A Soft Robot Inverse Kinematics for Virtual Reality

Bern, James M.; May, William C.; Osborn, Austin; Stella, Francesco; Zargarzadeh, Sadra; Hughes, Josie

DOI
10.1109/ICRA57147.2024.10611603

Publication date
2024

Document Version
Final published version

Published in
Proceedins of the IEEE International Conference on Robotics and Automation, ICRA 2024

Citation (APA)
Bern, J. M., May, W. C., Osborn, A., Stella, F., Zargarzadeh, S., & Hughes, J. (2024). A Soft Robot Inverse Kinematics for Virtual Reality. In *Proceedins of the IEEE International Conference on Robotics and Automation, ICRA 2024* (pp. 14957-14963). (Proceedings - IEEE International Conference on Robotics and Automation). IEEE. https://doi.org/10.1109/ICRA57147.2024.10611603

2024 IEEE International Conference on Robotics and Automation (ICRA)
May 13-17, 2024. Yokohama, Japan

2024 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-8457-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICRA57147.2024.10611603

A Soft Robot Inverse Kinematics for Virtual Reality

James M. Bern[1], William C. May[1,*], Austin Osborn[1,*]
Francesco Stella[2,3,*], Sadra Zargarzadeh[2,*], and Josie Hughes[2]

***Abstract*— We show how a variety of techniques from Computer Graphics can be leveraged to intuitively control the shape (configuration) of arbitrary 3D Soft Robots in VR. Our pipeline, Virtual Reality Soft Robot Inverse Kinematics (VR-Soft IK), overcomes fundamental limitations of general-purpose drag-and-drop soft robot control interfaces by leaving the 2D computer screen for 3D Virtual Reality (VR). VR-Soft IK uses a simulation based on the Finite Element Method (FEM) and a control method based on sensitivity analysis. Additionally, we show that our general control pipeline can be fused with techniques from 3D character animation to skin our simulation with a high-resolution surface mesh, pointing a way toward Mixed Reality Soft Robots. This full Skinned VR-Soft IK pipeline uses skeletal animation and GPU picking. We demonstrate the utility of our pipeline by doing real-time, open-loop control of the real-world 3D soft robotic arm Helix.**

I. Introduction

Soft robots leverage soft materials and structures, actuators, and control algorithms to enable compliant interactions with the environment [1]. This bio-inspired approach enables interactions that can offer inherent safety around humans, robustness to and uncertainty in the environment. These properties are essential for robots to perform tasks in and around human environments in roles such as healthcare, agriculture, or home assistance. However, due to fundamental modeling and control challenges, soft robots remain challenging to effectively control [2].

Our work starts with the observation of a fundamental challenge facing general-purpose user interfaces for open-loop soft robot control (inverse kinematics) [3]–[5]. Specifically, soft robots move in 3D, but computer screens are only 2D. This means quickly and accurately prescribing 3D IK targets on a 2D computer screen can be extremely challenging. We will show that Virtual Reality (VR) is a compelling way around this challenge, allowing users to intuitively prescribe IK targets in 3D.

Alternative, hardware-based approaches to intuitive soft robot posing include incorporating touchless sensors into the robot to enable intuitive interaction modalities for teaching and control [6]. While enabling fluent interactions, the speed and flexibility of control are limited by the placement of discrete sensors over the robot's body.

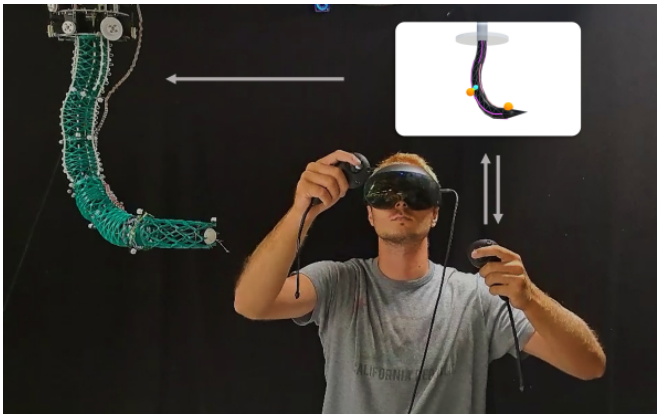

Fig. 1. We present VR-Soft IK, an intuitive way to control the shape of an arbitrary 3D soft robot in Virtual Reality (VR). The user is immersed in a virtual environment, where they are presented with a finite element method (FEM)-based simulation of a cable-driven soft robot (image inset in top right). Using both hands, the user specifies feature points on the simulation mesh and drags around target positions. A gradient-based optimization routine automatically finds cable control inputs that bring the robot as close as possible to the user-specified target. These control inputs are streamed to the real-world robot (top left) in real-time.

Other approaches leverage kinematically-similar devices operating in a leader-follower configuration [7], [8]. These are elegant solutions yet require replication of hardware. Our VR-based approach will require only a simulation.

An exciting body of work at the intersection of soft robotics and VR has investigated applying soft robots to VR, specifically as haptic interfaces. The most popular breed of such soft robotic haptic interfaces is gloves [9]–[11]; however, other soft wearable VR robots have been experimented with including exosuits [12] and necklaces [13]. We are inspired by these works, but, in this paper, our major focus is on *applying VR to soft robotics*, not vice versa.

We present an extension of the model-based, open-loop Soft IK control method [4] to VR. Soft IK, overviewed in Section II-B, leverages differentiable simulation and sensitivity analysis to enable open-loop posing of arbitrary soft robots [4]. Soft IK is a general method previously extended to locomotion trajectory generation [14] and stiffness control [15]. In this work, we extend Soft IK by bringing the soft robot simulation into VR. Other work has also explored bringing visualizations of soft robots into extended reality using some of the techniques we will describe in this paper [16]. We draw inspiration from this work and show how VR can be pushed even further, enabling not just visualization but also intuitive control.

