Trajectory optimization for cable-driven soft robot locomotion

James M. Bern*, Pol Banzet*, Roi Poranne*† and Stelian Coros*
* Department of Computer Science, ETH Zurich, Switzerland
† Department of Computer Science, University of Haifa, Israel
jamesmbern@gmail.com, pol.banzet@protonmail.com, roi.poranne@inf.ethz.ch, scoros@gmail.com

***Abstract*—Compliance is a defining characteristic of biological systems. Understanding how to exploit soft materials as effectively as living creatures do is consequently a fundamental challenge that is key to recreating the complex array of motor skills displayed in nature. As an important step towards this grand challenge, we propose a model-based trajectory optimization method for dynamic, cable-driven soft robot locomotion. To derive this trajectory optimization formulation, we begin by modeling soft robots using the Finite Element Method. Through a numerically robust implicit time integration scheme, forward dynamics simulations are used to predict the motion of the robot over arbitrarily long time horizons. Leveraging sensitivity analysis, we show how to efficiently compute analytic derivatives that encode the way in which entire motion trajectories change with respect to parameters that control cable contractions. This information is then used in a forward shooting method to automatically generate optimal locomotion trajectories starting from high-level goals such as the target walking speed or direction. We demonstrate the efficacy of our method by generating and analyzing locomotion gaits for multiple soft robots. Our results include both simulation and fabricated prototypes.**

I. Introduction

Soft tissues are an integral component of every biomechanical design, and they play a key role in defining the performance, efficiency and robustness of the movements we see in the animal kingdom. In the field of Soft Robotics, a fundamental quest is therefore to understand how to exploit soft materials as effectively as living creatures do. Over the past three decades, this quest has led to the development of robots that are composed almost entirely out of flexible materials [12, 8, 19, 18].

Elastic and deformable materials endow soft robots with abilities that are typically out of reach for traditional, piecewise rigid robotic systems. For example, they can squeeze through small spaces [18], they can traverse unstructured terrains without any perception or feedback loops [8], and they can even be subjected to extreme perturbations without sustaining damage [21]. Nevertheless, the mobile capabilities demonstrated by today's soft robots fall well behind those of their traditional counterparts: while the world's most advanced rigid robots can walk and run in complex environments, soft robots are just beginning to crawl. This is because designing and controlling soft robots demands that the designer understands, anticipates and plans for large structural deformations. Without appropriate modeling and simulation tools, robot designers must rely on intuition and painstaking manual experimentation when creating new types of soft robots.

Our long term goal is to develop a mathematical framework within which to formally frame soft robotic design and motion control problems. As a first step forward, in this paper we introduce an efficient trajectory optimization method that is specifically tailored to cable-driven soft robots. To derive our trajectory optimization formulation, we begin by modeling soft systems using the Finite Element Method (FEM) [9, 1]. Through a numerically robust implicit time-stepping scheme, forward dynamics simulations are used to predict the motion of the robot over arbitrarily long time horizons. Leveraging sensitivity analysis, we show how to efficiently compute analytic derivatives that encode the way in which entire physically-simulated motion trajectories change with respect to parameters that control cable contractions. This information enables the use of a forward shooting method to automatically generate optimal locomotion trajectories starting from high-level goals such as the target walking speed or direction.

Prior work in soft robot locomotion makes heavy use of hand-designed control trajectories [18, 21, 14]. This approach has been shown to yield impressive results for relatively simple designs and motion tasks. However, as is the case with rigid robots, increasing soft robot complexity will necessitate algorithmic frameworks for motion planning and control. In simulation, controllers for soft systems have been generated before using evolutionary approaches [4, 7] or various forms of model-predictive control [20, 6, 17, 11]. Due to the many modeling approximations that are typically exploited by such methods (e.g. unreasonably large number of actuators that can generate arbitrary deformations) these efforts have been confined to simulated environments, without a clear path to applications to real-world soft robots. One notable exception in the soft robotics literature is the work of [15], which develops a trajectory optimization formulation for a sufficiently realistic model of a soft tentacle. We use a very different model (FEM) in our work, and to the best of our knowledge, ours is the first model-based trajectory optimization method for dynamic soft robot locomotion. This specific problem domain also sets our work apart from other recently proposed, model-based control methods for soft robots [1, 10, 5].

Succinctly, our contributions are as follows:

- A lightweight, differentiable simulation model for cable-driven soft robots with contacts.

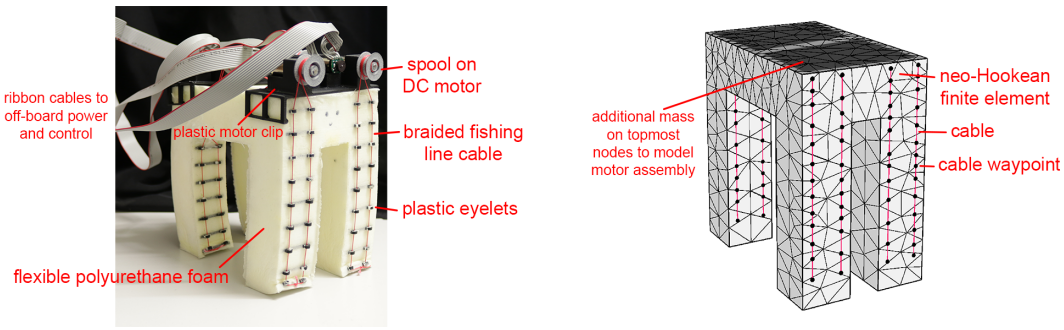

Figure 1. Our *Puppy* robot (left) is a cable-driven quadruped cast from flexible foam. In this work we will show how to find optimal locomotion trajectories for such robots by leveraging a finite element simulation (right).

