Journal Title
XX(X):1–40

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

TMTDyn: A Matlab Package for Modeling and Control of Hybrid Rigid-Continuum Robots Based on Discretized Lumped System and Reduced Order Models

S.M.Hadi Sadati[1,2], S. Elnaz Naghibi[3], Ali Shiva[4,5], Brendan Michael[4], Ludovic Renson[1], Matthew Howard[4], Caleb D. Rucker[6], Kaspar Althoefer[3], Thrishantha Nanayakkara[5], Steffen Zschaler[4], Christos Bergeles[2], Helmut Hauser[1], and Ian D. Walker[7]

Abstract

A reliable, accurate, and yet simple dynamic model is important to analyze, design and control hybrid rigid-continuum robots. Such models should be fast, as simple as possible and user-friendly to be widely accepted by the ever-growing robotics research community. In this study, we introduce two new modeling methods for continuum manipulators: a general reduced-order model (ROM) and a discretized model with absolute states and Euler-Bernoulli beam segments (EBA). Additionally, a new formulation is presented for a recently introduced discretized model based on Euler-Bernoulli beam segments and relative states (EBR). We implement these models to a Matlab software package, named $TMTDyn$, to develop a modeling tool for hybrid rigid-continuum systems. The package features a new High level Language (HLL) text-based interface, a CAD-file import module, automatic formation of the system EOM for different modeling and control tasks, implementing Matlab C-mex functionality for improved performance, and modules for static and linear modal analysis of a hybrid system. The underlying theory and software package are validated for modeling experimental results for (i) dynamics of a STIFF-FLOP continuum appendage, and (ii) general deformation of a fabric sleeve worn by a rigid link pendulum. A comparison shows higher simulation accuracy (8-14% normalized error) and numerical robustness of the ROM model, and computational efficiency of the EBA model with near real-time performances that makes it suitable for large systems. The challenges and necessary modules to further automate the design and analysis of hybrid systems with a large number of states are briefly discussed in the end.

Keywords

TMT Lagrange dynamics, Hybrid mechanisms, Continuum robots, Cosserat rod, Tissue, Fabric, Software, High Level Language

Nomenclature

Acronyms

BVP	Boundary Value Problem
CC	Constant Curvature (Kinematics)
COM	Center of Mass
EOM	Equation of Motion
FEM	Finite Element Method
MK	Main Kinematic Chain
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PVW	Principle of Virtual Work
ROM	Reduced-order Model
TMT	TMT Method for Driving EOM
VC	Variable Curvature (Kinematics)

Sets & Constants

$\mathbb{N}$	Natural Numbers
$\mathbb{R}$	Real Numbers
$\mathbb{Z}$	Integer Numbers
g	Gravitational Constant, 9.81 [m/s^2]

Operators

$*$	Rotating a Vector with a Quaternion
Δ	Arithmetic Difference

[1]Department of Engineering Mathematics, University of Bristol, bristol, UK
[2]Department of Imaging and Biomedical Engineering, Kings College London, London, U.K.
[3]Department of Engineering and Materials Science, Queen Mary University of London, London, UK
[4]Department of Informatics, Kings College London, London, UK
[5]Dyson School of Engineering, Imperial College London, London, UK
[6]University of Tennessee, Knoxville, TN, USA
[7]Clemson University, Clemson, SC, USA

Corresponding author:
S.M.Hadi Sadati, Department of of Engineering Mathematics, University of Bristol, Bristol, BS8 1TH, U.K.
Email: smh_sadati@kcl.ac.uk

$\star$	Translation/Quaternion Pair Rotation
$\times$	Cross Product for Vectors & Mapping from $\mathbb{R}^i$ to $\mathbb{SO}(i)$ for Quaternions
$a:b$	Vector of $[a, a+1, ..., b-1, b]$
e	Lie transformation Base

Subscripts & Superscripts

$'$	Spatial Differentiation
$,$	Partial Differentiation
0	Initial/Boundary Value
$^{-1}$	Inverse for a Matrix & Conjugate for a Quaternion
$\bar{}$	Terms in state space, $\bar{X} = T_x^\top X T_x$
$^=$	Linearized terms at a given point
$\dot{}$	Temporal Differentiation
$\hat{}$	Unit Vector
ω	Terms in modal space
com	COM Variables
$\top$	Matrix/Vector Transpose
$^+$	Updated values in numerical integration mid-step function
a	Cross-sectional Area Variables
b	Euler-Bernoulli Beam Variables
c	Constraint Variables
d	Mesh Variables
h	DOF Variables
i	General Purpose Numerator
j	Joint Variables
l	Load Variables
m	Bodies' Variables
p	Actuation Pressure
r	ROM Variables
r	Rotated Vector
t	Geometrical Transformation Variables
u	Input Variables
v	Viscous Damping Variables

Other Symbols

α	Local Bending/Twist Angle
$\bar{d}$	EOM gravitational & inertial velocity dependent terms
$\bar{M}$	$T^\top M T$ mass matrix in state space
χ	Position/Orientation Vector
δ	Differential Variation
ϵ	Deformed Local Position/Translation Vector
η	Modal Damping Ratio
Γ	4×4 Transformation Matrix
$\hat{d}_i$	Local Curvilinear Frame Directors with $\hat{d}_3$ Tangent to The Backbone
$\hat{i}, \hat{j}, \hat{k}$	Reference Cartesian Frame Unit Vectors for x, y, z-axes
λ	Lagrange Multiplier
μ	Viscous Damping Coefficient
ν	Viscous Damping Power Law
Ω	Angular Velocity
ω	Modal undamped natural frequency
Φ	Mode Shape Vector
ϕ	Rotation Angle in an Axis-Angle Rotation
ψ	Pneumatic Actuators Location Polar Angle
ρ	Position Vector
σ	Material Density
τ	Moment
θ	Absolute Bending/Twist Angle
Ξ	A Translation/Quaternion Pair
ξ	Local Strain Vector
ζ	Local Curvatures/Torsion Vector
a	Cross-section Area
C	ROM Coefficient Matrix
D	Coefficient Matrix of d
d	Velocity Dependent Terms in 2nd time derivatives
E	Elastic Modulus
f	Action Force Vector
G	Shear Modulus
I	Mass 2nd Moment of Inertia
J_a	Cross-section 2nd Moment of Area
K	Linearized Elastic Stiffness Matrix
k	Elastic Stiffness
l	Length
M	Mass Matrix
m	Body Mass or ROM Mass per Unit Length
n	Number of Elements
o, O	Pneumatic Actuator Location Variables
Q	Rotation Quaternion $[Q_0, Q_\rho]$
q	State Vector
Q_0	Rotation Quaternion Scalar
Q_ρ	Rotation Quaternion Vector

R	Rotation Matrix
r	Radius
S	ROM Shape Function Vector
s	Length Variable Along a Rod Backbone
T	Transformation Jacobian
V	Linearized Viscous Damping Matrix
w	Action in PWV Method
$\mathbf{I}$	Unit Matrix

Introduction

Mimicking highly dexterous and deformable biological bodies has been a trending topic of multi-disciplinary research, called soft robotics, using intrinsically soft materials in the form of continuum robotic platforms Rus and Tolley (2015). Performing delicate tasks Cianchetti et al. (2014), high maneuverability in unstructured and confined environments Burgner-Kahrs et al. (2015); Cianchetti and Menciassi (2017); Walker et al. (2016), dexterous grasping Katzschmann et al. (2015), mimicking biological tissue and organs He et al. (2018), bioinspired dynamic locomotion Wehner et al. (2016) such as crawling Rich et al. (2018), terrestrial Godage et al. (2012) or submerged locomotion Cianchetti et al. (2015) are among the promises made by the researchers in the field. Soft robots are appealing to investigate new design and theoretical concepts such as variable stiffness structures McEvoy and Correll (2018), morphological computation Nakajima et al. (2018) and embodied intelligence Nakajima et al. (2015), to simplify the control and sensing tasks through robot embodiment Fchslin et al. (2012); Thuruthel et al. (2018a).

However, compliance has disadvantages, such as uncertain deformations, limited control feedback, reduced control bandwidth, stability issues, underdamped modes, and lack of precision in tasks involving working against external loads Blanc et al. (2017); Cianchetti et al. (2013). These usually result in modeling and/or control challenges for such designs. Besides, now more than anytime, unified frameworks are needed to transfer our well-established knowledge of dynamic system analysis, path planning and control design for rigid-body robots to soft robotic research Kapadia et al. (2010); Renda and Seneviratne (2018); Renda et al. (2018); Della Santina et al. (2018b) and to model hybrid rigid-soft body systems Sadati et al. (2018c); Patern et al. (2018). Such frameworks should be as simple as possible and easy to use, to be widely accepted by the ever-growing soft robotics research community with researchers from different disciplines and backgrounds. It should provide fast performance, to be suitable for control and design problem of soft systems with large state spaces. Also, it needs to be integrable with standard software platforms, e.g. C/C++ language, Matlab software, ROS (Robotic Operation System, see ros.org), etc., widely used in the community.

Here, we introduce two new modeling approaches for continuum rods and actuators, a general reduced-order model (ROM), and a discretized model with absolute states and Euler-Bernoulli beam segments (EBA). These models enable us to perform more accurate simulation of continuum manipulators, as well as modeling 2D and 3D continuum geometries, something which has been missing in similar recent research Renda et al. (2018). In addition, a new formulation is presented for a recently introduced discretized model by Renda et al. (2018); Shiva et al. (2018) which is based on Euler-Bernoulli beam theory and relative states (EBR). These models are implemented in a Matlab software package, that we name $TMTDyn$, to establish a new modeling and simulation tool for hybrid rigid-continuum body systems. The package is improved with a new High level Language (HLL) text-based interface, a CAD-file import module, automatic formation of the system EOM for different modeling and control tasks, implementing C-mex functionality for improved performance, and other modules for static and linear modal analysis of a hybrid system. Our main goal is to make deriving EOM of hybrid rigid-continuum body robots, performing dynamic system analysis, state observation, and control system design tasks more accessible to the interdisciplinary soft robotics research community and people with limited expertise in dynamic system modeling. The rest of this section is dedicated to a brief review of the different elements of this research.

High Level and Domain Specific Languages

ROS, Orocos (Open Robot Control Software, see orocos.org), SmartSoft (see smart-robotics.sourceforge.net), OpenRTM (see openrtm.org), and Matlab Robotics System Toolbox (see mathworks.com/products/robotics) are some robotic software platforms developed to make robotics programming and configuration as accessible as possible to experts from different application domains. Domain-Specific Languages (DSLs), also known as High Level Languages (HLL), and Model-driven Engineering (MDE) are interesting emerging areas in the robotics research community, e.g. distributed robotics, system control, and vision, with significant potential in facilitating the programming of future robots. A DSL is a dedicated programming language for a particular problem domain, offering specific abstractions and notations, to decrease the coding complexity and increase programmer productivity. DSLs have been used for programming complex systems, such as robots, control systems, etc, for which traditional general-purpose languages do not provide a good correlation between the implementation requirements and language features. To address this, DSLs are powerful and systematic ways to provide two main features; (i) quick and precise adaptation by domain experts, who are not familiar with general purpose programming languages; (ii) hiding the architecture complexity by software engineers to facilitate complex configuration and design architectures before transferring to domain experts.

However, none of the aforementioned robotics software platforms are fully compatible with soft structure robots, which need modules for handling highly articulated geometries with repeated elements, discretization or employing reduced order model assumptions to simplify the modeling and control state space for a continuum geometry. Moreover, it is not always straightforward to access the derived Equations of Motion (EOM) which is necessary for off-the-shelf dynamic system analysis, design optimization, and control system design tasks. On the other hand, most recent efforts for publishing modeling packages for soft robots

Coevoet et al. (2017); Gazzola et al. (2018); Renda and Seneviratne (2018); Hu et al. (2018) were more focused on the feasibility of their modeling and control approach, cannot be extended easily due to not providing classical EOM for the system (mostly because of using differential equations and Finite Element Method (FEM) solvers, e.g. SOFA (Simulation Open Framework Architecture, see www.sofa-framework.org) package Coevoet et al. (2017) and ChainQueen Hu et al. (2018)), and almost none of them provide an easy-to-use HLL to reach researchers with less expertise in mechanics of continuum structures. As a part of this research, we address these challenges by developing a Matlab based modeling package for modeling and control of hybrid robotic systems with minimal modeling and control states. To this end, the key challenge is to integrate continuum mechanics with traditional rigid body dynamics, which is widely being used in robot control research. To address this issue, we start with a brief review of the modeling methods for mechanical systems and, later, continuum elements.

Mechanical System Dynamics

Deriving EOM for modeling hybrid rigid-continuum body mechanisms has been a challenge in soft robotic research Boyer (2014); Rus and Tolley (2015); Burgner-Kahrs et al. (2015); Sadati et al. (2017b). The Newton-Euler method Jung et al. (2011), Lagrange dynamics Mustaza et al. (2018); Godage et al. (2016), the Principle of Virtual Work (PVW) Trivedi et al. (2008); Sadati et al. (2016, 2017a), TMT methods for deriving the Lagrange dynamics Sadati et al. (2018b) and Bond Graph approaches Sutar and Pathak (2017) have been used to derive EOM of such systems. Here, we disregard the Cosserat rod dynamics Till and Rucker (2017), FEM Coevoet et al. (2017) and the Moving Least Squares Material Point Method (MLS-MPM) Hu et al. (2018), methods that result in a system of differential equations which are not inherently compatible with control design methods for rigid-body systems. Based on the Newton-Euler method, the EOM of a system can be derived by knowing the kinematic relations and dynamic interactions between the system components. This is a straight forward step-by-step derivation method that results in a large number of modeling states, including the internal forces. The Euler-Lagrange and Principle of Virtual Work methods benefit from a limited number of independent states (generalized coordinates) for which methods of controller design are investigated intensively. Most commercially available dynamical system modeling software, e.g. MSC. ADAMS Negrut and Dyer (2004), utilize Lagrange dynamics formulation. Complex and hard-to-interpret final sets of equations and complexity in dealing with nonlinear elastic and damping elements are the drawbacks of this method. Hence, usually, an extra step is needed to collect the final EOM in a close form vector formalism. The TMT method is based on parts of Lagrange's investigations on analytical mechanics, dealing with generalized coordinates, virtual work and inertial forces, which was published in 1788 and before his well-known "Lagrange method" Wisse and Linde (2007). L. Schwab and M. Wisse have named this method "TMT" because of the final form of the system inertial matrix (M) in the generalized coordinates space ($T^\top MT$), where T is the Jacobian transformation matrix between the Cartesian and generalized coordinates spaces and superscript ($^\top$) is the matrix transpose operator. As a result, higher order derivatives are eliminated in the derivation process and the EOM are directly derived in a closed vector formalism. The derived terms are independent, hence suitable for parallel numerical evaluation, and the final form of EOM is easy-to-interpret and suitable for controller design. It is easy to incorporate constraints, to derive inverse maps and to change the system EOM if the constraints or structural properties are changed without deriving the Lagrangian again. We have recently developed a Matlab software package, called $AutoTMTDyn$ to derive TMT EOM of rigid-body mechanisms Sadati et al. (2018d, 2015).* $AutoMTDyn$ was originally developed for deriving the TMT EOM of rigid-body systems Sadati et al. (2015) and used for analyzing free-fall righting maneuvers of a robot cat Sadati and Meghdari (2017), lumped system modeling of continuum appendage Sadati et al. (2017b), and dynamic analysis of a spider web structure Sadati et al. (2018a). In this paper, we introduce a new version of $AutoTMTDyn$, now called $TMTDyn$, which is equipped with a new High level Language (HLL) text-based interface, CAD-file import module, automatic formation of the system EOM for different modeling and control tasks, implementing Matlab C-mex functionality for improved performance, and modules for static and linear modal analysis of a hybrid system. We employed the TMT method to derive EOM of continuum bodies based on discretized and reduced order solutions. As a result, a unified software package is sought after for deriving EOM, control design, and numerical simulation of hybrid rigid-continuum body systems. To this end, a brief review is provided on different modeling elements and assumptions for continuum rods.

Mechanics of Continuum Structures

If taking a theoretical approach in constrast to pure learning Braganza et al. (2007); Thuruthel et al. (2018b), combined reduced-order solution and learning Thieffry et al. (2018a) approaches, and byond the distinction between 2D vs. 3D and static vs. dynamic models, two key stages can be identified which determine the modeling strategy of a soft robot Burgner-Kahrs et al. (2015); Gazzola et al. (2018); Sadati et al. (2017b) (Table 1). (1) Modeling assumptions for (1-I) the system kinematics, (1-II) system conservation law (system mechanics), and (1-III) material constitutional law (material mechanics). This stage results in a system of differential equations for the mechanics of a continuum media. (2) The method to solve this resulting system, which can be based on (2-I) direct or (2-II) indirect methods. Here, we focus on modeling methods for 1-dimensional (1D) continuum elements, (continuum rods), as the most studied continuum structure in soft robotic research. Later we explain how to generalize models for continuum rods to 2-dimensional (2D) (membrane) and 3-dimensional (3D) continuum structures.

Table 1. Different elements of a model for continuum rods.

