5th Annual IEEE Conference on Automation Science and Engineering
Bangalore, India, August 22-25, 2009

Automated Trajectory Synthesis From Animation Data Using Trajectory Optimization

Elliot R. Johnson and Todd D. Murphey
Mechanical Engineering
Northwestern University
Evanston/Chicago, IL 60208
erjohnso@colorado.edu and t-murphey@northwestern.edu

***Abstract*— Trajectory optimization is a technique for finding dynamically feasible trajectories of a mechanical system that approximate a desired trajectory. These tools provide an excellent abstraction from a system's dynamics, allowing a user to animate a robot without considering dynamic complexities, including coupling, instability, and uncontrollable subspaces. The animator focuses on the more relevant ideas of expression and communication. The trajectory optimization bridges the gap between animator and robot, automatically projecting trajectories into the system's reachable set and determining the necessary inputs to the system. These techniques handle closed kinematic chains, constraints, and unstable systems without modification.**

We give a detailed overview of the optimization technique and present an example from the autonomous marionette project. A waving motion is animated for a two-dimensional arm and the corresponding dynamically-feasible trajectory is found for an under-actuated marionette arm. The trajectory is tested on robotic marionette platform and correctly approximates the original waving animation. The marionette arm is an excellent test bed because it is under-actuated, highly nonlinear, and highly coupled.

I. INTRODUCTION

Animatronic systems are typically complex, nonlinear mechanical systems with dozens of degrees of freedom. Ideally, an animator could focus on expression and communication to create lifelike motion without considering the underlying dynamics. Current tools are not robust enough to permit such an approach. Animating these systems is a tedious task that relies on a trial and error approach and the experience of skilled technicians. The animator directly designs the inputs to the system, and the animation is reproduced by playing back the inputs with no feedback.

Animatronic mechanical design also reflects these problems. The robots are fully-actuated and extremely rigid and strong to be as kinematic as possible. This increases the cost and raises safety concerns about people working in close proximity to such powerful machines.

Allowing the animator to specify desired trajectories instead of manually designing inputs is difficult for two reasons. First, the desired trajectory has to be converted into a dynamically feasible one. Second, the inputs to track the trajectory have to be automatically generated. Both problems are simplified by making the system fully-actuated and rigid, but they are unavoidable to some degree. On the other hand, if better tools are developed, the systems could be cheaper to build, easier to animate, and safer to work around.

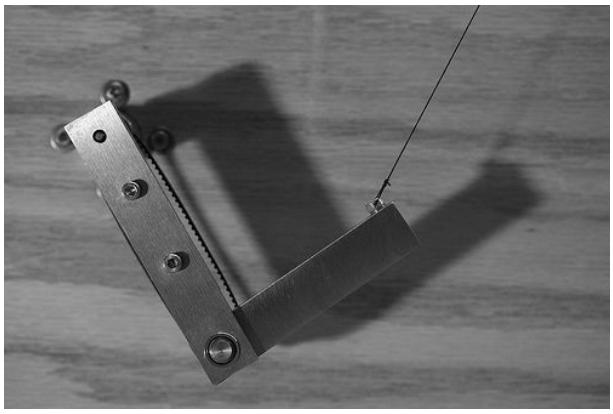


Fig. 1: Trajectory optimization makes it easy to animate a under-actuated, highly-nonlinear marionette arm.

We have been developing autonomous robotic marionette platforms for entertainment applications. A user describes a scene using choreography at a semantically high level. The system then synthesizes actual trajectories (for one or more marionettes) and provides the inputs and stabilizing controllers needed to perform the scene on a hardware platform (Fig. 1).

Proper abstractions are absolutely necessary to make this problem tractable. In particular, the first step of generating trajectories from a choreography needs to be isolated from the dynamics of a puppet (e.g. 'swing' dynamics, infeasible motions) and, in fact, shouldn't be concerned with 'extra' parts of the trajectory like string lengths and positions. Therefore we need a robust tool that can find dynamically feasible trajectories that approximate a desired trajectory and can solve for the correct inputs (i.e. string lengths and origins) to create that motion.

