Constructing and Implementing Motion Programs for Robotic Marionettes

Patrick Martin, *Student Member, IEEE,* Elliot Johnson, *Student Member, IEEE,* Todd Murphey, *Member, IEEE,* and Magnus Egerstedt, *Senior Member, IEEE*

Abstract

This paper investigates how to produce control programs for complex systems in a systematic manner. In particular, we present an abstraction-based approach to the specification and optimization of motion programs for controlling robot marionettes. The resulting programs are based on the concatenation of motion primitives and are further improved upon using recent results in optimal switch-time control. Simulations as well as experimental results illustrate the operation of the proposed method.

Index Terms

Dynamics; Optimal Control; Robotics

I. INTRODUCTION

When puppeteers compose plays, they break them down into components of varying granularity. For example, each act is composed of scenes and each scene is composed of puppet "routines" which, in turn, are comprised of simpler, individual motions, all of which have their own characteristics and duration. Previous work on robotic puppets has focused on the generation of puppet motions from human data. In particular, [18] investigated the transformation of motion capture data into motor controls for actuating a marionette. Their approach used a kinematic model to merge motion capture data with the geometry of the marionette. In [5], an alternative approach was taken where a number of mechanically different robotic marionettes can be controlled using computer issued motion commands as well as directly from a puppeteer interface mechanism. The research presented in this paper aims at transforming complex control tasks for robotic marionettes into strings of simpler control primitives. The objective behind this work is to be able to input high-level puppet motion programs, or *plays*, and then go from these plays to actual control laws for implementation on autonomous puppets, as shown in Figure 1. Some of the presented results are based on those in [7], [15], [16], and the main contribution in this paper is an integrated optimization framework; from the

P. Martin and M. Egerstedt are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA, e-mail: patrick.martin@gatech.edu, magnus@ece.gatech.edu.

E. Johnson and T. Murphey are with the Department of Mechanical Engineering, Northwestern University, Evanston, IL, 60208, e-mail: elliot.r.johnson@u.northwestern.edu, t-murphey@northwestern.edu

(a) Puppet in initial configuration. (b) Puppet in wave motion. (c) Puppet starting a walk. (d) The final step in the walk mode.

Fig. 1. An image sequence of a puppet executing a play consisting of a *wave* and a *walk* mode.

generation of suitable motion primitives all the way to optimized, executable control code. Technically speaking, the use of trajectory morphing for generating the appropriate motion primitives is new, while the remainder of the paper is based on building blocks developed in [7], [15], [16] but whose combination constitutes a systematic novel, approach to generating abstraction-based control programs.

We use Motion Description Languages (MDL) [4], [6], [11], [14] to formalize high-level specifications for puppetry. Specifically, a MDL is a string of symbols, each specifying what control law the system should be executing, together with an interrupt condition corresponding to the termination of this control law. (Alternative high-level specification languages include linear temporal logic [13], [17] and maneuver automata [3], [9].) In order for this language to be successful, it is important that it is expressive enough to be able to characterize actual puppet plays, and as such we draw inspiration from the way such plays are staged by professional puppeteers. In fact, the standard way in which puppet plays are described is through four parameters, namely *temporal duration*, *agent*, *space*, and *motion* (when?, who?, where?, and what?) [2], [8]. Most plays are moreover based on *counts* in that each puppet motion is supposed to happen at a particular count. (This becomes even more important if multiple puppets are acting simultaneously on stage or if the play is set to music). At each specified count, a motion is initiated, continued, or terminated. These are considerations that will be taken into account in the development of our control programs.

II. MOTION PRIMITIVES

Before one can start reasoning about motion strings, one first have to generate those motions. Such motions can, in principle, be generated using heuristics, but here we give a nonlinear optimal control approach for generating feasible motions for marionettes, based on so-called trajectory morphing. Here, one starts with a conceptual or source model, used to generate trajectories, and a dynamic or target model that we want to imitate the trajectories with. For the marionette, the conceptual model is usually a human being while the dynamic model is the marionette. Unlike the human, the marionette is relatively under-actuated and cannot directly apply torques to the joints. As such,

a basic problem becomes that of feasibility of trajectories. Standard gradient descent optimal control techniques generally will not produce feasible motions unless the descent directions are projected onto the trajectory manifold.

