PuppetMaster: Robotic Animation of Marionettes

SIMON ZIMMERMANN, ETH Zurich
ROI PORANNE, ETH Zurich and University of Haifa
JAMES M. BERN, ETH Zurich
STELIAN COROS, ETH Zurich

We present a computational framework for robotic animation of real-world string puppets. Also known as marionettes, these articulated figures are typically brought to life by human puppeteers. The puppeteer manipulates rigid handles that are attached to the puppet from above via strings. The motions of the marionette are therefore governed largely by gravity, the pull forces exerted by the strings, and the internal forces arising from mechanical articulation constraints. This seemingly simple setup conceals a very challenging and nuanced control problem, as marionettes are, in fact, complex coupled pendulum systems. Despite this, in the hands of a master puppeteer, marionette animation can be nothing short of mesmerizing. Our goal is to enable autonomous robots to animate marionettes with a level of skill that approaches that of human puppeteers. To this end, we devise a predictive control model that accounts for the dynamics of the marionette and kinematics of the robot puppeteer. The input to our system consists of a string puppet design and a target motion, and our trajectory planning algorithm computes robot control actions that lead to the marionette moving as desired. We validate our methodology through a series of experiments conducted on an array of marionette designs and target motions. These experiments are performed both in simulation and using a physical robot, the human-sized, dual arm ABB YuMi® IRB 14000.

CCS Concepts: • **Theory of computation → Nonconvex optimization**; • **Computing methodologies → Physical simulation**; • **Computer systems organization → Robotic control**;

Additional Key Words and Phrases: Computer graphics, robotics, puppeteering, sensitivity analysis

ACM Reference Format:
Simon Zimmermann, Roi Poranne, James M. Bern, and Stelian Coros. 2019. PuppetMaster: Robotic Animation of Marionettes. *ACM Trans. Graph.* 38, 4, Article 103 (July 2019), 11 pages. https://doi.org/10.1145/3306346.3323003

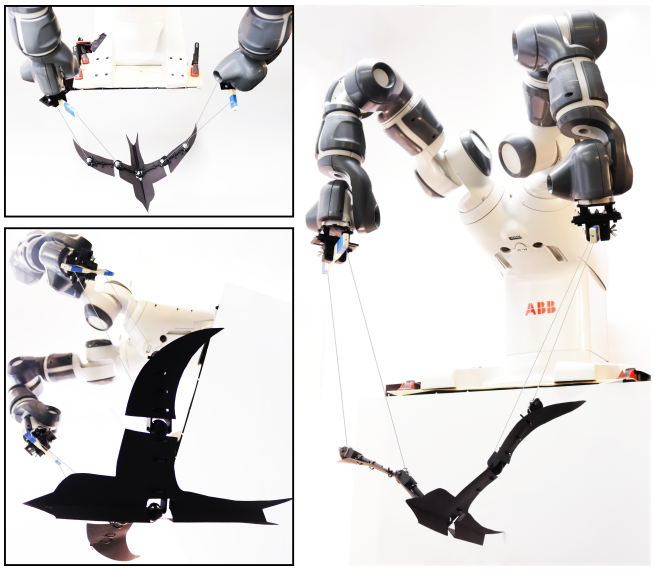


Fig. 1. A robot (ABB YuMi® IRB 14000) controlling a bird-shaped, string-driven marionette. This type of puppet is notoriously difficult to animate. Our control framework generates optimal motion trajectories for the robot puppeteer such that the marionette performs a user-specified motion.

1 INTRODUCTION

Marionettes are articulated, string-actuated puppets that have provided a medium for animation in performance arts since Ancient Greece. In the hands of a skilled puppeteer, marionettes produce motions that are incredibly expressive, fluid and compelling. However, their deceptively natural movements conceal the fact that marionettes are very challenging to control. Marionettes are underactuated, high-dimensional, highly non-linear coupled pendulum systems. They are driven by gravity, the tension forces generated by a small number of cables, and the internal forces arising from mechanical articulation constraints. As such, the map between the actions of a puppeteer and the motions performed by the marionette is notoriously unintuitive, and mastering this unique art form takes unfaltering dedication and a great deal of practice.

With the long term goal of endowing robots with human-level dexterity when it comes to manipulating complex physical systems, we present *PuppetMaster* – a physics-based motion planning framework for robotic animation of marionettes. As illustrated in Fig. 2, the input to our computational puppeteering system consists of: 1) a kinematic description of the robot puppeteer (i.e. the hierarchical arrangement of actuators and rigid links); 2) the design of the marionette, which includes the articulated puppet itself, the control handles that the robot will be manipulating, as well as the strings that attach the puppet to the handles; and 3) a target motion that the marionette should aim to reproduce. Our core technical contribution is a novel trajectory optimization method built upon a physics simulator using an implicit time-stepping scheme. To compute derivatives of the system's forward dynamics, we show how to apply sensitivity analysis techniques to *entire motion trajectories*, as well as how to effectively exploit the specific sparsity structure imposed by the time domain. Our mathematical model forms the

Authors' addresses: Simon Zimmermann, ETH Zurich; Roi Poranne, ETH Zurich and University of Haifa; James M. Bern, ETH Zurich; Stelian Coros, ETH Zurich.

0730-0301/2019/7-ART103 $15.00
https://doi.org/10.1145/3306346.3323003

basis of a general, unified framework that enables optimization algorithms to concurrently reason about: the robot puppeteer's workspace (i.e. the space of reachable configurations for its end effectors) and movements; the limited and indirect control provided by string-based actuation setups; the dynamic motions generated by the puppet in response to the robot's actions; and continuous design parameters such as the lengths of the strings and their points of attachment on the control handles. The output generated by our planning framework consists of optimal motion trajectories that are specified in the configuration space of the robot puppeteer.

To evaluate the efficacy of our robotic puppeteering framework, we designed a set of experiments of increasing complexity. These experiments include different types of under-actuated dynamical systems that we wish a robot to skillfully control, and a varied array of motion-based task specifications. To assess the degree to which simulation results carry over to the real world, we further fabricated several physical prototypes that are puppeteered by an ABB YuMi – a human-sized, dual-arm robot.

2 RELATED WORK

The study of motion has been a core research topic in computer graphics since the field's very beginning. Nevertheless, animation predates computers by thousands of years: since ancient times, children and adults alike have been fascinated by physical systems – mechanical automatons, animatronic figures, robotic creatures, puppets and marionettes – that are designed to generate natural movements. In recent years, these types of physical animation devices have started to receive considerable attention from the computer graphics community. We have witnessed, for example, computational approaches to designing mechanical toys [Coros et al. 2013; Song et al. 2017; Thomaszewski et al. 2014; Zhang et al. 2017], physical characters that produce motions by virtue of precisely controlled deformations [Bern et al. 2017; Gauge et al. 2014; Megaro et al. 2017; Skouras et al. 2013; Xu et al. 2018], robotic creatures that walk [Megaro et al. 2015; Schulz et al. 2017], fly [Du et al. 2016] or even roller blade [Geilinger et al. 2018], etc. Each of these application domains demands techniques that are tailored to the unique challenges and characteristics of different types of physical systems. The problem that we tackle in this work, namely robotic manipulation of marionettes, is closest to the work of Skouras and colleagues [Skouras et al. 2013]. However, while they address the problem of designing *quasi-static* movements, our goal is to generate *highly dynamic* motion trajectories for string-driven physical systems using dexterous robots as puppeteers.

In the early days of computer graphics, animation techniques relied largely on interpolation of artist-prescribed key-frames. With such an approach, the onus is on the animator to manually create the illusion that a character's motions obey the laws of physics. The tedious and error-prone nature of this process led to the sub-field of physically based character animation, which has generated a rich body of research over the past three decades. Our work draws inspiration from techniques that formulate motion synthesis as optimization problems [Cohen 1992; Witkin and Kass 1988]. In these formulations, equations of motion based on Newton's second law are incorporated as constraints, resulting in motions that look physically correct. However, in favor of optimization problems that are easier to solve, these constraints are often treated in a soft manner, resulting in animations where external fictitious forces are merely minimized [Barbič et al. 2009; Pan and Manocha 2018; Schulz et al. 2014]. Given that we wish simulation results to carry over to the real world, we instead develop a trajectory optimization technique that guarantees Newton's second law of motion is always satisfied, even for *intermediate* results, up to unavoidable discretization errors introduced by numerical integration schemes.