Note that marionettes exemplify the two problems mentioned earlier. They are extremely under-actuated (an arm with 4-5 degrees of freedom is controlled by one string) and the inputs (i.e. string length and position) to generate a particular trajectory are non-obvious. At the same time, the rich history behind puppeteering shows that these system are

capable of imitating real human motion.

We have found that a projection operator-based trajectory optimization technique [3] works well for this problem. It allows us to specify trajectories without considering the strings, and handles the closed kinematic chains without any difficulty.

Trajectory optimization is analogous to an iterative gradient descent optimization in finite dimensions. A cost function, $g()$, measures the error between a trajectory and the desired trajectory[1]. We start with some initial trajectory ξ_0 and find a direction ζ to move in that reduces the cost ($g(\xi_0 + \zeta) < g(\xi_0)$). Moving in this direction gives an improved trajectory $\xi_1 = \xi_0 + \zeta$. This process repeats until we end in an local minimum of the cost function. In this paper, we review this algorithm in detail, including how ζ is found and how the projection operator is used to ensure that the resulting trajectories are admissible trajectories for the system dynamics.

A key idea in this approach is that once an animator is freed from the dynamics of a system, they are animating a conceptual model of the system rather than the dynamic model. The conceptual model only includes important aspects of the system. Extraneous parts can be ignored and left for the optimization to determine. The conceptual model for a marionette, for example, is a fully-actuated kinematic humanoid without any puppet strings. Taking this idea further suggests that we could even specify the trajectory using a very different system (e.g. human motion capture for an octopus puppet) provided we have a mapping between the configuration spaces. However, we do not elaborate on this idea in this paper.

We will discuss a two-dimensional marionette arm as an example of this technique. The arm is controlled by a single string attached at the end of the arm. The 2D arm is a highly nonlinear, under-actuated system. Trajectories for the arm are created by manually animating a normal two-dimensional arm. A Mathematica implementation of the optimization algorithm finds the best dynamically-feasible trajectory and its inputs. The results are verified on a hardware implementation of the system. This arrangement allows an animator to animate the dynamically-complex system without regards to the dynamics. Everything involving the projection and dynamics is fully automated.

II. Trajectory Optimization

In trajectory optimization, we have a 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):

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

[1]A *trajectory* is a set of continuous curves representing the states and inputs to a system over a time interval $[t_0, t_f]$ that may or may not satisfy the dynamics of the system. An *admissible trajectory* always satisfies the system dynamics.

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).

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

$$m(x_f) = (x_f - x_d(t_f))^T P_1 (x_f - x_d(t_f)) \quad (2b)$$

where Q, R, and P_1 are (possibly time-varying) positive definite matrices that weight the errors between the trajectory and the desired.

The search in (1) is constrained by the system's dynamics, preventing a standard gradient-descent approach. If we found a descent direction and added it to the current trajectory, 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).

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

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).

Seed η_0 = Initial Trajectory
for $i = 0, 1, 2, \dots$ **do**
  Find optimal descent direction
  $\zeta_i = \operatorname{argmin}_\zeta Dg(\eta_i) \circ \zeta + \frac{1}{2} q(\zeta, \zeta)$
  Check for terminal condition
  if $Dg(\eta_i) \circ \zeta = 0$ **then**
    break
  Scale step size to get a sufficient decrease
  $\gamma_i = \operatorname{argmin}_\gamma g(\eta_i + \gamma \zeta_i)$
  Update trajectory and project back into $\mathcal{T}$
  $\eta_{i+1} = \mathcal{P}(\eta_i + \gamma_i \zeta_i)$
return η_i

From this perspective, the algorithm is relatively straightforward. First we use a quadratic model of the cost function to find a descent direction ζ. If no direction gives a decrease, the current trajectory is a local minimum and the search ends. Otherwise, the descent direction is scaled using scalar γ_i to guarantee a sufficient decrease in cost. Finally, the current trajectory is added with the step direction and 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.