A. Trajectory Morphing Using Projection Operators

We use a projection-operator based approach to trajectory optimization [10] to project infeasible, dynamic motions (obtained from the source model) onto feasible dynamic motions. The returned trajectory is always admissible, even if the system dynamics are nonlinear, uncontrollable, or unstable. In particular, given 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

$$\arg\min_{\xi} h(\xi) = \int_{t_o}^{t_f} l(\tau, x(\tau), u(\tau))\mathrm{d}\tau + m(x(T))$$

where $\xi = (x(\cdot), u(\cdot))$ and $l(\dots)$ and $m(\dots)$ are incremental and terminal cost functions, typically quadratic.

Solutions to the equation above need to be constrained by the system's dynamics. A standard gradient-descent approach will not work if we naively find a descent direction and adjust the current trajectory with it, since the result typically would not be dynamically admissible. However, the constrained optimization can be converted to an unconstrained problem by introducing a projection operator.

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

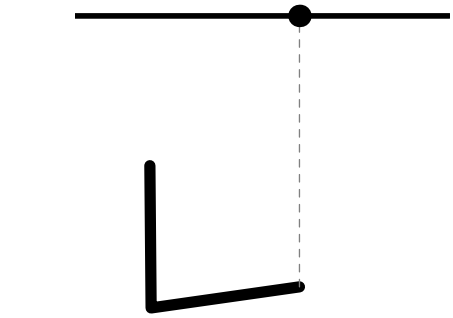
$$\arg\min_{\xi} g(\xi) = h(\mathcal{P}(\xi)).$$

If ξ locally minimizes this equation then the projected trajectory $\eta = \mathcal{P}(\xi)$ locally minimizes the original, constrained problem above. The projection operator consists of linearizing the equations of motion and constructing a stabilizing feedback law–this allows one to take an infeasible path (such as one that has discontinuities) and generates a smooth, feasible path. The key is that $\mathcal{P}(\cdot)$ is truly a projection; two or more applications of this mapping yields the same result as one (i.e., $\mathcal{P}((x_d(\cdot), u_d(\cdot))) = \mathcal{P}(\mathcal{P}((x_d(\cdot), u_d(\cdot))))$. Details of this construction can be found in [10].

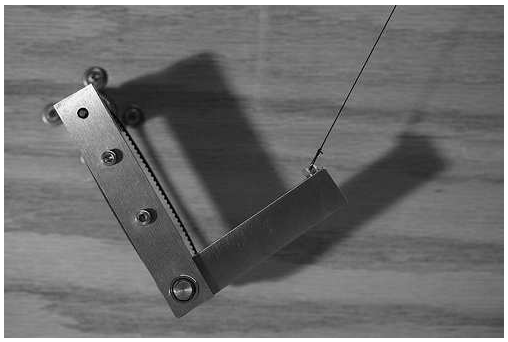
The resulting algorithm is relatively straightforward. First, a projection operator is designed for each new trajectory so that we can avoid requiring projection operators to be globally valid and instead just ask that they work in some neighborhood around a trajectory; this holds for any C^1 system. Next we find a descent direction at iteration k, ζ_k, 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 this trajectory is admissible, so the trajectory is projected back into the trajectory manifold as $\eta_{k+1} = \mathcal{P}(\eta_k + \gamma_k\zeta_k)$, where γ_k is the step size, e.g., based on the Armijo line search [1].

B. Results for a 2D Marionette Arm

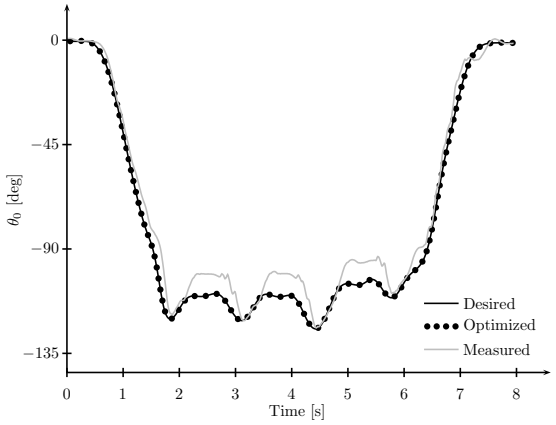
We now present a simple example–both in simulation and experiment–that illustrates the method just discussed. Consider the two-dimensional marionette arm shown in Figure 2(b). The upper and lower arms are free to rotate



