete equations of motion in generalized coordinates, or to choose special coordinate representations that give simple, but scalable, equations. Equations of motion in generalized coordinates typically become unmanageable for large systems; this has been a major obstacle in using variational integrators for complex systems when generalized coordinates are required.

Continuous dynamics was once in a similar state. Equations of motion were either written explicitly, or algorithms that depend on specific coordinate choices were used. In particular, there is a long tradition of representing systems as collections of free bodies and adding constraints to impose structure [30]. This is still the dominant method when the coordinate choice is irrelevant (e.g. computer graphics, video games). The widely used Open Dynamics Engine (`ODE`) [26] takes this approach.

Controls applications, on the other hand, typically require generalized coordinates [6] so that important notions like controllability and observability are meaningful. For simple systems, we usually derive the full equations of motion, either by hand or with the help of software like Mathematica or Autolev [24]. This approach does not scale to large systems. Instead, we must turn to algorithmic methods of computing the dynamics at each time step.

Featherstone [8] developed methods that use a tree[1] representation of a mechanical system to calculate the (continuous) forward dynamics. They key to this approach is using recursive

[1]More generally, we may represent a system as a graph with tree descriptions as the subset of directed acyclic graphs [4]. However, the applications presented in this paper do not derive any immediate benefits to considering trees from a graph theoretic perspective, so we do not advocate this perspective here.

equations that explicitly reuse information and avoid the repetition seen in the equations of motion. Several existing software packages (e.g. OpenHRP [1]) use this approach to simulate mechanical systems.

This same approach can be used to derive variational integrators in generalized coordinates. The discrete dynamics are calculated algorithmically instead of needing the full equations of motion. We use a different tree representation that is based on homogeneous coordinate transformations. 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 [14] 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 and other control analysis [11]. We do not, however, discuss derivatives in this paper beyond those needed for dynamic simulation.

This paper is an expanded version of [12]. It includes thorough explanations and implementation details. The algorithm has been expanded to include generic potential energies, holonomic constraints, and external forcing. Furthermore, the examples presented here demonstrate many important and unique features of the algorithm and include comparisons with existing techniques.

We begin with an introduction to our tree representation in Sec. II. Sections II-A and II-B show how to compute positions, velocities, and their derivatives from the tree. Discrete mechanics are introduced in Sec. III and then used to implement a variational integrator based on the tree descriptions in Sec. III-A. Section III-B briefly describes how continuous dynamics can also be derived in this framework. Section III-C demonstrates how potential energies are calculated for the system. Section III-D and Sec. III-E extend the variational integrator to include constraints and non-conservative forcing. Section IV introduces a software package called `trep` (available at *http://trep.sourceforge.net/*) that implements the presented methods. Finally, Sec. V presents examples and comparisons with existing methods.

A. Homogeneous Representations

We assume the reader is familiar with homogeneous representations of rigid body transformations. Homogeneous representations provide a uniform way to represent translations and rotations and play a dual role as both transformations and configurations. See [20] for a complete introduction and reference.

II. Tree Representations

Mechanical systems often have a natural hierarchical structure–the wrist depends on the forearm, which depends on the upper arm. A tree representation formalizes this structure by attaching coordinate frames to bodies in a system, organizing them into parent-child relationships, and defining rigid body transformations from the parent to each child. Each frame has a single parent. The root of the tree–and only frame without a parent–is the stationary world frame s.

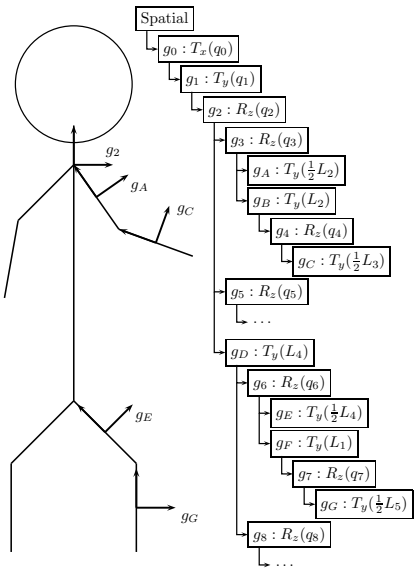


Fig. 1: A planar human is represented with a tree structure. Note that although we are restricted to simple transformations, we can represent complex mechanical systems by composing multiple transformations.

We use six basis transformations, represented as elements of $\mathrm{SE}(3)$: translations along and rotations about the parent's X, Y, and Z axes. 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. Complex joints are modeled by compositions of the six transformations. More complex joints (e.g. screws, cams, gears) are modeled with compositions and additional constraints.

Fig. 1 is an example tree description for a planar human. Note that coordinate frames do not necessarily have to have an associated mass and so we can define as many frames as desired. Consequently, a tree representation is not unique. There are an infinite number of representations for a given system.

Fig. 1 also demonstrates the naming convention used throughout this paper. Constant transformations 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

correspondence between configuration variables and variable transformations. They are not absolutely necessary for this approach, but they significantly simplify the discussion, notation, and equations without 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 provides uniform generalized coordinates consisting only of real-valued numbers.

