Trajectory Optimization in Discrete Mechanics

Jarvis Schultz, Elliot Johnson, and Todd D. Murphey
Northwestern University
Evanston, IL, U.S.A.

Abstract Trajectory optimization involves both the optimization of inputs and the feedback regulation of the resulting trajectories. This article covers the main points of adapting trajectory optimization to the numerical routines in discrete mechanics. We start with a brief description of discrete mechanics and variational integrators and then move on to linearization, estimation, local projections in discrete function spaces, and iterative methods in trajectory optimization. Throughout, we discuss the results in the context of open-source software `trep` that implements all of the techniques for interconnected rigid body systems.

1 Introduction

Variational and energy-based techniques in numerical integration (Betsch, 2004; Betsch and Leyendecker, 2006; Lew et al., 2003, 2004; Marsen and West, 2001) have received a great deal of attention recently because of the long time horizon properties they often exhibit. However, the resulting discrete-time approximations of the trajectories are not particularly amenable to classical techniques in control—the discrete-time equations of motion are typically implicit, there is no classical notion of state, and local controllers are not readily computable. Instead, Discrete Mechanics and Optimal Control (DMOC) (Ober-Blöbaum et al., 2011; Leyendecker et al., 2010) is an approach that takes the discrete variational approximation and treats the implicit equations as constraints on a high dimensional optimization, for use in a numerical constrained optimization software package. The DMOC approach has the advantage of encoding all the mechanical information into the discrete-time constraints, but has the significant disadvantage of optimization becoming more challenging as the time resolution improves (i.e., for every factor of ten decrease in the time step, the dimension of the optimization increases by an exponent of ten). Moreover, the DMOC

approach does not provide feedback laws for stabilizing the optimized trajectories.

The work presented in this chapter provides a different approach to optimization of mechanical systems, based on trajectory optimization. All of the results are formulated directly in terms of the discrete function space, allowing one to use methods that are analogous to methods in infinite-dimensional function spaces. These lead to numerical methods that iteratively improve trajectories as well as providing feedback laws—both of which are critical to embedded systems relying on the resulting optimized trajectories. Significant content was pulled from Johnson et al. (2014); Schultz and Murphey (2013, 2014); Johnson (2012). More information can be obtained from these sources.

Lastly, this work relies heavily on a software package we have developed called `trep`. It is available at `http://nxr.northwestern.edu/trep` and it is recommended that the serious reader download `trep` and explore the examples presented throughout this chapter. `Trep` has functionality for taking an arbitrary tree-based description of a mechanical system (Johnson and Murphey, 2009) and computing a variational integrator, its continuous dynamics, its first and second-order linearizations, local optimal controllers, and nonlinear optimized trajectories. All computations presented in this chapter were performed in `trep`.

2 Variational Integrators and the Discrete Action Principle

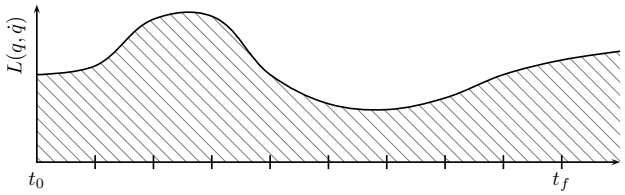
In this section we provide a brief overview of variational integrators and present the specific variational integrator used throughout this chapter. More detailed introductions and discussions can be found in Kharevych et al. (2006); West (2004); Lew (2003); Marsen and West (2001).

The idea behind variational integrators is to discretize the action with respect to time before finding the discrete-time equations of motion. Doing so leads to integration schemes that avoid common problems associated with numerically integrating a continuous differential equation. These problems can occur because the numerical approximations that are introduced do not respect fundamental mechanical properties like conservation of momentum, energy, and a symplectic form, all of which are relevant to mechanical systems (both forced and unforced).

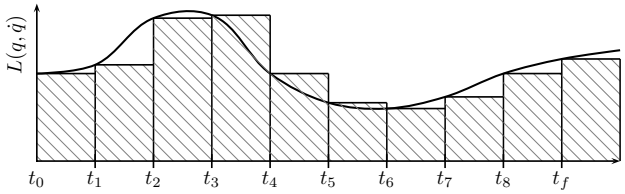
The continuous-time dynamics of a mechanical system are described by the Euler-Lagrange equations (Murray et al., 1994)

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = F(q, \dot{q}, u)$$

where q is the system's generalized coordinates, u represents the external inputs (e.g, motor torque), L is the Lagrangian (typically kinetic energy minus potential energy for finite-dimensional mechanical systems), and F is the forcing function that expresses external forces in the generalized coordinates.



(a) Continuous Action Integral



(b) Discrete Action Sum

Figure 1. The continuous Euler-Lagrange equation is derived by minimizing the action integral (a). The discrete Euler-Lagrange equation is derived by minimizing the approximating action sum (b).

The Euler-Lagrange equations can be derived from extremizing the action integral, typically referred to as the (least) action principle. The action integral—the integral of the Lagrangian with respect to time along an arbitrary curve in the tangent bundle—is illustrated as the shaded region in Fig. 1(a). The action principle stipulates that a mechanical system will follow the trajectory that extremizes the action with respect to variations in $q(t)$. Applying calculus of variations to the action integral shows that it is extremized by the Euler-Lagrange equation.

A variational integrator is derived by choosing a discrete Lagrangian, L_d that approximates the action over a discrete time step:

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

where q_k is a discrete-time configuration that approximates the trajectory (i.e. $q_k \approx q(t_k)$). This approximation can be achieved with any quadrature rule; more accurate approximations lead to more accurate integrators (Marsen and West, 2001). A concrete example of a discrete Lagrangian approximation is discussed in Sec. 2.1.

By summing the discrete Lagrangian over an arbitrary trajectory, the action integral is approximated by a discrete action, as shown in Fig. 2. The action principle is then applied to the action sum to find the *discrete trajectory* that extremizes the discrete action. The result of this calculation is the discrete Euler-Lagrange (DEL) equations:

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

where $D_n L_d$ is the *slot derivative*[1] of L_d.

The DEL equations depend on the previous, current, and future configuration (but they do not depend on the velocity, making this integrator an appealing representation of dynamics for embedded systems that measure configurations but not velocities). The DEL equation can also be written in an equivalent *position-momentum* form that only depends on the current and future time steps

$$p_k + D_1 L_d(q_k, q_{k+1}) = 0 \tag{1a}$$

$$p_{k+1} = D_2 L_d(q_k, q_{k+1}) \tag{1b}$$

where p_k is the discrete generalized momentum of the system at time k. (By these definitions, it should be clear that $-D_1 L_d(q_k, q_{k+1})$ and $D_2 L_d(q_k, q_{k+1})$ are both playing the role of a Legendre transform in discrete time, and are accordingly referred to as the left and right Legendre transforms, respectively.)

Equation (1) imposes a constraint on the current and future positions and momenta. Given an initial state p_k and q_k, (1a) is solved numerically to find the next configuration q_{k+1}. In general, (1a) is a non-linear equation that cannot be solved explicitly for q_{k+1}. In practice, the equation is solved using a numeric method such as the Newton-Raphson algorithm. The next momentum is then calculated explicitly by (1b). After an update, k is incremented and the process is repeated to simulate the system for as many time steps as desired.

[1] The slot derivative $D_n L(A_1, A_2, \dots)$ represents the derivative of the function L with respect to the n-th argument, A_n. In many cases, the arguments to the function L will be dropped for clarity and compactness. Hence, it is helpful to keep in mind that the slot derivative applies to the argument order provided in a function's definition.

Variational integrators can be extended to include non-conservative forcing (e.g., a motor torque or damping) by using a discrete form of the Lagrange-d'Alembert principle (Ober-Blöbaum et al., 2011). The continuous force is approximated by a left and right discrete force, F_d^- and F_d^+:

$$\int_{t_k}^{t_{k+1}} F(q(s), \dot{q}(s), u(s)) \cdot \delta q \mathrm{d}s \\ = F_d^-(q_k, q_{k+1}, u_k) \cdot \delta q_k + F_d^+(q_k, q_{k+1}, u_k) \cdot \delta q_{k+1}$$

where u_k is the discretization of the continuous force inputs: $u_k = u(t_k)$. As with the discrete Lagrangian, the discrete forcing can be approximated by any quadrature rule. A specific example is presented in Sec. 2.1.

For clarity, we use the following abbreviations for the discrete Lagrangian and discrete forces throughout this paper:

$$L_k = L_d(q_{k-1}, q_k) \\ F_k^{\pm} = F_d^{\pm}(q_{k-1}, q_k, u_{k-1}).$$

The forced DEL equations are then given by

$$p_k + D_1 L_{k+1} + F_{k+1}^- = 0 \tag{2a}$$
$$p_{k+1} = D_2 L_{k+1} + F_{k+1}^+. \tag{2b}$$

Again, (2) provides a way to calculate the configuration and momentum at the next time step from the current time step. Given the previous state (p_k and q_k) and the current input (u_k), the next configuration is found by implicitly solving (2a). The momentum at the next time step is then calculated explicitly by (2b).

In the following section, we provide an example of a variational integrator for a simple one dimensional system. We will use this example to help keep the calculations as concrete as possible during the development of the structured linearization results presented in Section 3.2.

2.1 Example: Pendulum

Consider the pendulum shown in Fig. 2 with $m = \ell = 1$. The pendulum has a single degree of freedom θ and is controlled by a torque u applied at the base.

The Lagrangian for the pendulum is

$$L(\theta, \dot{\theta}) = \tfrac{1}{2} m \ell^2 \dot{\theta}^2 + mg\ell \cos\theta = \tfrac{1}{2}\dot{\theta}^2 + g\cos\theta.$$

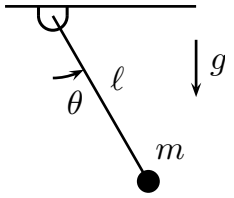


Figure 2. The pendulum is controlled by a torque at the pivot.

The generalized force due to the torque input is:

$$F(\theta, \dot{\theta}, u) = u.$$

The discrete Lagrangian is found by approximating the integral of the continuous-time Lagrangian over a short time interval Δt using the midpoint rule $\theta = \frac{\theta_k + \theta_{k+1}}{2}$ and $\dot{\theta} = \frac{\theta_{k+1} - \theta_k}{\Delta t}$:

$$\begin{aligned} L_d(\theta_k, \theta_{k+1}) &= L\left(\tfrac{\theta_k+\theta_{k+1}}{2}, \tfrac{\theta_{k+1}-\theta_k}{\Delta t}\right)\Delta t \\ &= \tfrac{(\theta_{k+1}-\theta_k)^2}{2\Delta t} + g\Delta t \cos\tfrac{\theta_{k+1}+\theta_k}{2}. \end{aligned}$$

The forcing is approximated with a combination of a midpoint and forward rectangle rule (though other choices of quadrature would be fine as well):

$$\begin{aligned} F_d^-(\theta_k, \theta_{k+1}, u_k) &= F(\tfrac{\theta_k+\theta_{k+1}}{2}, \tfrac{\theta_{k+1}-\theta_k}{\Delta t}, u_k)\Delta t = u_k \Delta t \\ F_d^+\theta_k, \theta_{k+1}, u_k &= 0. \end{aligned}$$

The first derivatives of L_d are needed to implement the variational integrator in (2):

$$D_1 L_d = -\tfrac{\theta_{k+1}-\theta_k}{\Delta t} - \tfrac{g\Delta t}{2}\sin\tfrac{\theta_{k+1}+\theta_k}{2} \tag{3}$$

$$D_2 L_d = \tfrac{\theta_{k+1}-\theta_k}{\Delta t} - \tfrac{g\Delta t}{2}\sin\tfrac{\theta_{k+1}+\theta_k}{2}. \tag{4}$$

The variational integrator update equations are found by substituting (3) into (2a) and (4) into (2b):

$$p_k - \tfrac{\theta_{k+1}-\theta_k}{\Delta t} - \tfrac{g\Delta t}{2}\sin\tfrac{\theta_{k+1}+\theta_k}{2} + u_k\Delta t = 0 \tag{5a}$$

$$p_{k+1} = \tfrac{\theta_{k+1}-\theta_k}{\Delta t} - \tfrac{g\Delta t}{2}\sin\tfrac{\theta_{k+1}+\theta_k}{2}. \tag{5b}$$

We choose initial conditions $p_k = 0.5$, $q_k = \theta_k = 0.2$, a time step of $\Delta t = 0.1s$, and an applied torque of $u_k = 0.8$. These values are substituted

in (5a), and a numeric root-finding algorithm finds the unknown θ_{k+1}. In this case, the Newton-Raphson method was used to find $\theta_{k+1} = 0.2471$. Finally, the updated discrete momentum is calculated using (5b): $p_{k+1} = 0.3627$. Note this example makes it clear that implicitly defined updates are to be expected, as mentioned earlier in Sec. 2. However, as we will see, these implicit updates have explicit linearizations that can be computed as functions of the configuration alone. In Sec. 3 we discuss a choice of state and quadratures that enable such a linearization.