- An explanation of how to cast soft robot locomotion as a tractable trajectory optimization problem, including the calculation of relevant derivatives.
- Two fabricated prototypes including a soft quadruped.

II. Simulation Model

We model the body of a soft robot using the Finite Element Method, while the cables used for actuation are modeled as stiff unilateral springs. Contacts with the ground are approximated using a differentiable, penalty-based model. We use an implicit time integration scheme, both because it enables the use of large time-steps, and also because it robustly handles the numerically stiff nature of the cables. We now describe each part of our simulation model in detail.

A. FEM Modeling of Soft Robot Bodies

Our approach to computing the FEM energy is the same as the approach taken in [1], and we briefly summarize it here. The body of a soft robot is discretized by a tetrahedral mesh, with linear elements and a compressible neo-Hookean material model. In order to model the motor assembly seated atop the robot, we simply add additional mass to the topmost nodes (See Figure 1). The specific choice of material model is not pivotal, so long as the simulation captures the behaviour of the real material well. While it does not capture e.g. the viscoelastic behavior of the flexible foam, we find the neo-Hookean model to be satisfactory for our application.

The positions of all nodes are assembled into a vector $\boldsymbol{x}$, which we refer to as the *state* of the robot. The deformation energy density of each element using a compressible Neo-Hookean material model is defined by

$$\Psi(\boldsymbol{x}) = \frac{\mu}{2}\text{tr}(\mathcal{F}^\mathsf{T}\mathcal{F} - I) - \mu \ln J + \frac{\kappa}{2}(\ln J)^2, \qquad (1)$$

where $\mathcal{F}$ is the deformation gradient, μ and κ are material parameters, I is the identity matrix and $J = \det(\mathcal{F})$. The total deformation energy $E^{\text{FEM}}(\boldsymbol{x})$ stored in the mesh is computed by summing up the energy stored in all elements.

For computational efficiency we choose to work with coarse finite element meshes. Such meshes experience the phenomenon of *numerical stiffening*, wherein a coarse mesh acts more stiffly in simulation than does its dense counterpart [3]. We therefore choose to fit the parameters of the material model—Young's modulus and Poisson's ratio—experimentally through trial and error, so as to achieve the best possible visual match between simulation and reality. This enables us to achieve a more accurate model than if we were to e.g. look up parameters in a table. While numbers from a table might be correct in the sense that are the true to the real-world material, because of numerical stiffening they will not produce accurate results when applied to a coarse finite element mesh. A coarse mesh with a good visual parameter fit strikes an effective balance between speed and accuracy—fast enough to run trajectory optimization, and accurate enough for the results of that optimization to carry over to the real world.

B. Cables

Cables are represented by polylines, where each vertex is bound to the surface of the mesh using barycentric coordinates. The current length of a cable $L(\boldsymbol{x})$ can be computed directly from the nodal positions $\boldsymbol{x}$, as the total length of the segments of the polyline. The *deformation* of a cable is defined by

$$\Gamma(\alpha, \boldsymbol{x}) = L(\boldsymbol{x}) - \alpha, \qquad (2)$$

where α is a controllable *rest length*. A cable with positive deformation is under tension, while a cable with negative deformation is slack. We define the total energy stored in a cable to be

$$E^{\text{cable}} = \mathcal{Q}(\Gamma(\alpha, \boldsymbol{x})),$$

where $\mathcal{Q}$ is a smooth one-sided quadratic [1], used to model a unilateral spring.

At any given time, a certain length of cable is wrapped around each spool, or *bound*, and the remainder is off the spool, or *free*. A natural kinematic quantity for controlling our robots is therefore the signed length of cable pulled into, or let out of, a spool, relative to the initial length of bound cable. We call this quantity the *contracted length*, and denote it by u. Other choices for the control variable are possible, and we refer the reader to the appendix for one such choice.

We can express the rest length α as a function of u:

$$\alpha(u) = \alpha^0 - u, \qquad (3)$$

where α^0 is the *assembly length* of the cable, defined to be the amount of free cable present when the robot is assembled and at rest. Figure 2 illustrates these definitions. We set the absolute position of the motor θ to zero at assembly, which gives us the simple conversion $u = r\theta$, where r is the radius of the spool.

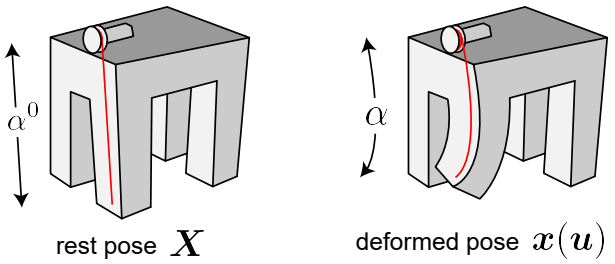

Figure 2. The assembly length α^0 is the length of free cable when the physical robot is first assembled and at rest. The rest length α is the amount of free cable when the robot is actuated by $\boldsymbol{u}$.