Note that R1 disallows 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]

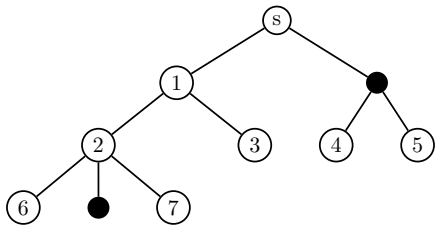
R2 prevents using the same configuration variable to drive multiple transformations, which is useful for modeling systems with parallel linkages. Parallel linkages and closed chains are instead handled by connecting open chains with holonomic constraints.

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 $g_{s,k}(q)$).

A tree representation by itself is a clean organization of a system's mechanical structure, but it also provides a foundation for efficiently calculating quantities needed for dynamic simulation and analysis. In the following section, we present methods for computing the position, velocities, and their derivatives for all frames in a tree.

Note: 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.

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.
$v^b_{s,i} \in T_eSE(3)$	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.
$\mathrm{anc}(i)$	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.
$g'_k = \frac{\partial}{\partial q_k} g_k$	The derivative of a transformation of the i-th frame to its parent frame.
$g''_k = \frac{\partial^2}{\partial q_k \partial q_k} g_k$	The second derivative of a transformation of the i-th frame to its parent frame.
$\dot{g}_k = \frac{\partial}{\partial t} g_k$	The time derivative of a transformation of the i-th frame to its parent frame.
$(\cdot)^\wedge : \mathbb{R}^6 \to T_eSE(3)$	The 'hat' operator maps twists to the $T_eSE(3)$.
$(\cdot)^\vee : T_eSE(3) \to \mathbb{R}^6$	The 'unhat' operator maps elements of $T_eSE(3)$ to twists in $\mathbb{R}^6$

A. Frame Positions

The rigid body transformation from any frame in a tree to the spatial reference frame is a straightforward calculation. 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 $g_{s,k}(q)$:

$$g_{s,k}(q) = \begin{cases} I & k = s \\ g_{s,\mathrm{par}(k)} g_k & k \neq s \end{cases} \tag{1}$$

Eq. (1) 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 acyclic by definition. 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 (1) 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 (1) with respect to a configuration variable q_i is found with the standard derivative:

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

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

[2]SO(3) parameterizations require constraints on the nine matrix elements to ensure the SO(3) matrix remains orthogonal [20].

ancestor of g_k, the derivative is always zero:

$$\tfrac{\partial}{\partial q_i} g_{s,k}(q) = \begin{cases} 0 & k = s \\ & i \notin \mathrm{anc}(k) \\ \tfrac{\partial}{\partial q_i}\left(g_{s,\mathrm{par}(k)}\right) g_k & i \neq k \\ g_{s,\mathrm{par}(k)} g_k' & i = k \end{cases} \tag{2}$$

where $g_k' = \frac{\partial}{\partial q_k} g_k(q_k)$. Using (1) and (2), 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 ideas.

$$\tfrac{\partial^2}{\partial q_j \partial q_i} g_{s,k}(q) = \begin{cases} & k = s \\ 0 & i \notin \mathrm{anc}(k) \\ & j \notin \mathrm{anc}(k) \\ g_{s,\mathrm{par}(k)} g_k'' & i = k = j \\ \tfrac{\partial}{\partial q_j}[g_{s,\mathrm{par}(k)}] g_k' & i = k \neq j \\ \tfrac{\partial}{\partial q_i}[g_{s,\mathrm{par}(k)}] g_k' & i \neq k = j \\ \tfrac{\partial^2}{\partial q_j \partial q_i}[g_{s,\mathrm{par}(k)}] g_k & i \neq k, j \neq k \end{cases} \tag{3}$$

Eq. (3) uses itself recursively along with (1) and (2) 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.

B. 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 $\hat{\xi}$ [20]. The twist is constant (i.e. it does not depend on the configuration) for each type of transformation:

$$\hat{v}^b_{s,k}(q,\dot{q}) = \begin{cases} 0 & k = s \\ g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k + \hat{\xi}_k \dot{q}_k & k \neq s \end{cases} \tag{4}$$

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

[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 [20]. 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.

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

$$\begin{cases} 0 & k = s \\ & i \notin \mathrm{anc}(k) \\ g_k^{-1\prime}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & i = k \\ g_k^{-1}\tfrac{\partial}{\partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & i \neq k \end{cases} \tag{5}$$

Eq. (5) is evaluated recursively with itself, the local rigid body transformations, and (4).