Dexterous manipulation of complex physical systems is another topic of research in computer animation that is relevant to our work. In particular, the trajectory optimization formulation we introduce complements recent model-based [Bai et al. 2016; Clegg et al. 2015] and learning-based [Clegg et al. 2018] techniques developed for manipulation of cloth and clothing. Our technique leverages derivatives of the physics simulation which we use to compute a marionette's motions. In this respect, we borrow concepts from recent differentiable simulators for rigid bodies [Todorov 2014]. However, in our formulation, first and second derivatives are computed via sensitivity analysis, and they can be taken with respect to the robot's actions, or with respect to continuous parameters that define the marionette's design. First order Sensitivity analysis, in both its direct and adjoint form [McNamara et al. 2004], is often used to compute gradients for steady-state [Auzinger et al. 2018] or quasi-static problems that are governed by force equilibrium [Ly et al. 2018; Pérez et al. 2017]. In this paper, we show how to extend this very useful optimization technique to second order, for trajectories computed using forward dynamics simulations, and how to exploit the specific structure imposed by the time domain to increase the computational efficiency of the motion optimization process.

For the history and engineering aspects of traditional marionette design and manipulation, we refer the interested reader to [Chen et al. 2004], and we note that the challenge of controlling marionette motions has been studied before. For example, sharing our vision, Murphey and his collaborators developed dynamic models tailored to marionettes [Johnson and Murphey 2007]. Building on the technique described in [Hauser 2002], they also formulated trajectory optimization as unconstrained problems [Murphey and Johnson 2011; Schultz and Murphey 2012]. This was achieved by defining a projection operator that takes a candidate, non-physical trajectory and finds a valid one. The operator is introduced in the optimization problem as an objective, making constraints redundant. However, computing the gradient of this new objective demands an optimization problem to be solved. Our method on the other hand provides analytic expressions for both first and second derivatives. Furthermore, while previous work focuses largely on simple motions (e.g. treating the marionette as a simple suspended mass [Murphey and Johnson 2011]) or transferring the motions of a human puppeteer onto a robot [Yamane et al. 2004], we demonstrate a diverse array of marionette motions that are generated automatically.

It is also worth noting the connection between marionette animation and the transport of cable-suspended payloads using cranes [Zameroski et al. 2006] or helicopters [Bernard and Kondak 2009; Bisgaard et al. 2009]. In this context, the main focus is the handling of oscillations in high speed, rest-to-rest motions. Trajectory optimization approaches have been proposed to tackle this challenge, as

seen for example in [Sreenath et al. 2013] and [Sreenath and Kumar 2013], the latter of which focuses on collaborative transportation via multi-drone systems. Methods based on reinforcement learning have also been successfully developed [Crousaz et al. 2014; Faust et al. 2013; Palunko et al. 2013], and they present an interesting alternative to the model-based marionette control methodology we investigate in this paper.

3 OVERVIEW

As illustrated in Fig. 2, our control framework takes as an input a kinematic description of a robot puppeteer, the design of a string-driven marionette, and a target motion. We visualize the marionette as a virtual stick figure, and use two distinct visual styles, as seen in the figure, to portray the motion of the puppet, represented by a vector $\mathbf{x}$ which encodes the trajectory for each point mass, and the target motion $\hat{\mathbf{x}}$. Strings are used to attach the marionette to rigid handles that are directly manipulated by the robot. We refer to the points of attachment as $\mathbf{p}$, and they control the pull forces applied to the marionette through each string. The output of our system consists of choreographed motions for the robot puppeteer, which we encode using joint angle trajectories $\mathbf{q}$. We refer the reader to the accompanying video for demonstrations and results.

We solve the robotic puppeteering control problem with a second-order trajectory optimization method that exploits derivatives of motions generated by forward dynamics simulation. These derivatives are computed using sensitivity analysis, as described in Section 4. By exploiting the specific structure imposed by the time domain, we formulate a computationally-efficient trajectory optimization algorithm. In Section 5 we discuss the simulation model we use for our marionettes. Finally, in Section 6 we show experimental results, starting with simple systems such as a pendulum and a double pendulum, and ending with animal-inspired marionette designs.

4 PRELIMINARIES

The control problem we seek to solve is characterized by two sets of variables: *state* variables $\mathbf{x}$, and *control* variables $\mathbf{p}$, which represent the motion trajectories of the marionette and of the robot, respectively. The motion of the marionette is governed by the robot's actions and by its own non-linear dynamics. Without loss of generality, we can therefore express $\mathbf{x}$ as an implicit function of $\mathbf{p}$. While $\mathbf{x}(\mathbf{p})$ does not have a closed-form expression (we must compute it through numerical simulation), the dependency between $\mathbf{x}$ and $\mathbf{p}$ is captured by the relation

$$\mathbf{G}(\mathbf{x}, \mathbf{p}) = 0, \tag{1}$$

which, as we will discuss shortly, encodes Newton's second law of motion for entire trajectories. Our control problem can therefore be formulated as:

$$\min_{\mathbf{p}} \quad O(\mathbf{x}(\mathbf{p}), \mathbf{p}), \tag{2}$$

where $O(\mathbf{x}(\mathbf{p}), \mathbf{p})$ is an objective function that quantifies, for example, the degree to which the marionette's motion matches its target, and the smoothness of the control actions. This problem formulation is very generic. For example, if $\mathbf{x}$ was defined as the configuration of an elastic object, $\mathbf{p}$ as the set of parameters that define its rest shape and $\mathbf{G}(\mathbf{x}, \mathbf{p})$ as the sum of internal and external forces, then Eq. 2 would describe an inverse elastic shape design problem [Ly et al. 2018; Megaro et al. 2017; Pérez et al. 2017]. First order sensitivity analysis has become a standard go-to technique for solving such problems, and it is therefore also suitable for our robotic marionette control task. Briefly, sensitivity analysis provides an analytic expression for the Jacobian $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}}$ and it therefore enables the use of first order gradient-based methods (e.g. L-BFGS) to minimize Eq. 2. To provide some intuition, for our problem setting, this Jacobian encodes the way in which an *entire motion trajectory* $\mathbf{x}$ changes as a function of the control actions $\mathbf{p}$.

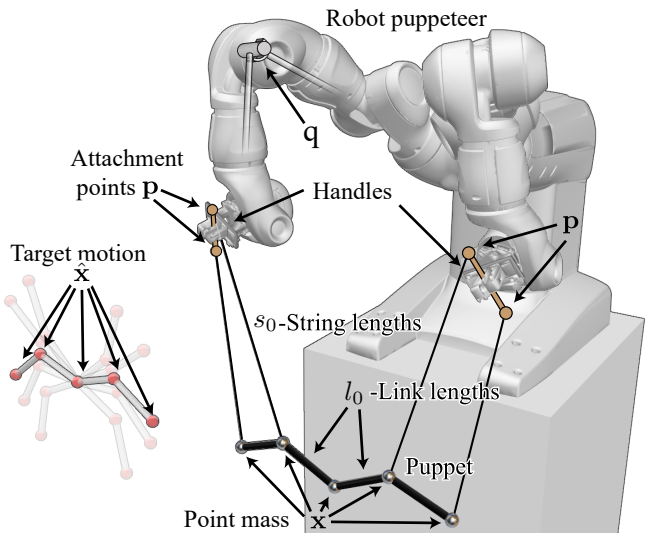


Fig. 2. An overview of the main components of our system, which takes as input a kinematic description of the robot puppeteer, a marionette design and a target motion. The output of our control framework consists of precisely choreographed motions for the robot puppeteer.

First Order Sensitivity Analysis. Sensitivity analysis begins by applying the chain rule on $O(\mathbf{x}(\mathbf{p}), \mathbf{p})$:

$$\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}} = \frac{\partial O}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}} + \frac{\partial O}{\partial \mathbf{p}}, \tag{3}$$

The analytic expression for the *sensitivity* term $\mathbf{S} := \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}}$ can be found using the fact that $\mathbf{G}(\mathbf{x}, \mathbf{p})$ is always zero (i.e. we assume that for *any* $\mathbf{p}$ we can compute $\mathbf{x}(\mathbf{p})$ such that Eq. 1 is satisfied), which implies:

$$\frac{\mathrm{d}\mathbf{G}}{\mathrm{d}\mathbf{p}} = \frac{\partial \mathbf{G}}{\partial \mathbf{x}}\mathbf{S} + \frac{\partial \mathbf{G}}{\partial \mathbf{p}} = \mathbf{0}. \tag{4}$$

By rearranging this equation, we get:

$$\mathbf{S} = -\left(\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right)^{-1} \frac{\partial \mathbf{G}}{\partial \mathbf{p}}, \tag{5}$$

and plugging into (3), we obtain:

$$\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}} = -\frac{\partial O}{\partial \mathbf{x}} \left(\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right)^{-1} \frac{\partial \mathbf{G}}{\partial \mathbf{p}} + \frac{\partial O}{\partial \mathbf{p}}. \tag{6}$$

We note that through a reordering of matrix multiplications, the well-known adjoint method [Cao et al. 2003] avoids computing $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}}$

directly as it evaluates $\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}}$. This is oftentimes more computationally efficient. However, as we will show next, we can leverage $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}}$ to derive a second-order, generalized Gauss-Newton solver that exhibits much better convergence properties than first order alternatives.