C. Contacts

Contacts in deformable objects are notoriously difficult to model. We approach the problem using a penalty method. The benefits are twofold: 1) it more appropriately models the fuzzy interface between the soft robot and the environment and 2) it keeps our model spatially continuous, which makes it amenable to continuous trajectory optimization.

Specifically, we require two consecutive states, $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$, to model contacts. Each node on the boundary of the robot is assigned a *contact energy*, comprised of a normal component and a tangential component. The normal component of the contact energy is $\mathcal{Q}(-y_k)$, where $\mathcal{Q}$ is again the one-sided quadratic [1], and y_k is the y-coordinate of the node's current position. This energy increases rapidly as the node begin to penetrate the floor, and is zero when the node is above the floor. The tangential component of the contact energy is

$$\mu N_{k-1}\left((x_k - x_{k-1})^2 + (z_k - z_{k-1})^2\right)$$

where μ is a tunable scalar weight for the roughness of the surface and N_{k-1} is the magnitude of the normal force at the previous time-step. This energy discourages nodes in contact with the ground from moving tangentially, emulating friction. To summarize, the contact energy is:

$$E^{\text{contact}} = \mathcal{Q}(-y_k) + \mu N_{k-1}\left((x_k - x_{k-1})^2 + (z_k - z_{k-1})^2\right)$$

D. Time integration

We denote the total energy of our system at time t_k as $E_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$ where

$$E_k = E_k^{\text{FEM}} + E_k^{\text{cables}} + E_k^{\text{contacts}} + E_k^{\text{gravity}}.$$

This includes the elastic energy stored in the foam and cables, the contact energies, and gravitational potential energy. Our goal is to compute the state trajectory $\boldsymbol{x}$ given control trajectory $\boldsymbol{u}$. Let $\boldsymbol{x}_k$, $\boldsymbol{u}_k$ denote the nodal positions and cable contractions respectively at time t_k. The trajectory is governed by Newton's second law

$$\boldsymbol{g}_k = \boldsymbol{f}_k - \boldsymbol{m}\boldsymbol{a}_k = 0. \tag{4}$$

where $\boldsymbol{f}_k = -\frac{\partial E_k}{\partial \boldsymbol{x}_k}$ is the vector of nodal forces, $\boldsymbol{m}$ is the mass matrix, and $\boldsymbol{a}_k = \left(\frac{\boldsymbol{x}_k - 2\boldsymbol{x}_{k-1} + \boldsymbol{x}_{k-2}}{h^2}\right)$ is the acceleration discretized according to implicit Euler with step size h. Given control $\boldsymbol{u}_k$ and the previous two states $\boldsymbol{x}_{k-1}, \boldsymbol{x}_{k-2}$, we solve (4) for the new state $\boldsymbol{x}_k$. Specifically we use Newton's method to find $\boldsymbol{x}_k$ minimizing the functional $E + \frac{h^2}{2}\boldsymbol{a}_k^T \mathbf{m}\boldsymbol{a}_k$ as described in [16].

III. Locomotion Optimization

Given a control trajectory $\boldsymbol{u} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K)$ we can use the information in the previous section to compute state trajectory $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K)$ via forward simulation. In this section we turn to our main goal, which is to find a control trajectory that makes the robot locomote. We formulate the problem as a trajectory optimization, where the objective is to match a target trajectory for the robot's center of mass. Specifically, given a desired center of mass trajectory $(\hat{\boldsymbol{x}}_1^{\text{COM}}, \ldots, \hat{\boldsymbol{x}}_K^{\text{COM}})$ our goal is to minimize

$$\mathcal{O}(\boldsymbol{x}(\boldsymbol{u})) = \sum_{k=1}^{K} \left\| \boldsymbol{x}_k^{\text{COM}}\left(\boldsymbol{x}_k(\boldsymbol{u})\right) - \hat{\boldsymbol{x}}_k^{\text{COM}} \right\|^2, \tag{5}$$

where $\boldsymbol{x}^{\text{COM}}(\boldsymbol{x}_k)$ is the center of mass of $\boldsymbol{x}_k$. In practice we add a regularizer to our objective to ensure the optimization is well-posed. We describe this regularizer in the appendix.

Figure 3. A top-down view of the *Puppy*'s target center of mass trajectory $\hat{\boldsymbol{x}}^{\text{COM}}$ (left) for walking in a straight line, and the corresponding simulated center of mass trajectory $\boldsymbol{x}^{\text{COM}}$ made with optimal control signals $\boldsymbol{u}$ (right). In both cases the robot's body is shown at its starting position in yellow. Note that the short grey portion at the beginning of the simulated trajectory corresponds to the preparatory phase of motion, discussed in Section III-B.

In order to minimize (5), we use a *direct shooting* approach in combination with sensitivity analysis to compute analytical derivatives. We now explain the details of our optimization.