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

$$\begin{cases} & k = s \\ 0 & i \notin \mathrm{anc}(k) \\ & j \notin \mathrm{anc}(k) \\ g_k^{-1\prime\prime}\hat{v}^b_{s,\mathrm{par}(k)} g_k + 2 g_k^{-1\prime}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & \\ \quad + g_k^{-1}\hat{v}^b_{s,\mathrm{par}(k)} g_k'' & i = k = j \\ g_k^{-1\prime}\tfrac{\partial}{\partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\tfrac{\partial}{\partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & i \neq k = j \\ g_k^{-1\prime}\tfrac{\partial}{\partial q_j}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\tfrac{\partial}{\partial q_j}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & i = k \neq j \\ g_k^{-1}\tfrac{\partial^2}{\partial q_j \partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & i \neq k, j \neq k \end{cases} \tag{6}$$

Eq. (6) is evaluated using itself, local rigid body transformations, (4), and (5). We are also interested in derivatives of body velocities with respect to the configuration variable time derivatives.

$$\tfrac{\partial}{\partial \dot{q}_i}\hat{v}^b_{s,k}(q,\dot{q}) = \begin{cases} 0 & k = s, \\ & i \notin \mathrm{anc}(k) \\ \hat{\xi}_k & i = k \\ g_k^{-1}\tfrac{\partial}{\partial \dot{q}_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & i \neq k \end{cases} \tag{7}$$

$$\tfrac{\partial^2}{\partial \dot{q}_j \partial \dot{q}_i}\hat{v}^b_{s,k}(q,\dot{q}) = 0 \tag{8}$$

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

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

$$\begin{cases} & k = s \\ & k = j \\ 0 & i \notin \mathrm{anc}(k) \\ & j \notin \mathrm{anc}(k) \\ g_k^{-1\prime}\tfrac{\partial}{\partial \dot{q}_j}\hat{v}^b_{s,\mathrm{par}(k)} g_k + g_k^{-1}\tfrac{\partial}{\partial \dot{q}_j}\hat{v}^b_{s,\mathrm{par}(k)} g_k' & k = i \\ g_k^{-1}\tfrac{\partial^2}{\partial \dot{q}_j \partial q_i}\hat{v}^b_{s,\mathrm{par}(k)} g_k & k \neq i \end{cases} \tag{9}$$

Eq. (1) through (9) 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 approach.

C. Primitive Transformations

Eq. (1) through (9) 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 be parameterized easily. 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.

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

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 (1). Derivatives of $p_{s,k}$ are similarly extracted from the corresponding derivative of $g_{s,k}$ in (2) and (3).

D. Caching

Values are frequently reused in the above equations. 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 (1) through (9).

This technique, called caching, improves performance well enough to use the tree representation for fast and accurate dynamic simulation for large mechanical systems. In the following section, we introduce a variational integrator based on the tree representation.

III. 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 a quantity called the Lagrangian. The Lagrangian $L(q, \dot{q})$ for k interconnected rigid bodies 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 [5][22]. 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:

$$L(q, \dot{q}) = \sum_k \tfrac{1}{2} v_{s,k}^{bT} M_k v_{s,k}^b - \sum_k V_k(q) \tag{10}$$

Assuming we can evaluate $V_k(q)$, (10) can be evaluated numerically with (4). We continue to assume that $V_k(q)$ and derivatives are known. Their actual computation is discussed in Sec. III-C.

The Least Action principle states that the system will naturally follow the trajectory that minimizes[4] the action, S:

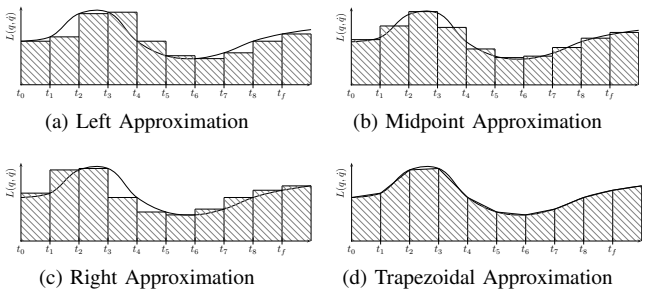
$$S(q([t_0, t_f])) = \int_{t_0}^{t_f} L(q(\tau), \dot{q}(\tau)) \mathrm{d}\tau \tag{11}$$

Minimizing (11) with a variational principle results in the Euler-Lagrange equations [18]:

$$\frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{q}}(q, \dot{q}) - \frac{\partial L}{\partial q}(q, \dot{q}) = 0 \tag{12}$$

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

[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.



(a) Left Approximation (b) Midpoint Approximation

(c) Right Approximation (d) Trapezoidal Approximation

Fig. 2: The discrete Lagrangian approximates segments of the continuous action integral. The area of each shaded region represents a value of the discrete Lagrangian.

important mechanical properties like conservation of energy and momentum.

Alternatively, we can introduce the discrete approximation earlier the derivation. This leads to integration methods that respect the fundamental symmetries in dynamics.[5] This approach is called discrete mechanics and the resulting integrators are known as Variational Integrators [15]. Variational integrators conserve (or nearly conserve, depending on the particular integrator) fundamental quantities like momentum and energy [16]. They are also well suited for problems involving holonomic constraints, impacts, and non-smooth phenomenon [9].

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, [19] 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.

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

Figure 2 shows several choices of approximations to create a discrete Lagrangian. The order of accuracy for the approximation is directly related to the order of accuracy for the resulting trajectory [19].