[1]James M. Bern, William C. May, and Austin Osborn are with the Computer Science Department at Williams College jmb15@williams.edu
[2]Francesco Stella, Sadra Zargarzadeh, and Josie Hughes are with the Computational Robot Design & Fabrication Lab, EPFL. josie.hughes@epfl.ch
[3]Francesco Stella is with the Department of Cognitive Robotics, Delft University of Technology, Delft, The Netherlands.
[*]These authors contributed equally.

Beyond Virtual Reality, which places the user in a purely virtual space, Mixed Reality (MR) presents the user with a hybrid of the physical and virtual worlds. Some headsets, like the Microsoft HoloLens [17], achieve this with a transparent display. Others, like the Meta Quest 2, use "pass-through," [18] overlaying virtual content on real-world video streamed from externally-facing cameras. Mixed Reality has exciting applications for entertainment, allowing users to do surreal things in the real world [19], [20]. It also has exciting applications for robotic control, enabling users to intuitively manipulate complex robotic systems [21]–[23]. We see great potential for Mixed Reality to be applied to soft robotics and will take a first step towards this vision in Section III when we extend our VR-Soft IK pipeline with skinning.

Specifically, we contribute:

- VR-Soft IK, an extension of the Soft IK control method to Virtual Reality using CPU raycasting.
- Skinned VR-Soft IK, a method for seamlessly incorporating a high-resolution character mesh into our pipeline using skeletal animation and GPU picking.
- Experiments on a real-world soft robot demonstrating the real-time use of our system.

II. Method: VR-Soft IK

In this section, we describe how to implement VR-Soft IK, a soft robot inverse kinematics for virtual reality. Our overall system promises an ease of 3D spatial control that far exceeds that which is possible with vanilla, 2D Soft IK. The methods are general enough to handle any soft robot that can be modeled in a FEM-based simulator and could be easily extended to handle, e.g., a rod-based simulator. The methods are general to essentially any current consumer-grade VR headset and could be easily extended to a gesture-based interface.

A. FEM-based Simulation

The *simulation* or forward kinematics (FK) problem gives us a choice of robot control inputs and asks us to predict its resulting position. To simulate an arbitrary cable-driven soft robot, we discretize the robot's volume into a tetrahedral finite element mesh and stack the positions of each node into a vector $\boldsymbol{x}^{\text{rest}} = \begin{bmatrix}\boldsymbol{x}_1^{\text{rest}} & \dots & \boldsymbol{x}_N^{\text{rest}}\end{bmatrix}^T$, which we call the robot's *rest position* or *rest shape*. We model the cables as one-sided springs running frictionlessly through via points in the mesh and stack the *cable contractions* (motor angle times spool radius) into a vector $\boldsymbol{u}$. The simulation models a real-world cable being pulled onto a spool by decreasing the rest length of the corresponding simulated spring.

Given a choice of control inputs $\boldsymbol{u}$, we minimize the total potential energy of the system E to find the resulting statically stable shape of the robot

$$\boldsymbol{x}(\boldsymbol{u}) = \arg\min_{\boldsymbol{x}} E(\boldsymbol{x}, \boldsymbol{u}),$$

which is equivalent to solving $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{0}$ for $\boldsymbol{x}$, where $\boldsymbol{F}$ is the total force on each node, stacked into a vector.

B. Soft IK

The *inverse kinematics* (IK) problem asks, given a target position for the robot, what are optimal motor angles that deform the robot into this target position [4]. In principle, we could specify a target position $\boldsymbol{x}'$ for every node in the mesh, and minimize an objective like $\|\boldsymbol{x}(\boldsymbol{u}) - \boldsymbol{x}'\|^2$, which sums the squared distance between deformed position $\boldsymbol{x}_i$ and target position $\boldsymbol{x}_i'$ for all nodes. However, specifying an entire target shape $\boldsymbol{x}'$ is cumbersome. Instead, we specify target positions $\boldsymbol{p}'$ for one or more *feature points* $\boldsymbol{p}$. There are several ways we could go about specifying feature points. First, we could choose a feature point to be the position of a particular node, e.g., one near the robot's tip [24]. More generally, we could choose a feature point to be the weighted sum of the positions of multiple nodes, e.g., the robot's center of mass [14]. In this work, we use a special kind of weighted sum called *barycentric coordinates* to specify feature point positions within, or on the surface of, the mesh's tetrahedra.

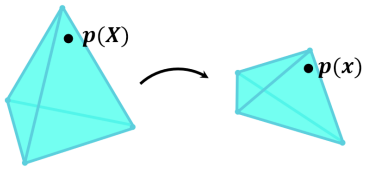

Fig. 2. Barycentric coordinates enable us to specify a feature point within (or on the surface of) a tetrahedron. As the tetrahedron deforms, the point deforms along with it, as if the point were "glued in place." Here, the tetrahedron's rest shape is $\boldsymbol{X}$ and its deformed shape is $\boldsymbol{x}$. The quantities $\boldsymbol{p}(\boldsymbol{X})$ and $\boldsymbol{p}(\boldsymbol{x})$ are the feature point's position in the mesh's rest shape and deformed shape respectively.

Specifying a feature point that refers to a tetrahedron with vertex indices i_a, i_b, i_c, i_d is done by selecting *barycentric weights* $\alpha, \beta, \gamma, \delta$ that obey $\alpha + \beta + \gamma + \delta = 1$. A barycentrically-specified feature point's position is found by the weighted sum $\alpha\boldsymbol{x}_{i_a} + \beta\boldsymbol{x}_{i_b} + \gamma\boldsymbol{x}_{i_c} + \delta\boldsymbol{x}_{i_d}$. Note that in order for a feature point to not lie outside the tetrahedron, it must have $\alpha, \beta, \gamma, \delta \geq 0$. Since feature points refer to deformed mesh shape, i.e., since $\boldsymbol{p} = \boldsymbol{p}(\boldsymbol{x}(\boldsymbol{u}))$, our "sparse" IK objective is

$$\mathcal{O}_{\text{IK}}(\boldsymbol{u}) = \|\boldsymbol{p}(\boldsymbol{x}(\boldsymbol{u})) - \boldsymbol{p}'\|^2.$$