A. Sensitivity Analysis

We minimize the objective $\mathcal{O}(\boldsymbol{x}(\boldsymbol{u}))$ using the Gauss-Newton method. This approach requires the gradient of the objective $\frac{d\mathcal{O}}{d\boldsymbol{u}}$. We expand using the chain rule to obtain

$$\frac{d\mathcal{O}}{d\boldsymbol{u}} = \frac{\partial \mathcal{O}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{u}}.$$

The term $\frac{\partial \mathcal{O}}{\partial \boldsymbol{x}}$ is straight-forward to compute, but the term $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$ requires more work. To map from $\boldsymbol{u}$ to $\boldsymbol{x}$ we must solve (4) K times in sequence. While it would be possible to differentiate

this procedure directly using automatic differentiation, we instead leverage a powerful technique called sensitivity analysis. For an introduction to this technique, and a related technique known as the adjoint method, we refer the reader to [2].

All steps of the trajectory $(\boldsymbol{u}, \boldsymbol{x}(\boldsymbol{u}))$ must satisfy Equation (4), and hence all K copies of Equation (4) can be assembled into the matrix equation

$$\underbrace{\begin{bmatrix}\boldsymbol{g}_1\\ \vdots \\ \boldsymbol{g}_K\end{bmatrix}}_{\boldsymbol{g}} = \underbrace{\begin{bmatrix}\boldsymbol{f}_1\\ \vdots \\ \boldsymbol{f}_K\end{bmatrix}}_{\boldsymbol{f}} - \underbrace{\begin{bmatrix}\boldsymbol{m} & & \\ & \ddots & \\ & & \boldsymbol{m}\end{bmatrix}}_{M} \underbrace{\begin{bmatrix}\boldsymbol{a}_1\\ \vdots \\ \boldsymbol{a}_K\end{bmatrix}}_{\boldsymbol{a}} = \boldsymbol{0}. \tag{6}$$

This can be written in a more compact form as

$$\boldsymbol{g} = \boldsymbol{f} - M\boldsymbol{a} = \boldsymbol{0}. \tag{7}$$

Taking the total derivative of $\boldsymbol{g}(\boldsymbol{u}, \boldsymbol{x}(\boldsymbol{u})) = \boldsymbol{0}$ with respect to the control trajectory $\boldsymbol{u}$ results in

$$\frac{d\boldsymbol{g}}{d\boldsymbol{u}} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = \boldsymbol{0}. \tag{8}$$

Note that $\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{u}} = \boldsymbol{0}$, which implies $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}$. Substituting this into Equation (8) and rearranging we arrive at

$$\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}, \tag{9}$$

which can be solved for $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$. While we could do this naively by solving a single massive linear system, we instead employ a computationally-efficient option. We observe that the system has a special block structure as shown in Fig. 4. Indeed, the partial derivative $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} - \boldsymbol{m}\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$ is block lower triangular. More specifically,when using implicit Euler integration scheme, $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ has nonzero blocks only along the diagonal and the two bands below it, for a total of three non-zero bands.

More in-depth, recall that for any equation $\boldsymbol{AX} = \boldsymbol{B}$ where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X}$ are all matrices, we can solve for each column of $\boldsymbol{X}$ separately. Therefore, it suffices to know how to solve for the j-th column of blocks $\frac{d\boldsymbol{x}}{d\boldsymbol{u}_j}$, which relate changes in the control signal at time t_j to changes in the entire state trajectory. Clearly, $\boldsymbol{u}_j$ cannot influence the position of the mesh at any time before t_j, i.e. $\frac{d\boldsymbol{x}_i}{d\boldsymbol{u}_j} = \boldsymbol{0}$ for all $j > i$. Consequently, the derivative $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$ is also block lower triangular. Given the choice of implicit Euler as integrator, multiplying the i-th row of $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ by the j-th column of $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$ yields the equation

$$\frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{x}_i}\frac{d\boldsymbol{x}_i}{d\boldsymbol{u}_j} + \frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{x}_{i-1}}\frac{d\boldsymbol{x}_{i-1}}{d\boldsymbol{u}_j} + \frac{\partial \boldsymbol{g}_i}{\partial \boldsymbol{x}_{i-2}}\frac{d\boldsymbol{x}_{i-2}}{d\boldsymbol{u}_j} = -\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}_j}, \tag{10}$$

which is a recurrence relation for $\frac{d\boldsymbol{x}_i}{d\boldsymbol{u}_j}$, and the total derivative of the i-th physics constraint $\boldsymbol{g}_i = \boldsymbol{0}$ with respect to the j-th control $\boldsymbol{u}_j$. This means that $\frac{d\boldsymbol{x}_i}{d\boldsymbol{u}_j}$ depends on $\frac{d\boldsymbol{x}_{i-1}}{d\boldsymbol{u}_j}$ and $\frac{d\boldsymbol{x}_{i-2}}{d\boldsymbol{u}_j}$. For $i = j$ both of these terms will vanish, since a control applied at time t_j cannot influence the mesh's shape at any previous times. For $i = j + 1$, only the second term will vanish. We can therefore solve the column $\frac{d\boldsymbol{x}}{d\boldsymbol{u}_j}$ by block, starting at the entry $\frac{d\boldsymbol{x}_j}{d\boldsymbol{u}_j}$ and working our way down.

We note that the first system we solve is

$$\frac{\partial \boldsymbol{g}_j}{\partial \boldsymbol{x}_j}\frac{d\boldsymbol{x}_j}{d\boldsymbol{u}_j} = -\frac{\partial \boldsymbol{f}_j}{\partial \boldsymbol{u}_j}, \tag{11}$$

which is none other than the "one step dynamics" relationship obtained by taking the total derivative of the j-th physics update rule with respect to the j-th control $\boldsymbol{u}_j$. In the quasi-static case this reduces to the system solved for quasi-static sensitivities in [1].

