Automated Trajectory Morphing For Marionettes Using Trajectory Optimization

Elliot R. Johnson and Todd D. Murphey
Electrical and Computer Engineering
University of Colorado at Boulder
Boulder, Colorado 80309
erjohnso@colorado.edu and murphey@colorado.edu

***Abstract*— We present a method to automatically morph trajectories from one mechanical system into trajectories of another system with potentially very different dynamics. Our method relies on a mapping from the source system's configuration space to that of the target system to create a desired trajectory for the target system. A projection operator-based trajectory optimization finds a dynamically-admissible trajectory for the target system that approximates the desired trajectory. The optimization algorithm is extremely robust and often finds a satisfactory trajectory even for highly nonlinear systems with instabilities, uncontrollable modes, or closed kinematic loops. This robustness simplifies the mapping step by essentially allowing us to ignore the target's dynamics and focus on the purpose of the trajectory.**

We demonstrate the process using a marionette as an example. The marionette has complex dynamics, uncontrollable modes, and closed kinematic chains. The trajectory morphing process allows us to specify trajectories for the puppet using human motion capture or animation tools without considering the actual abilities of the puppet.

I. INTRODUCTION

We present a method for automatically morphing trajectories from one mechanical system into trajectories of another system with potentially very different dynamics. Our method relies on a mapping from the source system's configuration space to that of the target system to create a desired trajectory for the target system. A projection operator-based trajectory optimization finds a dynamically-admissible trajectory for the target system that approximates the desired trajectory. The optimization algorithm is extremely robust and often finds a satisfactory trajectory even for highly nonlinear systems with instabilities, uncontrollable modes, or closed kinematic loops. This robustness simplifies the mapping step by essentially allowing us to ignore the target's dynamics and focus on other important aspects of the trajectory like expressiveness or obstacle avoidance.

Generating trajectories for complex dynamic systems like a human robot or automated marionette is a difficult problem. The designer must balance their effort between developing rich, expressive movements and staying within the capabilities of the system.

The most straightforward approach to this problem is to decouple the two requirements. First, a trajectory is created without regard to dynamics. This enables us to use any convenient method to create a trajectory: hand animation, motion capture, etc. Once a trajectory is created, the system dynamics are considered to find a dynamically admissible trajectory that best approximates the original trajectory.

Generalizing this idea further, the desired trajectory can even be specified for a different system. It would be easier to use motion capture data from a human, for example, than to capture data from a marionette. If there is a suitable mapping from the configuration space of the source system to that of the target system, the trajectory can be mapped to the target system even if the result is not dynamically admissible.

The term *trajectory* has different meanings throughout this paper. In general, a *trajectory* is a set of continuous curves representing the states and inputs to a system over a time interval $[t_0, t_f]$. In this usage, a trajectory may or may not satisfy the dynamics of the system. An *admissible trajectory* always satisfies the system dynamics. The set of all admissible trajectories of a system is denoted by $\mathcal{T}$. Thus, $\xi = (x(\cdot), u(\cdot))$ is an admissible trajectory if and only if $\dot{x}(\tau) = f(\tau, x(\tau), u(\tau)) \; \forall \, \tau \in [t_0, t_f]$.

The trajectory morphing process has two major components: a map between the configuration spaces of the two systems and a trajectory optimization. The map is created by hand and maps configurations of the source system to the target system. The map ignores the dynamics of both systems and the trajectories produced are unlikely to be dynamically admissible. The trajectory optimization, on the other hand, considers the dynamics.

We use a projection-operator based approach to trajectory optimization [3]. It searches the trajectory space of the target system for an admissible trajectory that best resembles the desired one. The returned trajectory is always admissible, even if the system dynamics are nonlinear, uncontrollable, or unstable.