Second Order Sensitivity Analysis. In our first experiments we observed that first order optimization, i.e. L-BFGS, converges far too slowly. As an alternative, Newton's method requires the Hessian $\frac{\partial^2 O}{\partial \mathbf{p}^2}$, which can be derived using second order sensitivity analysis. To begin with, we differentiate (3):

$$\frac{\mathrm{d}^2 O}{\mathrm{d}\mathbf{p}^2} = \frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}} = \frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\left(\frac{\partial O}{\partial \mathbf{x}}\mathbf{S}\right) + \frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\frac{\partial O}{\partial \mathbf{p}}. \tag{7}$$

The formulas above involve third-order tensors, which lead to notation that is slightly cumbersome. To simplify our exposition, we treat tensors as matrices and assume that contractions are clear from context. For a conceptually similar but more formal derivation, we refer the interested reader to [Jackson and Mccormick 1988]. The second term in Eq. 7 is straightforward:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\frac{\partial O}{\partial \mathbf{p}} = \mathbf{S}^T\frac{\partial^2 O}{\partial \mathbf{x}\partial \mathbf{p}} + \frac{\partial^2 O}{\partial \mathbf{p}^2}, \tag{8}$$

while the first term evaluates to

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\left(\frac{\partial O}{\partial \mathbf{x}}\mathbf{S}\right) = \left(\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\frac{\partial O}{\partial \mathbf{x}}\right)\mathbf{S} + \frac{\partial O}{\partial \mathbf{x}}\left(\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{S}\right), \tag{9}$$

with

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\frac{\partial O}{\partial \mathbf{x}} = \mathbf{S}^T\frac{\partial^2 O}{\partial \mathbf{x}^2} + \frac{\partial^2 O}{\partial \mathbf{x}\partial \mathbf{p}}. \tag{10}$$

Here, $\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{S}$ is a third-order tensor, and $\frac{\partial O}{\partial \mathbf{x}}\left(\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{S}\right)$ stands for

$$\frac{\partial O}{\partial \mathbf{x}}\left(\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{S}\right) = \sum_i \frac{\partial O}{\partial x_i}\left(\frac{\mathrm{d}^2 x_i}{\mathrm{d}\mathbf{p}^2}\right).$$

The second-order sensitivity term $\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{S}$ must be further broken down:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{S} = \left(\mathbf{S}^T\frac{\partial}{\partial \mathbf{x}}\mathbf{S} + \frac{\partial}{\partial \mathbf{p}}\mathbf{S}\right). \tag{11}$$

The partial derivatives of $\mathbf{S}$ can be found by taking the second derivatives in (4) and rearranging the terms. This results in

$$\frac{\partial}{\partial \mathbf{x}}\mathbf{S} = -\left(\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right)^{-1}\left(\frac{\partial^2 \mathbf{G}}{\partial \mathbf{x}^2}\mathbf{S} + \frac{\partial^2 \mathbf{G}}{\partial \mathbf{p}\partial \mathbf{x}}\right), \tag{12}$$

$$\frac{\partial}{\partial \mathbf{p}}\mathbf{S} = -\left(\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right)^{-1}\left(\frac{\partial^2 \mathbf{G}}{\partial \mathbf{x}\partial \mathbf{p}}\mathbf{S} + \frac{\partial^2 \mathbf{G}}{\partial \mathbf{p}^2}\right), \tag{13}$$

where once again we assume that the tensor expressions are self-evident. Combining all of the terms above leads to the following formula for the Hessian:

$$\frac{\mathrm{d}^2 O}{\mathrm{d}\mathbf{p}^2} = \frac{\partial O}{\partial \mathbf{x}}\left(\mathbf{S}^T\frac{\partial}{\partial \mathbf{x}}\mathbf{S} + \frac{\partial}{\partial \mathbf{p}}\mathbf{S}\right) + \mathbf{S}^T\left(\frac{\partial^2 O}{\partial \mathbf{x}^2}\mathbf{S} + 2\frac{\partial^2 O}{\partial \mathbf{x}\partial \mathbf{p}}\right) + \frac{\partial^2 O}{\partial \mathbf{p}^2}. \tag{14}$$

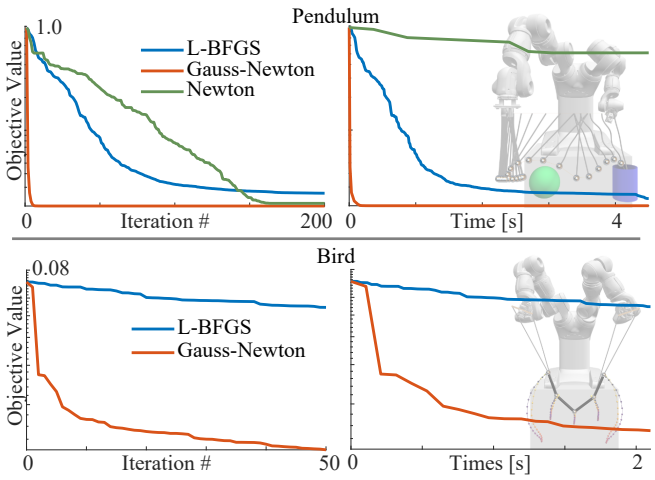


Fig. 3. Comparison of the generalized Gauss-Newton approximation with Newton and L-BFGS. The top row shows the number of iterations and timings for one of the pendulum examples in Fig. 5. Newton's method occasionally stalls in non-convex regions, which largely contributes to the less than satisfactory convergence. Its additional computational cost makes it impractical for this problem. The bottom row shows the comparison for the bird example in Fig. 8. In this case, Newton's method takes an excessive amount of time and is not shown. In both cases it can be seen that the generalized Gauss-Newton converges much faster. It is worth noting that all intermediate results, whether or not the process converged, obey physics becaues of the trajectory optimization strategy we employ.

Generalized Gauss-Newton. Although Newton's method generally converges much faster than L-BFGS or gradient descent, there are two issues with it. First, evaluating the second-order sensitivity term takes a non-negligible amount of time. Second, the Hessian is often indefinite, and needs to be regularized. Both problems can be dealt with by simply excluding the tensor terms in (14). The result is a generalized Gauss-Newton approximation for the Hessian:

$$\mathbf{H} = \mathbf{S}^T\frac{\partial^2 O}{\partial \mathbf{x}^2}\mathbf{S} + \mathbf{S}^T\frac{\partial^2 O}{\partial \mathbf{x}\partial \mathbf{p}} + \frac{\partial^2 O}{\partial \mathbf{x}\partial \mathbf{p}}^T\mathbf{S} + \frac{\partial^2 O}{\partial \mathbf{p}^2}. \tag{15}$$

Although $\mathbf{H}$ is not guaranteed to be positive-definite, we note that in our case, O is a convex function of $\mathbf{x}$, and therefore the first term is always well-behaved. Furthermore, for our control problem, O does not explicitly couple $\mathbf{x}$ and $\mathbf{p}$, and therefore the mixed derivative (i.e. the second) term vanishes. The last term can be indefinite, but we rarely encountered this case in practice.

Sensitivity analysis for motion trajectories. The implicit relationship described by Eq. (1) is very general, and can easily be derived for different types of dynamical systems and numerical integration schemes. Here, we illustrate this process for mass spring systems, since this is how we choose to model marionettes. In this context, the vector $\mathbf{x}$ has dimension $3nT$, where n is the number of mass points, T is the number of time steps, and the constant 3 indicates that each mass point lives in a 3-dimensional space. We turn directly to the time-discretized setting and let $\mathbf{x}_i$, the i-th $3n$ block in $\mathbf{x}$, denote the configuration of the system at time t_i. As described in [Martin et al. 2011], an Implicit Euler time stepping scheme is

equivalent to finding $\mathbf{x}_i$ that minimizes the functional

$$U_i(\mathbf{x}_i) = \frac{h^2}{2}\ddot{\mathbf{x}}_i^T \mathbf{M}\ddot{\mathbf{x}}_i + W(\mathbf{x}_i, \mathbf{p}_i) + \mathbf{g}^T\mathbf{x}_i, \tag{16}$$

where $\ddot{\mathbf{x}}_i = (\frac{\mathbf{x}_i - 2\mathbf{x}_{i-1} + \mathbf{x}_{i-2}}{h^2})$ is the time-discretized acceleration, $\mathbf{M}$ is the system's mass matrix, h is the step size, $W(\mathbf{x}_i, \mathbf{p}_i)$ defines the internal potential deformation energy stored in strings and trusses, and the last term accounts for gravity; $\mathbf{p}_i$, similar to $\mathbf{x}_i$, encodes the control actions taken at time step i. It is easy to verify that $\nabla_{\mathbf{x}_i} U_i = 0$ (i.e. $\mathbf{x}_i$ is a minimizer of U_i) is equivalent to Newton's second law, $\mathbf{M}\ddot{\mathbf{x}}_i = \mathbf{F}(\mathbf{x}_i, \mathbf{p}_i)$ where $\mathbf{F}(\mathbf{x}_i, \mathbf{p}_i) = -\nabla_{\mathbf{x}_i} W(\mathbf{x}_i, \mathbf{p}_i)$, which means that the equations of motion are satisfied.