We minimize this objective using gradient descent and compute its gradient using the *adjoint method*. Briefly, to compute a gradient of the form $\frac{d\mathcal{O}}{d\boldsymbol{u}} = \frac{\partial\mathcal{O}}{\partial\boldsymbol{u}} + \frac{\partial\mathcal{O}}{\partial\boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$, where $\frac{\partial\mathcal{O}}{\partial\boldsymbol{u}}$ and $\frac{\partial\mathcal{O}}{\partial\boldsymbol{x}}$ are easy to compute analytically but $\boldsymbol{x}(\boldsymbol{u})$ is only known to satisfy some constraint $\boldsymbol{F}(\boldsymbol{x}(\boldsymbol{u}), \boldsymbol{u}) = \boldsymbol{0}$, we can use a family of techniques colloquially known as "sensitivity analysis." The Jacobian $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$ is the *sensitivity*, telling us how small changes in control inputs $\boldsymbol{u}$ affect changes in the corresponding statically stable mesh position $\boldsymbol{x}(\boldsymbol{u})$. The key observation is that differentiating the constraint yields matrix-matrix equation $\frac{d\boldsymbol{F}}{d\boldsymbol{u}} = \frac{\partial\boldsymbol{F}}{\partial\boldsymbol{u}} + \frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = \boldsymbol{0}$, which can, in principle, be solved for the Jacobian of interest $\frac{d\boldsymbol{x}}{d\boldsymbol{u}} = -\left(\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}\right)^{-1}\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{u}}$. Computing $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$ directly in this fashion is called "direct sensitivity analysis."

An alternative called the "adjoint method" is often more efficient, especially for even modestly high-dimensional $\boldsymbol{u}$. The adjoint method finds the vector $\boldsymbol{\lambda} = \frac{\partial \mathcal{O}}{\partial \boldsymbol{x}} \left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right)^{-1}$ by solving matrix-vector *adjoint equation* $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}^T \boldsymbol{\lambda}^T = \frac{\partial \mathcal{O}}{\partial \boldsymbol{x}}^T$, and uses it to evaluate the equivalent expression for the gradient $\frac{d\mathcal{O}}{d\boldsymbol{u}} = \frac{\partial \mathcal{O}}{\partial \boldsymbol{u}} + \boldsymbol{\lambda}^T \frac{\partial \mathcal{O}}{\partial \boldsymbol{u}}$. We apply this technique, by expressing $\frac{d\mathcal{O}_{\text{IK}}}{d\boldsymbol{u}} = \frac{\partial \mathcal{O}_{\text{IK}}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{u}}$, and computing $\frac{\partial \mathcal{O}_{\text{IK}}}{\partial \boldsymbol{x}} = \frac{\partial \mathcal{O}_{\text{IK}}}{\partial \boldsymbol{p}} \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{x}}$ analytically.

C. VR User Interface

1) Feature Point Specification: We allow the user to dynamically specify ("pick") feature points $\boldsymbol{p}$ anywhere on the simulation mesh's surface. Each VR controller's position and orientation imply a *ray*, usually drawn as a straight line emanating from the controller. If this ray intersects the mesh, we draw a sphere at the intersection point as a preview to the user of where a new feature point will be spawned if they press the trigger button.

Here is how that works. Consider one controller's ray, with the equation $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$, where $t > 0$. Here, $\boldsymbol{o}$ is the ray's origin, $\boldsymbol{d}$ is the ray's direction, and t is the distance traveled from the ray origin. We will cast this ray at the (triangle mesh) surface of our simulation mesh and find its closest intersection point. This is a typical operation in ray tracing, which we can call a *ray cast*; here is a simple approach. We perform ray-triangle intersections for all surface triangles, one at a time on the CPU. For a triangle with vertex positions $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$, combining the ray equation with the definition of barycentric coordinates yields the linear system

$$\begin{cases} \mathbf{o} + t\mathbf{d} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ \alpha + \beta + \gamma = 1, \end{cases}$$

which we solve for barycentric weights α, β, γ and ray distance t. If all these quantities are greater than zero, then we have an intersection. We choose the closest intersection, i.e., the one with the smallest t, to be our new feature point.

2) Target Position Specification: When the user picks out a new feature point on the surface of the simulation mesh, we spawn two spheres in our VR scene, one at the feature point and one at its target position. Initially, the two are coincident. With either VR controller, the user can grab onto the target position sphere and drag it around. Sphere picking is achieved using the ray-sphere intersection routine from [25]. While the user is grabbing the target position sphere, we enable them to move it around either by moving their VR controller spatially or by using the thumbstick on the VR controller. These behaviors are implemented in, e.g., Unity's XR Grab Interactable [26].

3) Other Operations: We can provide the user with several other useful operations. First, the user can toggle the optimization on and off. It is easier to specify feature points when the robot is not moving. Second, we provide the user with the ability to remove feature points, again using a ray-sphere intersection. Finally, we can enable the user to move pre-existing feature points around *on the surface of the mesh*, by raycasting at the surface of the simulation mesh every frame. This enables subtle adjustments without having to remove a feature point and add a new one.

III. Extension: Skinned VR-Soft IK

In Section II, we showed how to perform Soft IK in VR by presenting a user with a 3D simulation of the soft robot. However, what if we could give the user something *more*? What if we could enable the user to interact with a representation of the robot that was more intuitive or more engaging than a simple visualization of our simulation? In this section, we take a first step towards mixed reality soft robotics and present Skinned VR-Soft IK, which incorporates a high-resolution surface mesh into our picking and optimization routines. Screenshots are shown in Figure 3.