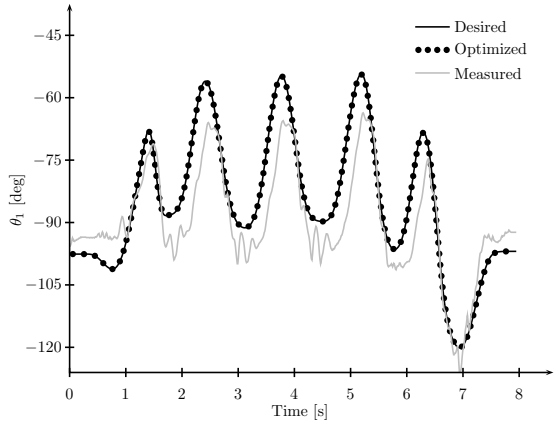
(a) Source Model



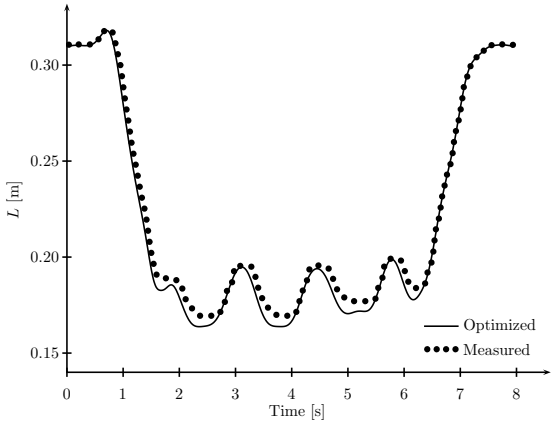
(b) Experimental Hardware



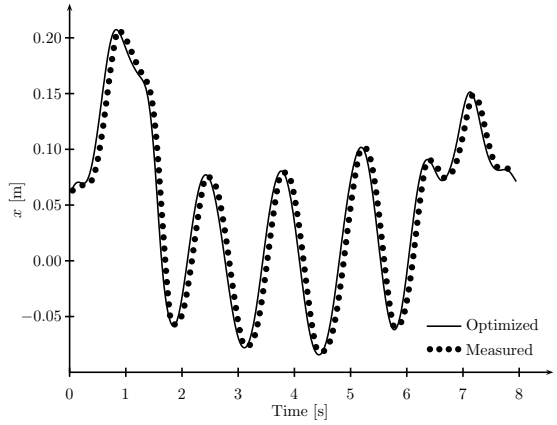
(c) Shoulder Angle



(d) Elbow Angle



(e) String Length



(f) String Position

Fig. 2. Animation-based source models (a) and experimental hardware setup (b). Plots of the desired, optimized, and measured shoulder (c) and elbow (d) angles as well as the corresponding system inputs (e,f). The optimized trajectories match the desired trajectories well.

about their respective joints. The shoulder is fixed while the arm is controlled by a single string tied to end of the arm. The other end of the string moves horizontally and the length of the string is also controllable. The dynamic model is defined using a mixed dynamic-kinematic model [12] to include the strings as kinematic inputs.

The source model trajectories are generated using the Blender[1] computer animation package. This software allows us to create a skeleton that represents a two-link, two-dimensional human arm and to animate the skeleton (Figure 2(a)). A 10 second waving motion was generated with the arm starting from a neutral lowered position. It lifts up, waves for three periods, and then lowers back down to the original position.

The results of the optimization can be seen in Figures 2(c)-2(f). Figures 2(c) and 2(d) show that optimized trajectory matches the desired trajectory very well. Figures 2(e) and 2(f) show the corresponding inputs that were found by the optimization. The optimization results were verified on a hardware implementation of the system and the arm follows the trajectory well and produces a recognizable waving motion.

The motions generated by this approach enable us to create a library of possible motions the puppet can execute. The next section addresses the concatenation of these motions so that entire puppet plays can be described and, consequently, optimized.