1- Modeling Assumptions
I- Kinematics
i- Continuous Geometry
a- Variable Curvature (VC- finite strain assumption)
b- Truncated Series Shape Assumption
c- Constant Curvature (CC)
ii- Discretized Geometry
a- Series Rigid-Body Kinematics
b- Screw Theory
c- Forward Discretization of VC
II- Mechanics (Conservation Law)
i- Cossert Rod Method
ii- Principle of Virtual Work (PVW)
iii- Beam Theory
iv- Lagrange Dynamics
III- Material Mechanics (Constitutional Law)
i- Linear Elasticity (Hooke's Law)
ii- Finite Strain Theory (Neo-Hookean, Money-Rivlin,... Gent, etc)
iii- Visco-Hyperviscoelastic
2- Solution
I- Direct Methods
i- Analytical Integration
ii- Numerical Forward Integration Steps
II- Indirect methods
i- Optimization Based Methods
ii- Finite Element Methods (FEM)
iii- Reduced Order Method (Ritz Method)
iv- Combination of the above

Mechanics of Continuum Rods

Two methods are widely used to describe continuum rod kinematics; (1-I-i) continuous and (1-I-ii) discretized Kinematics. (1-I-i-a) Variable Curvature (VC- finite strains assumption) Trivedi et al. (2008), (1-I-i-b) truncated series shape functions Godage et al. (2011, 2016); Sadati et al. (2018b), and (1-I-i-c) Constant Curvature (CC- as a subset of general shape function approach), which is probably the most simple and widely used assumption for soft manipulator modeling Webster and Jones (2010) report instances of employing continuous kinematics. (1-I-ii-a) Employing series rigid-body kinematics, by simplifying a continuum rod as a hyper-redundant mechanism with finite but large enough number of segments, based on transformation matrices for consecutive but distinct rotational and translational joints Della Santina et al. (2018a); Shiva et al. (2018), or methods based on (1-I-ii-b) Screw Theory Renda et al. (2017); Renda and Seneviratne (2018); Renda et al. (2018) and (1-I-ii-c) forward discretization of VC differential equations Shiva et al. (2018), where a skew-symmetric matrix of local curvatures/torsion vector is used to describe the local relative rotations along the backbone, are instances of using discretized representation of such system kinematics. (1-II-i) The Cosserat Rod method Trivedi et al. (2008); Burgner-Kahrs et al. (2015), (1-II-ii) Principle of Virtual Work (PVW) Sadati et al. (2017a), (1-II-iii) Beam Theory Sadati et al. (2017a), and (1-II-iv) Lagrange Dynamics Godage et al. (2016); Sadati et al. (2018b); Della Santina et al. (2018a) are used to derive the system governing equation (conversational law). The material constitutional law (material mechanics) is usually derived based on (1-III-i) linear elasticity theory (Hookes law) Trivedi et al. (2008); Godage et al. (2016); Sadati et al. (2017b), (1-III-ii) finite strain theory (considering large strain in hyperelastic materials, such as Neo-Hookean Trivedi et al. (2008); Sadati et al. (2017a,b); Shiva et al. (2018), Mooney-Rivlin, Gent Shiva et al. (2018), etc), or (1-III-iii) by considering hyper-viscoelastic properties Sadati et al. (2018b); Mustaza et al. (2018). Any combination of the above choices results in a system of Ordinary (ODE) or Partial Differential Equations (PDE) to be solved numerically based on the system initial and boundary conditions. Using shape functions or discretized kinematics results in PDEs with decoupled spatial and time domains where direct solutions based on (2-I-i) analytical, if possible, or (2-I-ii) numerical forward integration steps in spatial and time domain can be used to solve the resulting initial value problems Godage et al. (2016); Sadati et al. (2017a, 2018b). Such systems turn into a Boundary Value Problem (BVP), if static solutions are sought, where forward integration is valid when distributed loads, e.g.body weight, are neglected Shiva et al. (2018). Alternatively, indirect solutions can be sought. (2-II-i) Optimization-based methods, i.e. single shooting Sadati et al. (2017a,b, 2018b), multiple shooting, and concatenation methods, are suitable for BVPs resulting from static models with general loads, or for learning the coefficients of an approximate series solution Godage et al. (2011) or gains in a neural network model Thuruthel et al. (2018b). (2-II-ii) Finite Element Methods (FEM) or similar segmentation methods Cianchetti and Menciassi (2017); Bieze et al. (2018) are suitable if spatial (kinematics) discretization methods are used where, instead of a forward integration over the spatial domain, a system of nonlinear equations is formed with a large but sparse coefficient matrix. The system equilibrium point in static cases or at every time step of a dynamic simulation is found by calculating the pseudo-inverse of the coefficient matrix, while satisfying all the geometrical, dynamical and optimal control constraints Cianchetti and Menciassi (2017). While considering truncated series solutions as the system kinematics (2-II-iii) reduced order Thieffry et al. (2018b) or Ritz Sadati et al. (2018b) method for solving a PDE problem, different choices of weighting functions can be used to improve the accuracy of the solution, e.g. in the case of Ritz-Galerkin methods Tunay (2013); Sadati et al. (2018b). Finally, (2-II-iv) a combination of the above methods can be used, usually for solving PDEs resulting from complex geometries. As an example of general practice in many commercially available FEM solvers, Tunay used a discrete Galerkin method, where weighted governing equations are used to construct the FEM solution for pneumatic actuators with general deformation Tunay (2013). Sadati et al. used forward integration on spatial domain for the PDEs resulting from employing reduced order solutions for continuum manipulator kinematics, and then combined that with single shooting optimization method to find the system static solution under excessive external tip loads Sadati et al. (2017a, 2018b). Bieze et al. combined FEM and optimization methods to solve the closed-loop control problem of continuum manipulators Bieze et al. (2018).

* Available at https://github.com/hadisdt/AutoTMTDyn

Gazzola et al. combined FEM with forward integration on time-domain in dynamic simulations Gazzola et al. (2018). Thieffry et al. constructed a reduced order model based on dominant deformation modes that are found from multiple FEM based simulations of a system under different loading conditions. The coefficients of such solution were then optimized to solve for general cases Thieffry et al. (2018b). Duriez, Bieze and Thieffry used a SOFA FEM modeling package for real-time dynamic simulation of soft structures Cianchetti and Menciassi (2017); Bieze et al. (2018); Thieffry et al. (2018b).

In a comparative study with experimental results with a single module STIFF-FLOP appendage Sadati et al. (2017b), we have recently shown the advantage of:

- a lumped system approach (1-I-ii-a & 1-II-iv & 2-I-i) for dynamic analysis and traditional control design
- CC and modified CC (1-I-i-c & 1-II-ii & 2-I-i) for considering structural complexity and design parameter study
- Cosserat rod theory (1-I-i-a & i-II-i & 2-II-i) for accuracy in general cases
- reduced order series solutions (1-I-i-b & 2-II-i) for real-time performance, all based on 1-III-i if needed.

We show that combining reduced order kinematics, the Cosserat rod mechanics, numerical integration on spatial domain and optimization based solution (1-I-i-b & 1-II-i & 2-I-ii & 2-II-i) produces most of the aforementioned advantages, i.e. accuracy, simple control design, real-time performance, considering structural complexity for a single STIFF-FLOP appendage in planar motion with excessive external load at the tip Sadati et al. (2018b). Further, we generalize our solution for multi-segment arms in general 3D dynamic motion and compare the accuracy and numerical performance of the results with models with other assumptions. Additionally, the discretization method presented by Renda et al. Renda et al. (2018), which is based on Screw Theory and transformation matrices, is modified to use absolute (independent) states to achieve discretized models for multi-dimensional continuum geometries with a large number of states and significantly improved numerical efficiency.

Aims and Objectives

In the following sections, first, we discuss rigid system kinematics and how they can be unified with variable curvature and discretized continuum kinematics, with relative and absolute (independent) states. A new general yet efficient reduced order solution for the rod backbone is discussed based on truncated series. Then, hybrid system dynamics is discussed where a lumped-system representation of the Cosserat rod theory and a new discretization method based on absolute (independent) states is presented. As a result, the relations for one-dimensional continuum elements can be generalized to model two-dimensional (membrane or fabric) and three-dimensional (tissue block) geometries. Subsequently, we discuss how these modeling methods are implemented in the newly developed $TMTDyn$ package with new functionalities and modules. The software package and experimental results from a (STIFFness controllable Flexible and Learn-able manipulator for surgical OPerations) continuum appendage Fras et al. (2015) are used to investigate the computational performance and simulation accuracy of the discussed continuum rod models. Finally, the package is tested for dynamic modeling a system with a 2D continuum geometry, a rigid link pendulum with a fabric sleeve on, and compared with experimental results. A discussion and conclusion are presented as the final section of this paper.

Hybrid Systems Mechanics

A dynamic system with inertial, compliant and constraining elements can be assumed as a set of lumped (point) masses, usually assumed at the system elements' Center of Mass (COM) locations, with moments of inertia which are connected with spring/dampers and joints to the adjutant lumped masses. For a continuum system, where usually a system of differential equations describes the system mechanics, a differential format of the lumped-system approach can be employed. To this end, first, the free body diagram of the load balance in a single differential element is drawn, then the lumped-system equivalence of the system is assumed where the parameters are differential terms. In the following sections, first the rigid- and continuum-body system kinematics using quaternions and the reduced-order method of using truncated polynomial series are described. Then, the TMT dynamics of such systems are discussed. Finally, we explain how to derive the lumped-system equivalence of Cosserat rod and reduced-order methods with relative and absolute (independent) states.

Hybrid System Kinematics

Rigid Body Kinematics- System kinematics describes the geometric relations between the system elements in terms of rotation and translation. As a result, equations describing the position vector and orientation of each point of the system (usually only COMs and joint axis) are derived. The rotations/orientations can be described in terms of rotation matrices $R_{[3\times3]}$, quaternions $Q_{[1\times4]}$ or screw theory and Lie group notations, and the translation is a vector ρ_t. Rotation matrices and Screw Theory result in the same final set of equations, despite their differences in notation. Quaternion representation of rotations, implemented in the $TMTDyn$ package, results in fewer states in the system EOM (3 for a quaternion rotation and 6 for a rotation matrix) compared to screw transformations and angle-axis representation of rotations. This method, only if all the four quaternion elements are considered as the system states, does not have inherent singular points associated with the transformation matrices and screw transformations. Also, Matlab is shown to be faster in optimizing the derived equation when quaternions are used. However, quaternion arithmetic requires more numerical operations, as described below.

Unit quaternion representation of a rotation ϕ around a unit vector $\hat{\rho}_{[1\times3]}$ is $Q = [Q_0, Q_\rho]$, where $Q_0 = \cos(\phi/2)$ and $Q_\rho = \sin(\phi/2)\hat{\rho}$, hence, basic rotation around frame x-, y-, or z-axis is simply setting $\hat{\rho}$ to $\hat{i}$, $\hat{j}$, or $\hat{k}$ respectively. Alternatively, if a general 3D rotation/orientation is

described with the three elements of Q_ρ, we have $Q_0 = \sqrt{1 - Q_\rho Q_\rho^\top}$. The method of defining a rotation with only three parameters does not suffer from singularity around 0 [deg] (which is associated with the singularity of CC and series rigid-link kinematics of continuum rods at straight configuration). Note that $Q_0 = \sqrt{1 - Q_\rho Q_\rho^\top}$ is also an equally valid scalar part. Following any of the above formulations, by definition, can never produce a quaternion with both positive and negative scalar part and really employ the whole space of quaternions, rather only the ones with positive or scalar parts. As a result, this method will suffer from inherent representational singularities, just like Euler angles do but for rotations approaching ± 180 [deg] about any axis. One should use such method only for rotations $\in (-\pi, pi)$, not inclusive of $\pm\pi$. An alternative is defining a rotation with all the four quaternion elements Tunay (2013); Till and Rucker (2017); Rucker (2018) which is not considered here due to the extra needed state.

Consecutive local rotations are handled by simple right-hand multiplication $R_{1:n} = \Pi_{i=1}^{n} R_i$, or matrix multiplication of quaternions as $Q_{1:n} = Q_{1:n-1} \times Q_n$, where $Q\times$ is equivalent to a matrix product as

$$Q\times = \begin{bmatrix} Q_0 & -Q_{\rho_1} & -Q_{\rho_2} & -Q_{\rho_3} \\ Q_{\rho_1} & Q_0 & -Q_{\rho_3} & Q_{\rho_2} \\ Q_{\rho_2} & Q_{\rho_3} & Q_0 & -Q_{\rho_1} \\ Q_{\rho_3} & -Q_{\rho_2} & Q_{\rho_1} & Q_0 \end{bmatrix}, \qquad (1)$$

and $1:n$ means considering all the instances with index 1 to n. For rotating/transforming a vector ρ to ρ_r with quaternions, we have $\rho_r = R\rho$ with rotation matrices and $[0, \rho_r] = \left[[Q]_\times [0, \rho]\right]_\times \text{conj}(Q)$, where $\text{conj}(Q) = [Q_0, -Q_\rho]$ is the definition of quaternion inversion ($Q^{-1} = \text{conj}(Q)$). We present the above quaternion manipulation by $\rho_r = Q * \rho$ in the rest of this text for simplicity. A translation (ρ_t) followed by a rotation (R or Q) can be represented by a 4×4 transformation matrix $\Gamma = \{R, \rho_t\}$ as $\Gamma(1:3, 1:4) = [R, \rho_t], \Gamma(4, 1:4) = [0, 0, 0, 1]$, where consecutive transformations are handled by simple multiplication of Γ as $\Gamma = \Pi_{i=1}^{n} \Gamma_i$, and a vector transformation by $\rho_r = \Gamma\rho$. We use a similar quaternion transformation pair as $\Xi = \{Q, \rho_t\}$, where consecutive transformations are handled by $\Xi_{1:n} = \{Q_{1:n}, \rho_{t_{1:n}}\}$, in which $\rho_{t_{1:n}} = \rho_{t_{1:n-1}} + Q_{1:n} * \rho_{t_n}$ and $Q_{1:n} = Q_{1:n-1} \times Q_n$. We present the above quaternion pair transformation as $\Xi_{1:n} = \Xi_{1:n-1} \star \Xi_n$ in the rest of this text for simplicity. All the vectors in the rest of this paper are w.r.t. and expressed in the system reference frame, unless stated otherwise.

Each joint can be described as a set of transformations (Γ_j or Ξ_j) and the system kinematics represents the joint axis transformations w.r.t. and expressed in the system reference (inertial) frame. Then, the linear velocity of the body COM, to which the joint (j) is immediately connected in the system main kinematic chain, is $\dot{\rho}_m = T_{\rho_m} \dot{q}$, where $T_{\rho_m} = \rho_{m,q}$ is the Jacobian of the transformation that maps the link COM position vector between the Cartesian and the system state spaces, $\rho_m = \Xi_j * \rho_{\text{com}}$ is the COM position vector, q is the system state (DOF) vector, ρ_{com} is the COM local vector (w.r.t. and expressed in local frame), superscripts ($\dot{}$) and ($\ddot{}$) are for the first and second temporal derivatives, and subscript ($_,$) represents partial derivatives as $X_{,x} = \partial X / \partial x$.

The COM rotational velocity Ω_m is derived w.r.t. the reference frame but expressed in the link local frame, since it is easier to calculate the link second moment of inertial w.r.t. the link local frame. We have, $\Omega_m = T_{Q_j} \dot{q}$, where $T_{Q_j} = (2Q_j^{-1} \times Q_{j,q})_{[2:4,1:n_q]}$ is the Jacobian of the transformation that maps the link COM orientation quaternion to the system state space, and n_q is the number of system states (DOFs). Finally, the link (rigid-body) inertial matrix (M) and the Jacobian of the transformation map to the system state space (T_m) become

$$M = \begin{bmatrix} m\mathbf{I}_{[3\times3]} & 0 \\ 0 & J_m \end{bmatrix}, \; T_m = \begin{bmatrix} T_{\rho_m} & 0 \\ 0 & T_{Q_j} \end{bmatrix}, \qquad (2)$$

where m is the body mass, J_m is the body second moment of inertia $[3 \times 3]$ matrix, and $\mathbf{I}_{[n\times n]}$ is an $[n \times n]$ unity matrix.

Continuum Rod VC Kinematics- We use 1-dimensional (1D) continuum elements, i.e. continuum rods, as the basis of modeling continuum geometries in this work. A higher dimensional geometry (e.g. a mesh geometry) can be constructed based on lumped masses (e.g. nodes of a mesh) interconnected by these 1D elements (e.g. edges of the mesh). Using the Cosserat rod method, which considers all six transnational (strains- ξ) and rotational (curvatures/torsion- ζ) differential states, is beneficial for 1D continuum elements. It is not the most efficient method for higher dimensional geometries, where the strain field is sufficient to calculate the distortions too; however, it is well suited for our main purpose here, which is deriving easy to interpret EOM of hybrid systems suitable for system dynamics analysis and controller design. Such models are not aimed at exact geometrical analysis and design, for which FEM methods are more suited. They rather are built upon simplifying assumptions for improved performance with the aim of investigating the large state space of such systems to clarify their underlying dynamics and/or control opportunities/challenges. We have recently used Variable Curvature (VC), based on rotation matrices, and Beam theory to investigate continuum manipulator mechanics in static and dynamic motions Sadati et al. (2017b,a, 2018b) and for a hybrid rigid-continuum body system proposing of a highly-articulated inter-locking interface for stiffness control of a continuum appendage Sadati et al. (2018c,a). We showed that employing a beam theory approach simplifies solving the BVP for static analysis of a continuum rod with numerical optimization-based or reduced order model methods. However, it is not a good candidate as a part of a unified framework for modeling hybrid systems and especially in a lumped system approach framework. VC kinematics and the Cosserat rod theory are used to model the static mechanics of continuum rods, based on rotation matrices Gazzola et al. (2018) and quaternion Trivedi et al. (2008); Burgner-Kahrs et al. (2015) representation of rotations.

Here, the equations are derived in a local physical curvilinear coordinates $[\hat{d}_1,\ \hat{d}_2,\ \hat{d}_3]$, where s is the variable along the backbone, $\hat{d}_3$ is tangent to the backbone, and at the rod base we have $[\hat{d}_1,\ \hat{d}_2,\ \hat{d}_3](s=0) = [\hat{i},\ \hat{j},\ \hat{k}]$ (Fig. 1.a).

The rod backbone curve spatial configuration (ρ_s) and 1×4 rotation quaternion unit vector (Q_s), expressed in inertial Cartesian coordinates ($[\hat{i},\ \hat{j},\ \hat{k}]$), are derived according to VC

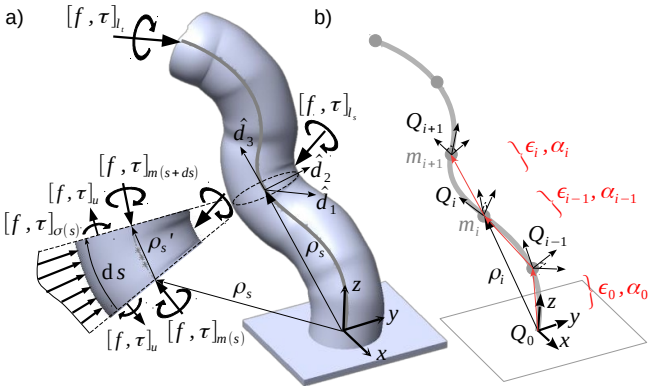


Figure 1. a) Variable curvature kinematics and free body diagram of Cosserat rod method for one differential element along the continuum backbone. Subscript ($_l$), ($_u$), and ($_\sigma$) are for the external point loads at the along the backbone, loads due to internal pressures and/or tendons' tension (inputs) if any, and distributed loads, e.g. due to gravity, respectively. $[f,\tau]_s$ are the structural internal loads induced in response to the aforementioned loads. Note the direction of the load vectors. b) Discretized VC kinematics and lumped mass representation of Cosserat rod method for the same continuum rod as in (a). Lumped mass representation of Cosserat rod method is equivalent to assuming a concentrated mass at the base of each differential/discretized element along the rod backbone.

as in Rucker (2018); Trivedi et al. (2008)

$$Q'_s = Q_s \times [0, \zeta_s]/2,\ \rho'_s = Q_s * (\xi_s + [0,0,1]), \qquad (3)$$

where superscript ($'$) is used spatial differentiation as $x' = \partial x/\partial s$. The simple implementation of quaternion rotation is more computationally expensive than using rotation matrices R_s as $R'_s = R[\zeta_s]_\times$ and $\rho'_s = R(\xi_s + [0,0,1])$, where $[\ ]_\times$ denotes the standard mapping from $\mathbb{R}^3$ to $\sim\!\rtimes(3)$ Burgner-Kahrs et al. (2015). However, quaternions are reported to be better in terms of numerical integration accuracy, and preserving frame orthogonality and vector length Trivedi et al. (2008); Rucker (2018).

