2011 American Control Conference
on O'Farrell Street, San Francisco, CA, USA
June 29 - July 01, 2011

Control Aesthetics in Software Architecture for Robotic Marionettes

Todd D. Murphey and Elliot R. Johnson

Abstract— **This paper considers the design of software for embedded control of robotic marionettes using choreography to specify the marionette motion. Marionettes are actuated by strings, so the mechanical description of the marionettes either creates a multi-scale or degenerate system—making simulation of the constrained dynamics challenging. Moreover, the marionettes have 40-50 degrees of freedom with closed kinematic chains. Choreography requires motion primitives typically originating from human motion that one wants the marionette to imitate, resulting in a high dimensional nonlinear optimal control problem that needs to be solved for each primitive. Once one has motion primitives to use, they must be pieced together in a way that preserves stability, resulting in an optimal timing control problem. These three computational components lead to software requirements for the embedded system, including efficient computation of the 1) discrete time dynamics that preserve the constraints and other integrals of motion, 2) nonlinear optimal control policies (including optimal control of LTV systems), and 3) optimal timing of choreography. All of these need to take fast convergence into account. We show how to provide all these capabilities in a single framework. Moreover, in order to meet these requirements new results on projection operators on finite dimensional function spaces are needed—both of which are critical to ensuring acceptable convergence of the algorithms. We conclude with our current results and application of these ideas to other systems.**

I. INTRODUCTION

Efficient methods in simulation for highly articulated rigid body systems have been studied for many years [28], [27], [4], [3], [2], [11]. However, less emphasis on *control* calculations for these same systems has been present. This paper focuses on the questions of how to both simulate and control an arbitrarily complex rigid body system while keeping scalability and convergence of the resulting numerical routines. We have been using robotic marionette systems (seen in Fig. 1) as an example of a complex system that requires careful embedded control that can handle many degrees of freedom.

This work is a collaboration with Georgia Tech, The Atlanta Center for Puppetry Arts, and Disney Imagineering. Disney Imagineering has played a central role in helping develop the hardware platform partially because animatronics in theme parks are very heavy, slow, and expensive and robotic marionettes promise to be both more agile and less costly and cumbersome. However, controlling marionettes is a very hard technical problem; the marionettes have many degrees of freedom, have mechanical degeneracy due to the strings, and are highly constrained.

(a) (b)

Fig. 1. The robotic marionette system in (a) is actuated by small wheeled robots that run on the underside of a tarp. The goal is to use motion captured from the dancers in (b) as reference data for the marionettes. Software must enable transforming the dancers' motion into dynamically admissible motion for the marionettes and allow one to piece these motions together using choreography.

However, we *know* that puppeteers can solve these high dimensional motion planning problems—puppeteers do successfully get marionettes to convincingly imitate human motion—so we know the problems are solvable. However, how can we mimic how puppeteers manage complexity and uncertainty and apply these insights to critical areas such as embedded control of active prosthetics? Our current hypothesis is that the choreography used in marionette theater plays a critical role in how puppeteers manage complexity, and that what we learn from studying choreography in marionettes will affect our understanding of other complex applications such as embedded control of prosthetics (briefly discussed at the end of this article). Key questions include how do we deal with complexity in motion control and how do we represent what we want a complex system to do? Choreography provides a way of canonically identifying useful motions (e.g., walking, running, waving, reaching) that can be pieced together. How one pieces these motions together determines whether the motions stably and smoothly transition.

This paper is organized as follows. Section II describes typical approaches in dynamic simulation and optimal control and what software requirements these approaches create. Section III discusses what special considerations should be taken into account when working in discrete time—as software always does. Section IV illustrates our software approach with a brief discussion of some successful example systems that the software automatically optimizes.

T.D. Murphey t-murphey@u.northwestern.edu
E.R. Johnson elliot.r.johnson@u.northwestern.edu
Department of Mechanical Engineering, Northwestern University, 2145 Sheridan Road, Evanston, IL, USA 60208

This material is based upon work supported by the National Science Foundation under award IIS-0917837. 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.

II. Typical Approach

In this section we discuss typical analytical approaches to optimal control and what sort of software infrastructure these approaches assume. We start by describing how one may describe the dynamics of a mechanical system and then move on to computing optimal controllers for that system.