2.2 Example: `trep` Pendulum

In this section, we present and discuss the Python code to implement the pendulum example from the previous section using `trep`. We begin with a few simple `import` statements, and by storing the parameters of the example as follows:

```
1  import numpy as np
2  import trep
3  import scipy.optimize as so
4
5  # set mass, length, and gravity:
6  m = 1.0; l = 1.0; g = 9.8;
7
8  # set state and step conditions:
9  pk = 0.5 # discrete generalized momentum
10 qk = 0.2 # theta config
11 uk = 0.8 # input torque
12 dt = 0.1 # timestep
```

Next we create an empty `trep` system and then define a list of frames to describe the system in accordance with `trep`'s tree structure (Johnson and Murphey, 2009), and the labels shown in Fig. 2. Then we add the frames to the system object. Finally, a gravity potential and torque are added to the system.

```
14 # create system
15 system = trep.System()
16 # define frames
17 frames = [
18         trep.rz("theta_1", name="PendAngle"), [
19             trep.ty(-l, name="PendMass", mass=m)]]
20 # add frames to system
21 system.import_frames(frames)
```

```
# add gravity potential
trep.potentials.Gravity(system, (0,-g,0))
# add a torque at the base
trep.forces.ConfigForce(system, "theta_1", "tau")
```

In the system description, there are a total of 3 frames. First, there is a `World` frame that is implicitly defined; its positive y-axis points up with respect to gravity. The world frame has a child frame, `PendAngle` that is parameterized by a degree-of-freedom, `theta_1`, specifying a rotation about the world's z-axis. Finally, the `PendAngle` frame has a child, `PendMass`, with a mass of m, and located by a constant translation of $-l$ in the `PendAngle` y-axis.

The next step is to create and initialize a variational integrator object as follows:

```
# create and initialize variational integrator
mvi = trep.MidpointVI(system)
mvi.initialize_from_state(0, np.array([qk]), np.array([pk]))
```

Now we solve the DEL equations by calling the variational integrator's `step` method as shown in the following:

```
# take single step with VI:
mvi.step(mvi.t1+dt, np.array([uk])) # args are t2, u1
```

Finally, we use the Python module `SciPy` to numerically solve (5a) and (5b) so that we can compare the `trep` results to the ones obtained in the previous section. Then we print and compare results.

```
# compare with manual computation results:
def DEL1(qkp1):
    return pk - (qkp1-qk)/dt - g*dt/2.*np.sin((qkp1+qk)/
                                              2.0) + uk*dt
# Implicitly solve DEL1 to get new config
qkp1 = so.newton(DEL1, qk)
# get new momentum
pkp1 = (qkp1-qk)/dt - g*dt/2.0*np.sin((qkp1+qk)/2.0)

# print results
print "============================================="
print "trep VI results:\tanalytical results:"
print "============================================="
```

```
print "qk+1 = ",mvi.q2[0],"\t","qk+1 = ",qkp1
print "pk+1 = ",mvi.p2[0],"\t","pk+1 = ",pkp1
print "================================================"
```

Running this complete script produces the following output:

```
================================================
trep VI results:        analytical results:
================================================
qk+1 =  0.247136194156  qk+1 =  0.247136194156
pk+1 =  0.362723883111  pk+1 =  0.362723883111
================================================
```

Thus `trep`'s solution and the analytical solution from the previous section are in agreement. The complete code for this example is provided with `trep`'s source code as the `examples/papers/cism2013/pend-single-step.py` file, or the file can be directly accessed at `http://git.io/trep-pend-step`.

2.3 Energy Behavior in Variational Integrators

Variational integrators have many desirable numerical properties such as exact constraint satisfaction and stable energy behavior. The stable energy behavior can be particularly useful in situations where one is interested in long simulation time-horizons, nearly conservative systems, or in taking larger timesteps. As a demonstration of the energy behavior, consider a simple harmonic oscillator with governing continuous differential equation given by

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

where x is the oscillator's position. We simulate this system for 1000 seconds with a second-order midpoint variational integrator, and with a fourth-order explicit Runge-Kutta integrator with an initial configuration $x(0) = 1$, an initial velocity $\dot{x}(0) = 0$, parameters $k = m = 1$, and a timestep of 0.01 seconds. Since this system is conservative, its energy should be constant for all time. It is clear from Fig. 3 that the Runge-Kutta scheme is artificially dissipating energy, unlike the variational integrator. Moreover, lower-order integrators (such as Euler) tend to be even worse, diverging from reasonable energy behavior much more quickly.

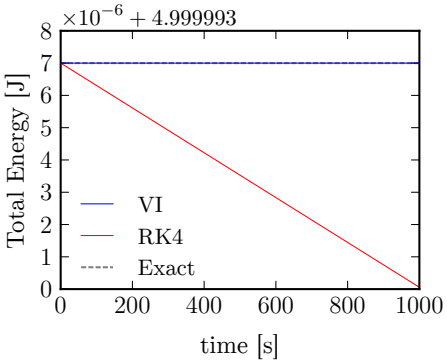


Figure 3. Energy behavior of a fourth-order Runge-Kutta method compared with a second-order variational integrator; the Runge-Kutta method is incorrectly dissipating energy.

3 One Step Maps and Linearizations

3.1 Choice of State

In continuous time, the configuration and velocity of a mechanical system are often concatenated into a single state $x = \begin{bmatrix} q & \dot{q} \end{bmatrix}$ to create a first-order representation of the system. This choice cannot be easily used for the variational integrator because the finite-difference approximation of the velocity involves configurations at different time steps. Instead, the one-step representation of the integrator (Hairer et al., 2004) in Eq. (2) suggests that for the variational integrator a convenient choice for the state is:

$$x_{k+1} = \begin{bmatrix} q_{k+1} \\ p_{k+1} \end{bmatrix} = f(x_k, u_k), \tag{6}$$

where the function $f(x_k, u_k)$ is implicitly defined by Eq. (2). However, the Implicit Function Theorem guarantees that such a function exists provided that the derivative

$$M_{k+1} = D_2 D_1 L_{k+1} + D_2 F^-_{k+1} \tag{7}$$

is non-singular at q_k, p_k, and u_k. For a discussion of situations where M_{k+1} is singular, see Johnson et al. (2014). This guarantee justifies abstracting the discrete dynamics of the variational integrator this way even though the underlying implementation still calculates the update q_{k+1} by numerically solving (2a). The purpose of this abstraction is to define the form for

the linearization of the discrete dynamics. In the next section, we derive this linearization and find that the derivatives of the abstract $f(x_k, u_k)$ representation are calculated explicitly.

3.2 Linearization of Discrete Trajectories

Analysis and optimal control methods often rely on the first-order linearization of system dynamics about a trajectory (Anderson and Moore, 1990). The first-order linearization of the discrete dynamics for the state-space form in Eq. (6) is:

$$\begin{aligned}\delta x_{k+1} &= \frac{\partial f}{\partial x_k}\delta x_k + \frac{\partial f}{\partial u_k}\delta u_k \\ \begin{bmatrix}\delta q_{k+1}\\ \delta p_{k+1}\end{bmatrix} &= \begin{bmatrix}\frac{\partial q_{k+1}}{\partial q_k} & \frac{\partial q_{k+1}}{\partial p_k}\\ \frac{\partial p_{k+1}}{\partial q_k} & \frac{\partial p_{k+1}}{\partial p_k}\end{bmatrix}\begin{bmatrix}\delta q_k\\ \delta p_k\end{bmatrix} + \begin{bmatrix}\frac{\partial q_{k+1}}{\partial u_k}\\ \frac{\partial p_{k+1}}{\partial u_k}\end{bmatrix}\delta u_k. \end{aligned} \tag{8}$$

Six components are required to calculate this linearization. These derivatives are found directly from the variational integrator equations (2), and all of them result in explicit equations.

Derivatives of q_{k+1} are found by implicitly differentiating (2a) and solving for the desired derivative. We start by finding $\frac{\partial q_{k+1}}{\partial q_k}$:

$$\begin{aligned}&\tfrac{\partial}{\partial q_k}\left[p_k + D_1L_{k+1} + F^-_{k+1} = 0\right]\\ &0 + D_1D_1L_{k+1} + D_2D_1L_{k+1}\tfrac{\partial q_{k+1}}{\partial q_k} + D_1F^-_{k+1}\\ &\qquad\qquad + D_2F^-_{k+1}\tfrac{\partial q_{k+1}}{\partial q_k} = 0\\ &\left[D_2D_1L_{k+1} + D_2F^-_{k+1}\right]\tfrac{\partial q_{k+1}}{\partial q_k} = -\left[D_1D_1L_{k+1} + D_1F^-_{k+1}\right]\\ &\tfrac{\partial q_{k+1}}{\partial q_k} = -M^{-1}_{k+1}\left[D_1D_1L_{k+1} + D_1F^-_{k+1}\right]\end{aligned} \tag{9}$$

where M_{k+1} is as defined by (7) and is assumed to be non-singular (otherwise the Implicit Function Theorem would not apply, making the state representation invalid).

The process is repeated to calculate $\frac{\partial q_{k+1}}{\partial p_k}$ and $\frac{\partial q_{k+1}}{\partial u_k}$:

$$\frac{\partial q_{k+1}}{\partial p_k} = -M^{-1}_{k+1} \tag{10}$$

$$\frac{\partial q_{k+1}}{\partial u_k} = -M^{-1}_{k+1}\cdot D_3F^-_{k+1}. \tag{11}$$

Notice that each of these derivatives depends on the new configuration q_{k+1} (e.g, $D_1D_1L_{k+1} = D_1D_1L_d\left(q_k, q_{k+1}\right)$). Before evaluating the derivatives, q_{k+1} must be found by solving (2a).

Derivatives of p_{k+1} are found directly by differentiating (2b):

$$\begin{aligned}\frac{\partial p_{k+1}}{\partial q_k} &= \left[D_2 D_2 L_{k+1} + D_2 F^+_{k+1}\right] \frac{\partial q_{k+1}}{\partial q_k} + \\ &\quad D_1 D_2 L_{k+1} + D_1 F^+_{k+1} \qquad (12)\\ \frac{\partial p_{k+1}}{\partial p_k} &= \left[D_2 D_2 L_{k+1} + D_2 F^+_{k+1}\right] \frac{\partial q_{k+1}}{\partial p_k} \qquad (13)\\ \frac{\partial p_{k+1}}{\partial u_k} &= \left[D_2 D_2 L_{k+1} + D_2 F^+_{k+1}\right] \frac{\partial q_{k+1}}{\partial u_k} + D_3 F^+_{k+1}. \qquad (14)\end{aligned}$$

These derivatives depend on (9)–(11), so (9)–(11) must be evaluated first. Once calculated, their values are used in (12)–(14) along with the known value of q_{k+1} to find the derivatives of p_{k+1}. Once all six derivatives are calculated, they are organized into the two matrices in (8) to get the complete first-order linearization about the current state. Lastly, note that the linearization is expressed entirely in terms of the discrete Lagrangian's dependence on the configuration and the discrete forcing function's dependence on the configuration and the continuous-time force. This is critical in understanding how to calculate the linearization without resorting to symbolic software.