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

$$S(q([t_0, t_f])) = \int_{t_0}^{t_f} L(q(\tau), \dot{q}(\tau))\mathrm{d}\tau \approx \sum_{k=0}^{n-1} L_d\left(q_k, q_{k+1}\right) \tag{13}$$

The action sum is minimized with a variational principle to get an implicit difference equation–analogous to the Euler-Lagrange equations in (12)–called the Discrete Euler-Lagrange (DEL) equation.

$$D_1 L_d\left(q_k, q_{k+1}\right) + D_2 L_d\left(q_{k-1}, q_k\right) = 0 \tag{14}$$

where $D_n f(\ldots)$ is the derivative of $f(\ldots)$ with respect to its n-th argument. Note that the derivation of (14) is analogous to the approach used to derive the continuous dynamics equation (12).

Whereas the continuous Euler-Lagrange equation is an ordinary differential equation that is integrated to find the trajectory of the system, the discrete Euler-Lagrange equation (14) presents a root-finding problem to get the next configuration. Given two initial configurations q_0 and q_1, we solve

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

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.

A. Creating a Variational Integrator

There are generally two 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 too large and complex.

Alternatively, the system can be described using special coordinate choices that result in special Lagrangian forms [13]. The most common examples are treating everything as a point mass ($L(q) = \dot{q}^T M q + V(q)$) [17] or treating each body as being free in space and imposing the mechanical structure through constraints [3]. 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 achieve comparable performance and still work in generalized coordinates*. The integrator works for arbitrary systems, is not dependent on symbolic algebra software, and by taking advantage of caching, performance scales well.

We begin by considering (15). At each time step, we must solve $f(q_{k+1}) = 0$. The Newton-Raphson root finding algorithm [23] performs 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:

```
Seed q_{k+1} = q_k
while f(q_{k+1}) ≠ 0 do
    z = −Df^{-1}(q_{k+1}) · f(q_{k+1})
    q_{k+1} = q_{k+1} + z
return q_{k+1}
```

(16)

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

$$Df(q_{k+1}) = D_2 D_1 L_d\left(q_k, q_{k+1}\right) \tag{17}$$

[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 [29].

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

We must now choose a discrete Lagrangian to implement (15) and (17). A common choice is the generalized midpoint approximation [28].

$$L_d(q_k, q_{k+1}) = L\left((1-\alpha)q_k + \alpha q_{k+1}, \tfrac{q_{k+1}-q_k}{\Delta t}\right)\Delta t \quad (18)$$

where $\alpha \in [0,1]$ is an algorithm parameter and $\alpha = \frac{1}{2}$ leads to second order accuracy [28]. Figures 2a, 2b, and 2c correspond to (18) with $\alpha = 0$, $\alpha = \frac{1}{2}$, and $\alpha = 1$, respectively.

We find derivatives of (18) using the chain rule:

$$D_1 L_d(q_k, q_{k+1}) = \frac{\partial}{\partial q} L(\dots)(1-\alpha)\Delta t - \frac{\partial}{\partial \dot{q}} L(\dots) \quad (19)$$

$$D_2 L_d(q_{k-1}, q_k) = \frac{\partial}{\partial q} L(\dots)\alpha\Delta t + \frac{\partial}{\partial \dot{q}} L(\dots) \quad (20)$$

$$D_2 D_1 L_d(q_k, q_{k+1}) = \tfrac{\partial^2}{\partial q \partial q} L(\dots)(1-\alpha)\alpha\Delta t + \quad (21)$$
$$\tfrac{\partial^2}{\partial \dot{q} \partial q} L(\dots)(1-\alpha) - \tfrac{\partial^2}{\partial q \partial \dot{q}} L(\dots)\alpha - \tfrac{\partial^2}{\partial \dot{q} \partial \dot{q}} L(\dots)\tfrac{1}{\Delta t}$$

Eq. (19), (20), and (21) allow us to calculate (15) and (17) in terms of the continuous Lagrangian and its derivatives.

We continue by finding the necessary derivatives of the continuous Lagrangian (10):

$$\frac{\partial L}{\partial q_i} = \frac{\partial}{\partial q_i}\left[\sum_k \tfrac{1}{2} v_{s,k}^{bT} M_k v_{s,k}^b - \sum_k V_k(q)\right]$$
$$= \sum_k v_{s,k}^{bT} M_k \frac{\partial v_{s,k}^b}{\partial q_i} - \sum_k \frac{\partial V_k}{\partial q_i}(q) \quad (22)$$

Eq. (22) is evaluated from the tree structure using (4) and (5). The remaining derivatives are found similarly.