The projection operator approach uses a novel modification. Instead of a constrained search in the admissible trajectory space of the system, it performs an unconstrained search in the larger general trajectory space and uses a projection operator to project trajectories into admissible trajectories. In this setting, the search is roughly analogous to an optimization in finite dimensions using an iterative gradient descent method. At each iteration, a new descent direction is found by solving a linear optimal control problem. An Armijo search is performed in the descent direction until it

finds a new trajectory that improves the current score. The new trajectory is projected back into the admissible trajectory space, and the process is repeated. We will discuss the trajectory optimization in more detail later.

The overall morphing process is relatively automatic and robust. Once the mapping exists and the trajectory optimization has been properly configured, new trajectories can be morphed without any modifications.

We will discuss a simple marionette system as an example application of trajectory morphing. Trajectory morphing is exceptionally well matched to the marionette problem. The desired trajectories are often complex, expressive movements, but the dynamics are too complex to consider while designing trajectories. There are uncontrollable modes. In addition, the closed kinematic chains created by the marionette strings give the configuration manifold an unusual shape. In practice, the configuration is represented as a set of real numbers to numerically simulate the system. The irregular shape of the manifold leads to large, irregular pockets in this simulated configuration space that must be avoided. The trajectory morphing process works despite these issues. Trajectories are specified for a simple human model using hand animation or motion capture; methods that can easily capture complex and expressive trajectories. The trajectories are mapped to the puppet using a straightforward map which will be described later. The trajectory optimization works within the controllable subspace and admissible configurations to find dynamically correct trajectories that resemble the original desired plan. The trajectory optimization even finds the inputs that correspond to the admissible trajectory. In this sense, the optimization also acts as a model inversion tool to find the input for a desired output. This is a nontrivial problem for a complex mechanical system.

II. Trajectory Optimization

In trajectory optimization, we have some desired trajectory $\xi_d = (x_d(\cdot), u_d(\cdot))$ that may or may not satisfy the system dynamics. We want to find an admissible trajectory that is similar to the desired. This is a constrained optimization problem defined by (1):

$$\underset{\xi \in \mathcal{T}}{\operatorname{argmin}}\, h(\xi) = \int_{t_o}^{t_f} l(\tau, x(\tau), u(\tau))\mathrm{d}\tau + m(x(T)) \tag{1}$$

where $\xi = (x(\cdot), u(\cdot))$ and $l(\dots)$ and $m(\dots)$ are incremental and terminal cost functions. The standard cost functions are shown in (2).

$$\begin{aligned} l(\tau, x, u) =& (x - x_d(\tau))^T Q (x - x_d(\tau)) \\ &+ (u - u_d(\tau))^T R (u - u_d(\tau)) \end{aligned} \tag{2a}$$

$$m(x_f) = (x_f - x_d(t_f))^T P_1 (x_f - x_d(t_f)) \tag{2b}$$

where $Q(\cdot)$, $R(\cdot)$, and P_1 are positive definite matrices that weight the errors between the some trajectory and the desired.

Because solutions to (1) need to be constrained by the system's dynamics, a standard gradient-descent approach will not work. If we found a descent direction and adjusted the current trajectory with it, the result would not be dynamically admissible. However, the constrained optimization can be converted to an unconstrained problem by introducing the projection operator.

Suppose there exists a projection operator $\mathcal{P}(\xi)$ that maps potential trajectories to admissible ones. We repose the original problem by defining a new cost function $g(\xi) = h(\mathcal{P}(\xi))$ to get the unconstrained optimization in (3).

$$\underset{\xi}{\operatorname{argmin}}\, g(\xi) = h(\mathcal{P}(\xi)) \tag{3}$$

If ξ locally minimizes (3), then the projected trajectory $\eta = \mathcal{P}(\xi)$ locally minimizes (1), solving the original problem. The search is still over an infinite dimensional space, but we can now choose descent directions and new trajectories without satisfying the complex, nonlinear dynamics of the system. Using a typical algorithm for unconstrained optimization in finite dimensions as a guide, we can design an algorithm to solve (3).

