IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.xxxxxxx

Sorotoki: A Matlab Toolkit for Design, Modeling, and Control of Soft Robots

BRANDON J. CAASENBROOD[1], ALEXANDER Y. POGROMSKY[1], AND HENK NIJMEIJER[1] (Life Fellow, IEEE).

[1]Department of Mechanical Engineering, Eindhoven University of Technology, de Rondom 71, Eindhoven, 5600MB, Noord Brabant, the Netherlands (e-mail: b.j.caasenbrood@{tue.nl; gmail.com}, a.pogromski@tue.nl, h.nijmeijer@tue.nl)

Corresponding author: Brandon Caasenbrood (e-mail: b.j.caasenbrood@gmail.com).

This work is partly supported by NWO, Netherlands Organization for Scientific Research; and is part of the Wearable Robotics perspective program under the name *Soft Structures and Actuators* (P15-06: P2). Website: `www.wearablerobotics.nl`

ABSTRACT In this paper, we present Sorotoki, an open-source toolkit in MATLAB that offers a comprehensive suite of tools for the design, modeling, and control of soft robots. The complexity involved in researching and building soft robots often stems from the interconnectedness of design and control aspects, which are rarely addressed together as a unified problem. To address such complex interdependencies in soft robotics, the Sorotoki toolkit provides a comprehensive and modular programming environment composed of seven Object-Oriented classes. These classes are designed to work together to solve a wide range of soft robotic problems, offering versatility and flexibility for its users. We provide here a comprehensive overview of the Sorotoki software architecture to highlight its usage and applications. The details and interconnections of each module are thoroughly described, collectively explaining how to gradually introduce modeling complexity for various soft robotic scenarios. The effectiveness of Sorotoki is also demonstrated through a range of case studies, including novel problem scenarios and established works widely recognized in the soft robotics community. These case studies cover a broad range of research problems, including: inverse design of soft actuators, passive and active soft locomotion, object manipulation with soft grippers, meta-materials, model reduction, model-based control of soft robots, and online shape estimation. Additionally, the toolkit provides access to four open-hardware soft robotic systems that can be fabricated using commercially available 3D printers. For more information about Sorotoki, readers are encouraged to visit: `https://bjcaasenbrood.github.io/SorotokiCode/`

INDEX TERMS Soft robotics, software, design, modeling, control, matlab

I. INTRODUCTION

SINCE the early 1980s, roboticists have been developing fluid-driven robots inspired by biological systems. Examples include the pneumatic three-link soft robot manipulator developed by Wilson et al. [1]–[3] and the fluidic four-fingered soft gripper presented by Suzumori et al. [4], [5], which both showing high dexterity for advanced object manipulation without the need for advanced (contact-aware) controllers. Although the design and control of these robots were simple, their level of dexterity and adaptability was previously unseen in rigid robotics and strongly resembled biological systems. These benefits were achieved through the use of "*soft materials*" paired with fluidic actuation, where "*soft*" refers to the collective mechanical properties of highly compliant materials such as flexibility, compressibility, and mechanical robustness. Just as nature exhibits diverse evolutionary solutions to environmental stimuli during locomotion and manipulation, soft materials possess a plethora of beneficial mechanical properties that can be applied to robotics. Today, the philosophy of building robots from soft materials has significantly matured and has become a well-recognized field known as "*soft robotics*". Although definitions may vary depending on one's scientific background, a formal definition found in the Encyclopedia of Robotics [6] states that "*soft robots are robotic systems with purposefully designed compliant elements embedded into their mechanical structure*." While mechanical compliance may compromise speed and precision (see Section I-A), it offers viable solutions to issues commonly encountered with traditional robots [7].

A major benefit over its rigid counterpart is that soft robots are less likely to cause injury during collisions, making them a more suitable alternative for tasks involving close human-robot interaction. Second, soft robots possess the ability to adapt to unstructured environments and manipulate a diverse array of objects through their ability to change shape and conform adaptively to their surroundings [8], [9]. These features are somewhat analogous to those found in nature, such as the trunk of an elephant grasping tree branches or the tentacles of an octopus squeezing through narrow spaces. Soft materials can be tuned to allow for delicate grasping [10], [11] or high-power densities [12]. Additionally, their high adaptability paired with tunable low compliance make them extremely robust towards abrupt impacts or high compression forces [13], making them suitable for various tasks in harsh environments without breaking [14]–[16]. Moreover, soft robots are typically constructed from low-cost materials and fabricated through straightforward manufacturing processes, such as rubber casting, making them more cost-effective compared to traditional rigid robots. With recent advances in soft material Additive Manufacturing (AM), soft robots can even be fully 3D-printed [17], [18]. This not only reduces production time and cost, but also allows for the embedding of printed onboard logic [19], [20]. Additionally, soft robots tend to be lightweight, making them easier to handle and transport, making them ideal for wearable robotics.

A. PROBLEM FORMULATION

Although significant progress has been made since their inception, generalized solutions for the design and control of soft robots are still lacking in comparison to those available for rigid robots. This can be partially attributed to the inherently nonlinear and high-dimensional nature of the mathematical descriptions for deformable robotic bodies composed of compliant materials. This presents major challenges in finding suitable models that enable fast simulation, which ultimately hinders efficient structural design and model-based controller design. Despite the numerous challenges in soft robotics, two major research trends can be recognized within the soft robotics community:

A: Design of soft actuators and sensors. A majority of the soft robots are actuated in two ways [7]: (i) local actuation through variable length tendons (e.g., cables [21] or shape-memory alloy wires [22]) or (ii) distributed actuation through responsive soft materials [23] or surface loads using fluidics [14], [24], [25], commonly implemented as fluidic networks embedded within the soft body. The latter method is often referred to as Fluidic Elastomer Actuators (FEAs) or Soft Fluidic Actuators (SFA). FEAs can be designed through either geometric asymmetries in their structural design or by incorporating a composition of different materials, such as fibers or meshes, that induce the desired deformation when pressurized. While FEA designs date back many decades, there remain significant gaps in understanding and applying established engineering principles to their design. FEAs frequently experience large deformation when actuated, leading to slow actuation due to material relaxation or, in more severe cases, fatigue or tearing caused by ballooning [9] – an almost inherent effect in elastomers. Furthermore, low-compliance soft elastomer actuators often undergo parasitic deformation when exposed to external forces, like gravitational load. To efficiently solve the design cycle in soft actuators, it is essential to have a comprehensive understanding of the nonlinear deformation characteristics of soft materials under static and dynamic conditions.

In parallel, proprioceptive soft sensing technology is still in early stages. The high compliance of soft robots often makes it difficult to apply common embedded sensors, such as encoders, capacitive sensors, strain gauges, and inertial sensors. These sensors are well-suited for rigid robots with articulated joints, as they are effective in measuring local joint displacement. However, in soft robots, displacement is often distributed, rendering these sensors less suitable. Furthermore, these sensors must be designed to minimize their impact on mechanical impedance, in order to minimize changes to the structural dynamics and operational workspace. A common approach is to incorporate microfluidic channels filled with a conductive liquids [26]–[28], such as Eutectic Gallium-Indium (EGaIn), into the soft body, that are placed antagonistic to the soft actuator. Upon deformation, the resistance changes, allowing for the correlation of specific deformation profiles with the soft robot. Other solutions incorporate integrated Hall sensors to measure changes in the magnetic field of ferromagnets distributed throughout the body [29], [30], or utilize fiber-optic grating bending sensors [31]. Generally, the correlation between sensor output and deformation is complex, often necessitating the collection of extensive *a-priori* measurement data, which may be synthetic, to be mapped onto motion primitives.

The geometry of soft actuators and sensors plays a crucial role in determining their functionality and performance. Currently, most soft robotic components are designed using Computer-Aided Design (CAD) software similar to those used for rigid robots. However, as the geometric complexity of soft robots increases, particularly with the increasing trend towards bio-inspired and 3D-printed designs, there is a growing need for software that can handle free-form designs and have predictive capabilities for soft material deformation.

B: Modeling and control of soft robots. With the aim of achieving comparable performance to rigid robots and eventually biological creatures, there is a strong demand for advanced closed-loop control in soft robotic systems. However, the challenges in soft actuation and sensing extend directly to several modeling and control paradigms for soft robots [32], [33]: (i) their high dexterity and adaptability are challenging to incorporate into a modeling framework, and (ii) due to their continuum elastic bodies composed of a finite number of actuators and sensors, soft robots are inherently under-actuated and under-sensed, a problem common to infinite-dimensional systems (e.g, continuum systems). In regard to the closed-loop control of soft robots, the field has introduced two distinct branches that can competently address the relevant issues.

First, model-based control uses first-principle mathematical models of the system being controlled to design and implement controllers. The derived models often have conventional structures, e.g., Lagrangian or Port-Hamiltonian, that is (closely) analogous to classic rigid robotics [32]–[35]. As such, they extend (with minor modification) to existing control strategies including model-based feedback control [25], [36], impedance control [37], adaptive control [38], [39], iterative learning control [40], and energy-shaping control [41]–[44]. Also, model-based approaches provide physical interpretations of the control gains, making controllers more transparent in terms of stability guarantees. Nonetheless, for some scenarios, first-principle modeling approaches (i.e., Euler-Lagrange methods) will not suffice. For example, during environment or self-contact, it can be challenging to select a finite-dimensional state representation of the soft robotic model that balances precision and computational efficiency. Also, *a-priori* unknown system uncertainties, such as unreliability of sensors and actuators, model mismatches, and time-varying parameters, can impede the approach altogether.

Second, data-driven modeling focuses on utilizing existing data to establish the connection between inputs and outputs, instead of depending on prior knowledge or assumptions to formulate a theoretical model. The training data can be derived from measurements or high-fidelity surrogate models, which empowers the model to adapt as new information becomes available, similar to adaptive control. Despite often being black-box or grey-box approximators, these methods can be fine-tuned for low dimensionality, rendering them highly efficient for simulation purposes. As an alternative, synthetic data generated from digital environments can be used to train learning controllers, e.g., in Reinforcement Learning (RL) [45], [46]. Model-Predictive Control (MPC) can also be applied within a data-driven framework [47], [48]. However, in both cases, it is crucial that the training data is comprehensive enough to encompass the entire dynamic workspace. This requirement prevents the generalizability of the method to be applicable to unseen scenarios and necessitates retraining for specific control objectives. Additionally, control policies learned through virtual environments may not be effective in the physical system due to differences with reality, known as the Simulation-to-Reality (Sim2Real) barrier [49].

B. CONTRIBUTION OF SOROTOKI SOFTWARE

To address some of these challenges, we introduce `Sorotoki` (short for Soft Robotics Toolkit), an open MATLAB® toolkit for soft robotics that offers a range of tools for design, modeling, and control. `Sorotoki` aims to reduce barriers to entry in the field of soft robotics by providing a comprehensive software package that integrates various layers of modeling and control approaches, including continuum mechanics, dynamic systems and control theory, topology optimization, computer graphics, real-time control, and vision-based sensing. These diverse capabilities provide a highly flexible programming environment that can facilitate the development of innovative soft robotics research. The main feature of the `Sorotoki` are listed below:

1) **Design and fabrication** – Implicit modeling using Signed Distance Functions (SDFs), mesh generation, computational design, STL generation for 3D printing;
2) **Modeling and control** – Finite element models, efficient reduced-order soft beam models (Lagrangian or port-Hamiltonian), programmable interconnections of a network of dynamic systems, e.g., soft robots, pressure vessels, and inertial rigid bodies;
3) **Actuation and sensing** – Real-time, high-precision, fluidic control platform using Raspberry Pi, vision-based sensing using RGB-depth camera;
4) **Visualization** – Fast and responsive 3D graphics rendering, mesh deformation modifiers, FK/IK-rigging.
5) **Accessibility** – A minimal programming syntax, characterized by the ability to express complex problems with a minimal number of lines of code.
6) **Open hardware** – Four 3D-printable soft robots (e.g., soft hand, soft manipulator).

We briefly detail the organization of the paper. Section II reviews existing open-source soft robot software packages. Section III then assists the reader in getting started with the toolkit and introduces the open-source soft robotic systems. In Section V, we detail the software architecture, the theory underlying the `Sorotoki` functions, and how the theory can be applied through coding examples in `Sorotoki`. Once the reader is familiar with the basic software architecture, Section VI presents advanced study cases based on seminal works in soft robotics research. The paper concludes in Section VII with a summary and outlook for future work.

II. RELATED WORKS

Over the past two decades, significant advancements have been made in the field of soft robotics. To support the growing community, researchers in the field have made efforts to provide open-source software tools alongside their scientific contributions. This section provides a review of related work on open-source software packages for soft robotics, comparing these software packages and discussing how `Sorotoki` addresses any gaps in functionality.

One widely used tool is the `SOFA` (Stand Alone Open Framework for Animation) software [50], which is an open-source framework for real-time physically-based simulations of deformable mechanical systems. Relevant to soft robotics, `SOFA` is commonly used to simulate the behavior of soft robots and to design and test control algorithms on real platforms [50], [51]. SOFA employs the Finite Element Method (FEM) to describe the continuum deformations of inertial elastic bodies, a numerical technique that solves partial differential equations (PDEs) that describe physical systems by dividing the domain of the system into small elements and approximating the PDEs with a set of algebraic equations [52], [53]. FEM models generally provide high-accuracy volumetric deformation simulations of soft materials, but their high state dimension, which can often be in the thousands or millions of degrees of freedom, can render them computationally expensive for state feedback. To enhance efficiency, Goury et al. [54] have explored model reduction using snapshot Proper

Orthogonal Decomposition (POD).

Snapshot POD [55], [56] is a method for significantly reducing the dimensionality of a model by collecting snapshots of its state and utilizing Singular Value Decomposition (SVD) to identify the principal components. The projection is achieved by taking a linear combination of the principal components, weighted by their corresponding coefficients (also known as "*modes*"). The resulting projection is then a reduced-order model of the original system, which can be used for faster simulation. In addition to improving speed, this approach also provides accurate, robust, and efficient models suitable for closed-loop controller design [57]–[60]. The numerical FEM models in `SOFA` incorporate both the structural geometry and material properties, facilitating easy transfer of control policies to a physical system. This has enabled successful control synthesis using `SOFA` in various experimental settings [58], [61]. `SOFA` also includes tools for real-time visualization and data analysis, making it a valuable platform for testing and debugging control algorithms. Recently, Schegg et al. [45] introduced an interface between `SOFA` and `OpenAI` called `SofaGym`. This wrapper enables the training of reinforcement learning (RL) policies using real-time simulation models, and it incorporates model reduction to further improve the efficiency of RL that otherwise suffer from computationally-intensive simulations.

Another software package that utilizes the nonlinear finite element approach similar to `SOFA` is the `Gibbon` toolbox, developed by Moerman et al. [62]. `Gibbon` is a MATLAB-based pre-processor and post-processor for `FEBio` [63]. The toolkit has recently been used to solve the nonlinear deformation of bending soft pneumatic actuators using finite shell elements [64], generate designs using a multi-objective heuristic [65], and analyze soft bending actuators composed of an adaptive fiber-elastomer composite [66]. It also features various tools for image segmentation, meshing, and visualization, with a focus on biomedical engineering.

Despite the availability of open-source FEM packages for modeling and controlling soft robots, challenges still exist in using FEM for design-based optimization of these systems, particularly due to the complexity of hyper-elastic materials and unconventional (fluidic) actuation, which are popular themes within the field. Currently, there are limited options for frameworks that effectively and efficiently address these issues, although recent developments are promising. Smith et al. [67] recently proposed a versatile free-form design and fabrication workflow called `SoroForge`, which builds upon [64], [65]. Unlike volumetric representation, their approach can design complex soft actuator exteriors using a highly-flexible and fast node-tree interface of implicit function primitives. However, these generative CAD solutions are limited to only addressing quasi-static deformations and do not consider the deformation induced by control.

There are several software packages specifically designed for the dynamic locomotion of soft robots, which take into account the structural design, actuator placement within the soft body, control actions, and even adapt the body's topology accordingly. One such example is `EvoSoro` developed by Kriegman et al. [49], which builds on the work of Hiller et al. [68] and Cheney et al. [69]. This study discretizes a soft continuum body into small voxels, which can be assigned different cell types: soft or hard passive cells, or two different muscle cell types that undergo periodic contraction with an $+\pi$ phase offset. The dynamic behavior of the system is modeled through a network of mass particles and springs, and a Compositional Pattern-Producing Network (CPPN) is utilized to determine the optimal combination of material type and placement within a specified domain, enabling locomotion. This concurrent optimization of topology and control policy, referred to as *"co-design"*, is a subject of active research within the field of soft robotics [18]. Another example of co-design in soft robotics is `EvoGym` [70], which optimizes for a wide range of tasks such as locomotion and object manipulation (e.g., carrying and throwing). More recently, `DiffTachi` (the successor to `QueenChain` [71]) is a differential programming environment that allows users to directly provide gradient-based information into a neural network controller using a least-squares Material Point Method (MPM). Unlike FEM, MPM is a mesh-free approach that describes the continuum using a finite number of hybrid Euler-Lagrangian elements referred to as"*material points*". In terms of learning control policies, `SoMoGym` by Graule et al. [72] uses reinforcement learning (RL) to teach locomotion and object manipulation in soft robots, and has successfully bridged the gap between simulation and reality (Sim2Real). It is important to note that `QueenChain`, `DiffTachi`, and `SoMoGym` focus purely on learning control and not design. Simultaneous optimization of (free-form) design and control for soft robots remains an open challenge.

Parallel to volumetric-based FEM or MPM soft robotic software, there also exists a branch of dynamic beam (or rod) models for soft robots. These beam model approaches for soft robots have long been a viable alternative to FEM-based models – examples include the Piecewise-Constant Curvature (PCC) model [25], [77], [78], the augmented PCC rigid-body model [37]–[39], and various non-constant curvature descriptions [37], [79]–[81]. As the formulations of these models are often synonymous to rigid robot models, they have a rich basis of control-oriented research [44], [78], [82], [83]. `TMTDyn` by Sadati et al. [84] is a MATLAB toolkit that automates the derivation of dynamic models for hybrid rigid-continuum body soft robots, based on discretized lumped systems and reduced-order models. More recently, `SoroSim` was developed by Mathew et al. [75], which is a MATLAB toolbox with a graphical user interface for modeling, analysis, and control of soft, rigid, and hybrid robots. `SoroSim` is based on the Geometric Variable Strain (GVS) approach applied to the geometric Cosserat beam theory in SE(3), introduced relatively recently by Renda et al. [81] and Boyer et al. [80]. Its Lagrangian formulation also allows for various complex control designs, such as a geometrically-exact inverse kinematic controller that accounts for under-actuation of tendon-based actuation [75]. `SoroSim` has been used for dynamic

TABLE 1. Comparison between different open-source software provided by the soft robotics community that are tailored either towards design, modeling, or control of soft robots. *Inverse design here refers to automated algorithms that freely optimize the topology of the soft body. †Sim2Real here implies the software has been used on a real soft robotic platform – either successfully transferring open or closed-loop control policies to reality; or optimized design solution to real soft systems.

Software	Front-end	Model(s)	Hyperelastic	Tendon	Fluidic	Locomotion	Manipulation	Inverse Design*	Control	Sim2Real†
`SOFA` [51], [73]	Python	(Reduced) FEM	✓	✓	✓	✓	✓	✗	✓	✓
`Gibbon` [62]	Matlab	FEM	✓	✓	✓	✗	✗	✓	✗	✗
`SoRoForge` [67]	Matlab	Shell-FEM	✓	✗	✓	✗	✗	✓	✗	✓
`SoftIK` [74]	C++	FEM	✓	✓	✗	✓	✓	✗	✓	✓
`EvoGym` [70]	Python	Mass-spring	✗	✗	✗	✓	✓	✓	✗	✗
`EvoSoro` [49], [69]	Python	Mass-spring	✗	✗	✗	✓	✓	✓	✗	✓
`DiffTachi` [71]	C++	MPM	✗	✗	✗	✓	✓	✓	✗	✗
`TMTDyn` [75]	Matlab	ROM + EBA	✗	✓	✓	✗	✓	✗	✓	✓
`SoroSim` [75]	Matlab	Cosserat	✗	✓	✓	✓	✓	✗	✓	✓
`(Py)Elastica` [76]	Python	Cosserat	✗	✓	✗	✓	✓	✗	✓	✗
`SoMoGym` [72]	Python	Rigid-link	✗	✗	✓	✓	✓	✗	✓	✓
`Sorotoki` (ours)	Matlab	FEM + Cosserat	✓	✓	✓	✓	✓	✓	✓	✓

models of flexible flying rods, hybrid rigid-soft manipulators, design optimization for soft robot swimmers, and inverse dynamic control. The toolbox has also been employed for the simultaneous swimming and grasping dynamics of underwater soft-rigid hybrid robots [85].

Alternatively, `PyElastica` (a wrapper for `Elastica` [86]) by Tekinalp et al. [46] is an open-source software package written in `Python` that provides the capability to simulate an assembly of Cosserat beams. The software is based on the work of Gazzola et al. [76]. Unlike the Geometric Variable Strain (GVS) approach proposed in [75], [80], [81], `PyElastica` employs a discrete formulation of the Cosserat partial differential equation (PDE) through a finite rod elements referred to as Discrete Elastic Rods (DERs). Additionally, its `c++` architecture enables it to handle problems with higher computational complexity. As demonstrated in [87], `Elastica` has been used to model snake muscular systems, bird wing flapping, and bio-hybrid robots using soft contractile filaments. Furthermore, `PyElastica` has been utilized in energy-based control and even extended to model the full muscular-skeletal system of an octopus' tentacle modeled as an collection of Cosserat beams [88], and mimic biological movements accordingly [89]. It also provides wrappers for `OpenAI` to enable reinforcement learning.

To summarize, Table 1 provides an overview of the functionalities of the previously discussed software packages for soft robotics. The table highlights the diversity of software options available, each with specialized solution approaches for specific sub-problems in soft robotics. However, there are limited tools that address the combined issues of design and control in an interdependent, holistic manner. For instance, developing a model-based controller for a specific soft robot heavily relies on its structural geometry, material composition, network of soft actuators, and their dynamic interaction with the soft body. On the other hand, finding a suitable soft robot design requires *a-priori* knowledge of the material properties and their deformation under the admissible control inputs. This complex interplay between design, modeling, and control makes studying and developing soft robots a challenging task. `Sorotoki` aims to address these challenges by integrating many different scientific disciplines of design and control into a unified toolkit.

III. GETTING STARTED WITH SOROTOKI

In the following section, we briefly detail a starter's guide for `Sorotoki`. The software package `Sorotoki` is available via `mpm` (Matlab Package Installer [90]):

```
mpm install sorotokicode
```

To install the toolkit, request the documentation, or preview demonstrations of the toolkit, we call

```
sorotoki install  % install sorotoki
sorotoki build    % build mex files
sorotoki test     % run testsuite
sorotoki doc      % online documentation
```

in the MATLAB command prompt, respectively. The online documentation[1] provides information on the features and capabilities, installation instructions, general use, syntaxing and a selection of toy examples. The documentation assists users in comprehending and efficiently navigating the software for their research purpose, and it serves as a complement to the work presented herein.

IV. OPEN-SOURCE SOFT ROBOTS OF SOROTOKI

Aside from software, we also present a selection of open-source soft robotic systems, see Figure 1, as part of the

[1]The online documentation and installation instructions can be accessed at the following https://bjcaasenbrood.github.io/SorotokiCode

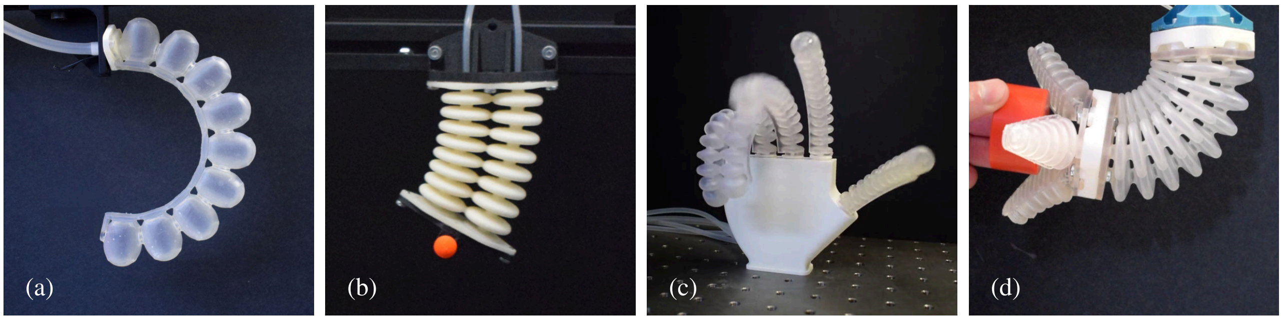

FIGURE 1. Open-source soft robots and soft actuators that are included within the Sorotoki toolkit. All systems are fully 3D-printed using either Selective Laser Sintering (SLS) or Stereolithography (SLA) and their 3D files can be found on the repository. All systems are driven by pneumatics. (a) A two-bellow soft robot suitable for planar motion. (b) An optimized PneuNet bending actuator. (c) A soft robotic hand composed of five soft bending actuators, whose fingers are easily replaceable. (d) A three-bellow soft robot manipulator with a mounted soft gripper at the end-effector. The center axis is hollow, allowing for electronic cables when compact sensors (e.g., IMUs) are mounted on the soft gripper.

`Sorotoki` toolkit. These systems all feature fluidic actuation and can be fabricated using conventional additive manufacturing techniques such as Selective Laser Sintering (SLS), Stereo-LithogrAphy (SLA), or Direct Light Projection (DLP). FormLabs Elastic 80A™ resin or a flexible TPU with a shore hardness of less than 80A are suggested for the deformable bodies in SLA/DLP and SLS printing, respectively. For further details concerning the SLS/DLP manufacturing process, consult Proper et al. [91]. The 3D models (in `.stl` format) are publicly available through `mpm`:

```
mpm install sorotokibots
```

A brief description for each soft robot is provided below.

1) Soft bending actuator

The first system is a soft bending actuator (Figure 1a), an alternative to the PneuNet actuator proposed by Mosadegh et al. [92]. Like PneuNets, our soft actuator consists of an array of bellows placed on a relatively inextensible medium. The stiffness gradient in the actuator allows for pure bending to occur when the network of bellows is pressurized. The geometry of the PneuNet-based soft robot was optimized using `Sorotoki`'s topology optimizer, which was specifically tailored for use with FormLabs Elastic 80A resin. The soft actuator is fully 3D-printed using SLA and can accept pressures in the range of $-10 \leq u \leq 100$ kPa at its central pressure input.