$\frac{\partial^2 L}{\partial q_i \partial q_j} =$

$$\sum_k \left[\frac{\partial v_{s,k}^{bT}}{\partial q_j} M_k \frac{\partial v_{s,k}^b}{\partial q_i} + v_{s,k}^{bT} M_k \frac{\partial^2 v_{s,k}^b}{\partial q_i \partial q_j}\right] - \sum_k \frac{\partial^2 V_k}{\partial q_i \partial q_j}(q) \quad (23)$$

Eq. (23) is evaluated from the tree description using (4), (5), and (6).

$$\frac{\partial L}{\partial \dot{q}_i} = \sum_k v_{s,k}^{bT} M_k \frac{\partial v_{s,k}^b}{\partial \dot{q}_i} \quad (24)$$

Eq. (24) is evaluated from the tree using (4) and (7).

$$\frac{\partial^2 L}{\partial q_i \partial \dot{q}_j} = \sum_k \left[\frac{\partial v_{s,k}^{bT}}{\partial \dot{q}_j} M_k \frac{\partial v_{s,k}^b}{\partial q_i} + v_{s,k}^{bT} M_k \frac{\partial^2 v_{s,k}^b}{\partial q_i \partial \dot{q}_j}\right] \quad (25)$$

Eq. (25) is evaluated from the tree structure using (4), (5), (7), and (9).

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

Eq. (26) is evaluated from the tree using (4), (7), and (8). Once (22) - (26) can be evaluated, we can completely evaluate (19), (20), and (21) and, therefore, implement a variational integrator for an arbitrary tree description.

B. Continuous Lagrangian Dynamics

We note that this approach also works for the continuous dynamics in generalized coordinates. The Euler-Lagrange equation (12) is expanded using the chain rule:

$$\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}} \ddot{q} + \frac{\partial^2 L}{\partial q \partial \dot{q}} \dot{q} - \frac{\partial L}{\partial q} = 0 \quad (27)$$

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], (27) can be solved for $\ddot{q}$:

$$\ddot{q} = \left(\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}}\right)^{-1}\left(\frac{\partial L}{\partial q} - \frac{\partial^2 L}{\partial q \partial \dot{q}}\dot{q}\right) \quad (28)$$

We can evaluate the above using (22), (25), and (26). A standard numeric integration package such as MATLAB integrates (28) to simulate the system. Again, this avoids explicitly calculating the equations of motion, which tend to be intractably large for complex systems.

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. Springs are equally easy to implement, but are not included in this discussion.

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:

$$V(q) = -m_k \vec{g} \cdot p_{s,k} \quad (29)$$

The two derivatives are straightforward:

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

$$\tfrac{\partial^2 V}{\partial q_i \partial q_j}(q) = -m_k \vec{g} \cdot \tfrac{\partial^2}{\partial q_i \partial q_j} p_{s,k} \quad (31)$$

Eq. (29), (30), and (31) are evaluated using (1), (2), and (3) 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.

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.

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

D. Constraints

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 deal with holonomic constraints only and therefore do not discuss non-holonomic constraints here. However, [2] shows how to do this, and that technique could be implemented using the method presented in this 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:

$$h(q) = \begin{bmatrix} h_1(q) & \dots & h_m(q) \end{bmatrix}^T \in \mathbb{R}^m$$

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 [18] yields the constrained Euler Lagrange equations:

$$\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}} \ddot{q} + \frac{\partial^2 L}{\partial q \partial \dot{q}} \dot{q} - \frac{\partial L}{\partial q} = \tfrac{\partial h}{\partial q}^T(q)\lambda \tag{32a}$$

$$\tfrac{\partial^2 h}{\partial q \partial q}(q) \cdot (\dot{q}, \dot{q}) + \tfrac{\partial h}{\partial q}(q)\ddot{q} = 0 \tag{32b}$$

where the Lagrange multipliers λ are related to the constraint forces and must be solved during integration. Note that the original constraint $h(q)$ doesn't directly appear in (32). Instead, the holonomic constraints are enforced in (32b) 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 Constrained Dynamics: The constrained variational integrator is derived by minimizing the discrete action sum (13) subject to $h(q_k) = 0 \ \forall \ k = 0 \dots N$. This leads to the constrained DEL equations [19]:

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

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.

The discrete Lagrange multipliers λ_k and the term $Dh^T(q_k)$ have the same interpretation as the continuous case. The term $Dh^T(q_k)\lambda_k$ represents the forcing applied by the constraints with λ_k being proportional to the magnitude of each constraint force.

From the above, constraints are included in the simulation by providing $h(q)$ and $Dh(q)$ for each type of constraint.[8] As an example, we derive a 'wire' constraint.

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

$$\begin{aligned} h(q) &= 0 \\ &= ||p_1 - p_2||^2 - L^2 \\ &= (p_1 - p_2)^T (p_1 - p_2) - L^2 \end{aligned} \tag{33}$$

The above is evaluated using (1) from a tree representation. The derivative is found manually

$$\begin{aligned} \frac{\partial h}{\partial q_i}(q) &= \tfrac{\partial}{\partial q_i}\left((p_1 - p_2)^T (p_1 - p_2) - L^2\right) \\ &= 2\,(p_1 - p_2)^T \left(\tfrac{\partial p_1}{\partial q_i} - \tfrac{\partial p_2}{\partial q_i}\right) \end{aligned} \tag{34}$$

and is evaluated using (1) and (2).

Again, by implementing (33) and (34) and providing a way to define the constraint, the simulator can use the wire constraint in arbitrary systems. Holonomic constraints allow us to model a wide range of joints including pin joints, screw motions, etc.