A. Dynamics

When computing dynamics we are typically trying to compute dynamics of the form

$$\dot{x} = f(x, u) \tag{1}$$

where $x = (q, \dot{q})$ and $q \in Q$ describes the configuration of the system. For rigid body systems, it has historically been convenient to write down the rigid body system in Newton-Euler coordinates (i.e., $Q = SE(3)^n$, where n is the number of rigid bodies in the system. This yields a state space of dimension $12n$ that is subject to constraints. For the marionette, for instance, just the body has 10 rigid bodies, so the state space for just the body would be 120 dimensions. If one includes the actuators, one adds a minimum of one degree of freedom for the length of each of the five strings and three to six degrees of freedom for each of the five actuators (depending on whether the actuators are planar or not). For the marionettes this bring the total nominal dimension of the state space up to $12{\cdot}10{+}2{\cdot}5{+}12{\cdot}5 = 190$. It should be clear that we don't want to be solving for feedback controllers in a 190 dimensional space if we can avoid it.

Because of these issues, we don't want to represent Eq. (1) as Newton-Euler equations and instead insist on working in generalized coordinates. In the case of the marionettes, this reduces the dimension of the state to $2m$, where m is the number of generalized coordinates. This give us 22 dynamic degrees of freedom for the marionette itself and another 18 degrees of freedom for the actuators, yielding 80 states. By utilizing a kinematic reduction [6], [5] we can reduce the state of the actuators down to 18 because the actuators are fully actuated. This gives us equations of motion of the following form.

$$\begin{aligned} \dot{x}_a &= u \\ \dot{x}_p &= f(x_p, x_a) \end{aligned} \tag{2}$$

where x_a is the kinematic configuration of the actuators and $x_p = (q_p, \dot{q}_p)$ is the dynamic configuration *and* its velocity of the marionette itself. (For details on this, see [17].) This leaves us with a much smaller, more manageable system to work with that only has a total of 62 dimensions in its state space.

The last thing to point out is that because of the strings there are holonomic constraints relating the end of the arm to the length of the string (e.g., when the string length is constant L, the end of the arm evolves on a sphere of radius L). This creates a constraint

$$h(x_a, q_p) = 0 \tag{3}$$

that must be maintained during the simulation.

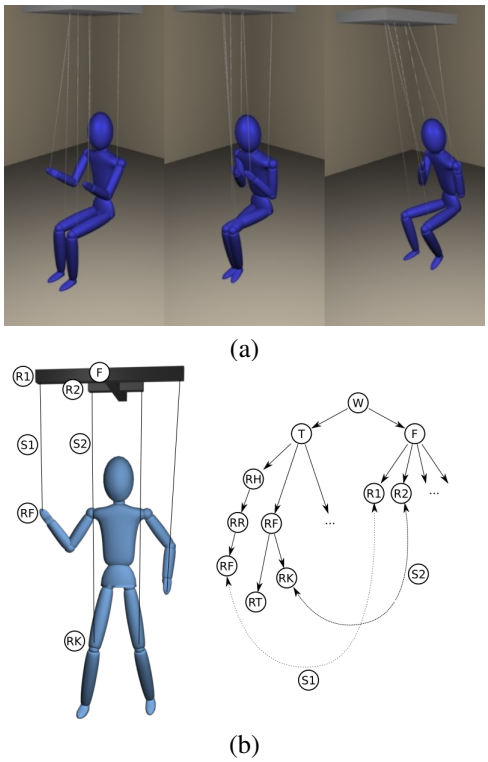


Fig. 2. Simulation of complex rigid bodies can take advantage of the mechanical topology of the system. For instance, a marionette is being simulated in (a) using a tree structure representation of the humanoid form in (b) and representing the constraints by cycles in the graph (see [16]).

Assume for a moment that Eqs (2) and (3) can be stably simulated in an efficient manner. How do we construct the differential equation and constraints in a systematic manner? The standard way to do this is based on Featherstone's early work [11] on articulated body dynamics. This work was largely used in the context of animation, where the requirements are substantially different than embedded control (for instance, if one can get an animation to "look right" once, the task is done, whereas controlled physical systems have to be repeatable). Recursive approaches to calculating dynamics [25], [11] take advantage of special representations of mechanical systems that allow the values needed for simulation to be calculated quickly and avoid redundant calculations.