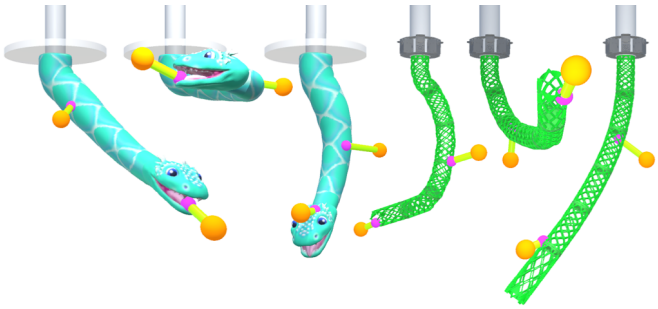

Fig. 3. Skinned VR-Soft IK incorporates a high-resolution surface mesh ("skin"), perhaps of a friendly animated character (left) or a visually accurate CAD model of the real-world robot (right). Note that we are not simply overlaying the high-resolution mesh as a post-processing step. Rather, the deformed shape of the high-resolution mesh is fully incorporated into our picking and optimization routines, allowing the user to specify feature points *directly on the surface of the high-resolution mesh*. Note, for example, the feature point specified inside of the dragon's mouth.

A. Mapping Between Simulation Mesh and Character Mesh Using Skeletal Animation

First, we show how to fuse our FEM-based simulation with a high-resolution triangle mesh, which we can think of as the virtual robot's "skin." We call the deformed shape of the skin $\boldsymbol{s}$. We will build up a differentiable function $\boldsymbol{s}(\boldsymbol{x})$, which deforms the skin according to the simulation.

1) Skeletal Animation Background: Skeletal animation pipelines are used ubiquitously in animated film and games, and enable artists to efficiently and intuitively deform triangle (surface) meshes by using a *skeleton* (or "rig.") The process begins with the surface mesh in a *bind* pose. For a humanoid character, this is typically a "T-pose," i.e., the character standing straight up with their arms stuck out such that the character looks like the letter T. An artist called a "rigger" adds a skeleton of virtual *bones* to the mesh, and assigns bone weights to each of the mesh's vertices. E.g., Vertex 7 might be 30% of influenced by the movement of Bone 1, and 70% influenced by the movement of Bone 2. These weights "bind" the mesh to its skeleton. An animator can now conveniently deform the mesh *by posing its skeleton*.

Mathematically, we can encode the j-th bone's current orientation and position into a rotation matrix $\boldsymbol{R}_j^{\text{curr}}$ and translation vector $\boldsymbol{t}_j^{\text{curr}}$, which we can assemble a homogeneous transformation matrix

$$\boldsymbol{B}_j^{\text{curr}} = \begin{bmatrix} \boldsymbol{R}_j^{\text{curr}} & \boldsymbol{t}_j^{\text{curr}} \\ \mathbf{0}^T & 1 \end{bmatrix}.$$

A similar transformation matrix $\boldsymbol{B}_j^{\text{bind}}$ can be assembled for each bone's bind orientation and position. The combined transform $\boldsymbol{B}_j = \boldsymbol{B}_j^{\text{curr}} \left(\boldsymbol{B}_j^{\text{bind}}\right)^{-1}$ describes how a vertex fully attached to the j-th bone would be transformed by that bone's movement. In *linear blend scanning*, vertices are not attached to just one bone but rather are attached to multiple bones in a weighted fashion, and so the i-th vertex's deformed position is calculated as the linear combination

$$\boldsymbol{s}_i = \sum_j W_{i,j} \boldsymbol{B}_j \boldsymbol{s}_i^{\text{bind}},$$

where W_{ij} is the *blend weight* of the i-th vertex for the j-th bone, and $\boldsymbol{s}_i^{\text{bind}}$ is the bind position of the i-th vertex in homogenous coordinates. This equation can be evaluated in parallel for all vertices using a special type of program called a *vertex shader*, which runs on the GPU.

2) Adding Bones to Soft IK: Our insight is to define the skeleton's shape $\boldsymbol{B}(\boldsymbol{x}(\boldsymbol{u}))$ in terms of the simulation mesh's deformed shape, again using barycentric coordinates. E.g., Bone 1's origin might be at the position of Node 7, and Bone 1's x-axis might point from Node 7 to Node 2.

In summary, the character mesh's deformed shape $\boldsymbol{s}$ is driven by the skeleton's deformed shape $\boldsymbol{B}$, which is driven by the simulation mesh's deformed shape $\boldsymbol{x}$, which is finally driven by the control inputs $\boldsymbol{u}$, i.e., the character mesh's deformed shape is a function of the form $\boldsymbol{s}(\boldsymbol{B}(\boldsymbol{x}(\boldsymbol{u})))$. If we now specify feature points that refer to triangles that are part of the skin mesh, then these feature points will be a function of the form $\boldsymbol{p}(\boldsymbol{s}(\boldsymbol{B}(\boldsymbol{x}(\boldsymbol{u}))))$, giving us the differentiable Skinned VR-Soft IK objective

$$\mathcal{O}_{\text{IK}}^{\text{skinned}}(\boldsymbol{u}) = \|\boldsymbol{p}(\boldsymbol{s}(\boldsymbol{B}(\boldsymbol{x}(\boldsymbol{u})))) - \boldsymbol{p}'\|^2,$$

which we break down in Figure 4. In our implementation, we compute $\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}}$ analytically from the definition of barycentric coordinates, estimate $\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{B}} \frac{\partial \boldsymbol{B}}{\partial \boldsymbol{x}}$ using finite differences, and compute $\frac{d\boldsymbol{x}}{d\boldsymbol{u}}$ using the adjoint method as in Section II-B.

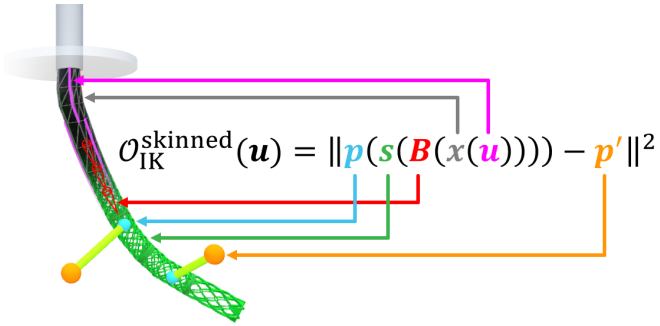

Fig. 4. Color-coded breakdown of the full Skinned VR-Soft IK objective. The robot's cables $\boldsymbol{u}$ (pink) deform the simulation mesh $\boldsymbol{x}$ (gray), which deforms the bones $\boldsymbol{B}$ (red), which deform its high-resolution skin $\boldsymbol{s}$ (green), upon which we specify barycentric feature points $\boldsymbol{p}$ (blue). Minimizing the Skinned VR-Soft IK objective $\mathcal{O}_{\text{IK}}^{\text{skinned}}$ finds control inputs that drive these feature points as close as possible to their user-specified target positions $\boldsymbol{p}'$ (orange). The bones run through the center of the entire mesh (there are 12 total, forming a polyline "spine"), but, for visual clarity, we are drawing only three of them above. Locations of bones are drawn approximately.