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 introduces external forcing to the continuous Euler-Lagrange equation [18]. 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 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:

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

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

$$F = \begin{bmatrix} f_x \\ f_y \\ f_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

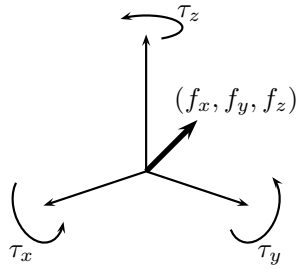


Fig. 3: The wrench F combines a force applied to the origin of a coordinate frame and torques applied about each axis.

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

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

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

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

$$\delta \sum_{k=0}^{N-1} L_d(q_k, q_{k+1}) + \sum_{k=0}^{N-1} \left(f_d^-(q_k, q_{k+1}, t_k, t_{k+1}) \cdot \partial q_k + f_d^+(q_k, q_{k+1}, t_k, t_{k+1}) \cdot \partial q_{k+1}\right) = 0 \quad (36)$$

Solving (36) leads to the forced discrete Euler-Lagrange equation:

$$D_2 L_d(q_{k-1}, q_k) + f_d^+(q_{k-1}, q_k, t_{k-1}, t_k) + D_1 L_d(q_k, q_{k+1}) + f_d^-(q_k, q_{k+1}, t_k, t_{k+1}) = 0 \quad (37)$$

We again use (37) to solve for q_{k+1} from a given previous and current configuration, q_{k-1} and q_k.

1) Transforming Forces: We can include non-generalized forces in an Euler-Lagrange simulation by transforming its wrench representation, F, into generalized coordinates:

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

where $J_{s,i}^b$ is the body Jacobian [20] of the frame to which the force is applied.

The Jacobian can be calculated from values provided by the tree representation (specifically, (1) and (2)):

$$J_{s,i}^b = \begin{bmatrix} \left(g_{s,i}^{-1}\frac{\partial g_{s,i}}{\partial q_0}\right)^\vee & \left(g_{s,i}^{-1}\frac{\partial g_{s,i}}{\partial q_1}\right)^\vee & \cdots & \left(g_{s,i}^{-1}\frac{\partial g_{s,i}}{\partial q_n}\right)^\vee \end{bmatrix}$$

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

IV. Implementation: `trep`

We have developed a simulation package called `trep`[9] based on Sections II and III. `trep` allows a user to create a tree representation of a mechanical system and provides cached implementations of (1) through (9). A variational integrator is implemented on top of the tree representation to simulate arbitrary mechanical systems in generalized coordinates. `trep` can calculate continuous mechanics (i.e. $\ddot{q}, \lambda$) but does not provide numeric integration facilities.

`trep` is implemented as a Python package with a C backend for performance critical sections. This arrangement makes it convenient to use without sacrificing speed. In this section, we consider several important aspects of the implementation.

A. Variational Integrator

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

$$f\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} D_2 L_d(q_{k-1}, q_k) + f_d^+(q_{k-1}, q_k, t_{k-1}, t_k) + \\ D_1 L_d(q_k, q_{k+1}) + f_d^-(q_k, q_{k+1}, t_k, t_{k+1}) - Dh^T(q_k)\lambda_k \\ h(q_{k+1}) \end{bmatrix} \quad (38)$$

We can improve the performance of (38) by noticing that several terms are constant with respect to parameters q_{k+1} and λ_k. We define the terms

$$p_k = D_2 L_d(q_{k-1}, q_k) + f_d^+(q_{k-1}, q_k, t_{k-1}, t_k)$$
$$\pi_k = Dh(q_k)$$

In the absence of forcing, p_k is the momentum quantity conserved by the integrator [28]. In general, however, it is defined only for computation convenience. The integrator equation becomes:

$$f\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} p_k + D_1 L_d(q_k, q_{k+1}) + f_d^-(q_k, q_{k+1}, t_k, t_{k+1}) - \pi_k^T \lambda_k \\ h(q_{k+1}) \end{bmatrix} \quad (39)$$

The derivative is:

$$Df\left(\begin{bmatrix} q_{k+1} \\ \lambda_k \end{bmatrix}\right) = \begin{bmatrix} D_2 D_1 L_d(q_k, q_{k+1}) + D_2 f_d^-(q_k, q_{k+1}, t_k, t_{k+1}) & -\pi_k^T \\ \pi_{k+1}\,(= Dh(q_{k+1})) & 0 \end{bmatrix} \quad (40)$$

This avoids calculating $D_2 L_d(\cdot)$ 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:

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

Require: q_0, q_1
  Set tree: $q = (1-\alpha)q_0 + \alpha q_1$, $\dot{q} = \frac{q_1 - q_0}{\Delta t}$
  $p_1 = D_2 L_d(q_0, q_1) + f_d^+(q_0, q_1)$
  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

`trep` includes additional optimizations that avoid redundant calculations and improve performance with compromising or simplifying the dynamics, but these are not discussed here.

The next section presents several example simulations that were carried out in `trep`

V. Example: The Scissor Lift

The mechanism shown in Fig. 4 is commonly found in industrial lifts and is an excellent example for comparing simulations with constraints. We consider the mechanism to be hanging rather than lifting to avoid introducing actuation.