The work in this paper is based on the methods presented in [16] (based on [11]). Systems are represented as graphs where each node is a coordinate frame in the mechanical system and the nodes are connected by simple rigid body transformations (typically translations along and rotation about the X, Y, and Z axes though any rigid body screw motion can be used). Transformations are either constant or parameterized by real-valued variables. The set of all variables establishes the generalized coordinates for the system. Figure 2 is an example of a marionette. The graph description can include closed kinematic chains, but in practice the graph is converted to an acyclic directed graph (i.e, a tree) and augmented with holonomic constraints to close the kinematic

chains. This approach leads to fast ways to calculate $f(\cdot,\cdot)$ in Eq. (2) and $h(\cdot)$ in Eq. (3). Moreover, one can use the same structure to efficiently calculate the linearization [14], which is critical to nonlinear optimal control calculations, discussed next.

B. Nonlinear Optimal Control

Optimal control typically starts out with a cost function of some sort, often of the form

$$J = \int_{t_0}^{t_f} L(x(t), x_{\text{ref}}(t), u(t))dt + m(x(t_f), x_{\text{ref}}(t_f)) \quad (4)$$

where $L(\cdot)$ represents a weighted estimate of the error between the state and the reference state (which is potentially not a feasible trajectory for the system). Minimizing this cost function subject to the dynamics in Eqs. (2) and (3) can be done using iterative descent methods. In particular, one uses the equivalence between the constrained minimization and the unconstrained minimization of the objective function composed with a differentiable projection $\mathscr{P}(\cdot)$ onto the constrained subspace. That is, the two minimizations

$$\min_{v \in W \subseteq V} g(v) = \min_{v \in V} g(\mathscr{P}(v))$$

(where V is the vector space and W is the differentiable submanifold of admissible vectors) are equivalent [12]. The projection operator $\mathscr{P}(\cdot)$ comes from computing a feedback law (discussed in more detail in the next section). In particular, if one interprets the "gradient" descent algorithm as starting at some nominal trajectory $\xi = (x(t), u(t))$ and solving for a descent direction $\zeta = (z, v)$ that optimizes the local quadratic model

$$\zeta = \arg\min_{\zeta} Dg(\xi) \cdot D\mathscr{P}(\xi) \cdot \zeta + \|\zeta\|^2,$$

then one just has to solve a standard time-varying LQR problem. This means that one has to be able to compute the time-varying linearization

$$\dot{z} = A(t)z + B(t)v \quad (5)$$

where $A(t) = \frac{\partial f}{dx}(x(t), u(t)) = D_1 f(x(t), u(t))$ and $B(t) = \frac{\partial f}{du}(x(t), u(t)) = D_2 f(x(t), u(t))$. One has to be able to do so for arbitrary trajectories in the state space, potentially including infeasible trajectories (in the case of linearizing about the desired trajectory). Solving for the descent direction involves solving the Riccati equations

$$\dot{P} + A(t)^T P + PA(t) + Q - PB(t)R^{-1}B(t)^T P = 0. \quad (6)$$

If we additionally want to guarantee quadratic convergence, then we can solve a different LQR problem

$$\zeta = \arg\min_{\zeta} Dg(\xi) \cdot D\mathscr{P}(\xi) \cdot \zeta + \|\zeta\|^2_{D^2 J}$$

where

$$\begin{aligned} D^2 J(\xi) \cdot (\zeta^1, \zeta^2) \quad &= \quad D^2 g(\xi) \cdot (D\mathscr{P}(\xi) \cdot \zeta^1, D\mathscr{P}(\xi) \cdot \zeta^2) \\ &\quad + Dg(\xi) \cdot D^2 \mathscr{P}(\xi) \cdot (\zeta^1, \zeta^2). \end{aligned} \quad (7)$$

The second derivative $D^2\mathscr{P}(\cdot)$ requires that we be able to also calculate $\frac{\partial^2 f}{dx^2}$, $\frac{\partial^2 f}{du^2}$, and $\frac{\partial^2 f}{dxdu}$. The details of this approach can be found in [12] and elsewhere, but for us the key thing is that we have to be able to compute Eqs. (2), (3), (4), (5), (6), and (7) in software. The difficulty, as we will see, is that we are representing the optimal control problem in continuous time, which creates problems due to the fact that the actual computations are in discrete time. We will discuss this more momentarily in Section III.