III. MOTION DESCRIPTION LANGUAGES

Given that we can describe the marionette dynamics by

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

the motions obtained in the previous section can be encoded by a (possibly time varying) feedback law $u = \kappa(x, t\alpha)$. Here α is a motion scaling parameter that encodes the "intensity" of the motion, e.g., the difference between a fast and a slow wave motion is given by α rather than by κ.

In order to concatenate together strings of such control laws, we need some further inspiration from actual puppeteering. In fact, a puppet play is typically described with four important properties: *who* should act, *what* motion should they do, *where* should they operate, and *when* should the action occur, e.g., [2], [8]. In this paper, we only investigate the motion of individual marionettes and, as such, we ignore the "who" component of the specification.

Assume that we have created a library of M control laws, denoted by $\mathcal{K}^i = \{\kappa_1, \cdots, \kappa_M\}$. This captures the "what" component. Moreover, we assume that the "stage" is divided into l different sections (typically this number is 9, namely LowerLeft, LowerCenter, LowerRight, MiddleLeft, MiddleCenter, MiddleRight, UpperLeft, UpperCenter, UpperRight), with the set of regions being given by $\mathcal{R} = \{r_1, \ldots, r_l\}$, which gives us the "where". Finally, the counts specify the timing (the "when") of each puppet motion; thus, we let the length of each puppet action be denoted by $\tau \in \mathbb{R}^+$.

Based on this discussion, we can define a Motion Description Language for puppetry plays from symbols of the form $(\kappa, \alpha, r, \tau)$. This symbol specifies that the puppet executes the control law κ, with scaling α, within region r

[1] *http://www.blender.org,* 2007.

for τ seconds. Assuming that the puppet system starts at time t_0, the following MDL string

$$(\kappa_1, \alpha_1, r_1, \tau_1)(\kappa_2, \alpha_2, r_1, \tau_2) \cdots (\kappa_M, \alpha_M, r_1, \tau_M)$$

causes the evolution of the puppet to be

$$\dot{x} = \begin{cases} f(x, \kappa_1(x, t, \alpha_1)), \ t \in (t_0, \tau_1) \\ f(x, \kappa_2(x, t, \alpha_2)), \ t \in (\tau_1, \tau_2) \\ \vdots \\ f(x, \kappa_M(x, t, \alpha_M)), \ t \in (\tau_{M-1}, \tau_M) \end{cases}$$

within region r_1. It should be noted that we here only use MDL strings of finite length which ensures the absence of Zeno-type phenomena involving infinitely many mode transitions in finite time.

A. Optimizing MDL Strings

Now that an appropriate specification language for puppet plays is selected we can turn to computing optimal timing and scaling parameters, τ and α, respectively, based on the developments in [15], [16], that optimizes over MDL strings by adjusting the interrupt times as well as the energy scaling parameters.

Rather than solving a large-scale problem explicitly, we start with a canonical, two-primitive MDL string. In fact, consider the following optimal control problem

$$\min_{\tau_1, \alpha_1, \alpha_2} J(\tau_1, \alpha_1, \alpha_2) = \int_{t_0}^{t_f} L(x, t)dt + C_1(\alpha_1) + C_2(\alpha_2) + \Delta(\tau_1) + \Psi_1(x(\tau_1)) + \Psi_2(x(t_f)),$$

subject to

$$\dot{x} = \begin{cases} f(x, \kappa_1(x, t, \alpha_1)), \ t \in (t_0, \tau_1) \\ f(x, \kappa_2(x, t, \alpha_2)), \ t \in (\tau_1, t_f) \end{cases}$$
$$x(t_0) = x_0,$$

where $t_f = \tau_1 + \tau_2$. This optimal control problem is the atomic problem involving how to execute the two-instruction play $(\kappa_1, \alpha_1^{nom}, r_1, \tau_1^{nom}), (\kappa_2, \alpha_2^{nom}, r_2, \tau_2^{nom})$ under the following interpretations:

- $\Delta(\cdot)$: function that penalizes deviations from a nominal switch time, τ_1^{nom}
- $C_j(\cdot)$: function that measures how much energy it takes to use parameter $\alpha_j, \ j = 1, 2$ as well as possibly deviations from the nominal α_j^{nom}.
- $\Psi_1(\cdot)$: function that ensures that the puppet is close to r_1 at time τ
- $\Psi_2(\cdot)$: function that ensures that the puppet is close to r_2 at time t_f
- $L(\cdot, \cdot)$: function that may be used to ensure that the desired reference trajectory is followed.