3.3 Example: Pendulum *(cont.)*

We continue the pendulum example from 2.1 by calculating the first linearization (again, at the initial conditions $p_k = 0.5$, $q_k = \theta_k = 0.2$, with a time step of $\Delta t = 0.1s$, and an applied torque of $u_k = 0.8$). The derivatives of the discrete Lagrangian L_d are:

$$\begin{aligned}D_1 D_1 L_d &= \ \ \frac{1}{\Delta t} - \frac{g\Delta t}{4}\cos\frac{\theta_{k+1}+\theta_k}{2} = 9.7610\\ D_2 D_1 L_d &= -\frac{1}{\Delta t} - \frac{g\Delta t}{4}\cos\frac{\theta_{k+1}+\theta_k}{2} = -10.2389\\ D_1 D_2 L_d &= -\frac{1}{\Delta t} - \frac{g\Delta t}{4}\cos\frac{\theta_{k+1}+\theta_k}{2} = -10.2389\\ D_2 D_2 L_d &= \ \ \frac{1}{\Delta t} - \frac{g\Delta t}{4}\cos\frac{\theta_{k+1}+\theta_k}{2} = 9.7610.\end{aligned}$$

The derivatives of the discrete forcing are trivial:

$$\begin{aligned}D_1 F^-_d = D_2 F^-_d &= 0\\ D_3 F^-_d &= \Delta t.\end{aligned}$$

Using these values with (7), we find $M_{k+1}^{-1} = (-10.2389 + 0)^{-1} = -0.0976$. These are used with (9)–(11) to calculate the derivatives of q_{k+1}:

$$\begin{aligned}
\frac{\partial q_{k+1}}{\partial q_k} &= 0.0976 \cdot (9.7610 + 0) = 0.9533 \\
\frac{\partial q_{k+1}}{\partial p_k} &= 0.0976 \\
\frac{\partial q_{k+1}}{\partial u_k} &= 0.0976 \cdot 0.01 = 0.00976.
\end{aligned}$$

These values are part of the linearization, but are also required to calculate the derivatives of p_{k+1} from (12)–(14).

$$\begin{aligned}
\frac{\partial p_{k+1}}{\partial q_k} &= (9.7610 + 0) \cdot 0.9533 + -10.2389 + 0 = -0.9333 \\
\frac{\partial p_{k+1}}{\partial p_k} &= (9.7610 + 0) \cdot 0.0976 = 0.9533 \\
\frac{\partial p_{k+1}}{\partial u_k} &= (9.7610 + 0) \cdot 0.00976 + 0 = 0.09533
\end{aligned}$$

The six values define the entire first-order linearization:

$$\delta x_{k+1} = \begin{bmatrix} 0.9533 & 0.0976 \\ -0.9333 & 0.9533 \end{bmatrix} \delta x_k + \begin{bmatrix} 0.00976 \\ 0.09533 \end{bmatrix} \delta u_k.$$

The first-order linearization frequently appears in analysis applications. For example, we can examine the controllability matrix of the pendulum at this configuration to verify that it is linearly controllable:

$$\begin{gathered}
\mathcal{C} = \begin{bmatrix} B & AB \end{bmatrix} = \begin{bmatrix} 0.00976 & 0.0186 \\ 0.09533 & 0.0818 \end{bmatrix} \\
\operatorname{rank}(\mathcal{C}) = 2.
\end{gathered}$$

3.4 Example: `trep` Pendulum *(cont.)*

As in Sec. 2.2, we will repeat the example of the previous section, but this time using `trep` to perform all of the computations. The example script for this section shares the same first 34 lines as the code in Sec. 2.2. Thus, we assume that we already have a `trep` system representing the pendulum defined, and a variational integrator created, initialized, and have solved for a single time step. Thus we proceed onto calculating the derivatives of the discrete Lagrangian.

While `trep` itself often needs derivatives of the discrete Lagrangian, it only exposes methods for calculating derivatives of the continuous Lagrangian. However, `trep` provides all of the necessary continuous derivatives. This illustrates the flexibility of `trep`; by combining derivatives that

it does have, we can obtain other derivatives we may be interested in. Assume we want to calculate $D_1 D_1 L_d$; let's first look at the general form of a midpoint-rule variational integrator

$$L_d\left(q_k, q_{k+1}\right) = L\left(\frac{q_k + q_{k+1}}{2}, \frac{q_{k+1} - q_k}{\Delta t}\right)\Delta t. \tag{15}$$

Note that in this approximation, q is the midpoint value of q_k and q_{k+1} i.e. $q(q_k, q_{k+1}) = \frac{q_{k+1}+q_k}{2}$; similarly, $\dot{q}(q_k, q_{k+1}) = \frac{q_{k+1}-q_k}{\Delta t}$. Now, applying the chain rule to this equation, we can obtain a general expression for $D_1 D_1 L_d$ as

$$D_1 D_1 L_d(q_k, q_{k+1}) = \frac{\Delta t}{4}\frac{\partial^2 L}{\partial q \partial q} - \frac{1}{2}\frac{\partial^2 L}{\partial \dot{q} \partial q} - \frac{1}{2}\frac{\partial^2 L}{\partial q \partial \dot{q}} + \frac{1}{\Delta t}\frac{\partial^2 L}{\partial \dot{q} \partial \dot{q}}. \tag{16}$$

The expressions for $D_2 D_1 L_d$, $D_1 D_2 L_d$, and $D_2 D_2 L_d$ are obtained similarly.

With these expressions, the code for calculating the first two derivatives of the discrete Lagrangian is given by

```
34 # calc derivatives of discrete Lagrangian:
35 q = system.get_config("theta_1")
36 print "D1D1Ld = ",\
37     dt/4*system.L_dqdq(q,q) - \
38     1/2.*system.L_ddqdq(q,q) - \
39     1/2*system.L_ddqdq(q,q) + \
40     1/dt*system.L_ddqddq(q,q)
41 print "D2D1Ld = ",\
42   dt/4*system.L_dqdq(q,q) + \
43   1/2.*system.L_ddqdq(q,q) - \
44   1/2*system.L_ddqdq(q,q) - \
45   1/dt*system.L_ddqddq(q,q)
```

The code for calculating the derivatives of q_{k+1} and p_{k+1} is seen in the following:

```
47 # calc derivatives of qk+1
48 print "dqk+1/dqk = ",mvi.q2_dq1()[0][0]
49 print "dqk+1/dpk = ",mvi.q2_dp1()[0][0]
50 print "dqk+1/duk = ",mvi.q2_du1()[0][0]
51
52 # calc derivatives of pk+1
53 print "dpk+1/dqk = ",mvi.p2_dq1()[0][0]
54 print "dpk+1/dpk = ",mvi.p2_dp1()[0][0]
55 print "dpk+1/duk = ",mvi.p2_du1()[0][0]
```

To calculate the linearizations in Eq. (8), we could assemble matrices of the already calculated derivatives from above. However, to further explore the components of `trep`, we will instead use the `discopt` module. The `discopt` module contains convenient wrappers around the variational integrator components, discrete optimizations, linearizations, and optimal control tools. The code is as follows:

```
57 # calculate A and B using discopt module:
58 dsys = discopt.DSystem(mvi, np.array([0,dt]))
59 dsys.set(np.array([qk,pk]),np.array([uk]),0)
60 A = dsys.fdx()
61 B = dsys.fdu()
62 C = np.hstack((B, np.dot(A,B)))
63 print "A = \n",A
64 print "B = \n",B
65 print "C = \n",C
66 print "rank(C) = ",np.rank(C)
```

The output of the script matches the results of the previous section:

```
D1D1Ld =  9.7610974193
D2D1Ld =  -10.2389025807
dqk+1/dqk =  0.953334338555
dqk+1/dpk =  0.0976667169278
dqk+1/duk =  0.00976667169278
dpk+1/dqk =  -0.933313228895
dpk+1/dpk =  0.953334338555
dpk+1/duk =  0.0953334338555
A =
[[ 0.95333434  0.09766672]
 [-0.93331323  0.95333434]]
B =
[[ 0.00976667]
 [ 0.09533343]]
C =
[[ 0.00976667  0.01862181]
 [ 0.09533343  0.08176927]]
rank(C) =  2
```

The complete code for this example is provided with `trep`'s source code as the `examples/papers/cism2013/pend-linearization.py` file, or the file can be directly accessed at `http://git.io/trep-pend-lin`. Note that

the full script also contains examples of calculating second-order linearizations.

3.5 Constrained Systems

In this section, we discuss how the approach described in Sec. 3.1 and Sec. 3.2 to calculate the discrete linearizations extends to constrained variational integrators. We also present representative examples of second-order derivative calculations required for second order linearizations. Variational integrators are particularly well-suited to systems with holonomic constraints because the update equation for a constrained variational integrator explicitly incorporates the holonomic constraint. This is opposed to replacing the holonomic constraint with a locally equivalent acceleration constraints and then projecting the update onto the feasible set, a common approach in direct numeric integration of ordinary differential equations. Variational integrators enforce the holonomic constraint at every time step while still preserving the symplectic form and conserving momentum.

Variational integrators for constrained systems (Marsen and West, 2001) are derived using the same Lagrange-multiplier method used in the continuous case (Murray et al., 1994). Given a continuous-time constraint of the form $h(q) = 0$, the DEL equations for a forced, constrained variational integrator are:

$$p_k + D_1 L_{k+1} + F^-_{k+1} - Dh^T(q_k)\lambda_k = 0 \tag{17a}$$

$$h(q_{k+1}) = 0 \tag{17b}$$

$$p_{k+1} = D_2 L_{k+1} + F^+_{k+1} \tag{17c}$$

where λ_k are the Lagrange multipliers that can be interpreted as discrete-time constraint forces. In this case, given p_k and q_k, a numerical root-finding algorithm solves (17a) and (17b) to find q_{k+1} and λ_k. The updated momentum p_{k+1} is then explicitly calculated from (17c).

The Lagrange multipliers are completely determined by q_k, p_k, and u_k, so the state representation from Section 3.1 is unchanged. Accordingly, the same derivatives are needed to find the linearizations. Rather than derive every equation, we calculate one component of the first and second derivatives to demonstrate the process.

For the first-order linearization, we first find $\frac{\partial q_{k+1}}{\partial q_k}$. We start by differentiating (17a):

$$\begin{aligned} &\tfrac{\partial}{\partial q_k}\left[p_k + D_1 L_{k+1} + F^-_{k+1} - Dh^T(q_k)\lambda_k = 0\right] \\ \Rightarrow \quad &\tfrac{\partial q_{k+1}}{\partial q_k} = -M_k^{-1}\Big[C_{q_k} - Dh^T(q_k)\tfrac{\partial \lambda_k}{\partial q_k}\Big] \end{aligned} \tag{18}$$

where

$$C_{q_k} = D_1 D_1 L_{k+1} + D_1 F_{k+1}^- - D^2 h^T(q_k)\lambda_k.$$

To evaluate this derivative, we must calculate $\frac{\partial \lambda_k}{\partial q_k}$. This is found by differentiating (17b), substituting in (18), and solving for $\frac{\partial \lambda_k}{\partial q_k}$:

$$\begin{aligned}
\tfrac{\partial}{\partial q_k}\left[h(q_{k+1}) = 0\right] \\
Dh(q_{k+1})\tfrac{\partial q_{k+1}}{\partial q_k} = 0 \\
Dh(q_{k+1})M_k^{-1}\left[C_{q_k} - Dh^T(q_k)\tfrac{\partial \lambda_k}{\partial q_k}\right] = 0 \\
Dh(q_{k+1})M_k^{-1}C_{q_k} - Dh(q_{k+1})M_k^{-1}Dh^T(q_k)\tfrac{\partial \lambda_k}{\partial q_k} = 0 \\
\tfrac{\partial \lambda_k}{\partial q_k} = \left[Dh(q_{k+1})M_k^{-1}Dh^T(q_k)\right]^{-1} Dh(q_{k+1})M_k^{-1}C_{q_k}. \qquad (19)
\end{aligned}$$

To calculate $\frac{\partial q_{k+1}}{\partial q_k}$, the constrained DEL equation (17) is solved numerically to find q_{k+1} and λ_k. These values are used in (19) to find $\frac{\partial \lambda_k}{\partial q_k}$. Finally, $\frac{\partial q_{k+1}}{\partial q_k}$ is calculated with (18). The same approach is used to find the remaining components of the first derivative (Johnson et al., 2014), so we do not repeat the derivation here.