C. Choreography and Hybrid Optimal Control

In [23], [8], [24] we developed an optimal control interpretation of choreography. In particular, we formalize choreography as a sequence of *modes* that can be pieced together to form a script. Each mode has its own dynamics, creating a system with dynamics

$$\dot{x} = f(x(t), u(t)) = f_i(x(t), u(t)) \quad t \in (\tau_i, \tau_{i+1})$$

where each i corresponds to a different mode of the system. To optimize such a system, one needs to be able to minimize an objective function J with respect to the switching times τ_i of the system. This derivative $\frac{\partial J}{\partial \tau_i}$ with respect to the switching times depends on the switching time adjoint equation

$$\dot{\rho} + A(t)^T \rho + \frac{\partial L}{\partial x} = 0 \quad (8)$$

along with a boundary condition at $\rho(t_f)$ (see [15], [7], [10], [9]). This adjoint equation only needs to be computed once to compute all the derivatives of J. If one wants to compute the second derivative of J, one needs the second-order switching time optimization

$$\dot{P} + A(t)^T P + PA(t) + \frac{\partial^2 L}{\partial x^2} + \sum_k \rho_k \frac{\partial^2 f^k}{\partial x^2} = 0 \quad (9)$$

along with its boundary condition $P(t_f)$ [7]. This adjoint equation, along with a solution to Eq. (8), only needs to be computed once to compute all the derivatives of J. Note that the second-order switching time adjoint equation is the same as the Riccati equation in Eq. (6) except that the Riccati equation has an different final term. Indeed, both Eq. (8) and Eq. (9) both only require first and second derivatives of f_i with respect to the state, so those are all that are needed for software; hence, the choreographic optimization requires the same software capabilities as the smooth optimization in Section II-B.

III. Discrete time with scalability

As previously mentioned, the continuous representation of dynamics found in Eq. (1) is not what we actually use to do computations. Moreover, when there are constraints, such as those seen in Eqs. (2) and (3), standard methods such as Runge-Kutta methods fail to preserve the constraints. Typically one would use solvers designed for Differential Algebraic Equations (DAEs) that project the numerical prediction onto the set of constrained solutions defined by the constraint in Eq. (3). We have found, however, that for high-index DAEs such as the marionette a tremendous amount of "artificial stabilization" is required to make the simulation of the DAE stable. This artificial stabilization—which typically

takes the constraint $h(q)$ as a reference and introduces a feedback law that "stabilizes" the constraint—changes the dynamics of the system, and if the feedback gain is high often creates a multi-scale simulation problem that is incompatible with real-time operation. As an alternative, we consider variational integrators [13], [26], [18], [21], [20], [30], [19].

Variational integration methods use the stationary action principle as a foundation for numerical integration that does not involve differential equations. This approach has several advantages–known conservation properties (such as guarantees about conservation of momenta, the Hamiltonian, and the constraints) as well as guaranteed convergence to the correct trajectory as the time step converges to zero. More importantly, variational integration techniques exactly simulate a *modified Lagrangian* system where the modified Lagrangian is a perturbation of the original Lagrangian. The Discrete Euler-Lagrange (DEL) equations are

$$\begin{aligned} D_1 L_d(q_k, q_{k+1}, k) + D_2 L_d(q_{k-1}, q_k, k) &= F_k \quad (10) \\ h(q_{k+1}) &= 0 \quad (11) \end{aligned}$$

where L_d is a discretized form of the Lagrangian and F_k is an external force integrated over the k time step. This forms a root solving problem in which, given q_{k-1} and q_k, one solves for q_{k+1}. Repeating this rootsolving procedure forms the basis of simulation. Using this method, we can (using a recursive technique similar to the one described in Section II-A), simulate the marionette in real-time using time steps of $0.01\ s$ without adding any sort of numerical heuristics such as artificial stabilization.