Discretized VC Kinematics- Discretized versions of Eq. 3 with rotation matrices are discussed in Takano et al. (2017); Renda and Seneviratne (2018); Renda et al. (2018); Shiva et al. (2018). Shiva et al. used first order discretiztion, $R_{i+1} = R_i(\zeta_i \Delta s + \mathbf{I}_{[3\times3]})$ (i is the element numerator,), which is probably the simplest assumption, failing to conserve the principal properties of a rotation matrix. Renda et al. used the same method in the context of screw theory as $R_{i+1} = R_i e^{\zeta_i \Delta s}$, where $e^{\zeta \Delta s} = \zeta \Delta s + \mathbf{I}_{[3\times3]}$ Renda and Seneviratne (2018); Renda et al. (2018). Takano et al. used the most accurate representation for ζ with Euler angles (three consecutive rotations around local frame principle unit vectors) with 1-2-3 $(x-y-z)$ order ($R_{xyz_\zeta} = R_{x_{\zeta_1}} R_{y_{\zeta_2}} R_{z_{\zeta_3}}$), as $R_{i+1} = R_i R_{xyz_{\zeta_i}}$. A similar representation is discussed in Shiva et al. (2018), appendix section, arguing that the order of the rotations is not important as long as small enough elements are considered along with the backbone (infinitesimal curvatures/torsion). They showed that using any of the above methods does not affect the accuracy of modeling a short appendage with beam theory, even for large deformations. Here we use the first representation used by Shiva et al. and Renda et al., since it

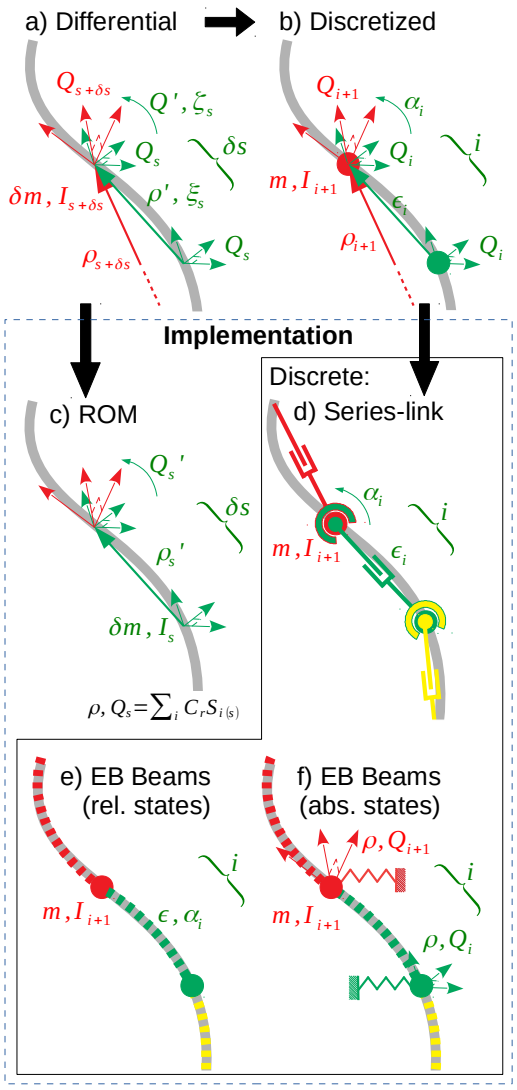


Figure 2. Different modeling assumptions for a continuum rod. a) Rod differential Variable Kinematics (VC), b) discretized VC framework, c) Reduced Order Model (ROM) based on a polynomial series solution for the backbone kinematics, d) equivalent highly-articulated series rigid link mechanism, e) discretized model with relative states from Euler-Bernoulli (EB) beams, f) discretized model with absolute states (w.r.t. reference frame) and EB beam compliant connections. In the case of absolute states, we may assume that segments are connected to the ground with zero stiffness elastic elements and to each other with EB beams. Backbone is shown by a continuous gray line and dashed curves are EB beam sections.

is easy to interpret its inverse as $\zeta_i = R_i^\top (R_{i+1} - R_i)/\Delta s$, which is necessary for modeling a continuum rod with absolute (independent) modeling states (a new contribution of this paper). Using quaternions and their properties, the final form of the discretized equations are

$$\begin{aligned} Q_{i+1} &= Q_i \times [1, \alpha_i/2], \\ \rho_{i+1} &= Q_i * \epsilon_i + \rho_i, \end{aligned} \qquad (4)$$

for VC kinematics and

$$\begin{aligned} [0, \alpha_i] &= 2Q_i^{-1} \times (Q_{i+1} - Q_i), \\ \epsilon_i &= Q_i^{-1} * (\rho_{i+1} - \rho_i), \end{aligned} \qquad (5)$$

for their inverse. Here, $\alpha = \zeta \Delta s$ is the local bending/twist angle vector, and $\epsilon = \Delta\rho = (\xi + [0,0,1])\Delta s$ is the deformed local position/translation vector. Notice that Q_i is the absolute orientation quaternion at each point, but α describes the relative orientation of consecutive elements. The above equations show that the deformation of a discretized element can be modeled as a quaternion transformation pair $\Xi = \{Q_\alpha, \epsilon\}$, which is a 3D translational joint with state space ϵ and initial value ϵ_0, followed by a 3D rotational joint with state α, initial value α_0 and quaternion representation of $Q_\alpha = [1, \alpha/2]$ (Fig. 1.b). The same can be said if using R as $\Gamma = \{R_{xyz_\alpha}, \epsilon\}$. Having an element's initial bending/twist angle (α_0) and a local translation vector (ϵ_0), the local deformation of the discretized geometry ($\Delta\alpha = \alpha - \alpha_0$ and $\Delta\epsilon = \epsilon - \epsilon_0$) can be calculated for deriving the element viscoelastic mechanical action due to system deformation.

Reduced Order Kinematics- Reduced Order Models (ROM) for continuum rod kinematics is discussed by Godage et al. Godage et al. (2011, 2016) based on the pressure chambers' length for a pneumatic soft manipulator. The presented solution is hard to interpret, results in complicated dynamic derivations, and the mechanical coupling between the actuation chambers' input pressure and length are not considered. A large number of coefficients that should be learned through experimental trials is another drawback of such a method. In our previous work, we showed the advantageous numerical performance and accuracy of using a truncated Lagrange polynomial series passing through some arbitrary points along the backbone Sadati et al. (2018b). The proposed solution is easy to interpret for shape estimation and controller design, since the used polynomial is constructed using Cartesian coordinates of physical points, and has small number of states (6 for a short appendage consisting of Cartesian coordinates of 2 points at the appendage tip and mid-length). Both methods solve the singularity problem of using Constant Curvature and rotation matrix representations. However, we used the CC assumption to compensate for the imaginary torsion of a FrenetSerret frame and to find the physical torsion of the appendage cross-section based on the input chambers' pressure. In addition, the cross-section shear was neglected and a mean axial strain is assumed along the backbone. The kinematics was combined with Beam theory for static modeling and PVW for dynamic modeling, using Ritz and Ritz-Galerkin solutions. Duriez, Bieze and Thieffry have recently generalized the same concept to modeling and control of complex continuum geometries by extracting the dominant deformation maps using the SOFA FEM package Cianchetti and Menciassi (2017); Bieze et al. (2018); Thieffry et al. (2018b).

Here, we drop using FrenetSerret frames and present a new general ROM approach to account for the cross-section local strains, as well as dealing with curvatures/torsion without any secondary assumptions, e.g. CC. Additionally, a simple polynomial is used instead of a Lagrange polynomial that results in simpler and faster derivation of the system kinematics. An inverse linear problem is solved to find the initial value of the polynomial coefficients based on the position and orientation of some nodes along the rod backbone. The final solution is more suitable for Cosserat rod and PVW methods. We assume that the manipulator geometry is defined by 6 truncated polynomial series of order $n_r + 1$ for position and n_r for orientation map, as

$$[\rho, Q_\rho] = \sum_{i=1}^{n_r} (C_r S_i) + S_0, \tag{6}$$

where Q_ρ is the vector part of the quaternion representation of backbone orientation (Q) with $Q_0 = \sqrt{1 - Q_\rho Q_\rho^\top}$, n_r is the polynomial order, $C_{r[6\times n_r]}$ is the polynomial coefficient matrix which is considered as the system modeling states, $S_i = [s, s, s, 1, 1, 1]s^i$ and $S_0 = [0, 0, s, 0, 0, 0]$ are the shape function matrices that satisfies the rod base boundary conditions, i.e. being perpendicular to the base. Defining $S = S_{1:n_r}$, we can rewrite the above equation as $[\rho, Q_\rho] = C_r S + S_0$. The state (coefficient matrix) initial values (C_{r0}) are found based on position (ρ_0) and orientation ($Q_{\rho 0}$) of a few points along the manipulator backbone (s_r) by solving the following inverse problem, $C_{r0} = ([\rho_0, Q_{\rho 0}] - S_{0_{s_0}})S_{s_0}^{-1}$. The above inverse problem can be solved efficiently using Matlab **inv** function. ρ_0 and $Q_{\rho 0}$ can be simply measured from experimental observations using magnetic or visual trackers. For the local strain ξ and curvatures/torsion ζ, from Eq. 3 and similar to the inverse map in Eq. 4, we obtain

$$[0, \zeta] = 2Q^{-1} \times Q', \ \xi = Q^{-1} * \rho' - [0, 0, 1]. \tag{7}$$

xi and ζ are used to calculate the mechanical action of the rod structural compliance. Alternatively, the system geometry can be described using only four polynomials, three for ρ and one for twist angle ($\theta_{\hat{1}}$), where the bending angles are found by compensating the FrenetSerret frames non-physical orientation using $\theta_{\hat{1}}$. Such a solution results in a system with a smaller number of states, but more complex equations to handle, and inherent singular points.

Hybrid System Dynamics

Dynamic modeling and controller design for hybrid systems are practiced, by having a kinematic model with a finite number of states. The discussed discretization method and reduced order model are the key elements in modeling such systems. We use the TMT method for deriving vector formalism of inertial terms in Lagrange EOM. The TMT method benefits from a smaller number of steps and simpler procedures compared to deriving EOM based on a system Lagrangian. A separate set of equations are derived for each body in the system that speeds up the derivation and optimization of derived equations, as well as providing the possibility of implementing parallel numerical simulation methods for large systems. However, the Principle of Virtual Work is used to derive the actions for the system springs, dampers, and input forces, due to the flexibility and simplicity of this approach in handling different element types.

TMT Method for EOM Inertial Terms- From Newton's second laws of motion for the inertial terms (related to M) and gravitational forces (related to g) of a rigid body EOM in general free motion, we have $M\ddot{\chi} = f_\chi$, where $\chi = [\rho, Q_\rho]$, M as in Eq. 2, f_χ are the forces in Cartesian

coordinates, e.g. body weight $f_\chi = f_g = M[g,0,0,0]$, and $g = [0,0,9.81]$ [m/s^2] is the gravity vector. From mapping the system states from Cartesian coordinates to state space coordinates using the Jacobian transformation matrix, we have $T_m = \chi_{,q}$, where q is the vector of system states), and hence $\dot{\chi} = T_m\dot{q}$, $\ddot{\chi} = T_m\ddot{q} + (T_m\dot{q})_{,q}\dot{q}$. Combining the above relations and transforming the equation to the state space (using $T_m^\top$) we obtain, $T_m^\top M T_m \ddot{q} = T_m^\top(-MD_m\dot{q} + f_g)$, where $D_m = (T_m\dot{q})_{,q}$. While the above relation is valid for a large dynamic system, deriving the necessary vectors (M, T_m, D_m, f_g) for individual bodies in a system provides flexibility in dealing with bodies of different type. So we have

$$\begin{aligned} \sum_{i=1}^{n_m} \bar{M}_i\, \ddot{q} &= \sum_{i=1}^{n_m} \bar{d}_i, \\ \bar{M}_i &= T_{m_i}^\top M_i T_{m_i}, \\ \bar{d}_i &= T_{m_i}^\top(-M_i D_{m_i}\dot{q} + f_{g_i}), \end{aligned} \tag{8}$$

where $T_{m_i[6\times n_q]}$, $M_{m_i[6\times 6]}$, $D_{m_i[6\times n_q]}$, $f_{g_i[6\times 1]}$, i is a general numerator for the body number in a system with n_m bodies, $q_{[n_q\times 1]} = q_{1:n_m}$ is the vector of all the system states, and $n_q = \Sigma_{i=1}^{n_m} n_{q_i}$ is the number of states in the system. Note that the Jacobian matrices are calculated w.r.t. q (all the system states) and not q_i. Eq. 8 can be solved for $\ddot{q}$ to form an ODE problem, and then integrated over time using a numerical integration method, e.g. 4$^{\text{th}}$-order RungeKutta method implemented in Matlab software $ode15s$ or $ode113$ function.

Springs, Dampers, External & Input Loads- Following PVW, the mechanical action of such elements in the system state space (q) is $w\delta q = fT_f\delta q$, where f is the force vector exerted by the element, and $T_f = \chi_{f,q}$ is the Jacobian that transforms the loads exerting location/orientation from Cartesian space to the system states space. The term $w^\top = T_f^\top f$ will be added to the right side of Eq. 8 as

$$\sum_{i=1}^{n_m} \bar{M}_i\, \ddot{q} = \sum_{i=1}^{n_m} \bar{d}_i - \sum_i w_i^\top, \tag{9}$$

where $T_{f_i[n_{f_i}\times n_q]}$, $f_{j[n_{f_i}\times 1]}$, $w_{i[n_q\times 1]}$ represents other mechanical actions in the system due to viscoelastic elements, external, internal, and body forces, etc., and n_{f_i} is the dimension of vector f_i.

For external loads (f_l) with action point ρ_l, we have $T_l = \rho_{f,q}$. If the elements are acting directly on the system states, i.e. elements parallel to the DOFs, $T_q = q$ is for the acting point, f_{l_q} is the DOF direct input, $f_{k_q} = k_q(q - q_{0_k})$ is the force of a parallel spring, q_{0_k} is the resting value of the spring, and $f_{v_q} = \mu_q\dot{q}^\nu$ is the force of a parallel viscous damper with power ν. For simple axial (1D) elements, connecting two points of the system (ρ_{f_1}&ρ_{f_2}), we have $T_a = \bar{\rho}_{f,q}$, where $\bar{\rho}_f = \Delta\rho_f = \rho_{f_1} - \rho_{f_2}$ is the line of action, $f_l = |f_l|\hat{\bar{\rho}}_f$ is the external force with value $|f_l|$ along the element unit direction vector where $\hat{\bar{\rho}} = \bar{\rho}_f/\sqrt{\bar{\rho}_f\bar{\rho}_f^\top}$, $f_k = k\bar{\rho}_f(1 - l_{0_k}/\sqrt{\bar{\rho}_f\bar{\rho}_f^\top})$ is the spring force vector, l_{0_k} is the spring resting length, and $f_v = \mu\dot{\bar{\rho}}_f^\nu$ is the viscous damping force vector. Note that the order of $\rho_{a|b}$ is not important in calculating $\bar{\rho}_f$, as long as it remains consistent. Finally, by monitoring the sign of the deformation of the element $\Delta l_k = \sqrt{\bar{\rho}_f\bar{\rho}_f^\top} - l_{0_k}$, tension only ($\Delta l_k > 0$, e.g. rope) and compression only ($\Delta l_k < 0$, e.g. impact) elements can be modeled. The same procedure can be adopted for continuum elements, e.g. an Euler-Bernoulli beam or reduced order model of a deforming beam, which is discussed in the following sections.

Continuum Body Dynamics- Eq. 9 can be easily adapted for a discretized continuum rod using the lumped mass method. The differential form of TMT terms are

$$\bar{M}_i = \int_0^{l_i} \mathrm{d}\bar{M}_i\mathrm{d}s,\ \bar{d}_i = \int_0^{l_i} \mathrm{d}\bar{d}_i\mathrm{d}s,\ w_i^\top = \int_0^{l_i} \mathrm{d}w_i^\top\mathrm{d}s, \tag{10}$$

where l_i is the length of the i^{th} ROM element. For each ROM element in the system, the above spatial integrals can be handled with a numerical forward integration method, e.g. trapezoidal rule implemented in Matlab software **trapz** function, in each integrating time step of Eq. 8. The Cosserat rod method is used to derive the TMT method differential terms in Eq. 10 for ROM elements. To this end, we start with the differential form and then differential lumped mass representation for Cosserat rod model.

Cosserat Rod Mechanics- The Cosserat rod theory presents the conservation law to balance the material local internal loads (f_s, τ_s) due to local external (f_l, τ_l- external loads at the rod tip or body), internal (f_u, τ_u- due to internal actuation pressure or tendon tension) and body loads (f_g, τ_g- due to body weight or other uniform loads, e.g. a magnetic field). Different methods of deriving Cosserat rod mechanics are presented in the literature, based on distributed load balance in an infinitesimal element Trivedi et al. (2008), and differentiation of the shear force balance on a long segment (applying variational calculus) Rucker et al. (2010); Burgner-Kahrs et al. (2015); Renda et al. (2018). Governing equations for the above approaches can be derived using the Principle of Virtual Work too Grazioso et al. (2018). Here, we follow Trivedi et al. (2008) by considering the distributed load balance for free body diagram of a single differential element along the rod backbone (Fig. 1.a). For the load balance, expressed in the reference frame, we have

$$\begin{aligned} (f_s - Q_s * f_u)' + f_g &= \sigma a_s\ddot{\rho}_s, \\ (\tau_s - Q_s * \tau_u)' + \rho_s' \times f_s + \tau_g &= J_{m_s}\dot{\Omega}_s, \end{aligned} \tag{11}$$

where ρ_s and Q_s are as in Eq. 3, J_{m_s} is the cross section second moment of inertia, $f_g = \sigma a_s g$, σ is the material density, and a_s is the rod cross-section area. Note that the effect of f_s, τ_u are considered in the force ($(Q_s * f_u)'$) and moment ($(Q_s * \tau_u)'$) balance equations after being transformed to the reference frame. External local loads f_{l_s}, τ_{l_s} are local shear loads considered as local boundary conditions

$$\begin{aligned} f_s - Q_s * f_u + f_g\Delta s + f_{l_s} &= 0, \\ \tau_s - Q_s * \tau_u + \Delta\rho_s \times f_s + \tau_g\Delta s + \tau_{l_s} &= 0, \end{aligned} \tag{12}$$

where Δ represents the variation between the integration boundaries. Knowing all the boundary conditions $(\xi, \zeta, f_s, \tau_s)$ at a point and all the external contact forces along the backbone f_l, τ_l, Eq. 11 can be integrated between

the external contact points and free ends, using Eq. 12 to update f_s, τ_s at each external contact point (between two consecutive spatial integration steps). In a discretized or reduced order modeling framework, Eq. 11 and 12 are handled alongside after direct (ROM) or indirect (discretization methods) spacial integration of Eq. 11. Hooke's law (linear stress-strain relation), expressed in reference frame, is the usual choice for the system constitutional law as $f_s = Q_s * (k_\xi \xi_s)$ and $\tau_s = Q_s * (k_\zeta \zeta_s)$, where k_ξ and k_ζ are diagonal stiffness matrices based on the rod material.