As an illustration of calculating necessary derivatives for a second order linearization, we continue by calculating $\frac{\partial^2 q_{k+1}}{\partial q_k \partial q_k}$.

$$\begin{aligned}
&\tfrac{\partial^2}{\partial q_k \partial q_k}\left[p_k + D_1 L_{k+1} + F_{k+1}^- - Dh^T(q_k)\lambda_k = 0\right] \\
\Rightarrow \quad &\tfrac{\partial^2 q_{k+1}}{\partial q_k \partial q_k} = -M_{k+1}^{-1}\left(C_{q_k q_k} - Dh^T(q_k)\tfrac{\partial^2 \lambda_k}{\partial q_k \partial q_k}\right) \qquad (20)
\end{aligned}$$

where

$$\begin{aligned}
C_{q_k q_k} =& D_1 D_1 D_1 L_{k+1} + D_1 D_1 F_{k+1}^- \\
&+ \Big[D_2 D_1 D_1 L_{k+1} + D_2 D_1 F_{k+1}^- \\
&\qquad + D_1 D_2 D_1 L_{k+1} + D_1 D_2 F_{k+1}^-\Big]\tfrac{\partial q_{k+1}}{\partial q_k} \\
&+ \left[D_2 D_2 D_1 L_{k+1} + D_2 D_2 F_{k+1}^-\right] \circ \left(\tfrac{\partial q_{k+1}}{\partial q_k}, \tfrac{\partial q_{k+1}}{\partial q_k}\right) \\
&- D^3 h^T(q_k)\lambda_k - 2D^2 h^T(q_k)\tfrac{\partial \lambda_k}{\partial q_k}.
\end{aligned}$$

In the above, the notation $M \circ (X, Y)$ represents a bilinear operator M operating on X and Y. We find the corresponding second derivative of λ_k by differentiating (17b) twice:

$$\frac{\partial^2}{\partial q_k \partial q_k}\left[h(q_{k+1}) = 0\right]$$
$$D^2h(q_{k+1}) \circ \left(\frac{\partial q_{k+1}}{\partial q_k}, \frac{\partial q_{k+1}}{\partial q_k}\right) + Dh(q_{k+1})\frac{\partial^2 q_{k+1}}{\partial q_k \partial q_k} = 0.$$

We substitute in (20) and solve for $\frac{\partial^2 \lambda_k}{\partial q_k \partial q_k}$:

$$\frac{\partial^2 \lambda_k}{\partial q_k \partial q_k} = \left[Dh(q_{k+1})M_{k+1}^{-1}Dh^T(q_k)\right] \cdot \left[Dh(q_{k+1})M_{k+1}^{-1}C_{q_k q_k} - D^2h(q_{k+1}) \circ \left(\frac{\partial q_{k+1}}{\partial q_k}, \frac{\partial q_{k+1}}{\partial q_k}\right)\right]. \quad (21)$$

To calculate this $\frac{\partial^2 q_{k+1}}{\partial q_k \partial q_k}$, we solve for the next state, calculate the first derivatives, evaluate (21) to find $\frac{\partial^2 \lambda_k}{\partial q_k \partial q_k}$, and finally evaluate (20) to find the second derivative. This same procedure is used to calculate the other components of the constrained second derivative. Further details of these calculations are available in Johnson et al. (2014).

Note that the constrained momentum update (17c) is identical to the unconstrained case (2b), so the constrained linearizations are identical to the unconstrained case.

3.6 Examples of Linearizations in Control and Estimation

In Sec. 3.2, we mentioned that in optimal control and estimation techniques often rely on being able to calculate linearizations about trajectories. In this section, we provide several examples of classic optimal control and estimation techniques utilizing variational integrators and their corresponding linearizations.

Example: LQR Control Here we use the Linear Quadratic Regulator (LQR) method to generate a stabilizing feedback controller for the mechanical marionette in Fig. 4. The marionette has 22 dynamic configuration variables, 18 kinematic configuration variables (Johnson and Murphey, 2007), and 6 holonomic constraints. The corresponding state-space model has 80 state and 18 input variables.

For a non-linear system like the marionette, the dynamics can be linearized about a known trajectory. The LQR solution for the linearization yields a feedback law that stabilizes the system near the known trajectory (subject to conditions on local controllability and observability (Anderson and Moore, 1990)).

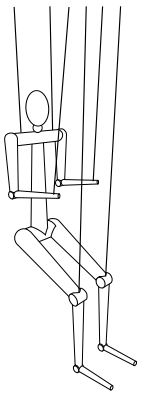


Figure 4. The marionette model has 40 configuration variables and 6 holonomic constraints.

The discrete LQR problem finds an optimal feedback law for a discrete linear system (Anderson and Moore, 1990):

$$z_{k+1} = A_k z_k + B_k \mu_k,$$

(where z_k is a perturbation to x_k). For the linearized dynamics about a non-linear system's trajectory, this corresponds to $A_k = \frac{\partial f(x_k, u_k)}{\partial x_k}$ and $B_k = \frac{\partial f(x_k, u_k)}{\partial u_k}$. The solution to the discrete LQR problem is found by solving the discrete Ricatti equation:

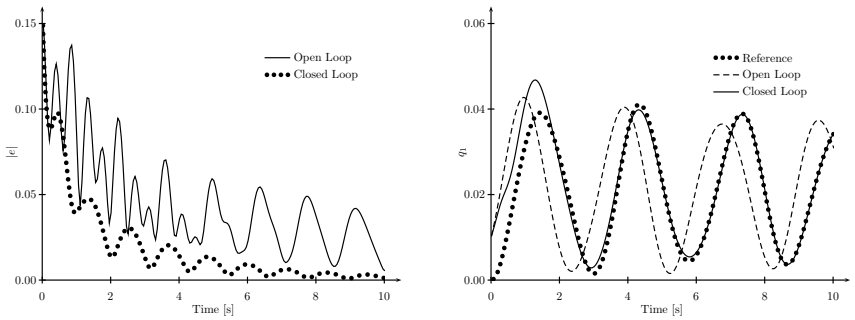
$$P_k = Q_k + A_k^T P_{k+1} A_k - A_k^T P_{k+1} B_k \left[R_k + B_k^T P_{k+1} B_k\right]^{-1} B_k^T P_{k+1} A_k \tag{22a}$$

$$P_{k_f} = Q_{k_f} \tag{22b}$$

where Q_k and R_k correspond to the cost of the linearized state and inputs, and Q_{k_f} determines the terminal cost of the linearized state. The Ricatti equation is solved to find P_k by recursively evaluating (22a) backwards in time from the boundary condition (22b). The solution is used to calculate a stabilizing feedback law:

$$\mathcal{K}_k = \left[R_k + B_k^T P_{k+1} B_k\right]^{-1} B_k^T P_{k+1} B_k.$$

The marionette was simulated and linearized about a 10.0 second trajectory using the midpoint variational integrator in `trep`. The reference trajectory was generated by changing the string lengths of the arms and legs



(a) (b)

Figure 5. (a)The discrete LQR feedback law significantly improves the norm of the error response of the marionette compared to the open-loop simulation. (b) The discrete LQR feedback law also significantly improves the individual error response of the marionette compared to the open-loop simulation. This is the trajectory of the configuration of the vertical orientation of the torso.

using $\pm 0.1 \sin(0.6\pi t)$ input signals. The linearization was used to create a locally stabilizing controller by solving the discrete LQR problem with identity for each cost matrix. A perturbation of 0.1 *rad* was then added to the initial condition of the vertical orientation of the torso and the simulation was performed with and without the added stabilizing feedback controller.

The norm of the resulting error between the perturbed and original trajectories is shown in Fig. 5(a) and the trajectory of the vertical orientation of the torso is shown in Fig. 5(b) as an example of stabilization of one of the states. The closed-loop trajectory converges significantly faster compared to the open-loop trajectory as expected. The ability to generate locally stabilizing feedback laws for complex systems that are simulated with variational integrators is a useful application of the methods described here. The source code for this example is distributed with `trep`'s source code in the file `examples/papers/cism2013/marionette.py`. The code can also be found at `http://git.io/trep-marionette-opt`.

The optimization was performed on an Intel i7-2760QM CPU at 2.40GHz. The simulation takes approximately 2.11 ms per step, the linearization takes approximately 1.12 ms per step. The second-order linearization takes approximately 22.15 ms per step, though it was not required for this example.

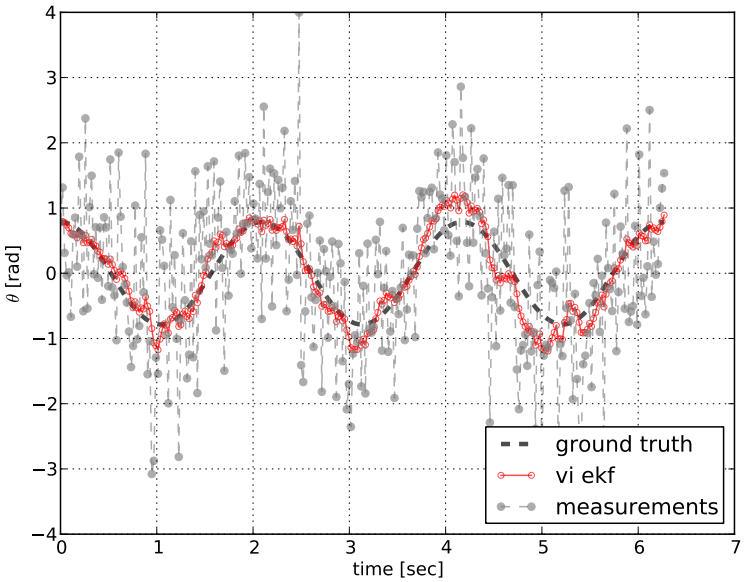


Figure 6. Performance with a variational integrator EKF applied to a free-swinging pendulum.

Example: Extended Kalman Filter In an Extended Kalman Filter (EKF), a nonlinear system is linearized about a state estimate to provide a local, linear approximation allowing the application of a standard Kalman filter. The one-step nature of Eq. (6) plus the linearizations of Sec. 3.2 allow the application of EKFs to nonlinear, variational integrator-based discrete systems. An illustration of this can be seen in Fig. 6. This plot was produced by applying an EKF to the same pendulum as in Sec. 2.2 and Sec. 3.3. The pendulum was simulated in a free swing to produce the ground-truth trajectory. Zero-mean Gaussian noise was added to the ground-truth trajectory for θ to provide simulated measurement values of configuration alone. EKF predictions were made by using the same DEL equations as in the previous sections, and the necessary linearizations were computed as in Sec. 3.2. Fig. 6 shows that the noise is appropriately filtered by the EKF. For more results on utilizing variational integrators in standard

estimation algorithms see Schultz and Murphey (2014, 2013).

4 Trajectory Optimization

This section discusses projection operator-based trajectory optimization for continuous and discrete dynamic systems. This strategy iteratively improves a known trajectory until a local minimum of the cost is found. The optimization works locally for systems with constraints and with underactuated systems.

Trajectory optimization using projection operators has been studied previously in the context of continuous-time systems (Hauser, 2002). It is discussed here for reference and comparison with the recently-formulated discrete time equivalent (Johnson, 2012). The presented algorithm applies to arbitrary discrete time dynamic systems, but is motivated by discrete time models based on variational integrators as described in previous sections.

The trajectory optimization problem is stated for both discrete and continuous systems in Sec. 4.1. The projection operator is introduced in Sec. 4.2 and some important properties of the discrete operator are presented in Sec. 4.3. Section 4.4 introduces the optimization algorithm. Each iteration of the algorithm can be broken into three steps that are each discussed separately in Sec. 4.4. The first step is calculating the projection operator, the second is calculating the descent direction, and the third is performing a line search. These three steps apply to both the continuous and discrete time domains, although the present focus is on discrete systems.

Results of several example applications are presented in Sec. 5. The first example, presented in Sec. 5.1, involves solving the optimal control to invert and stabilize a cart-pendulum system. Sec. 5.2 presents detailed `trep` code for solving a trajectory optimization problem for the same cart-pendulum, but a simpler reference is chosen to reduce the amount of code required. Finally, Sec. 5.3 solves the optimal control problem for the high degree-of-freedom marionette discussed in Section 3.6.