Applying calculus of variations techniques to the cost function results in the following optimality conditions

$$\frac{\partial J}{\partial \tau} = \lambda(\tau_1^-) f(x, \kappa_1(x, \tau_1, \alpha_1)) - \lambda(\tau_1^+) f(x, \kappa_2(x, \tau_1, \alpha_2)) + \frac{\partial \Delta}{\partial \tau_1} + \frac{\partial \Psi_1}{\partial x}^T f(x, \kappa_2(x, \tau_1, \alpha_2)) = 0$$
$$\frac{\partial J}{\partial \alpha_2} = \mu(\tau_1^+) = 0$$
$$\frac{\partial J}{\partial \alpha_1} = \mu(t_0) = 0,$$

where the co-states λ and μ satisfy the following discontinuous (backwards) differential equations:

$$\begin{aligned}
\dot{\lambda} &= -\frac{\partial L}{\partial x} - \lambda \frac{\partial f_2}{\partial x},\ t \in (\tau_1, t_f), \quad \lambda(t_f) = \frac{\partial \Psi_2}{\partial x}(x(t_f)) \\
\dot{\lambda} &= -\frac{\partial L}{\partial x} - \lambda \frac{\partial f_1}{\partial x},\ t \in [0, \tau_1), \quad \lambda(\tau_1^-) = \lambda(\tau_1^+) + \frac{\partial \Psi_1}{\partial x}(x(\tau_1)) \\
\dot{\mu} &= \lambda \frac{\partial f_2}{\partial \alpha_2},\ t \in (\tau_1, t_f), \quad \mu(t_f) = \frac{\partial C_2}{\partial \alpha_2} \\
\dot{\mu} &= \lambda \frac{\partial f_1}{\partial \alpha_1},\ t \in [0, \tau_1), \quad \mu(\tau_1^-) = \frac{\partial C_1}{\partial \alpha_1}.
\end{aligned}$$

where f_1 and f_2 denote the system dynamics under the influence of κ_1 and κ_2, respectively.

Since the switch times are decoupled, these optimality conditions can be directly extended to cover plays with more than two modes. As such, this construction allows us to produce a compiler that takes *plays* and outputs *strings of control modes with an optimized temporal duration and mode parameterization*, as given in the algorithm below:

```
Algorithm
Given M modes:
(i, κ_1, α_1^nom, r_1, τ_1^nom)(κ_2, α_2^nom, r_2, τ_2^nom) ··· (κ_M, α_M^nom, r_M, τ_M^nom)
Set τ_i(0) = τ_i^nom, i = 1, ..., M − 1
Initialize α_i(0) = α_i^nom, i = 1, ..., M
Optimization (k = 0)
 Repeat
  Compute x(t) forwards using α_i(k), τ_i(k)
  Compute λ(t), μ(t) backwards
  Compute ∂J/∂τ_i, ∂J/∂α_i
  Gradient Descent
   τ_i(k + 1) = τ_i(k) − γ(k) ∂J/∂τ_i (τ_i(k)), i = 1, ..., M − 1
   α_i(k + 1) = α_i(k) − ℓ(k) ∂J/∂α_i (α_i(k)), i = 1, ..., M
  Set k = k + 1
 Until ‖∇J‖ ≤ δ
```

In this algorithm $\gamma(k)$ and $\ell(k)$ denote the step-sizes for the τ and α parameters. Since the numerical values for τ and α can differ significantly, it is appropriate to use separate step-sizes. In the next section we discuss using this algorithm to generate control code for a particular puppet hardware platform.