Trajectory Optimization Algorithm

Given an initial admissible trajectory η_0:
For $i = 0, 1, 2, \dots$:

1) Create a projection operator around η_i.
2) Find a descent direction:
$$\zeta_i = \underset{\zeta}{\operatorname{argmin}}\, Dg(\eta_i) \circ \zeta + \tfrac{1}{2} q(\zeta, \zeta) \tag{4}$$
3) If $Dg(\eta_i) \circ \zeta = 0$, stop.
4) Find a step giving a sufficient decrease:
$$\gamma_i = \underset{\gamma}{\operatorname{argmin}}\, g(\eta_i + \gamma \zeta_i) \tag{5}$$
5) Advance Step:
$$\eta_{i+1} = \mathcal{P}(\eta_i + \gamma_i \zeta_i) \tag{6}$$

From this perspective, the algorithm is relatively straightforward. First, a projection operator is designed for each new trajectory so that we can avoid requiring projection operators to be globally valid and instead just ask that they work in some neighborhood around a trajectory. Next we find a descent direction ζ that gives the biggest decrease in cost based on a quadratic model of the cost function. If no direction gives a decrease, the current trajectory is a local minimum. Otherwise, the descent direction is used to find a new trajectory with a lower cost. It is unlikely that $\eta_i + \gamma_i \zeta_i$ is admissible, so the trajectory is projected back into the trajectory manifold.

A. Projection Operator

The projection operator, $\mathcal{P}(\xi)$, maps trajectories, $\xi = (\alpha, \mu)$, into admissible trajectories, $\eta = (x, u)$, of a dynamic system [4]. The projection operator is defined using a linear

feedback law.

$$\begin{aligned}\mathcal{P} \;:\; \xi = (\alpha, \mu) &\to \eta = (x, u) \\ x(0) &= \alpha(0) \\ \dot{x} &= f(x(t), u(t)) \\ u(t) &= \mu(t) + K(t)(\alpha(t) - x(t))\end{aligned} \tag{7}$$

where $K(t)$ stabilizes the system over some domain. The feedback component of the projection operator plays a large role in the robustness of the optimization, particularly its ability to handle unstable systems.

The controller also determines the domain of $\mathcal{P}(\xi)$ because it may only be able to stabilize trajectories near the current trajectory. In practice, it is assumed that the domain is limited, so a new controller is designed around each new admissible trajectory in the optimization algorithm. This avoids the difficult problem of finding a globally stabilizing and well performing controller.

Stabilizing controllers are generated by solving a finite-time linear quadratic regulator (LQR) problem [2]. The system is represented as a time-varying linear system by linearizing about the current trajectory and LQR finds a stabilizing state feedback controller. The LQR approach is robust and automated. Generally, we choose a set of LQR weightings and then automatically generate a stabilizing controller about any trajectory as needed. Additionally, many software packages like MATLAB and LabVIEW provide efficient and robust tools to solve the LQR problem.

Note that it is desirable for the projection operator to have as large of a domain as possible so that potentially large steps can be take. However, the trajectory optimization is set up that it will continue to work even if the domain is extremely small.

The projection operator itself is continuous and differentiable. The first derivative is defined as

$$\begin{aligned}D\mathcal{P}(\eta) : \gamma = (\beta(\cdot), \nu(\cdot)) &\to \zeta = (z(\cdot), x(\cdot)) \\ z(t_0) &= 0 \\ A(\tau) &= D_x f(\tau, x(\tau), u(\tau)) \\ B(\tau) &= D_u f(\tau, x(\tau), u(\tau)) \\ \dot{x}(\tau) &= A(\tau)z(\tau) + B(\tau)v(\tau) \\ v(\tau) &= \nu(\tau) + K(t)[\beta(\tau) - z(\tau)]\end{aligned} \tag{8}$$

Note that the derivative is a linear projection operator itself. The range of $D\mathcal{P}(\xi)$ defines the tangent trajectory space of ξ, denoted as $T_\xi\mathcal{T}$. An element $\zeta \in T_\xi\mathcal{T}$ if and only if $\zeta = D\mathcal{P}(\xi) \circ \zeta$.