4.1 Problem Statements

We seek trajectories for a continuous or discrete dynamic system that minimize an appropriately differentiable cost function. Let $\mathcal{T}$ be the set of dynamically admissible trajectories for a system. The trajectory space is embedded in a inner product space V so that $\mathcal{T} \subseteq V$. In continuous time, $\mathcal{T} = \{\xi = (x, u) \in V : \dot{x}(t) = f(x(t), u(t), t)\}$. In discrete time, $\mathcal{T} = \{\xi = (x, u) \in V : x(k+1) = f(x(k), u(k), k)\}$.

Throughout this section, $\xi = (x, u)$ is always an element of $\mathcal{T}$ and $\delta\xi =$

$(\delta x, \delta u)$ is always an element of the tangent trajectory space $T\mathcal{T}$. Elements of V use the symbols $\bar{\xi} = (\bar{x}, \bar{u})$ while elements in the tangent space TV use the notation $\delta\bar{\xi} = (\delta\bar{x}, \delta\bar{u})$. (Elements $\bar{\xi} \in V$ will be referred to as trajectories even though they may not satisfy the system dynamics.)

The discrete trajectory space is formally finite-dimensional. However, in this derivation it is treated as arbtirary-dimensional like the continuous trajectory space. This approach leads to an algorithm that explicitly takes advantage of the discrete dynamics, and avoids directly optimizing over the entire (potentially very large) dimensionality of the discrete space. For the remainder of the chapter, we refer to both spaces simply as function spaces.

Continuous Problem Statement: Given an initial trajectory $\xi_0 = (x, u) \in \mathcal{T}$, find

$$\xi^* = \arg\min_{\xi \in \mathcal{T}} h(\xi)$$
$$\text{where } h(\xi) = \int_{t_0}^{t_f} \ell(x(t), u(t), t)\mathrm{d}t + m(x(t_f)).$$

Discrete Problem Statement: Given an initial trajectory $\xi_0 = (x, u) \in \mathcal{T}$, find

$$\xi^* = \arg\min_{\xi \in \mathcal{T}} h(\xi)$$
$$\text{where } h(\xi) = \sum_{k=k_0}^{k_f-1} \ell(x(k), u(k), k) + m(x(k_f)).$$

In both cases, the initial condition $x(0)$ of the trajectory may be included in the optimization or considered fixed.

For example, suppose we want the system to track a desired state path x_d as closely as possible. We can treat this as an optimization problem with the costs

$$\begin{aligned} \ell(x, u, k) &= (x - x_d(k))^T Q (x - x_d(k)) + u^T R u \\ m(x) &= (x - x_d(k_f))^T Q (x - x_d(k_f)) \end{aligned} \tag{23}$$

where Q and R are positive definite matrices. By solving this optimization, we obtain a trajectory that approximates the desired path even if the path is not dynamically feasible or the system is underactuated.

These are *constrained* optimizations because solutions must satisfy the dynamics of the system. The approach described here introduces a projection operator that takes arbitrary curves in V and projects them to nearby trajectories in $\mathcal{T}$. The projection operator effectively removes the dynamics

constraint, at least locally, at the cost of needing to include the projection operator in the unconstrained optimization. The unconstrained optimization is then solved using an iterative approach that improves the cost at each iteration by calculating a descent direction, and by performing a one dimensional line search along that direction.

The next section introduces projection operators and their properties.

4.2 Projection Operators

The optimization algorithm is based on a projection operator $\xi = \mathcal{P}(\bar{\xi})$ that maps curves $\bar{\xi} \in V$ to nearby trajectories of the system $\xi \in \mathcal{T}$. The implementation details of discrete projection operators are discussed in Sec. 4.3. We begin by introducing some common properties of both continuous and discrete projection operators.

The continuous and discrete projection operators are both created by integrating the system dynamics while tracking the $\bar{\xi}$ with a linear feedback law. To be projections, both operators must have the property that if $\xi \in \mathcal{T}$, then $\mathcal{P}(\xi) = \xi$. A consequence of this is that the projections are *idempotent* i.e.

$$\mathcal{P}(\mathcal{P}(\xi)) = \mathcal{P}(\xi). \tag{24}$$

Both projection operators are twice differentiable[2]. The first derivative $\delta\xi = D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi}$ is also a projection:

$$D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi} = D\mathcal{P}(\bar{\xi}) \circ D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi}. \tag{25}$$

The derivative maps elements of $\delta\bar{\xi} \in T_{\bar{\xi}}V$ to tangent trajectories $\delta\xi \in T_{\mathcal{P}(\bar{\xi})}\mathcal{T}$. The derivative has the useful invariance property that it is the same whether evaluated at $\bar{\xi}$ or the projected trajectory $\mathcal{P}(\bar{\xi})$:

$$D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi} = D\mathcal{P}(\mathcal{P}(\bar{\xi})) \circ \delta\bar{\xi}. \tag{26}$$

These properties are important for calculating the descent direction in Sec. 4.4.

[2]We presume that the system dynamics are twice differentiable.

4.3 Discrete Projection Operators

The projection operator for a discrete dynamic system $x(k+1) = f(x(k), u(k), k)$ is given by

$$\begin{aligned}
\xi = (x, u) =& \mathcal{P}\left(\bar{\xi} = (\bar{x}, \bar{u})\right) : \\
& x(k_0) = \bar{x}(k_0) && \text{(27a)} \\
& x(k+1) = f(x(k), u(k), k) && \text{(27b)} \\
& u(k) = \bar{u}(k) - \mathcal{K}(k)(x(k) - \bar{x}(k)) && \text{(27c)}
\end{aligned}$$

where $\mathcal{K}(k)$ is a stabilizing feedback law for the dynamic system $f(x, u, k)$. $\mathcal{K}(k)$ is typically found by solving the discrete LQR (Anderson and Moore, 1990; Locatelli, 2001) problem for the linearization of the non-linear system about the current trajectory. As mentioned in Sec. 3.6 the LQR problem for a nonlinear system linearized about a particular trajectory provides a stabilizing feedback about the trajectory.

Proposition 1. The discrete projection operator defined by (27) satisfies the idempotent projection property (24).

Proof. Let $(x_1, u_1) = \mathcal{P}(\bar{x}, \bar{u})$, then:

k	$x_1(k)$	$u_1(k)$
0	$\bar{x}(0)$	$\bar{u}(0) + \mathcal{K}(0)(x_1(0) - \bar{x}(0)) = \bar{u}(0)$
1	$f(\bar{x}(0), \bar{u}(0))$	$\bar{u}(1) + \mathcal{K}(1)(x_1(1) - \bar{x}(1))$
2	$f(x_1(1), u_1(1))$	$\bar{u}(2) + \mathcal{K}(2)(x_1(2) - \bar{x}(2))$
3	$f(x_1(2), u_1(2))$	$\bar{u}(3) + \mathcal{K}(3)(x_1(3) - \bar{x}(3))$
	$\vdots$	

Applying the projection again, let $(x_2, u_2) = \mathcal{P}(x_1, u_1)$:

k	$x_2(k)$	$u_2(k)$
0	$x_1(0) = \bar{x}(0)$	$u_1(0) + \mathcal{K}(0)(x_2(0) - x_1(0)) = u_1(0)$
1	$f(\bar{x}(0), \bar{u}(0)) = x_1(1)$	$u_1(1) + \mathcal{K}(1)(x_2(1) - x_1(1)) = u_1(1)$
2	$f(x_1(1), u_1(1)) = x_1(2)$	$u_1(2) + \mathcal{K}(2)(x_2(2) - x_1(2)) = u_1(2)$
3	$f(x_1(2), u_2(2)) = x_1(3)$	$u_1(3) + \mathcal{K}(3)(x_2(3) - x_1(3)) = u_1(3)$
	$\vdots$	

By induction, the trajectories are equal. □

First Derivative of $\mathcal{P}(\bar{\xi})$ The derivative of the projection operator $\delta\xi = D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi}$ is found by differentiating (27) in the direction $\delta\bar{\xi} = (\delta\bar{x}, \delta\bar{u})$:

$$\begin{aligned}
\delta\xi = (\delta x, \delta u) =& D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi} : \\
\xi &= \mathcal{P}(\bar{\xi}) && \text{(28a)} \\
\delta x(k_0) &= \delta\bar{x}(k_0) && \text{(28b)} \\
\delta x(k+1) &= \frac{\partial f}{\partial x}(k)\delta x(k) + \frac{\partial f}{\partial u}(k)\delta u(k) && \text{(28c)} \\
&= Df(k) \circ \delta\xi(k) \\
\delta u(k) &= \delta\bar{u}(k) - \mathcal{K}(k)(\delta x(k) - \delta\bar{x}(k)) && \text{(28d)}
\end{aligned}$$

where the notation $\frac{\partial f}{\partial x}(k)$ represents $\frac{\partial f}{\partial x}(x(k), u(k), k)$, and also applies to $\frac{\partial f}{\partial u}(k)$ and $Df(k)$. Note that $\mathcal{P}$ is differentiable if f is differentiable.

The derivative of the discrete projection operator is a projection itself that maps $\bar{\xi}$ to trajectories of the discrete linear system $\delta x(k+1) = A(k)\delta x(k) + B(k)\delta u(k)$ with $A(k) = \frac{\partial f}{\partial x}(k)$ and $B(k) = \frac{\partial f}{\partial u}(k)$. Therefore, the discrete $D\mathcal{P}(\bar{\xi}) \circ \delta\bar{\xi}$ satisfies the idempotent projection property (25).

The invariance property (26) follows directly from (28a).

There are limitations to the projection operators defined here. In general, it is difficult to find globally stabilizing linear feedback laws for nonlinear systems. We typically settle for a locally stabilizing controller about a known trajectory ξ_0 and work with curves $\bar{\xi}$ sufficiently close to ξ_0. This is well suited for optimization problems as each iteration improves a known trajectory by only a small perturbation.

Although the details of the continuous projection operators were not discussed here, both continuous and discrete projection operators have identical properties and are represented by identical notation. They differ only in implementation details. The next section takes advantage of the similarities and identical notation to describe the overall optimization algorithm for both the continuous and discrete domains simultaneously.

4.4 Line Searches and Iterative Methods in Optimization

The trajectory optimization problem described in Sec. 4.1 is constrained to solutions that satisfy the system dynamics. This constraint is problematic for an iterative gradient descent algorithm. Even if the current iterate is a valid trajectory $\xi_i \in \mathcal{T}$, and the descent direction is guaranteed to be in the tangent space of ξ_i: $\delta\xi \in T_{\xi_i}\mathcal{T}$, linear combinations of the two $\xi_i + \delta\xi_i$ will not, in general, be an admissible trajectory because the trajectory manifold for the nonlinear system is not a vector space. The projection operator

provides a means to remove the constraint, allowing the optimization to take place in the unconstrained inner product space V by defining a new cost $g(\bar{\xi}) = h(\mathcal{P}(\bar{\xi}))$ to optimize.

Proposition 2. The optimizations $\min_{\bar{\xi}} g(\bar{\xi})$ and $\min_{\xi \in \mathcal{T}} h(\xi)$ are equivalent in the sense that if $\bar{\xi}^*$ minimizes $g(\cdot)$, then $\xi^* = \mathcal{P}(\bar{\xi}^*)$ minimizes $h(\cdot)$, and if ξ^* minimizes $h(\cdot)$, then ξ^* also minimizes $g(\cdot)$

Proof. Suppose ξ^* minimizes $h(\cdot)$ but not $g(\cdot)$. Then there exists $\delta\bar{\xi} \in V$ such that:

$$Dg(\xi^*) \circ \delta\bar{\xi} \neq 0$$
$$Dh(\xi^*) \circ D\mathcal{P}(\xi^*) \circ \delta\bar{\xi} \neq 0.$$

Letting $\delta\xi = D\mathcal{P}(\xi^*) \circ \delta\bar{\xi} \in T_{\xi^*}\mathcal{T}$:

$$Dh(\xi^*) \circ \delta\xi \neq 0,$$

but this contradicts ξ^* being a minimizer of $h(\cdot)$.