$$\begin{aligned} \mathcal{P} \ : \ \xi = (\alpha, \mu) &\rightarrow \eta = (x, u) \\ x(0) &= \alpha(0) \\ \dot{x} &= f(x(t), u(t)) \\ u(t) &= \mu(t) + K(t)(\alpha(t) - x(t)) \end{aligned} \tag{4}$$

where $f(x, u, t)$ is the system's dynamic function and $K(t)$ is a linear feedback control law that stabilizes the system over some domain (i.e. $K(t)$ does not have to be globally stabilizing). 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 linearized about the current trajectory and stabilized using LQR theory to find a 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. 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

$$\begin{aligned} D\mathcal{P}(\eta) : \gamma = (\beta(\cdot), \nu(\cdot)) &\rightarrow \zeta = (z(\cdot), x(\cdot)) \\ z(t_0) &= 0 \\ A(\tau) &= D_x f(\tau, x(\tau), u(\tau)) \\ B(\tau) &= D_u f(\tau, x(\tau), u(\tau)) \\ \dot{x}(\tau) &= A(\tau) z(\tau) + B(\tau) v(\tau) \\ v(\tau) &= \nu(\tau) + K(t)[\beta(\tau) - z(\tau)] \end{aligned} \tag{5}$$

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 [3] and [4].

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:

$$\zeta_i = \underset{\zeta}{\operatorname{argmin}} \, Dg(\xi_i) \circ \zeta + \tfrac{1}{2} q \circ (\zeta, \zeta) \tag{6}$$

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

The unconstrained optimization is transformed into a constrained optimization by restricting $\zeta \in T_{\xi_i}\mathcal{T}$. Since $T_{\xi_i}\mathcal{T}$ is a **linear subspace**, this constrained optimization is a linear optimal control problem that can be solved using linear optimal control tools rather than an iterative search.

The solution is found by solving a two-point boundary-valued problem (TPBVP). The TPBVP is solved either directly using modern numeric techniques [10] or as 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. An Armijo line search [8] finds a $\gamma \in (0, 1]$ that gives a sufficient, but not necessarily optimal, decrease.

$$\underset{i=0,1,2,\ldots}{\operatorname{argmin}} \ g(\xi_i + \gamma\zeta_i) < g(\xi_i) + \alpha\gamma Dg(\xi) \circ \zeta \qquad \gamma = \beta^i \tag{7}$$

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 (i.e. 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. Defining the Cost Function

The optimization requires that we choose the cost weighting matrices Q and R in (2). These are similar to cost weightings used in LQR theory. Weightings are chosen arbitrarily, but the mapping from the conceptual model to the dynamic

model suggests how to weight the optimization. Configuration variables that are common to both the conceptual and dynamic configuration spaces should be weighted relatively highly. Variables that do not appear in the conceptual model should be weighted relatively little. These are considered the "free variables" that the optimization algorithm can change as necessary to improve the trajectory.

IV. 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. 2. 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.

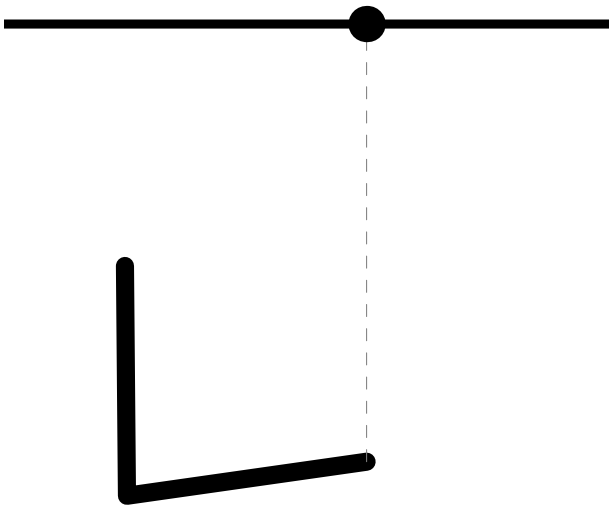


Fig. 2: Animations can be defined for a simple 2D arm and morphed into trajectories for this marionette arm.