A complete discussion of the projection operator, including expressions for higher derivatives, can be found in [4] and [3].

B. Descent Direction

The descent direction for each step is found from the same definition as a finite dimensional optimization, but using a completely different procedure. In both cases, a quadratic model is created based on the current point. We then seek the direction that maximizes the decrease in the cost function:

$$\zeta_i = \underset{\zeta}{\operatorname{argmin}}\, Dg(\xi_i) \circ \zeta + \tfrac{1}{2} q \circ (\zeta, \zeta) \tag{9}$$

The bilinear operator $q \circ (\cdot, \cdot)$ is chosen to give different descent algorithms. The identity operator leads to a steepest-descent direction. Letting $q = D^2 g(\xi_i)$ leads to a Newton-descent direction [8]. Typically, the steepest-descent direction has better global convergence properties, so it is used for the first few iterations since the initial trajectory might be far away from the optimum trajectory. The Newton-descent direction, on the other hand, has excellent local convergence properties and is used in later iterations where the current trajectory is likely to be near the optimum trajectory.

In the finite dimensional case, the minimizer of (9) is $\zeta = q^{-1} Dg(\xi_i)$. In the infinite dimensional case, however, q^{-1} does not exist so a different approach is needed.

In direct contradiction to an earlier step, this unconstrained optimization is transformed into a constrained optimization by restricting $\zeta \in T_{\xi_i}\mathcal{T}$. In this case, however, $T_{\xi_i}\mathcal{T}$ is a linear subspace and the resulting constrained optimization is a linear optimal control problem that can be solved using linear optimal control tools.

We do not include the derivation here, but the solution is found by solving a two-point boundary-valued problem (TPBVP). The TPBVP either be solved directly using modern numeric techniques [10] or decomposed into two initial value problems using a Ricatti transformation. For a complete derivation and explanation, see [3].

C. Armijo Line Search

Once the descent direction is found, we need to find an appropriate step size. Instead of performing the actual minimization in (5), we use an Armijo line search [8] to find a $\gamma \in (0, 1]$ that gives a sufficient, but not necessarily optimal, decrease.

$$\underset{i=0,1,2,\ldots}{\operatorname{argmin}}\; g(\xi_i + \gamma\zeta_i) < g(\xi_i) + \alpha\gamma Dg(\xi) \circ \zeta \qquad \gamma = \beta^i \tag{10}$$

where $\beta \in (0, 1)$ and $\alpha \in (0, 1)$ are algorithmic parameters. This is simply an algorithm to decrease the step size in a way that guarantees convergence [7].

For optimization in finite dimensions, the step size is decreased to avoid overshooting and increasing the cost. In trajectory optimization, it is additionally needed to keep $\xi_i + \gamma\zeta_i$ in the domain of $\mathcal{P}(\xi_i)$. Otherwise, the projection operator may fail to stabilize the trajectory and will fail. For the marionette, this can happen by driving the system into configurations that are inconsistent with the closed kinematic chains (ie, tearing off the arm with the string). By decreasing the step size until the new trajectory is in the domain of $\mathcal{P}(\xi_i)$, the Armijo line search plays an important role in ensuring the final trajectory is admissible.

III. Using Trajectory Optimization for Morphing

In trajectory morphing, we have a conceptual or source model that we use to generate trajectories and a dynamic or target model that we want to imitate the trajectories with. It is

assumed that we already have a method of easily generating trajectories for the source system. The trajectory optimization tool takes a desired trajectory for the dynamic system and searches for an admissible trajectory that best resembles the desired. In order to get the desired trajectory, we need a map from the configuration manifold of the conceptual system to that of the dynamic system. This mapping will be specific to the two models and is created manually.