Through Eq. 16, computing the motion of the marionette using forward simulation is straightforward. We start from an input *control* trajectory $\mathbf{p}$ (the robot standing still is a valid choice for $\mathbf{p}$), and two fixed configurations, $\mathbf{x}_0$ and $\mathbf{x}_{-1}$, which together represent the starting state of the dynamical system. In order, from $i = 1$ to T, we then find $\mathbf{x}_i$ that minimizes U_i using Newton's method.

In the context of forward dynamics, this standard numerical integration scheme also reveals the structure of the implicit relationship described by Eq. (1). In particular, $\mathbf{G}_i$, the i-th $3n$ block in $\mathbf{G}$, is defined as $\nabla_{\mathbf{x}_i} U_i$. It is therefore clear that for any motion trajectory computed through physics simulation, $\mathbf{G}(\mathbf{x}, \mathbf{p}) = 0$, as required for sensitivity analysis.

We discuss the specifics of our marionette model in the next section, but we first highlight two important observations. First, it is very easy to extend the concepts described above to different numerical integration schemes. For example, the only change required to use BDF-2 (backward differentiation formula of second order), which exhibits much less numerical damping than Implicit Euler, is a different definition of the discretized acceleration. Second, upon inspection, it is easy to see that the time domain imposes a very specific structure on the system of equations that must be solved to compute the Jacobian $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{p}}$ in Eq. 4. This structure is visualized in Fig. 4, and can be easily exploited to speed up computations. In particular, since $\mathbf{G}_i$ depends explicitly only on $\mathbf{x}_i$, $\mathbf{x}_{i-1}$, $\mathbf{x}_{i-2}$ and $\mathbf{p}_i$ (i.e. all other partial derivatives are $\mathbf{0}$), $\frac{\partial \mathbf{G}}{\partial \mathbf{p}}$ has a block diagonal form, and $\frac{\partial \mathbf{G}}{\partial \mathbf{x}}$ has a *banded* block diagonal form. This allows us to solve the resulting system using block forward-substitution, rather than storing and solving the entire linear system represented by $\frac{\partial \mathbf{G}}{\partial \mathbf{x}}$. In our experiments, this results in a $5x$ speedup. We also note that the resulting $\mathbf{S}$ is block triangular, which correctly indicates that $\mathbf{x}_i$ does not depend on $\mathbf{p}_j$ if $j > i$, or intuitively, the robot's actions at any moment in time only affect future motions of the marionette.

Iterative optimization. Using either $\frac{\mathrm{d}^2 O}{\mathrm{d}\mathbf{p}^2}$ or its approximation $\mathbf{H}$, we can minimize (2) using a standard unconstrained optimization scheme, but we note one key difference: for each candidate $\mathbf{p}$, we must always compute the corresponding $\mathbf{x}$ to ensure that (1) holds before evaluating any of the derivatives. Once $\frac{\partial O}{\partial \mathbf{p}}$ and $\mathbf{H}$ (or $\frac{\mathrm{d}^2 O}{\mathrm{d}\mathbf{p}^2}$) are computed, we find the search direction $\mathbf{d}$ by solving

$$\mathbf{H}\mathbf{d} = -\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}}. \tag{17}$$

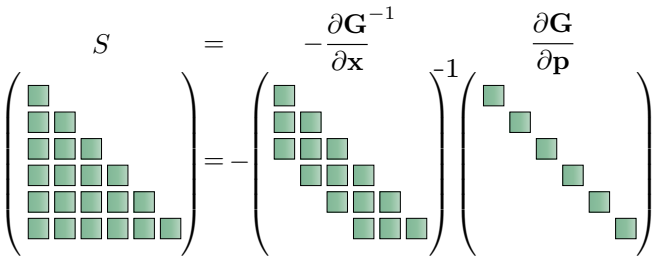


Fig. 4. The structure of the system can be used to compute S faster by block forward-substitution.

We use a backtracking line search to find the step size α, where again, $\mathbf{x}$ need to be recomputed for each test candidate $\overline{\mathbf{p}} = \mathbf{p} + \alpha\mathbf{d}$ in order to evaluate $O(\mathbf{x}(\overline{\mathbf{p}}), \overline{\mathbf{p}})$. We summarize the optimization procedure in Algorithm 1. Note that in the next section we make a transition from handle positions $\mathbf{p}$ to robot joint angles $\mathbf{q}$, but the algorithm essentially stays the same.

Algorithm 1: Trajectory optimization

Input: Dynamical system, initial $\mathbf{p}$, initial $\mathbf{x}_0, \dot{\mathbf{x}}_0$,
Output: Optimal control trajectory $\mathbf{p}$

while *criterion not reached* **do**
  Compute $\mathbf{x}(\mathbf{p})$ using forward simulation
  Compute $\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}}$ (Eq. (3))
  Compute $\mathbf{H}$ ((14) or (15))
  Solve $\mathbf{H}\mathbf{d} = -\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}}$
  Run backtracking line search in d
  `/* Simulate after every line search iteration */`
end

Relation to DDP. A well-known technique for optimal control is the Differential Dynamic Programming method (DDP). While Newton's method approximates the motion control objective through a global quadratic function, DDP uses a sequence of local quadratic models derived for each individual time step. The resulting structure gives rise to the characteristic backwards/forwards nature of DDP algorithms. Although both DDP and Newton's method feature quadratic convergence, the debate regarding which one performs best dates back to at least the 80s [Murray and Yakowitz 1984; Pantoja 1988] and continues today [Mizutani 2015]. Like Newton's method, DDP and its Gauss-Newton approximation, the iterative Linear Quadratic Regulator (iLQR), rely on a state transition function $\mathbf{x}_{i+1} = f(\mathbf{x}_i, \mathbf{p}_i)$ and its first and second derivatives. In practice, DDP and iLQR are typically applied in conjunction with explicit integration schemes, because then f takes on an analytic form, and its derivatives can be readily computed. Consequently, the techniques we have described to compute derivatives of implicitly-integrated forward dynamics simulations open up exciting opportunities for DDP/iLQR optimal control formulations as well.

5 ROBOTIC PUPPETEERING

5.1 The puppet model

With the foundations for motion optimization laid out, we now describe the specifics of the simulation model we employ for marionettes. Traditional marionettes are piece-wise rigid structures. In our simulation model, we assume mass is concentrated at the joints. This matches the way we design and fabricate our physical prototypes, which employ heavy steel balls as joint sockets, and much lighter 3D printed parts as structural frames (see the inset figure). Furthermore, since we are using fully implicit integration schemes for forward simulation, we can model these structural frames as very stiff springs without worrying about stability problems. A mass-spring representation of marionettes is therefore natural. This simple model has the additional benefit that it is unconstrained, and the mathematical formulation derived in the previous section needs only reason about positions, and not about rotations as well.

The mass point representing each joint can optionally be connected via strings to rigid control handles held by the robot. Important for our control problem is the world location of the attachment point of string j at time index i. We denote this quantity by $\mathbf{p}_i^j$. Similarly, we let $\mathbf{x}_i^k$ represent the world position of the k-th mass point at time index i. Its corresponding target location, which is provided through the target animation, is represented by $\hat{\mathbf{x}}_i^k$. We assemble the positions of all mass points at time index i into a vector $\mathbf{x}_i$, and the positions of all string attachment points at time index i into a vector $\mathbf{p}_i$. Finally, we assemble all $\mathbf{x}_i$ into $\mathbf{x}$, and all $\mathbf{p}_i$ into $\mathbf{p}$, forming the state and control trajectories respectively.