2) Planar soft actuator

The second system is a planar soft actuator that comprises two pneumatic bellow networks that are connected in parallel (Figure 1b). Similar to the previous soft actuator, bending occurs due to a pressure differential between the two pneumatic networks. However, the system is also capable of pure elongation and contraction if the pressure in both networks is equal. This enhances the motion capabilities of the soft robot, enabling it to move within a planar workspace of approximately 100×100 mm. The system has two pressure range of $-10 \leq u \leq 50$ kPa.

3) Composable soft robotic hand

The third system provided by `Sorotoki` is a soft robotic hand with a higher level of complexity compared to the previous soft robots (Figure 1c). This design is inspired by the work of Laake et al. [93] and Fras et al. [24]. The soft robotic hand consists of five independently controlled soft fingers that can be actuated using pneumatics or fluidics. Each finger is fabricated using SLA with Elastic 80A, while the base is fabricated using FDM with PLA. The dimensions and scale of the soft robotic hand are similar to those of a human hand, with approximate dimensions of $190 \times 100 \times 40$ mm. All fingers have a length of 90 mm except for the thumb, which is slightly shorter at 80 mm. The soft robotic hand has five seperate inputs that accept pressures of $-10 \leq u \leq 60$ kPa.

4) Full soft manipulator with soft gripper

The final soft robot provided by `Sorotoki` is a soft robotic manipulator that features three independent bellow networks and a three-fingered soft robotic gripper attached to the end effector (Figure 1d). With independent actuation of each bellow network, the manipulator has a full 3D workspace of approximately $150 \times 150 \times 150$ mm. The soft elements are fabricated using Elastic 80A resin, while the rigid connector pieces are made using Rigid 10K resin. The gripper has demonstrated the ability to successfully grip objects with a diameter of 40 mm, with a maximum payload of 100 g without significant parasitic deformation. The central axis is hollow, enabling the pneumatic tubing of the gripper and the cables for state estimation sensors (e.g., IMUs) to be embedded. The manipulator has three inputs that accepts $-10 \leq u \leq 30$ kPa, and the gripper accepts $-30 \leq u \leq 60$ kPa.

V. SOFTWARE ARCHITECTURE

In this section, we will present the software architecture of the `Sorotoki` toolkit. The toolkit consists of seven Object-Oriented classes, each designed to address a specific sub-problem within the field of soft robotics. We will introduce each class in the following sequence:

- In Section V-A we will discuss the class `Sdf`: a Signed Distance Function (SDF) class that are used to build spatial geometries – *"Implicit CAD"*;

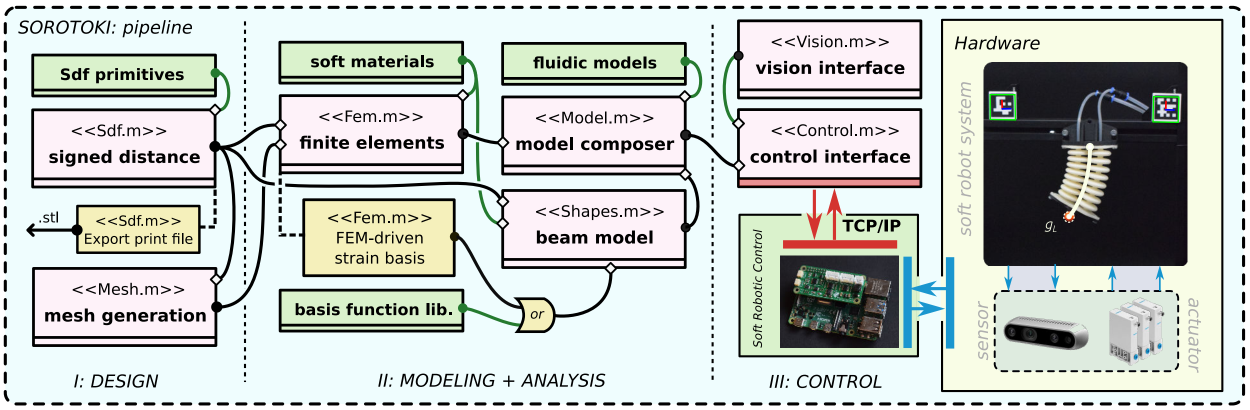
SOROTOKI: pipeline
Sdf primitives
<<Sdf.m>> signed distance
.stl
<<Sdf.m>> Export print file
<<Mesh.m>> mesh generation
soft materials
<<Fem.m>> finite elements
<<Fem.m>> FEM-driven strain basis
basis function lib.
or
fluidic models
<<Model.m>> model composer
<<Shapes.m>> beam model
<<Vision.m>> vision interface
<<Control.m>> control interface
TCP/IP
Soft Robotic Control
Hardware
soft robot system
sensor
actuator
I: DESIGN
II: MODELING + ANALYSIS
III: CONTROL

FIGURE 2. The Sorotoki software toolkit is structured according to a problem-solution pipeline, consisting of seven Object-Oriented classes that address common subproblems in the field of soft robotics research. These classes are: Sdf, Mesh, Fem, Shapes, Model, Control, and Vision. The software architecture flowchart employs the symbol (•) to represent class outputs and the symbol (⋄) to denote inputs.

- In Section V-B we will discuss `Mesh` responsible for mesh generation for the finite element solvers;
- In Section V-C we discuss the class `Fem`: a finite element solver required for high-detail soft robot simulations;
- In Section V-D, we detail the class `Shapes` responsible for beam models used for fast (real-time) simulations;
- In Section V-E we explain `Model` – a model composer to interconnect models, and the control synthesis;
- Following, in Section V-F, we highlight `Control` that serves as a control interface for fluidic platform communicating to MATLAB® via TCP/IP;
- Finally, Section V-G will explain `Vision` – a Vision-based tool for state estimation through optical markers.

To assist the reader, we have included a software architectural flowchart in Figure 2. The flowchart demonstrates how the classes can be interconnected to increase system complexity while maintaining the structured and separable nature of the subproblems. At first, the interconnectivity may seem complex; thus, to aid the reader's comprehension, let us consider some common questions asked by soft roboticists:

- Q1: How do I design a soft robot (parametrically)?
- Q2: How do I analyze its deformation behavior?
- Q3: Can I run simulation models in real-time?
- Q4: How do I apply model-based control on a soft robot?

In response, Figure 3 provides a brief answer through a distilled software architecture of the seven `Sorotoki` classes. Classes can be combined to tailor the specific needs of its users. Throughout the paper, we will explain the functionality of each class, together with illustrative examples and corresponding MATLAB scripts. The topics covered include design, modeling and analysis, model reduction, and control and vision-based sensing, presented chronologically.

A. SIGNED DISTANCE FUNCTION

Signed Distance Fields (SDFs) have been widely applied in various areas of computer graphics, including the representation of implicit surfaces [94], [95], collision detection in robotics [96], [97]. In particular, SDFs have gained attention for their use in implicit modeling [67], a technique for representing 3D shapes as continuous functions, rather than discrete mesh descriptions.

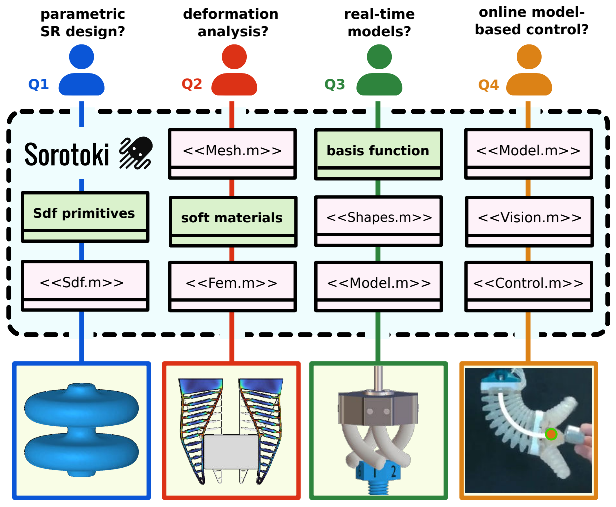

FIGURE 3. Distilled software architecture that highlights Sorotoki classes and solution pipelines based on common question prompts provided by soft robotics students and researchers.

In `Sorotoki`, SDFs are implemented in the class `Sdf.m` and can be used to construct general 2D and 3D geometries. They can also be utilized to model static or dynamic contact environments, generate 3D models of soft actuators that are suitable for 3D printing, and compute inertia tensors for continuum bodies in $\mathbb{R}^2$ and $\mathbb{R}^3$.

1) Implicit modeling using SDFs.

In this section, we briefly outline the mathematical foundations underpinning the `Sdf` class. As the name suggests, signed distance functions are a type of function that encodes distance information relative to an object defined implicitly. Adopting the notation used in [94], given a domain $\Omega \subset \mathbb{R}^n$ and its boundary $\partial\Omega$, these signed distance functions can be written in the following general form:

$$\texttt{sdf}(\boldsymbol{p}) = \begin{cases} -d(\boldsymbol{p},\Omega) & \text{if } \boldsymbol{p} \in \Omega, \\ +d(\boldsymbol{p},\Omega) & \text{if } \boldsymbol{p} \in \mathbb{R}^n \backslash \Omega, \end{cases} \tag{1}$$

where $d(p, \Omega) := \inf_{y\in\Omega} ||p - y||_2$ is a scalar function that returns the smallest Euclidean distance from a sample point $p \in \mathbb{R}^n$ to the boundary $\partial\Omega$.

SDFs provide a simple yet efficient way of determining the location of a set of points relative to a domain Ω implicitly. The SDF is a scalar function that encodes the Euclidean distance of a sample point $p \in \mathbb{R}^n$ to the boundary $\partial\Omega$ of the domain. By evaluating the sign of the SDF, it is possible to classify the set of points as being within or outside the boundary. This enables set operations such as union, difference, and intersection to be performed.

In the signed distance package, these operations are implemented using MATLAB's arithmetic operators between two or more instances of the `Sdf` class, including '+' (union), '−' (difference), '/' (intersection), $*$ (scaling), and '.$*$' (repeating). By utilizing these set operations and a library of basic SDF primitives, it is possible to construct a wide range of complex geometries with relative ease. Subsequently, the SDFs can be transformed into a `.stl` file using the Marching Cube algorithm [98], enabling 3D printing. This functionality is implemented in the command `Sdf.export`.

> Example 1: Implicit CAD using SDFs

To demonstrate the use of signed distance functions in `Sorotoki`, we present an example of 2D and 3D implicit modeling scheme as shown in Figure 4. This example illustrates the utilization of various SDF primitives, which are combined through standard set operations, such as union, difference, and intersection, to generate complex geometries. The accompanying code is provided below:

```
%% EXAMPLE: Sdf class
% generate 2D sdf
c    = sCircle(1);
r    = sRectangle(1);
sdf  = r.rotate(pi/4)-c;

% generate 3D sdf
S1   = sSphere(0.5, [0, 0, 1]);
S2   = sSphere([0, 0, 0.5]);
C    = sCube([0 1]);
sdf  = (C - S1)/S2;
```

2) SDF differentiablility

Contrary to mesh-based geometries, signed distance functions (SDFs) possess closed-form differentials. Specifically, if Ω is a subset of $\mathbb{R}^n$ with piecewise smooth boundaries, the SDF is (*i*) differentiable almost everywhere, and (*ii*) its gradient satisfies $|\nabla \texttt{sdf}| = 1$. As a result, the unit-normal vector $n(p)$ pointing away from the boundary $\partial\Omega$ can be expressed as $n(p) := \nabla\texttt{sdf}(p)$. The gradient can be estimated using a finite-difference scheme:

$$n_i(p) \approx \frac{1}{\varepsilon}\Big[\texttt{sdf}(p + \varepsilon\delta_i) - \texttt{sdf}(p)\Big], \tag{2}$$

where δ_i is a vectorized Kronecker delta and ε a small step.

Such finite difference routine is efficiently implemented such that the normal, tangent, and bi-normal

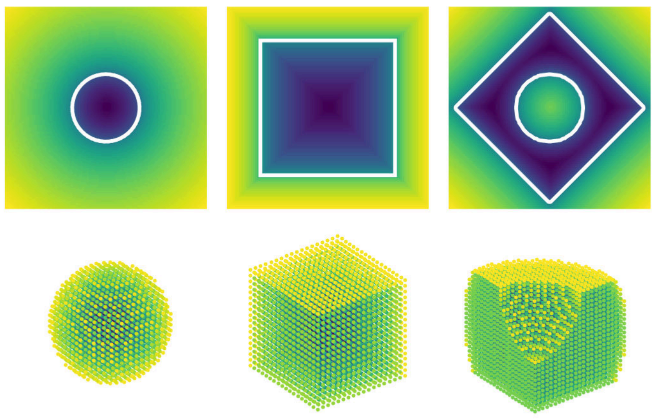
FIGURE 4. Exemplary functionality of the Signed Distance Function (Sdf) operators in Sorotoki. The top figures are two-dimensional Sdfs, whereas below are three-dimensional Sdfs. Sorotoki allows the user to combine Sdf using Matlab's arithmetics, like '+', '–', and '/', to perform unions, differences, and intersections, respectively. These set operations like union, difference, and intersect lead to new (differentiable) SDFs.

vector computations can be called using `[N,T,B] = Sdf.gradient(p)`. These gradient vector computations are crucial for contact dynamics with the environment whose topology may be arbitrarily complex. The normal vector can also be useful in finding the closest-point projection onto the surface $\partial\Omega$, namely $\text{proj}_{\partial\Omega}(p) := p - \texttt{sdf}(p) \cdot \nabla\texttt{sdf}(p)$. The projection operator is implemented as `[P,d] = Sdf.project(p)`, which takes a point cloud `p` and returns a point cloud `P` that is mapped onto the boundary of the SDF. It also returns the Euclidean distance $d(\mathbf{p}, \partial\Omega)$ from the surface. This can be extremely useful in simulations of soft robotic grippers for grasping, or obstacle avoidance for soft manipulators.

B. MESH GENERATION FOR FINITE ELEMENT ANALYSIS

In finite elements and computer graphics, mesh tessellation is a common language used to describe the structural geometry through a finite collection of vertices and edges. In `Sorotoki`, meshes and mesh generation features are packaged into the class `Mesh.m`. In general, a mesh defines a discrete representation of a continuum body that is subdivided into smaller convex sub-volumes, referred to as "*elements*". The nodal and elemental information are stored in data structures that can be accessed using `msh.Node` and `msh.Element`, respectively. For two-dimensional FEM problems, it is common to use linear elements such as `Tri3` and `Quad4` or quadratic elements like `Tri6` and `Quad8`. For three-dimensional FEM problems, the common practice is to use hexahedron elements (i.e., `Hex8`) or tetrahedral elements (i.e., `Tet4` and `Tet12`). There are also polygonal tessellations, often denotes as `PolyN` finite elements [99]. `Sorotoki` supports all these types.

1) Mesh generation from SDFs

The `Sorotoki` toolkit explores several routines for mesh generation, which are all contained in the class `Mesh.m`.

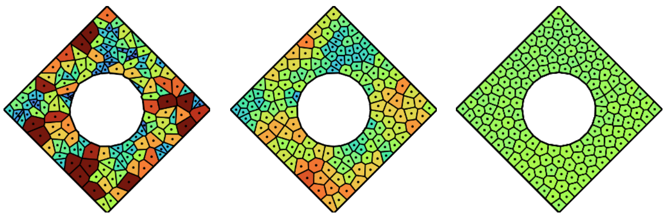

FIGURE 5. Example of mesh generation in Sorotoki. The figure shows the evolution of an unstructured polygonal mesh based on Lloyd's algorithm. The colors relate to the relative element size with respect to the mean element size, given by ▬ ∈ [0,2] mm^2. Notice that only after a few iterations, the centers of the Voronoi cells become homogeneously distributed within the domain Ω.

Our primary focus is on using a modified version of the `PolyMesher` software developed by Talischi et al. [99]. Their work provided a stable foundation for generating unstructured meshes of `PolyN` elements. The approach starts by defining a material domain implicitly using SDFs (as discussed in Section V-A). The number of elements is chosen a priori, and then repeated random sampling of Equation (1) is performed until the number of samples that fall within the specified domain matches the number of elements. A bounded Voronoi diagram is generated using the samples and the centers of the Voronoi cells are updated using Lloyd's algorithm [100]. To generate a mesh from an `Sdf` class, one can call `msh = Mesh(Sdf)` followed by `msh = msh.generate`.

> Example: Meshing of SDFs

We provided a mesh generation example in Figure 5 where we used the SDF function from the previous example to generate our tessellation. The code is given below. In Figure 5, we see the evolution of the Voronoi cells that produce the `PolyN`-type mesh. Observe that after a few iterations of Lloyd's algorithm, the centroids are distributed homogeneously over the compact domain Ω (as shown by the color distribution).

```
%% EXAMPLE: Mesh class
%  generate sdf mesh domain
c   = sCircle(1);
r   = sRectangle(1);
sdf = r.rotate(pi/4)-c;

% sdf conversion to mesh
msh = Mesh(sdf,'NElem',150);
msh = msh.generate();
```

2) Mesh from common file formats

An alternative option is to use the mesh generation tools provided by the `Partial Differential Toolbox` in Matlab. Such function is also included in `Mesh.generate`. SDFs can also be used in this process, although an intermediate step is required. For two-dimensional domains, SDFs are first converted into binary images and then the image boundary detection is used to convert them to either a linear mesh (`Tri3`) or a quadratic mesh (`Tri6`). Direct input of black-and-white `.jpg` or `.png` images is also supported. For three-dimensional domains, SDF functions are converted to an `.stl` file using the Marching Cube algorithm [98] and then provided to the Matlab PDE toolbox to generate the tessellation. Importing and exporting `.stl` or `.obj` files directly is also possible.

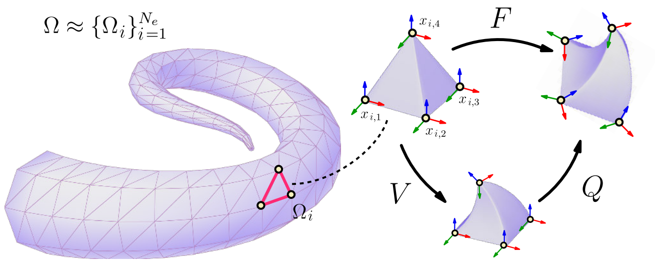

FIGURE 6. Illustration of the Finite Element Method (FEM), where a solid geometry Ω is subdivided into N_e finite elements Ω_e. Each element has dim(x_e) DOFs, which allows the computation of the deformation gradient F. The deformation gradient can be decomposed into $F = QV$, an isochoric deformation part V and rigid-body rotation part F.

C. FINITE ELEMENT MODELING

Following the mesh generation process, `Sorotoki` offers a nonlinear finite element solver for both quasi-static and fully dynamic simulations. An illustration of the FEM approach is given in Figure 6. FEM-based tools are crucial when describing large deformations in soft robots, which also accounts for hyperelastic materials and geometric nonlinearities. The FEM package is provided in a class called `Fem.m` and can be instantiated using `fem = Fem(Mesh)`. This class serves two main purposes: (*i*) to solve static or dynamic continuum problems with high accuracy, and (*ii*) to solve gradient-based optimization problems, also known as inverse design problems. It is important to note that, unlike `SOFA`, the focus of `Sorotoki` is on high-detail simulations rather than real-time implementation for control. The presented FEM simulation models are not intended for real-time applications, but rather for system identification and analysis.

1) High-detail finite element model

The nonlinear dynamics of the finite element model in `Sorotoki`, similar to `SOFA` and `Gibbon`, can be described by the general Newton-Euler equation of motion:

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f}_{\text{mat}}(\boldsymbol{x},\dot{\boldsymbol{x}}) + \boldsymbol{f}_{\text{g}} = \boldsymbol{f}_{\text{u}}(\boldsymbol{x},\boldsymbol{u},t) + \boldsymbol{f}_{\Omega_{\text{env}}}(\boldsymbol{x},\dot{\boldsymbol{x}},t), \tag{3}$$

where $\boldsymbol{x}, \dot{\boldsymbol{x}}$, and $\ddot{\boldsymbol{x}}$ are the global nodal displacement, velocities and accelerations of the mesh tesselation, respectively; $\boldsymbol{M}$ the constant generalized mass matrix, $\boldsymbol{f}_{\text{mat}}$ the internal soft material forces, $\boldsymbol{f}_{\text{g}}$ the constant gravitational forces, $\boldsymbol{f}_u$ a user-defined input, and $\boldsymbol{f}_{\Omega_{\text{env}}}$ the normal reaction forces and tangent friction forces imposed by the dynamic contact with a (possibly time-dependent) environment Ω_{env}. The environment Ω_{env} can be described using the SDF functionality (see Section V-A) using the syntax `fem.addContact(sdf)`. A broad collection of generalized external inputs can be added using: `fem.addLoad`, `fem.addDisplace`, `fem.addGravity`, and `fem.addTendon`. Alternatively,

TABLE 2. Table of deformations measures relevant for continuum mechanics problem. All measures can be related to the first-order deformation tensor *F*, following the works [52], [53], [101].

Deformation measure	Derivation
Relative volume change	$J = \det(\boldsymbol{F})$
Polar decomposition	$\boldsymbol{F} = \boldsymbol{Q}\boldsymbol{V}$
Right Cauchy-Green tensor	$\boldsymbol{C} = \boldsymbol{F}^\top \boldsymbol{F}$
First strain invariant	$I_1 = \mathrm{trace}(\boldsymbol{C})$,
Second strain invariant	$I_2 = \frac{1}{2}\left[\mathrm{trace}(\boldsymbol{C})^2 - \mathrm{trace}(\boldsymbol{C}^2)\right]$
First strain invariant	$I_3 = \det(\boldsymbol{C})$

time-varying pressure inputs can be added using the command `fem.addPressure`.

Without loss of generality, the material force can be decomposed into a position-dependent and velocity-dependent part: $\boldsymbol{f}_{\mathrm{mat}}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{f}_{\mathrm{e}}(\boldsymbol{x}) + \boldsymbol{f}_{\mathrm{d}}(\dot{\boldsymbol{x}})$, i.e., an elastic and dissipation contribution, respectively. We assume that the dissipation is given by $\boldsymbol{f}_{\mathrm{d}} = \mathbf{R}\dot{\boldsymbol{x}} = \zeta \mathbf{M}\dot{\boldsymbol{x}}$ with damping coefficient $\zeta > 0$. Materials can be assigned using `fem.addMaterial`. Note that the conservative elastic material forces $\boldsymbol{f}_{\mathrm{e}}$ require more involved computation. Since this computation is not straightforward, we briefly explain the derivation of the nonlinear hyper-elastic material forces in (3), which follows standard nonlinear finite element procedures [52], [53], [101].

Intermezzo: Deformation gradient

A fundamental measure of deformation in continuum mechanics is the deformation gradient, denoted by $\boldsymbol{F}$. The deformation gradient characterizes the local deformation for a neighborhood of the continuum body Ω. Since a subvolume of the continuum body cannot be reduced to a point, it follows that $\det(\boldsymbol{F}) = J > 0$ and $\boldsymbol{F}^{-1}$ exists. The term J is called the relative volume change and it is equal to 1 for isochoric deformations, such as rigid body deformations. Given these properties, the deformation gradient can then be factorized into $\boldsymbol{F} = \boldsymbol{Q}\boldsymbol{V}$, where $\boldsymbol{V} \succ 0$ is the right-handed stretch tensor and $\boldsymbol{Q} \in \mathrm{SO}(3)$ is a rotation matrix belonging to the special orthogonal group [52], [53], [101]. For convenience, we summarize the derived quantities of $\boldsymbol{F}$ in Table 2 that will be used throughout this section.

Intermezzo: Derivation of hyperelastic forces

Let Ω_i denote the subspace spanned by the i-th element of the finite element mesh, and let $\boldsymbol{x}_i$ denote its nodal displacement vector. The elasticity of the constitutive soft material can be described by a strain-energy density function $\Psi : \boldsymbol{F} \to \mathbb{R}_{\geq 0}$. A comprehensive discussion on common constitutive models for Ψ will be provided later in the subsequent paragraph. The elastic potential energy of the continuum body is given by $\mathcal{U}_{\mathrm{e}} = \int_\Omega \Psi(\cdot)\, dV$, and the conservative hyper-elastic force contribution can be computed as $\boldsymbol{f}_{\mathrm{e}} := \nabla_{\boldsymbol{x}} \mathcal{U}_{\mathrm{e}}$. This contribution can be approximated using piecewise finite element interpolation and integrated using the Gauss quadrature rule [52] as follows:

$$\begin{aligned} \boldsymbol{f}_{\mathrm{e}}(\boldsymbol{x}) &= \sum_{i=1}^{N_e} \frac{d}{d\boldsymbol{x}_i} \left\{ \int_{\Omega_i} \Psi(\boldsymbol{F}(\boldsymbol{x}_i, s))\, ds \right\}, \\ &\approx \sum_{i=1}^{N_e} \sum_{j=1}^{N_w} w_j \underbrace{\frac{\partial \Psi}{\partial \boldsymbol{F}}(\boldsymbol{F}(\boldsymbol{x}_i, s_j))}_{\text{PK1}} \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_i}(\boldsymbol{x}_i, s_j), \end{aligned} \tag{4}$$

where the Gauss weights are denoted by $w_j > 0$, and the number of finite elements and Gauss samples are represented by N_e and N_w, respectively.

The term $\partial\Psi/\partial\boldsymbol{F}$ is also referred to as the first Piolla-Kirchhoff (PK1) stress tensor, which can be represented in closed-form for many constitutive models. The term $\partial \boldsymbol{F}/\partial \boldsymbol{x}_e$ denotes the deformation Jacobian, which can also be given in closed-form but depends on the choice of element type. It should be noted that these tensor calculations are highly nonlinear, making their computation the most time-consuming aspect of the finite element assembly. To enhance computational efficiency, the toolkit employs `.mex` executable code, generated during installation (`Matlab Coder` is required).