For the marionette, our conceptual model is usually a human being while the dynamic model is the marionette. The two models differ in size and use different input methods that result in different dynamic behavior. To create a mapping, we focus on the similarities between the two systems.

The human is a skeleton that is almost completely controllable. The marionette is the same skeleton with additional strings that have position and length actuation. Unlike the human, the marionette is relatively under-actuated and cannot directly apply torques to the joints. Since the optimization will take the dynamics into account, however, we can ignore the differences in actuation while creating the map.

By ignoring the dynamics, we can make the map relatively simple. Parts of the human model that are not related to the marionette are discarded (for example, torque inputs). Parts common to both models are mapped directly (shoulder and elbow angles). Finally, we generate reasonable values for parts of the marionette that are unrelated to the human (string lengths and positions).

The mapping itself suggests how we choose the weightings for the optimization cost function. The configuration variables that were mapped directly from the conceptual model are weighted the heaviest. The configuration variables that did not correspond to the conceptual model are weighted very little, but are still weighted to keep them bounded and reasonable. Note that because of the low weighting and the general robustness of the trajectory optimizer, the actual values generated for the puppet's unique configuration variables are typically not very important and insignificantly affect the final results.

IV. Results for a 2D marionette arm

We now present a simple example that illustrates the method just discussed. Consider the two-dimensional marionette arm shown in Fig. 1. The upper and lower arms are free to rotate about their respective joints. The shoulder is fixed. The arm is controller by a single string tied to end of the arm. The other end of the string moves horizontally. The length of the string is also controllable. The dynamic model is defined using a mixed dynamic-kinematic model [5] to include the strings as kinematic inputs.

We define motions for the arm using a computer animation package [1] The software allows us to create a skeleton that represents a two-link, two-dimensional arm and to animate the skeleton using key-frames and interpolation. This allows us to define the exact motion we want without worrying about the marionette dynamics. A 10 second waving motion was manually. The arm starts from a neutral lowered position. It lifts up, waves for about 3 periods, and then lowers back down to the original position.

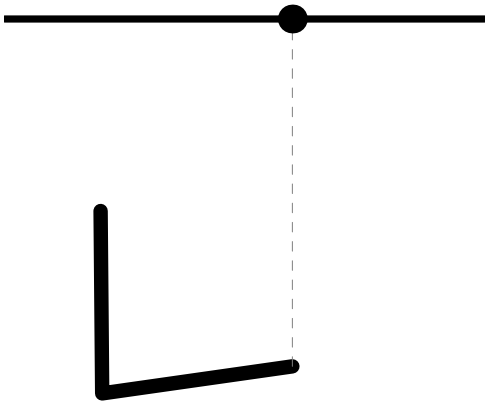

Fig. 1. Animations can be defined for a simple 2D arm and morphed into trajectories for this marionette arm.

We generated a mapping between the model arm and the marionette arm. The joint angles are mapped directly between the models. The tip of the model arm was projected onto the line that the string moves on to get a desired string position. Finally, the desired string length was found so that the closed kinematic loop would be consistent with the other configuration variables.

The optimization was set up to emphasize the joint angles. The two angles were weighted one hundred times heavier than the string parameters. The initial trajectory was the arm simply hanging in the starting position of the initial trajectory for the entire time interval. The optimization converged in 11 iterations and took nearly 4 hours[1].

The cost is plotted against iterations in Fig. 3. The optimization converges towards the minimum very quickly at first as the steepest-descent direction moves toward the minimum. After a few steps, the trajectory is near the optimum trajectory. The Newton-descent direction takes over after the 5th iteration and quickly converges on the final trajectory.