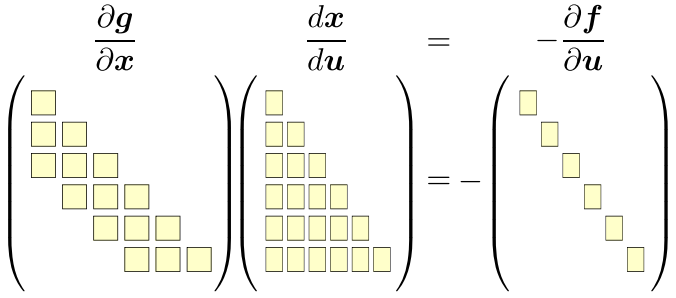

Figure 4. The sparsity structure of the matrices in Equation (9). The key point is that $\frac{d\boldsymbol{g}}{d\boldsymbol{x}}$ is block lower triangular.

B. Implementation details

We turn now to the specific details of our implementation. These details all concern the choice of variables we actually optimize over. We begin by splitting the control trajectory $\boldsymbol{u}$ into two phases. We then reparameterize $\boldsymbol{u}$ using splines. Finally we employ two additional reparameterizations.

1) Control trajectory: We seek a two part control trajectory $\boldsymbol{u}$, consisting of a *preparatory* control trajectory $\boldsymbol{u}^{\text{prep}}$ and a *cyclic* control trajectory $\boldsymbol{u}^{\text{cycle}}$. The preparatory phase takes the robot from its rest state to the beginning of its walk cycle, where the cyclic control trajectory can be repeatedly run to make the robot locomote. For an example of these two parts in practice see Figure 5.

When optimizing, we use the control trajectory

$$\boldsymbol{u} = (\boldsymbol{u}^{\text{prep}}, \boldsymbol{u}^{\text{cycle}}, \boldsymbol{u}^{\text{cycle}}), \tag{12}$$

consisting of the intro phase followed by two copies of the cyclic control trajectory. Initially we included just a single copy of the cyclic control trajectory, but this led to overly-aggressive policies such as the robot diving forward and losing its balance completely. Such pathological policies make sense according to our objective, as diving forward is in fact moving the center of mass closer to its target. However after diving forward, the robot can make no further progress, and so this strategy is not a desirable locomotion strategy according to our intuition. What we really want is a cyclic control trajectory that both 1) moves the robot forward, and 2) finishes with the robot ready to begin another cycle. We accomplish this by including an additional copy of cyclic control trajectory in our overall control trajectory. We note that one could certainly include additional copies, though this would come at a significant computational cost. In all of our examples we found two copies to be sufficient.

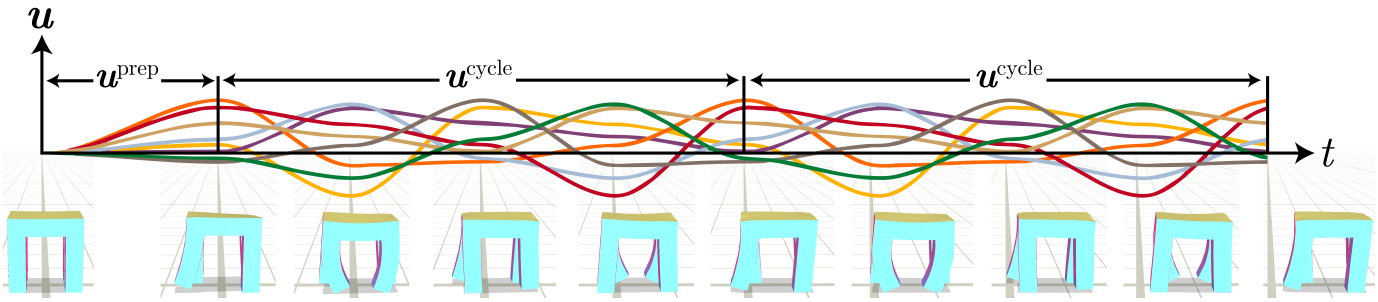

Figure 5. We optimize for a control trajectory composed of a preparatory trajectory followed by two copies of a cyclic trajectory. Each colored trace is the actuation signal for one tendon pair. Note how in the preparatory phase, the *Puppy* raises its hind leg in preparation for the cyclic phase.

2) Splines: In order to ensure smooth control trajectories, as well as to reduce the dimensionality of the search space, we reparameterize the control trajectory of each tendon with a zero-tangent cubic Hermite spline. We space the control points equally along the time axis, and and assemble the u-values of all control points into the reduced optimization vector $\boldsymbol{z}$. The family of curves spanned by our choice of reparameterization is quite simple, but we have found it to be sufficient to generate walking motions. The choice of zero tangents is particularly convenient, as it means that in order to enforce bounds on $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{z})$ we can simply enforce the same bounds on $\boldsymbol{z}$.