Let's say we start form the DEL equations and assume, by application of the implicit function theorem, that the solution exists and is locally unique [22]. Then, once we have made a choice of state (we choose $x_k = (q_k, p_k)$ where p_k is the generalized momentum), we have an update equation of the form

$$x_{k+1} = f_k(x_k, u_k)$$

just as we would if we had started from a differential equation. That is, the general form of the discrete time equation we wish to optimize is no different–in principle–in the variational case than it is in the standard ODE case. More importantly, the fact that f_k is implicitly defined by the DEL equations does not affect whether the linearization is implicitly defined. In fact, one can calculate an *exact* linearization of the DEL equations, including constraints and closed kinematic chains [14]. So we may wish to know what the discrete-time analog of Eqs. (6) and (7) are. (These can be found in [1].) The difficulty is that one cannot linearize Eq. (1) directly because that is the infinitesimal linearization; to get a discrete-time linearization one would nominally have to solve the state transition matrix (STM) locally over the time step. Approximating the STM leads to a linearization that does not respect the constraints, leading to a local optimal controller that essentially fights the constraints. In contrast, taking variations directly with respect to x_k yields an algebraic calculation for the linearization and higher-order derivatives with respect to the state. As with variational integration, the key to linearization is to take variations with respect to the discrete state rather than the continuous state.

For nonlinear optimal control in the discrete time setting, we need to know if the resulting projection operator is in fact a projection and whether it is differentiable.

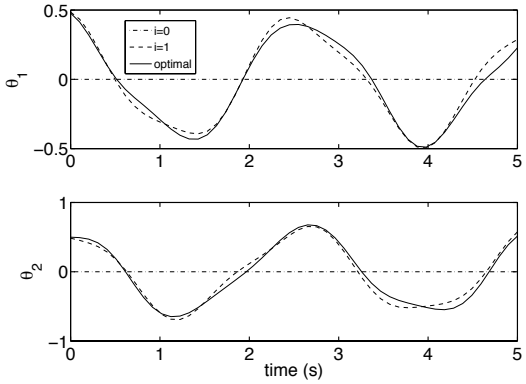


Fig. 3. Projection-based variational optimization of a planar double pendulum [29]

To see that such a projection is valuable, consider Fig. 3 [29], where a planar double pendulum trajectory is being optimized. The initial guess for the optimal solution is the "zero" solution, the optimal solution is the solid black line, and the *first* iteration of Newton's method using the projection operation is the dotted line. Hence, one step of Newton's method almost solves the global optimization in this case. Naturally, that will not always be the case, but this is an indication of how much generating a differential projection operation can help.

Let $\overline{\xi} = (\overline{x}, \overline{u})$ be a desired, potentially infeasible, curve in the space the trajectories reside in and let $\xi = (x, u)$ be admissible trajectories. The continuous time projection operator is defined by $\xi = \mathscr{P}(\overline{\xi})$ such that

$$\begin{aligned} x(t_0) &= \overline{x}(t_0) \\ \dot{x} &= f(x, u) \\ u &= \overline{u} - K(t)(x - \overline{x}) \end{aligned}$$

where the feedback gain $K(t)$ comes from solving the Riccati equation in Eq. (6). One can verify that $\mathscr{P}(\cdot)$ is a projection and that it is C^∞ if f is C^∞. What do we do if we are using Eqs. (10) and (11) instead of Eqs. (2) and (3)? Then the discrete projection operator $\mathscr{P}_d(\cdot)$ is $\xi_d = \mathscr{P}_d(\overline{\xi}_d)$ such that

$$\begin{aligned} x_0 &= \overline{x}_0 \\ x_{k+1} &= f_k(x_k, u_k) \\ u_k &= \overline{u}_k - K_k(x_k - \overline{x}_k) \end{aligned} \quad (12)$$

where the discrete time feedback gain K_k comes from solving a discrete time Riccati equation. To see that it is a differentiable projection, we introduce the next Lemma.

Lemma 3.1: $\mathscr{P}_d(\cdot)$ is a projection.