The above formulation results in a BVP problem which is hard to integrate into a unified modeling framework for hybrid systems. In the next sections, three methods for dynamic modeling of a continuum rod are discussed, using the discretized (Eq. 4 & 5) and reduced order model (Eq. 6 & 7) kinematics, discussed earlier, and using TMT differential terms (Eq. 10), and PVW for compliance elements and loads explained above.

Discretized Continuum Dynamics with Relative States- Discretizing Eq. 11, a highly articulated system with length l, n_d elements, and relative states ($q = [\epsilon, \alpha]$) is formed with the kinematic relation expressed in Eq. 4. M and T_m are found by substituting ρ_i and Q_i from Eq. 4 into Eq. 2, to find the TMT inertial terms as in Eq. 8. This forms the right hand of Eq. 11. The external loads are handled based on their exerting point, found from Eq. 4. In such systems, beam elasticity and damping, and the internal pressure/tendon tension acts parallel to the states q, so we set $k|\mu_q = [k|\mu_\epsilon, k|\mu_\alpha]$ and $f_{l_q} = [f_u, \tau_u]$, and follow the relevant procedure for compliance elements and loads explained above. These form the left-hand side of Eq. 11. Finally, the above terms are used alongside other terms in Eq. 9. The proposed procedure is easy to implement; however, the derived equations tend to be complex after having less than ten elements, which results in long segments, resulting in inaccurate results, slow derivation and simulation Sadati et al. (2018d); Takano et al. (2017). The method is not suitable for large system models.

Discretized Continuum Dynamics with Absolute States- To avoid complex derivations for a high number of elements, we may assume the discretized system states to be the lumped-masses' Cartesian positions and take the vector part of their unit quaternion orientation as $q = [\rho, Q_\rho]$. Q_0 is derived based on Q_ρ to form a unit quaternion. The system kinematics is the same as q and increasing the number of elements does not increase the complexity of the derivations. M and T_m are found by substituting ρ and Q into Eq. 2, to find the TMT inertial terms as in Eq. 10. The external loads become loads directly acting on system states $f_{l_q} = f_l$. The inverse map presented in Eq. 5 is used to derive ϵ_i and α_i, based on which the beam elasticity (w_k) and damping (w_v), and the internal pressure/tendon tension (w_u) actions are calculated as

$$\begin{aligned} \chi_b &= [\epsilon_i, \alpha_i],\ T_b = \chi_{b,q}, \\ w_k^\top &= T_b^\top k_{\epsilon|\alpha}(\chi_b - \chi_{b_0}), \\ w_v^\top &= T_b^\top \mu_{\epsilon|\alpha}(T_b \dot{q})^\nu, \\ w_u^\top &= T_b^\top [f_u, \tau_u], \end{aligned} \tag{13}$$

where χ_{b_0} is the beam initial position vector and bending/twist angle that can be fund based on the system states' initial condition q_0 as $\chi_{b_0} = \chi_b(q_0)$. The above terms are used alongside other terms in Eq. 9. The proposed method allows handling a large number of elements. To the best of our knowledge, this is the first time that such a discretization method is used for modeling an actuated continuum manipulator, as well as its integration to a unified hybrid system modeling framework. However, similar discretization methods are widely being used to solve hyperbolic PDEs, e.g. Eq. 11, numerically. Meeting Courant−Friedrichs−Lewy condition is necessary for converging the solution which usually results in systems with very large number of elements and hence slow performance Skeel and Berzins (1990). We do not analyze the convergence criteria in this draft, but a comparison with experimental results and the other presented methods in this draft are provided later.

Reduced Order Model Dynamics- In the case of reduced order model kinematics, the system spatial and temporal domains are decoupled. So we keep the differential form of Eq. 11, and perform a forward numerical integration over the ROM terms in each time step of the final system EOM numerical temporal integration. Here the states are the elements of C_r in Eq. 6 which gives $6 \times n_r$ states. The system kinematics is presented in Eq. 6 as $\chi_s = [\rho_s, Q_{\rho_s}]$, and T_m is found by substituting ρ_s and Q_{ρ_s} into Eq. 2. for $\mathrm{d}M$ we have, $\mathrm{d}m = \sigma a \mathrm{d}s$ and J_{m_s} is found based on the second moment of inertia for planar objects with the shape of the rod cross section. TMT differential terms are found from Eq. 10. The contact point kinematics of an external load at location s_l along the backbone is found by substituting $s = s_l$ in Eq. 6. Using the inverse map in Eq. 7 to find ξ, ζ, the differential form of Eq. 13 is used to find the action derivatives for viscoelastic structure and internal pressures/tendon tensions. This method does not suffer from discretization inaccuracy; however, the modeling accuracy depends on the order of the polynomial, while a higher number of terms does not necessarily improve the accuracy. Initial bent configurations, rods with initial arbitrary geometries are easy to handle, by choosing appropriate values for χ_{s_0}.

Intermediate Numerical Step for Higher Order Nonlinear Terms- High order nonlinear terms due to soft structure hyper viscoelastic behavior can be handled by considering nonlinear stiffness and damping coefficients. Such assumptions do not change the method of deriving such elements' actions as for compliance elements and loads explained before. However, the value of the nonlinear coefficients should be updated during the numerical simulation. An intermediate numerical integration step is introduced to update the nonlinear coefficients based on the system current states. We have used this approach in our earlier work to account for the braid constraint of pneumatic actuators in continuum manipulators and the material hyper-elastic deformation Sadati et al. (2018b); Shiva et al. (2018). This intermediate step can be used for event handling and saturation constraints.

Table 2. TMTDyn package structure. Underlined file names are provided or edited by the user (continued in Table 3).

1. *System Declaration*
- **User Interface** (*"system.m"*)
  - System Symbolic Parameters: $par.sym$
  - HLL System Declaration
    - Bodies: $body$
      - Lumped Mass
      - Mesh Body- connected similar masses
      - 1D VC Beam- reduced order model
    - 1D Connections: $joint$
      - Axial Spring/Damper
      - Input Actuator
      - Constraints
      - Flexible Beams
        - Euler-Bernoulli Beam
        - VC Beam- reduced order model
    - External Loads: $exload$
- **Parameters Check** (*"check.m"*)
  - Assign Default Values & Error Check
- **Mesh Import** (*"import.m"*)
  - Assign Masses & Links to a CAD File

Table 3. ...continued from Table 2.

2. *EOM Derivation*
- **TMT Derivation of EOM** (*"tmt_eom_derive.m"*)
  - Joints' kinematics
    - Forming state space: q,f_{k_q},f_{v_q},f_{u_q}
    - Local transformations: Q_l,Ξ_l
  - System Bodies: $mass$
    - Abs. Q,Ξ of main kinematic joints
    - Derive EOM matrices: M,T_m,D_m,f_{g_m},ρ_m
  - System Connections
    - Spring/Damper/Input: $sprdmp$
      - Abs. Q,Ξ for both joints' ends
      - Derive $T_j^\top$,f_{k_j},f_{v_j},f_{u_j},K_j,V_j,ρ_j
    - Constraints: $cnst$
      - Derive $T_c^\top$,D_c,u_c
  - External Loads
    - Abs. Q,Ξ for point of exerting load
    - Derive $T_l^\top$,f_l

3. *Derivation Optimization & Forming EOM*
- **Derivation Optimization & Storage**
  - Optimized Matlab & C functions (*"save_func.m"*)
  - C-mex file generation (*"save_mex.m"*)
- **EOM Formation & Optimization**
  - Mid-step in num. integration (*"int_mid_step.m"*)
  - Matlab & C-mex EOM func. (*"save_eom_mex.m"*)
- **Modal Analysis Terms Formation & Optimization**
  - Matlab & C-mex EOM func. (*"save_modal_mex.m"*)

4. *System Numerical Simulation*
- **Simulation Parameter:** $par.var$ (*"system.m"*)
- **Static Equilibrium:** $\ddot{q}$=$\dot{q}$=0 (*"equil.m"*)
- **Modal Analysis** (*"modal.m"*)
- **Dynamic Simulation** (*"dyn_sim.m"*)

5. *Post-process*
- **Simple Simulation Animation** (*"anim.m"*)
- **Post-process-** provided by the user (*"post_proc.m"*)

Linear Modal Analysis- A linearized version of Eq. 9, without time-varying external forces, at a given point q_0 is

$$\sum_{i_m} \bar{\bar{M}}_{i_m} \ddot{q} + \sum_{i_v} \bar{\bar{V}}_{i_v} \dot{q} + \sum_{i_k} \bar{\bar{K}}_{i_k} q + A_0 = 0, \qquad (14)$$

where $\bar{\bar{M}} = \bar{M}_{,q(q_0)}$, $\bar{\bar{V}} = w_{v,q(q_0)}$, and $\bar{\bar{K}} = w_{k,q(q_0)}$ are linearized matrix coefficients for inertia, stiffness, and viscous damping respectively, and A_0 is a constant term due to linearizion of $\bar{d}$, gravitational forces, springs resting values, etc.. A_0 does not contribute to the system modal analysis. Eq. 14 can be used for linear modal analysis of a system without damping or a proportionally damped system, subject to proper choice of damping coefficients ($\bar{\bar{V}} = A_1\bar{\bar{M}} + A_2\bar{\bar{K}}$) and $\nu = 1$, where A_i is a general constant. The eigenvalue problem for Eq. 14 can be solved with the Matlab **eig** function to find the system undamped natural frequencies (ω) and matrix of mode shapes (Φ) as $[\Phi_\omega, \omega] = \text{eig}(-\bar{\bar{M}}^{-1}\bar{\bar{K}})$. Then the system modal damping ratio is $\eta_\omega = (\omega M_\omega)^{-1} V_\omega$, where $M_\omega = \Phi_\omega^\top \bar{\bar{M}} \Phi_\omega$ and $V_\omega = \Phi_\omega^\top \bar{\bar{V}} \Phi_\omega$ are the linearized inertial and viscous damping matrices in modal space.

Constraints, Controller and Observation Design- Constraints can be modeled as soft constraints, e.g. elastic connections, and hard ones, e.g. geometric constraints. The former can be handled by adding viscoelastic elements to the system, while for the former the following closed form expression can be adopted from Eq. 9

$$\begin{bmatrix} \Sigma_i \bar{M}_i & T_c^\top \\ T_c & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda_c \end{bmatrix} = \begin{bmatrix} \Sigma_i \bar{d}_i - \Sigma_i w_i^\top \\ -d_c + u_c \end{bmatrix}, \qquad (15)$$

where $T_c = \chi_{c,q}$ is the Jacobian transformation matrix for the constraint, $\chi_c = 0$ is the geometrical constraint relation, λ_c is Lagrange multiplier, $d_c = (T_c\dot{q})_{,q}\dot{q}$ and u_c is a control term that can be used to set a desired acceleration for the constraint term as $u_c = \ddot{\chi}_c + C_c$. u_c can be used to design a Jacobian based nonlinear controller by setting it to the desired acceleration control input for the constraint geometry (χ_c). C_c can be a constant or a PID term to compensate for the numerical integration and/or inversion errors for a constraint, or to cancel out the control tracking error in a controller design.

Software Algorithm & Implementation

The discussed modeling frameworks for soft robots have enabled us to incorporate their states into the traditional modeling framework of rigid body dynamics. In the following section, these modules of the $TMTDyn$ Matlab software package are explained for the purpose of automating the derivation, simulation, and visualization of the model for a hybrid system. The $TMTDyn$ package structure and flowchart are presented in Table 2, 3 and Fig. 3. The package consists of 5 main stages and 10 sub-stages, implemented in 13 modules each in a separate Matlab file. Some modules have sub-functions, embedded in the same file with local only access. The connection between the stages/modules are provided by three means: (i) passing a Matlab structure variable, named par and defined in the $system.m$ file, between the modules, (ii) generating the necessary functions (Matlab, C or C-mex format) after completion of some stages and calling them in the next stages, and (iii) calling the main modules in a single file ($system.m$) that returns the resulting parameters to the Matlab workspace environment for later use. The system geometry, in the form of a set of bodies that are interconnected with compliant elements and constraints, is described in the User Interface module ($system.m$) using an HLL. The elements of the HLL interface are explained in Appendix 1 Table 10-12. Next, the $check.m$ module is called to check the defined parameters and assign default values if needed. The system is passed to the $tmt\_eom\_derive.m$ module to derive the hybrid system TMT EOM. Depending on the method of choice for optimization and storing the derived EOM, Matlab, C-mex or C++ functions are generated by calling the $save\_func.m$ and $save\_mex.m$ modules. The intermediate steps in the numerical simulation should be defined in $int\_mid\_step.m$ by the user. Then, $int\_eom\_mex.m$ is called to generate the code that is used for numerical static and dynamic simulations. $equil.m$, $modal.m$, and $dyn\_sim.m$ are called to perform static equilibrium analysis, linear modal analysis, and numerical dynamic simulation. Finally, the results are passed to $anim.m$, providing a simple animation, and to $post\_proc.m$ (provided by the user) for any intended post-processing of the results. Different elements of the package are explained further below. Pseudocodes for each file are presented in Appendix 2.

System Declaration

The package control parameters, and the system properties (geometrical, inertial and stiffness) and inputs (actuator inputs, external loads) are described in $system.m$ file, using a text-based High Level Language. The HLL inputs are Matlab structure variables that store the package control parameters (par), properties of the simulation environment ($world$), and a dynamic system consisting of bodies ($body$), joints ($joint$), mesh elements from a CAD (Computer Aided Design) file ($mesh$), and external loads ($exload$). The fields, types, input options, default values, and suggested unit for these structure variables are explained in detail in Appendix 1 Table 10-12. Then different modules are called respectively based on the package control parameters par. These modules derive the EOM ($tmt\_eom\_derive(...)$), solve static equilibrium ($equil(...)$), perform linear modal analysis ($modal(...)$), simulate the system dynamics ($dyn\_sim(...)$), present and record simple animation of the system geometry and motion ($anim(...)$), and perform user-provided post processes on the results ($post\_proc(...)$). Pseudocode for $system.m$ is presented in Appendix 2 Algorithm 1.

par contains the package control ($par.anim$, $par.mov$, $par.derive$, $par.fun$, $par.mex$, $par.equil$, $par.modal$, $par.dyn$, $par.nint$) and modeling ($par.sym$, $par.var$) parameters. The modeling parameters, e.g. system dimensions, can be set to numeric values or symbolic variable, stored in $par.sym$. It is possible to vary the symbolic values, stored in $par.sym$, after deriving the system EOM to provide flexibility. The numeric values remain constant speeding up the derivation, function optimization/generation and simulation processes. $par.sym$ is considered as an input for the generated functions from derived EOM. $par.var$ contains the numeric values for $par.sym$ to be set after EOM derivation, e.g. for numerical simulation. For detailed explanation of package control parameters see Appendix 1 Table 10-12.

$world.g$ stores the gravity vector as $[g_x, g_y, g_z]$. $body$, $joint$, and $exload$ store the system structural parameters. Assume a system with n_m different bodies or body sets ($body$), n_j joints or joint sets ($joint$), and n_l external loads ($exload$). A single $joint$ or $exload$ my describe a set of n_d elements with similar properties, which we call a mesh. Series of links with relative DOFs, nodes with absolute DOFs, and any arbitrary interconnections between the nodes in a mesh are possible. A $joint$ with rom field defines a continuum rod modeled with ROM. A mesh geometry cannot be defined with a ROM $joint$. The first $joint$ that is connected to a $body$ is a member of the Main Kinematic chain (MK). Any other $joint$ connected to a body is a compliant element (if $spring$, $damp$, or $input$ fields are defined) or a geometric constraint (if $fixed$ field is defined). A $body$ that is connected to a mesh $joint$ defines the inertial properties of the mesh elements. An alternative way to define a mesh body is importing a mesh CAD file. The $mesh$ structure stores the CAD file name, a $body$ field to be assigned to the mesh nodes, and two $joint$ fields to define the bodies absolute DOFs (w.r.t. reference frame) and to assign to the mesh edges. To generate a mesh geometry with a ROM $joint$, a $mesh$ with such joints can be defined to assign to a CAD file imported geometry. Brief descriptions for the fields in each structure variable are provided below. We use **SI** units throughout this draft.