Intermezzo: Newmark-beta solver

The Newmark-β method is an implicit numerical integration scheme extensively used to solve high-dimensional structural dynamic problem [53], [104]. We briefly explain the algorithm implemented in the function `fem.simulate`. First, let us subdivide the time domain such that $(0, \ldots, T)$ with uniform timesteps $\Delta t = t_{i+1} - t_i$. Then, given the initial conditions for (3), we wish to compute the state evolution $\boldsymbol{x}(t_i)$ and $\dot{\boldsymbol{x}}(t_i)$. For conciseness, let us write the discrete states of the FEM model as $\boldsymbol{x}(t_i) = \boldsymbol{x}^{(i)}$. Through the extended mean value theorem, we can formulate the general Newmark-β scheme as

$$\dot{x}^{(i+1)} = \dot{x}^{(i)} + \Delta t \left[(1-\beta_1)\ddot{x}^{(i)} + \beta_1 \ddot{x}^{(i+1)} \right], \tag{5}$$

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \Delta t^2 \left[\frac{\dot{x}^{(i)}}{\Delta t} + (\tfrac{1}{2} - \beta_2)\ddot{x}^{(i)} + \beta_2 \ddot{x}^{(i+1)} \right], \tag{6}$$

where $\beta_1, \beta_2 \geq \frac{1}{2}$. Now, in the expressions above only the forward-time acceleration $\ddot{x}^{(i+1)}$ is the unknown partial solution, hence we conveniently write $\boldsymbol{w} := \ddot{\boldsymbol{x}}^{(i+1)}$. Substitution into the flow (3), we find:

$$\boldsymbol{r}(\boldsymbol{w}) := \boldsymbol{M}\boldsymbol{w} + \nabla_{\boldsymbol{x}} \mathcal{H}(\boldsymbol{w}) + \boldsymbol{f}_{\mathrm{d}}(\boldsymbol{w}) - \boldsymbol{f}_{\mathrm{u}}^{(i+1)} - \boldsymbol{f}_{\Omega_{\mathrm{env}}}^{(i+1)}, \tag{7}$$

where $\mathcal{H}$ is the Hamiltonian. Following, the residual vector (7) forms an optimization problem in the form $\mathrm{argmin}_{\boldsymbol{w}} \|\boldsymbol{r}(\boldsymbol{w})\|_2$ for unknown accelerations $\boldsymbol{w}$. This implicit relation can be solved numerically using the Newton Raphson method. Given the n-th iteration, the recursive solver reads

$$\boldsymbol{w}^{(n+1)} = \boldsymbol{w}^{(n)} + \alpha_+ \left[\boldsymbol{A}(\boldsymbol{w}^{(n)}) \right]^{-1} \boldsymbol{r}(\boldsymbol{w}^{(n)}), \tag{8}$$

where $\boldsymbol{A} := \left[\boldsymbol{M} + \beta_1 \Delta t \boldsymbol{R} + \beta_2 \Delta t^2 \boldsymbol{K}_T \right]$ is the hessian matrix, and α_+ a positive update coefficient. The matrix $\boldsymbol{K}_T$ denotes

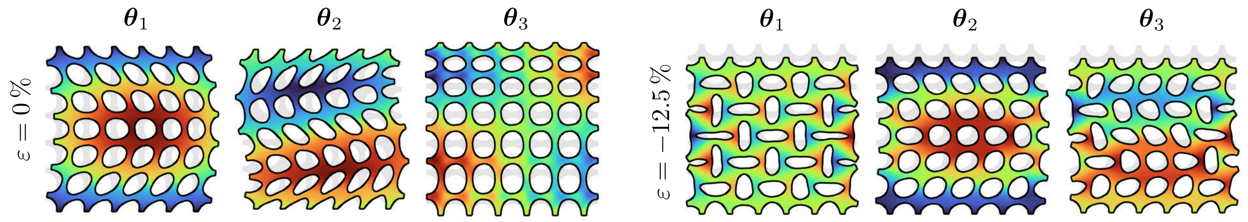

FIGURE 7. Nonlinear buckling mode analysis of periodic circular porous elastic structure inspired by [102], [103]. The horizontal displacements are indicated by 🟥 $\in [-5, 5]$ mm. (top) The first four eigenmodes of the elastomer structure for $\varepsilon = 0\%$ compression, no buckling modes appear. (bottom) The first four eigenmodes for $\varepsilon = 12.5\%$ compression. Notice that the first mode θ_1 is a buckling mode where the collapses holes orient periodically either vertically or horizontally.

TABLE 3. Table of material models included in *Sorotoki*, including the NeoHookean (NH), Mooney-Rivlin (MR), and Yeoh (YH) model.

Material model	Parameters	Energy-density potential Ψ
Neo-Hookean (NH)	(μ)	$\Psi_{\text{NH}} := \frac{\mu}{2}\,(I_1 - 3)$
Mooney-Rivlin (MR)	(c_1, c_2)	$\Psi_{\text{MR}} := \sum_{i=1}^{2} c_i\,(I_i - 3)$
Yeoh (YH)	(c_1, c_2, c_3)	$\Psi_{\text{YH}} := \sum_{i=1}^{3} c_i\,(I_1 - 3)^i$

the tangent stiffness related to the local gradient of the conservative force, given by $\boldsymbol{K}_T := \nabla_{\boldsymbol{x}}(\boldsymbol{f}_{\text{e}} - \boldsymbol{f}_{\text{u}} - \boldsymbol{f}_{\Omega_{\text{env}}})$. The number of steps can significantly deceased by choosing α_+ adaptively. We follow the adaptive gradient descent approach proposed by Malitsky et al. [105].

2) Hyperelastic models and soft material presets

An important aspect of soft robotics in general is to accurately describe large nonlinear deformations of inertial continuum bodies under motion. Yet, due to these large deformations, many classical Hookean elasticity models may not be accurate for elastomer materials.

To address this, `Sorotoki` provides a library of hyperelastic constitutive material models: Neo-Hookean (NH), Mooney-Rivlin (MR), and Yeoh model (YH). The strain energy densities for these models are derived based on the strain invariants I_1, I_2, and I_3 provided in Table 2 and are shown in Table 3. The material models presented in Table 3 are implemented in `Sorotoki` under the class `Material`, but have specific constructors tailored towards each material, `NeoHookean`, `Mooney`, and `Yeoh`. Regarding their parameters, the work of Marechal et al. [106] provides an open-source database that includes a broad collection of soft materials commonly used in soft robotics, gathered through uniaxial material tests. Based on their dataset and relevant other literature [18], [52], [54], [101], `Sorotoki` offers some preset material models of soft materials commonly used in soft robotics, such as the Ecoflex30/50 series, Dragonskin10/30 series, NinjaFlex, and Formlabs Elastic50A/80A material. These material classes also include the physical data for density, viscosity, and tangential contact friction. Following (4), the first Piola-Kirchhoff (PK1) stress tensor can be evaluated by `P = Material.PiollaStress(F)`.

3) Finite element solvers and (nonlinear) modal analysis

To solve the structural forward dynamics of the system (3), the toolkit uses an implicit Newmark-β solver [104], which is briefly outlined earlier. Implicit solvers offer improved stability compared to explicit methods, such as the Runge-Kutta solver (`ode45`), particularly when larger time steps are employed. However, the cost of larger time steps is a decreased numerical precision. Alternatively, for quasi-static problems when $\ddot{\boldsymbol{x}} = \dot{\boldsymbol{x}} = 0_n$, we aim to seek the solutions to the static force equilibrium $\boldsymbol{f}_{\text{res}}(\boldsymbol{x}) = 0_n$ where $\boldsymbol{f}_{\text{res}} := -\boldsymbol{f}_{\text{mat}} + \boldsymbol{f}_{\text{g}} + \boldsymbol{f}_{\text{u}} + \boldsymbol{f}_{\Omega_{\text{env}}}$ is the force residual vector. The nonlinear equality for nodal displacements $\boldsymbol{x}$ is solved using a reduced implementation of the Newmark solver. To call these solvers, dynamic simulations are executed with `fem.simulate` and quasi-static simulations with `fem.solve`. Upon completion of a simulation, all displacements, velocities, forces, and stress information are stored in the `fem.solver.sol` data structure. This log file can be accessed for data analysis or during simulation to facilitate state feedback control.

Alternatively, we can explore nonlinear modal analysis at any quasi-static equilibrium configuration $\boldsymbol{x}^* \in \mathcal{X}$ of the system (3). Let $\boldsymbol{K}_{T,\text{e}} := \left[\frac{\partial \boldsymbol{f}_{\text{e}}}{\partial x_1} \; \ldots \; \frac{\partial \boldsymbol{f}_{\text{e}}}{\partial x_2}\right]$ be the Jacobian matrix of the (nonlinear) elastic potential forces, also referred to as the tangent stiffness. Then, the local eigenvalue problem for the linearized FEM model around the point $\boldsymbol{x}^*$ is given by

$$\Big[\boldsymbol{K}_{T,\text{e}}(\boldsymbol{x}^*) - \lambda_i \boldsymbol{M}\Big]\boldsymbol{\theta}_i = 0_n, \tag{9}$$

where λ_i is a real scalar eigenvalue and $\boldsymbol{\theta}_i$ is its corresponding eigenmode. The dynamic analysis is implemented in Sorotoki using `fem = fem.eigen(x)`, which stores the necessary data in `fem.Log`. It is important to note that, unlike linear analysis, the eigenmode set $\{\boldsymbol{\theta}_i\}$ obtained from the eigenvalue decomposition in (9) is highly dependent on the linearization point $\boldsymbol{x}^*$ and may thus not be unique for all $\boldsymbol{x}^* \in \mathcal{X}$.

> Example: Nonlinear buckling analysis via decomposition

An excellent case study of the eigenvalue problem in nonlinear elasticity systems is the buckling behavior of patterned elastomer metamaterials, as studied by Bertoldi et al. [102] and later by Overvelde et al. [103]. In their studies, an elastomer specimen with a periodic circular porous structure was subjected to uniaxial compression. The specimen displayed an inward buckling phenomenon at a critical loading point, resulting in the specimen exhibiting a negative Poisson ratio,

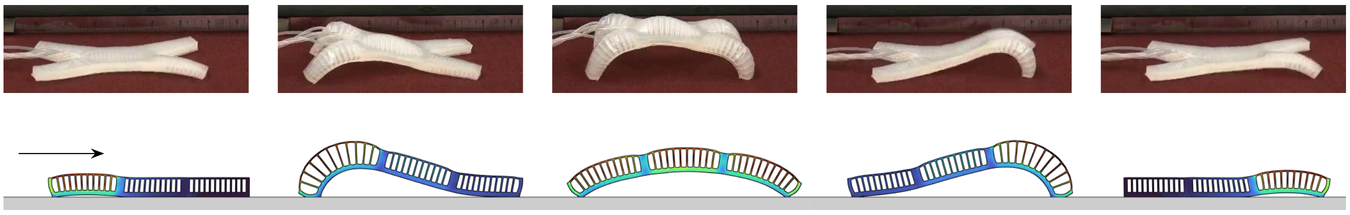

FIGURE 8. (top) Experiment of soft multi-gait crawling soft robot developed by Shepard et al [107] performing an undulating forward motion by periodic pressurization of its internal pressure chambers, back legs $\rightarrow$ middle $\rightarrow$ front legs. The soft robot is made from an elastomer material with strain-inhibit layer at the bottom to enhance bending. The Von Mises stresses are shown as ▬ $\in [0, 10]$ MPa. (bottom) Simulation recreation of the experiments performed by Shepard et al [107] using the Finite Element solver in the Sorotoki toolkit. Images used from a public video based on the work from Shepard et al. [107] with copyright permission under CC-BY license.

i.e., auxetic behavior. To be specific, the structure undergoes a so-called "*bifurcation*" where solutions switch stability or new solutions arise for a critical parameter value. In this case, the bifurcation parameter is the compression ratio ε.

In accordance with [103], a square elastomer specimen with circular holes was modeled using a Neo-Hookean material model, with Young's modulus $E = 19$ (kPa) and Poisson ratio $\nu = 0.45$. As reported in [103], the critical buckling point was observed to occur at approximately $\varepsilon = -12.5\%$ uniaxial compression.

A numerical solution is obtained through quasi-static analysis using the function `fem.solve`. To model compression, a displacement load was added using `fem.addDisplace('top',...)`. The resulting equilibrium configuration was then utilized in the eigenvalue problem via the function `fem.eigen(x)`. The eigenmodes for the zero-stress and $\varepsilon = -12.5\%$ compression cases are illustrated in Figure 7. It is worth noting that the first three eigenmodes of the elastomer structure at $\varepsilon = 0\%$ compression exhibit no buckling modes. Conversely, the first eigenmode θ_1 at $\varepsilon = -12.5\%$ compression displays a buckling mode, wherein the collapse of the holes is periodically oriented either vertically or horizontally. This buckling mode is in accordance with the experiments from [103] and [102]. The supplementary code is provided below:

```
%% EXAMPLE: Fem class
msh = preset.mesh.porous_square;
fem = Fem(msh);

% get compression value
dX  = -0.125 * msh.BdBox(2);

% assign material
fem.Material = NeoHookean(1.0, 0.45);

% add displacement and forces
fem = fem.addSupport('bottom', [1, 1]);
fem = fem.addSupport('top', [1, 0]);
fem = fem.addDisplace('top', [0, dX]);

%  quasi-solve and eigen-analysis
fem = fem.solve('TimeStep', 1/60);
fem = fem.eigen(fem.solver.sol.x);
```

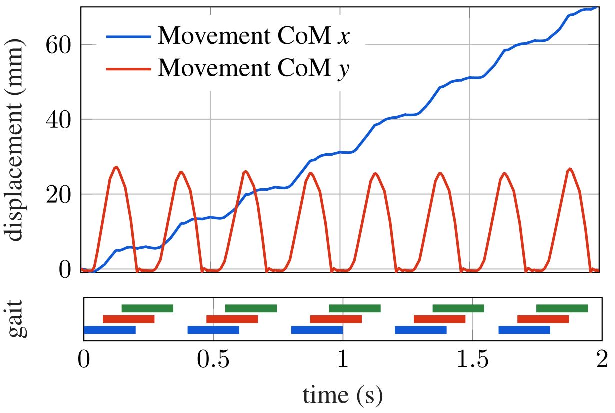

FIGURE 9. (top) Numerical simulation of the center of mass displacement and gait for undulating soft crawler. The horizontal displacement is given by (🟦), and the vertical as (🟥). (bottom) The gait cycle of the soft crawler where the sequence {🟦, 🟥, 🟩} shows the fluidic activation.

> Example: Locomotion dynamics of undulating soft robot

To demonstrate a dynamic finite element method (FEM) simulation that incorporates contact, we will utilize `Sorotoki` to model the locomotion of a multi-gait soft robot crawler inspired by the work of Shepard [107]. The study by Shepard et al. [107] presents a soft robot system that consists of five pressure chambers - four for each leg and one for the spine. The pressure chambers are actuated in a sequential manner to produce an undulating motion. The work of Shepard et al. [107] demonstrates that complex locomotion can be achieved through the use of open-loop controllers and the dynamic interaction between the soft robot and its environment.

To simplify the model, we assume general plane motion. The geometry of the soft crawler's cross-section is first provided to `Mesh.m` to generate a triangular mesh. Then, a finite element method (FEM) model is generated, with the material model `fem.Material = Ecoflex0030`. To model the environment, the function `fem = fem.addContact(sLine)` is utilized, which simply creates an unbounded horizontal line. In accordance with [107], a harmonic excitation is applied to each chamber, as expressed by the following: $u_i = A \cdot \mathrm{sat}\left[\sin(\omega t - \phi_i)\right]$, where the index $i \in \{1, 2, 3\}$ represents the front, middle, and back pressure chambers embedded in the soft body.

The excitation signal parameters are set as follows: $A = 45$ (kPa), $\omega = 5\pi$ (rad), and $\phi_i = \frac{\pi}{3} \cdot (i-1)$ (rad). The saturation function is defined as $\text{sat}(x) = 0$ for $x < 0$, and $\text{sat}(x) = x$ for $x \geq 0$. Gravitational acceleration is added, and the dynamic simulation solver is invoked using `fem.simulate`. Figure 8 presents a comparison between the soft robot described in [107] and the dynamic simulation performed by `Sorotoki`.

```
%% EXAMPLE: Fem class
msh = preset.mesh.multigait_crawler;

% mesh to fem
fem = Fem(msh);
fem = fem.addMaterial( Ecoflex50 );

% add inextensible bottom layer
fem = fem.addMaterial( Dragonskin30 );
E1  = fem.findElements([0, 150, 0, 2]);
fem = fem.setMaterial(E1,2);

% add forces and constraints
fem = fem.addGravity();
fem = fem.addContact(sLine());

type = 'BoxSelect';
for i = 1:3
    box = [50*(i-1), 50*i, 0, 15];
    Ie  = fem.findEdges(type, box);
    fem = fem.addPressure(Ie, ...
      @(t) 45*kpa * activation(t,i));
end

% solve dynamics
fem = fem.simulate('TimeStep',1/500);

%% -- open-loop controller
function y = activation(t, i)
    phi = pi / 3 * (i - 1);
    y = uclamp(sin(5*pi*t - phi));
end
```

The results of the simulation performed using `Sorotoki` show a morphological behavior that is consistent with the experimental recordings. Figure 9 depicts the trajectory of the center of mass (CoM) of the soft robot during the undulating locomotion. Note that an identical stair-like evolution of the CoM is also observed in the work of Shepard et al. [107].

4) Gradient-based computational (inverse) design

Besides modeling, the field of computational inverse design can also benefit from the use of FEM models. Building up the `Fem` class, the objective is to find a topological structure of a continuum system based on a desired deformations or compliance. One widely adopted method is the Solid Isotropic Material with Penalization (SIMP) approach, which is a commonly used material interpolation technique in topology optimization [108]. In the SIMP method, each finite element $e \in \{1, 2, ..., n_e\}$ is assigned a continuous density variable $\rho_e \in (0, 1]$, which serves as an indicator of the material distribution within the mesh. If $\rho_e = 1$, the element is considered solid, while if $\rho_e = 0$, the element is considered void. This assignment of density variables enables the modification of the strain energy density in (4):

$$\tilde{\Psi}_e = [\varepsilon + (1 - \varepsilon)\rho_e{}^p]\, \Psi, \tag{10}$$

where $0 < \varepsilon \ll 1$ a lower bound on the densities, and $p > 1$ a penalty factor for penalizing intermediate densities during the optimization process. By collecting the density values $\boldsymbol{\rho} = \text{col}\{\rho_1, \rho_2, ..., \rho_{N_e}\}$, the inverse design problem can be formulated in terms of two unknowns: the displacement field $\boldsymbol{x}$ and the density field $\boldsymbol{\rho}$. Consequently, the computational design problem for general soft material structures can be expressed as a nonlinear topology optimization problem of the following form:

$$\begin{aligned} \underset{\boldsymbol{\rho}}{\text{minimize}} \quad & \Phi = -\beta_1 \boldsymbol{L}^\top \boldsymbol{x} \; + \; \beta_2 \boldsymbol{f}_\text{e}^\top \boldsymbol{f}_\text{u} \\ \text{subject to} \quad & \tilde{\boldsymbol{r}}(\boldsymbol{x}, \boldsymbol{\rho}) = 0, \\ & \boldsymbol{v}^\top \boldsymbol{\rho} \leq v^\star, \\ & \boldsymbol{\rho} \in \mathcal{P} \end{aligned} \tag{11}$$

where $\boldsymbol{L}$ a sparse unit-vector composed of nonzero entries for the degrees-of-freedom corresponding to the desired morphology of the soft robot, $\boldsymbol{v}$ the element volumes, $v*$ the desired volume infill, $\mathcal{P} = \{\boldsymbol{\rho} \in \mathbb{R}^{n_e} \mid 0 < \rho_i \leq 1\}$ admissible set for the design variables, and β_1 and β_2 are positive scalars that can be adjusted to vary the optimization problem, with $\beta_1 \ll \beta_2$ resulting in compliance minimization and $\beta_1 \gg \beta_2$ leading to a compliant mechanism. To solve the optimization problem in (11), we utilize the Method of Mixed Asymptotes (MMA) proposed by Svanberg [109], [110]. Earlier work on this computational design approach was presented in [111] and we refer to this work for the analytic gradients required for the MMA solver.

The optimization routine in the `Sorotoki` framework is incorporated into the `Fem` class and can be invoked by utilizing the command `fem.optimize('type')`, where `'type'` represents the optimization problem at hand. For minimizing compliance, the cost function is self-adjoint [108], hence objective function and constraints are linear operators. However, when dealing with compliant mechanisms, it is necessary to specify the selection vector $\boldsymbol{L}$, which can be defined using the `fem.addOutput(id)` command. The value of `id` represents the nodal indices of interest, which can be identified using the `fem.Mesh.findNode` routine.

D. REDUCED-ORDER SOFT BEAM MODELS

While the finite element method (FEM) is known for producing reliable and highly accurate results, its high-dimensional state can make it computationally slow, making direct applications for closed-loop control challenging. To address this issue, the `Sorotoki` toolkit offers reduced-order models based on Cosserat beam theory [32], [112], [113], similar to `SoroSim` [75] and `PyElastica` [76]. In Cosserat beam theory, deformable solids are modeled as elastic strings that are governed by finite strain theory. This formulation can be applied to the dynamic modeling of slender soft robots as one-dimensional spatial curves passing through the geometric center of the deformable soft body. As shown in Figure 10, a (slender) soft robot can be described using geometric

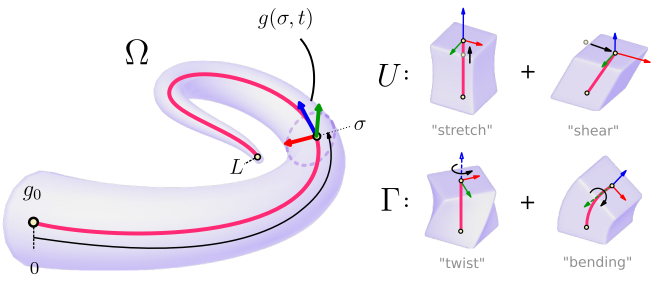

FIGURE 10. Illustration of the soft beam model using geometric Cosserat beam theory, where the backbone curve is $g \in SE(3)$ shown as (▬). The geometric strain vector $\xi := \mathrm{vec}\{\Gamma, U\}$ a vector of size 6 consisting of stretch-shear strains U and twist-bending strains Γ.

Cosserat beam models, representing it as a parameterized curve on the group of rigid-body transformations SE(3):

$$g : [0, L] \times [0, +\infty) \to \mathrm{SE}(3), \tag{12}$$

where $\mathrm{SE}(3) \cong \mathrm{SO}(3) \times \mathbb{R}^3$ composed of a (special) orthogonal rotation matrix and a translation vector.

The objective of this approach, similar to the finite element method, is to solve a dynamic system in a continuous manner, often through projecting the problem onto a finite-dimensional subspace. To address the infinite dimensionality of the curve g and make the continuum kinematics computationally tractable, various methods have been proposed, including elemental discretization [76], [114] (which is analogous to Section V-C). A widely adopted alternative is modal approximation [80], [115]. The concept of modal decomposition for describing the kinematics of continuum robots dates back to the early 1990s [115], [116], and some modal representations (e.g., first-order fourier series) even provide closed-form solutions to the inverse kinematics.

The method for constructing soft beam models in the `Sorotoki` toolkit is expressed using the syntax `shp = Shapes(pod,dof)`. In this expression, `pod` is a modal interpolation matrix that is derived from a modal basis selected by the user, and `dof` is a vector of six unsigned integers (`uint8`) that couples the beam degrees of freedom, including extension, bending, torsion, and shear, to their modal representation. The `Shapes` class serves two primary purposes: (*i*) to enable fast forward dynamic simulation of soft robots and (*ii*) to simplify the design of model-based controllers for both online and offline environments. Compared to the FEM model in (3), the soft beam models implemented in `Sorotoki` typically have a significantly lower dimensional representation, resulting in improved computational speed and, in some cases, real-time performance. This enables model-based controllers on real platforms, at the cost of model accuracy.

1) Computationally-efficient soft beam models

Following the geometric Cosserat beam frameworks [80], [81], [117], the reduced nonlinear dynamics of a soft beam model, fixed to a non-inertial base, can be represented using a Lagrangian formulation:

$$\begin{aligned}\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{f}_{\mathrm{mat}}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \\ \ldots + \boldsymbol{f}_{g}(\boldsymbol{q}) = \boldsymbol{\tau}(\boldsymbol{q},\boldsymbol{u}) + \boldsymbol{f}_{\Omega_{\mathrm{env}}}(\boldsymbol{q},\dot{\boldsymbol{q}}),\end{aligned} \tag{13}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ represent the modal coefficients, velocity, and acceleration, respectively; $\boldsymbol{M}$ denotes a state-dependent generalized inertia matrix, and $\boldsymbol{C}$ denotes the Coriolis matrix. The material forces are expressed as $\boldsymbol{f}_{\mathrm{mat}}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{K}(\boldsymbol{q})\boldsymbol{q} + \boldsymbol{R}\dot{\boldsymbol{q}}$, where $\boldsymbol{K}$ is a generalized stiffness matrix and $\boldsymbol{R}$ is a generalized damping matrix. The environmental forces are represented by the vector $\boldsymbol{f}_{\Omega_{\mathrm{env}}}$, and the generalized input is given by $\boldsymbol{\tau} = \boldsymbol{G}\boldsymbol{u}$, where $\boldsymbol{G}(\boldsymbol{q})$ is the input mapping. As in Section V-C, material and contact models can be assigned using comparable syntax, `Shapes.addMaterial` and `Shapes.addContact`, respectively. The intrinsic length of a curve can be altered by utilizing the function `Shapes.setLength`, while its cross-sectional geometry can be modified through the function `Shapes.setGeometry(sdf)`, which accepts a two-dimensional SDF function that may be arbitrarily complex.

Due to the complexity of deriving the forward kinematics and dynamics in the Cosserat model, the following subsections provide a clear summary of the finite-dimensional basis representation and its relationship to reduced kinematics. The system matrices, on the other hand, are notoriously lengthy expressions and thus omitted in this work. The reader is referred to [78] for a full derivation of model (13).