One side effect of using splines is that our final trajectory will likely contain unnecessary slack. Depending on the hardware implementation, this can cause problems such as cables slipping off of their spools. We remedy this with a simple post-processing step. For an optimal control trajectory $\boldsymbol{u}$ and a corresponding trajectory of deformations $\boldsymbol{\Gamma}(\boldsymbol{u}, \boldsymbol{x}(\boldsymbol{u}))$, the trajectory of slack $\boldsymbol{s}$ is found by clamping $\boldsymbol{\Gamma}$ to $(-\infty, 0]$ and then taking the absolute value. The zero-slack control trajectory $\boldsymbol{u}^{\text{post}} = \boldsymbol{u} + \boldsymbol{s}$ is functionally equivalent to $\boldsymbol{u}$, and can be used instead.

3) Tendon pairs and bilateral symmetry: In all of our robot designs, each motor drives a pair of tendons. This is a conscious design choice to avoid twisting of the legs. In order to incorporate such *tendon pairs* into our optimization framework, we use an additional reparameterization, wherein each tendon in a given pair shares the same control signal.

We employ one final reparameterization in order to obtain symmetric motions. Examples with bilateral symmetry—such as the *Puppy*—have the property that each tendon pair has a *mirrored pair* on the other side of the robot. Our reparameterization has that a given tendon pair's trajectory is the same as that of its mirrored pair offset by half a period.

IV. Results

We used our system to design locomotion trajectories for two soft robot examples, and built physical prototypes of each. Video-capture data for determining our robots' speeds is shown in Figure 6 and summarized in Table I. We refer the reader to our supplementary video for footage of our examples walking.

Each of our robots consists of a continuum soft foam body, with a rigid motor assembly seated on top. Each motor assembly consists of Pololu 298:1 Micro Metal Gearmotors with 3D-printed spools and a rigid 3D-printed clip-on platform. Power supply and RoboClaw 2x7A Motor Controllers are off-board. The robot's body is cast from FlexFoam-iT!$^{\text{TM}}$III expanded polyurethane foam from Smooth-On Inc. We refer the reader to [13] for a more in-depth discussion of foam casting as it applies to soft robots. We employ 3D-printed eyelets to couple braided fishing line cables to the robot's foam body. We make use of flexible Bowden tubes to cleanly route the cables from the motors through the robot's body to the legs. The eyelets are shallowly-inserted into the foam, and have a very small footprint in the direction of deformation. The motor assembly sits atop of the robot, and while it does contribute additional weight, it has little impact on the robot's deformation behavior. Aside from the eyelets, Bowden tubes, and motor assembly, the robot is completely soft.

To integrate physics forward in time, we use constant time-step $h = 0.033\,\text{s}$. Our trajectories have 24 time-steps in the

Table I
Robot speeds

	Speed in simulation *[body lengths per minute]*	Speed in reality *[body lengths per minute]*
Tripod	6.1	7.4
Puppy	8.6	7.2

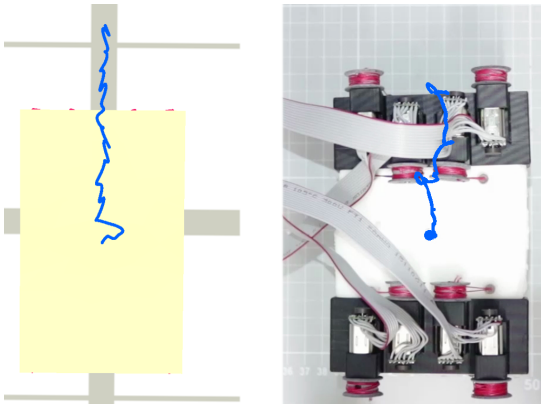

Figure 6. Quantitative comparison of the *Puppy*'s trajectory for the straight line motion seen from a top-view in simulation (left) and in reality (right). We are tracking a feature point on the center of the *Puppy*'s back.

preparatory phase, and 72 time-steps in the cyclic phase. Since we include two copies of the cyclic phase, this means our control and state trajectories are 168 steps, or 5.6 seconds long. The splines we use to reparameterize the control trajectory $\boldsymbol{u}$ have four equally-spaced keyframes. Since the *Puppy* example has 8 actuators, this means the reduced optimization vector $\boldsymbol{z}$ is of size 32 or 16 depending on whether or not we use the bilateral symmetry reparameterization as well. A single iteration of the Gauss-Newton method (including line search) run on the *Puppy* example takes around 2 minutes on a desktop PC. Our optimizations converge quickly, typically finding an acceptable locomotion trajectory in under ten iterations.

A. Tripod

As a first fabricated example of our pipeline, we present a tripod robot. The *Tripod* is $14\,\mathrm{cm}$ long. The *Tripod*'s body weighs $24\,\mathrm{g}$, and its motor assembly weighs $44\,\mathrm{g}$. The outer two legs are unactuated and provide stability. The inner leg is controlled by two motors, one of which bends it to the left, and the other of which bends it to the right. When cocontracted, the motors compress the center leg so that it no longer contacts the ground, as shown in Figure 7. The *Tripod* is a quite a simple robot, and as such is a good first test case for those interested in implementing a similar system. Despite its simplicity it locomotes quite quickly.

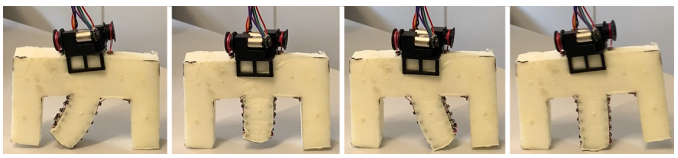

Figure 7. The *Tripod* is a simple foam robot with two actuators that moves by contracting its central leg.