$body$ has the following fields. m is the body mass, I is the 3×3 inertia matrix, $.l\_com$ is the COM position vector, and

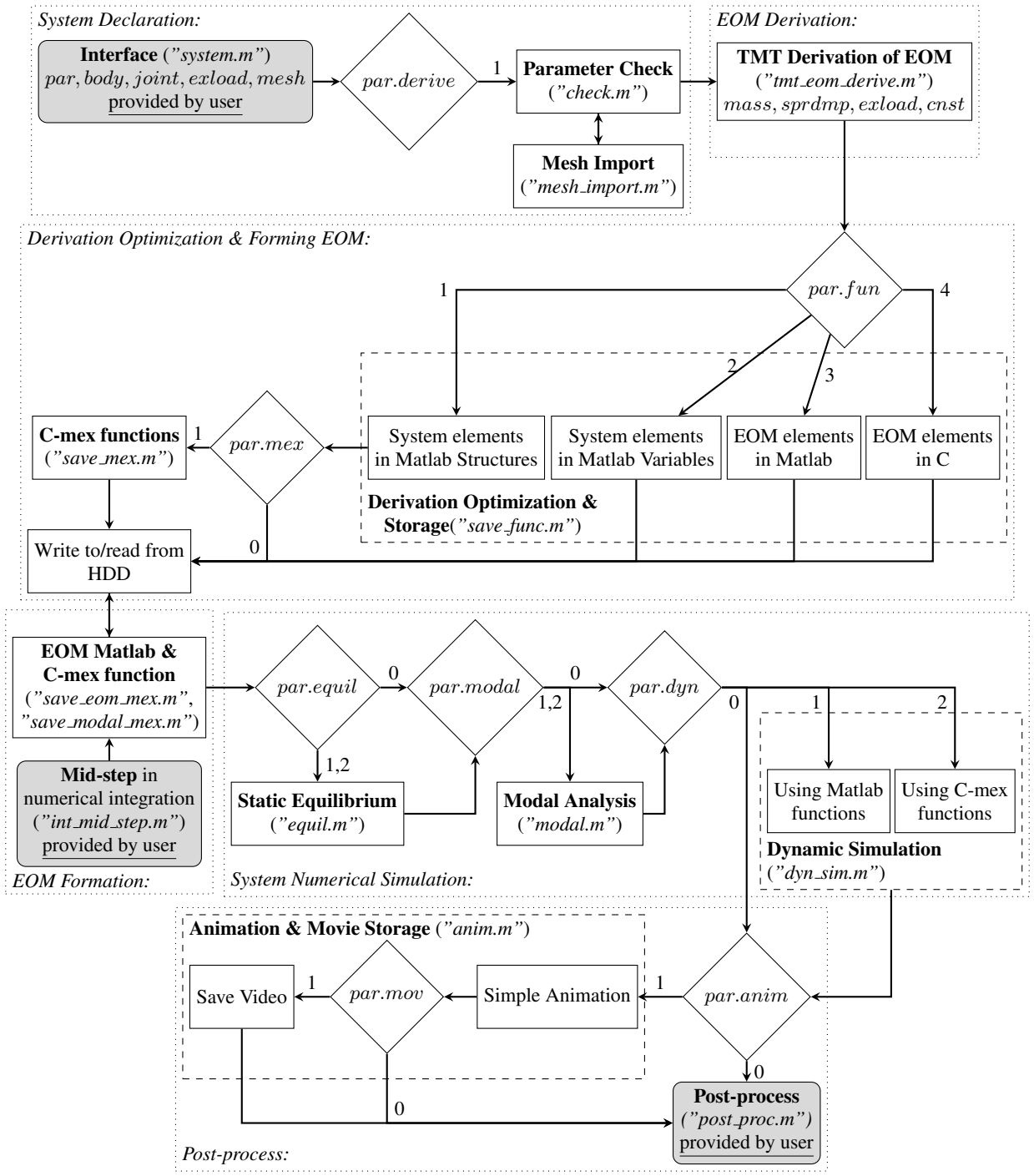


Figure 3. TMTDyn package flowchart.

tip is the link tip position vector, both in local frame. If a $body$ describes a mesh with n_d elements, each of the above fields can have different values for each mesh elements. A $n_d \times 1$ column vector is assigned to each parameter that each row stores the value for a mesh element with corresponding row number. We follow the convention throughout this paper, that individual values for mesh elements are stored in a different column of a vector or matrix, except for I that becomes a $3 \times 3 \times n_d$ cube variable. In case of ROM continuum elements, all the fields present differential values with "per unit length" unit. n_m is the total number of bodies defined, i_m is their unique numerator, and $n_{md} = \Sigma_{i=1}^{n_m} n_{d_i}$ is the total number of masses defined in the system.

$joint$ describes the system geometric connections and defines one of the following: i) system DOFs if being a member of MK, ii) a continuum ROM element if it contains rom filed and is in MK, iii) a compliant connection, e.g. spring, viscous damper, or actuator input, if not being a member of MK and contains any of $spring$, $damp$, or $input$ fields, iv) a geometric constraint if the $fixed$ field is defined and $joint$ is not in MK. It can define a mesh too if it is connected to multiple instances of a $body$, in any of the above cases, except for a ROM $joint$. $joint$ has the following fields. rom defines if the $joint$ is a ROM link. rom has one filed $order$ setting the ROM polynomial order (n_r) that should be defined (SBD). $first$ and $second$ are 1×2 or $1 \times n_d + 1$ row vectors defining the $body$ number and

instance/axial location that the $joint$ ends are connected to. The first element of the row defines the $body$ number. Either of the ends should be connected to a single $body$. In the case of a 1×2 vector, i) the second element defines the ROM continuum beam length, if $joint$ is a ROM (has rom field) in MK, ii) it is axial location of contact on the continuum beam, if $joint$ is not in MK and the target body is a ROM, or iii) it is the instance of the $body$ that the $joint$ is connected to if $joint$ is not in MK and the target $body$ is not ROM. In the case of a $1 \times n_d + 1$ vector, i) the $joint$ creates a mesh with n_d elements if $joint$ is in MK, and ii) it connects to instances of the body defined in $2 : n_d + 1$ elements if $joint$ is not in MK. $joint$ cannot be a ROM when defining a mesh. Leaving these fields completely empty (fields' default value) means connecting to the reference frame (ground), and if a scalar is assigned, the first instance of the $body$ is used. $first$ and $second$ must have the same number of elements, if defined and their assigned vectors have three or more elements.

Each $joint$ can define n_t number of consecutive transformation between its ends that are defined in the tr field. Each tr has a translation followed by a rotation pair defined in $trans$ and rot fields with zero default values. $trans$ is a 1×3 position vector in local frame. rot can be i) a principal axis and rotation set defined as elements of a 1×2 vector respectively (eg. $[2, \theta]$ defines a rotation of θ around local frame y-axis), ii) a 1×3 vector defining the axis of a unit quaternion (Q_r), or iii) an angle-axis 1×4 vector with the first element as the angle (cannot handle DOF). $trans$ and rot can define a fixed transformation, if their elements are set to numeric values, or a free DOF, if set to inf. The properties for each DOF are defined in a dof field with zero default value. n_h is the number of DOF definitions and $n_q = \Sigma_{j=1}^{n_j} n_{h_j} n_{d_j}$ is the total number of states (generalized coordinates) in a system. dof has the following fields mostly with zero default values: initial value ($init$), initial values axial location for a ROM $Joint$ ($init_s$), elastic properties in $spring$ (spring coefficient($spring.coeff$), spring initial value ($spring.init$), and initial compassion ratio ($spring.compr$)), active direction (dir), damping properties in $damp$ (viscous damping coefficient ($damp.visc$) and damping power ($damp.pow$)), and direct actuator input ($input$). The spring/damping/input elements act in parallel to the DOF (having the same displacement). If the $spring.init$ is set to nan, this value is assigned automatically based on the system initial configuration (system DOFs' initial value). dir sets the active direction of these elements, not the DOF itself. A $joint$ defines a mesh by assigning matrices and vectors with n_d rows to its fields, except for a ROM $joint$. All instances of a mesh $joint$ have the same tr but can have different DOF properties. This is possible by setting the dof subfields to matrices with n_d rows. For a ROM $joint$, $init$ is a matrix with 3 rows, for the 3-value Cartesian location of a point at an axial location defined by $init\_s$, and an unlimited number of columns, the number of points along the axis which is equal to the number of $init\_s$ elements. An inverse problem is solved in the $check.m$ file to find proper initial values for the coefficients of a n_r-order polynomial that passes through all these points. An initially curved beam can be defined as a ROM body by setting proper values for the $init$ and $init\_s$ fields.

A $joint$ in MK does not need any definition for a transformation in the $second$ connecting body frame. The $second$ body local frame is assumed to be attached to the joint itself. $tr2nd$ defines such transformation w.r.t. the $second$ body local frame if $joint$ is not in MK. We name these "linking" $joints$. They have the same fields as tr but cannot have inf elements, i.e. define any new DOFs. A ROM or linking $joint$ can have dir, $spring$, $damp$, and $input$ fields as stated above. The assigned values for a ROM $joint$ should have the same number of columns as the order of the ROM polynomial. An elastic Euler-Bernoulli beam can be defined by assigning six-column vectors/matrices to these fields. The six columns correspond to $[\xi, \zeta]$ states of an Euler-Bernoulli beam as in Eq. 5. $spring.initi$ values can be set to nan for an EB beam but not for a ROM one. By default, such a beam is defined along the z-axis of the frame defined by tr. Alternatively, a new right-hand orthogonal frame is calculated using the aforementioned z-axis and the director axis defined in $xaxis$ field. The matrices assigned to these fields can have n_d rows (except for a ROM $joint$), to define and/or assign different values to elements of a mesh $joint$. If a row vector is assigned for a mesh $joint$, same values are assigned to all the mesh elements. Finally, $fixed$ and $control$ define the constrained directions and their desired control value. $fixed$ accepts a boolean vector and a geometric constraint is defined for each direction that is set to 1. A 1×3 vector can be used to constrain local Cartesian directions ($[x, y, z]$) or a 1×6 vector to constrain the different states of an EB beam ($[\alpha, \epsilon]$). This does not override the $spring$, $damp$, or $input$ fields for the constrained directions. The dynamic actions associated with the compliant elements parallel to the constrained directions should result in zero. A spring and damper in parallel to these directions act as PD control terms to minimize the numerical simulation errors in satisfying the constraints. $control$ can be set to a symbolic variable and then updated during the simulations with the desired acceleration (since EOM is of 2^{nd} differential order) of the constraint geometry. Setting it to zero (its default value) fixes the constraint. It is useful for designing a Jacobian nonlinear controller. The dir field is ignored for constraints. Also $fixed$ field is ignored for a ROM $joint$.

An alternative way to define a complex geometry is importing a CAD mesh file with extension "iges" or "stl". The $mesh$ structure has a $file\_name$ to store the CAD file name, tol to set the points' overlap tolerance of the CAD file, tr to set initial position and orientation of the imported body, a $body$ and two $joint$ sub-structures with the same fields as stated before. A $body$ is assigned to each node of the CAD file that at least two edges are attached to it. Nodes that are only attached a single edge are assumed fixed connecting points with the ground. Each "body" has absolute DOFs, described in $mesh.joint(1)$. A linking $joint$ is assigned to each edge of the CAD file based on $mesh.joint(2)$. The CAD-file coordinates transform based on tr filed and then used to define the initial condition of the imported elements. The links are numbered based on the order of extracted lines from the CAD file and overlapping of the nodes. Hence, it is hard to predict the assigned numbers to the links and masses. A plot is shown at the end of the importing process with labels showing the assigned mass

numbers. The imported $bodies$ will be added after all the previously defined $body$ instances, hence their labels may start with numbers greater than 1. The imported $joints$ will be added before all the previously defined joints to satisfy our definition of MK $joints$. The import process is handled in the $mesh\_import.m$ file. The file return structure vectors for created $body$ and $joint$ instances. Only one $mesh$ element can be defined (one CAD-file can be imported) at the moment. Pseudocode for this file is presented in Appendix 2 Algorithm 2.

External loads on the system are defined in $exload$. n_l is the number of defined external loads. $exload$ has the following fields. $exbody$ defines the exerting body number and the body instance, in case of a mesh, or axial location of the external load, in case of a ROM body. $exload$ can define a set of loads if a matrix with multiple rows is assigned to $exbody$, the same as defining a mesh with a joint that is explained earlier. The exerting point location in the local frame is defined by the $trans$ and rot fields. The external force is defined as a 1×6 vector, in the form of $[f, \tau]_l$, in the $ftau$ field. $ftau$ elements can be set to symbolic parameters and later updated in the $int\_mid\_step.m$ file during the simulation steps. $ftau$ is measured in the local frame of the $body$ which is defined by $refbody$. If $refbody$ is not defined, $ftau$ is then measured in the reference frame. A ROM $body$ cannot be set as $refbody$. For a complete explanation of these fields, and their acceptance and default values see Appendix 1.

Most of the above fields can be left undefined or empty. The $check.m$ module assigns the default values, formats the input matrices, and performs some simple complication checks for the parameters. Pseudocode for this file is presented in Appendix 2 Algorithm 3. Examples of different hybrid systems are provided later to clarify the application of the HLL and text-based user interface. The inputs are then passed to the $tmt\_eom\_derive.m$ module to derive the system TMT EOM. After calling the $mesh\_import.m$ and $check.m$ modules, the code proceeds to derive the system TMT EOM and generate optimized Matlab, C-mex, and/or C functions.

TMT EOM Derivation & Optimization

The system TMT EOM is derived in a set of Matlab structure variables. First the code iterates through all the $joint$ vector elements (n_j), their instances (n_d) and ROM order (n_r). This step identifies inf elements in $trans$ and rot subfields, and generates the system states $(q, \dot{q})$ and collects the elastic f_{k_q}, viscous damping (f_{v_q}) and input (f_{u_q}) actions in parallel to each DOF for the whole system. This step calculates each joint instance rotation ($joint(i_j).Q(i_d).loc$) and transformation ($joint(i_j).TQ(i_d).loc$) in the local frame too.

Then the code iterates through all $body$ vector elements (n_m) and finds their MK $joint$. The $joint$ type (ROM or not) and number of instances (n_d) define the type and number of $mass$ elements in the system. The MK joints rotation ($joint(i_j).Q(i_d).abs$) and transformation ($joint(i_j).TQ(i_d).abs$) in the reference frame are calculated. The terms for each inertial element (M for $\bar{M}_i$, T for T_{m_i}, Dd for D_{m_i}, and fg for $f_{g_{m_i}}$) are stored as fields of a $mass$ structure vector. These terms are collected for the whole system in single variables (M for $\bar{M}$, T for T_m, Dd for D_m, and fgv for f_{g_m}). $r\_jtips_{[}n_m d \times 6]$ matrix stores the base and tip position vector of all the $mass$ vector elements. $rom.mass$ stores zeros for rigid links and length of the beam for ROM rods.

Then the code iterates through all the $joint$ vector elements (n_j) and their instances (n_d) to find the linking joints and constraints. The $joint$ type (axial element, EB beam or ROM beam) defines the type of compliant elements in the system. The terms for each linking $joint$ (Tt for $T_j^\top$, kx to calculate f_{k_j}, vd to calculate f_{v_j}, dl for $\bar{\rho}_j$, in for f_{u_j}, k_mat for K_j, v_mat for V_j, and dir for joint active direction) are stored as fields of the $sprdmp$ structure vector. These terms are collected for the whole system in single variables (fj_k for f_{k_j}, fj_vd for f_{v_j}, fj_in for f_{u_j}, fj_sdi for $f_{k_j} + f_{v_j} + f_{u_j}$, fj_k_mat for K_j, and fj_vd_mat for V_j). The $r\_ks_{[}n_m d \times 6]$ matrix stores the base and tip position vector of all the $sprdmp$ vector elements. $rom.sprdmp$ stores zeros for rigid links and length of the beam for ROM rods. The code checks if any of the link directions are fixed. $lambda$ row vector and $cnst$ structure vector store the terms for constrain ($lambda$ for λ_{c_i}, T for T_{c_i}, D for D_{c_i}, and in for u_c). Tcn for T_c, Dcn for D_c, and Ccn for u_c are single variables that collect all the constraint terms in the system.

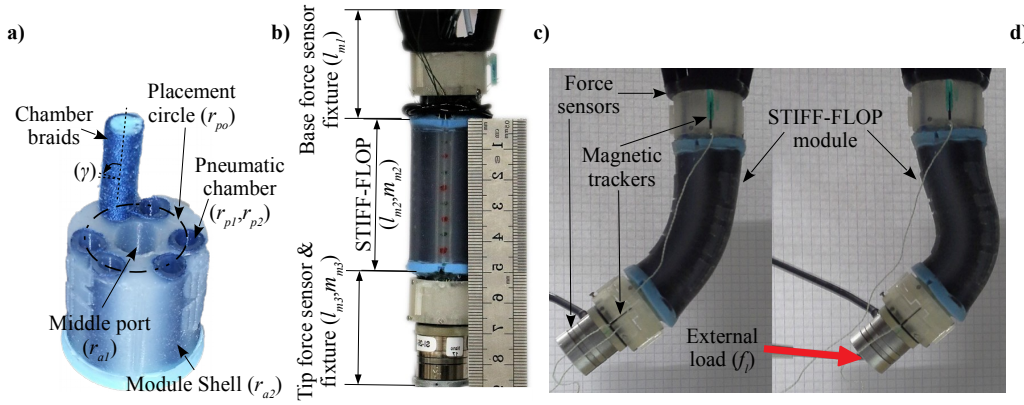
Finally, the code iterates through all the $exload$ vector elements (n_j) and their instances (n_d). The terms for each external load term (Tt for $T_{l_i}^\top$ and $ftau$ for f_{l_i}) are stored as fields of the $loads$ structure vector. These terms are collected for the whole system in single variables ($Ttef$ for $T_l^\top$ and $ftau\_ef$ for f_l).

The structure vectors are used in a framework as in Eq. 9 while the single variables that collect the terms for the whole system form the EOM closed vector form for the whole system. We use the structure vectors to have more flexibility in handling different link types and for faster derivation and optimization of the derived parameters. For a complete definition of the variable names refer to Appendix 2 Algorithm 4-6 and source codes available at Sadati (2017).

Derived parameters are passed to $save\_func.m$ and $save\_mex.m$ modules for converting to Matlab, C, and C-mex functions and are stored as separate files in a folder named "code" in the active Matlab folder. Matlab $matlabFunction$, $ccode$, and $codegen$ functions are used to perform this task. These functions optimize the derived equations by searching for and collecting repeated terms in the code. C-mex functions are only available for structure vectors. The generated Matlab functions are used to debug the model and code, by setting $par.equil$, $par.dyn$, and $par.modal$ to 1. Once the model is debugged, C-mex functions can be used for increased numeric performance, by setting $par.equil$, $par.dyn$, and $par.modal$ to 2. Pseudocode for these modules is presented in Appendix 2 Algorithm 7.

Dynamic System Analysis & Post-Processing

The system EOM are constructed using the generated functions in the previous step. The $save\_eom\_mex.m$ module generates the EOM for dynamic and static analysis. The system states are named Z consisting of the geometric states q and Lagrange multiplier λ_c vector and their temporal derivatives. This module generates the EOM code as a string



d)

Sym.	Value
r_{a_1}[mm]	4.5
r_{a_2}[mm]	12.5
r_p[mm]	2.5
r_{p_o}[mm]	8.5
l_{m_2}[mm]	50
l_{m_3}[mm]	14.5
m_2[gr]	28
m_3[gr]	13.47
γ[deg]	87
E[KPa]	205
σ[Kg/m^3]	1820
μ_ϵ[Ns/m]	0.1-100
μ_α[Nms/rad]	0.5-1e-5
ν	2

Figure 4. a) Structural design, b) parameters and c) sequences of experiments with a pneumatically actuated STIFF-FLOP continuum appendage. d) The structural parameters and their units are based on experimental measurments.

variable ($string\_all$) and stores it as a Matlab function file, using the Matlab $fprintf$ function. The module calls fj_k, fj_vd, and fj_in to account for the compliance and inputs in parallel to the system states. Then it iterates through all the $mass$ vector elements and generates a piece of code that calculates and sums up the summation in Eq. 8. The code performs a spatial integration if the $mass$ is a ROM element based on its corresponding element in $rom.mass$. Then it generates the necessary code for the linking joints in $sprdmp$, external loads in $loads$, and constraints in the $cnst$ structure vectors. Finally, the code to calculate Eq. 9 is added to $string\_all$. The differential terms are omitted for static equilibrium analysis. $string\_all$ is written in $EOM.m$ file for dynamic simulation and in $EOM\_eq.m$ for static analysis. A C-Mex function is generated for each of the above Matlab functions. Pseudocode for this module is presented in Appendix 2 Algorithm 8. A similar module ($save\_modal\_mex.m$) generates necessary equations for linear modal analysis based on the system linearized EOM as in Eq. 14. The generated functions are stored in the $EOM\_modal.m$ file. $par.equil$, $par.modal$, and $par.dyn$ values control if an analysis is performed and which generated function is used.