IV. EXPERIMENTAL RESULTS

To illustrate how the MDL compilation process and puppet platform operate, we formulate an example puppet play and compile if for our target platform. We implemented two modes `waveLeft` and `walk` using combinations of sinusoidal inputs. Say we have the following MDL script

$$(\texttt{waveLeft}, 1, r_1, 4)(\texttt{walk}, 1, r_1, 6),$$

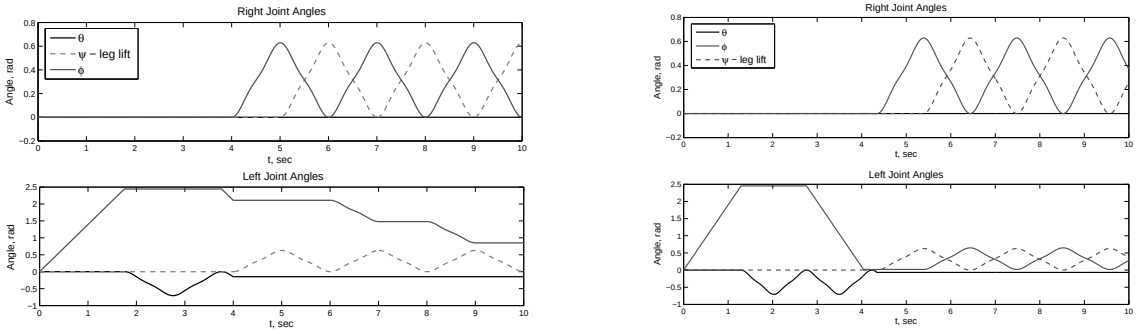
where $\alpha_1^{nom} = \alpha_2^{nom} = 1$ and $\tau_1^{nom} = 4$ and $\tau_2^{nom} = 6$. We optimize this two-mode motion program using the following cost function:

$$J(\tau, \alpha_1, \alpha_2) = \int_0^{10} x^T P x \; dt + \rho(\tau^0 - \tau)^2 + w_1\alpha_1^2 + w_2\alpha_2^2,$$

where $x = [\theta_r \;\; \phi_r \;\; \theta_l \;\; \phi_l \;\; \psi_r \;\; \psi_l]^T$ gives the configuration of the puppet, with the angles of arm rotation being θ_l and θ_r, where the subscripts denote left and right, respectively. The arm lift angles are ϕ_l and ϕ_r, and the leg lift angles are given by ψ_l and ψ_r. (Note that for the purpose of MDL optimization, the algorithm relies on the ability to simulate the system forward (the state) and backward (the co-state) in time in order to compute the gradient direction. As this is an iterative process, we need to do this repeatedly, which leads us to use a *kinematic* model with less degrees of freedom than a full-blown dynamic model in order to enable switched-time optimization in a computationally tractable manner.)

In the cost function, we weight the joint angle trajectories of the puppet, q, with a diagonal matrix P. The switch time, τ, is penalized from deviations from its nominal value with some $\rho > 0$. Moreover, the energy parameters are penalized with squared terms combined with constant weights, $w_1 > 0$ and $w_2 > 0$.

Before optimizing the MDL string, we simulate the system with its nominal motion program, resulting in the trajectories shown in Figure 3(a). The plot shows that this behavior will generate an odd motion, with the left arm (angle ϕ in bottom graph) hovering in the air while a walk motion executes. This behavior would look more natural if the lifting angle of the left arm was closer to the side of the puppet before initializing its walk behavior.



(a) (b)

Fig. 3. Plots of the joint angles for left and right limbs during the execution of the original 3(a) and optimized 3(b) MDL strings.

Using this design goal, we weight the penalty matrix P in so that the lifting angle should be close to its initial configuration, i.e. hands at both sides. After compilation of this nominal motion program and its execution in simulation, the resulting trajectories (Figure 3(b)) show the left hand being lowered in time for a walk motion. The termination of the compilation process returned the improved values $\tau = 4.3423$, $\alpha_1 = 1.3657$, $\alpha_2 = 0.9566$, and final cost $J = 30.0684$.

The optimized "play" is ported onto the actual platform and the execution of this particular optimal "wave and walk" play discussed in the previous paragraphs is shown in Figure 1. There it can be seen that the optimized MDL does in fact not leave the arm hanging half-way in between motions, as was the case for the nominal, non-optimized play.