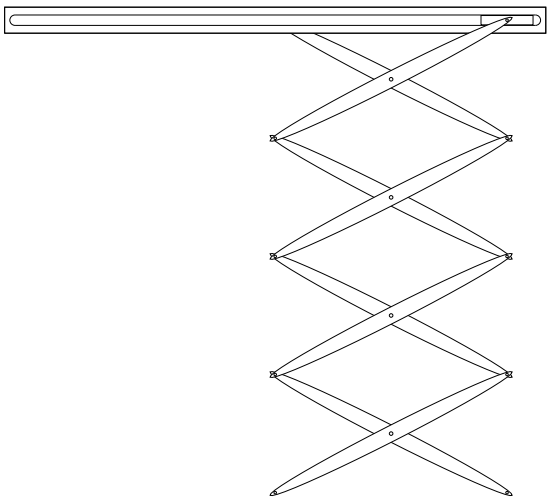


Fig. 4: The scissor lift has many bodies and closed kinematic chains but only one degree of freedom.

The scissor lift has many links and many closed kinematic chains, but can be reduced to a single degree of freedom (DOF). 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. 5. 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.

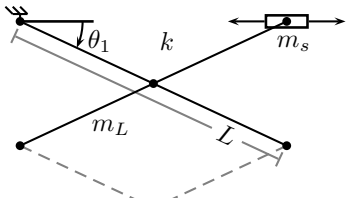


Fig. 5: A schematic for the top of the scissor lift. Each link has mass m_L at the center and rotational inertia I.

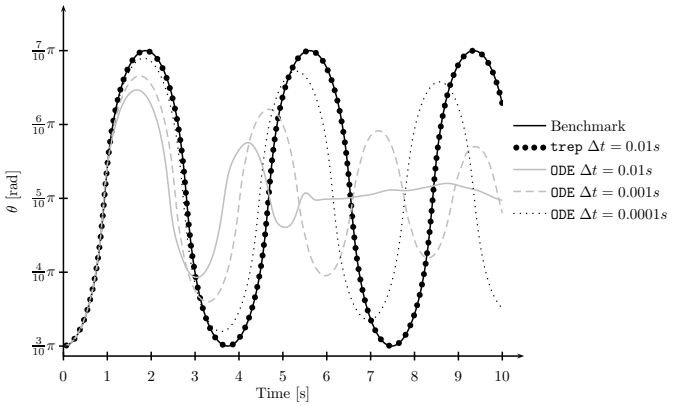


Fig. 6: Simulated trajectories for the scissor lift with 5 segments.

The Lagrangian for a lift is

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

where N is the number of segments.

A. Trajectory Accuracy

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. 6.

The variational integrator tracks the benchmark solution almost perfectly while the `ODE` simulation is clearly wrong. The $0.01s$ trajectory dissipates most of the energy immediately. Small time steps dissipate energy more slowly but still depart from the true trajectory quickly. All three `ODE` solutions result in an incorrect period of oscillation.

The variational integrator continues to perform well for long time scales. Even after 1000 seconds, there is only a slight phase shift from accumulated error while the amplitude, shape, and frequency are still close to the benchmark trajectory.

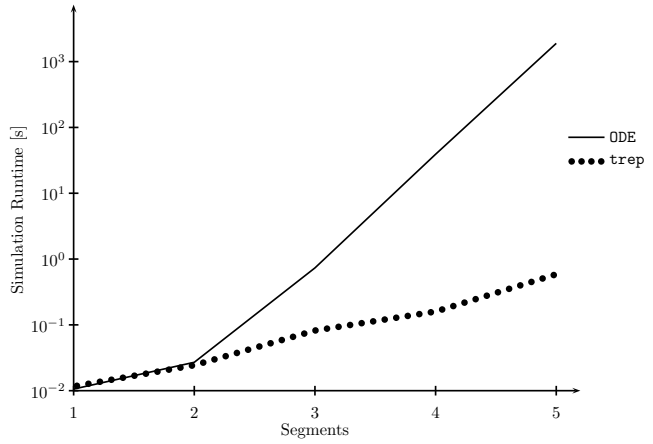


Fig. 7: Simulation runtime vs. Number of Segments in a scissor Lift. The simulation runtime was found by setting the time step so that the simulation time was $15s \pm 1\%$ when accumulated error exceeded 0.1. The y-axis is logarithmic.

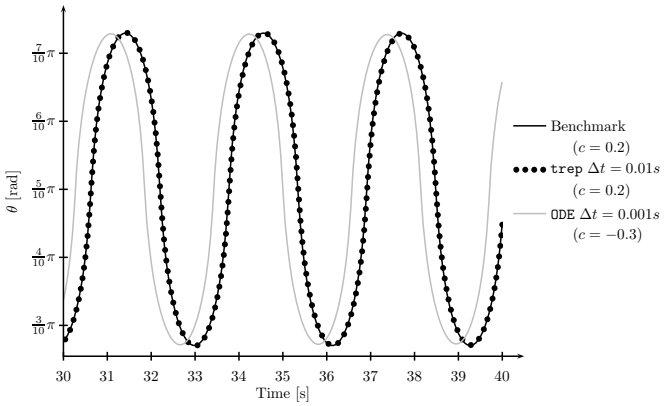


Fig. 8: The scissor lift is simulated with damping and a sinusoidal input force until it reaches a steady state periodic solution. `ODE` and `trep` were used to determine the scissor lift's damping constant from benchmark data. `trep` found the correct damping constant while `ODE` predicts an unstable system. The phase offset indicates differences between `ODE` and the benchmark during the transient response.