2) Finite dimensional projection

To start, our aim is to obtain a finite-dimensional approximation of the local geometric strain vector, denoted as $\xi := (\boldsymbol{g}^{-1}\frac{\partial \boldsymbol{g}}{\partial \sigma})^{\wedge} := (\Gamma^{\top}, \boldsymbol{U}^{\top})^{\top}$, where $\sigma \in [0, L]$ is a spatial coordinate and $(\cdot)^{\wedge} : \mathrm{se}(3) \to \mathbb{R}^6$ (see [35]). Here, Γ_i and U_i are the torsion-curvature and elongation-shear curve parameters, respectively. To achieve this, we employ a Ritz-Galerkin modal discretization approach following the work of Boyer et al. [80]. This approach assumes that the strain can be accurately represented through a finite series of orthonormal basis functions:

$$\begin{aligned}[\xi_i]_{\theta_i}(\sigma, \boldsymbol{q}_i) &= \sum_{j=1}^{k_i} \theta_{i,j}(\sigma) q_{i,j} + \xi_i^{\circ}(\sigma), \\ &= \underbrace{\begin{bmatrix}\theta_{i,1}(\sigma) & \ldots & \theta_{i,k_i}(\sigma)\end{bmatrix}}_{\theta_i^{\top}(\sigma)} \boldsymbol{q}_i + \xi_i^{\circ}(\sigma)\end{aligned} \tag{14}$$

where $\boldsymbol{\theta}_i$ is the modal approximation vector related to the i-th strain component, $\boldsymbol{q}_i$ is its corresponding modal coefficient, and $[\cdot]_{\theta}$ denotes the subspace projection operator. By collecting all terms $q_{i,j}$ and $\theta_{i,j}$, we compactly express the finite-dimensional approximation as an affine operation:

$$[\xi]_{\Theta}(\sigma, \boldsymbol{q}) = \Theta^{\top}(\sigma)\boldsymbol{q} + \xi^{\circ}(\sigma) \tag{15}$$

where $\Theta := \mathrm{blkdiag}\{\theta_1, \ldots, \theta_6\}$ is referred to as the *"modal approximation matrix"*, and $\boldsymbol{q} := \mathrm{vec}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_6\}$ is the generalized coordinate vector of the global soft beam model

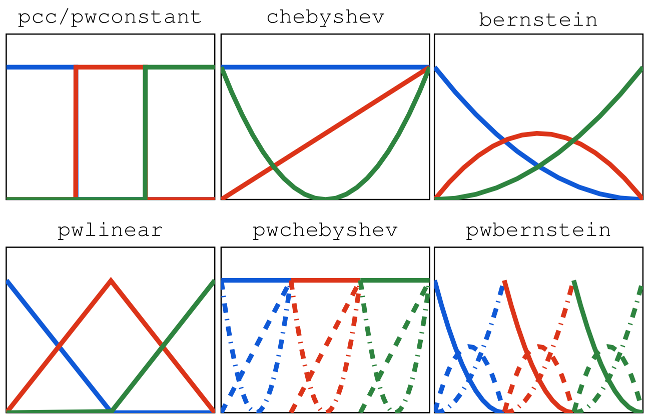

FIGURE 11. The library of modal basis functions is implemented in the Sorotoki toolkit, with a modal ordering of {🟦, 🟥, 🟩}. These function bases include Piecewise Constant Curvature (PCC), Piecewise Linear (PWL), full and piecewise Chebyshev, and full and piecewise Bernstein polynomials.

in (13). Note that a geometric strain entry may be constrained and therefore not contribute to the overall continuum dynamics, thus θ_i, q_i are fixed to zero without loss of generality. The choice of basis plays a crucial role and often relies on *ad-hoc* approaches. It is therefore critical to choose an appropriate basis for optimal performance of the soft robot model.

3) Library of modal strain bases

In `Sorotoki`, the general constructor for creating a modal basis is defined as `pod = Basis(N,M,'type')`, where `N` represents the number of samples (i.e., the level of discretization of the spatial curve), `M` is the degree of the basis, and `'type'` is an input that specifies the basis type.

The literature presents various types of modal bases, with the Piecewise Constant Curvature (PCC) approach being the most commonly used [25], [118]. The Piecewise Constant Curvature approach is suitable for certain conditions, for example, when homogeneous bending moment and homogeneous material properties are considered. However, it lacks the ability to ensure the continuity of the strain field at the boundaries between sections, resulting in jumps in the strain profile. As a result, researchers have been exploring alternative representations that more effectively preserve the continuity conditions of the deformable continuums. Examples of alternative representations of bases include piecewise linear [119], affine curvature [120], [121], Fourier cosine/sine series [115], [122], Legendre or Chebyshev [80], [117], and actuation load bases [81]. The `Sorotoki` package offers access to a library of anonymous functions, facilitating the utilization of a range of basis functions. As an illustration, a collection of basis functions is shown in Figure 11.

Remark: On the modal order

Generally, finding a suitable reduction basis and reduction order can be a challenging task. The general assumption is that if the basis belongs to a regular function space (i.e., Sobolev space) and the modal index k_i goes to infinity, the strain approximation converges (uniformly) to the exact solution on the interval $[0, L]$. However, as increasing the modal order enhances precision, it also greatly impacts computational performance. Thus, finding a balance between accuracy and computational speed is of utmost importance for the successful implementation of soft robotic models, often mandating an *ad-hoc* approach.

4) Data-driven strain basis from offline FEM simulations

To address the challenges of improving efficacy in the modal reduction of soft beam models, we propose a novel approach that merges the finite element method and soft beam modeling. This approach involves extracting geometric modal information from FEM simulation data to construct a low-dimensional strain basis, which we refer to as the Data-driven Variable Strain (DVS) basis. The DVS basis is similar in concept to the snapshot method presented by the `SOFA` toolkit [51], [54], [73], but adapted for use with one-dimensional curves on SE(3). It takes into account the underlying geometric features of the soft robot and represents them in a minimal subspace representation. This approach leads to a substantial reduction in the number of states while still maintaining high accuracy in deformations, high computational efficiency, and providing a clear structure for passive and active joints. The derivation has two steps:

- **Step 1: Recovery of geometric strain from FEM:** The reconstruction of the DVS basis begins with obtaining geometric strain data from a Finite Element Method (FEM) simulation. Such simulation supports either `Fem.simulate` or `Fem.solve`, and the resultant information is stored in the `fem.solver.sol` data structure. Mathematically, the simulation retrieves the states $\boldsymbol{x}^{(i)} := \boldsymbol{x}(t_i)$ at discrete time instances $t_i \in \{0, ..., T\}$, which in turn provides the nodal position vectors $\boldsymbol{p}$ and the deformation gradient $\boldsymbol{F}$ at any point in the mesh. Using the polar decomposition $\boldsymbol{Q} = \boldsymbol{F}\boldsymbol{V}^{-1} \in$ SO(3), see Table 2, we can retrieve the rigid body transformation of the FEM mesh

$$g_{\text{FEM}}(\boldsymbol{s}, \boldsymbol{x}^{(i)}) = \begin{pmatrix} \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{x}^{(i)}) & \boldsymbol{p}(\boldsymbol{s}, \boldsymbol{x}^{(i)}) \\ 0 & 1 \end{pmatrix}, \tag{16}$$

where $\boldsymbol{s} \subseteq \Omega$ is an arbitrary point inside the undeformed mesh. It is important to note that if $\boldsymbol{s}$ does not correspond to a nodal location of the mesh, interpolation using elemental shape functions is employed. Now, let $\bar{\gamma} : [0, L] \to \Omega$ be a unit-speed reference backbone curve that is contained within the mesh domain Ω. Then, we can retrieve $g_{\text{FEM}}(\bar{\gamma}, \boldsymbol{x}^{(i)})$. Subsequently, the geometric strain can be approximated as $\xi_{\text{FEM}} \approx (g_{\text{FEM}})^{-1}\delta g_{\text{FEM}}$. Here, δg_{FEM} represents the spatial derivative of the reference curve w.r.t. σ, which is calculated using the central difference method. It is worth noting that the choice of $\bar{\gamma}$ is free, allowing for the estimation of geometric strain for many complex structures. The full procedure is outlined in Algorithm 1.

- **Step 2: POD snapshot basis:** Next, we employ the "*Snapshot Proper Orthogonal Decomposition*" (POD) as

Algorithm 1: Recover geometric strain field ξ_{FEM} from offline dataset of soft robot FEM simulations

```
Input: Nodal displacements $x$, mesh tesselation $\mathcal{T}$,
       reference curve $\bar{\gamma}$, and sample set $\mathcal{S}$
Output: Geometric strain field $\xi_{\text{FEM}}$ at time $t_i$
1  for $i$ = each spatial sample $\sigma_i \in \mathcal{S}$ do
2      get reference position $\bar{p} \leftarrow \bar{\gamma}(\sigma_i)$ ;
3      get element $\mathcal{E} \leftarrow$ InElement($\bar{p}, \mathcal{T}$) ;
4      if $\mathcal{E} == \emptyset$ then
5          get edge $\mathcal{E} \leftarrow$ OnClosestEdge($\bar{p}, \mathcal{T}$);
6      end
7      initialize $\Phi^{(0)} \leftarrow I_3$ ;
8      initialize $\delta\gamma^{(0)} \leftarrow 0_3$ ;
9      for $j$ = each vertex spanned by element $\mathcal{E}$ do
10         get nodal displacement $X \leftarrow$ FEM($x_j$) ;
11         get deformation gradient $Y \leftarrow$ FEM($x_j$) ;
12         $[Q, V] \leftarrow$ PolarDecomposition($Y$) ;
13         $\alpha \leftarrow$ ElementInterpolation($\bar{p}$) ;
14         update $\Phi^{(j)} \leftarrow$ AverageSO3($\Phi^{(j)}, \alpha Q$) ;
15         update $\delta\gamma^{(j)} \leftarrow \delta\gamma^{(j)} + \alpha X$ ;
16     end
17     $g_{\text{FEM}}^{(i)} \leftarrow$ SE3($\Phi^{(j)}, \bar{p} + \delta\gamma^{(j)}$);
18 end
19 for $i$ = each spatial sample $\sigma_i \in \mathcal{S}$ do
20     $\delta g_{\text{FEM}}^{(i)} \leftarrow$ CentralDiff($g_{\text{FEM}}^{(i-1)}, g_{\text{FEM}}^{(i+1)}$) ;
21     assemble strain $\xi_{\text{FEM}}^{(i)} \leftarrow (g_{\text{FEM}}^{(i)})^{-1} \delta g_{\text{FEM}}^{(i)}$;
22 end
```

described in [54], [73]. This data-driven approach determines a suitable orthonormal basis from simulated or experimental data [56]. Let $y_i(\sigma, t) := \xi_{\text{FEM},i}(\sigma, t)$ represent the measurement of the i-th entry of the strain ξ_{FEM}. For each discrete time t_i, the sample is condensed into a column vector $y_i^{(t)} := \text{col}\{y_i(0,t), \dots, y_i(L,t)\}$ and then stacked into the "*snapshot matrix*" $S_i = \text{row}\left\{y_i^{(0)}, \dots, y_i^{(T)}\right\}$ where T is the finite horizon time. The correlation matrix $\mathcal{C}_i = \frac{1}{m} S_i^\top S_i$ is then computed with $m = \dim(y_i)$, and the spectral decomposition is performed:

$$\mathcal{C}_i V_i = \lambda_i V_i, \tag{17}$$

where $V_i = \text{row}\{v_{i,1}, , ... v_{i,m}\}$ is a eigenvector basis and $\lambda_i = \text{diag}\{\lambda_{i,1}, , ..., \lambda_{i,m}\}$ is a diagonal matrix of sorted eigenvalues. By selecting $k_i \leq m$ such that $\lambda_{i,k_i} \leq \delta$, where δ is a desired threshold, we obtain a truncated orthonormal basis $\{v_{i,j}\}_{j=1}^{k_i}$. This process is repeated until the modal interpolation matrix Θ, required for (15), is fully obtained. Finally, a Gram–Schmidt orthogonalization procedure is performed to ensure that its columns are mutually orthogonal.

> Example: Data-driven basis from PneuNet FEM simulation

To demonstrate the reconstruction of the DVS basis, we consider a soft bending actuator, similar to the well-known PneuNet actuator. A FEM simulation model is constructed where the soft actuator is subjected to a linearly increasing

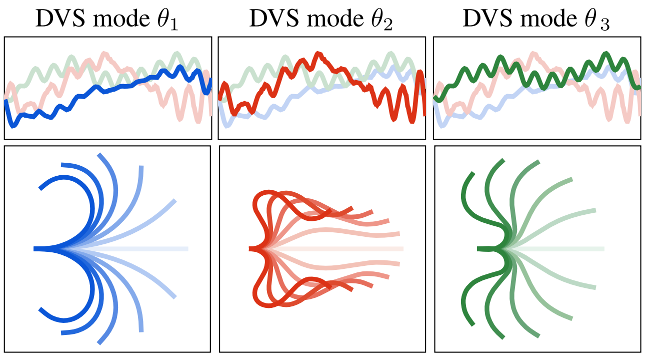

FIGURE 12. Example reconstruction of the Data-driven Variable Strain (DVS) basis for a PneuNet actuator grasping a cylindrical object. (top) The first three modes of the DVS basis related to planar bending, *i.e.*, planar curvature. The ordering is {🟦, 🟥, 🟩}. (bottom) The positional forward kinematics when regarding the bending modes {🟦, 🟥, 🟩} individually.

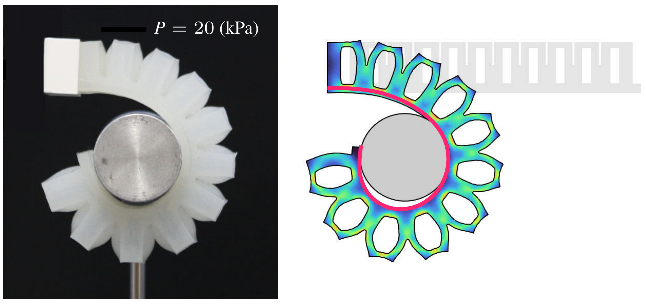

FIGURE 13. Example reconstruction of the Data-driven Variable Strain (DVS) basis for a PneuNet actuator grasping a cylindrical object. A comparison between the true physical system, the FEM model, and the soft beam model shown in (🟪). The Von Mises stresses are shown as ▨ $\in [0, 5]$ MPa.

pressure up to 40 kPa and curls around a cylindrical object when pressurized. Figure 12 (top) illustrates the true system and the FEM simulation obtained through `fem.simulate`. The FEM class is then integrated into the `Shapes` class through the syntax `shp = Shapes([],dof)`, where `dof = [0,3,0,0,0,0]` indicates the desire to recover the first three curvature bending modes from the `fem` object class. The length and base orientation are specified using `shp.setLength` and `shp.setBase`, respectively. The basis is then reconstructed by calling `shp.reconstruct`. The first three curvature bending modes are displayed in Figure 12 (bottom). It can be observed that the geometrical features of the PneuNet actuator are encoded in the basis, with the 12 embedded pressure chambers represented by the DVS strain basis. The associated code is provided below:

```
%% EXAMPLE: Shapes class (DVS basis)
fem = preset.fem.pneunet('contact',true);
fem = fem.simulate();

dof = [0,3,0,0,0,0];  % 3-DOF bending
shp = Shapes([], dof, 'length', 120);
shp = shp.setBase(SE3(eye(3), [0,0,2]));

% generate DVS basis and rebuild
shp = shp.reconstruct(fem);
shp = shp.rebuild();
```

5) Forward beam kinematics

Once a basis representation Θ has been selected, the forward kinematics of the continuum body can be efficiently solved using exponential maps for the group SE(3). As such, the backbone curve is approximated by

$$[g]_\Theta(\sigma, q) = g_0 \exp_{\mathrm{SE}(3)} \left[\Omega_k(\sigma, q)\right]. \tag{18}$$

where Ω_k is the k-th order approximation of the Magnus expansion [75], [81], [123]. We use a first order approximation of the Magnus expansion given by $\Omega_1 = \int_0^\sigma [\hat{\xi}]_\Theta(s, q)\ ds$. On the other hand, the local velocity twist is represented by $\eta := (g^{-1}\dot{g})^\vee$, which, similarly to rigid robotics, is linear in the joint velocities $\dot{q}$. Regarding computation, the velocity twist of a point σ on the curve g can be represented by:

$$[\eta]_\Theta(\sigma, q, \dot{q}) = \underbrace{\left[\mathbf{Ad}^{-1}_{[g](\sigma,q)} \int_0^\sigma \mathbf{Ad}_{[g](s,q)} \Theta(s)\ ds\right]}_{J(\sigma,q)} \dot{q} \tag{19}$$

where $J(\sigma, q)$ denotes the geometric Jacobian that maps the joint velocities $\dot{q}$ to velocity twist in the body frame. For conciseness, we write $J_\sigma(q) := \mathbf{Ad}_{[g](\sigma,q)} J(\sigma, q)$ for the global frame. The Jacobian matrix is of paramount importance, not only for inverse kinematics but also for mapping external wrenches onto the generalized joint torques. For instance, it can be used to calculate the environmental forces as $f_{\Omega_{\mathrm{env}}} = \int_0^L J_\sigma^\top F_{\mathrm{env}}\ d\sigma$, where F_{env} represents a wrench related to the environment Ω_{env} described by SDFs.

Given the expressions in (15), (18), and (19), we can numerically evaluate the forward kinematics. We use a two-step Runge-Kutta integration solver that approximates the spatial integration. The forward kinematics solver is called by `shp = Shapes.string(q,dq)`, which stores all necessary numerical evaluations into a data structure `shp.system`, or `p = Shapes.solveFK(q)` for the backbone alone.

6) Inverse beam kinematics (shape control)

The inverse kinematics problem for soft continuum manipulators involves finding a solution q such that either (*i*) the end-effector reaches a specified setpoint, or (*ii*) desired shape of the backbone is achieved. These manipulators often exhibit high levels of redundancy, so called "*hyper-redundancy*" [115]; leading to different solution approaches common to rigid robotics. Few modal basis representations possess a closed-form solution to the inverse kinematics, and they are typically solved using an iterative numerical method (e.g., Newton Raphson). In `Sorotoki`, the inverse kinematics solver for soft beam models is implemented as `Shapes.solveIK`. We briefly detail the theory.

Suppose the desired shape of the soft manipulator is $g^\star(\sigma) \in W_\sigma$ with the set of possible configurations of the backbone curve at σ given by

$$W_\sigma := \left\{X \in \mathrm{SE}(3) \mid X = [g]_\Theta(\sigma, q),\ q \in \mathcal{Q}\right\}. \tag{20}$$

Note that W_L spans the workspace of the end-effector, and $W_\Omega := \{W_\sigma \mid \sigma \in [0, L]\}$ the workspace of the entire soft body. For sake of readability, we redefine $g_i(q) := [g]_\Theta(\sigma_i, q)$ and $g_i^\star := g^\star(\sigma_i)$. We also rewrite the geometric Jacobian by $J_i(q) := \mathbf{Ad}_{[g](\sigma_i,q)} J(\sigma_i, q)$.

Then, the inverse shape kinematics problem for the discretized soft manipulator can be formulated as an optimization problem of the following form:

$$\begin{aligned} \underset{q}{\text{minimize}} \quad & \Phi = \sum_{i=1}^{N_p} \left\| K_p \log_{\mathrm{SE}(3)} \left[g_i^{-1}(q) g_i^\star\right]^\vee \right\|_2 \\ \text{subject to} \quad & g_i, g_i^\star \in \mathcal{W}_\Omega, \end{aligned} \tag{21}$$

where $\log_{\mathrm{SE}(3)}$ denotes the logarithmic mapping from the Lie group to its algebra, see [124], [125]. Despite the highly nonlinear nature of the optimization problem, its solution procedure is a relatively straightforward two-step procedure:

Given an initial guess $q^{(0)} \in \mathcal{Q}$, the aim is to compute an incremental update step that brings us closer to a local minimizer of the objective function Φ. For clarity, let $\Xi_i := g_i^{-1} g_i^\star$ represent the geometric error between the soft robot and the desired shape. The state increment can then be expressed as:

$$\lambda_i^{(k)} = J_i^\top(q^{(k)}) \left[K_p \mathrm{T}_{\mathrm{SE}(3)}(\Xi_i) \log_{\mathrm{SE}(3)}(\Xi_i)\right]^\vee, \tag{22}$$

$$q^{(k+1)} = q^{(k)} + \sum_{i=1}^{N_p} \Lambda_i \left[\lambda^{(k)} - N_i(q^{(k)}) \nabla \Psi_{\mathrm{sub}}\right], \tag{23}$$

where $\mathrm{T}_{\mathrm{SE}(3)}$ denotes the tangent operator map on the group SE(3), see [124], K_p an artificial stiffness tensor, $N_i = (I - J_i^\dagger J_i)$ represents the null-space projection, and Λ_i a diagonal activation matrix. The trivial choice is $\Lambda_i = I_n$. The null space projector can be extremely useful in exploring the high redundancy in soft robots, allowing sub-tasks Ψ_{sub} to be considered in parallel to the primary inverse kinematic problem. Classical examples of such subtasks include: minimizing elastic energy or obstacle avoidance. The iterative solver in (22) and (23) runs until convergence in $q^{(k)}$ is achieved.

> Example: Contact kinematics of ultra-gentle soft gripper

To showcase the forward and inverse kinematic solvers of `Sorotoki`, we will describe the ultra-gentle underwater soft gripper developed by Sinatra et al. [10]. The soft gripper consists of six soft fingers attached to a rigid palm base, where each soft gripper was designed to apply low contact pressure and minimize harm to common jellyfish species. An illustration of the system is shown in Figure 14. The delicate compliance of the soft gripper is achieved through the use of an extremely low durometer silicone matrix (Shore 20A). The actuator has a simple rectangular geometry, with a narrow cross-section of approximately 10×2 mm and an internal off-center rectangular hole. The thinnest part of the soft actuator, called the membrane, is approximately 0.35 mm thick, and length of about 130 mm.

In this study, we aim to reproduce a grasping scenario of a jellyfish modeled as a static SDF object. To achieve this, we first initiate a soft finger by utilizing the `Shapes` class. We employ a third-order Chebyshev basis to approximate the strain field. The geometric properties of the soft finger are

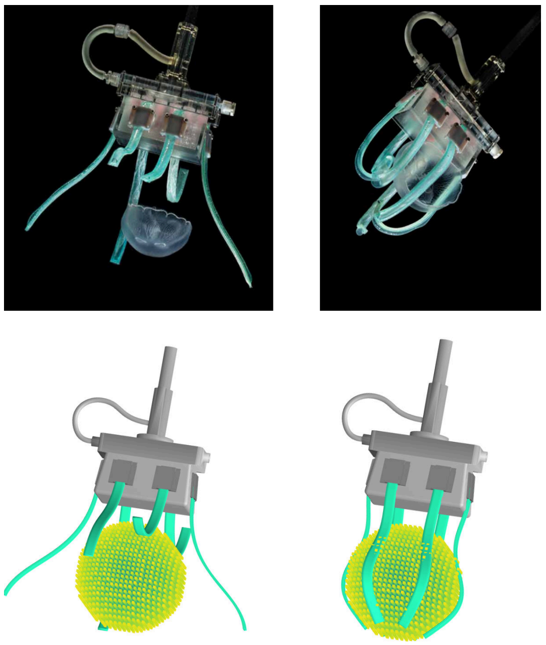

FIGURE 14. (top) A snapshot of the ultra-gentle soft robot gripper developed by Sinatra et al. [10] is shown, demonstrating its ability to grasp a delicate jellyfish. Figure is produced with copyright permission. (bottom) The reconstructed soft gripper using Sorotoki is presented, where each tentacle finger is modeled individually using the Shapes class. It can be observed that the soft tentacles envelop the SDF object, indicating a balanced solution between task and subtask. Raw image obtained from Sinatra et al. [10] , used with copyright permission with rights belonging to ©2019 Science.

specified through the functions `Shapes.setLength` and `Shapes.setGeometry`. Using a for-loop, we generate each soft finger sequentially, defining the spatial location of the fixed base with `Shapes.setBase`. Prior to deployment, each soft finger undergoes predeformation, which is calculated through the application of the forward kinematics solver `Shapes.FK(q)`. The joint configuration `q` is selected to match the experiments presented in Figure 14.

Subsequently, the deformed backbone is projected onto the surface of the SDF jellyfish through the use of the function `sdf.project`. The inverse kinematics solver is then invoked with `Shapes.solveIK`, resulting in the image depicted in Figure 14. Note that the inverse kinematics solver effectively places the soft finger onto the surface of the SDF object without causing penetration. The code for the forward and inverse kinematics is presented below:

```
%% EXAMPLE: Shapes class
% jellyfish SDF
sdf = sSphere(0,0,-60,40)

% generate Shapes class
N = 200; % discretization
M = 3;   % number of modes
pod = Basis(N,M,'chebyshev');

shp = Shapes(pod,[0,M,M,0,0,0]);

shp = shp.addContact(sdf);
shp = shp.setLength(120);
shp = shp.setGeometry(sRectangle(4,1));

% inverse kinematics on SDF topology
for k = 1:6
    shp = shp.setBase(G{k});  % set g0

    pos = shp.solveFK(q0{k}); % fw. kin.
    prj = sdf.project(pos);   % project

    qd  = shp.solveIK(prj);   % inv kin.
    shp.render(qd);           % render
end

sdf.show();  % render jellyfish
```

E. MODEL COMPOSER

In many instances, soft robots comprise multiple dynamic components that are interconnected to form the overall system. For instance, the soft robotic hand depicted in Figure 1c comprises five actively controlled soft fingers connected to a rigid palm base, each of which exhibits its own fluid-structure interaction. While each soft finger can be modeled through the `Shape` class, the class itself lacks a composer or solver that addresses the interconnections between a network of dynamic systems.

To address the issue, we propose the `Model` class, which concatenates dynamic systems to systematically increase complexity. The class is equipped with an implicit solver that facilitates communication of state information between subsystems. The goal of the `Model` class is two-fold: (i) to facilitate the composition of multiple dynamic components that form a soft robotic system, and (ii) to leverage the dynamic network structure to design controllers through interconnection of subsystems. For example, a soft robot, the fluidic actuation, and the model-based controller can be modeled as three separate entities and be composed into one global closed-loop system. This approach also enables the adaptive controllers that require additional state dynamics for online estimation of parameters. Contact interaction, on the other hand, are embedded functions inside `Fem` and `Shapes` that are called during each state increment of the forward dynamic solver.

1) Interconnected network of dynamical systems.