B. GPU Feature Point Specification

The final ingredient in our Skinned VR-Soft IK pipeline is a way to interactively specify feature points on the surface of a skinned character mesh. While it might at first seem tempting to apply the same CPU ray-casting scheme we used in Section II-C.1, there is a problem. Because of how we leveraged the GPU to do skinning, we cannot easily access to the character mesh's deformed shape on the CPU. However, we can actually specify feature points *on the GPU* using a technique from real-time graphics [27], [28] (see Figure 5).

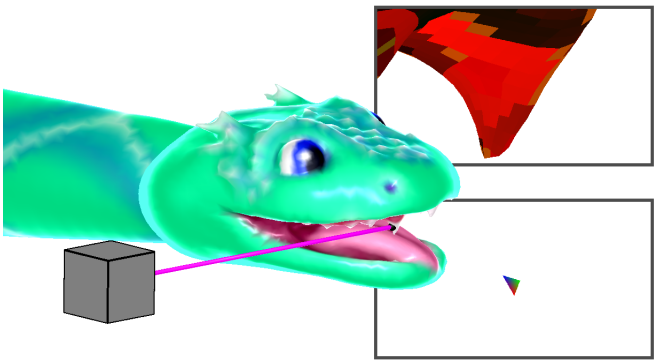

Fig. 5. We employ a GPU-based picking method to enable the user to efficiently specify feature points on the surface of a high-resolution surface mesh as it deforms. In the figure above, we are using GPU-based picking to specify the black feature point on the dragon's tooth. This is done by rendering the mesh twice from the perspective of the ray, which is drawn in magenta. The corresponding virtual camera is drawn as a gray box. In the first render pass (top right), we draw each triangle a different color. Note that while some are visually very similar to our eyes, each shade of red used above is different to a computer. Note that we have used a wider field of view here so that you can see the overall shape of the tooth hit by the ray. The color of a pixel at the center of the render buffer corresponds to the first triangle hit by the ray! In the second render pass (bottom right), we render just the first triangle hit by the ray, coloring its vertices pure red, green, and blue. Again, the color of a pixel at the center of the render buffer is used, this time to find the barycentric coordinates of the feature point.

Here is how we do a GPU raycast. Recall that our goal is to find the first triangle hit by ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$, where $\boldsymbol{o}$ and $\boldsymbol{d}$ are the position and orientation of a VR controller. Additionally, we would like the barycentric coordinates (α, β, γ) of this ray-triangle intersection. To find the triangle first hit by the ray, we render the character mesh from the perspective of a camera with position $\boldsymbol{o}$ and negative z-axis $\boldsymbol{d}$ using a special shader program. This shader program colors every triangle in the mesh a different color, where a triangle's color encodes its unique index i, which we do with the formula

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} i \bmod 256 \\ \lfloor i/256 \rfloor \bmod 256 \\ \lfloor i/256^2 \rfloor \bmod 256 \end{bmatrix} \Big/ 255.0$$

where (R, G, B) are the color's red, green, and blue components respectively, which range from 0.0 to 1.0, and $\lfloor \cdot \rfloor$ is the floor function. The trick is that a pixel at the center of the screen is colored by first triangle hit by ray $\boldsymbol{r}(t)$. We can retrieve that single pixel's color as an 8-bit unsigned integer from the GPU, and find its index with the formula

$$i = \lfloor R/255.0 \rfloor + \lfloor 256G/255.0 \rfloor + \lfloor 256^2 B/255.0 \rfloor.$$

To find the barycentric coordinates of intersection point $\boldsymbol{p}$, we can again leverage the GPU. We render only the triangle we just found, using a special shader program. This shader program colors the triangle's zeroth vertex pure red, its first vertex pure green, and its third vertex pure blue. These vertex colors are barycentrically interpolated across the triangle, and so by again retrieving the location of a pixel at the center of the screen, we can read out the barycentric coordinates $(\alpha, \beta, \gamma) = (R, G, B)/255.0$.

IV. Example: Helix Manipulator

We demonstrate VR-Soft IK on the soft robot manipulator Helix [29], shown in Figure 6 and our Supplementary Video. The robot is composed of a series of six trimmed helicoid elements, forming a 0.9m long structure. The elements were architected to optimize the robot's workspace, robustness to errors in control [30], and desired Cartesian stiffness [31]. The robot's body is divided into three sections, each of which is actuated by three cables. The top section is composed of a single trimmed helicoid segment, and the bottom two sections are composed of two trimmed helicoid segments each. The gripper is a single segment attached to the end. Bowden tubes enable independent actuation of the sections.

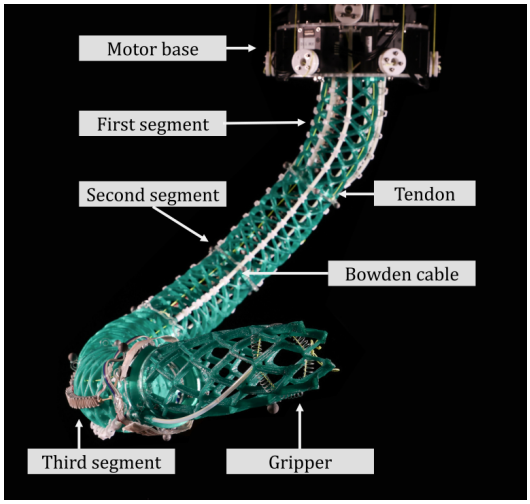

Fig. 6. The soft manipulator Helix has a cylindrical body, which is actuated by nine independently motorized cables. Its body segments are printed from thermoplastic polyurethane (TPU).