B. Complexity Analysis

Dynamics algorithms are also compared by complexity. That is, the number of operations needed to calculate $f(\cdot)$ in $\dot{x} = f(x, u)$. However, this is not always the relevant notion of complexity; it assumes that we are only interested in the complexity of evaluating a single step of an integrator and ignored other factors, like the quality of the solution and post-operations performed to fix constraints. As seen above in Fig. 6, results from the two methods can differ significantly in quality for a given time step. A meaningful comparison must take into account both the computational effort and the quality of the solution.

Our suggested approach is to compare the computational effort to generate a trajectory that is within a specified error of a benchmark trajectory. Given a benchmark trajectory $\theta_b(t)$ and a simulation result $\theta_s(t)$, the simulation error is defined as

$$e = \int_{t_0}^{t_f} \left(\theta_b(\tau) - \theta_s(\tau)\right)^2 \mathrm{d}\tau \tag{41}$$

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 7 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 time step must be reduced to maintain an acceptable error.

The results of this analysis are obviously 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.

C. System Identification

A common argument for the artificial energy loss seen with holonomic constraints in continuous systems is that all real systems have friction and damping, so energy conservation is irrelevant. There are two strong points against this argument. First, friction is not random noise in the dynamics; it affects the system in a structured way that we can and should model. Second, the rate of artificial energy loss will not equal the true energy loss. If more real loss is occurring, damping will still have to be added. If less real loss is occurring, energy *must be added to the system!*

To illustrate this point, consider a system identification experiment for the scissor lift. A linear friction model (i.e. $\tau = -c\dot{\theta}$) dissipates energy at each joint at the ends of the links. A sinusoidal input force acts on the sliding block to establish a non-trivial steady state solution.

Mathematica's `NDSolve[]` generates a benchmark solution from the one-dimensional Lagrangian model (augmented to include the dissipation and forcing). The damping constant for each model was varied until the model produced a steady state solution with an amplitude within 1% of the benchmark solution's amplitude.

The benchmark system was simulated with a damping constant of $c = 0.2$. `trep` used a time step of 0.01s and found the correct damping constant $c = 0.2$ with an amplitude error of -0.11%. `ODE` used a time step of 0.0013s[10] and found a damping constant of $c = -0.3$ with an amplitude error of -0.97%. The resulting trajectories are shown in Fig. 8.

[10] `ODE`'s time step was chosen so that the simulation took approximately as long `trep`

Note that the model found using `ODE` is *unstable* because of the artificial energy dissipation. The model is useless for further analysis.

It is a common misconception that variational integrators cannot model dissipative systems. This example emphasizes that variational integrators are able to handle both dissipation and forcing.

VI. 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 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 versatile robust trajectory exploration algorithms [11]. Generalized coordinates are essential for these tools 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.

`trep` is also missing some important components. Some, like collisions and impacts [27], have been studied and developed for variational integrators [9]. These algorithms just need to be adapted to work with the tree representation. Others, like non-holonomic constraints [7], have been largely ignored in discrete mechanics (with some exceptions [2]) 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 [21] that can be attached to coordinate frames.

[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. V-B.

VII. 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.

References

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

[2] 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.

[3] 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.

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

[5] R.W. Brockett, A. Stokes, and F. 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.

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

[7] 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.

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

[9] 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.

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

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

[12] E.R. Johnson and T.D. Murphey. Discrete and continuous mechanics for tree representations of mechanical systems. In *International Conference on Robotics and Automation*, 2008.

[13] 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.

[14] K.Yamane. *Simulating and Generating Motions of Human Figures*. Springer-Verlag, 2004.

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

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

[17] 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.

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

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

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

[21] 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.

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

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

[24] D. Schaechter, T. Kane, D. Levinson, and P. Mitiguy. Autolev, 2008. http://www.autolev.com/.

[25] R. Smith. Dynamics Simulation: A whirlwind tour (current state, and new frontiers), 2004. http://ode.org/slides/parc/dynamics.pdf.
[26] R. Smith. Open Dynamics Engine, 2008. http://www.ode.org.
[27] 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.
[28] M. West. Variational integrators. *California Institute of Technology Thesis*, 2004.
[29] M. West, C. Kane, J. E. Marsden, and M. Ortiz. Variational integrators, the Newmark scheme, and dissipative systems. 1999.
[30] A. Witkin, M. Gleicher, and W. Welch. Interactive dynamics. *ACM SIGGRAPH Computer Graphics*, 24(2):11–21, 1990.