On the other hand, suppose $\bar{\xi}^*$ minimizes $g(\cdot)$ but $\xi = \mathcal{P}(\bar{\xi}^*)$ does not minimize $h(\cdot)$, then there exists $\delta\xi \in T_{\xi^*}\mathcal{T}$ such that:

$$Dh(\xi) \circ \delta\xi \neq 0$$
$$Dh(\mathcal{P}(\bar{\xi}^*)) \circ D\mathcal{P}(\mathcal{P}(\bar{\xi}^*)) \circ \delta\xi \neq 0$$
$$Dg(\bar{\xi}^*) \circ \delta\xi \neq 0,$$

but this contradicts $\bar{\xi}^*$ being a minimizer of $g(\cdot)$. □

This unconstrained optimization can be solved using iterative numeric methods. The overall algorithm is shown in Alg. 1. Each iteration involves designing a projection operator, choosing a descent direction, and performing a search along the descent direction. These steps are described in detail in the following subsections.

Creating the Projection Operator Both continuous and discrete projection operators rely on a stabilizing linear feedback law. In the optimization algorithm, each iteration starts with a known trajectory and searches for new nearby trajectories along a descent direction calculated at the current iterate. This local nature allows us to work with projection operators that are only locally stabilizing rather than requiring global stability. A updated stabilizing feedback law is required for each new trajectory.

Algorithm 1 Trajectory Optimization

Require: $\xi_0 \in \mathcal{T}$
Ensure: $\xi_i = \arg\min_{\xi \in \mathcal{T}} h(\xi)$
1: $i \leftarrow 0$
2: **loop**
3: $\quad \mathcal{K} \leftarrow CreateProjection(\xi_i)$
4: $\quad \delta\xi \leftarrow FindDescentDirection(\xi_i, \mathcal{K})$
5: $\quad$ **if** $|\delta\xi| \approx 0$ **then**
6: $\quad\quad$ **return** ξ_i
7: $\quad$ **end if**
8: $\quad \lambda \leftarrow PerformLineSearch(\xi_i, \delta\xi)$
9: $\quad \xi_{i+1} \leftarrow \mathcal{P}(\xi_i + \lambda\delta\xi)$
10: **end loop**

While any algorithm that provides a stabilizing linear feedback law will work, the LQR optimal control problem is a convenient method because it only requires data already available from simulation. Specifically, it only requires linearizations of the dynamics, which are already necessary for the trajectory optimization, and it handles arbitrary trajectories so long as they are differentiable. The choice of the cost matrices Q and R in the LQR problem can have a significant impact on performance. Experience has shown that the choice $Q = \mathcal{I}$, $R = \mathcal{I}$ generally provides acceptable feedback laws.

Finding the Descent Direction In gradient-descent optimization, the descent direction is calculated by minimizing a quadratic approximation of the cost near each iterate. For finite-dimensional problems, the resulting descent direction is $z = -M^{-1}Dg(x_i)$, where M is the quadratic term of the approximation. Common choices are $M = \mathcal{I}$, which leads to the steepest descent algorithm, and $M = D^2g(x_i)$, which leads to Newton's method (Kelley, 1999). This algorithm is simple, but it relies on a matrix representation of the model M.

In trajectory optimization, the vector space is a function space, so no matrix representation exists. Instead, the approximation of the cost is minimized directly at each iteration. This section develops the quadratic approximation of the cost, and shows that the required minimization is equivalent to a Linear Quadratic (LQ) optimization problem for the linearization of the system about the current trajectory.

The cost is approximated near the current trajectory by the Taylor series:

$$g(\xi_i + \delta\bar{\xi}) \approx g(\xi_i) + Dg(\xi_i) \circ \delta\bar{\xi} + \tfrac{1}{2} D^2 g(\xi_i) \circ (\delta\bar{\xi}, \delta\bar{\xi})$$

$$\begin{aligned} = h(\mathcal{P}(\xi_i)) + Dh(\mathcal{P}(\xi_i)) \circ D\mathcal{P}(\xi_i) \circ \delta\bar{\xi} + \tfrac{1}{2} Dh(\mathcal{P}(\xi_i)) \circ D^2\mathcal{P}(\xi_i) \circ (\delta\bar{\xi}, \delta\bar{\xi}) \\ + \tfrac{1}{2} D^2 h(\mathcal{P}(\xi_i)) \circ (D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}, D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}) \end{aligned}$$

where ξ_i is the current trajectory and $\bar{\xi}$ is a small perturbation from the trajectory.

The curve ξ_i is guaranteed to be an admissible trajectory (i.e, $\xi_i \in \mathcal{T}$) by the initial condition of the algorithm and the projection step at the end of each iteration (Line 9 in Alg. 1). Applying the identity $\xi = \mathcal{P}(\xi)\ \forall\ \xi \in \mathcal{T}$, and applying the invariance property (26) to $D^2\mathcal{P}(\cdot)$, the model becomes

$$\begin{aligned} = h(\xi_i) + Dh(\xi_i) \circ D\mathcal{P}(\xi_i) \circ \delta\bar{\xi} + \tfrac{1}{2} Dh(\xi_i) \circ D^2\mathcal{P}(\xi_i) \circ (D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}, D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}) \\ + \tfrac{1}{2} D^2 h(\xi_i) \circ (D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}, D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}) \end{aligned}$$

$$= h(\xi_i) + Dh(\xi_i) \circ D\mathcal{P}(\xi_i) \circ \delta\bar{\xi} + \tfrac{1}{2} q(\xi) \circ (D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}, D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}) \quad (29)$$

where the second order terms have been collected into the *quadratic model*:

$$q(\xi) \circ (\delta\xi, \delta\xi) = Dh(\xi_i) \circ D^2\mathcal{P}(\xi_i) \circ (\delta\xi, \delta\xi) + D^2 h(\xi_i) \circ (\delta\xi, \delta\xi). \quad (30)$$

The descent direction is found by minimizing (29) over all directions in TV:

$$\delta\bar{\xi}^* = \underset{\delta\bar{\xi}}{\operatorname{argmin}}\, h(\xi_i) + Dh(\xi_i) \circ D\mathcal{P}(\xi_i) \circ \delta\bar{\xi} + \tfrac{1}{2} q(\xi) \circ (D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}, D\mathcal{P}(\xi_i) \circ \delta\bar{\xi}).$$

Noting that $\delta\bar{\xi}$ is always projected into $T_{\xi_i}\mathcal{T}$ by $D\mathcal{P}(\xi_i)$ in this unconstrained optimization, the search is restricted to descent directions in this space:

$$\delta\xi^* = \underset{\delta\xi \in T_{\xi_i}\mathcal{T}}{\operatorname{argmin}}\, 2Dh(\xi_i) \circ \delta\xi + q(\xi) \circ (\delta\xi, \delta\xi) \quad (31)$$

where we have dropped the constant term $h(\xi_i)$ and multiplied the cost by 2 without affecting the optimal direction.

In this case, the constrained optimization problem (31) is easier to solve than the unconstrained case because every element $\delta\xi \in T_{\xi_i}\mathcal{T}$ is a trajectory of a (continuous or discrete) linear system. It is a linear optimal control problem with a cost comprising a linear and quadratic term, better known as a Linear Quadratic (LQ) problem.

Solutions to the LQ problem are found by solving a set of (continuous or discrete) Ricatti equations. The solution provides an affine control law $\delta u = -K\delta x + C$ where K and C are (continuous or discrete) time-varying values found from the Ricatti equations. Thus, (31) is solved by finding the optimal feedback law from the LQ solution, choosing an initial condition for the descent direction, and integrating the system forward with the feedback input. The trajectory found by the forward simulation is the descent direction that locally minimizes the cost approximation. Details of solving (31) for the discrete case are discussed in Sec. 4.5.

In order for (31) to have a well defined minimum, the quadratic model must be positive definite:

$$q(\xi) \circ (\delta\xi, \delta\xi) > 0 \quad \forall\ \delta\xi \in T_\xi \mathcal{T}.$$

The quadratic model (30) is not guaranteed to be positive definite. Additionally, even if it is positive definite, it is a well-known result in optimization that descent directions calculated from the second derivative often perform poorly when the current iteration is not near the optimal solution (Kelley, 1999). The quadratic model q is therefore generalized, as in finite dimensional optimization, to an arbitrary bilinear model to avoid these problems. The specific form of this bilinear model depends on the time domain. The discrete form for one possible model is discussed in Sec. 4.5.

The completed algorithm for calculating the descent direction is shown in Alg. 2. The following section discusses the initial conditions for the descent direction.

Algorithm 2 Calculating the Descent Direction

Require: $\xi_i \in \mathcal{T}$ and $q(\xi) \circ (\delta\xi, \delta\xi) > 0$

1: $\mathcal{K}, C \leftarrow SolveLQProblem()$
2: $\delta x_0 = ChooseInitialCondition()$
3: $\delta\xi = SimulateLinearizedSystem(dx_0, \delta u = -K\delta x - C)$
4: **return** $\delta\xi$

If we take $\delta\xi$ and use it in $g(\xi + \lambda\delta\xi)$, then there should be some λ sufficiently small that $g(\xi + \lambda\delta\xi) < g(\xi)$ (because $\delta\xi$ is a local descent direction). But how do we ensure that iterating on this process leads to a sequence that converges to an optimizer? This is the subject of the next section.

Performing the Line Search The line search is required in iterative optimization to guarantee a sufficient decrease in the cost of each new iterate (Kelley, 1995). In trajectory optimization, the line search plays an

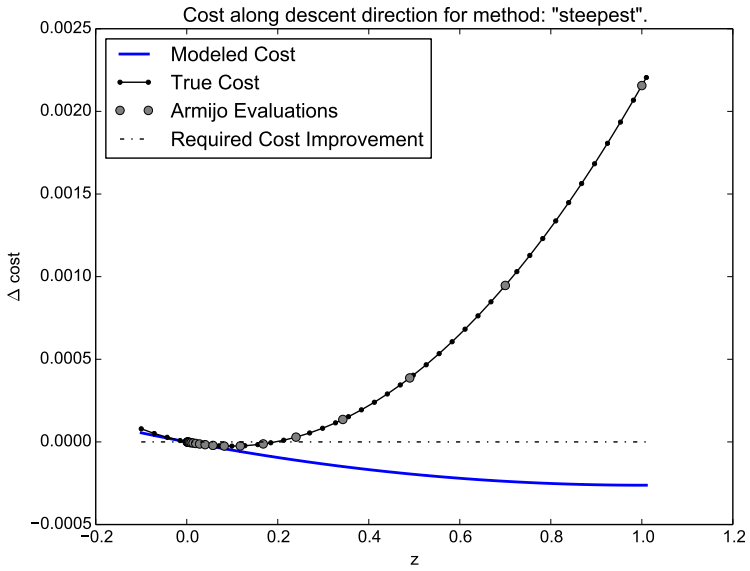


Figure 7. If one starts with a large value of λ and systematically decreases it until $g(\xi+\lambda\delta\xi) < g(\xi)+\alpha\lambda Dg(\xi)\circ\delta\xi$, an iterative optimization algorithm is guaranteed to converge. This figure was generated using the `trep` method `trep.discopt.DOptimizer.descent_plot`.

additional role by reducing the size of the descent step until the new trajectory is in the stabilizable subspace of the current projection operator. Otherwise, the line search is identical to the finite dimensional case, and any number of existing line search algorithms can be used.

A modified Armijo line search (Armijo, 1966) is shown in Alg. 3. The original Armijo requirement simply states that for a choice of α, there is a choice of λ such that $g(\xi + \lambda\delta\xi) < g(\xi) + \alpha\lambda Dg(\xi) \circ \delta\xi$, and that satisfying this choice at every iteration guarantees convergence. This can visualized in the Fig. 7. Note the additional check on line 5 to guarantee that the perturbed trajectory is successfully stabilized by the projection operator. There are several indications that a trajectory is not successfully stabilized including state variables, Ricatti variables, or control gains tending towards infinity. With an implicit dynamics update provided by a variational integrator an unsuccessful projection usually results in failure of the root-finding algorithm for solving the DEL equations.