The class `Model.m` allows users to compose an arbitrarily large network of dynamical systems that are presented in the state-space structure $\Sigma_i : \dot{z}_i = \boldsymbol{f}_i(z_i, \boldsymbol{u}_i, t)$. Then, the network of dynamical systems can written as

$$\Sigma_{\text{net}} : \dot{z} = \boldsymbol{F}_{\text{net}}(z, \boldsymbol{u}, t) \qquad (24)$$

The network system matrices are assembled as: $\boldsymbol{F}_{\text{net}}(z) := \text{blkdiag}(\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_n)$. The implementation in `Sorotoki` is straightforward. Let `f0 = @(z,u,t) ode(z,u,t)` be an anonymous function that defines a state space model, where `z`, `u`, `t` are the state vector, the input vector, and a time variable, respectively. In contrast to standard ODE solvers in MATLAB® (e.g., `ode45.m`), the input `u` is treated as an additional input to the ODE. This simplifies the design of controllers as it enables the definition of control laws outside of the ODE function caller, as opposed to the standard ODE packages in MATLAB®.

To proceed, we first convert the function to a `sys0 = StateSpace(@f0)` class, which stores information on the state dimension, input dimension, and numerically computes the Hessian. Then, the model class is constructed using `mdl = Model(@f0)`. Other dynamic systems, represented by the `StateSpace` class `sys1`, `sys2`, and `sys3`, are added to the network by `mdl.addSystem(sys1,sys2,sys3)`. The `Model` class is also compatible with other classes, such as the `Fem` and `Shapes` class, allowing for example the interconnection between a FEM model and a soft beam model. To compute the forward dynamics, we call `mdl.simulate([0,T])` which solves the state trajectories on the finite horizon domain $[0, T]$ using an implicit Trapezoidal solver (i.e., similar to `ode23t` solver).

2) Assigning the controller

Once a network of dynamic systems has been composed in the `Model` class, assigning a control is straightforward. Controllers can be defined as auxiliary anonymous functions `@(mdl) Control(mdl)` and added using `mdl.addControl(@Control)`. At each time instance of the implicit solver, this controller function is called and prompted with the current instance of the `Model`. Once prompted, the solver retrieves the global input vector $\boldsymbol{u}$ as in (24). All system information of the network can be retrieved within the function by `mdl.getState[I,J]`, where `I` is the system index and `J` the indices of the states of subsystem `J`. Such implementation presents a highly dexterous and efficient controller design platform.

3) Fluidic reservoir with volume-variance

The majority of soft robots are actuated through fluidics. In recognition of this, `Sorotoki` offers a variant of the `StateSpace` class, called `Fluidic`, which incorporates fluid-pressure dynamics of an enclosed pressure reservoir based on the ideal gas law. The dynamics of such fluidic reservoir is determined by the compressibility of the fluid and the capacitance of the reservoir that is related to its volume $V > 0$. As a soft robot deforms, so does its internal volume. As such, consider a scalar variable $J : \mathcal{Q} \to (0, +\infty]$ such that we can describe the volume by $V(J(\boldsymbol{q})) = J(\boldsymbol{q})V_0$ where V_0 is the intrinsic volume. From a physical point of view, the scalar $J(\boldsymbol{q})$ can be seen as the relative volumetric change of the reservoir depending on the state of the soft robot. Assuming constant temperature T and polytropic coefficient n_k, the pressure dynamics can be described by

$$\dot{p} = \frac{n_k}{V(J)}\left(RT\dot{m} - \frac{dV}{dJ}(J)\dot{J}p\right) - \mu_{\text{leak}}p \tag{25}$$

where the input is $\dot{m}$ the mass flux into the reservoir, R the ideal gas constant, and μ_{leak} a pressure leakage coefficient. For a perfectly enclosed system, the parameter $\mu_{\text{leak}} = 0$. Note however, that the volumetric change $\dot{J}$ might not always be available or it is difficult to derive. As an alternative, we substitute $\dot{J}$ with an "*dirty derivative*" approximation $\dot{\tilde{J}}$ following the work of Loria et al. [126]:

$$\dot{z} = \alpha_1[z + \alpha_2 J], \tag{26}$$

$$\dot{\tilde{J}} = z + \alpha_1 J, \tag{27}$$

where $\alpha_1, \alpha_2 > 1$ are filter gains. Hence, the system of equations (25), (26), and (27) leads to the full pressure dynamics. In case of an isochoric compression (*i.e.*, no volume change $\dot{V} = 0$) with $J = 1 \Rightarrow \dot{J} = 0$, we can revert to a single-input-single-output (SISO) system with $u = \dot{m}$ (*i.e.*, the mass flow $\dot{m}$ as the system's input).

Remark: On fluidic volume changes derived from FEM

The `Sorotoki` toolkit is equipped with functionalities that provide the volume of enclosed regions of the FEM. When `fem.solve` or `fem.simulate` is called, the time-series data of these volumes can be stored, and thus can be used to identify how the state deformation $\mathbf{q}$ is related to the volume change using, e.g., methods of nonlinear regression.

For many control applications for soft robotics, fluidic reservoirs are internally pressure regulated. Hence, the mass flow $\dot{m}(p, p_d)$ depends on the internal pressure p and a (possibly time-varying) pressure trajectory p_d. The mass flow controller of a proportional pressure-regulated reservoir is modeled here as PI-type controller:

$$\dot{m} = \Psi_{\text{v}}(\nu, p), \tag{28}$$

$$\nu = k_p(p - p_d) + k_i \int p(\tau) - p_d(\tau)d\tau, \tag{29}$$

where $\Psi_{\text{v}}(\nu, p)$ i a valve flow function related to mass flow saturation of the pressure valves, and k_p, $k_i > 0$ the proportional and integral control gains, respectively. In `Sorotoki`, such fluidic system is provided as `Fluidic` class. The volume of the reservoir can be set using `Fluidic.setVolume(@V)` where `V = @(x)...` is an auxiliary anonymous function. The mass flow function can be modified by the user using a similar command `Fluidic.setMassFlow(@M)`. By default, the flow function is $\Psi_{\text{v}}(\nu, p) = \nu$, and the regulation `Fluidic.isRegulated = true`.

> Example: Simulation soft hand with fluidic soft fingers

To demonstrate the versatility of the `Model` class and the `Fluidic` class, consider the example of modeling the soft robotic hand previously shown in Figure 1. Prior to the simulation model, we conducted an experiment to allow a qualitative comparison with our modeling approach. Here, a predefined harmonic pressure signal is introduced to each of the soft fingers, with each harmonic having a $+\frac{\pi}{6}$ offset relative to its neighboring soft finger. The pressure signal is sinusoidal with an upper and lower bounds of -5 and 80 kPa,

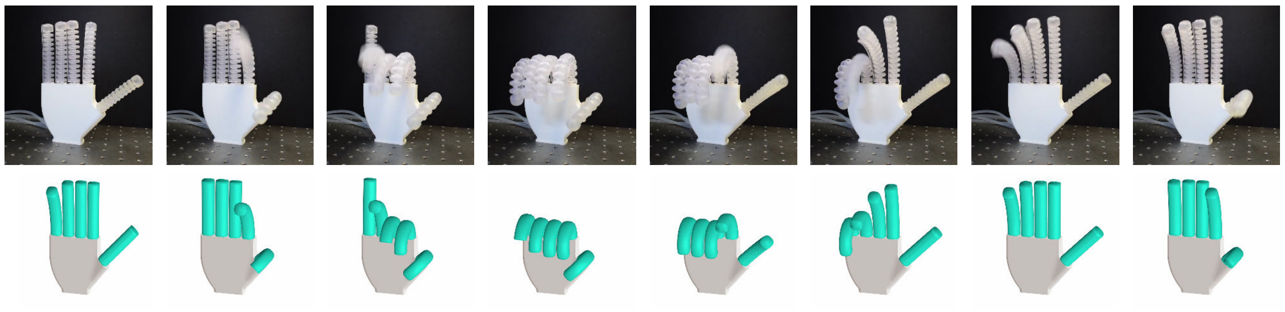

FIGURE 15. (top) Experimental snapshots of the open-source soft robotic hand provided by the Sorotoki toolkit. The five-fingered robot is subjected to a harmonic oscillator that commands oscillatory pressure to the individual fingers. Since each oscillator has a preset phase difference ($\phi = \pi/6$), the fingers of the soft robot hand undergo a periodic swinging motion. (bottom) Reconstructed soft robotic hand using Sorotoki, where each finger is modelled using the Shapes class and then composed using the Model class. Following, each oscillator is added using "mdl.addSystem" and then their ouputs are connected to the inputs of each Shapes class. As can be seen, the dynamics between the numerical model and the experiments are in close agreement.

respectively. As depicted in Figure 15, an oscillatory motion arises, where the fingers sequentially undergo bending.

The objective is to recreate the experimental oscillatory behavior of the soft hand, incorporating the fluid dynamics and continuum dynamics of the soft fingers. Each soft finger can be modeled separately using the `Shapes` class and its internal fluidic network using the `Fluidic` class. We start by sculpting the model for the soft fingers, assuming a fifth-order Chebyshev polynomial basis and considering planar bending curvature only. We assume that only the first mode can be actively controlled by the fluidic network, thus the input map is manually assigned using `Shapes.setInputMap(@(q) ... )`. The `Shapes` class is duplicated five times using a for-loop routine, with the base frame assigned accordingly at each iteration using `Shapes.setBase`.

For each soft finger, it is assumed that it has its own fluidic network that is equipped with internal pressure regulation. However, as each soft finger deforms, the internal volume of the pressure reservoir also changes. This volumetric change is modeled as $V(\alpha) = V_0(1 + \tanh(\alpha))$, where it is assumed that $\alpha = 0.06q_1$ (i.e., the first joint of the soft beam) and V_0 the initial volume. Then, a for-loop is used to include the both `Shapes` and `Fluidic` systems to the network.

Finally, `mdl.addControl(@Control)` is used to add the control law. In this auxiliary function, we specify the pressure reference for the fluidic network, and we return the state deformations of the soft fingers to the fluidic network; required for the computation of the volumetric change. The dynamic simulation is solved implicitly using the `mdl.simulate` command. The code for the dynamic simulation of the soft robotic hand is shown in Figure 15.

```
%% EXAMPLE: Model class
SHP = preset.shapes.soft_hand;

% set input map for each finger
G =  @(x) [5; 0; 0; 0; 0];
for i = 1:5
    SHP{i} = SHP{i}.setInputMap(G) ;
end

% building fluidics model
V0 = (4*pi/3) * 35^3;
V = @(x) V0 * (1 + tanh(x/0.06));

fld = Fluidic('NInput',3);
fld = fld.setVolume(V);

% building model
mdl = Model([]);

for ii = 1:10
    if ii < 5    % add soft finger
        mdl = mdl.addSystem( SHP{ii} );
    else          % add fluidic network
        mdl = mdl.addSystem(fld);
    end
end

% assign control and simulate
mdl = mdl.addControl( @Control );
mdl = mdl.simulate([0, 5.0]);

% -- controller function block
function u = Control(mdl)
    phi = pi/6 * (0:4).';
    Pd  = 80 * sign(sin(7 * mdl.t + phi));

    u = zeros(20,1);
    u(6:3:end) = Pd;
    u(7:3:end) = mdl.getState(1:5,[1]);
    u(1:5)     = mdl.getState(6:10,[1]);
end
```

As depicted in Figure 15, the dynamics of the numerical model and the experiments are in close qualitative agreement. This serves as a testament to the efficacy of the dynamic model composer `Model` in building dynamic complexity through its modular functionalities. Additionally, the ability to represent the controller as an auxiliary function that can retrieve state information at any given time confers `Sorotoki` with a high degree of flexibility in offline controller design.

F. FLUIDIC CONTROL HARDWARE

The implementation of online controllers for physical soft robotic systems is a crucial aspect of `Sorotoki`. Although there are various options available in the research community [18], `Sorotoki` has a specific focus on fast closed-loop control. Drawing from our prior work [127], `Sorotoki` incorporates a TCP communication wrapper (`tcpip`) that enables real-time communication with a host computer, such as

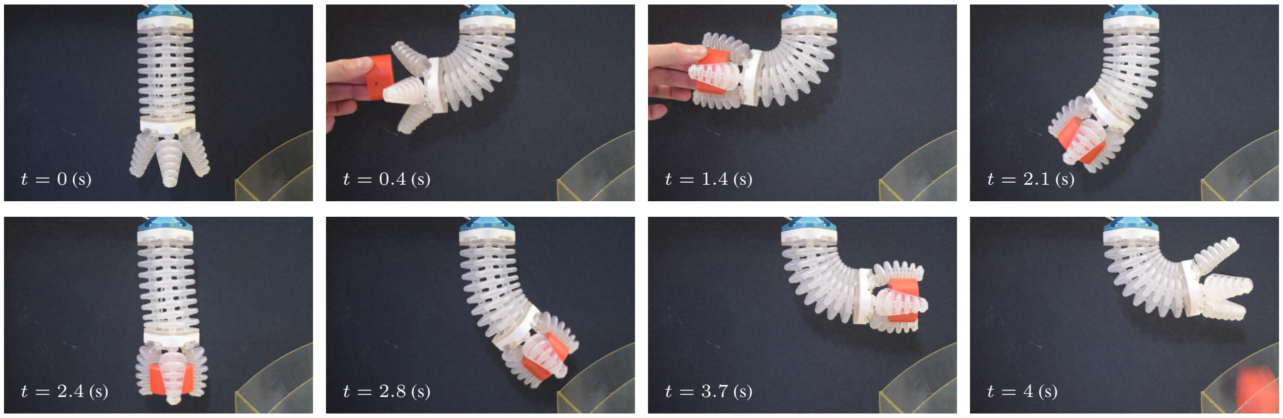

FIGURE 16. Implementation of open-loop control of a 3D-printed soft robot manipulator with a soft gripper using the Sorotoki toolkit. The soft robot has four independent pressure inputs: three for the bellows network embedded in the soft body and one for the soft gripper. The Sorotoki toolkit communicates a desired pressure profile to a Raspberry Pi board computer, which is interfaced with an expandable array of proportional pressure regulators. A straightforward pick-and-place task can then be easily programmed using the Control interface, using auxiliary MATLAB functions.

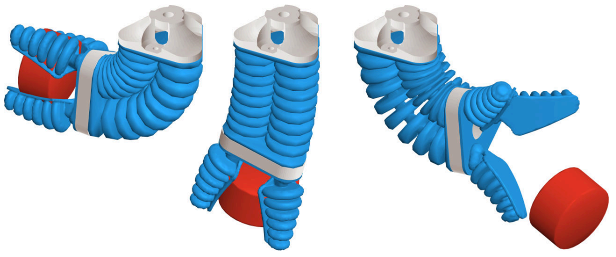

FIGURE 17. Simulation of the soft robot manipulator with gripper using the Shapes and Model class. The rigid-body is modeled by the Newton-Euler equation of motions implemented via StateSpace.

a Raspberry Pi (RPI). This host computer is connected to six pressure control boards, each capable of supporting up to two proportional pressure control valves from Festo. As a result, `Sorotoki` offers up to twelve pressure-regulated control ports with a range of -100 to 100 kPa that can be directly controlled using script-based programming in MATLAB®. Its entirety, including the software, is open-source and readily reproducible by researchers with diverse backgrounds [127].

By calling `brd = Control('ip','pwd')`, connection with the fluid control platform is establish, where `'ip'` is the IP address and `'pwd'` the password of the RPI. On the RPI, the Python script `ConnectToMatlab.py` must be executed that makes connection with MATLAB and awaits control commands. To initiate the control loop, a while-loop is used whose condition statement is `brd.loop(T)` where `T` is finite horizon time. Within the while-loop, all functionalities of `Sorotoki` are available, thus model-based controller design is possible for instance using the `Sdf`, `Fem`, `Shapes`, `Model` classes. Each pressure regulation can be controlled using the command `brd.setInput(id,P)` or get pressure measurements using `P = brd.getOutput(id)`.

> Example: Pick-and-place control of soft robot manipulator

As an example of the capabilities of the fluid control platform, we used it for a pick-and-place application involving the aforementioned soft robot manipulator with soft gripper. The soft robot has four independent pressure inputs: three for the bellows network embedded into the soft body and one for the soft gripper. The `Sorotoki` toolkit communicates a desired pressure profile to a Raspberry Pi board computer, which is interfaced with an expandable array of proportional pressure regulators. As shown in Figure 16, the system successfully manipulates a 40 mm cylinder of 45 (g) into its container.

The system has also been successfully simulated using the `Shapes` and `Model` class, shown in Figure 17, where the cylinder is modeled as a Newton-Euler rigid body system (see `RigidBody` preset classes).

G. COMPUTER VISION

The final Object-oriented class addresses the challenge of soft sensing through vision. The class, named `Vision`, can be instantiated using `cam = Vision(Id)`, where `Id` is a user-specified index obtained from the cell array of available webcams using `webcamlist`. Alternatively, the `Vision` class is capable of reading sensor data from the RealSense D400 series RGB-Depth camera from Intel. The class is equipped with a suite of vision techniques that make use of the `OpenCV` Python implementation. It features three key functions: (i) extraction of optical markers from an image using RGB and depth data, (ii) calibration of the world coordinate frame using Aruco markers, and (iii) monitoring of a soft robot in real-time using camera feedback. The detection of color markers utilizes a circular Hough transform [128], which provides the pixel location of a circle within the specified search conditions. These tools provide a broad range of options for state estimation of a soft robot, which can be easily incorporated into closed-loop control schemes.

VI. SOFT ROBOTICS STUDY CASES

In the subsequent section, we will dive into the capabilities of the toolkit. To provide a comprehensive overview of the toolkit, various problem scenarios will be considered, each with specific problem settings aimed at the design, modeling, or control of soft robots. We will also focus on widely cited papers in the field of soft robotics and demonstrate these, mostly experimental, works using the `Sorotoki`.

Remark on result reproducibility

We like to emphasize that the full code can be accessed in the repository under the folder `./examples/paper/`, enabling users of the toolkit to reproduce all presented simulation results in the following section. Presets are found under the `./+preset/` folder.

A. MULTI-LEGGED SOFT PASSIVE WALKER

In the first case study, the `Sorotoki` toolkit will be used to examine the dynamics of a multi-legged soft passive walker. The work of Suzumori and Saito [129] served as a key source of inspiration for this modeling problem. They proposed using a specialized soft structure that consists of an array of V-shaped soft legs, which exhibit stable intrinsic locomotion when placed on an inclined surface. This behavior was observed in experiments, as shown in Figure 18. The natural locomotion is driven by the elastic deformations of the V-shaped legs and their interaction with the environment, while propelled forward by gravity. To increase the amplitude of these harmonics, small weights are placed at intermediate locations on the connecting soft body between the pairs of V-shaped legs. Each pair of soft legs is tuned to a natural resonance frequency, and when coupled in parallel through a central deformable elastic body, synchronization occurs between the legs during locomotion. In other words, after a transient period, each leg pair will converge to a similar limit cycle, but with a phase offset relative to its neighbor(s).

The objective of this study is to reproduce the dynamics of the soft passive walker described in Suzumori et al. [129] using `Sorotoki`. In the work of Suzumori et al. [129], the material parameters of the soft passive walker were given as: Young's modulus $E_0 = 1.2$ MPa, Poisson ratio $\nu_0 = 0.49$, and density $\rho_0 = 3600 \cdot 10^{-12}$ kg mm^{-3}. Since the material model is not exactly specified in [129], a Neo-Hookean model was utilized with Rayleigh damping $\zeta = 1.5$.

To design the geometry of the V-shaped soft legs, the `sStrut(V1,V2,W)` function was utilized. This function generates an element of the `Sdf` class, requiring two nodal positions `V1,V2` and the strut's width `W` as input. The function was used to assemble a pair of legs iteratively, using the union operator implemented as MATLAB's '+' arithmetic. The legs were then horizontally repeated three times with a uniform spacing of 25 mm. A coupling soft body was added, along with two weights at intermediate locations. The resulting SDF is first converted to an `.png` contour image and then imported using `msh = Mesh('SDF.png','ElementSize',1.0)` to generate the finite element mesh.

The mesh is then utilized to construct the finite element model, i.e., `fem = Fem(msh)`. The timestep for the implicit solver is set at $\Delta t = 0.33$ ms, which is set using `fem.setTimeStep(dt)`. Instead of modeling the inclined surface, which would also require rotating the mesh, the direction of the gravitational acceleration vector is modified as follows: $g := \mathrm{Rot}_y(\varphi) a_g$ with $\varphi = -\frac{\pi}{6}$ (rad). The gravitational acceleration is then added using the class

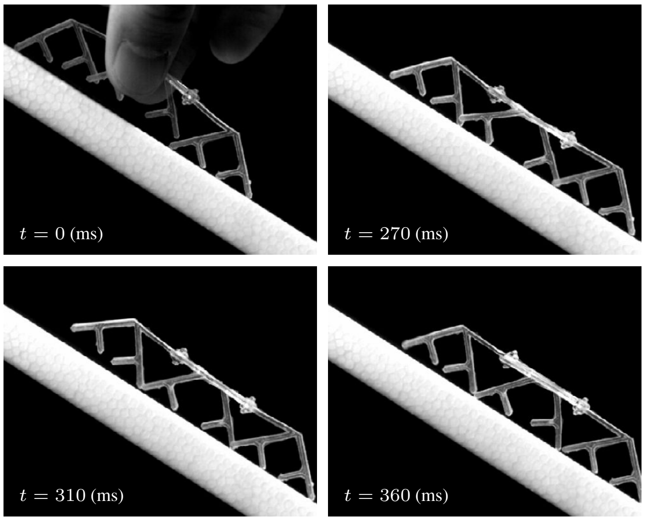

FIGURE 18. Snapsnots of the multi-legged passive walker from Suzumori and Saito [129] can be observed. The soft walker is placed on an inclined surface with a slope of $\varphi = -30°$ and initiates locomotion from an offset in the gravitational potential. These video frames were captured using a high-speed camera. Images from Suzumori et al. [129], used with permission with rights belonging to ©2008 IEEE.

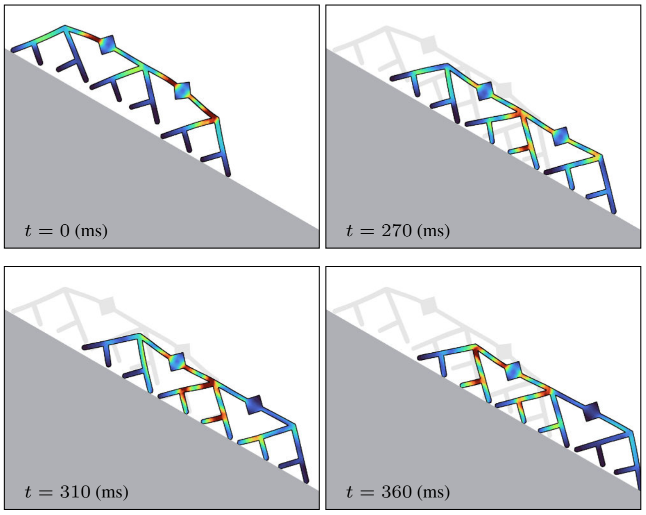

FIGURE 19. Snaphots of the multi-legged passive walker from Sorotoki. The experimental setup is similar to that described in [129]. By comparing the gaits in Fig. 18, a resemblance can be seen between the results obtained in Sorotoki and those reported in [129]. The Von Mises stresses are shown as 🟦🟩🟨🟥 $\in$ [0, 100] MPa.

function `fem.addGravity`. The inclined surface is modeled as a horizontal line SDF, which is used as a contact environment for the FEM model through the function `fem.addContact`. It should be noted that in Figure 18, the soft walker is held in place by two fingers, which results in initial deformations of the soft body and nonzero initial conditions for the dynamic locomotion. To account for this, the mesh is pinned at the grasp locations, and the initial quasi-static deformations are solved for using `fem.solve`. Finally, the forward dynamics are solved implicitly using a Newmark-β solver by calling the routine `fem.simulate`.

Figure 19 shows snapshots of the dynamic simulation of the

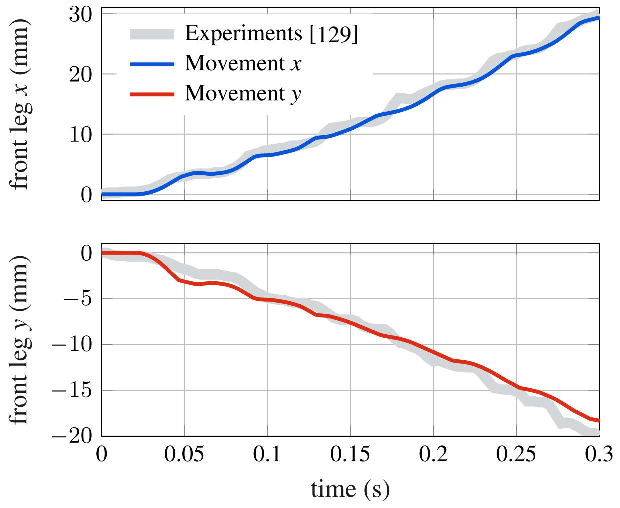

FIGURE 20. Comparison of the front leg rectilinear displacement between the experimental data obtained from Suzumori et al. [129] and the numerical data produced by Sorotoki shows that, although there are discrepancies, the step-like behavior and the traveled distance in the horizontal and vertical directions appear to be truthfully captured in comparison to the original experimental data.

soft passive walker at times corresponding to those depicted in Figure 18. Although slight deviations are noticeable, the overall dynamic characteristics of the locomotion are captured closely by the dynamic FEM model produced using `Sorotoki`. To further demonstrate the validity of the model, a comparison of the rectilinear displacement of the front leg between the experimental data and the simulated model is presented in Figure 20. The experimental data is obtained from [129] and is shown in Figure 20 (in gray). As demonstrated in the figure, the step-like behavior is accurately captured by the numerical model, and the horizontal and vertical distances traveled by the numerical model (within reasonable discrepancy) qualitatively match the original experimental data.

To examine the gait cycle, we introduce the state variables $\theta_1, \theta_2, \theta_3$ to represent the joint angles between the V-shaped legs, as depicted in Figure 21. The trajectory of these angles over a small time window of 200 ms is shown in Figure 22. The angular movements exhibit a clear and consistent "*stable*" gait cycle, indicating that synchronization indeed occurs between the deformable soft legs due to their interaction with the deformable soft body. An analysis of the stable gait cycle reveals a gait period of approximately $T_{\text{gait}} \approx 47.5$ ms or $f_{\text{gait}} \approx 21.1$ Hz.

The numerical simulations presented in this study have effectively demonstrated the capabilities of the `Sorotoki` framework in accurately capturing the complexities commonly encountered in dynamic contact analysis of soft robots. Furthermore, it has been demonstrated that the methodology proposed by Suzumori et al. [129] can be efficiently replicated using a minimal amount of code, specifically, approximately 30 lines of code within `Sorotoki`'s programming language.