B. Puppy

We also fabricate a soft quadrupedal puppy. Each of the *Puppy*'s four legs are controlled by two motors, for a total of eight actuator degrees of freedom. The general cable layout of each leg is the same as for the middle leg of the tripod. The *Puppy* is $13\,\mathrm{cm}$ long. The *Puppy*'s body weighs $90\,\mathrm{g}$, and its motor assembly weighs $140\,\mathrm{g}$. This makes the puppy top-heavy and unstable. The *Puppy* serves as a more complex example, leveraging our framework to show a variety of gaits in addition to walking on flat ground in a straight line.

we begin by finding a control strategy for straight locomotion on flat ground, with bilateral symmetry turned on. We perform two further optimizations for the *Puppy*. First, we optimize for the *Puppy* to walk up an incline. An inclined ground plane is physically equivalent to a flat ground plane with the direction of gravity rotated. We make this simple modification to our simulator, and then reoptimize for a walking trajectory. Our optimization successfully finds a policy that walks up the hill in simulation. We note that the policy that was optimized for flat ground actually causes the *Puppy* to fall over on this incline. This comparison can be seen in our supplementary video. Second, we optimize for the puppy to turn, by specifying a target trajectory that turns to the right (simply a quarter circle, instead of the straight line used before). For the turning optimization, we *turn off* the bilateral symmetry reparameterization. We do this because a gait that turns the robot must be asymmetric. Warm-started with the control trajectory flat ground motion, our optimization readily finds the turning motion shown in our supplementary video. Making use of one straight leg as a pivot the *Puppy* is able to turn, even though its cables induce deformations only in the sagittal plane.

V. Discussion and Future Work

A. Reparameterizations

Our optimization method makes use of multiple reparameterizations. The use of splines results in smooth, low-acceleration trajectories, and massively reduces the dimensionality of the search space. Bilateral symmetry is also enforced by means of reparameterization, which further reduces the search space.

However, all this reparameterization is not for free. We have no guarantee that our particular slice of the overall search space contains a trajectory that meets our goals. Potentially-superior non-spline or asymmetric trajectories cannot be found when we are using the corresponding reparameterizations. However, if we know we are looking for such trajectory—as is the case for making the *Puppy* turn—we can simply turn the relevant reparameterization off. We can even warm start the optimization for an asymmetric trajectory with a related optimal symmetric trajectory (e.g. warm starting the optimization for turning with the result of the optimization for straight line walking, as we did for the *Puppy*). Future work should be done to find further effective ways to explore the overall search space.

B. Simulation shortcomings

While quite lightweight, our model is still largely sufficient for our control purposes. However, we do note some mismatches between simulation and reality. We highlight relevant areas for improvement in our model here.

One region for improvement is at the interface between our robots and the ground. Locomotion behavior is very much a function of surface. Our model is precise enough to capture a robot walking on a smooth homogeneous table, but the real world is rarely quite so well behaved. Our simple penalty-based model of frictious contacts is a reasonable first step, but future work is needed to model and successfully traverse the varied and complex surfaces we see in the wild.

A second area for improvement is how we model the mass contribution of the motor assemblies. We opt for the simple strategy of simply adding additional mass to the topmost nodes of our finite element mesh. This approach is reasonable enough for walking on flat ground. However on a slope this strategy predicts different moments about the feet than what we get in reality, with the net result being that the simulated robot is more stable than its real-world counterpart. Future work could

incorporate the motor assemblies as rigid bodies, though we note this will complicate the simulation.

C. Towards more complex robots

The physical prototypes in this work are meant to demonstrate the use of the overall system, and were chosen for their simplicity. They have single-piece foam bodies, simple overall geometries, and likely the simplest feet imaginable. Multiple exciting avenues of future work are possible when it comes to pushing the complexity of the robots we can control.

We can start by thinking about fabricating our robots from non-homogeneous materials. This opens the door to perhaps co-optimizing for spatially-varying material parameters. We can also draw some inspiration from nature. Animals have claws and other features on their feet that play a large role in their ability to move on varied surfaces. We can explore how to employ similar strategies to improve the controllability soft robots. Controlling this new generation geometrically-complex soft robots will likely require high fidelity models.

VI. Conclusion

The research community's interest in soft robotic locomotion dates back decades [19]. Nevertheless, soft robots remain very difficult to design and build, and very challenging to control. In this paper we presented a trajectory optimization method for automatically designing soft robot locomotion strategies that carry over to the real world. Prototyping in our simulator is far less time consuming than prototyping in the real world, and the trajectory optimization we propose may find approaches to locomotion that a human designer would not. Our hope is that work like this will empower researchers and enthusiasts alike to explore bold new designs and control strategies for soft robot locomotion, and fulfill the dream of exploiting soft materials as effectively as the members of the animal kingdom.