V. Conclusions

In this paper we presented the a motion description language for specifying and encoding autonomous puppetry plays in a manner that is faithful to standard puppetry choreography. The resulting strings of control-interrupt pairs are then compiled in the sense that they are parsed by a dynamical system that produces optimized, hybrid control laws corresponding to strings of motions, locations, and temporal durations for each motion primitive. This paper also discusses some issues arising in modeling and how to capture relevant motion primitives from empirical data. Experimental results illustrate the viability of the proposed approach. There is still need for scalability of these results. In particular, scaling the techniques presented here is nontrivial, as it requires simulation of the nonlinear and linearized equations of motion. This is part of our future work.

Acknowledgment

This work is supported by the National Science Foundation under grant # IIS-0757378. We also acknowledge the helpful comments of Jon Ludwig at the Center for Puppetry Arts in Atlanta, Georgia.

References

[1] L. Armijo. Minimization of Functions Having Lipschitz Continuous First-Partial Derivatives. *Pacic Journal of Mathematics*, Vol. 16, pp. 1-3, 1966.

[2] B. Baird. *The Art of the Puppet*. Mcmillan Company, New York, 1965.

[3] A. Bicchi, A. Marigo, and B. Piccoli. Encoding Steering Control with Symbols. *IEEE Conference on Decision and Control*, Maui, Hawaii, Dec. 2003.

[4] R.W. Brockett. On the Computer Control of Movement. In the *Proceedings of the 1988 IEEE Conference on Robotics and Automation*, pp. 534–540, New York, April 1988.

[5] I.M. Chen, R. Tay, S. Xing, and S.H. Yeo. Marionette: From Traditional Manipulation to Robotic Manipulation. *IEEE Robotics and Automation Magazine*, Vol. 12, No. 1, pp. 59-74, 2005.

[6] M. Egerstedt and R.W. Brockett. Feedback Can Reduce the Specification Complexity of Motor Programs. *IEEE Transactions on Automatic Control*, Vol. 48, No. 2, pp. 213–223, Feb. 2003.

[7] M. Egerstedt, T.Murphey, and J. Ludwig. Motion Programs for Puppet Choreography and Control. *Hybrid Systems: Computation and Control*, Springer-Verlag, pp. 190-202, Pisa, Italy April 2007.

[8] L. Engler and C. Fijan. *Making Puppets Come Alive*. Taplinger Publishing Company, New York, 1973.

[9] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries. *IEEE Transactions on Robotics*, Vol. 21, No. 6, pp. 1077–1091, Dec. 2005.

[10] J. Hauser, A Projection Operator Approach to Optimization of Trajectory Functionals. In *IFAC World Congress*, Barcelona, Spain, 2002.

[11] D. Hristu-Varsakelis, M. Egerstedt, and P.S. Krishnaprasad. On The Structural Complexity of the Motion Description Language MDLe. *IEEE Conference on Decision and Control*, Maui, Hawaii, Dec. 2003.

[12] E. Johnson and T. Murphey, Dynamic Modeling and Motion Planning for Marionettes: Rigid Bodies Articulated by Massless Strings. In *International Conference on Robotics and Automation*, pp. 330 - 335, April 2007.

[13] M. Kloetzer and C. Belta. Hierarchical Abstractions for Robotic Swarms. *IEEE International Conference on Robotics and Automation*, Orlando, FL, 2006

[14] V. Manikonda, P.S. Krishnaprasad, and J. Hendler. Languages, Behaviors, Hybrid Architectures and Motion Control. In *Mathematical Control Theory*, Eds. Willems and Baillieul, pp. 199–226, Springer-Verlag, 1998.

[15] P. Martin and M. Egerstedt. Optimal Timing Control of Interconnected, Switched Systems with Applications to Robotics Marionettes. *Journal of Discrete Event Dynamic Systems.* To appear 2009.

[16] P. Martin and M. Egerstedt. Optimal Timing Control of Robotic Marionettes. *Workshop on Discrete Event Systems*, Gothenburg, Sweden, May 2008.

[17] P. Tabuada and G. Pappas. Linear Time Logic Control of Discrete-Time Linear Systems. Accepted for publication in *IEEE Transactions on Automatic Control*.

[18] K. Yamane, J.K. Hodgins, and H.B. Brown: ”Controlling a Marionette with Human Motion Capture Data,” In *International Conference on Robotics and Automation*, Volume 3, pp. 3834 - 3841, September 2003.