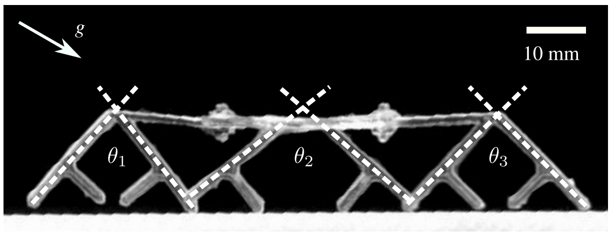

FIGURE 21. Definition of the angular deflection of the three pairs of soft legs, denoted by θ_1, θ_2, and θ_3, respectively. It should be noted that each pair has an intrinsic V-shaped structure, thus their stable equilibrium position during rest is approximately $\theta_i^* \approx 90$ °. Raw image obtained from Suzumori et al. [129], used with copyright permission with rights belonging to ©2008 IEEE.

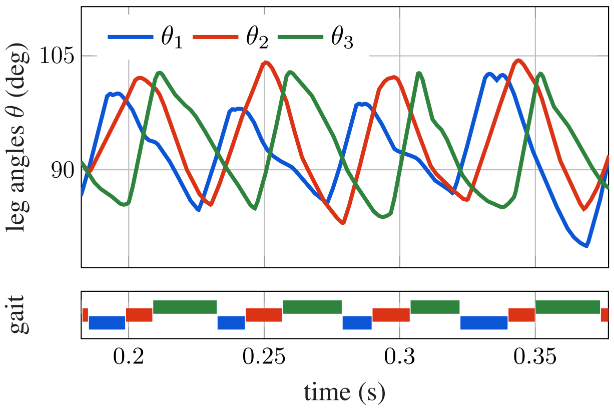

FIGURE 22. Angular deflection of the V-shaped structure of the soft passive walker, simulated with Sorotoki. A clear gait cycle is observed in these deflections, indicating synchronization between the deformable structures due to the coupling of the soft body. By analyzing the stable gait cycles, a gait period of approximately $T_{\text{gait}} \approx 47.5$ ms or $f_{\text{gait}} \approx 21.1$ Hz is found.

B. COMPUTATIONAL DESIGN OF PNEUNET ACTUATOR

In this section, we demonstrate the use of finite element models to aid in the design of PneuNet actuators, a popular type of soft robot actuator. PneuNet actuators, which have been in use since the 1980s (e.g. Teleshev gripper), utilize a rectangular-shaped actuator with a stiffness differential to achieve a bending motion. Recent developments in the field, such as the work of Mosadegh et al. [92] and Ilievski et al. [130], have proposed a modern variation they name "*PneuNet*" actuators that incorporate an inextensible but flexible bottom layer to further enhance the bending motion. The motion of a soft actuator depends on the interaction between the soft material, structural geometry, and the locations where external loads are applied. In their work, Mosadegh et al. [92] demonstrated the importance of geometry in the performance of PneuNet actuators by proposing a new design, called the fast PneuNet (fPN), that improved upon the earlier slower PneuNet (sPN) designs presented in [130]. The fPN design requires less gas for inflation and thus significantly increases the actuator's performance. Design optimization for PneuNets remains an active area of research, as evidenced by recent studies [65], [131]. This demonstrates the continued interest in design optimization of PneuNets, despite decades of research since its inception.

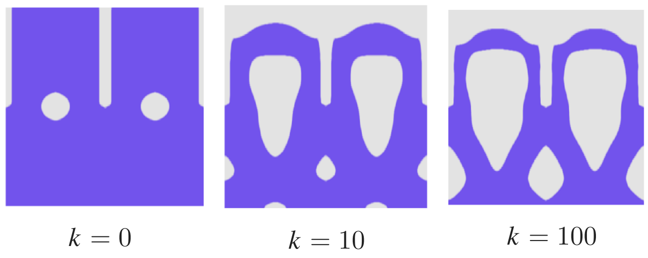

FIGURE 23. Evolution of the topology-based optimization routine in Sorotoki. At $k = 0$, we see the initial guess for the PneuNet actuator, and at $k = 100$ we see a converged solution of the optimizer. Observe that the algorithm proposes a solution very similar to the PneuNet, but instead, it has a teardrop shape rather than the classical rectangular shape. It is worth mentioning that the optimizer accounts for the hyper-elastic material properties - in this case, Elastic 80A resin by Formlabs.

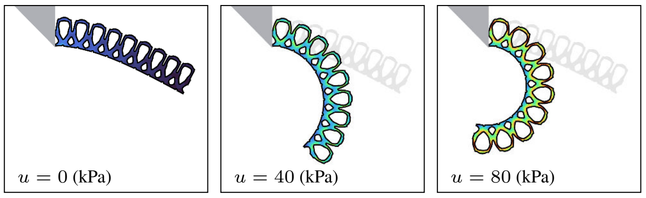

FIGURE 24. Nonlinear finite element simulation of the optimized PneuNet actuator using Sorotoki. The system is subjected to a linear ramp upto 80 kPa, and we observe the classical bending behavior of PneuNet actuators. The Von Mises stresses are shown as 🟦 $\in$ [0, 10] MPa.

The purpose of this example is to demonstrate the use of Sorotoki's design capabilities to optimize and create a PneuNet actuator. We will apply an inverse design method to find the optimal configuration of a soft material that undergoes pure bending when pressurized. This approach extends upon the work presented in our earlier work [111]. To extend of our prior study, we aim to show that the optimized designs produced through this computational design method can effectively overcome the Sim2Real hurdles. To find the optimal material arrangement, we will use a nonlinear topology optimization technique, specifically designed for compliant mechanisms applicable to soft materials.

The objective in the nonlinear topology optimization approach is to find the optimal material distribution $\rho^\star = \text{argmin}_\rho - \boldsymbol{L}^\top \boldsymbol{x}(\rho)$, where $\boldsymbol{L}$ is a sparse unit vector that selects the nodal displacements that promote bending motion. Once an optimum is found, the material distribution $\rho^\star$ can be transformed into a 3D model and manufactured using a commercial printable materials, such as Elastic 80A resin (Formlabs). The optimization algorithm can take into account the specific mechanical properties of the selected printing material, allowing for an optimal design that is tailored to the material properties and design specifications.

To simplify the problem, we consider a single pressure chamber of the PneuNet actuator. To do this, a rectangular design domain with a size of 15×30 mm is defined using a `Sdf` library within `Sorotoki`. The `Mesh` class is then utilized to generate a mesh, which is used to construct a finite element model (FEM) using the `Fem` class. The `Fem` class takes several arguments to set up the optimization solver conditions, including the volume infill, penalty value, filter radius, time steps, and the maximum number of iterations for the Method of Moving Asymptotes (MMA).

The initial material distribution is set using the command `fem.initialTopology(sdf)` with `sdf = sCircle(5,[7.5,15])`, which creates a hole in the center of the actuator. The center element of the mesh is designated as an invariant pressure input and influences neighboring elements that satisfy the void conditions (i.e., $\rho_i \leq \varepsilon$ with $\varepsilon = 0.1$) using an efficient flood-fill algorithm. The influenced void elements undergo synchronous volumetric expansion to simulate a positive pressure load. Given its similarities to muscular contraction, the syntax for this function is added as `fem.addMyocyte`. The material properties of the Elastic 80A resin from FormLabs are then specified using `fem.Material = Elastic80A` [18]. Boundary conditions are added to the FEM model using the syntax `fem.addSupport`. Finally, the optimization routine is started using the `fem.optimize` command.

The evolution of the material distribution during the first 100 optimization steps is depicted in Figure 23. These interpolated isosurfaces are taken from the discrete FEM mesh and show the intermediate design solutions. Surprisingly, the optimization algorithm generates a design that is reminiscent of the fast PneuNet design presented by Mosadegh et al. [92], but with a bellows-shaped pressure chamber in the form of an upside-down teardrop shape.

Next, the focus shifts to validating the optimization results. The aim is two-fold: (*i*) to validate that the optimization algorithm indeed produces the desired bending motion, and (*ii*) to verify if the design suggestion can be successfully transferred to reality (Sim2Real). To do this, the results of the optimization from `fem.optimize` are converted into a triangular mesh using `msh = fem.exportMesh`. Then, boundary conditions are assigned, such as a clamped boundary, gravitational loads, and internal pressure loads for each embedded PneuNet chamber. The same material model for `Elastic80A` resin is chosen. The quasi-static FEM simulation results of the optimized PneuNet actuator for linearly increasing pressure loads of $u = 80$ kPa are shown in Figure 24. As can be seen, the desired bending behavior is achieved in the simulation.

Next, the optimized isosurface shown in Figure 23 can be transformed into a 3D CAD model and printed as a physical soft actuator using a Form3+ SLA printer (FormLabs) with Elastic 80A resin. To validate its performance, the soft actuator is subjected to a linearly increasing pressure load of 80 kPa in 5s window. As seen in Figure 25, the optimized soft actuator successfully performs the desired bending motion, indicating the feasibility of crossing the Sim2Real barrier.

To quantify the discrepancies between the FEM predictions and the actual system, an optical marker is placed at the tip of the soft actuator. The spatial coordinates of the optical marker are obtained using the `Vision` class

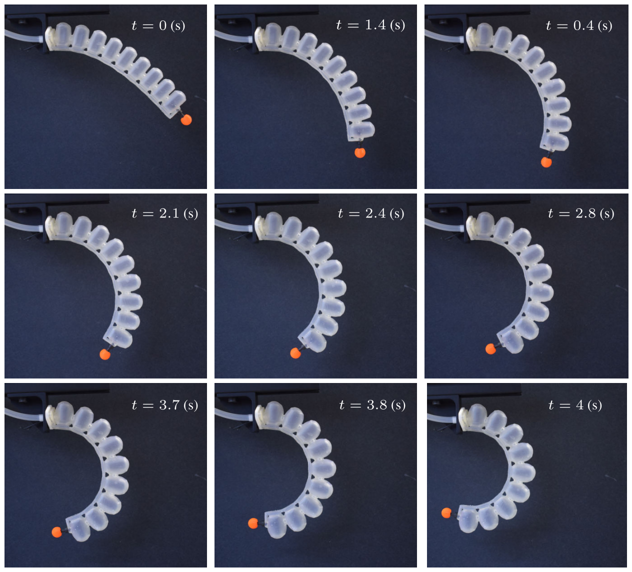

FIGURE 25. Validation study of a 3D-printed *PneuNet* actuator optimized using Sorotoki. The soft actuator is printed using SLA on a Form3+ printer using Elastic 50A UV-resin. Similar to the numerical simulations, we vary the pressure from 0 to 80 kPa with a linear ramp. To measure the planar displacement, an orange marker is placed such that the Vision class can be employed.

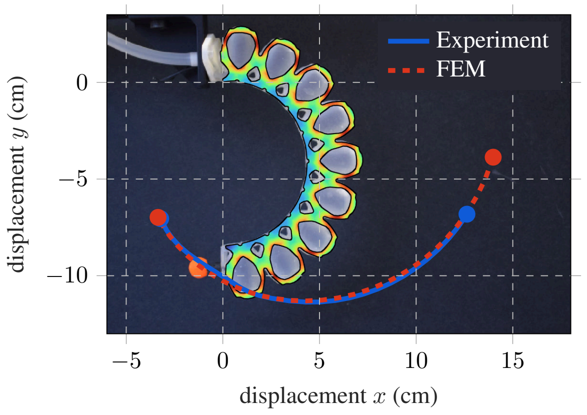

FIGURE 26. Comparison between the numerical simulation in Sorotoki and the experimental results where the orange marker is tracked using the Vision.m class is shown. The Von Mises stresses are shown as $\in [0, 10]$ MPa. The results indicate a close overall trend between the simulation and experiment. However, a discrepancy in the initial deformation ($u = 0$ kPa) is observed. It is hypothesized that this discrepancy is attributed to the inherent creep of SLA resin materials, which leads to a predeformed continuum due to the slow relaxation of the material prior to actuation.

of `Sorotoki`, which uses the color-filtered Hough-space circle transformation to return the pixel coordinates of the marker. These measurements are collected using a RealSense D435 RGB-depth camera (Intel). To retrieve the spatial location of the color marker, we use the command `cam = Vision('realsense')` together with the function `cam.getMarker(R,rgb)`, where `R` is an estimate of the color marker radius in pixels, and `rgb = [r,g,b]` is the RGB color value of the marker.

The comparison between the FEM predictions and experimental results is presented in Figure 26. The deformation patterns of the FEM model and the physical system show close agreement, with an average error of ± 2 mm. However, there is a noticeable difference in the initial conditions, as shown in Figure 26. For $u = 0$ kPa, under pure gravitational loads, the deformations deviate significantly. The cause of this discrepancy is believed to be related to post-curing and internal stress relaxation of the photopolymerization process. This suggests that the stress-free configurations of the FEM model and the physical system may differ, but accounting for this stress-relaxation phenomenon in photopolymer printing is outside the scope of this study and the `Sorotoki` toolkit.

Despite the presence of some differences, the numerical and experimental examples presented in this study highlight the ability of the computational design framework within `Sorotoki` to generate purposeful and useful material distributions with limited prior knowledge of conventional soft robotic design practices. This not only speeds up the design process, but it also opens up the possibility of creating new and innovative soft robot forms [111].

C. DYNAMIC GRASPING AND MANIPULATION USING A HIGHLY DEXTEROUS SOFT GRIPPER

In this section, we will examine the use of reduced-order models for soft beams within the context of the `Sorotoki` software. These models are designed for efficient simulation by exploring minimal state representations of the dynamics of continuum systems. To demonstrate the capabilities of the soft beam modeling framework within `Sorotoki`, we will consider a specific example of a soft robotic system proposed by Suzumori and Suzuki in their seminal work [4], [5]. Despite being published in the late 1980s, the work by Suzumori et al. is still recognized as a seminal contribution to the field of soft robotics and remains relevant to this day.

In their research, Suzumori et al. developed a highly dexterous soft gripper consisting of four microfluidic soft actuators driven by an electro-pneumatic control system. Each finger has three internal pressure chambers, which together provide three controllable degrees of freedom at the fingertip, including pitch, yaw, and linear stretch. Unlike classic rigid grippers, the soft body of the gripper conforms to external forces, enabling intrinsic adaptation during grasping or manipulation tasks. As an illustration of the static grasping capabilities of this soft gripper, Figure 27 depicts two grasping configurations: a pinch grasp for a 40 mm diameter glass beaker (left) and a two-finger pair-pinch grasp for a 5 mm thick metal wrench (right). Suzumori et al. then employed inverse kinematic and compliance control to relate the tip position and compliance to input pressure values. As shown in Figure 27, they were able to successfully hold and turn a 10 mm hexagonal bolt, with an average speed of 0.25 revolutions per second. Due to the wide range of dexterous actions performed by the gripper, the soft gripper proposed by [4], [5] remains a seminal contribution to the field of soft robotics, demonstrating the potential of the technology.

The objective of this investigation is to replicate the static grasping and dynamic bolt-screwing experiments as con-

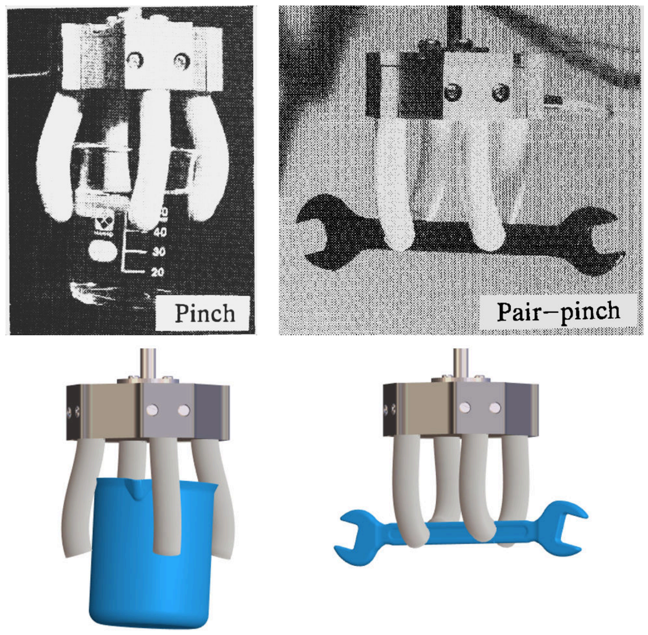

FIGURE 27. (top) Original pinch and pair-pinch grasping experiments of the soft gripper published in Suzumori et al. [4]. The high compliance of each soft finger allows for an adaptive, stable grasp that conforms to the shape of the rigid object. (bottom) A reconstructed soft gripper using the Sorotoki framework. To model each soft finger, we utilized the Shapes class and composed the entire gripper using the Model class. The rigid objects were modeled using the Sdf class. We observed a close resemblance between our simulation model and the original experiments performed by Suzumori et al. in [4]. Used with copyright permission with rights belonging to ©1991 IEEE.

ducted by Suzumori in their published works [4], [5], utilizing the `Sdf`, `Shapes`, and `Model` classes available in the `Sorotoki` software. The soft gripper's design specifications are based on the parameters provided by the available literature, which comprise a radius of $R_0 = 6$ (mm) and an assumed length of $L_0 = 80$ (mm) for each finger. Although the properties of the soft gripper's material are not mentioned explicitly, we use a Neo-Hookean model ansatz to address this issue. The proposed model has an elasticity modulus of $E_0 = 1.0$ (MPa), Poisson ratio of $\nu_0 = 0.3$, and density of $\rho_0 = 2000 \cdot 10^{-12}$ (kg mm^{-3}). The reduced-order model for each soft finger in interaction with a rigid object, denoted by $\Sigma_{\mathrm{SR},i}$, is described by the following equation:

$$\Sigma_{\mathrm{SR},i}: \quad \boldsymbol{M}(\boldsymbol{q}_i)\ddot{\boldsymbol{q}}_i + \boldsymbol{h}(\boldsymbol{q}_i,\dot{\boldsymbol{q}}_i) = \boldsymbol{G}(\boldsymbol{q}_i)\boldsymbol{u}_i + \dots$$
$$+ \sum_{j\in\mathcal{S}_{\Omega_{\mathrm{env}}}} \boldsymbol{J}_{v,j}^{\top}(\boldsymbol{q}_i)\Big[\boldsymbol{F}_{n,j}(\boldsymbol{q}_i) + \boldsymbol{F}_{t,j}(\boldsymbol{q}_i,\dot{\boldsymbol{q}}_i)\Big], \quad (30)$$

where $\boldsymbol{h}(\boldsymbol{q}_i,\dot{\boldsymbol{q}}_i)$ represents the collection of nonlinear internal forces, $\boldsymbol{u}_i$ is the prescribed pressure input, $\boldsymbol{J}_v(\boldsymbol{q}) := \lfloor \boldsymbol{J}(L,\boldsymbol{q})\rfloor_3$ is the linear velocity part of the generalized Jacobian matrix of the tip, $\mathcal{S}_{\Omega_{\mathrm{env}}}$ represents the nodal indices that are considered "*in contact*", and $\boldsymbol{F}_{n,j} = -\mu_e d_j \cdot \boldsymbol{e}_n$ and $\boldsymbol{F}_{t,j} = -\mu_v|\boldsymbol{F}_{n,j}|\,\mathrm{sgn}(\dot{d}_j)\cdot \boldsymbol{e}_t$ denoting the normal and tangent contact forces between the i-th finger and the rigid object, respectively. The parameters $\mu_e, \mu_v > 0$ represent the contact coefficients. The distance between the finger and the object is given by $d_j = \max\{0, \mathtt{sdf}_{\Omega_{\mathrm{env}}}(\gamma_j(\boldsymbol{q})) - r_j\}$, where $\mathtt{sdf}_{\Omega\mathrm{env}}(\cdot)$ represents the signed distance function of the inelastic contact object, $\gamma_j(\boldsymbol{q})$ the finger's center-curve position, and r_j the radius of the soft finger.

In this study, we utilize a third-order Chebyshev polynomial basis to model the deformation of the pneumatic soft robot's finger. The basis is sampled over $N_p = 100$ uniform nodes and is assembled into a matrix using the command `pod = Basis(100,3,'chebyshev')`. It is assumed that only free strains occur in the bending, while elongation and torsion are neglected. Each strain mobility vector is characterized by three modes of the Chebyshev basis, specified as `dof = [0,3,3,0,0,0]`, leaving the κ_x and κ_y curvatures free. The dynamic model for each finger is constructed using the command `shp = Shapes(pod,dof)`. The material properties are assigned using `shp.Material = NeoHookean(1.0, 0.3)`. To set the geometry of each finger, we call `shp.setLength(80)` to set its length, and `shp.setGeometry(sCircle(6))` to set a circular cross-section of radius 6 mm. Each finger of the soft gripper model is equipped with three fluidic chambers that are radially distributed along its circumference. As such, the input map $\boldsymbol{G}$ for each finger becomes a nonlinear, non-square matrix. We assume that these distributed forces can be represented by a tangent bundle of linear forces that are positioned 3 mm away from the center axis. To assign these forces, the `shp.addFluidic(@p)` command can be utilized, which requires an anonymous function `@p` that describes the desired motion path of the soft actuator (feed-forward).

The full soft gripper model, composed of four identical finger `Shapes` classes, can be assembled using a for-loop routine. In this process, a class representing each finger is copied and assigned a unique SE(3) base frame to each instance. Each finger is placed in a circular array with a radius of 37.5 mm relative to the center axis of the gripper body. The contact domain for each finger is specified using the method `shp.addContact(sdf)`, where `sdf` denotes the signed distance field of the contact object. For the beaker example, a cylindrical SDF with dimensions $40 \times 40 \times 60$ mm is considered, while for the wrench, a rectangular SDF with dimensions $5\times 10\times 100$ mm is used. Subsequently, each instance of the `Shapes` class is appended to the `Model` class constructor using the `mdl.addSystem(shp)`.

Once the full model has been assembled, it can be controlled using a feed-forward controller. In the case of a static grasping scenario with a glass beaker and wrench, we apply pressure ramps to each pressure chamber of the soft gripper through an auxiliary anonymous function, `@(mdl) Control(mdl)`. The function takes in a time variable and outputs a column vector of pressure signals, represented by $\boldsymbol{u} = (\boldsymbol{p}_1,\boldsymbol{p}_2,\boldsymbol{p}_3)^{\top}$. The controller is then assigned to the Model class using the command `mdl.addControl(@Control)`, which is executed at each simulation step. The forward dynamics of the soft gripper's interaction with the object are implicitly solved through the `mdl.simulate` routine. The simulated grasping con-

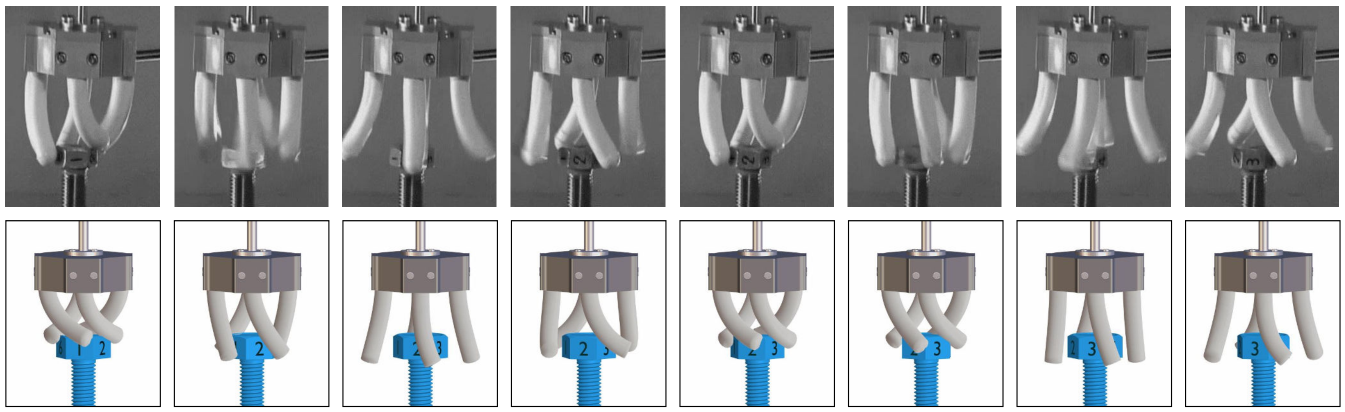

FIGURE 28. (top) Snapshots of the bolt screwing experiment with the soft gripper, as presented in the work of Suzumori et al. [4], [5]. The soft gripper periodically switches through a predefined set of configurations, enabling the holding and manipulation of a hexagonal bolt screw. In the experiment, a bolt turning rate of 0.25 rps was achieved. (bottom) The bolt screwing experiment is reproduced using Sorotoki. In the simulation, each finger is modeled utilizing the class Shapes and assembled together into Model. The contact interaction with the bolt is modeled using signed distance functions, and a rotational mass-damper model is used to describe the dynamics of the bolt. By utilizing solely the frictional interaction between the fingers and the screw, the experiment of Suzumori et al. is successfully reproduced.

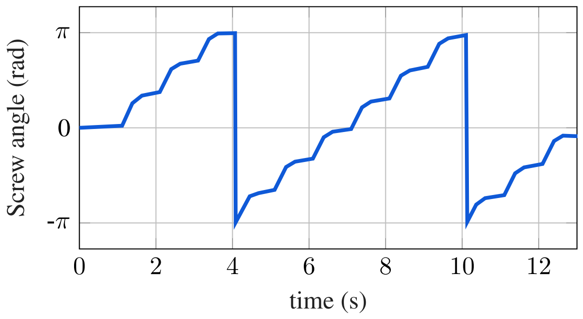

FIGURE 29. The evolution of the hexagonal bolt angle $\theta(t)$ is depicted, where the stair-like trajectory of the screwing motion, as observed in Suzumori's experiment, is apparent. Through careful parameter and input shaping, a bolt-screwing motion of 0.16 rps was achieved. This is slightly slower than the reported rate of 0.25 rps, however, the underlying morphological characteristics (stair-case pattern) are accurately captured.

figurations are shown in Figure 27. It is evident that there is close agreement with the experiments in [4], [5].