Static equilibrium analysis is used to solve for the static model of a system or initial equilibrium point before a dynamic simulation. If $par.equil$ is not zero, the $equil.m$ module is used to solve the system static equilibrium. Matlab's $fsolve$ function is used to solve the static equilibrium problem. $EOM\_eq.m$ is used if $par.equil = 1$, otherwise C-mex function is used. The $dyn\_sim.m$ module is used to perform a dynamic simulation, if $par.dyn$ is not zero. Matlab's $ode15s$ or $ode113$ functions are used to perform a numerical temporal integration, using the $EOM.m$ function if $par.dyn = 1$ or using the generated C-mex function otherwise.

The results (analysis time t and states Z) are passed to the $anim.m$ module to generate and record a simple animation of the results. The simple plots show rigid links with continuous lines, ROM continuum links with continuous line curves, and compliant joints (axial elements or beams) with dashed lines. $par.anim$ and $par.mov$ control if an animation is generated and/or recorded to a video format file. the $post\_proc.m$ module is considered to perform user-specified post-processes on the generated data. This file does not have any code by default.

In the next section, different examples of modeling multi-dimensional continuum and hybrid systems are provided. $TMTDyn$ enables us to provide comparisons between numerical performance and accuracy of using different modeling assumptions and complexity levels. This helps with deciding the most appropriate method for similar hybrid-system modeling tasks.

Case Studies, Validation & Discussion

Two cases of hybrid rigid-continuum systems are modeled and the results are verified in comparison to experimental results. The studies cases are: E1) dynamic motion of a single STIFF-FLOP continuum appendage in presence of external loading, and E2) dynamic deformation of a fabric sleeve worn on an elbow-like rigid-link pendulum. Different modeling assumptions for continuum rods are tested and compared for E1 in terms of modeling complexity, EOM derivation time, dynamic and static simulation time, and accuracy of the results in comparison with experiments. As a result, the most efficient and accurate method for modeling 1D continuum elements, i.e. continuum rods, is identified. The challenges in modeling a 2D (fabric) continuum media are investigated in E2. The case studies show how the proposed unified modeling framework and software package help in deriving simple Lagrange EOM for hybrid systems. Further discussion is provided on the advantages of such models for controller and observer design tasks.

The mean (M) and Mean Standard Deviation (MSTD) values for the experimentally measured values are compared with the numerical simulation results to evaluate absolute (Err) and percentage error (%Err) for each case study. A Lenovo Yoga 3 Pro 1370 laptop computer (Intel Core M-5Y71 CPU, $2 \times 1.2 - 2.9$GHz cores, 8 GB of RAM) with Ubuntu 18.04 LTS operating system was used to perform the analysis in this paper.

Dynamics of A Continuum Appendage

STIFF-FLOP Continuum Appendage- A STIFF-FLOP (STIFFness controllable Flexible and Learn-able manipulator for surgical OPerations) module Fras et al. (2015) is a

pneumatic continuum appendage (Fig. 4). Earlier tests show their high repeatability and negligible performance change due to aging and fatigue Shiva et al. (2018). The manipulator is made of silicone elastomer (Ecoflex 50 from Techsil) and selective actuation of the three braided pneumatic chamber pairs (6 chambers in total) via separate electronic proportional micro pressure regulators (Camozzi K8P) provides 3-DOF (one axial elongation and two side-bendings) of the appendage tip (Appendix 3 Fig. 9). The first chamber pair is placed along the manipulator $+y$-axis with 120 [deg] offset from the other pairs (Fig. 4.a). Chambers in each pair have about 40 [deg] polar offset with each other ($\phi_{po} \approx 20$ [deg] w.r.t. their symmetry line). The pressure regulators are connected to a compressor (BAMBI MD Range Model 150/500) and controlled via a data acquisition (DAQ) board (National Instruments Inc. NI-DAQmx USB-6411) and its control software. A control diagram for this study is presented in Appendix 3 Fig. 8. An ATI Mini40 6-axis force sensor is mounted at the appendage base and an ATI Nano17 6-axis force sensor is connected at the manipulator tip to measure the external loads at the appendage tip (f_{l_t}). Fig. 4.b shows the setup elements. Sequences of the experiments with and without tip external loads are shown in Fig. 4.c . The measured and identified structural parameters of the experimental setup are presented in Fig. 4.d.

Modeling Assumptions and Program Input- Each STIFF-FLOP module has 3 pairs of braided pneumatic actuators so we set $p_{2i-1} = p_{2i}$ for the three input pressures with index $i \in [1, 3]$. Their resulting axial force is $f_u = \Sigma_{i=1}^{6} p_i a_p$ and the vector of bending/torsion is $\tau_u = \Sigma_{i=1}^{6} p_i a_{\mathrm{p}} r_{O_i} \times [0, 0, 1]$, where r_p and $a_p = \pi r_p^2$ are the pneumatic chamber inner radius and area. Links are considered as EB beams with linear elasticity $K_\epsilon = \mathrm{diag}(a_{\mathrm{c}}[G, G, E])$ and $K_\alpha = \mathrm{diag}([E, E, G]).\mathrm{diag}(J)$ which are diagonal stiffness matrices associated with strains (ϵ) and curvatures/torsion (α) respectively, in the $\hat{d}_i$ frame. Here, E and $G \approx E/3$ are the material elasticity and shear modulus, $J = \pi/4(r_{a_2}^4 - r_{a_1}^4 - 6r_p^4).[1, 1, 2] - a_p \mathrm{diag}(r_O.r_O^\top)$ is a 1×3 vector consisting of the cross-section second moments of areas. r_O is a matrix of which rows are position vectors of the chambers in the manipulator cross-section plane as

$$\begin{aligned} r_{O_i} &= r_{p_o}.[\cos(\psi_{o_i}), \sin(\psi_{o_i}), 0]\,,\ i \in 1...6 \quad (16) \\ \psi_{o_{2i-1}} &= \pi/2 - 2(i-1)\pi/3 - \psi_0,\ i \in 1...3, \\ \psi_{o_{2i}} &= \pi/2 - 2(i-1)\pi/3 + \psi_0, \end{aligned}$$

where r_o is the radial offset of the chambers from the center.

The finite length or infinitesimal length masses are assumed to be cylinders which are attached to the point of interests, e.g. joint in discretized methods, at their COM. As a result, we have a symmetric distribution of mass in the body fixed local frames. For ROM the model we assume a hollow disk with differential moment of inertia $J_m = \mathrm{diag}((r_{a_2}^2 + r_{a_1}^2)[1/4, 1/4, 1/2])\mathrm{d}m_2)$, where $\mathrm{d}m_2 = m_2/l_{m_2}\mathrm{d}s$. For discretized methods we have $J_m = \mathrm{diag}(m_2 \left((r_{a_2}^2 + r_{a_1}^2)[\frac{1}{4}, \frac{1}{4}, \frac{1}{2}]) + (\frac{l_{m_2}}{n_d})^2[\frac{1}{12}, \frac{1}{12}, 0]\right))$. The effects of actuation chambers are neglected here.

The four modeling assumptions presented in Fig. 2 are implemented using the $TMTDyn$ package as in Tables 4 & 5. The properties of these models and the abbreviation used to describe each are presented in Table 6.

Numerical Integration Mid-step Function- An intermediate step is applied in the numerical integration to account for the material hyper-elastic deformation Gazzola et al. (2018) as $r^+ = r/\sqrt{|\epsilon_3|}$ for all radii, $l^+ = |\epsilon_3| * l$ for link lengths and $E^+ \mathrm{or} G^+ = E \mathrm{or} G/|\epsilon_3|$ Sadati et al. (2018b); Shiva et al. (2018), except for r_p which remains constant due to the dense braid constraint ($\gamma \approx \pi/2$) and in calculating f_{g} which is independent of the cross-section deformation. $|\epsilon_3|$ is the rod mean axial stretch which is $\int_{s=0}^{l} \xi_3 \mathrm{d}s$ for ROM model, where $\xi_{i_3} = sprdmp(i).dl$. For SRL and EBR models we have $\Sigma_{i=1}^{n_d} \epsilon_{i_3}$, where ϵ is a part of system total states (Z). The same relation is valid for the EBA model, but $\epsilon_{i_3} = sprdmp(i).dl$ is the linking EB beam lengths. In addition, the force and moment due to input pressure f_{u_p}, τ_{u_p} needs to be updated during the simulation. The code to implement these considerations is provided in $int\_mid\_step.m$ module. We introduce some symbolic variables in $par.sym$ for $|\epsilon_3|, f_{u_p}, \tau_{u_p}$ and update their corresponding values in $par.var$ in $int\_mid\_step.m$ module.

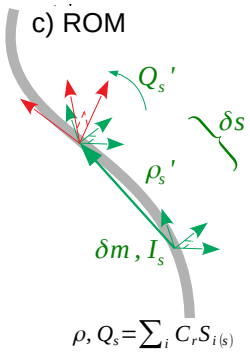
Experimental Results & Discussion- Two sets of experiments were carried out with and without external load at the manipulator tip, and for different input pressures. Each experiment takes about 55 [s] and dynamic data for the actuator inputs, the manipulator tip position, orientation, and force were recorded. Sample recordings from the two experiments in comparison to simulation results from EBR model with $n_d = 4$ are presented in Fig. **??**.

Matlab $ode15s$ was used to speed up the numerical integration in our simulation in this section. We observed that except for ROM, other models are very sensitive to the change of cross-section parameters due to the hyper-elasticity assumption. Hence, we neglected this change for the SRL, EBR, and EBA cases to avoid any numerical divergence in the simulations. Considering hyper-elasticity effects, if possible, improves the simulation results accuracy by up to %6. This is in accordance with our previous observations in Sadati et al. (2018b). We set $\mu_\xi = 0.1, \mu_\zeta = 1e-5$ for ROM and SRL, $\mu_\xi = 100, \mu_\zeta = 0.5$ for EBR, and $\mu_\xi = 100, \mu_\zeta = 0.01$ for EBA model.

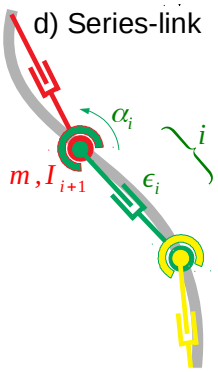
Fig. 5 presents a comparison between computational performance and accuracy of these models with experimental results. ROM consumes the least memory and computer CPU time to derive the EOM, but EBA is the best in terms of CPU time for equation optimization. SRL is the worst in this regard. It takes hours to optimize the EOM for a system with more than three consecutive links. The change in system links, either to improve accuracy of a single rod model as presented here or in a system with multiple links, affects the EBA model CPU usage the least. For planar geometries (results are not provided here), a system with three times n_r or n_d of those presented here consumes the same memory and CPU time. As a result, EBA is the best model for systems with a large number of bodies. EBR and EBA have the best simulation time, as well as static and dynamic, performance. However, EBR showed to be very sensitive to sudden changes in the input pressure and external force values. High viscous damping values were considered to prevent exponentially growing errors (numerical analysis diverge) in this case. As a result, EBR simulation outputs were not

Table 4. $TMTDyn$ package input for different models of a continuum rod, based on a pneumatically actuated STIFF-FLOP continuum appendage. Model labes are as in Fig. 2. Continues in Table 5...

$world.g = [0, 0, -g]$; gravity
$body(1).m = m_2/l_{m_2}$; ROM body
$body(1).I = I_2$;
$joint(1).second = [1, l_{m_2}]^\top$; ROM joint
$joint(1).rom.order = n_r$;
$joint(1).tr.trans = [inf, inf, inf]$;
$joint(1).tr.rot = [0, inf, inf, inf]$;
$joint(1).dof(3).init = (1 : n_r)l_{m_2}$; DOF properties
$joint(1).spring.coeff = [\text{diag}(K_v)^\top, \text{diag}(K_u)^\top]$; EB beam stiffness
$joint(1).damp.visc = [\mu_\xi, \mu_\zeta]$; EB beam viscous damping
$joint(1).damp.pow = \nu$; viscous damping power law index
$joint(1).input = [f_{u_p}, \tau_{u_p}]$; pressure inputs
$body(2).m = m_3$; tip force sensor
$body(2).I = I_3$;
$body(2).l\_com = [0, 0, l_{m_3}/2]$;
$joint(2).first = [1, l_{m_2}]^\top$; tip force sensor joint
$joint(2).second = 2$; $mesh = [\,]$; no mesh body
$exload(1).exbody = 2$; tip load
$exload(1).ftau = [f_{l_t}, \tau_{l_t}]$;
$exload(1).tr(1).trans = [0, 0, l_{m_3}]$;



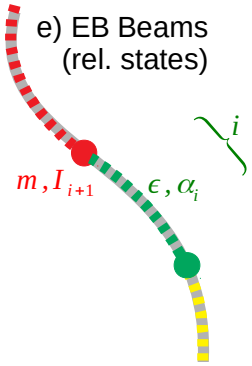
$world.g = [0, 0, -g]$; gravity
$body(1).m = m_2/n_d$; Rigid body
$body(1).I = I_{2d}$;
$body(1).tip = [0, 0, l_{m_2}/n_d]$; only required for animation plots
$K_\chi = [\text{diag}(K_v)^\top, \text{diag}(K_u)^\top]$; EB beam stiffness
$\mu_\chi = [\mu_\xi, \mu_\zeta]$; EB beam viscous damping
$f_{u_\chi} = [f_{u_p}, \tau_{u_p}]$; pressure inputs
$joint(1).first = [1, (0 : n_d - 1)]^\top$; rigid link series with 1st one connected to ground
$joint(1).second = [1, (1 : n_d)]^\top$;
$joint(1).tr(1).trans = [0, 0, l_{m_2}/n_d]$;
$joint(1).tr(2).trans = [inf, inf, inf]$;
$joint(1).tr(2).rot = [3, inf]$; rotation about $z-$axis
$joint(1).tr(3).rot = [1, inf]$; rotation about $x-$axis
$joint(1).tr(4).rot = [2, inf]$; rotation about $y-$axis
for $i_h = 1 : 6$; DOF properties **then**
$\quad joint(1).dof(i_h).spring.coeff = K_{\chi(i_h)}$;
$\quad joint(1).dof(i_h).damp.visc = \mu_{\chi(i_d)}$;
$\quad joint(1).dof(i_h).damp.pow = \nu$;
$\quad joint(1).dof(i_h).input = f_{\chi(i_h)}$;
end
$body(2).m = m_3$; tip force sensor
$body(2).I = I_3$;
$body(2).l\_com = [0, 0, l_{m_3}/2]$;
$joint(2).first = [1, n_d]^\top$; tip force sensor joint
$joint(2).second = 2$; $mesh = [\,]$; no mesh body
$exload(1).exbody = 2$; tip load
$exload(1).ftau = [f_{l_t}, \tau_{l_t}]$;
$exload(1).tr(1).trans = [0, 0, l_{m_3}]$;



reliable for fast dynamic motions. All the models show real-time performance (CPU time < 1 [s]) except ROM which has the highest CPU time demand in simulations. ROM presents significantly lower errors, even for $n_r = 1$, compared to other models, with 6 [mm] absolute error (Abs. Err.) & 9% normalized error (Norm. Err.) for static motion and 3.5 [mm] & 5.5% for dynamic motion in experiments without external loads. These values are 12.5 [mm]& 19% for static and 9 [mm]& 14% for dynamic motion in presence of external tip loads. Other models resulted in about twice these error values. This is probably due to the implementation of hyper-elasticity assumptions and the differential nature of the derived EOM. This is in accordance with our earlier results in Sadati et al. (2018b). The accuracy of almost all the models decreases with increasing n_r or n_d. This is more obvious for ROM and less for EBA. Increasing n_r or n_d more than 3 does not increase the modeling accuracy significantly while affecting the numerical performance. Results for EBR were not reliable since high viscous damping values filters parts of the dynamic motion. Our dynamic simulation results are more accurate than the static ones. However, we did not try to optimize the manipulator parameters or numerical analysis properties to find the best results, since this is not our main purpose in this study. It is possible to improve the accuracy by tuning these parameters or finding a way to effectively implement the hyper-elasticity assumption for all the modeling cases. Overall, we observed better accuracy of ROM but better numerical performance of EBA, especially for large systems. In the next section, EBA is adopted to model a hybrid system with a 2D continuum membrane.

Table 5. Continued from Table 4.