Proof: We need to show that the projection satisfies the property $\mathscr{P}_d(\bar{\xi}_d) = \mathscr{P}_d(\mathscr{P}_d(\bar{\xi}_d))$. First we calculate $(a, b) =$

$\mathscr{P}_d(\alpha, \mu)$ and get that $a_1 = \alpha_1$, $a_k = f(\alpha_{k-1}, \mu_{k-1})$,$b_1 = \mu_1$, and $b_k = \mu_k + K_k(a_k - \alpha_k)$. Now calculate $(x, u) = \mathscr{P}_d(a, b)$ and find that $x_1 = a_1$, $x_k = f(\alpha_1, \mu_1) = a_2$, $b_1 = \mu_1$, and—the critical part—$b_2 + K_2(x_2 - a_2) = b_2 + K_2(a_2 - a_2) = b_2$. By induction we find $\mathscr{P}_d \circ (\alpha, \mu) = \mathscr{P}_d \circ \mathscr{P}_d \circ (\alpha, \mu)$. Hence, $\mathscr{P}_d(\cdot)$ is a projection operation from discrete-time representations of curves $\bar{\xi}_d$ to discrete-time representations of trajectories ξ_d. ■

Next we need to calculate the derivative of $\mathscr{P}_d(\cdot)$, starting with $\xi_d = \mathscr{P}_d(\bar{\xi}_d)$ (we are going to drop the d from ξ_d for notational convenience).

$$\delta\xi = D\mathscr{P}_d(\bar{\xi}) \circ \delta\bar{\xi}$$

So, by Eq. (12), we get

$$\begin{aligned}
\delta x_{k_0} &= \delta\bar{x}_{k_0} \\
\delta x_{k+1} &= \frac{\partial f_k}{\partial x_k}\delta x_k + \frac{\partial f_k}{\partial u_k}\delta u_k = Df_k \circ \delta\xi_k \\
\delta u_k &= \delta\bar{u}_k - K_k(\delta x_k - \delta\bar{x}_k).
\end{aligned}$$

where $\frac{\partial f_k}{\partial x_k}$ is shorthand for $\frac{\partial f_k}{\partial x}(x_k, u_k, k)$. (The same applies to $\frac{\partial f_k}{\partial u_k}$ and Df_k.) As in the continuous case, the discrete projection is a discrete linear system. The second derivative is also straightforward (here we let $\delta\bar{\xi}^1$ and $\delta\bar{\xi}^2$ be two independent perturbations to $\bar{\xi}$).

$$\delta^2\xi = D^2\mathscr{P}(\bar{\xi}) \circ (\delta\bar{\xi}^1, \delta\bar{\xi}^2)$$

which implies, again by Eq. (12), that

$$\begin{aligned}
\delta^2 x_{k_0} &= 0 \\
\delta^2 x_{k+1} &= D^2 f_k \circ (\delta\xi_k^1, \delta\xi_k^2) + Df_k \circ \delta^2\xi_k \\
&= \frac{\partial f_k}{\partial x_k}\delta^2 x_k + \frac{\partial f_k}{\partial u_k}\delta^2 u_k + D^2 f_k \circ (\delta\xi_k^1, \delta\xi_k^2) \\
\delta^2 u_k &= -K_k\delta^2 x_k.
\end{aligned}$$

Rewriting the second derivative, we get:

$$\begin{aligned}
\delta^2 x_{k+1} &= \frac{\partial f_k}{\partial x_k}\delta^2 x_k + \frac{\partial f_k}{\partial u_k}\delta^2 u_k + D^2 f_k \circ (\delta\xi_k^1, \delta\xi_k^2) \\
&= \left[\frac{\partial f_k}{\partial x_k} - \frac{\partial f_k}{\partial u_k}K_k\right]\delta^2 x_k + D^2 f_k \circ (\delta\xi_k^1, \delta\xi_k^2).
\end{aligned}$$

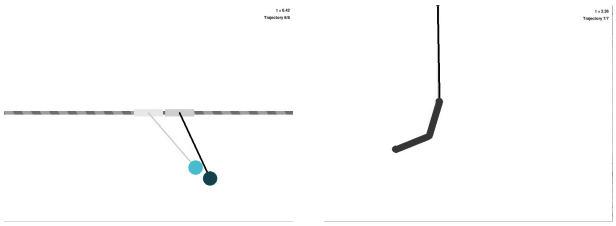
This is a discrete affine system, equivalent to a discrete linear system with an input:

$$\begin{aligned}
\delta^2 x_{k+1} &= \hat{A}_k\delta^2 x_k + \hat{B}_k \\
\hat{A}_k &= \left[\frac{\partial f_k}{\partial x_k} - \frac{\partial f_k}{\partial u_k}K_k\right] \qquad \hat{B}_k = D^2 f_k \circ (\delta\xi_k^1, \delta\xi_k^2).
\end{aligned}$$

Hence, the projection operation $\mathscr{P}_d$ is second differentiable with derivatives that are represented by discrete-time linear difference equations, allowing us to apply Newton's method to optimal control problems.