A. Evaluating Sim2Real Transfer

To characterize the accuracy of our control pipeline, we performed the tests summarized in Figures 7 to 9. During the experiments, the pose of the Helix robot was captured with a Motion Capture system (OptiTrack Prime 13) and a camera. The control inputs and pose of the simulated robot were also recorded.

We evaluate the accuracy of the VR-Soft IK pipeline in Figure 7, examining the resulting end-effector trajectories for two different target trajectories. First, we tested a hard-coded circular target trajectory for a feature point on the robot's tip, and second, we tested a more organic capture of a user freely specifying targets in VR. For the circular trajectory, the mean absolute error of the end effector's position in Cartesian space was 1.8% of the soft robot's length, with $[\text{mean}, \text{std}] = [15.4\text{mm}, 31\text{mm}]$; for the user-specified trajectory, it was 2.1% of the robot's length.

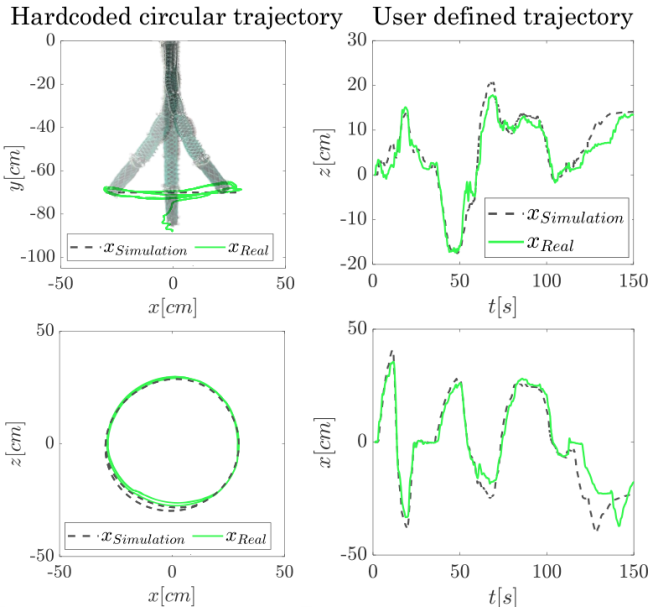

Fig. 7. Trajectory tracking performance on a hardcoded circular trajectory (left) and on user-defined trajectories (right). In the hardcoded circular trajectory (left), the tip target position followed $x = [r\cos(\omega t), h, r\sin(\omega t)]$, where ω being a constant velocity of 0.17 rad/s. The radius of the circle r and the height of the trajectory h were selected so that the trajectory was within the robot's workspace. In the user-specified trajectory (right), the user freely dragged the target points in the VR environment. In the reported measurement, the soft robot was commanded to perform abrupt movements between highly deformed poses, resulting in rapid changes in position along the x and z axes.

In Figure 8, we show the result of solving VR-Soft IK for various user-specified target positions $\boldsymbol{p}'$ and compare how the computed tendon lengths successfully translate to the poses of the real Helix robot. During the experiment, the view angle in the simulation was fixed to match the perspective of the camera. We observe a good qualitative match in the overall shape of the simulated and real robot. Even when target positions are not within the robot's workspace, they are still useful for enabling the user to pose the robot, and the soft robot deforms so as to minimize the squared error between the mesh point and the target. Because the targets are satisfied in this "soft" sense, the user can incrementally move the target points to precisely specify a desired shape within the robot's configuration space.

We are excited by the prospect of using Virtual Reality to make it more intuitive to command robots to do real-world tasks [32]. Our final test involves using VR-Soft IK to enable a user to complete a real-world task using Helix. Thanks to the match between the simulation and reality, the user is able to intuitively use Helix to pop balloons in the real world. This is shown in Figure 9 and in our Supplementary Video.

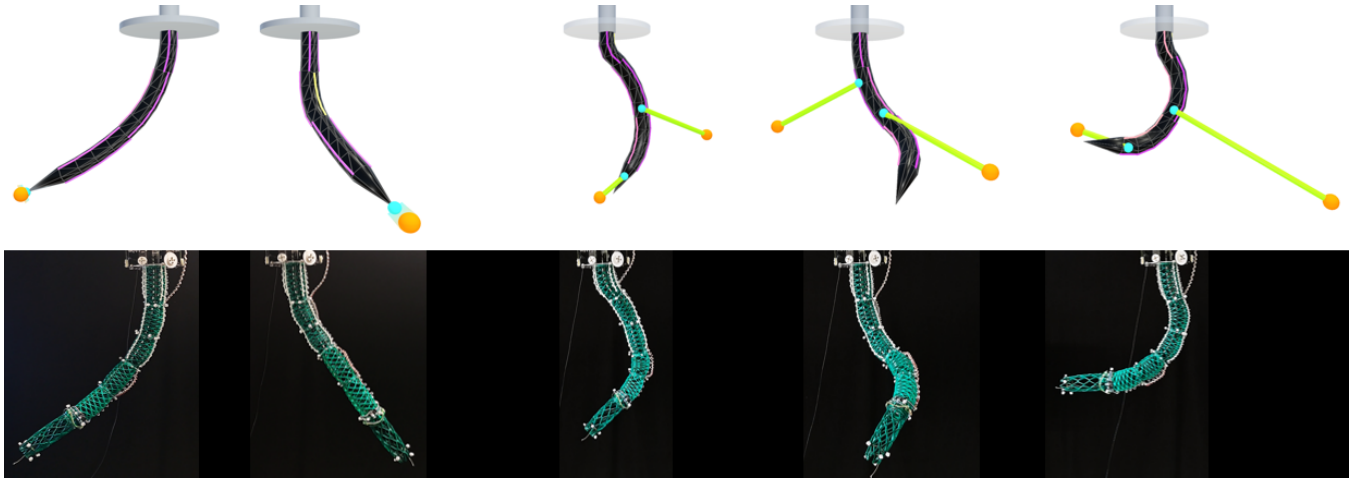

Fig. 8. This figure illustrates the correspondence between simulation and reality. On the top, we show the result of the control input found by VR-Soft IK for the specified feature points in simulations. On the bottom, we show the result of sending those control inputs to the real-world soft robotic manipulator Helix. In the leftmost two target poses, the user has specified just one feature point each, and so the optimization has a much easier time reaching them. The three rightmost target poses use multiple, more extreme feature points. The optimization cannot satisfy them exactly, rather it finds control inputs that drive the robot as close to them as possible.