$world.g = [0, 0, -g]$; gravity
$body(1).m = [0, 0, m_2/n_d]$; Rigid body
$body(1).I = I_{2d}$;
$body(1).tip = l_{m_2}/n_d$; only required for animation plots
$K_\chi = [\text{diag}(K_v)^\top, \text{diag}(K_u)^\top]$; EB beam stiffness
$\mu_\chi = [\mu_\xi, \mu_\zeta]$; EB beam viscous damping
$f_{u_\chi} = [f_{u_p}, \tau_{u_p}]$; pressure inputs
$joint(1).first = [1, (0 : n_d - 1)]^\top$; EB beam series with 1st one connected to ground
$joint(1).second = [1, (1 : n_d)]^\top$;
$joint(1).tr(1).trans = [0, 0, l_{m_2}/n_d]$;
$joint(1).tr(2).trans = [inf, inf, inf]$;
$joint(1).tr(2).rot = [inf, inf, inf]$; EB beam bendings/torsion (α)
for $i_h = 1 : 6$; DOF properties **then**
$joint(1).dof(i_h).spring.coeff = K_{\chi(i_h)}$;
$joint(1).dof(i_h).damp.visc = \mu_{\chi(i_d)}$;
$joint(1).dof(i_h).damp.pow = \nu$;
$joint(1).dof(i_h).input = f_{\chi(i_h)}$;
end
$body(2).m = m_3$; tip force sensor
$body(2).I = I_3$;
$body(2).l\_com = [0, 0, l_{m_3}/2]$;
$joint(2).first = [1, n_d]^\top$; tip force sensor joint
$joint(2).second = 2$; $mesh = [\,]$; no mesh body
$exload(1).exbody = 2$; tip load
$exload(1).ftau = [f_{l_t}, \tau_{l_t}]$;
$exload(1).tr(1).trans = [0, 0, l_{m_3}]$;



$world.g = [0, 0, -g]$; gravity
$body(1).m = m_2/n_d$; Rigid body
$body(1).I = I_{2d}$;
$K_\chi = [\text{diag}(K_v)^\top, \text{diag}(K_u)^\top]$; EB beam stiffness
$\mu_\chi = [\mu_\xi, \mu_\zeta]$; EB beam viscous damping
$f_{u_\chi} = [f_{u_p}, \tau_{u_p}]$; pressure inputs
$joint(1).second = [1, (1 : n_d)]^\top$; series rigid links with absolute states (connected to ground)
$joint(1).tr(1).trans = [0, 0, l_{m_2}/n_d]$;
$joint(1).tr(2).trans = [inf, inf, inf]$;
$joint(1).tr(2).rot = [0, inf, inf, inf]$; absolute quaternions rotation of each link (α)
$joint(2).first = [1, (0 : n_d - 1)]^\top$; EB beam series connecting the rigid bodies and 1st one connected to ground
$joint(2).second = [1, (1 : n_d)]^\top$; $joint(2).spring.coeff = [\text{diag}(K_v)^\top, \text{diag}(K_u)^\top]$; EB beam stiffness
$joint(2).spring.init = [0, 0, nan, 0, 0, 0]$; EB beam directions initial state (geometry)
$joint(2).damp.visc = [\mu_\xi, \mu_\zeta]$; EB beam viscous damping
$joint(2).damp.pow = \nu$; viscous damping power law index
$joint(2).input = [f_{u_p}, \tau_{u_p}]$; pressure inputs
$body(2).m = m_3$; tip force sensor
$body(2).I = I_3$;
$body(2).l\_com = [0, 0, l_{m_3}/2]$;
$joint(3).first = [1, n_d]^\top$; tip force sensor joint
$joint(3).second = 2$; $mesh = [\,]$; no mesh body
$exload(1).exbody = 2$; tip load
$exload(1).ftau = [f_{l_t}, \tau_{l_t}]$;
$exload(1).tr(1).trans = [0, 0, l_{m_3}]$;

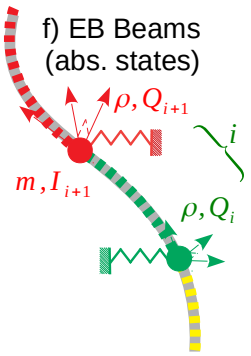


Table 6. Complexity of different rod (1D continuum element) models for a one-segment continuum manipulator with 3 active DOFs (1 axial elongation and 2 side bendings). All the models in our study have one extra rigid body mass at the tip, as tip force sensor, and an external load at the tip, that are not considered in this table.

Type	Model	No. of segments or ROM Polynomial order	Masses	joints	DOFs
Continuum	Reduced Order Model (ROM)	n_r	1	1	$6 \times n_r$
Discretized	Series Rigid Link (SRL)	n_d	n_d	n_d	$6 \times n_d$
Discretized	Euler-Bernoulli Beam with Relative States (EBR)	n_d	n_d	n_d	$6 \times n_d$
Discretized	Euler-Bernoulli Beam with Absolute States (EBA)	n_d	n_d	$2 \times n_d$	$6 \times n_d$

Dynamics of a Fabric Sleeve

Pendulum with Fabric Sleeve Setup-

A fabric sleeve, made of Jersey fabric, was cut and clamped on a rigid-link pendulum, cut to shape out of ABS clear plastic. The pendulum was fixed with a 1 DOF joint at the top and passively swings. The model was intended to capture the fabric dynamics due to the pendulum free motion. Three magnetic trackers were used to measure the link COM motion, and deformation of two points on the fabric

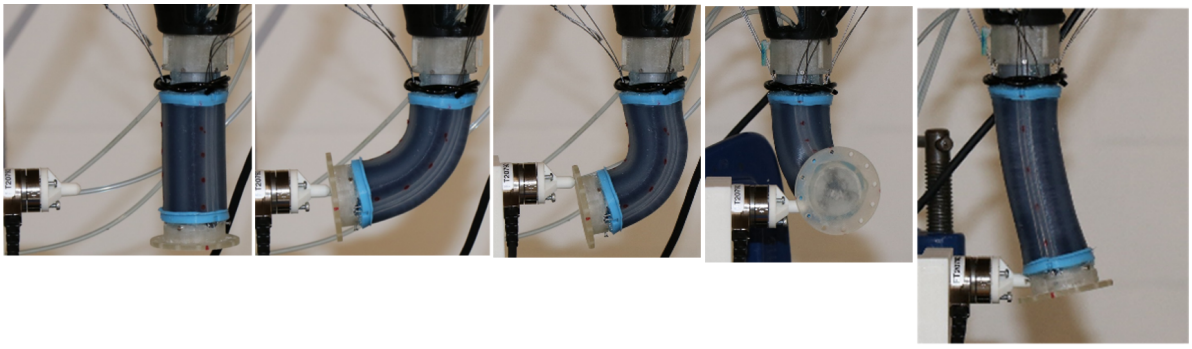


Table 7. Experiments on free and forced deformations of a STIFF-FLOP continuum appendage. An external load is exerted by a force sensor at the tip of an actuated module while the appendage tip deformation is measured by a magnetic tracker.

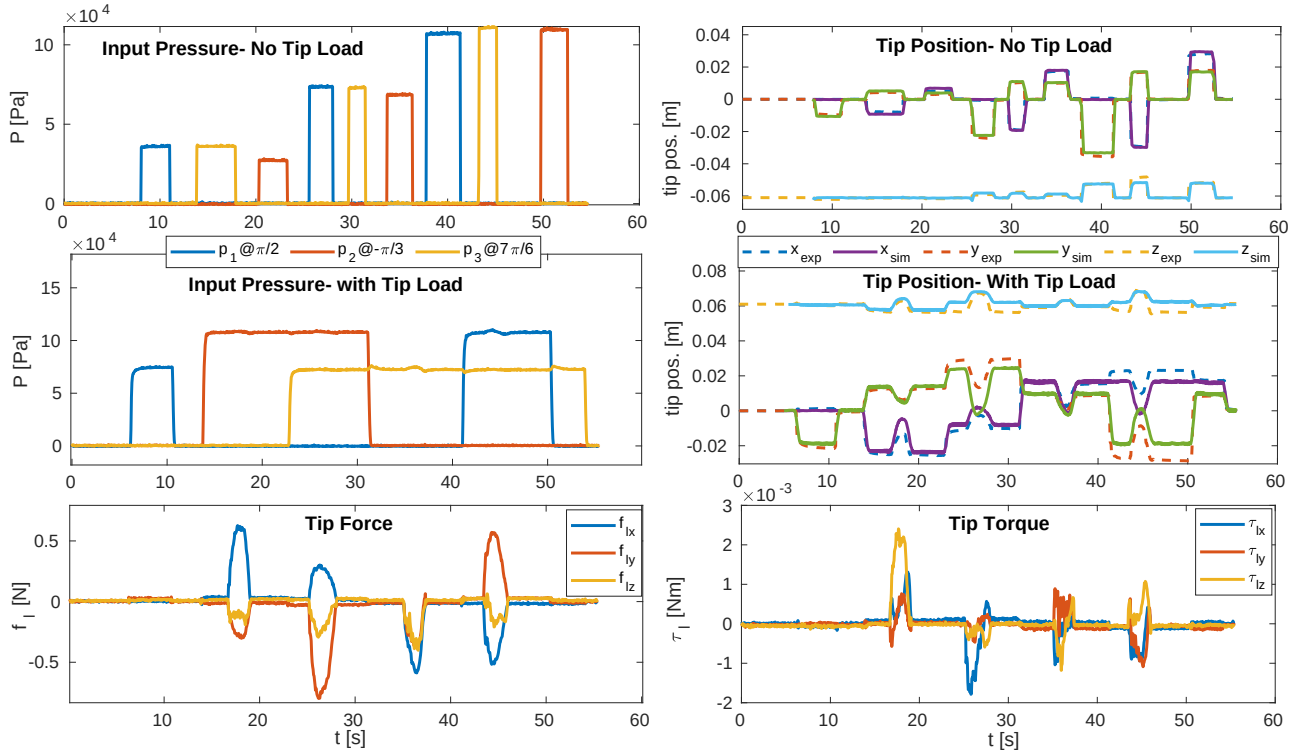


Table 8. Sample recordings from the two experiments with a STIFF-FLOP appendage in comparison to simulation results with EBR model with $n_d = 4$. top) experiments with no external load, middle & bottom) experiments with tip external loads.

(s_1, s_2). The results from such research can be useful for research on wearable sensors. The fabric parameters are not known and identified to present the best correlation with the experiments. Fig. 6 presents the fabric and setup dimensions and the simulation parameters.

Modeling Assumptions and Program Input- The fabric can be modeled as a membrane which is a 2D tension-only continuum geometry that does not withstand bending or compression. This can be done by assuming the fabric as a net of equally distributed masses with connecting linear springs. Setting $dir = 1$ in the package express that the springs are tension-only elements. We have used a similar method to model a spider web with $TMTDyn$ recently Sadati and Williams (2018). A drawback of such assumptions is that the model does not capture the crumbling of the fabric between the mesh nodes. Also, the membrane assumption may not be accurate for thick fabrics, such as the Jersey fabric used in these experiments. This is more important when the fabric takes shapes such as hollow columns, e.g. a fabric sleeve, that is more resistive against buckling and bending. The fabric behaves like a shell, a 2D continuum geometry that withstands bending and compression too, in these scenarios. We used EB beam elements to resolve this issue without the need to increase the number of nodes or introducing diagonal connections, e.g forming a tetrahedron mesh. This is a simplifying assumption that may be accurate enough to capture the underlying physics of a system with a thin membrane or shell geometries.

To model the system, we focus on the fabric model and import the link motion in the form of a constraint that follows an already recorded path ($\chi_c = \rho_{\text{com}}$) based on the experimental recording of ρ_{com}. We use Eq. 15 to model the proposed constrained system. One geometric constraint is enough to fully define the link 1 DOF motion. Due to the constraint motion of the link, the values for m_1 & I_{m_1} are not important for our analysis. Index 1 is used to define the link parameters and index 2 for the fabric ones. The fabric deforms when clamped on the link. The overall geometry of the clamped fabric is modeled with FreeCAD software as a wireframe sketch with a 3×5 grid of $n_d = 15$ nodes and 22

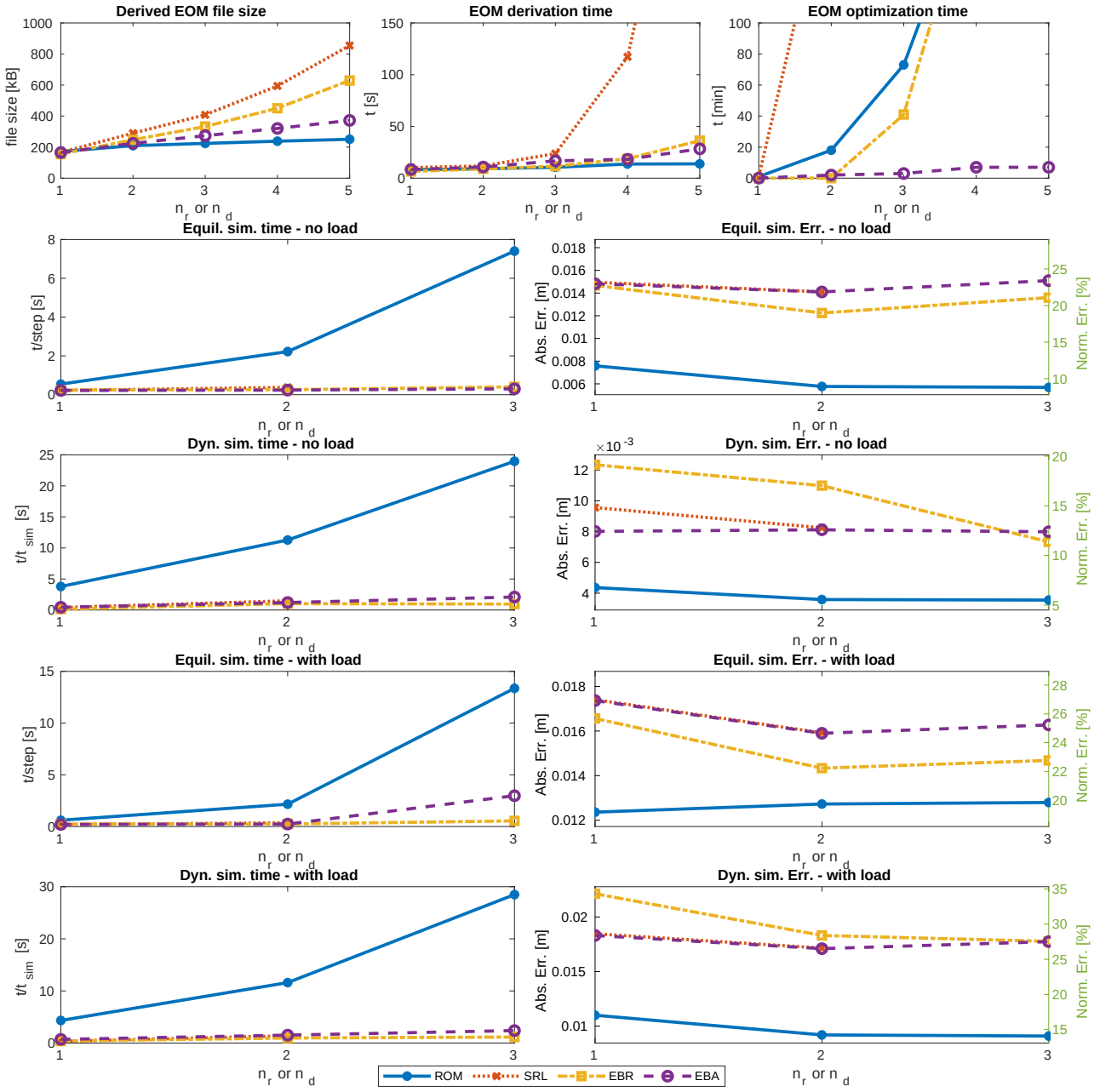


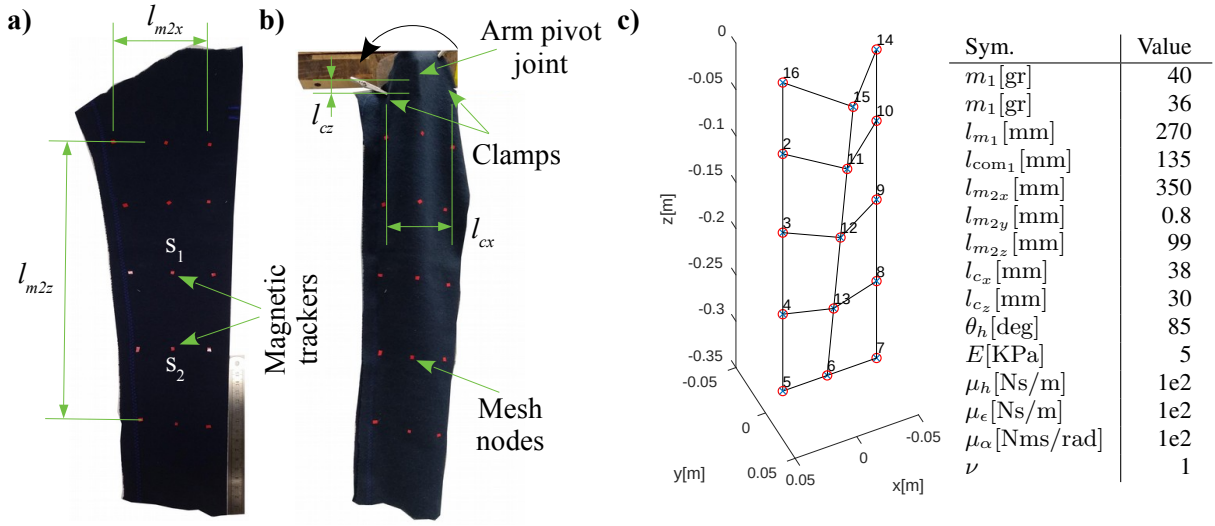
Figure 5. Comparison between computational performance and accuracy of different modeling assumption for a continuum rod (based on experiments with STIFF-FLOP continuum appendage).

edges as in Fig. 6.c. The CAD model is stored in iges format to be imported into the model later. Here, nodes 12 & 13 are equivalent to s_1 & s_2. The fabric is clamped at nodes 13 & 15 to the link at $[\pm l_{c_x}, 0, l_{c_z}]$. Two sets of six constraints are defined to fully fix each of these two nodes to the link.

The fabric mesh is modeled with lumped masses at the CAD-file wireframe nodes that are interconnected with EB beams. The system states are described with absolute states of the masses. This is similar to the EBA model for continuum rods that was discussed in the previous section. Nodes are rigid lumped masses with an equally distributed mass of m_2/n_d. The relation for thin plates is used to derive their second moment of inertia as

$$I_2 = \frac{m_2}{12 n_d}\mathrm{diag}([\, l_{m2_y}^2 + (\frac{l_{m2_z}}{5})^2, (\frac{l_{m2_x}}{3})^2 + (\frac{l_{m2_z}}{5})^2, (\frac{l_{m2_x}}{3})^2 + l_{m2_y}^2\,]). \quad (17)$$

Links are considered as EB ribbons with linear elasticity $K_\epsilon = \mathrm{diag}(a_c[G, G, E])$ and $K_\alpha = \mathrm{diag}([E, E, G]).\mathrm{diag}(J)$ as in the case of continuum rods in previous section. Here, $I = [l_{m2_y}^3 l_{m2_b}, l_{m2_y} l_{m2_b}^3, l_{m2_y}^3 l_{m2_b} + l_{m2_y} l_{m2_b}^3]$ is a 1×3 vector consisting of the cross-section second moments of areas, where $l_{m2_b} = (l_{m2_x} + l_{m2_z})/2$ is the mean width of the ribbons in the x, z-axis directions. The only parameter that needs updating during simulation is u_c, the desired acceleration of the pendulum COM position, either in the $x-$ or $z-$axis direction. To map the nodes motion to the beams



Sym.	Value
m_1[gr]	40
m_1[gr]	36
l_{m_1}[mm]	270
l_{com_1}[mm]	135
$l_{m_{2x}}$[mm]	350
$l_{m_{2y}}$[mm]	0.8
$l_{m_{2z}}$[mm]	99
l_{c_x}[mm]	38
l_{c_z}[mm]	30
θ_h[deg]	85
E[KPa]	5
μ_h[Ns/m]	1e2
μ_ϵ[Ns/m]	1e2
μ_α[Nms/rad]	1e2
ν	1

Figure 6. a) A fabric is cut to form a sleeve around a single link. b) The link forms a passive pendulum with calmped fabric sleeve. The link is fixed with a 1 DOF joint at the top and two magnetic trackers at nodes s_1 & s_2. Red dots are equivalent to the CAD-file nodes. c) CAD-file wireframe of the sleeve in the clamped configuration (as shown by $mesh\_import.m$ module) and the modeling parameters.

deformation map, one way is to define the $xaxis$ vector that defines beam specific frames. This results in better accuracy but cumbersome calculations to derive and simulate the system EOM. An alternative assumption is to have the beams in the local body frames, but assuming that they are initially deformed to reach the second connecting body. This is possible by setting $init = [nan, nan, nan, nan, nan, nan]$ which adopts the beam initial geometry (ϵ_0, α_0) to the system initial condition. We use the second method where 2D continuum geometries are modeled. This results in an almost 10 times decrease in the size of the file storing the system EOM. The inputs for the $TMTDyn$ package to model this setup is as in Tables 9.