To solve the equations of motion using Eq. (16), we formulate the potential energy of the system as

$$W(\mathbf{x}_i, \mathbf{p}_i) = W_{\text{puppet}}(\mathbf{x}_i) + W_{\text{string}}(\mathbf{x}_i, \mathbf{p}_i). \tag{18}$$

The energy $W_{\text{puppet}}(\mathbf{x}_i)$ measures the energy stored in the stiff springs we use to model a marionette's body parts. Let L be the set of pairs of joints connected by links. Then,

$$W_{\text{puppet}}(\mathbf{x}_i) = \sum_{\{j_1, j_2\} \in L} k \left(\left\| \mathbf{x}_i^{j_1} - \mathbf{x}_i^{j_2} \right\| - l_0^{j_1 j_2} \right)^2, \tag{19}$$

where k is a spring constant, and $l_0^{j_1 j_2}$ is the rest length defined by the marionette's physical design. We use $k = 10^4$ for all our experiments, as we found this value sufficient to render the deformations of the robot's body parts imperceptible. The energy $W_{\text{string}}(\mathbf{x}_i, \mathbf{p}_i)$ measures the tension energy stored in the strings. Each string connects a marionette joint x^{j_1} to an attachment point p^{j_2}. We define the set of joint-attachment point pairs by S, and define $W_{\text{string}}(\mathbf{x}_i, \mathbf{p}_i)$ as

$$W_{\text{string}}(\mathbf{x}_i, \mathbf{p}_i) = \sum_{(j_1, j_2) \in S} k\psi \left(\left\| \mathbf{x}_i^{j_1} - \mathbf{p}_i^{j_2} \right\| - s_0^{j_1 j_2} \right), \tag{20}$$

where $s_{j_1 j_2}^0$ is the length of the string, $k = 10^4$, and $\psi(x)$ is a C^2 one-sided quadratic function modeling the unilateral nature of strings [Bern et al. 2017]:

$$\psi(x) = \begin{cases} \frac{1}{2}x^2 + \frac{\epsilon}{2}x + \frac{\epsilon^2}{6} & x \geq 0 \\ \frac{1}{6\epsilon}x^3 + \frac{1}{2}x^2 + \frac{\epsilon}{2}x + \frac{\epsilon^2}{6} & 0 > x > -\epsilon \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

Here the constant $\epsilon = 1mm$ provides for a smooth transition between the regime where the string is slack (and therefore applies no force), and where it is taut and applies tension forces, which can pull but not push. First, second and third derivatives (the latter required for Eq. 6) with respect to $\mathbf{x}$ and $\mathbf{p}$ are straightforward to compute analytically for both energy terms.

5.2 Control

We discussed our control optimization method in Section 4. Here we describe the different terms that define Eq. 2.

Objectives. The main objective is, of course, to find a marionette motion that is similar to the user specified trajectory $\hat{\mathbf{x}}$. To this end, we define a simple quadratic objective that measures the similarity between the two:

$$O_{\text{traj}}(\mathbf{x}) = \|x - \hat{x}\|_2^2.$$

We additionally regularize the acceleration of the string attachment points to promote smooth motions, using the simple objective

$$O_{\text{acc}}(\mathbf{p}) = \|\ddot{p}\|_2^2,$$

where the acceleration vector $\ddot{p}$ is estimated using finite differences.

Up to this point, the control problem operated directly in the space of world trajectories for the string's attachment points. However, we ultimately need to command the movements of the robot puppeteer, which will be specified as joint angle trajectories. These could be computed in post-processing using inverse kinematics (IK). The pitfall of such a strategy is that it is quite likely that the motion planner generates trajectories $\mathbf{p}$ which are not within the workspace of the robot (i.e. they are not reachable), or they lead to inter-limb collisions. Following [Duenser et al. 2018], we eliminate this problem with a simple extension that enables us to directly solve for control actions specified in the robot's joint space.

Direct optimization of the robot's motions. To compute trajectories for the robot puppeteer's joint angles, it suffices to express each string attachment point $\mathbf{p}^j$ as a kinematic function of the robot's joint angles q,

$$\mathbf{p}_i^j = \text{FK}(\mathbf{l}^j, \mathbf{q}_i), \tag{22}$$

where FK is a standard forward kinematics function that computes the world coordinates of a point $\mathbf{l}^j$ expressed in the local coordinate frame of a robot's gripper, and $\mathbf{q}_i$ is a vector that stores the robot's joint angles at time index t_i. With this definition in place, we can minimize the objective $O(\mathbf{p}(\mathbf{q}))$ directly as a function of $\mathbf{q}$. The changes introduced by this step are minimal – we only need to apply the chain rule. For example, the gradient of the objective is given by

$$\frac{\mathrm{d}O}{\mathrm{d}\mathbf{q}} = \frac{\partial O}{\partial \mathbf{p}} \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}\mathbf{q}} + \frac{\partial O}{\partial \mathbf{q}},$$

where the Jacobian $\frac{\mathrm{dp}}{\mathrm{dq}}$ encodes how the world position of the attachment points change with respect to the robot's joint angles. This Jacobian is straightforward to compute given the definition of the forward kinematics map. The second derivative of the objective O with respect to $\mathbf{q}$ can be computed in an analogous manner. Solving directly for joint angle trajectories ensures that the optimization result is always feasible, and that it effectively exploits the workspace of different types of robots. Furthermore, this formulation allows us to define additional objectives to capture, for example, inter-limb collisions or limits on joint angles and velocities. These objectives take on standard forms, so we omit them here for brevity. We refer the interested reader to [Duenser et al. 2018] for formal definitions.

Finding a convenient starting configuration. In simulation, we can set the marionette's starting state (i.e. $\mathbf{x}_0$ and $\mathbf{x}_{-1}$) to any arbitrary values. However, when the robot starts to execute a performance, the physical marionette must start from exactly the same configuration. Consequently, $\mathbf{x}_0$ and $\mathbf{x}_{-1}$ should correspond to marionette poses that are easily achievable. The only practical choice is to pick them such that they represent a statically stable configuration given an initial robot pose $\mathbf{q}_{init}$. We find these configurations by setting the simulated robot in the desired starting pose and then running forward simulation until the system comes to rest.

6 EXPERIMENTAL RESULTS

In this section we summarize the results we generated with our system. We begin with a few preliminary results and continue to more elaborate examples that showcase our system's capabilities. Please refer to the accompanying video for a screen capture and recording of the actual motions. We implemented our algorithm in C++ using Eigen [Guennebaud et al. 2010], and ran it on a computer with an Intel Core i7-7709K 4.2Ghz. The average computation time for each forward simulation step (i.e. computing $\mathbf{x}(\mathbf{p})$) as well as for computing the search direction $\mathbf{d}$ for each individual dynamical system can be found in Table 1. Acceptable control solutions begin to emerge after a small number of iterations, and each intermediate optimization result is valid due to the formulation we employ.

Interactive specification of goals. For small systems, our application reacts at interactive rates. We began by testing a simple scenario involving a pendulum, where the goal is to place the suspended mass point inside a cup while avoiding a spherical obstacle. For this example, the robot's end effector is constrained to move only along a horizontal line through a dedicated control objective. The user can interactively position the cup and obstacle, and immediately observe the result. Fig 5 illustrates two examples that were obtained in this manner. In one example the robot speeds towards the obstacle in order to lift the weight above it, and then slows down to let it land in the cup. In the second example, the robot first moves back and forth to gain momentum before swinging the weight above the obstacle. This simple setting highlights the ability of our system to generate highly dynamic motions that are physically accurate.

We refer the reader to the accompanying video (and Fig. 6), where we show a similar example, this time involving a double pendulum. The dynamics of such systems are notorious for their chaotic behaviour, but for a reasonable planning horizon, we found we can still model them with adequate accuracy. Finally, we ran an additional experiment where the cup was fixed to the top of the robot's end effector. Our optimization method successfully found a very agile robot motion that swings the mass point just right so that it lands in the cup. This example also clearly shows the benefit of modeling strings as unilateral springs. We note that many of our other results, both in simulation and using physical prototypes, exploit the fact that cables can go slack.

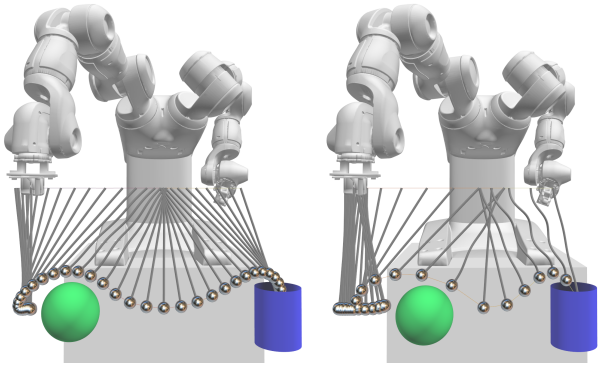


Fig. 5. Interactive positioning of goals and obstacles. The user can reposition the ball and the cup interactively, and the trajectory is updated in real-time. See the accompanying video for more detail.