Algorithm 3 Armijo Line Search

Require: $\xi \in \mathcal{T}$, $\delta\xi \in T_\xi\mathcal{T}$, $\alpha \in [0,1]$ and $\beta \in (0,1)$
Ensure: $g(\xi + \lambda\delta\xi) < g(\xi) + \alpha\lambda Dg(\xi) \circ \delta\xi$

```
m ← 0
loop
    λ = β^m
    ξ_n = P(ξ + λδξ)
    if the projection stabilized then
        if g(ξ + λδξ) < h(ξ) + αλDg(ξ) ∘ δξ then
            return λ
        end if
    end if
    m ← m + 1
end loop
```

4.5 LQ problems and Finding Descent Directions

Model for Steepest Descent Defining $q(\xi) \circ (\delta\xi, \delta\xi) = \langle \delta\xi, \delta\xi \rangle$, the descent direction optimization (31) becomes

$$\begin{aligned}\delta\xi^* &= \arg\min_{\delta\xi \in T_{\xi_i}\mathcal{T}} 2Dh(\xi) \circ \delta\xi + \langle \delta\xi, \delta\xi \rangle \\ &= \arg\min_{\delta\xi \in T_{\xi_i}\mathcal{T}} 2Dh(\xi) \circ \delta\xi + ||\delta\xi||^2.\end{aligned}$$

The quadratic term of this cost depends solely on the magnitude of $\bar{\xi}$, so the optimization finds the direction that minimizes the first-order linear term. This is the steepest descent method. It generally has slower convergence rates than Newton's method near the optimum (linear vs. quadratic convergence), but often performs better far from the optimizer and is less costly to compute.

Descent Direction Computation The generalized quadratic term in discrete time is:

$$q(\xi) \circ (\delta\xi, \delta\xi) = \left(\sum_{k=k_0}^{k_f-1} \begin{bmatrix} \delta x(k) \\ \delta u(k) \end{bmatrix}^T \begin{bmatrix} Q(k) & S(k) \\ S^T(k) & R(k) \end{bmatrix} \begin{bmatrix} \delta x(k) \\ \delta u(k) \end{bmatrix} \right) + \delta x^T(k_f) Q(k_f) \delta x(k_f). \quad (32)$$

Expanding (31) for the discrete case:

$$\delta\xi^* = \arg\min_{\delta\xi} 2Dh(\xi) \circ \delta\xi + q(\xi) \circ (\delta\xi, \delta\xi)$$

$$= \arg\min_{\delta\xi} \sum_{k=k_0}^{k_f-1} \left[2 \begin{bmatrix} \frac{\partial \ell}{\partial x}(k) & \frac{\partial \ell}{\partial u}(k) \end{bmatrix} \begin{bmatrix} \delta x(k) \\ \delta u(k) \end{bmatrix} + \begin{bmatrix} \delta x(k) \\ \delta u(k) \end{bmatrix}^T \begin{bmatrix} Q(k) & S(k) \\ S^T(k) & R(k) \end{bmatrix} \begin{bmatrix} \delta x(k) \\ \delta u(k) \end{bmatrix} \right] + 2Dm(x(k_f))\delta x(k_0) + \delta x^T(k_f) Q(k_f) \delta x(k_f). \quad (33)$$

This is the discrete LQ problem. The LQ problem definition and solution are standard in linear systems theory (Anderson and Moore, 1990). The LQ problem is solved by three discrete Ricatti equations for $P(k)$, $b(k)$, and $c(k)$ that calculate the optimal input to minimize this cost. The optimal input is an affine feedback law of the form $\delta u(k) = -\mathcal{K}(k)\delta x(k) - C(k)$. Once the optimal input is found, the linear system is simulated forward in time using the input, and the simulated trajectory becomes the numeric descent direction $\delta\xi$.

5 Experiments and Examples

5.1 Pendulum on a Cart

This section deals with the pendulum on a cart shown in Fig. 8. The pendulum is optimized in continuous and discrete time to find a trajectory that moves the pendulum from a hanging position to an inverted one. The source code for this example is distributed with `trep` in the file `examples/pend-on-cart-optimization.py` or it can be found at `http://git.io/trep-cart-pend-invert`.

The pendulum on a cart is actuated by moving the cart horizontally. There is no joint torque applied directly to the pendulum. This creates a singularity in the dynamics. When the pendulum is horizontal, accelerating the cart no longer applies a torque to the pendulum.

The desired trajectory moves the pendulum from a hanging position to the inverted position and back again, so the system must pass through this singularity. However, if we use an initial trajectory for the optimization of the pendulum hanging in the stable position, the optimization will not cross the singularity. It will converge to a trajectory that swings the pendulum to just below the horizontal position.

Instead a sequence of progressively more difficult reference trajectories was used to generate a sequence of initial trajectories that are closer to the final, optimal solution. To find initial trajectories that move past the system's singularity, a *fictitious* input was added to directly apply a torque to the pendulum. This fictitious input does not actually exist, but it is

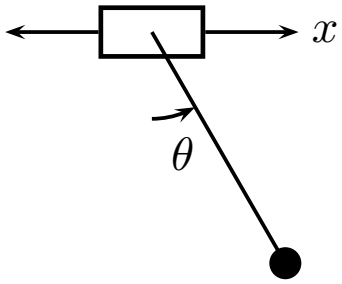


Figure 8. The pendulum on a cart is controlled by a horizontal force acting on the cart.

added to remove the singularity and allow the optimization to converge to an inverting trajectory.

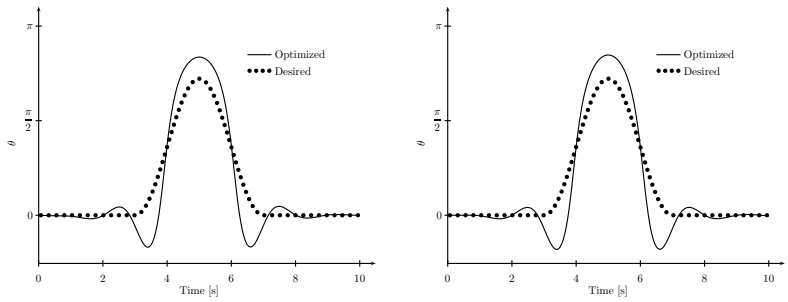
The optimization for the system with the fictitious input was run three times. Initially, a low cost was associated with the fictitious torque. In the following two optimizations, the cost was drastically increased. After the last optimization, the applied torque was essentially unused because of the associated cost. The trajectory for the final optimization was then used as the initial trajectory for the original problem with no fictitious input added.

This optimization was performed in both continuous and discrete time. The final results are shown in Fig. 9. Both optimizations converge to essentially the same trajectory. As can be seen, the final trajectory approximates the desired path. The path cannot be tracked exactly because of the limited dynamics of the system.

The discrete time optimization completes in 112 seconds on a 2.2GHz processor. Each iteration takes between 2 and 3 seconds depending on the number of Armijo steps taken.

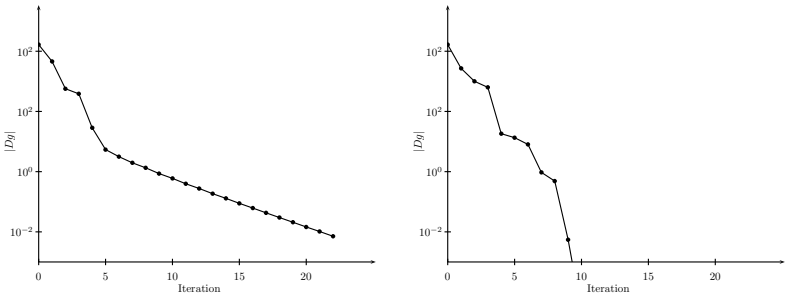
Fig. 10 shows convergence plots for the continuous and discrete optimizations during one of the intermediate optimizations. The discrete optimization converges in considerably fewer steps than the continuous one. The discrete time optimization was also more numerically robust. It tolerated larger ratios between the maximum and minimum costs of the individual states and reliably converged to stricter terminal conditions.

The continuous optimization, on the other hand, was more sensitive to



(a) Continuous Time (b) Discrete Time

Figure 9. Both optimizations converge to the same trajectory for the angle of the pendulum.



(a) Continuous Time (b) Discrete Time

Figure 10. Convergence plots for continuous and discrete time optimizations from a simulated desired trajectory.

the optimization parameters. Large cost ratios create stiffness that cause the descent direction calculation to fail when the Ricatti equation is numerically integrated. The discrete optimization is able to converge for ratios as high as $10^3 : 10^{-3}$ whereas the ratio had to be reduced to $100 : 1$ for the continuous optimization to proceed. In practice, large ratios allow the optimization to ignore components of the desired trajectory so that other components can be tracked more closely.

The terminal conditions also had to be relaxed for the continuous optimization to formally converge near the optimal solution. This is largely due to the error introduced by numeric integration being larger than the descent tolerance. The discrete optimization reliably converges with a terminal condition of $|Dg(\xi) \circ \delta\xi| < 10^{-6}$. The condition was relaxed to $|Dg(\xi) \circ \delta\xi| < 10^{-2}$ for the continuous optimization.

5.2 Example: `trep` Cart-Pendulum Trajectory Optimization

In this section we present `trep` code that performs a full trajectory optimization for the cart-pendulum system discussed in the previous section. The dynamic singularity that exists at $\pm\pi/2$ was avoided entirely to allow for less code. To avoid the singularity the pendulum starts at the unstable equilibrium and the reference trajectory never passes through the singularity. Avoiding the singularity entirely alleviates the need for the sequence of optimizations discussed in Sec. 5.1. The reference for θ is given by

$$\theta_d = \begin{cases} \pi & \text{if } 0 < t < 3 \\ \pi - (\alpha_d - \pi)\sin(\frac{\pi}{2}(t-3)) & \text{if } 3 \leq t < 7. \\ \pi & \text{if } t \leq 10 \end{cases}$$

For the rest of the state (the cart's x-position and two momenta), the reference is zero for all time. The initial state is $X(0) = [x(0)\ \theta(0)\ p_x(0)\ p_\theta(0)] = [0\ \pi\ 0\ 0]$; thus the system starts at rest at its unstable equilibrium. The reference trajectory describes tipping the pendulum over to a peak angle of α_d and then bringing it back up to the unstable equilibrium.

To begin the code, we import relevant modules, set constants related to the system and the simulation, and write a function that will allow us to calculate the reference trajectory for θ_d as follows:

```
1 from math import pi
2 import numpy as np
3 import trep
4 import trep.visual as visual
5 import matplotlib.pylab as mp
6 import trep.discopt as discopt
```

```
7
8  # set mass, length, and gravity:
9  m = 1.0; l = 1.0; g = 9.8; mc = 1;
10
11 # define initial state
12 q0 = np.array([0, pi]) # x = [x_cart, theta]
13 p0 = np.array([0, 0])
14 X0 = np.hstack([q0,p0])
15
16 # define time parameters:
17 dt = 0.1
18 tf = 10.
19
20 # define reference trajectory
21 ad = 7*pi/8.
22 def fref(t):
23     if 0<t and t<3:
24         return pi
25     elif 3<=t and t<7:
26         return pi + (ad-pi)*np.sin(pi/4.*(t-3))
27     elif t<=10:
28         return pi
29     else:
30         print "Error!"
```

Note that we have set the reference angle as $\alpha_d = 7\pi/8$. The next segment creates a simple `trep` model representing the system:

```
32 # create system
33 system = trep.System()
34 # define frames
35 frames = [
36     trep.tx("x_cart", name="CartFrame", mass=mc), [
37         trep.rz("theta", name="PendulumBase"), [
38             trep.ty(-l, name="Pendulum", mass=m)]]]
39 # add frames to system
40 system.import_frames(frames)
41 # add gravity potential
42 trep.potentials.Gravity(system, (0,-g,0))
43 # add a horizontal force on the cart
44 trep.forces.ConfigForce(system, "x_cart", "cart_force")
```

Now we create a vector of times, a variational integrator object, and a discrete system object.

```
# create a variational integrator, and a discrete system
t = np.arange(0,tf+dt,dt)
mvi = trep.MidpointVI(system)
dsys = discopt.DSystem(mvi, t)
```

Thus far, we have only used `trep` functionality that has already been demonstrated in previous sections. Next, we are going to build discrete states and input trajectories defining both an initial iterate and a reference. We will use a `trep` convenience function, `build_trajectory`, that automatically returns arrays of the correct size for `trep.discopt.DSystem` object where any non-specified components are set to zero.

```
# create an initial guess and reference trajectory
(Xinit,Uinit) = dsys.build_trajectory([q0.tolist()]*len(t))
qref = [[0, fref(x)] for x in t]
(Xref,Uref) = dsys.build_trajectory(qref)
```