IV. Examples

At the current stage of development, we have tested some low-dimensional examples to ensure that we get the same answers in `trep`[1] as we do using standard software implementations in *Mathematica*. These two cases are seen in Fig. 4. The underactuated cart-pendulum problem is one that has been well-studied as a nonlinear control problem. The key thing is that we can, by specifying the two node graph that describes the relationship between the two rigid bodies and their inertial characteristics, compute an optimal control law that swings the pendulum up to an inverted configuration. The simple marionette arm waving example is something we computed before in *Mathematica* [23], but here we only specify the constrained optimization problem in terms of the graph and obtain the optimal solution. Hence, at least for simple graphs, all the optimal control calculations that we wish to be able to do are feasible.



(a) (b)

Fig. 4. Successful optimizations include automatic optimization of the cart-pendulum problem in (a) and the marionette waving arm in (b). The cart-pendulum example can track the desired, infeasible trajectory given by the blue pendulum to an inverted configuration and back, starting from the hanging equilibrium. The marionette waving arm example can lift the arm (through a kinematic singularity), wave, and drop it back down. These calculations produce both the optimal trajectories as well as the feedback laws that stabilize them, and do so using only the graph description of the mechanical system.

For the full marionette we have not yet completed optimal control calculations, but—to indicate the real-time feasibility of the algorithms we discussed in the previous sections—we now provide some timing data for `trep`. For the marionette simulation in Fig. 2 we have a system that has 22 dynamic degrees-of-freedom and 18 kinematic degrees of freedom (corresponding to the actuation of the strings). The total number of constraints due to the strings is 6 and the total number of force inputs is 12. To evaluate the continuous dynamics that would be used in a standard integrator, one evaluation of $f(x, u)$ requires 2.7 ms, while the first derivative with respect to the state (i.e., the linearization) requires 24 ms and the second derivative with respect to the state requires 400 ms. Note that this does not say anything about how long it will take to simulate a particular length of time because the time step is not included here because we are not working in discrete time. With the variational integrator from Eqs. (10) and (11) with a time step of 0.01 (no other parameters are needed when using variational integrators, even with the degeneracies and constraints the strings introduce), the update step requires 5.53 ms while the first derivative with respect to the state (i.e., the exact discrete linearization) takes 2.4 ms and the second derivative derivative with respect to the state takes 130 ms. This means that, at minimum, we can simulate and evaluate controllability in real-time.

We have used these software techniques for the tendon-articulated hand in Fig. 5 and can compute linearizations

[1]The simulation portion of `trep`—written in Python and C—is already available at *http://trep.sourceforge.net*, but the optimal control techniques discussed here are not yet available in the distribution of the software.

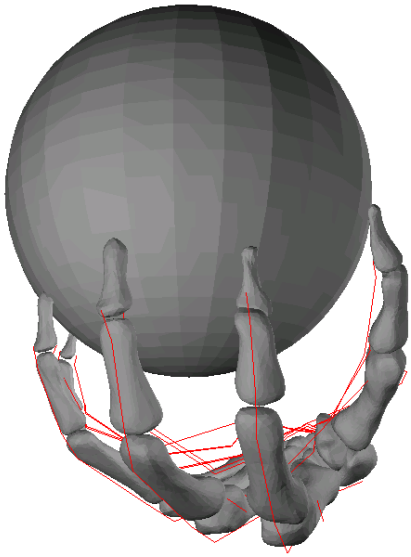


Fig. 5. The graph-based approach to calculating linearizations scales to complex mechanical systems like this dynamic model of a tendon-articulated human hand holding an object. The linearization at this configuration shows that the system is locally controllable.

and local LQR controllers for the hand. (This simulation capability is now being used with prosthetic control in a collaboration with the Rehabilitation Institute of Chicago.)