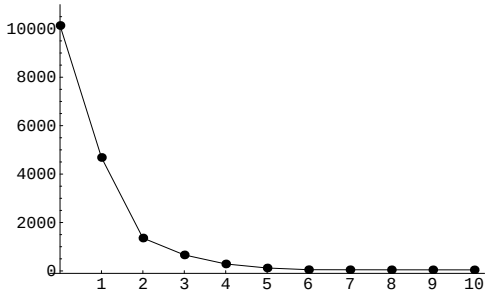

Fig. 3. The trajectory cost as a function of the iteration.

Fig. 4 shows several snapshots of the morphed trajectory compared to the desired trajectory. Note that there are small

[1]The optimization and dynamics were implemented in Mathematica 5.2 and ran on a 1.5 Ghz PowerPC G4 with 512MB of RAM

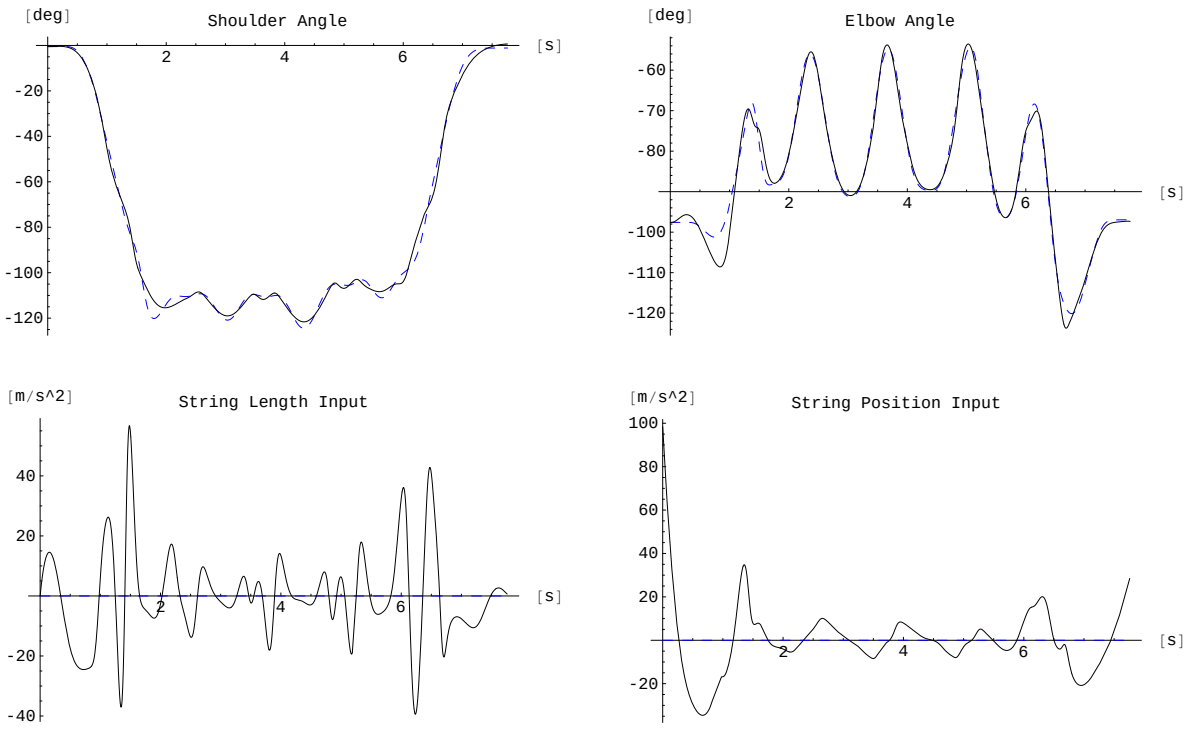

Fig. 2. *Top to Bottom, Left to Right:* Shoulder Angle, Elbow Angle, Second Derivative of String Length, Second Derivative of String Position. The solid and dashed lines are the optimized and desired trajectories, respectively.