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. The skeleton is animated without considering the marionette dynamics and limitations. An eight second waving motion was manually animated. The arm, starting from a lowered position, raises up, waves three times, and then lowers back down.

Mapping the conceptual arm to the marionette arm is trivial. The two joint angles are directly mapped to the same marionette angles and weighted heavily. The "extra" configuration variables are set to zero and weighted lightly. This mapping is used to created a desired trajectory from the animated arm.

Note that the desired trajectory does not have to be consistent with regards to the closed kinematic chain. The optimization will still work and result in a trajectory that is consistent.

An initial trajectory was generated by taking the initial conditions from the desired trajectory (modified to be satisfy the closed kinematic chain) and simulating the system with constant inputs. This results in a valid trajectory to seed the optimization. The optimization converged in 11 iterations and took nearly 4 hours[2].

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.

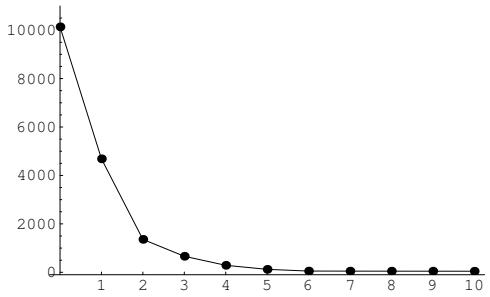


Fig. 3: The trajectory cost as a function of the iteration.

The results of the optimization can be seen in Fig. 4 and 5. Figure 4 shows that optimized trajectory matches the desired trajectory very well. Figure 5 shows the corresponding inputs[3] that were found by the optimization.

Fig. 6 shows several snapshots of the morphed trajectory compared to the desired trajectory. An animation of the results can be seen on our project website at *http://puppeteer.colorado.edu*

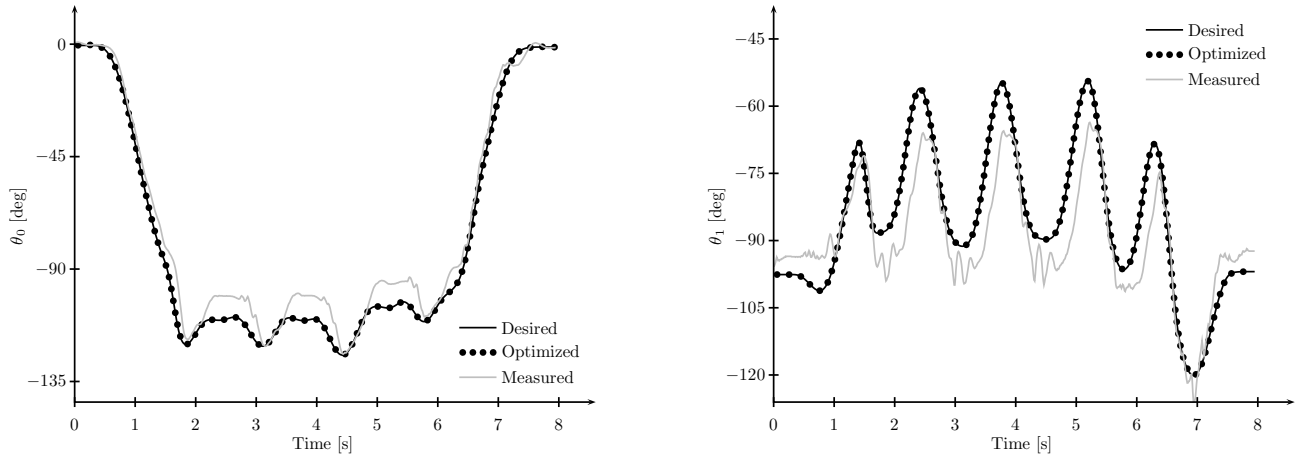
The optimization results were verified on a hardware implementation of the system. The results of the hardware test are also shown in Fig. 4 and 5. The arm follows the trajectory very well and produces a recognizable waving motion. Most of the error between the experiment and optimization results is due to a non-real-time controller implementation and imperfect friction model. Once both of these issues are addressed, the hardware platform should reproduce the optimization results even more faithfully.