V. CONCLUSIONS

We have been using the robotic marionette project as an example system that forces us to make software that can both simulate and control "complex" mechanical systems. The marionettes play a vital role in driving the system development—they have mechanical degeneracies, closed kinematic chains, and are high dimensional, but we know that puppeteers successfully control them. Therefore, they make a good testbed for understanding whether or not our software is producing reasonable results.

The techniques we use provide both optimal trajectories and control laws that stabilize those trajectories. Moreover, because we formulate the optimal control problem using a differentiable projection, we can analytically guarantee quadratic convergence locally around the optimal trajectory.

Moreover, puppeteers use the marionettes in dynamic, expressive ways, so we know that extremely conservative motions based on a "quasi-static" approach or an inverse-kinematics approach is very unlikely to produce interesting, artistic motions. Although we can now produce an optimal "imitation" of a human motion in the case of waving (seen in Fig. 4) we will not really understand if marionette imitation can be understood as an optimization problem until we can do the calculations for the full marionette. We are in the midst of doing so now.

REFERENCES

[1] B.D.O. Anderson and J.B. Moore. *Linear Optimal Control*. Prentice Hall, Inc, 1971.

[2] D. Baraff. Non-penetrating rigid body simulation. In *State of the Art Reports*, 1993.

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

[4] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *SIGGRAPH*, pages 137–146, 1996.

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

[6] F. Bullo and A.D. Lewis. Low-order controllability and kinematic reductions for affine connection control systems. *SIAM Journal on Control and Optimization*, 44(3):885–908, 2005.

[7] T. Caldwell and T. D. Murphey. Switching mode generation and optimal estimation with application to skid-steering. *Automatica*, 2010. In Press.

[8] M. Egerstedt, T. D. Murphey, and J. Ludwig. *Hybrid Systems: Computation and Control*, volume TBD of *Lecture Notes in Computer Science*, chapter Motion Programs for Puppet Choreography and Control, pages 190–202. Springer-Verlag, 2007. Eds. A. Bemporad, A. Bicchi, and G. C. Buttazzo.

[9] M. Egerstedt, Y. Wardi, and H. Axelsson. Optimal control of switching times in hybrid systems. In *IEEE Methods and Models in Automation and Robotics*, Miedzyzdroje, Poland, Aug. 2003.

[10] M. Egerstedt, Y. Wardi, and F. Delmotte. Optimal control of switching times in switched dynamical systems. In *IEEE Conference on Decision and Control*, Maui, Hawaii, Dec. 2003.

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

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

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

[14] E. Johnson and T. D. Murphey. Linearizations for mechanical systems in generalized coordinates. In *American Controls Conf. (ACC)*, pages 629–633, 2010.

[15] E. Johnson and T. D. Murphey. Second-order switching time optimization for nonlinear time-varying dynamic systems. *IEEE Transactions on Automatic Control*, 2010. Accepted for Publication.

[16] E. R. Johnson and T. D. Murphey. Scalable variational integrators for constrained mechanical systems in generalized coordinates. *IEEE Transactions on Robotics*, 2010.

[17] 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.

[18] 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.

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

[20] A. Lew, J. E. Marsden, M. Ortiz, and M. West. An overview of variational integrators. In *Finite Element Methods: 1970's and Beyond*, pages 98–115, 2004.

[21] A. Lew, J. E. Marsden, M. Ortiz, and M. West. Variational time integrators. *Int. J. Numer. Methods Engrg*, 60:153–212, 2004.

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

[23] P. Martin, E. Johnson, T. D. Murphey, and M. Egerstedt. Constructing and implementing motion programs for robotic marionettes. *IEEE Transactions on Automatic Control*, 2010. Accepted for Publication.

[24] T. D. Murphey and M. E. Egerstedt. Choreography for marionettes: Imitation, planning, and control. In *IEEE Int. Conf. on Intelligent Robots and Systems Workshop on Art and Robotics*, 2007. 6 pages.

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

[26] K. Nichols and T. D. Murphey. Variational integrators for constrained cables. In *IEEE Int. Conf. on Automation Science and Engineering (CASE)*, pages 802–807, 2008.

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

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

[29] K. Snyder and T. D. Murphey. Second-order DMOC using projections. In *IEEE Int. Conf. on Decision and Control (CDC)*, 2010.

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