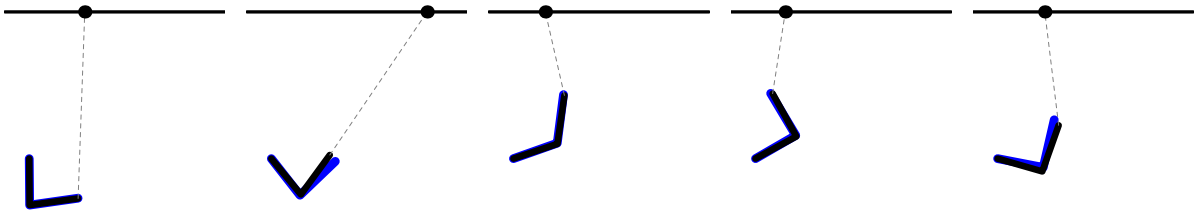

Fig. 4. Several frames from the resulting trajectory. The marionette and human models are drawn in black and blue, respectively. The complete animation can be seen at *http://puppeteer.colorado.edu*

differences between the desired trajectory and the feasible trajectory in frames 2 and 5. These are due to the uncontrollable parts of the marionette model. Fig. 2 plots the joint angles and inputs found by the optimization. The optimized trajectory is clearly a good approximation of the desired waving motion. For our model, the inputs are the second derivatives of the string position and length. We can see that the necessary inputs are complicated and could probably not be specified manually. An animation of the results can be seen on our project website at *http://puppeteer.colorado.edu*

V. CONCLUSIONS AND FUTURE WORKS

Trajectory morphing is a promising tool for generating trajectories for complex mechanical systems. It frees the designer to use the most convenient input method and model and is robust enough to handle degenerate nonlinear systems with instabilities and uncontrollable modes. It is also relatively automated. Once the mapping and cost functions have been designed, the process is basically a black box that converts trajectories between systems.

This approach is particularly well suited to our marionette problem. We can use motion capture data from a person to create the complicated, expressive trajectories found in traditional marionette performances. The optimization effectively morphs these to the puppet's dynamics despite the system being under-actuated and having closed kinematic chains.

The next step in this research will be to develop alternative cost functions. The quadratic model can give excellent results, but setting the weightings are often a matter of trial and error. We might want to consider eliminating the map between the two systems and define a cost function directly in terms of the conceptual model's trajectory. For example, we might use motion capture data from a two link arm to create trajectories for a ten link robotic arm. Rather than mapping the trajectory to the ten link arm and optimizing with the quadratic cost function, it would be more natural

to define a cost function that compares the configurations of the ten link arm directly against the two link model.

We would also like to see the trajectory optimization algorithm extended to work directly with variational integrators. Variational integrators simulate the dynamics of a mechanical system directly in a discrete domain rather than numerically approximating a continuous ODE [9] [6]. They have many desirable conservative properties and, for systems as large as our full marionette, tend to be more computationally efficient than conventional Euler-Lagrange dynamics. By directly integrating variational integrators with trajectory optimization tools, we expect the trajectory morphing will naturally scale to complex mechanical systems and still be fast enough to be practical.

VI. ACKNOWLEDGMENTS

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

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

REFERENCES

[1] Blender. *http://www.blender.org*, 2007.
[2] B.D.O. Anderson and J.B. Moore. *Linear Optimal Control*. Prentice Hall, Inc, 1971.
[3] J. Hauser. A projection operator approach to optimization of trajectory functionals. Barcelona, Spain, 2002.
[4] J. Hauser and D.G. Meyer. The trajectory manifold of a nonlinear control system. 1998.
[5] 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.
[6] E.R. Johnson and T.D. Murphey. Discrete and continuous mechanics for tree representations of mechanical systems. In *International Conference on Robotics and Automation*, 2008.
[7] C.T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics (SIAM), 1999.
[8] D.G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.
[9] J. E. Marsen and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, pages 357–514, 2001.
[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C Second Edition*. Cambridge University Press, 1992.