Appendix

A. Controlling tension directly

In this work we chose our control variable to be contracted length $\boldsymbol{u}$. Other choices are possible, and a particularly elegant one is to prescribe tension $\boldsymbol{\tau}$. This yields a method that is agnostic with respect to cable energy model, and to model ~~bilateral~~ unilateral cables we simply constrain $\boldsymbol{\tau} \geq \mathbf{0}$. The cable energy contribution written as a function of tension τ is simply

$$E^{\text{cable}}(\boldsymbol{x}, \tau) = L(\boldsymbol{x})\tau.$$

However, in order to map back from tension to a real-world quantity (like contracted length) it is necessary to reintroduce an energy model. To play motions back in the real-world, or to e.g. impose bounds on u, then directly controlling tension may not actually be such a convenient choice.

B. Slack-eating regularizer

A cable under any amount of slack (equivalently a cable with $\Gamma \leq 0$) holds zero energy. This introduces a flat region into our optimization landscape, where small changes to the contracted length make zero change to the robot's state. We solve this problem by adding a regularizer that eats slack. The specific regularizer we add is

$$\mathcal{Q}(\varepsilon - \Gamma(\boldsymbol{x}(\boldsymbol{u}), \boldsymbol{u})), \tag{13}$$

where $\mathcal{Q}$ is the smooth one-sided quadratic [1], and ε a small positive constant. This function contributes positive cost for $\Gamma < \varepsilon$, and drives the system towards a state where all cables are at least slightly taught.

References

[1] James M Bern, Grace Kumagai, and Stelian Coros. Fabrication, modeling, and control of plush robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3739–3746. IEEE, 2017.

[2] Andrew M Bradley. Pde-constrained optimization and the adjoint method. 2010.

[3] Desai Chen, David I. W. Levin, Wojciech Matusik, and Danny M. Kaufman. Dynamics-aware numerical coarsening for fabrication design. *ACM Transactions on Graphics (TOG)*, 36(4):84:1–84:15, July 2017.

[4] Nick Cheney, Josh Bongard, and Hod Lipson. Evolving soft robots in tight spaces. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 935–942, 2015.

[5] Eulalie Coevoet, Adrien Escande, and Christian Duriez. Optimization-based inverse model of soft robots with contact handling. *IEEE Robotics and Automation Letters (RA-L)*, 2(3):1413–1419, 2017.

[6] Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. Deformable objects alive! *ACM Transactions on Graphics (TOG)*, 31(4):69, 2012.

[7] Francesco Corucci, Nick Cheney, Francesco Giorgio-Serchi, Josh Bongard, and Cecilia Laschi. Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions. *Soft Robotics (SoRo)*, 5(4):475–495, 2018.

[8] Dylan Drotman, Saurabh Jadhav, Mahmood Karimi, Philip deZonia, and Michael T. Tolley. 3d printed soft actuators for a legged robot capable of navigating unstructured terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5532–5538, 2017.

[9] Christian Duriez. Control of elastic soft robots based on real-time finite element method. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3982–3987. IEEE, 2013.

[10] Fanny Ficuciello, A. Migliozzi, Eulalie Coevoet, A. Petit, and Christian Duriez. Fem-based deformation control for dexterous manipulation of 3d soft objects. In *IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 4007–4013, 2018.

[11] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. *arXiv preprint arXiv:1810.01054*, 2018.

[12] Xiaonan Huang, Kitty Kumar, Mohammad K. Jawed, Amir M. Nasab, Zisheng Ye, Wanliang Shan, and Carmel Majidi. Chasing biomimetic locomotion speeds: Creating untethered soft robots with shape memory alloy actuators. *Science Robotics*, 3(25), 2018.

[13] Nitish Kumar Stelian Coros Luca Somm, David Hahn. Expanding foam as the material for fabrication, prototyping and experimental assessment of low cost soft robots with embedded sensing. *IEEE Robotics and Automation Letters (RA-L)*, 4(2):761–768, 2019.

[14] Shixin Mao, Erbao Dong, Hu Jin, Min Xu, and KH Low. Locomotion and gait analysis of multi-limb soft robots driven by smart actuators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2438–2443, 2016.

[15] Andrew D Marchese, Russ Tedrake, and Daniela Rus. Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator. *The International Journal of Robotics Research (IJRR)*, 35(8):1000–1019, 2016.

[16] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus H. Gross. Example-based elastic materials. *ACM Transactions on Graphics (TOG)*, 30 (4):72:1–72:8, 2011.

[17] Zherong Pan and Dinesh Manocha. Active animations of reduced deformable models with environment interactions. *ACM Transactions on Graphics (TOG)*, 37(3):36, 2018.

[18] Robert F Shepherd, Filip Ilievski, Wonjae Choi, Stephen A Morin, Adam A Stokes, Aaron D Mazzeo, Xin Chen, Michael Wang, and George M Whitesides. Multigait soft robot. *Proceedings of the national academy of sciences (PNAS)*, 108(51):20400–20403, 2011.

[19] Koichi Suzumori, Shoichi Iikura, and Hirohisa Tanaka. Development of flexible microactuator and its applications to robotic mechanisms. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1622–1627, 1991.

[20] Jie Tan, Greg Turk, and C Karen Liu. Soft body locomotion. *ACM Transactions on Graphics (TOG)*, 31 (4):26, 2012.

[21] Michael T Tolley, Robert F Shepherd, Bobak Mosadegh, Kevin C Galloway, Michael Wehner, Michael Karpelson, Robert J Wood, and George M Whitesides. A resilient, untethered soft robot. *Soft robotics (SoRo)*, 1(3):213–223, 2014.