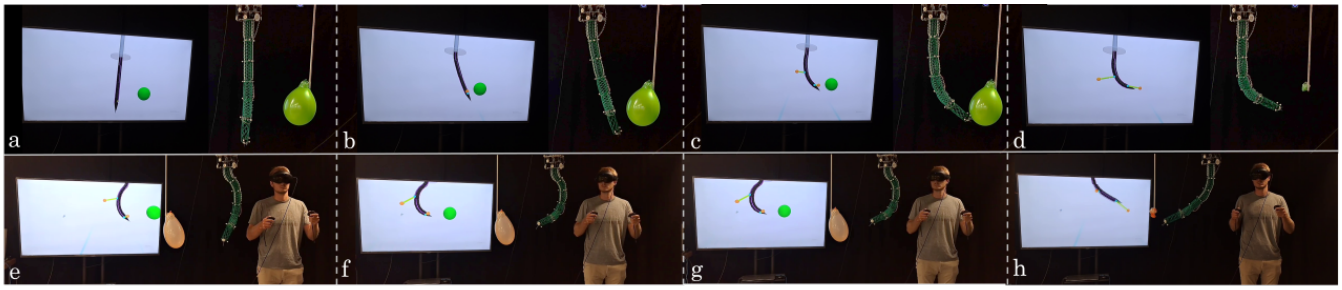

Fig. 9. Here we show a task where the user commands the robot to pop a real-world balloon by using our VR-Soft IK interface. The balloon's location was measured beforehand and a sphere of similar diameter was placed at the corresponding location in the virtual environment. Our overall pipeline strikes a balance between accuracy and efficiency, paving the way toward tasks that cross over between the virtual and physical worlds.

V. Discussion and Future Work

A. System Modularity and Rod-Based Simulators

In our implementation of the Skinned VR-Soft IK pipeline, we modeled a soft robot as a tetrahedral finite element mesh and skinned it using a basic skeletal rig. However, our overall pipeline could be generalized to control robots modeled using rods [30], [33]–[35]. To use our basic, un-skinned pipeline, we need a way of specifying feature points on the simulation. One way to achieve this for a rod-based simulation would be to cast rays at the line segments making up the simulation and take the closest point below some threshold distance. To use our full, skinned pipeline, we need a way of deforming the high-resolution surface mesh based on the simulation mesh. For a rod-based simulation, the rods can do double duty as bones in a skeletal rig.

B. Towards VR-Soft IK With Real-Time Contact

Until this point, Soft IK-type control methods have not considered contact with the environment. This is unfortunate, as one of the great promises of soft *is* their ability to safely contact the environment. One reason for our avoidance of contact is that state-of-the-art contact methods were either insufficiently reliable or too computationally expensive to be practical for real-time control applications. However, Computer Graphics is currently experiencing a renaissance in soft body contact simulation [36], [37]. Some of these methods have started to make their way into Soft Robotics [38], however, but their potential remains largely untapped.

VI. Conclusion

We showed how a VR-based pipeline can enable shape control of arbitrary soft robots. We also showed how a high-resolution skin could be fully integrated into this pipeline, pointing the way towards Mixed Reality Soft Robots (MRSRs). In Mixed Reality, we envision a user intuitively specifying virtual feature points *directly on the surface of a real robot*, seen through VR pass-through. The virtual representations of the robot we have discussed in this paper could, in a sense, "fade into the background," and real-world obstacles and objectives could be made visible. Additionally, there is the potential for the user to "reach out and touch" a mixed reality robot. We are excited about the future of Mixed Reality Soft Robots, especially how they may be applied to medicine and education.

REFERENCES

[1] F. Stella and J. Hughes, "The science of soft robot design: A review of motivations, methods and enabling technologies," *Frontiers in Robotics and AI*, vol. 9, p. 1059026, 2023.

[2] C. Della Santina, C. Duriez, and D. Rus, "Model-based control of soft robots: A survey of the state of the art and open challenges," *IEEE Control Systems Magazine*, vol. 43, no. 3, pp. 30–65, 2023.

[3] C. Duriez, "Control of elastic soft robots based on real-time finite element method," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 3982–3987.

[4] J. M. Bern, G. Kumagai, and S. Coros, "Fabrication, modeling, and control of plush robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3739–3746.

[5] E. Almanzor, F. Ye, J. Shi, T. G. Thuruthel, H. A. Wurdemann, and F. Iida, "Static shape control of soft continuum robots using deep visual inverse kinematic models," *IEEE Transactions on Robotics*, 2023.

[6] W. Liu, Y. Duo, J. Liu, F. Yuan, L. Li, L. Li, G. Wang, B. Chen, S. Wang, H. Yang *et al.*, "Touchless interactive teaching of soft robots through flexible bimodal sensory interfaces," *Nature communications*, vol. 13, no. 1, p. 5030, 2022.

[7] C. G. Frazelle, A. Kapadia, and I. Walker, "Developing a kinematically similar master device for extensible continuum robot manipulators," *Journal of Mechanisms and Robotics*, vol. 10, no. 2, p. 025005, 2018.

[8] A. Jacobson, D. Panozzo, O. Glauser, C. Pradalier, O. Hilliges, and O. Sorkine-Hornung, "Tangible and modular input device for character articulation," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–12, 2014.