V. CONCLUSIONS AND FUTURE WORKS

Trajectory optimization is a promising tool for animating complex mechanical systems. It frees the animator to use the more convenient conceptual models without dynamics 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

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

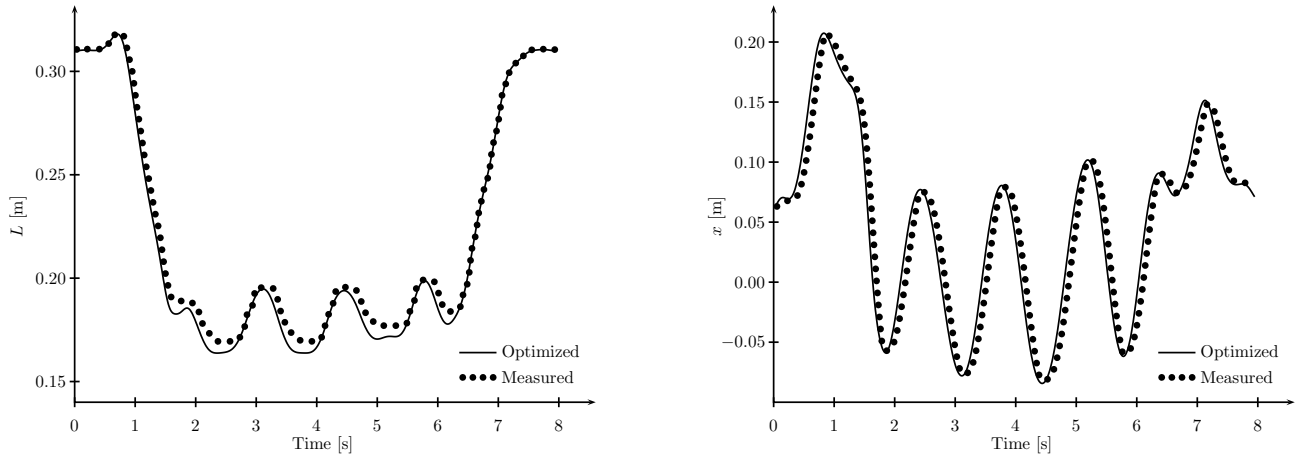
[3]Formally, the system inputs are actually the second time derivatives of the string length and position.



(a) Shoulder Angle

(b) Elbow Angle

Fig. 4: Plots of the desired, optimized, and measured arm angles. The optimized trajectories match the desired trajectories very well.



(a) String Length

(b) String Position

Fig. 5: Plots of the optimized and measured system inputs. The inputs were synthesized from scratch by the trajectory optimization.

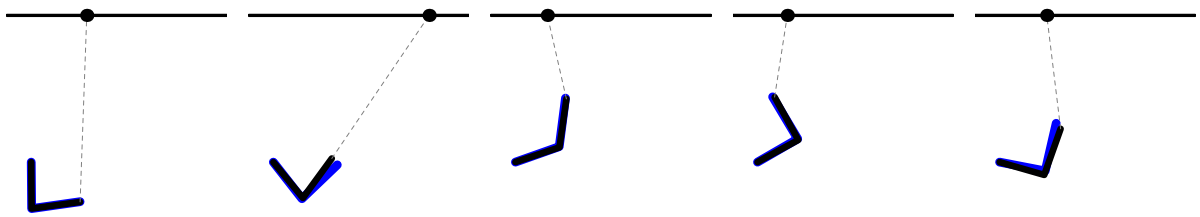


Fig. 6: Several frames from the resulting trajectory. The marionette and human models are drawn in black and blue, respectively. The complete animation can be seen at *http://puppeteer.colorado.edu*

designed, the process is essentially a black box that generates admissible trajectories and inputs from a desired animation.

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 is to extend the trajectory optimization algorithm 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.