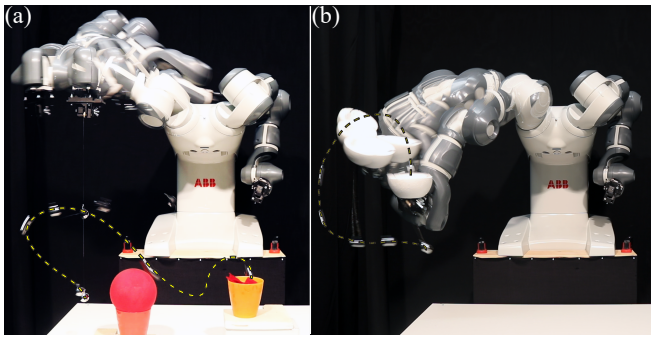


Fig. 6. Additional motions, also included in the accompanying video. Several robot poses are overlaid, and the trajectory of the point mass is approximately traced by a dashed line. (a) A double pendulum, which is known for its chaotic behaviour, can still be accurately modeled for a short time horizon. As in Fig. 5, we ask for a trajectory that avoids the obstacle, and reaches the cup. (b) In this case, the cup is fixed to the top of the handle and requires an agile motion to swing the point mass into the cup.

Puppet fabrication. We use heavy steel balls to fabricate ball-in-socket joints, and lightweight 3D printed frame structures for the marionette's bodies. The steel balls can be connected by strings to the control handles, and they are free to slide inside their sockets. As shown in Fig. 7, hinge joints are also easy to model and fabricate.

Periodic motions. Periodic motions are very common in animation, and for marionettes in particular. We optimize for periodic motions by adding an objective that minimizes the difference between the

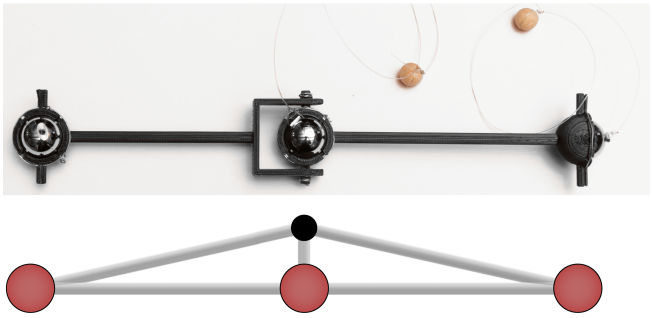


Fig. 7. Fabrication of a hinge joint. In simulation, these joint are represented using a simple frame structure. The black sphere represents a mass point of negligible weight, which functions to model the behaviour of a hinge joint.

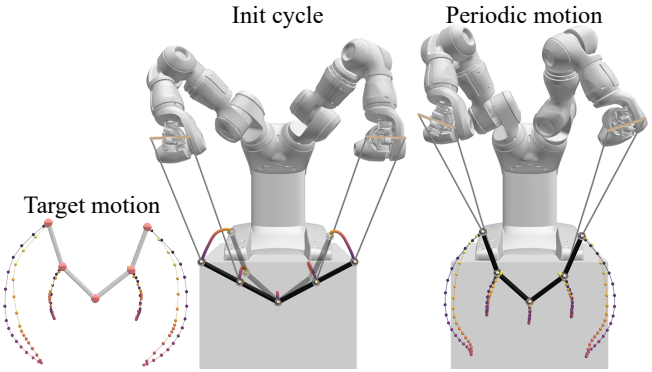


Fig. 8. Designing a periodic motion for the Bird puppet. A periodic motion is normally very dynamic, and there are virtually no points where the puppet is at rest. However, we require all motions to start in a resting configuration. To avoid this issue, periodic motions start with an initialization cycle, where the robot brings the puppet from rest to a specific position and velocity within the periodic motion. It then switches to the target periodic motion

initial and final positions and velocities of all the components: puppet joints and end effectors. Specifically, we extend the time horizon by one sample at each end, x_0 and x_{t+1}, which are not played back, and define the objective

$$O_{\text{CycPos}}(\mathbf{x}, \mathbf{p}) = \|\mathbf{x}_1 - \mathbf{x}_{t+1}\|^2 + \|\mathbf{x}_0 - \mathbf{x}_t\|^2.$$

As mentioned above, we start all of our motions at a rest pose. Starting a periodic motion from a rest pose would mean that the puppet should come to a rest after each cycle. We overcome this by running an *initialization cycle* before switching to the cyclic motion itself. Given a periodic motion, the user can pick a starting point on the period, and the system will find a motion that matches the position and velocity of the puppet at that point of the periodic motion. This is done using an objective for the final position and velocity. We show an example for the Bird puppet in Fig. 8, where the user wished to create a flying motion. The entry point into the periodic motion is at its beginning, when the wings are at the very top. The velocity at this point is admittedly small, but not vanishing. We stress that every other entry point works as well.

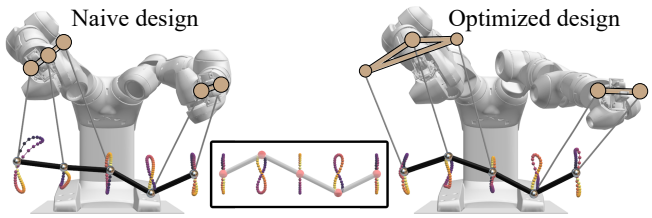
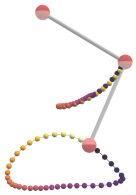


Fig. 9. Continuous design optimization for the Dragon puppet. With a naive handle design, where both handles are linear, the robot cannot reproduce the target motion to a satisfying degree. The design optimization procedure suggests to use a triangular handle, which then results in a much more faithful animation.

Design optimization. Designing appropriate handles and picking suitable string lengths is not an intuitive task, as these parameters shape the space of motions a marionette can perform. In some cases, a quick trial-and-error search suffices to find reasonable parameter values, but we found it very useful (and easy) to devise an automated procedure instead. Since all of our objectives are smooth with respect to design parameters, we can optimize for these as well, again using sensitivity analysis. At a conceptual level, it suffices to replace $\mathbf{p}$ by any design parameter in the derivation we presented in the previous sections. We use this approach to optimize the locations of the attachment points in local coordinates $\mathbf{l}^j$ from Eq. 22, and the string lengths $s_0^{j_1 j_2}$ from Eq. (20). Fig. 9 shows a typical case, where the motion trajectory generated with a sub-optimal handle design cannot reproduce the target closely enough. After design optimization, the resulting motion follows the target much more closely. In this case, design optimization found that using a triangle-shaped handle performs better for this specific motion. This would have been difficult to intuitively predict.



The design problem also involves discrete variables: the decision regarding which point masses should be stringed and to which attachment points plays a crucial role in shaping the space of permissible motions. We explore various possibilities in Fig. 10 for a simple design of a puppet's leg. The target motion appears in the inset. The figure shows different patterns of connectivity of the point masses. As can be seen, the quality of the resulting motion is influenced by this discrete design decision. We currently do not have a way of automatically optimizing this aspect of the design, but we rely on the interactivity of our system to let the designer experiment with different options.

Physical experiments. To investigate the consequences of modelling approximations and the corresponding sim-to-real gap, we recorded the motions of some of our physical marionettes using an Optitrack motion capture system. Fig. 11 summarizes some of our findings by reporting the mismatch between the simulation results and the recorded motions for the Chinese Dragon puppet. While the movements of the robot initially match the simulated trajectories quite well, drift does accumulate over time. Depending on the complexity of the underlying dynamical system, this drift can become problematic. For example, for the Puppy marionette, we have noticed that after about 10 motion cycles played in an

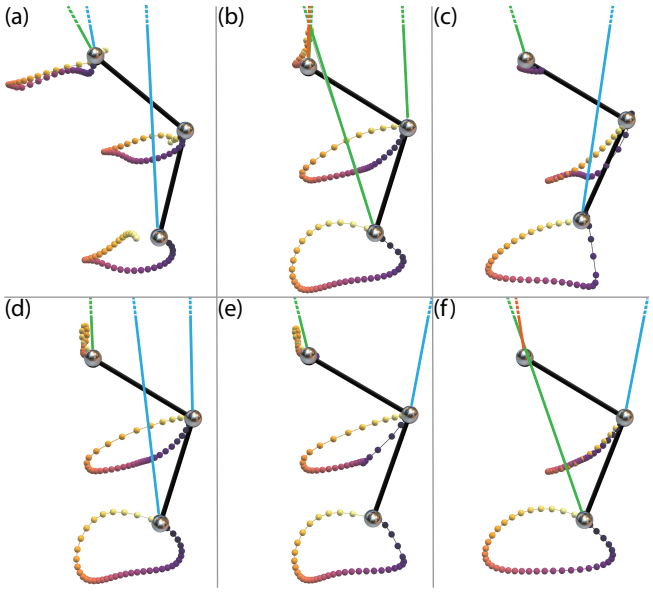


Fig. 10. Comparisons of various stringing options. The choice of how to string a puppet, e.g. where to attach a string to and to which point masses, can have a great impact on the space of permissible motions. Note that the handle locations are omitted from this figure for simplicity. The color coding of the strings indicates attachment to the same handle.