[9] S. Jadhav, V. Kannanda, B. Kang, M. T. Tolley, and J. P. Schulze, "Soft robotic glove for kinesthetic haptic feedback in virtual reality environments," *Electronic Imaging*, vol. 2017, no. 3, pp. 19–24, 2017.

[10] B. J. Caasenbrood, F. E. Van Beek, H. K. Chu, and I. A. Kuling, "A desktop-sized platform for real-time control applications of pneumatic soft robots," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2022, pp. 217–223.

[11] "Inside Reality Labs Research: Bringing Touch to the Virtual World — Meta — about.fb.com," https://about.fb.com/news/2021/11/reality-labs-haptic-gloves-research/, [Accessed 07-08-2023].

[12] A. Elor, S. Lessard, M. Teodorescu, and S. Kurniawan, "Project butterfly: Synergizing immersive virtual reality with actuated soft exosuit for upper-extremity rehabilitation," in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 2019, pp. 1448–1456.

[13] M. Chen, J. Yan, and Y. Yu, "Biometric visceral interface: A soft robotic immersive system for extended perception," in *Proceedings of the 25th International Symposium on Electronic Arts*, 2019.

[14] J. M. Bern, P. Banzet, R. Poranne, and S. Coros, "Trajectory optimization for cable-driven soft robot locomotion." in *Robotics: Science and Systems*, vol. 1, no. 3, 2019.

[15] J. M. Bern and D. Rus, "Soft ik with stiffness control," in *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2021, pp. 465–471.

[16] E. I. Borges, J. S. Rieder, D. Aschenbrenner, and R. B. Scharff, "Framework for armature-based 3d shape reconstruction of sensorized soft robots in extended reality," *Frontiers in Robotics and AI*, vol. 9, p. 810328, 2022.

[17] S. Park, S. Bokijonov, and Y. Choi, "Review of microsoft hololens applications over the past five years," *Applied sciences*, vol. 11, no. 16, p. 7259, 2021.

[18] P. Kudry and M. Cohen, "Enhanced wearable force-feedback mechanism for free-range haptic experience extended by pass-through mixed reality," *Electronics*, vol. 12, no. 17, p. 3659, 2023.

[19] K. McIntosh, J. Mars, J. Krahe, J. McCann, A. Rivera, J. Marsico, A. Israr, S. Lawson, and M. Mahler, "Magic bench: a multi-user & multi-sensory ar/mr platform," in *ACM SIGGRAPH 2017 VR Village*, 2017, pp. 1–2.

[20] "CoasterMania on Oculus Quest 2 — oculus.com," https://www.oculus.com/experiences/quest/7856648691073700/, [Accessed 10-09-2023].

[21] F. Kennel-Maushart, R. Poranne, and S. Coros, "Multi-arm payload manipulation via mixed reality," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 11 251–11 257.

[22] ——, "Interacting with multi-robot systems via mixed reality," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 633–11 639.

[23] J. Delmerico, R. Poranne, F. Bogo, H. Oleynikova, E. Vollenweider, S. Coros, J. Nieto, and M. Pollefeys, "Spatial computing and intuitive interaction: Bringing mixed reality and robotics together," *IEEE Robotics & Automation Magazine*, vol. 29, no. 1, pp. 45–57, 2022.

[24] J. M. Bern, Y. Schnider, P. Banzet, N. Kumar, and S. Coros, "Soft robot control with a learned differentiable model," in *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 2020, pp. 417–423.

[25] P. Shirley, "Ray tracing in one weekend," *Amazon Digital Services LLC*, vol. 1, p. 4, 2018.

[26] "XR Grab Interactable — XR Interaction Toolkit — 2.0.4 — 2.0," https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-grab-interactable.html, [Accessed 02-08-2023].

[27] T. Akenine-Moller, E. Haines, and N. Hoffman, *Real-time rendering*. AK Peters/crc Press, 2019.

[28] "Tutorial 29 - 3D Picking — ogldev.org," https://ogldev.org/www/tutorial29/tutorial29.html, [Accessed 31-07-2023].

[29] Q. Guan, F. Stella, J. Leng, C. Della Santina, and J. Hughes, "Trimmed helicoids: An architectured soft structure yielding soft robots with high precision, large workspace, and compliant interactions," *Nature npj Robotics*, 2023.

[30] F. Stella, Q. Guan, C. Della Santina, and J. Hughes, "Piecewise affine curvature model: a reduced-order model for soft robot-environment interaction beyond pcc," in *2023 IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 2023, pp. 1–7.

[31] F. Stella, J. Hughes, D. Rus, and C. Della Santina, "Prescribing cartesian stiffness of soft robots by co-optimization of shape and segment-level stiffness," *Soft Robotics*, 2023.

[32] A. Phung, G. Billings, A. F. Daniele, M. R. Walter, and R. Camilli, "Enhancing scientific exploration of the deep sea through shared autonomy in remote manipulation," *Science Robotics*, vol. 8, no. 81, p. eadi5227, 2023.

[33] N. N. Goldberg, X. Huang, C. Majidi, A. Novelia, O. M. O'Reilly, D. A. Paley, and W. L. Scott, "On planar discrete elastic rod models for the locomotion of soft robots," *Soft robotics*, vol. 6, no. 5, pp. 595–610, 2019.

[34] N. Naughton, J. Sun, A. Tekinalp, T. Parthasarathy, G. Chowdhary, and M. Gazzola, "Elastica: A compliant mechanics environment for soft robotic control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3389–3396, 2021.

[35] S. Duenser, J. M. Bern, R. Poranne, and S. Coros, "Interactive robotic manipulation of elastic objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3476–3481.

[36] L. Lan, G. Ma, Y. Yang, C. Zheng, M. Li, and C. Jiang, "Penetration-free projective dynamics on the gpu," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–16, 2022.

[37] M. LI and C. JIANG, "Second-order stencil descent for interior-point hyperelasticity," *ACM Trans. Graph*, vol. 1, no. 1, 2023.

[38] J. M. Bern, L. Z. Yañez, E. Sologuren, and D. Rus, "Contact-rich soft-rigid robots inspired by push puppets," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2022, pp. 607–613.