Subsequently, we aim to reproduce the hexagonal bolt screwing experiment of [4], [5], which involves a more complex simulation than the previous scenario due to the dynamic interaction between the environment and the soft robot. To accurately depict this interaction, we must also incorporate contact dynamics where we employ signed distance field to describe the space occupied by the hexagonal bolt. We assume the bolt's rotational dynamics can be described by a mass-damper system, represented by the following equation:

$$I_\Omega\,\ddot{\theta} = -\mu_\Omega\dot{\theta} - \sum_{i=1}^{N_{\text{finger}}}\sum_{j\in\mathcal{S}_\Omega} \boldsymbol{r}_j(\boldsymbol{q}_i) \times \boldsymbol{F}_{t,j}(\boldsymbol{q}_i,\dot{\boldsymbol{q}}_i)\cdot\boldsymbol{e}_3, \qquad (31)$$

where $I_\Omega =$ is the inertia of the hex-bolt, $\mu_\Omega = 2.5$ its friction coefficient, and $\boldsymbol{r}$ the relative position vector from the point of contact and the central turning axis of the screw. Note that we only include the tangential force components $\boldsymbol{F}_t$ that are responsible for motion, as the normal forces are assumed to have a net zero-torque contribution. The model described in equation (31) is incorporated into the simulation by using the command `mdl.addSystem(@f)`, where `@f(x,u,t)` is an anonymous function that represents the state space. The required input `u` for equation (31) is connected to the soft gripper by utilizing the `mdl.addControl(@Control)` command, which inputs tangential reaction forces into the screw model. The controller also includes the prescribed pressure profile $\boldsymbol{p}_{1,2,3}$ for each of the four soft fingers.

The qualitative comparison between the experiments conducted by Suzumori et al. [4], [5] and our surrogate model made using `Sorotoki` is depicted in Figure 28. In the simulation of the Suzumori soft gripper's screwing experiment, similar to the simulation of the static object grasping scenario, we observe that the qualitative behavior reflects the real-world experiment conducted in the 1980s. The soft finger exhibits similar deformation characteristics, and we also observe the step-like turning of the bolt screw, as reported in [5]. To highlight these rotational trajectories, we present the rotation angle $\theta(t)$ in Figure 29, which shows that an average bolt-screwing speed of 0.16 rps is achieved. Although this rate is slightly slower than the reported rate of 0.25 rps, it is believed that the underlying morphological characteristics are accurately captured. Note that although the system is of highly complex nature, the simulation program contains less than 100 lines of code (including visualization).

D. IMPEDANCE CONTROL OF SOFT ROBOT MANIPULATOR WITH STATIC ENVIRONMENTAL INTERACTION

The subsequent section will focus on the development of controllers utilizing the `Model` and `Shapes` classes. In prior experiments, the Suzumori et al. gripper was governed in an open-loop fashion, with complications arising from the interplay between the model and dynamic object. Our investigation will now examine the feasibility of model-derived controllers in the `Sorotoki` scenario, drawing upon Della

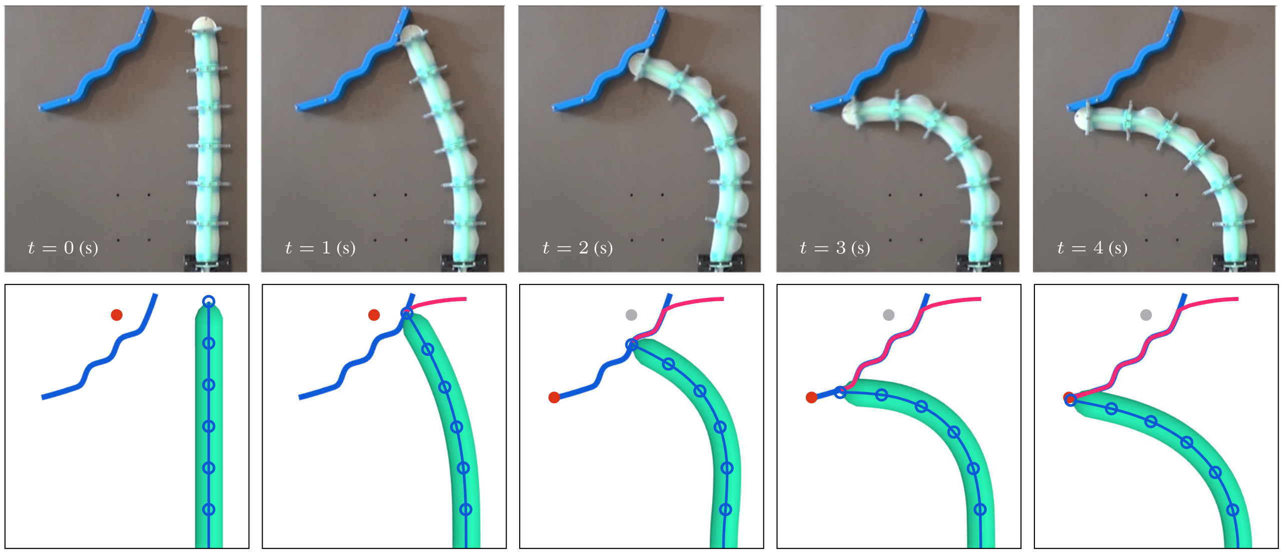

FIGURE 30. (top) Original experimental results presented in the work of Della Santina et al. [118] show a six-link soft manipulator controlled by the control law described in (32). The manipulator is in contact with a static environment. The controller first ensures that the soft robot approaches the environment (Phase 1: Approach) and then follows the rigid wavy surface (Phase 2: Explore). (bottom) The numerical simulations performed using the Sorotoki toolkit are inspired by the work of Della Santina et al. [118]. In the simulation, the active setpoints are denoted by (●) and the de-activated setpoints are denoted by (●). The end-effector trajectory is shown in (▬) and the environment is shown in (▬). The results of the simulation truthfully reflect the approaching and surface tracking behavior as seen in the experiments. Used with copyright permission with rights belonging to Sage Journals.

Santina et al.'s work [118] as a prospective case study.

Their work presents the design of model-based controllers for soft robot manipulators, highlighting two control architectures: dynamic tracking and surface tracking. The authors proposed a Cartesian impedance controller for the latter architecture, which actively regulates the desired compliance behavior of the soft robot's end-effector in a static environment. Additionally, the work presented a contact path planning algorithm that initially brings the robot close to a desired setpoint on the surface (Phase 1: Approach), and then adjusts the setpoint to maintain contact with the surface (Phase 2: Explore). The proposed controller was evaluated both numerically and experimentally on a six-link soft robot manipulator. They demonstrated that model-based control can significantly improve the soft robotic performance compared to open-loop.

To maintain high computational bandwidth, the impedance controllers used in Della Santina et al.'s study [118] incorporate an augmented rigid body model. This model employs Constant-Curvature (i.e. PCC) kinematics to approximate the center of mass of the continuously deformable robot by projecting its mass distribution into a lumped mass description. This leads to an Euler-Lagrangian representation similar to the commonly used Denavit-Hartenberg (DH) parametrization models in rigid robotics. Moreover, this approach maintains classical properties, such as positive semi-definiteness of the inertia matrix and passivity properties, which are highly sought after for stability analysis.

Our aim in this section is to replicate the results of the closed-loop controlled multi-link soft robot during dynamic contact that were presented in the study by Della Santina et al. [118]. Instead of employing the augmented rigid body model used in their work, we explore a reduced-order beam model, in which each link is represented as an inextensible, distributed-mass PCC segment. Our approach extends their work to a distributed mass robotic system. As a template for the soft manipulator model, we use a `Shapes` template under `preset.shapes.katzschmann`. According to [118], each CC segment of the soft manipulator has an intrinsic length of $\delta L_0 = 60$ mm and a mass of $m_0 = 334$ g. The robot has a rounded rectangular cross-section of 60 $\times$ 20 mm, which is described using `sdf = sSquircle`. The density of the soft robot manipulator, given its length and geometry, is approximated to be $\rho_0 = 1200 \cdot 10^{-12}$ $(\mathrm{kg\,mm}^{-3})$.

A crucial aspect of the simulation is the dynamic interaction with the environment. Therefore, a static environment must be assigned to the dynamic model. While [118] presents multiple examples of line contact, this study focuses on the experiment with a 40° slanted surface with wave indentations, as shown in Figure 30. The surface features were extracted from the image data presented in [118] and the `env = sPolyLine(V)` function was used to generate the SDF environment, where `V` is a polyline vector. The environment is then added using `Shapes.addContact(env)`. Once all settings are assigned to the class `Shapes`, the model is constructed by `mdl = Model(Shapes)`.

Given the `Model` class, we can now start deriving a contact-aware control law. For conciseness, let $\boldsymbol{J}(\boldsymbol{q}) := \boldsymbol{J}(L, \boldsymbol{q})$ be the geometric Jacobian of the end-effector. We also introduce the Cartesian inertia matrix as $\Lambda := (\boldsymbol{J}^\top \boldsymbol{M}^{-1} \boldsymbol{J}^\top)^{-1}$. Then, the proposed Cartesian stiffness controller given in [118] can be written as:

$$\boldsymbol{\tau} = \boldsymbol{J}^\top \Big[\boldsymbol{J}_M^{+\top} \boldsymbol{f}_e + \boldsymbol{f}_g + \boldsymbol{J}^\top \eta_C (\boldsymbol{I} - \boldsymbol{J}_M^{+\top} \boldsymbol{J}) \dot{\boldsymbol{q}} + \dots \\ + \boldsymbol{J}^\top (\boldsymbol{K}_c(\gamma_d - \gamma_L) - \boldsymbol{D}_c \boldsymbol{J} \dot{\boldsymbol{q}}) \Big], \quad (32)$$

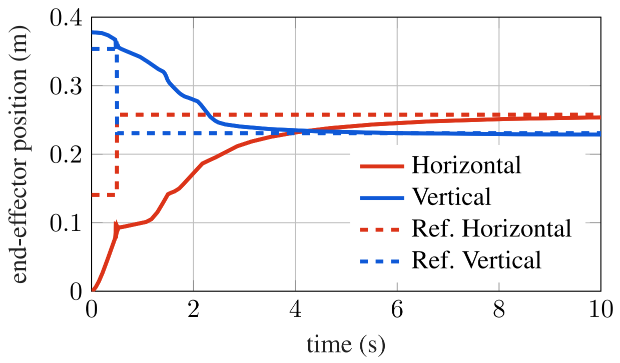

FIGURE 31. The end-effector trajectories of the six-link soft manipulator, where { ▬, ▬} denote the horizontal and vertical position, respectively. The setpoints assigned by the controller are indicated by the dashed lines. Note that at $t \approx 0.88$ s, the point of contact, the controller switches setpoint.

where γ_d and γ_L represent the desired and true end-effector positions, respectively; and K_c and D_c are the desired stiffness and damping of the end-effector. The closed-loop controller employs a so-called *"dynamically consistent pseudo-inverse"* of the Jacobian, denoted as J_M^+, which is defined as $J_M^+ := M^{-1}J^\top\Lambda$. The controller also utilizes the Cartesian Coriolis terms, denoted as $\eta_C(q, \dot{q})$, which are expressed in the Cartesian frame and given by:

$$\eta_C = \Lambda(JM^{-1}C - \dot{J}). \tag{33}$$

The closed-loop controller, implemented as an anonymous function, is derived from four system matrices: the inertia matrix $M(q)$, the Coriolis matrix $C(q, \dot{q})$, the Jacobian J, and its time derivative $\dot{J}$. In `Sorotoki`, these matrices are automatically computed at each solver step and stored in a data structure under `shp.system`. The closed-loop controller can access this data structure at any time.

It is noteworthy that the controller above uses the command `shp.system.fContact` (i.e., contact forces) to detect if the robotic system is in contact. Similar to path planning in [118], upon contact, a new desired equilibrium position is adopted. These desired equilibrium positions for the end-effector are in line with the approach presented in [118]. Subsequently, the implicit solver is invoked by calling `mdl.simulate` to solve the closed-loop dynamics. Snapshots of the dynamics have been presented in Figure 30 which are produced by calling the function `shp.render`. Figure 31 shows the trajectory of the end-effector (dashed lines are the desired setpoints), Figure 32 shows the evolution of the states, and Figure 33 shows the control action τ from (32).

When comparing the experimental results reported by Della Santina et al. [118] and the numerical simulations produced by `Sorotoki`, we observe qualitative similarities in the deformation characteristics. Most notably, the numerical implementation of the impedance Cartesian stiffness controller also follows the inclined surface until the setpoint is reached. These similarities highlight the reliability of `Sorotoki` in accurately reflecting true soft robotic systems, even in closed-loop scenarios.

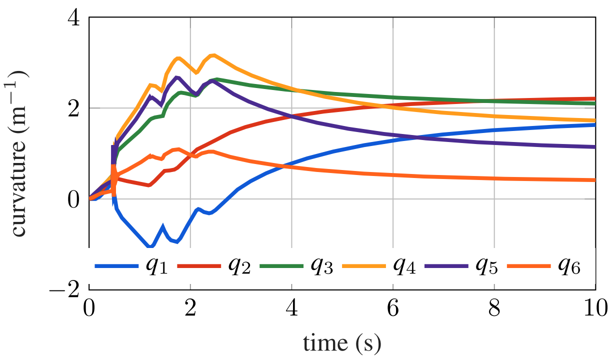

FIGURE 32. State evolutions of the six-link soft manipulator, where the states represent the planar curvature of each individual soft link.

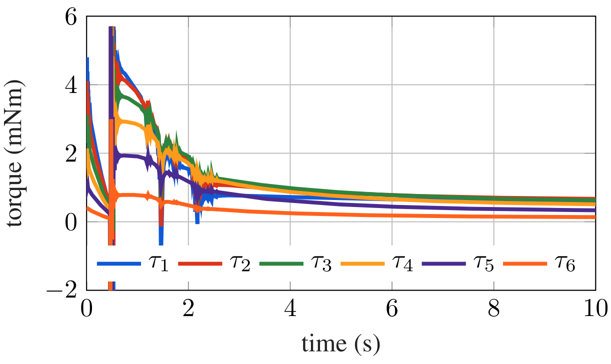

FIGURE 33. The control inputs τ in mNm produced by the control law in (32) exhibit a significant peak at the point of contact. This is due to two factors: (*i*) the sudden change of setpoint and (*ii*) the switch in control strategy to accommodate for compliance.

E. CONTACT ROBUST SHAPE SENSING OF SOFT ACTUATOR (PNEUNET) USING A FEM-BASED MODAL BASIS

In the next section, our focus shifts from simulation to the experimental domain. Our primary focus will be on the `Vision` and `Control` classes, and we aim to provide experimental validation for the Data-driven Variable Strain (DVS) basis approach detailed in Section V-D3. The objectives of this study case are (i) to derive a finite-dimensional Cosserat beam model of the PneuNet actuator and (ii) to implement a real-time shape sensing algorithm that is robust against external forces through the exploration of model information.

We begin our investigation by conducting a nonlinear dynamics analysis of a soft PneuNet actuator, the geometry of which has been selected to match that described in the work of Mosadegh et al. [92]. The soft actuator is suspended vertically in order to produce minimal deformation under zero-input conditions. In pursuit of high-accuracy simulation, we first perform a Finite Element simulation. The generation of the mesh is performed using the preset library `msh = preset.mesh.pneunet`, which utilizes an image of a PneuNet cross-section for mesh generation. The finite element model is then formed using `fem = Fem(msh)`, and the appropriate material properties (*i.e.*, Dragonskin10) and boundary conditions are assigned. The system is subjected to a linearly increasing and decreasing pressure ramp of 40

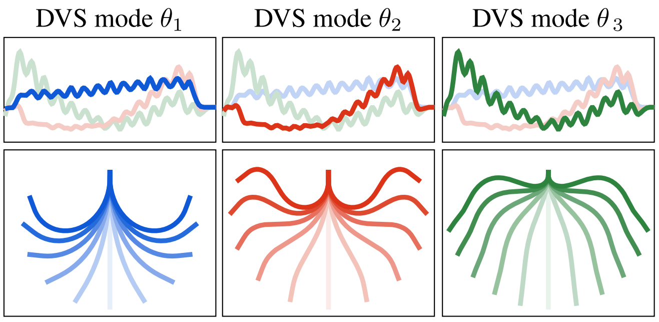

FIGURE 34. **(top) The first three DVS modes of the soft PneuNet actuator in vertical hanging condition, where the ordering is {▬, ▬, ▬}. (bottom) The respective deformation for each strain mode of the PneuNet actuator. Note that these differ from the DVS basis in Section V-D3.**

(kPa), and the dynamics are solved using `fem.simulate`. In accordance with the procedure outlined in Section V-D3, a third-order DVS basis for pure bending is then constructed, which are used to construct the `Shapes` class. The curvature-bending modes are shown in Figure 34.

To implement a real-time estimation algorithm, we can utilize the `Shapes` class. This can be achieved by employing the inverse kinematics algorithm described in Section V-D6. The algorithm can be invoked using the function `Shapes.solveIK(pos, q0)`, where `pos` represents the measured tip position, and `q0` is an optional initial estimate. The inverse kinematics solver is then called repeatedly within the real-time control loop. To ensure that the inverse kinematics solution aligns with the true system behavior, we also incorporate a null-space subtask projection. In this case, the gradient of the subtask is assumed to be $\nabla\Psi_{\text{sub}}(\boldsymbol{q}) = \boldsymbol{K}\boldsymbol{q} + \boldsymbol{f}_{\text{g}}(\boldsymbol{q}) - \boldsymbol{G}(\boldsymbol{q})u$, where $u = 30 \cdot \text{sat}\,[\sin(t)]$ (kPa) represents the prescribed pressure input assigned by the open-loop controller. This subtask serves to minimize the internal residual forces and can be seen as a quasi-static deformation solver guided by camera measurements; and is added to `Shapes.system.SubTask`. It is important to note that the model parameters have been pre-tuned to align with the experimental system presented in Figure 35. However, certain initial estimates of the system parameters have been used, which are derived from the finite element model.

To instantiate the camera class, we call `cam = Vision('realsense')` and establish a Secure Shell (SSH) connection with the control platform through `brd = Control('ip','pwd')`. Once the connection with the control platform is established, an online while-loop containing the necessary shape estimation algorithms is executed.

Figure 35 shows the experiment and overlayed on top are the real-time shape estimations from the inverse kinematics algorithm that uses the Data-driven Variable Strain (DVS) basis. Figure 36 shows the state trajectories and the time instance of contact. Despite the significant impact of contact forces on the soft actuator, the shape estimations qualitatively matches the deformation profile of the real soft robotic system. This is due to its low state dimension of $\dim(\mathbf{q}) = 3$, enabling it to achieve a bandwidth of +60 Hz with ease.

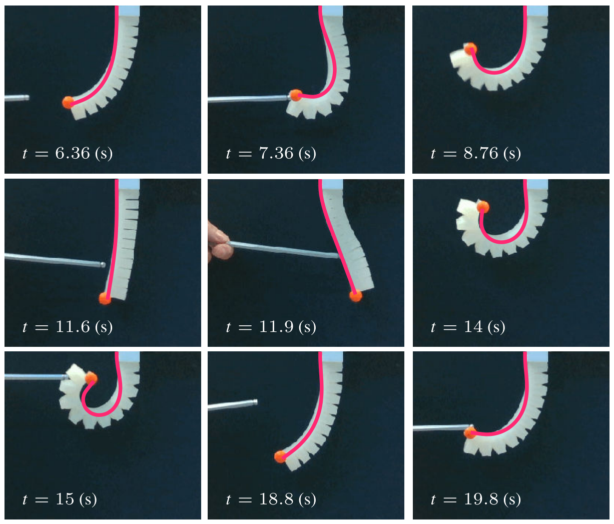

FIGURE 35. **Real-time shape estimation using the Shapes inverse kinematics algorithm in combination with the Data-driven Variable Strain (DVS) basis. The reconstructed backbone curve is depicted in (▬). Despite the presence of substantial contact forces on the soft actuator, the shape estimation algorithm accurately reflects the behavior of the real soft robotic system. Due to the relatively small state dimension of dim(*q*) = 3, the algorithm achieves a bandwidth of +60 Hz.**

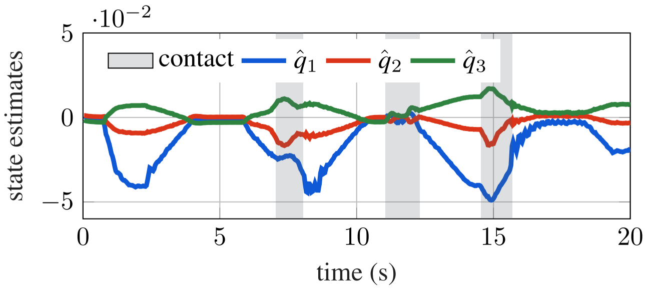

FIGURE 36. **Evolution of the state estimations during the PneuNet experiment, where the ordering is given by {▬, ▬, ▬}. Note that the joint variables represent the modal coefficients of the DVS basis related to curvature-bending.**

VII. CONCLUSION AND FUTURE WORK

This paper introduces `Sorotoki`, an open-source toolkit in MATLAB, that provides a comprehensive and modular programming environment to address the complex interdependencies associated with the design and control of soft robots. The toolkit consists of seven Object-Oriented classes that work together to solve a wide range of soft robotic problems. We hope that the versatility and flexibility of `Sorotoki` make it a valuable resource for researchers and practitioners in the field of soft robotics. The toolkit's effectiveness has been demonstrated through various case studies covering a broad range of soft robotics issues, including inverse design of soft actuators, passive and active soft locomotion, object manipulation with soft grippers, meta-materials, model reduction, model-based control of soft robots, and shape estimation.

A unique aspect of this software package is that it does not follow the traditional linear relationship between the complexity of soft robotics systems and the length of code required to represent them. Instead, complex system behavior can be effectively modeled using a minimal number of lines of code. The `Sorotoki` software package stands out for its ability to succinctly represent complex soft robotics systems. Despite the intricacy of soft robotics, the accompanying software package is highly effective in modeling complex system behavior with minimal code, making it accessible even to individuals with limited programming knowledge. Furthermore, the toolkit provides access to four open-hardware soft robotic systems that can be easily fabricated using commercially available 3D printers.

Nevertheless, the `Sorotoki` framework presents opportunities for improvement and expansion to enhance its comprehensiveness, extensiveness, and achieve faster simulation times. This is particularly crucial in the field of control and optimization, where computational performance is often a challenge. The current software package addresses this challenge by converting MATLAB code to its equivalent in `c++` using `mex` compilation. This enables real-time computation; however, the resulting `.mex` functions may have limitations due to inadequate memory allocation and a lack of parallel processing, leading to subpar performance. To overcome these limitations, an alternative solution worth exploring could be to utilize other programming languages known for better computational performance, such as `Python` (via `pybind`) and `Julia`. These languages have a strong open-access community, which may provide enhanced support and resources for optimizing the framework's computational performance.

On a concluding note, we would like to emphasize that any form of contribution is greatly appreciated. Our framework is envisioned as a collaborative effort between and for the soft robotics community. Researchers who are interested in contributing are welcome to reach out to the authors. Different forms of contributions are possible and more information can be found on the repository. Additionally, while we strive to maintain the framework to be as error-free as possible, in case of exceptions during execution or any concerns regarding accuracy or implementation, please do not hesitate to contact us directly or through the issue tracker on the repository.

VIII. ACKNOWLEDGMENTS

This work has been partially supported by the Netherlands Organization for Scientific Research (NWO) and is part of the Wearable Robotics perspective program with project number P16-05 (P2). Further information can be found on the website: `www.wearablerobotics.nl`. The authors would like to extend their gratitude to M. Meyer for her support with the preparation of the STL models for the soft hand, B. Proper for his support and expertise on Additive Manufacturing, and I. Kuling for the resources regarding soft material 3D printing through the ReShape Lab resources.

REFERENCES

[1] J. F. Wilson and U. Mahajan, "The Mechanics and Positioning of Highly Flexible Manipulator Limbs," *Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 111, no. 2, pp. 232–237, 1989.

[2] S. Weisburd, "The muscular machinery of tentacles, trunks and tongues," *Science News*, vol. 113, pp. 204–205, 1988.

[3] J. F. Wilson and N. Inou, "Bellows-type springs for robotics," in *Proc. Adv. Spring Technol. JSSE 60th Anniversary Int. Symp*, 2007, pp. 109–119.

[4] K. Suzumori, S. Iikura, and H. Tanaka, "Development of flexible microactuator and its applications to robotic mechanisms," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, vol. 2, 1991, pp. 1622–1627.

[5] ——, "Applying a flexible microactuator to robotic mechanisms," *IEEE Control Systems Magazine*, vol. 12, no. 1, pp. 21–27, 1992.

[6] C. Della Santina, M. G. Catalano, and A. Bicchi, "Soft Robots," in *Encyclopedia of Robotics*. Berlin, Germany: Springer, Nov. 2020, pp. 1–15.

[7] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, vol. 521, pp. 467–475, May 2015.

[8] J. Hughes, U. Culha, F. Giardina, F. Guenther, A. Rosendo, and F. Iida, "Soft Manipulators and Grippers: A Review," *Frontiers in Robotics and AI*, vol. 3, 2016.

[9] A. D. Marchese, R. K. Katzschmann, and D. Rus, "A Recipe for Soft Fluidic Elastomer Robots," *Soft Robotics*, vol. 2, no. 1, pp. 7–25, 2015.

[10] N. R. Sinatra, C. B. Teeple, D. M. Vogt, K. K. Parker, D. F. Gruber, and R. J. Wood, "Ultragentle manipulation of delicate structures using a soft robotic gripper," *Science Robotics*, vol. 4, no. 33, p. eaax5425, 2019.

[11] K. C. Galloway, K. P. Becker, B. Phillips, J. Kirby, S. Licht, D. Tchernov, R. J. Wood, and D. F. Gruber, "Soft Robotic Grippers for Biological Sampling on Deep Reefs," *Soft Robotics*, vol. 3, no. 1, pp. 23–33, 2016.

[12] S. Li, D. M. Vogt, D. Rus, and R. J. Wood, "Fluid-driven origami-inspired artificial muscles," *Proceedings of the National Academy of Sciences*, vol. 114, no. 50, pp. 13 132–13 137, 2017.

[13] N. W. Bartlett, M. T. Tolley, J. T. B. Overvelde, J. C. Weaver, B. Mosadegh, K. Bertoldi, G. M. Whitesides, and R. J. Wood, "A 3D-printed, functionally graded soft robot powered by combustion," *Science*, vol. 349, no. 6244, pp. 161–165, 2015.

[14] E. W. Hawkes, L. H. Blumenschein, J. D. Greer, and A. M. Okamura, "A soft robot that navigates its environment through growth," *Science Robotics*, vol. 2, no. 8, p. eaan3028, 2017.