Now we create matrices representing the Q and R weighting matrices in the Eq. (23). Then we instantiate a cost object, and use that object to instantiate an optimizer object. Note that we set the cost on θ to be several orders of magnitude higher than that of x, and they are both significantly higher than the cost on the momenta. This results in a strong preference for tipping the pendulum over, and a willingness to deviate from $x = 0$ in order to do so.

```
# create cost functions:
Q = np.diag([100, 50000, 0.1, 0.1])
R = np.diag([1])
cost = discopt.DCost(Xref, Uref, Q, R)

# define an optimizer object
optimizer = discopt.DOptimizer(dsys, cost)
```

The final step is to perform the full trajectory optimization, and plot the results. We set the optimizer to use four steps of gradient descent before moving to Newton's method. Note that this optimization could be entirely solved with gradient descent, but the faster convergence of Newton's method is convenient for an example problem.

```
# setup and perform optimization
optimizer.first_method_iterations = 4
```

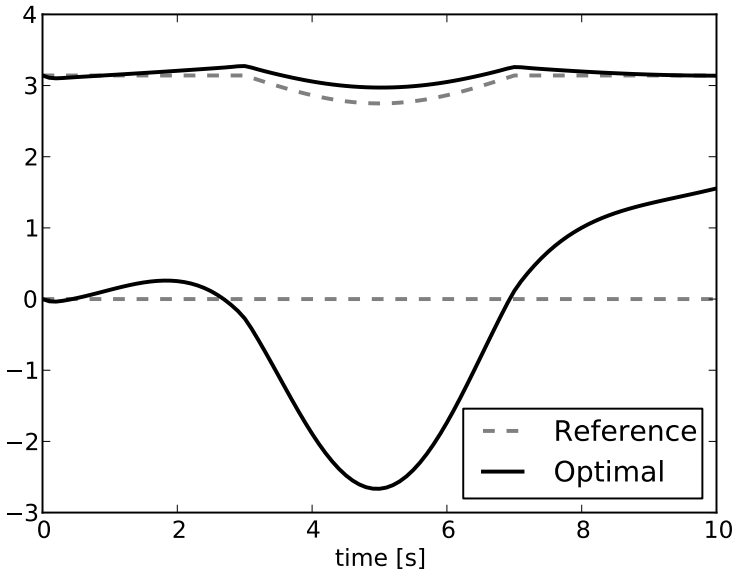
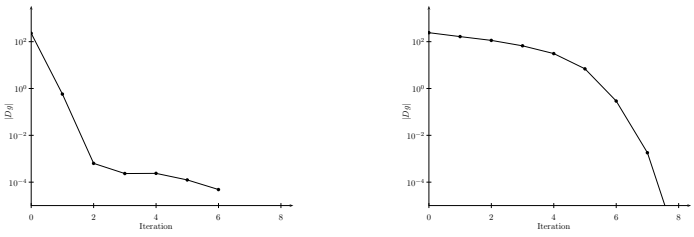


Figure 11. Image showing the reference configuration trajectory and the optimized configuration trajectory for the example cart-pendulum system of Sec. 5.2. The lower curves are x and the upper curves are θ.

```
66 finished, X, U = optimizer.optimize(Xinit, Uinit)
67
68 mp.hold(True)
69 l1 = mp.plot(t, Xref[:,0:2],"--",lw=2, color="gray")[0]
70 l2 = mp.plot(t, X[:,0:2],lw=2,color="black")[0]
71 mp.hold(False)
72 mp.legend([l1,l2],["Reference", "Optimal"],
73           loc="lower right")
74 mp.xlabel("time [s]")
75 mp.show()
```

In Fig. 11 it is clear that the optimal θ trajectory tracks the reference much better than the optimal x trajectory. This is in agreement with the chosen costs and makes sense because the system is nonminimum phase near the inverted equillibrium.



(a) Continuous Time (b) Discrete Time

Figure 12. Convergence plots for continuous and discrete time optimizations from a simulated desired trajectory.

5.3 Example: Marionette Nonlinear Optimization

We again consider the model of a humanoid marionette shown in Fig. 4. The marionette has 40 configuration variables and is actuated by 6 strings. The strings are modeled as holonomic constraints. Kinematic configuration variables control the two-dimensional position of the end-point of each string as well as the string length Johnson and Murphey (2007). There are no joint torques and only slight damping.

Two optimizations are discussed in the following subsections. The first optimization uses a desired trajectory that was generated separately by simulating the system, and the second discusses an optimization with a reference trajectory generated by human motion capture data.

Desired Motion: Simulated Trajectory A desired trajectory was created by simulating the system forward in time. The lengths of the arm and leg strings were varied sinusoidally to create a walking motion. The configuration trajectory was saved. The rest of the state (e.g, configuration velocity or discrete momentum) and the simulation inputs were discarded and replaced with uniformly zero trajectories. This results in a smooth reference trajectory that we expect the puppet to be able to track, but is still an infeasible trajectory.

The marionette was optimized to the desired trajectory in both continuous and discrete time. Both optimizations successfully converged to solutions that track the desired configuration very well. Convergence plots for both optimizations are shown in Fig. 12. The source code for the discrete optimization is distributed with `trep` in the file `examples/puppet-optimization.py`.

In this case, the continuous optimization initially converges faster than the discrete one. It tracks the desired trajectory almost perfectly after a single step. The discrete optimization makes slow progress initially but converges quickly after about five iterations. The discrete optimization takes 5:50s to finish. Each iteration takes between 15 and 60 seconds depending on the descent direction type and number of Armijo steps.

Although the convergence plot is flattering for the continuous optimization, there were numerous problems. As with the pendulum example of Sec. 5.1, the continuous optimization was sensitive to the optimization parameters. Large ratios between the maximum and minimum state cost cause the optimization to fail. The terminal conditions had to be relaxed again as well. The discrete time optimization suffers from neither of these problems.

Desired Motion: Motion Capture Data A more practical application of the trajectory optimization is finding trajectories to track data acquired from a motion capture system. In this example, a desired trajectory was generated using a Microsoft Kinect® to record a student walking in place. This process is illustrated in Fig. 13.

In this case, the continuous optimization was unable to converge. The discrete optimization converged successfully and found a trajectory that closely approximates the student's movement. Fig. 14 plots the desired trajectory and optimization result for the angle of the right elbow as an example. The trajectory found by the optimization tracks the desired trajectory very well. However, a large amount of noise was introduced. This is most likely caused by too large of a ratio between the weight of the configuration portion of the state compared the discrete momentum portions and the cost of the inputs.

6 Conclusion

Variational integrators provide an appealing alternative to numerically solving the equations of motion for mechanical systems. By representing variational integrators as discrete dynamic systems and calculating the linearization of the associated one-step map, their utility is extended to applications requiring analysis and optimal control. This approach reduces complexity, potential for error, and extraneous work compared to using a variational integrator for simulation while doing the analysis and optimization in the continuous domain with a separate set of equations. Moreover, it leads to feedback laws that are expressed purely in terms of configuration variables (instead of configurations and configuration velocities).

The methods described here can be efficiently implemented by using a

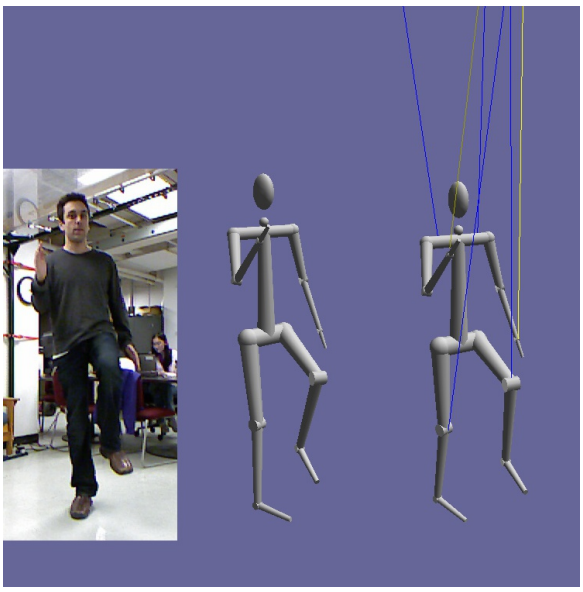


Figure 13. These three images show a single frame from the motion-capture optimization. The left-most picture shows images recorded by a Microsoft Kinect. The middle figure is the motion capture data found from the image. The right-most figure is the optimized trajectory.

recursive tree representation to calculate the required derivatives of the discrete Lagrangian and forcing function. The approach accommodates external forcing and holonomic constraints as described here, and is compatible with kinematic configuration variables (Johnson and Murphey, 2007; Johnson, 2012). Additionally, this method could be extended to calculate higher derivatives if needed.

Acknowledgment

This material is based upon work supported by the United States National Science Foundation under award CMMI-1200321, IIS-1018167, and CNS-1329891. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily

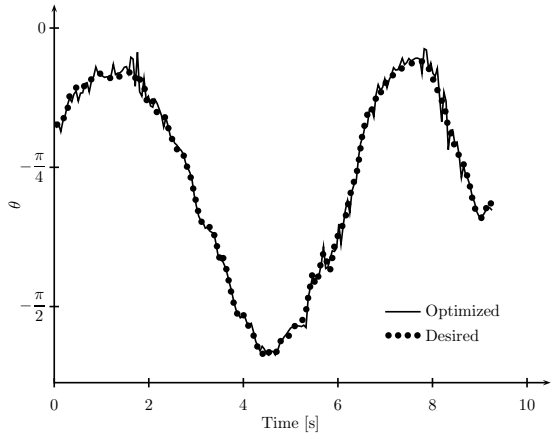


Figure 14. The optimal trajectory for the right arm elbow tracks the desired trajectory well.

reflect the views of the National Science Foundation.

Bibliography

B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice Hall, Inc, 1990.

L. Armijo. Minimization of functions having lipschitz continuous first-partial derivatives. *Pacific Journal of Mathematics*, 16, 1966.

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.

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.

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

J. Hauser. A projection operator approach to the optimization of trajectory functionals. In *IFAC World Congress*, Barcelona, Spain, July 2002.

E. Johnson. *Trajectory Optimization and Regulation for Constrained Discrete Mechanical Systems*. PhD thesis, Northwestern University, 2012.

E. Johnson and T. D. Murphey. Scalable variational integrators for constrained mechanical systems in generalized coordinates. *IEEE Trans. on Robotics*, 25(6):1249–1261, 2009.

E. Johnson, J. Schultz, and T. Murphey. Structured linearization of discrete mechanical systems for analysis and optimal control. *IEEE Trans. on Automation Sci. and Eng.*, PP(99):1–13, July 2014.

E. R. Johnson and T. D. Murphey. Dynamic modeling and motion planning for marionettes: Rigid bodies articulated by massless strings. In *International Conference on Robotics and Automation*, Rome, Italy, 2007.

C. T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

C. T. Kelley. *Iterative Methods for Linear Nonlinear Equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1995.

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.

A. Lew. Variational time integrators in computational solid mechanics. *California Institute of Technology Thesis*, 2003.

A. Lew, J. E. Marsden, M. Ortiz, and M. West. Asynchronous variational integrators. *Arch. Rational Mech. Anal.*, 167:85–146, 2003.

A. Lew, J. E. Marsden, M. Ortiz, and M. West. In *Finite Element Methods: 1970's and Beyond*, pages 98–115, 2004.

S. Leyendecker, S. Ober-BlöBaum, J. E. Marsden, and M. Ortiz. Discrete mechanics and optimal control for constrained systems. *Optimal Control Applications and Methods*, 31(6):505–528, 2010.

A. Locatelli. *Optimal Control: An Introduction*. Birkhäuser, 2001.

J. E. Marsen and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, 10:357–514, 2001.

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

S. Ober-Blöbaum, O. Junge, and J. E. Marsden. Discrete mechanics and optimal control: An analysis. *ESAIM: Control, Optimisation and Calculus of Variations*, 17(2):322–352, 2011.

J. Schultz and T. D. Murphey. Embedded control synthesis using one-step methods in discrete mechanics. In *American Controls Conf. (ACC)*, pages 5293–5298, Washington, D.C., June 2013.

J. Schultz and T. D. Murphey. Extending filter performance through structured integration. In *American Controls Conf. (ACC)*, pages 430–436, Portland, OR, June 2014.

M. West. Variational integrators. *California Institute of Technology Thesis*, 2004.