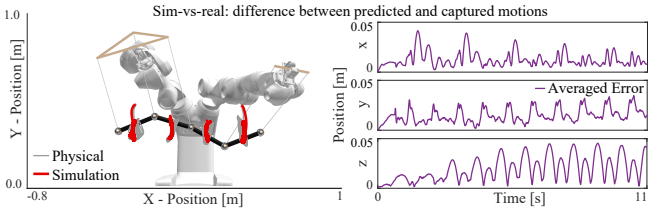


Fig. 11. Sim-vs-real comparison for the Chinese Dragon example. We tracked the motions of four markers placed on the marionette using a motion capture system. The figure on the left shows the measured (in grey) and predicted (in red) trajectories in the x - y plane, starting with the initialization step and followed by 10 periodic cycles. The plots on the right show the position error between simulation predictions and physical measurements averaged over the four markers along the x, y and z axes.

open-loop fashion, the motions of the physical prototype can get out of sync and no longer resemble the desired bounding gait. To overcome problems due to drift, feedback loops should be used to stabilize the nominal trajectories computed through optimization.

Further results. In addition to the examples shown in the paper, we show several more results in the accompanying video. The results show a variety of marionette designs and animations: a fish swimming, a snake slithering, and a puppy bounding and trotting. See Fig. 12 and Table 1 for more information on the puppets.

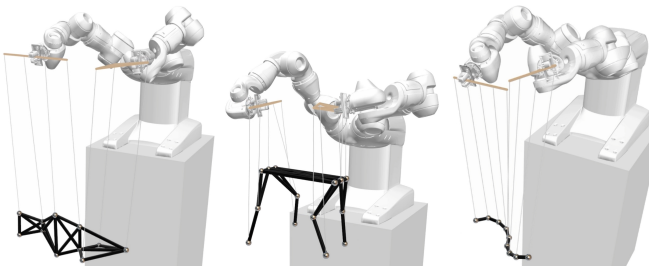


Fig. 12. Several marionettes animated with our system. Please refer to the accompanying video for motions performed both in simulation and using physical prototypes.

7 CONCLUSION, LIMITATIONS AND FUTURE WORK

We presented a control framework for robotic animation of marionettes. As demonstrated through a variety of experiments, the simulation-based motion optimization technique that we have developed provides a promising solution for this very challenging control problem. Nevertheless, our work represents only a first step in endowing robots with human-level skill, and it highlights exciting opportunities for future investigations. For example, there are aspects of a marionette's motion that we are currently not modeling: string collisions, play and friction in the mechanical joints, deformations of the frame structure, forces generated by clothing or other aesthetic layers, etc. Fortunately, as long as such characteristics of the dynamical system can be adequately modeled in a forward dynamics simulation, the control optimization step may remain unchanged. We believe this decoupling is very powerful, as it allows different modeling choices to be studied in isolation.

As there are still unmodeled aspects of a marionette's dynamics, some mismatches between the results predicted in simulation and the corresponding real-world motions are unavoidable. Empirically we have noticed that the sensitivities of a marionette's motions (i.e. how much they change due to an external perturbation) can depend quite strongly on the way in which it is designed. For now, the number of strings and assignment of attachment points to control handles is kept fixed in our framework, but in the future we plan to develop optimization techniques for such discrete design choices. We also currently play back the robot puppeteer's movements in an open-loop manner. Nevertheless, feedback mechanisms that are able to gracefully recover from perturbations or drift that accumulate over time are also very important and should be investigated. And last but not least, we are very excited by the prospect of mathematically analyzing the stability of a marionette's motions, and by the possibility of explicitly incorporating aspects of robust design and control in our trajectory optimization scheme.

Our long term goal is to enable robots to manipulate various types of complex physical systems – clothing, soft parcels in warehouses or stores, flexible sheets and cables in hospitals or on construction sites, plush toys or bedding in our homes, etc – as skillfully as humans do. We believe the technical framework we have set up for robotic puppeteering will also prove useful in beginning to address this very important grand-challenge.

Table 1. List of puppets discussed in the paper and video. We list the number of point masses, strings and attachments points, and the duration of the trajectory. The number of time steps was 40 in all cases. We also provide information regarding the number of iterations applied until a reasonable match of the target trajectory was reached, as well as the average time per iteration to compute the forward simulation step $\mathbf{x}(\mathbf{p})$ and the search direction $\mathbf{d}$. The latter includes the time required to compute the gradient and the hessian $\mathbf{H}$ as well as the solve of the linear system presented in Eq. 17. We note that our code was not optimized for run time and improvements can be made in order to significantly decrease these numbers.

Puppet	# point masses	# strings	# attachment points	animation	planning horizon	# iterations	time $\mathbf{x}(\mathbf{p})$	time $\mathbf{d}$
Pendulum	1	1	1	preliminary	1.0s / 1.33s	10	0.0041s	0.0561s
Double Pendulum	2	2	1	preliminary	1.0s / 1.33s	10	0.0043s	0.0585s
Single Leg	3	2 - 4	2 - 4	preliminary	1.0s	11	0.0027s	0.1252s
Chinese Dragon	5	5	5	flying	1.0s	19	0.0069s	0.4464s
Swallow	5	4	4	flying	1.33s	55	0.0065s	0.3216s
Fish	11	7	7	swimming	1.0s	21	0.0220s	2.2532s
Snake	9	9	9	slithering	1.6s	27	0.0102s	2.5396s
Puppy	14	10	10	bounding	1.0s	37	0.0216s	4.6403s
Puppy	14	10	10	trotting	1.0s	25	0.0270s	4.7045s

REFERENCES

Thomas Auzinger, Wolfgang Heidrich, and Bernd Bickel. 2018. Computational design of nanostructural color for additive manufacturing. *ACM Trans. Graph.* 37, 4 (2018), 159:1–159:16. https://doi.org/10.1145/3197517.3201376

Yunfei Bai, Wenhao Yu, and C. Karen Liu. 2016. Dexterous Manipulation of Cloth. *Comput. Graph. Forum* 35, 2 (2016), 523–532. https://doi.org/10.1111/cgf.12852

Jernej Barbič, Marco da Silva, and Jovan Popović. 2009. Deformable Object Animation Using Reduced Optimal Control. *ACM Trans. Graph.* 28, 3, Article 53 (July 2009), 9 pages. https://doi.org/10.1145/1531326.1531359

James M. Bern, Kai-Hung Chang, and Stelian Coros. 2017. Interactive design of animated plushies. *ACM Trans. Graph.* 36, 4 (2017), 80:1–80:11. https://doi.org/10.1145/3072959.3073700

M. Bernard and K. Kondak. 2009. Generic slung load transportation system using small size helicopters. In *2009 IEEE International Conference on Robotics and Automation*. 3258–3264. https://doi.org/10.1109/ROBOT.2009.5152382

Morten Bisgaard, Jan Bendtsen, and Anders La Cour-Harbo. 2009. Modelling of Generic Slung Load System. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2006-6816

Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. 2003. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution. *SIAM Journal on Scientific Computing* 24, 3 (2003), 1076–1089.

I.-Ming Chen, Raymond Tay, Shusong Xing, and Song Huat Yeo. 2004. Marionette: From Traditional Manipulation to Robotic Manipulation. In *International Symposium on History of Machines and Mechanisms*. Springer, Dordrecht, 119–133. https://doi.org/10.1007/1-4020-2204-2_10

Alexander Clegg, Jie Tan, Greg Turk, and C. Karen Liu. 2015. Animating human dressing. *ACM Trans. Graph.* 34, 4 (2015), 116:1–116:9. https://doi.org/10.1145/2766986

Alexander Clegg, Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. 2018. Learning to dress: synthesizing human dressing motion via deep reinforcement learning. *ACM Trans. Graph.* 37, 6 (2018), 179:1–179:10. https://doi.org/10.1145/3272127.3275048

Michael F. Cohen. 1992. Interactive Spacetime Control for Animation. *SIGGRAPH Comput. Graph.* 26, 2 (July 1992), 293–302. https://doi.org/10.1145/142920.134083

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational design of mechanical characters. *ACM Trans. Graph.* 32, 4 (2013), 83:1–83:12. https://doi.org/10.1145/2461912.2461953

Cédric De Crousaz, Farbod Farshidian, and Jonas Buchli. 2014. Aggressive optimal control for agile flight with a slung load. In *in IROS 2014 Workshop on Machine Learning in Planning and Control of Robot Motion*.

Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. 2016. Computational multicopter design. *ACM Trans. Graph.* 35, 6 (2016), 227:1–227:10. http://dl.acm.org/citation.cfm?id=2982427

Simon Duenser, James M. Bern, Roi Poranne, and Stelian Coros. 2018. Interactive Robotic Manipulation of Elastic Objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*. 3476–3481. https://doi.org/10.1109/IROS.2018.8594291

A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia. 2013. Learning swing-free trajectories for UAVs with a suspended load. In *2013 IEEE International Conference on Robotics and Automation*. 4902–4909. https://doi.org/10.1109/ICRA.2013.6631277

Damien Gauge, Stelian Coros, Sandro Mani, and Bernhard Thomaszewski. 2014. Interactive Design of Modular Tensegrity Characters. In *The Eurographics / ACM SIGGRAPH Symposium on Computer Animation, SCA 2014, Copenhagen, Denmark, 2014*. 131–138. https://doi.org/10.2312/sca.20141131

Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. 2018. Skaterbots: Optimization-based Design and Motion Synthesis for Robotic Creatures with Legs and Wheels. In *Proceedings of ACM SIGGRAPH*, ACM Transactions on Graphics (TOG) (Ed.), Vol. 37. ACM.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

John Hauser. 2002. A Projection Operator Approach To The Optimization Of Trajectory Functionals. *IFAC Proceedings Volumes* 35, 1 (2002), 377 – 382. https://doi.org/10.3182/20020721-6-ES-1901.00312 15th IFAC World Congress.

Richard H. Jackson and Garth P. Mccormick. 1988. Second-order Sensitivity Analysis in Factorable Programming: Theory and Applications. *Math. Program.* 41, 1-3 (May 1988), 1–27. https://doi.org/10.1007/BF01580751

E. Johnson and T. D. Murphey. 2007. Dynamic Modeling and Motion Planning for Marionettes: Rigid Bodies Articulated by Massless Strings. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 330–335. https://doi.org/10.1109/ROBOT.2007.363808

Mickaël Ly, Romain Casati, Florence Bertails-Descoubes, Mélina Skouras, and Laurence Boissieux. 2018. Inverse Elastic Shell Design with Contact and Friction. In *SIGGRAPH Asia 2018 Technical Papers (SIGGRAPH Asia '18)*. ACM, New York, NY, USA, Article 201, 16 pages. https://doi.org/10.1145/3272127.3275036

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus H. Gross. 2011. Example-based elastic materials. *ACM Trans. Graph.* 30, 4 (2011), 72:1–72:8. https://doi.org/10.1145/2010324.1964967

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. ACM, New York, NY, USA, 449–456. https://doi.org/10.1145/1186562.1015744

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus H. Gross, and Stelian Coros. 2015. Interactive design of 3D-printable robotic creatures. *ACM Trans. Graph.* 34, 6 (2015), 216:1–216:9. https://doi.org/10.1145/2816795.2818137

Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus H. Gross, and Bernhard Thomaszewski. 2017. A computational design tool for compliant mechanisms. *ACM Trans. Graph.* 36, 4 (2017), 82:1–82:12. https://doi.org/10.1145/3072959.3073636

E. Mizutani. 2015. On Pantoja's problem allegedly showing a distinction between differential dynamic programming and stagewise Newton methods. *Internat. J. Control* 88, 9 (2015), 1702–1711. https://doi.org/10.1080/00207179.2015.1013063

Todd D Murphey and E. Johnson. 2011. Control aesthetics in software architecture for robotic marionettes. In *American Control Conference (ACC), 2011*. IEEE, IEEE, 3825–3830.

D. M. Murray and S. J. Yakowitz. 1984. Differential dynamic programming and Newton's method for discrete optimal control problems. *Journal of Optimization Theory and Applications* 43, 3 (01 Jul 1984), 395–414. https://doi.org/10.1007/BF00934463

I. Palunko, A. Faust, P. Cruz, L. Tapia, and R. Fierro. 2013. A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots. In *2013 IEEE International Conference on Robotics and Automation*. 4896–4901. https://doi.org/10.1109/ICRA.2013.6631276

Zherong Pan and Dinesh Manocha. 2018. Active Animations of Reduced Deformable Models with Environment Interactions. *ACM Trans. Graph.* 37, 3, Article 36 (Aug. 2018), 17 pages. https://doi.org/10.1145/3197565

J. F. A. De O. Pantoja. 1988. Differential dynamic programming and Newton's method. *Internat. J. Control* 47, 5 (1988), 1539–1553. https://doi.org/10.1080/00207178808906114

Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. 2017. Computational design and automated fabrication of kirchhoff-plateau surfaces. *ACM Trans. Graph.* 36, 4 (2017), 62:1–62:12. https://doi.org/10.1145/3072959.3073695

J. Schultz and T. Murphey. 2012. Trajectory generation for underactuated control of a suspended mass. In *2012 IEEE International Conference on Robotics and Automation*. 123–129. https://doi.org/10.1109/ICRA.2012.6225032

Adriana Schulz, Cynthia R. Sung, Andrew Spielberg, Wei Zhao, Robin Cheng, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. 2017. Interactive robogami: An end-to-end system for design of robots with ground locomotion. *I. J. Robotics Res.* 36, 10 (2017), 1131–1147. https://doi.org/10.1177/0278364917723465

Christian Schulz, Christoph von Tycowicz, Hans-Peter Seidel, and Klaus Hildebrandt. 2014. Animating Deformable Objects Using Sparse Spacetime Constraints. *ACM Trans. Graph.* 33, 4, Article 109 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601156

Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus H. Gross. 2013. Computational design of actuated deformable characters. *ACM Trans. Graph.* 32, 4 (2013), 82:1–82:10. https://doi.org/10.1145/2461912.2461979

Peng Song, Xiaofei Wang, Xiao Tang, Chi-Wing Fu, Hongfei Xu, Ligang Liu, and Niloy J. Mitra. 2017. Computational design of wind-up toys. *ACM Trans. Graph.* 36, 6 (2017), 238:1–238:13. https://doi.org/10.1145/3130800.3130808

Koushil Sreenath and Vijay Kumar. 2013. Dynamics, Control and Planning for Cooperative Manipulation of Payloads Suspended by Cables from Multiple Quadrotor Robots. https://doi.org/10.15607/RSS.2013.IX.011

K. Sreenath, N. Michael, and V. Kumar. 2013. Trajectory generation and control of a quadrotor with a cable-suspended load - A differentially-flat hybrid system. In *2013 IEEE International Conference on Robotics and Automation*. 4888–4895. https://doi.org/10.1109/ICRA.2013.6631275

Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus H. Gross. 2014. Computational design of linkage-based characters. *ACM Trans. Graph.* 33, 4 (2014), 64:1–64:9. https://doi.org/10.1145/2601097.2601143

Emanuel Todorov. 2014. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. 6054–6061. https://doi.org/10.1109/ICRA.2014.6907751

Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. *SIGGRAPH Comput. Graph.* 22, 4 (June 1988), 159–168. https://doi.org/10.1145/378456.378507

Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2018. Bend-it: design and fabrication of kinetic wire characters. *ACM Trans. Graph.* 37, 6 (2018), 239:1–239:15. https://doi.org/10.1145/3272127.3275089

Katsu Yamane, Jessica K. Hodgins, and H. Benjamin Brown. 2004. Controlling a motorized marionette with human motion capture data. *International Journal of Humanoid Robotics* 01, 04 (Dec. 2004), 651–669. https://doi.org/10.1142/S0219843604000319

D. Zameroski, G. Starr, J. Wood, and R. Lumia. 2006. Swing-free trajectory generation for dual cooperative manipulators using dynamic programming. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 1997–2003. https://doi.org/10.1109/ROBOT.2006.1641998

Ran Zhang, Thomas Auzinger, Duygu Ceylan, Wilmot Li, and Bernd Bickel. 2017. Functionality-aware retargeting of mechanisms to 3D shapes. *ACM Trans. Graph.* 36, 4 (2017), 81:1–81:13. https://doi.org/10.1145/3072959.3073710