[15] G. Li, X. Chen, F. Zhou, Y. Liang, Y. Xiao, X. Cao, Z. Zhang, M. Zhang, B. Wu, S. Yin, Y. Xu, H. Fan, Z. Chen, W. Song, W. Yang, B. Pan, J. Hou, W. Zou, S. He, X. Yang, G. Mao, Z. Jia, H. Zhou, T. Li, S. Qu, Z. Xu, Z. Huang, Y. Luo, T. Xie, J. Gu, S. Zhu, and W. Yang, "Self-powered soft robot in the Mariana Trench," *Nature*, vol. 591, pp. 66–71, 2021.

[16] M. T. Tolley, R. F. Shepherd, B. Mosadegh, K. C. Galloway, M. Wehner, M. Karpelson, R. J. Wood, and G. M. Whitesides, "A Resilient, Untethered Soft Robot," *Soft Robotics*, vol. 1, no. 3, pp. 213–223, 2014.

[17] T. J. Wallin, J. Pikul, and R. F. Shepherd, "3D printing of soft robotic systems," *Nature Reviews Materials*, vol. 3, pp. 84–100, 2018.

[18] M. S. Xavier, C. D. Tawk, A. Zolfagharian, J. Pinskier, D. Howard, T. Young, J. Lai, S. M. Harrison, Y. K. Yong, M. Bodaghi, and A. J. Fleming, "Soft Pneumatic Actuators: A Review of Design, Fabrication, Modeling, Sensing, Control and Applications," *IEEE Access*, vol. 10, pp. 59 442–59 485, 2022.

[19] J. D. Hubbard, R. Acevedo, K. M. Edwards, A. T. Alsharhan, Z. Wen, J. Landry, K. Wang, S. Schaffer, and R. D. Sochol, "Fully 3D-printed soft robots with integrated fluidic circuitry," *Science Advances*, vol. 7, no. 29, 2021.

[20] M. Wehner, R. L. Truby, D. J. Fitzgerald, B. Mosadegh, G. M. Whitesides, J. A. Lewis, and R. J. Wood, "An integrated design and fabrication strategy for entirely soft, autonomous robots," *Nature*, vol. 536, no. 7617, pp. 451–455, 2016. [Online]. Available: http://dx.doi.org/10.1038/nature19100

[21] F. Renda, F. Boyer, J. Dias, and L. Seneviratne, "Discrete Cosserat Approach for Multisection Soft Manipulator Dynamics," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1518–1533, 2018.

[22] X. An, Y. Cui, H. Sun, Q. Shao, and H. Zhao, "Active-Cooling-in-the-Loop Controller Design and Implementation for an SMA-Driven Soft Robotic Tentacle," *IEEE Transactions on Robotics*, pp. 1–17, 2023.

[23] G. Vantomme, L. C. M. Elands, A. H. Gelebart, E. W. Meijer, A. Y. Pogromsky, H. Nijmeijer, and D. J. Broer, "Coupled liquid crystalline

oscillators in Huygens' synchrony," *Nature Materials*, vol. 20, pp. 1702–1706, 2021.
[24] J. Fras and K. Althoefer, "Soft Biomimetic Prosthetic Hand: Design, Manufacturing and Preliminary Examination," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–6.
[25] V. Falkenhahn, T. Mahl, A. Hildebrandt, R. Neumann, and O. Sawodny, "Dynamic Modeling of Bellows-Actuated Continuum Robots Using the Euler-Lagrange Formalism," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1483–1496, 2015.
[26] J. Tapia, E. Knoop, M. Mutný, M. A. Otaduy, and M. Bächer, "MakeSense: Automated sensor design for proprioceptive soft robots," *Soft Robotics*, vol. 7, no. 3, 2020.
[27] L. Teng, K. Jeronimo, T. Wei, M. P. Nemitz, G. Lyu, and A. A. Stokes, "Integrating soft sensor systems using conductive thread," *Journal of Micromechanics and Microengineering*, vol. 28, no. 5, p. 054001, 2018.
[28] Y.-L. Park, B.-R. Chen, and R. J. Wood, "Design and Fabrication of Soft Artificial Skin Using Embedded Microchannels and Liquid Conductors," *IEEE Sensors Journal*, vol. 12, no. 8, pp. 2711–2718, 2012.
[29] T. Baaij, M. K. Holkenborg, M. Stölzle, D. van der Tuin, J. Naaktgeboren, R. Babuška, and C. Della Santina, "Learning 3D shape proprioception for continuum soft robots with multiple magnetic sensors," *Soft Matter*, vol. 19, no. 1, pp. 44–56, 2023.
[30] E. H. Skorina, M. Luo, W. Tao, F. Chen, J. Fu, and C. D. Onal, "Adapting to Flexibility: Model Reference Adaptive Control of Soft Bending Actuators," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 964–970, 2017.
[31] K. C. Galloway, Y. Chen, E. Templeton, B. Rife, I. S. Godage, and E. J. Barth, "Fiber Optic Shape Sensing for Soft Robotics," *Soft Robotics*, vol. 6, no. 5, pp. 671–684, 2019.
[32] C. Armanini, F. Boyer, A. T. Mathew, C. Duriez, and F. Renda, "Soft Robots Modeling: A Structured Overview," *IEEE Transactions on Robotics*, pp. 1–21, 2023.
[33] 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.
[34] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. New York: John Wiley & Sons, 2006.
[35] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, 1994, vol. 29.
[36] E. Milana, F. Stella, B. Gorissen, D. Reynaerts, and C. Della Santina, "Model-Based Control Can Improve the Performance of Artificial Cilia," in *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2021, pp. 527–530.
[37] C. Della Santina and D. Rus, "Control oriented modeling of soft robots: The polynomial curvature case," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 290–298, 2020.
[38] M. Trumić, C. D. Santina, K. Jovanović, and A. Fagiolini, "Adaptive Control of Soft Robots Based on an Enhanced 3D Augmented Rigid Robot Matching," *IEEE Control Systems Letters*, vol. 5, no. 6, pp. 1934–1939, 2020.
[39] A. Kazemipour, O. Fischer, Y. Toshimitsu, K. W. Wong, and R. K. Katzschmann, "Adaptive Dynamic Sliding Mode Control of Soft Continuum Manipulators," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 3259–3265.
[40] M. Hofer, L. Spannagl, and R. D'Andrea, "Iterative Learning Control for Fast and Accurate Position Tracking with an Articulated Soft Robotic Arm," *ArXiv e-prints*, 2019.
[41] E. Franco and A. Garriga-Casanovas, "Energy-shaping control of soft continuum manipulators with in-plane disturbances," *International Journal of Robotics Research*, 2020.
[42] E. Franco and A. Astolfi, "Energy shaping control of underactuated mechanical systems with fluidic actuation," *International Journal of Robust and Nonlinear Control*, vol. 32, no. 18, pp. 10 011–10 028, 2022.
[43] H.-S. Chang, U. Halder, C.-H. Shih, N. Naughton, M. Gazzola, and P. G. Mehta, "Energy-shaping control of a muscular octopus arm moving in three dimensions," *Proceedings of the Royal Society A*, 2023.
[44] P. Borja, A. Dabiri, and C. D. Santina, "Energy-based shape regulation of soft robots with unactuated dynamics dominated by elasticity," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2022, pp. 396–402.
[45] P. Schegg, E. Ménager, E. Khairallah, D. Marchal, J. Dequidt, P. Preux, and C. Duriez, "SofaGym: An Open Platform for Reinforcement Learning Based on Soft Robot Simulations," *Soft Robotics*, vol. 0, no. 0, 2022.
[46] A. Tekinalp, S. H. Kim, T. Parthasarathy, and Y. Bhosale, "Pyelastica: A computational framework for cosserat rod assemblies," 2022.
[47] P. Hyatt, C. C. Johnson, and M. D. Killpack, "Model Reference Predictive Adaptive Control for Large-Scale Soft Robots," *Frontiers in Robotics and AI*, vol. 7, 2020.
[48] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using koopman operator theory," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 948–961, 2021.
[49] S. Kriegman, D. Blackiston, M. Levin, and J. Bongard, "A scalable pipeline for designing reconfigurable organisms," *Proceedings of the National Academy of Sciences*, vol. 117, no. 4, pp. 1853–1859, 2020.
[50] C. Duriez, "Control of elastic soft robots based on real-time finite element method." Proceedings - IEEE International Conference on Robotics and Automation, 2013, pp. 3982–3987.
[51] E. Coevoet, T. Morales-Bieze, F. Largilliere, Z. Zhang, M. Thieffry, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, J. Dequidt, and C. Duriez, "Software toolkit for modeling, simulation, and control of soft robots," *Advanced Robotics*, vol. 31, no. 22, pp. 1208–1224, 2017.
[52] N. H. Kim, *Introduction Analysis Finite Element to Nonlinear*. Springer, 2018.
[53] G. A. Holzapfel, *Nonlinear Solid Mechanics: A Continuum Approach for Engineering Science*. Heidelberg, Germany: Kluwer Academic Publishers, Jul. 2002, vol. 37, no. 4.
[54] O. Goury and C. Duriez, "Fast, Generic, and Reliable Control and Simulation of Soft Robots Using Model Order Reduction," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1565–1576, 2018.
[55] P. Benner, E. Sachs, and S. Volkwein, "Model order reduction for PDE constrained optimization," *International Series of Numerical Mathematics*, vol. 165, pp. 303–326, 2014.
[56] P. Astrid, S. Weiland, K. Willcox, and T. Backx, "Missing point estimation in models described by proper orthogonal decomposition," *IEEE Transactions on Automatic Control*, vol. 53, no. 10, pp. 2237–2251, 2008.
[57] M. Thieffry, A. Kruszewski, T.-M. Guerra, and C. Duriez, "Trajectory Tracking Control Design for Large-Scale Linear Dynamical Systems With Applications to Soft Robotics," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 2, pp. 556–566, 2019.
[58] K. Wu and G. Zheng, "FEM-Based Gain-Scheduling Control of a Soft Trunk Robot," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3081–3088, 2021.
[59] J. I. Alora, M. Cenedese, E. Schmerling, G. Haller, and M. Pavone, "Data-Driven Spectral Submanifold Reduction for Nonlinear Optimal Control of High-Dimensional Robots," *ArXiv e-prints*, 2022.
[60] S. Tonkens, J. Lorenzetti, and M. Pavone, "Soft Robot Optimal Control Via Reduced Order Finite Element Models," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 010–12 016.
[61] R. K. Katzschmann, M. Thieffry, O. Goury, A. Kruszewski, T.-M. Guerra, C. Duriez, and D. Rus, "Dynamically Closed-Loop Controlled Soft Robotic Arm using a Reduced Order Finite Element Model with State Observer," in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 2019, pp. 717–724.
[62] K. M. Moerman, "GIBBON: The Geometry and Image-Based Bioengineering add-On," *Journal of Open Source Software*, vol. 3, no. 22, p. 506, 2018.
[63] S. A. Maas, B. J. Ellis, G. A. Ateshian, and J. A. Weiss, "FEBio: finite elements for biomechanics," *Journal of Biomechanical Engineering*, vol. 134, no. 1, p. 011005., 2012.
[64] L. Smith, J. Haimes, and R. MacCurdy, "Stretching the Boundary: Shell Finite Elements for Pneumatic Soft Actuators," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2022, pp. 403–408.
[65] L. Smith, T. Hainsworth, J. Haimes, and R. MacCurdy, "Automated Synthesis of Bending Pneumatic Soft Actuators," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2022, pp. 358–363.
[66] V. G. Kamble, J. Mersch, M. Tahir, K. W. Stöckelhuber, A. Das, and S. Wießner, "Development of Liquid Diene Rubber Based Highly Deformable Interactive Fiber-Elastomer Composites," *Materials*, vol. 15, no. 1, p. 390, 2022.
[67] L. Smith and R. MacCurdy, "SoRoForge: End-to-End Soft Actuator Design," *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2023.
[68] J. Hiller and H. Lipson, "Dynamic Simulation of Soft Multimaterial 3D-Printed Objects," *Soft Robotics*, vol. 1, no. 1, pp. 88–101, 2014.

[69] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding," in *GECCO '13: Proceedings of the 15th annual conference on Genetic and evolutionary computation*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 167–174.

[70] J. S. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik, "Evolution Gym: A Large-Scale Benchmark for Evolving Soft Robots," *ArXiv e-prints*, 2022.

[71] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6265–6271.

[72] M. A. Graule, T. P. McCarthy, C. B. Teeple, J. Werfel, and R. J. Wood, "SoMoGym: A Toolkit for Developing and Evaluating Controllers and Reinforcement Learning Algorithms for Soft Robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4071–4078, 2022.

[73] C. Duriez, E. Coevoet, F. Largilliere, T. Morales-Bieze, Z. Zhang, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, and J. Dequidt, "Framework for online simulation of soft robots with optimization-based inverse model," in *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE, 2016, pp. 111–118.

[74] 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.

[75] A. T. Mathew, I. M. B. Hmida, C. Armanini, F. Boyer, and F. Renda, "SoRoSim: A MATLAB Toolbox for Hybrid Rigid-Soft Robots Based on the Geometric Variable-Strain Approach," *IEEE Robotics & Automation Magazine*, pp. 2–18, 2022.

[76] M. Gazzola, L. H. Dudte, A. G. McCormick, and L. Mahadevan, "Forward and inverse problems in the mechanics of soft filaments," *Royal Society Open Science*, vol. 5, no. 6, 2018.

[77] M. Stölzle and C. Della Santina, "Piston-Driven Pneumatically-Actuated Soft Robots: Modeling and Backstepping Control," *IEEE Control Syst. Lett.*, vol. 6, pp. 1837–1842, Dec. 2021.

[78] B. Caasenbrood, A. Pogromsky, and H. Nijmeijer, "Control-Oriented Models for Hyperelastic Soft Robots Through Differential Geometry of Curves," *Soft Robotics*, vol. 0, no. 0, 2022.

[79] Z. Wang, G. Wang, X. Chen, and N. M. Freris, "Dynamical Modeling and Control of Soft Robots with Non-constant Curvature Deformation," *ArXiv e-prints*, 2022.

[80] F. Boyer, V. Lebastard, F. Candelier, and F. Renda, "Dynamics of Continuum and Soft Robots: A Strain Parameterization Based Approach," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 847–863, 2021.

[81] F. Renda, C. Armanini, V. Lebastard, F. Candelier, and F. Boyer, "A Geometric Variable-Strain Approach for Static Modeling of Soft Manipulators with Tendon and Fluidic Actuation," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4006–4013, 2020.

[82] P. Pustina, P. Borja, C. D. Santina, and A. De Luca, "P-satI-D Shape Regulation of Soft Robots," *IEEE Robotics and Automation Letters*, pp. 1–8, 2022.

[83] O. Fischer, Y. Toshimitsu, A. Kazemipour, and R. K. Katzschmann, "Dynamic Task Space Control Enables Soft Manipulators to Perform Real-World Tasks," *Advanced Intelligent Systems*, 2022.

[84] S. M. H. Sadati, S. E. Naghibi, A. Shiva, B. Michael, L. Renson, M. Howard, C. D. Rucker, K. Althoefer, T. Nanayakkara, S. Zschaler, C. Bergeles, H. Hauser, and I. D. Walker, "TMTDyn: A Matlab package for modeling and control of hybrid rigid–continuum robots based on discretized lumped systems and reduced-order models," *International Journal on Robotics Research*, vol. 40, no. 1, pp. 296–347, 2020.

[85] A. T. Mathew, C. Armanini, A. A. S. A. Alshehhi, I. M. B. Hmida, and F. Renda, "Multifunctional Underwater Soft Robots: A Simulation Essay," *IOP Conference Series: Materials Science and Engineering*, vol. 1261, no. 1, p. 012008, 2022.

[86] 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.

[87] X. Zhang, F. Chan, T. Parthasarathy, and M. Gazzola, "Modeling and simulation of complex dynamic musculoskeletal architectures," *Nature Communications*, vol. 10, no. 1, pp. 1–12, 2019. [Online]. Available: https://doi.org/10.1038/s41467-019-12759-5

[88] H.-S. Chang, U. Halder, C.-H. Shih, N. Naughton, M. Gazzola, and P. G. Mehta, "Energy Shaping Control of a Muscular Octopus Arm Moving in Three Dimensions," *ArXiv e-prints*, 2022.

[89] T. Wang, U. Halder, E. Gribkova, R. Gillette, M. Gazzola, and P. G. Mehta, "A Sensory Feedback Control Law for Octopus Arm Movements," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 1059–1066.

[90] "mpm - File Exchange - MATLAB CentralFile Exchange - MATLAB Central," 2023, [Online; accessed 11. Sep. 2023]. [Online]. Available: https://nl.mathworks.com/matlabcentral/fileexchange/54548-mpm

[91] B. Proper, B. Caasenbrood, and I. Kuling, "Reliable cleaning of 3d sla/dlp printed soft fluidic actuators with complex internal geometry," in *6th IEEE International Conference on Soft Robotics, RoboSoft 2023*, 2023.

[92] B. Mosadegh, P. Polygerinos, C. Keplinger, S. Wennstedt, R. F. Shepherd, U. Gupta, J. Shim, K. Bertoldi, C. J. Walsh, and G. M. Whitesides, "Pneumatic networks for soft robotics that actuate rapidly," *Advanced Functional Materials*, vol. 24, no. 15, pp. 2163–2170, 2014.

[93] L. C. van Laake, J. de Vries, S. Malek Kani, and J. T. B. Overvelde, "A fluidic relaxation oscillator for reprogrammable sequential actuation in soft robots," *Matter*, vol. 5, no. 9, pp. 2898–2917, 2022.

[94] T. Reiner, G. Mückl, and C. Dachsbacher, "Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions," *Computers & Graphics*, vol. 35, no. 3, pp. 596–603, 2011.

[95] Z. Chen and H. Zhang, "Learning Implicit Fields for Generative Shape Modeling," *ArXiv e-prints*, 2018.

[96] J. Ortiz, A. Clegg, J. Dong, E. Sucar, D. Novotny, M. Zollhoefer, and M. Mukadam, "iSDF: Real-Time Neural Signed Distance Fields for Robot Perception," *ArXiv e-prints*, 2022.

[97] P. Liu, K. Zhang, D. Tateo, S. Jauhri, J. Peters, and G. Chalvatzaki, "Regularized Deep Signed Distance Fields for Reactive Motion Generation," *ArXiv e-prints*, 2022.

[98] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *ACM SIGGRAPH Computer Graphics*. New York, NY, USA: Association for Computing Machinery, 1987, vol. 21, no. 4, pp. 163–169.

[99] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes, "PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes," *Structural and Multidisciplinary Optimization*, vol. 45, no. 3, pp. 329–357, 2012.

[100] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[101] B. Smith, F. D. Goes, and T. Kim, "Stable neo-hookean flesh simulation," *ACM Trans. Graph.*, vol. 37, no. 2, mar 2018. [Online]. Available: https://doi.org/10.1145/3180491

[102] K. Bertoldi, M. C. Boyce, S. Deschanel, S. M. Prange, and T. Mullin, "Mechanics of deformation-triggered pattern transformations and superelastic behavior in periodic elastomeric structures," *Journal of the Mechanics and Physics of Solids*, vol. 56, no. 8, pp. 2642–2668, 2008.

[103] J. T. B. Overvelde, S. Shan, and K. Bertoldi, "Compaction Through Buckling in 2D Periodic, Soft and Porous Structures: Effect of Pore Shape," *Advanced Materials*, vol. 24, no. 17, pp. 2337–2342, 2012.

[104] N. M. Newmark, "A Method of Computation for Structural Dynamics," *Journal of the Engineering Mechanics Division*, vol. 85, no. 3, pp. 67–94, 1959.

[105] Y. Malitsky and K. Mishchenko, "Adaptive Gradient Descent without Descent," *ArXiv e-prints*, 2019.

[106] L. Marechal, P. Balland, L. Lindenroth, F. Petrou, C. Kontovounisios, and F. Bello, "Toward a Common Framework and Database of Materials for Soft Robotics," *Soft Robotics*, vol. 8, no. 3, pp. 284–297, 2021.

[107] R. F. Shepherd, F. Ilievski, W. Choi, S. A. Morin, A. A. Stokes, A. D. Mazzeo, X. Chen, M. Wang, and G. M. Whitesides, "Multigait soft robot," *Proceedings of the National Academy of Sciences*, vol. 108, no. 51, pp. 20 400–20 403, 2011.

[108] M. P. Bendsøe and O. Sigmund, *Topology optimization: theory, methods, and applications*, 2003, vol. 2nd Editio, no. 724.

[109] K. Svanberg, "The method of moving asymptotes—a new method for structural optimization," *International Journal for Numerical Methods in Engineering*, vol. 24, no. 2, pp. 359–373, 1987.

[110] ——, "Mma and gcmma-two methods for nonlinear optimization," *vol*, vol. 1, pp. 1–15, 2007.

[111] B. Caasenbrood, A. Pogromsky, and H. Nijmeijer, "A Computational Design Framework for Pressure-driven Soft Robots through Nonlinear Topology Optimization," in *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, pp. 633–638.

[112] F. Boyer, M. Porez, and A. Leroyer, "Poincaré–Cosserat Equations for the Lighthill Three-dimensional Large Amplitude Elongated Body Theory: Application to Robotics," *Journal of Nonlinear Science*, vol. 20, no. 1, pp. 47–79, 2010.

[113] J. C. Simo and L. Vu-Quoc, "A three-dimensional finite-strain rod model. part II: Computational aspects," *Computer Methods in Applied Mechanics and Engineering*, vol. 58, no. 1, pp. 79–116, 1986.

[114] J. Till, V. Aloi, and C. Rucker, "Real-time dynamics of soft and continuum robots based on Cosserat rod models," *International Journal of Robotics Research*, vol. 38, no. 6, pp. 723–746, 2019.

[115] G. S. Chirikjian and J. W. Burdick, "A modal approach to hyper-redundant manipulator kinematics," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 3, pp. 343–354, 1994.

[116] ——, "A Geometric Approach to Hyper-Redundant Manipulator Obstacle Avoidance," *Journal of Mechanical Design*, vol. 114, no. 4, pp. 580–585, 1992.

[117] B. Caasenbrood, A. Y. Pogromsky, and H. Nijmeijer, "Energy-based control for soft manipulators using cosserat-beam models," in *International Conference on Informatics in Control, Automation and Robotics - ICINCO*, 2021, pp. 311–319.

[118] C. Della Santina, R. K. Katzschmann, A. Bicchi, and D. Rus, "Model-based dynamic feedback control of a planar soft robot: trajectory tracking and interaction with the environment," *International Journal of Robotics Research*, vol. 39, no. 4, pp. 490–513, 2020.

[119] H. Li, L. Xun, and G. Zheng, "Piecewise Linear Strain Cosserat Model for Soft Slender Manipulator," *IEEE Transactions on Robotics*, pp. 1–18, 2023.

[120] C. Della Santina, "The Soft Inverted Pendulum with Affine Curvature," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 4135–4142.

[121] F. Stella, Q. Guan, C. D. 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, pp. 03–07.

[122] G. S. Chirikjian, "Theory and applications of hyper-redundant robotic manipulators," Ph.D. dissertation, 1992. [Online]. Available: https://www.researchgate.net/publication/41482361_Theory_and_applications_of_hyper-redundant_robotic_manipulators

[123] A. L. Orekhov and N. Simaan, "Solving Cosserat Rod Models via Collocation and the Magnus Expansion," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2020–24.

[124] F. Bullo and R. M. Murray, "Proportional Derivative (PD) Control on the Euclidean Group," *California Institute of Technology*, 1995. [Online]. Available: https://authors.library.caltech.edu/28018

[125] V. Sonneville, A. Cardona, and O. Brüls, "Geometrically exact beam finite element formulated on the special Euclidean group SE(3)," *Computer Methods in Applied Mechanics and Engineering*, vol. 268, pp. 451–474, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.cma.2013.10.008

[126] A. Loría, "Observers are Unnecessary for Output-Feedback Control of Lagrangian Systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 905–920, 2015.

[127] 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.

[128] J. Illingworth and J. Kittler, "The Adaptive Hough Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 690–698, 1987.

[129] K. Suzumori and F. Saito, "Micro rubber structure realizing multi-legged passive walking," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 445–450.

[130] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, "Soft Robotics for Chemists," *Angewandte Chemie International Edition*, vol. 50, no. 8, pp. 1890–1895, 2011.

[131] M. Raeisinezhad, N. Pagliocca, B. Koohbor, and M. Trkov, "Design Optimization of a Pneumatic Soft Robotic Actuator Using Model-Based Optimization and Deep Reinforcement Learning," *Frontiers in Robotics and AI*, vol. 8, p. 639102., 2021.

BRANDON J. CAASENBROOD received his MSc degree from the Eindhoven University of Technology (TU/e) in 2017. Following his graduation, he undertook a role as a junior researcher and educator at the same institution within the Dynamics and Control group. He commenced his doctoral studies at the start of 2018, supervised by prof.dr. Henk Nijmeijer and dr. Alexander Pogromsky. His research was embedded within the Wearable Robotics program, which is funded by the Dutch Research Council (NWO). His work focused on the soft robotics domain, with a particular emphasis on the model-based control and design optimization of soft robotic systems. In 2024, he successfully obtained doctoral degrees with the distinction of cum laude.

ALEXANDER Y. PORGROMSKY received the MSc degree (with honors) from the Baltic State Technical University, Russia in 1991 and his PhD from the St.Petersburg Electrotechnical University in 1994. From 1995 till 1997 he worked at the Laboratory 'Control of Complex Systems' (Institute for Problems of Mechanical Engineering, St. Petersburg, Russia). From 1997 to 1998 he was a research fellow with the Department of Electrical Engineering, division on Automatic Control, Linkoping University, Sweden. He has been a visiting researcher at the Department of Control Systems and Industrial Robotics, Saint-Petersburg National Research University of Information Technologies Mechanics and Optics.

HENK NIJMEIJER is emeritus Full Professor at Eindhoven University of Technology (TU/e) and Chair of the Dynamics and Control group. Henk's research focuses on Control Systems Engineering, Mechanical Engineering and Automotive Engineering. His areas of expertise include (advanced) control theory and systems, robotics, mechatronics, (system) dynamics and control systems engineering. The current project is 'Extremum seeking control in large-scale information systems. Henk is also Program Leader of TU/e research program 'Integrated Cooperative Automated Vehicles' (i-CAVE). This focuses on development of Autonomous vehicles that can drive independently on non-public roads and drive together on public roads.