Experimental Results & Discussion- Two sets of experiments were carried out to record the fabric deformation at points s_1 and s_2. The static solution for the fabric deformation equilibrium point is challenging to find. The final result is sensitive to the initial guess for the system states that is passed to Matlab's $fsolve$ solver and unrealistic solutions may be found if the guessed states are not close enough to the equilibrium states. It is more accurate if a dynamic simulation is performed, by knowing the system initial conditions, to find the deformed states of the system. To test this, the pendulum was set free from an initial angle $\theta_{h_0} \approx 85$ [deg] to swing under gravity and its joint internal viscous damping (μ_h). The fabric and pendulum motions were recorded using magnetic trackers. The pendulum swings passively under gravity. The recorded data for the pendulum was imported as the desired rigid link trajectory in the model (Fig. 7.left), and the simulation results were compared with experimental data for fabric deformation (Fig. 7.right). The simulation results show good accuracy in predicting node motion in the x-axis direction. The overall absolute error is ≈ 10 [cm] ($\approx 40\%$ normalized error based on l_{z_s}). However, modeling such complex hybrid structures can have other benefits other than capturing the real dynamics with minimum error. For example, it is interesting to observe that the oscillation frequency of the node motion along the z-axis direction is twice the frequency of their oscillation in the x-axis direction. Also, the fabric is slightly deformed toward gravity (decreasing mean value of sensor readings along the $z-$axis) during the pendulum swing in both the simulation and experimental results. A simple model for the hybrid system helps in capturing such basics physics which are useful for system design, control, and observation.

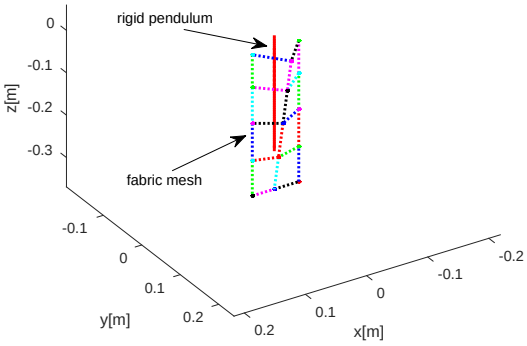
Conclusion and Discussion

Simple, reliable, and easy to use models for hybrid rigid-continuum systems which are suitable for controller and observer design are highly sought in the field of soft robotic research. Current models are either complex with a large state space, which is not suitable for control tasks (e.g. FEM), are not reliable in general unknown cases (e.g. machine learning methods), are not accurate enough (e.g. lumped system method), or are not compatible with energy and Lagrangian based modeling and controller design techniques (the Cosserat beam method). The performance and accuracy of the newly introduced discretized Renda et al. (2018) and reduced-order models Sadati et al. (2018b); Thieffry et al. (2018b) that have promised to overcome these challenges are not yet thoroughly investigated. Additionally, there is no unified framework to implement these methods for hybrid systems with a combination of multi-dimensional rigid and continuum elements.

In this paper, we develop two new models for continuum rods and actuators: a general reduced-order model (ROM), and a discretized model with absolute states and Euler-Bernoulli beam segments (EBA). These models enable us to perform more accurate simulation of continuum manipulators, as well as modeling 2D and 3D continuum geometries, which has so far been missing in similar investigations Renda et al. (2018). Furthermore, a new formulation is presented for a recently introduced discretized model by Renda et al. (2018); Shiva et al. (2018) which is based on Euler-Bernoulli beam theory and relative states

Table 9. $TMTDyn$ package input for the sleeve fabric models clamped to a rigid-link pendulum. Model labes are as in Fig. 6.

$world.g = [0, 0, -g]$; gravity
$body(1).m = m_2$; pendulum rigid link
$joint(1).second = 1$; $joint(1).tr.rot = [2, inf]$; 1 DOF rotation around $y-$axis
$joint(1).dof.init = \theta_{h_0}$; pendulum initial angle
$joint(1).dof.damp.visc = \mu_h$; pendulum joint viscous damping
Import mesh geometry:
$mesh.file\_name = cad.iges$; CAD-file name
$mesh.tol = 1e-3$; geometry import tolerance
$mesh.tr.trans = [0, 0, l_{c_z}]$; mesh geometry initial position/orientation
$mesh.tr.rot = [2, \theta_{h_0}]$;
$mesh.body.m = m_2/n_d$; equally distributed fabric mess over the nodes
$mesh.body.I = I_2$; Describing the mesh absolute DOF with $mesh.joint(1)$:
$mesh.joint(1).tr.trans = [inf, inf, inf]$; masses absolute state as system DOFs
$mesh.joint(1).tr.rot = [0, inf, inf, inf]$; quaternion representation of orientation
Describing the mesh EB beam connections with $mesh.joint(2)$:
$mesh.joint(2).spring.coeff = [diag(K_\epsilon), diag(K_\alpha)]$; linear elasticity of beams
$mesh.joint(2).spring.init = \text{nanones}(1, 6)$; beam initial state from system geometry
$mesh.joint(2).damp.visc = [\mu_\epsilon, \mu_\alpha]$; linear viscous damping
Pendulum control constraint:
$joint(2).second = 1$;
$joint(2).tr2nd.trans = [0, 0, -l_{\text{com}_1}]$;
$joint(2).fixed = 1$; constraining $x-$-axis
$joint(2).control = u_c$; Fabric clamps:
$joint(3).first = 1$;
$joint(3).second = 15$; clamp at node 16 based on mesh file plot
$joint(3).tr.trans = [l_{c_x}, 0, -l_{c_z}]$;
$joint(3).fixed = ones(1, 6)$; fully constrained joint
$joint(4).first = 1$; $joint(4).second = 13$; clamp at node 14 based on mesh file plot
$joint(4).tr.trans = [-l_{c_x}, 0, -l_{c_z}]$;
$joint(4).fixed = ones(1, 6)$;



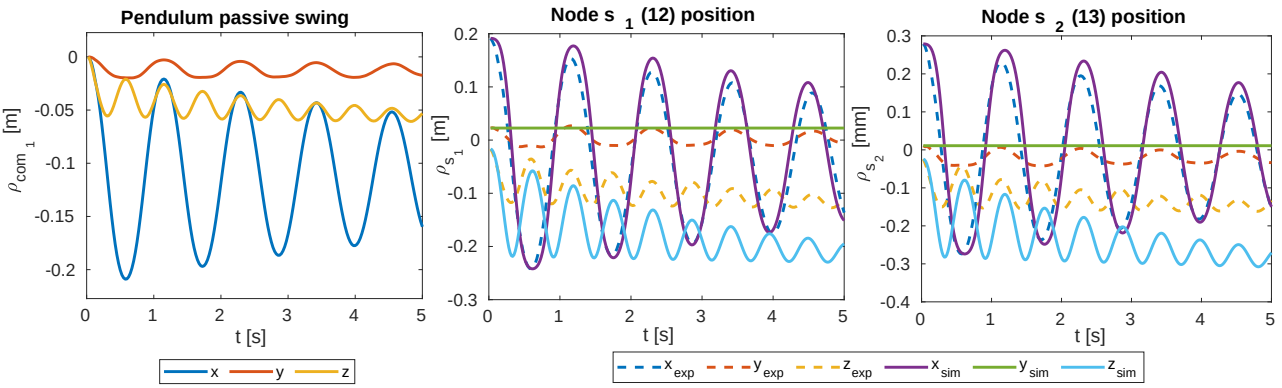


Figure 7. Sample recordings from the two experiments with a fabric worn passive pendulum in comparison with simulation results. Recordings from magnetic trackers attached to: left) pendulum link COM, middle) node s_1 (12 in CAD-file), and right) node s_2 (13 in CAD-file). The simulation results show good accuracy in predicting nodes motion in x-axis direction. The fabric slightly relaxes toward gravity while the pendulum swings. It causes slight decreasing values along the z-axis in the experimental recordings. The oscillation frequency of the motion along z-axis is twice the one for x-axis direction. These physics are captured by the simulation results too.

(EBR). These models are built in a new Matlab software package, called $TMTDyn$ Sadati et al. (2015), to develop a modeling and simulation tool for hybrid rigid-continuum systems. The package performance is boosted using a new High Level Language (HLL) text-based interface, a CAD-file import module, automatic formation of the system EOM for different modeling and control tasks, implementing C-mex functionality for improved performance, and other modules for static and linear modal analysis of a hybrid system.

The package is used herein to compare and validate the aforementioned modeling methods in comparison to experimental results on general motion of a STIFF-FLOP continuum appendage under external loads. We observed higher simulation accuracy (with as little as 8-14% normalized error) and numerical robustness (enabling consideration of material hyper-elasticity) of the ROM model, while EBA is less computationally cumbersome to derive and simulate with near real-time performance. EBR shows high sensitivity to sudden changes in the system actuation inputs and external loads and relatively higher computational cost both in derivation and simulation. The lumped system approach for modeling continuum rods as a hyper-redundant series-rigid-link system (SRL) is investigated as well. SRL has the highest computational cost to derive and optimize the system EOM. This shows the

importance of our investigation on discretized and reduced order methods for modeling hybrid systems.

Finally, the package was successfully tested for modeling a system with 2D continuum geometries. A CAD-file import module was introduced to ease importing simple 2D and 3D continuum geometries in the form of wireframe objects stored in IGES and STL CAD-file formats. A fabric sleeve was fixed to a rigid link pendulum to study cloth dynamics. We observed reliable consistency between our simulation and experimental results. To the best of our knowledge, this study is the first one that investigates low-cost analytical models for deformation of fabrics as a part of a hybrid system. Such models are important for wearable robotics research, and service robots which may deal with fabrics in their tasks. As a part of this case study, we showcased how the package can be used to easily design a nonlinear controller for a hybrid system based on the system nonlinear map (system Jacobian).

We plan to test our package for soft tissue palpation and probing tasks with medical applications. As a result, the package will be benchmarked for a hybrid system with 3D continuum geometry (the tissue sample). A similar study to what presented here for 1D continuum geometries (rods) on the numerical performance and accuracy of different modeling assumptions, is needed to compare different modeling methods for 2D and 3D continuum geometries in a hybrid system. We are working toward a nonsingular formulation for quaternion rotations involving non-unit quaternions. We also plan to further investigate the numerical stability, convergence, and computational efficiency of the introduced methods and the $TMTDyn$ software package in near future. This is possible with performing standard numerical and modeling tests in comparison to analytical solutions or results from high fidelity modeling methods such as FEM. Our final goal is to use this package for hybrid force and position control and geometry, force and stiffness estimation in soft robotic research.

Acknowledgment

This work is supported by the Leverhulme Trust Research Project, "Computing with spiders' web", number RPG-2016-345, granted to H.H. and F.V.; the U.K. Engineering and Physical Sciences Research Council (EPSRC) Grant EP/N03211X/2, granted to T.N.; European Union H2020 project FourByThree code 637095, granted to K.A., and the Royal Academy of Engineering (research fellowship RF1516/15/11), granted to L.R.

References

Bieze TM, Largilliere F, Kruszewski A, Zhang Z, Merzouki R and Duriez C (2018) Finite Element Method-Based Kinematics and Closed-Loop Control of Soft, Continuum Manipulators. *Soft Robotics* DOI:10.1089/soro.2017.0079. URL https://www.liebertpub.com/doi/abs/10.1089/soro.2017.0079.

Blanc L, Delchambre A and Lambert P (2017) Flexible Medical Devices: Review of Controllable Stiffness Solutions. *Actuators* 6(3): 23. DOI:10.3390/act6030023. URL http://www.mdpi.com/2076-0825/6/3/23.

Boyer F (2014) Multibody system dynamics for bio-inspired locomotion: from geometric structures to computational aspects. *Bioinspir. Biomim.* : 23.

Braganza D, Dawson D, Walker I and Nath N (2007) A Neural Network Controller for Continuum Robots. *IEEE Transactions on Robotics* 23(6): 1270–1277. DOI:10.1109/TRO.2007.906248. URL http://ieeexplore.ieee.org/document/4359261/.

Burgner-Kahrs J, Rucker DC and Choset H (2015) Continuum Robots for Medical Applications: A Survey. *IEEE Transactions on Robotics* 31(6): 1261–1280. DOI:10.1109/TRO.2015.2489500. URL http://ieeexplore.ieee.org/document/7314984/.

Cianchetti M, Calisti M, Margheri L, Kuba M and Laschi C (2015) Bioinspired locomotion and grasping in water: the soft eight-arm OCTOPUS robot. *Bioinspiration & Biomimetics* 10(3): 035003. DOI:10.1088/1748-3190/10/3/035003. URL http://stacks.iop.org/1748-3190/10/i=3/a=035003?key=crossref.7e7a029ec68cfb24c606d395db7d7611.

Cianchetti M and Menciassi A (2017) Soft Robots in Surgery. In: *Soft Robotics: Trends, Applications and Challenges*, *Biosystems & Biorobotics*, volume 9, 1 edition. Springer International Publishing. ISBN 978-3-319-46460-2, pp. 75–85. DOI:10.1007/978-3-319-46460-2. URL http://link.springer.com/10.1007/978-3-319-46460-2_10.

Cianchetti M, Ranzani T, Gerboni G, De Falco I, Laschi C and Menciassi A (2013) STIFF-FLOP surgical manipulator: Mechanical design and experimental characterization of the single module. In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. Tokyo, Japan: IEEE, pp. 3576–3581. DOI:10.1109/IROS.2013.6696866.

Cianchetti M, Ranzani T, Gerboni G, Nanayakkara T, Althoefer K, Dasgupta P and Menciassi A (2014) Soft Robotics Technologies to Address Shortcomings in Today's Minimally Invasive Surgery: The STIFF-FLOP Approach. *Soft Robotics* 1(2): 122–131. DOI:10.1089/soro.2014.0001. URL http://online.liebertpub.com/doi/abs/10.1089/soro.2014.0001.

Coevoet E, Morales-Bieze T, Largilliere F, Zhang Z, Thieffry M, Sanz-Lopez M, Carrez B, Marchal D, Goury O, Dequidt J and Duriez C (2017) Software toolkit for modeling, simulation, and control of soft robots. *Advanced Robotics* 31(22): 1208–1224. DOI:10.1080/01691864.2017.1395362. URL https://www.tandfonline.com/doi/full/10.1080/01691864.2017.1395362.

Della Santina C, Katzschmann RK, Bicchi A and Rus D (2018a) Dynamic Control of Soft Robots Interacting with the Environment. Livorno, Italy: IEEE, p. 9.

Della Santina C, Lakatos D, Bicchi A and Albu-Schffer A (2018b) Using Nonlinear Normal Modes for Execution of Efficient Cyclic Motions in Soft Robots. *arXiv:1806.08389 [cs]* URL http://arxiv.org/abs/1806.08389. ArXiv: 1806.08389.

Fras J, Czarnowski J, Macia M, Gwka J, Cianchetti M and Menciassi A (2015) New STIFF-FLOP module construction idea for improved actuation and sensing. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 2901–2906. DOI:10.1109/ICRA.2015.7139595.

Fchslin RM, Dzyakanchuk A, Flumini D, Hauser H, Hunt KJ, Luchsinger RH, Reller B, Scheidegger S and Walker R (2012) Morphological Computation and Morphological Control: Steps Toward a Formal Theory and Applications. *Artificial Life* 19(1): 9–34. DOI:10.1162/ARTL_a_00079. URL https://doi.org/10.1162/ARTL_a_00079.

Gazzola M, Dudte LH, McCormick AG and Mahadevan L (2018) Forward and inverse problems in the mechanics of soft filaments. *Royal Society Open Science* 5(6): 171628. DOI:10.1098/rsos.171628. URL http://rsos.royalsocietypublishing.org/content/5/6/171628.

Godage IS, Branson DT, Guglielmino E, Medrano-Cerda GA and Caldwell DG (2011) Shape function-based kinematics and dynamics for variable length continuum robotic arms. In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, pp. 452–457. DOI:10.1109/ICRA.2011.5979607.

Godage IS, Medrano-Cerda GA, Branson DT, Guglielmino E and Caldwell DG (2016) Dynamics for variable length multisection continuum arms. *The International Journal of Robotics Research* 35(6): 695–722.

Godage IS, Nanayakkara T and Caldwell DG (2012) Locomotion with continuum limbs. *IEEE International Conference on Intelligent Robots and Systems* : 293–298DOI:10.1109/IROS.2012.6385810.

Grazioso S, Di Gironimo G and Siciliano B (2018) A Geometrically Exact Model for Soft Continuum Robots: The Finite Element Deformation Space Formulation. *Soft Robotics* DOI:10.1089/soro.2018.0047. URL https://www.liebertpub.com/doi/full/10.1089/soro.2018.0047.

He L, Herzig N, Lusignan Sd and Nanayakkara T (2018) Granular Jamming Based Controllable Organ Design for Abdominal Palpation. In: *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. pp. 2154–2157. DOI:10.1109/EMBC.2018.8512709.

Hu Y, Liu J, Spielberg A, Tenenbaum JB, Freeman WT, Wu J, Rus D and Matusik W (2018) ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics. *arXiv:1810.01054 [cs]* URL http://arxiv.org/abs/1810.01054. ArXiv: 1810.01054.

Jung J, Penning RS, Ferrier NJ and Zinn MR (2011) A modeling approach for continuum robotic manipulators: Effects of nonlinear internal device friction. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, pp. 5139–5146. URL http://ieeexplore.ieee.org/abstract/document/6094941/.

Kapadia AD, Walker ID, Dawson DM and Tatlicioglu E (2010) A Model-based Sliding Mode Controller for Extensible Continuum Robots. In: *Proceedings of the 9th WSEAS International Conference on Signal Processing, Robotics and Automation*, ISPRA'10. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS). ISBN 978-960-474-157-1, pp. 113–120. URL http://dl.acm.org/citation.cfm?id=1807817.1807840.

Katzschmann RK, Marchese AD and Rus D (2015) Autonomous Object Manipulation Using a Soft Planar Grasping Manipulator. *Soft Robotics* 2(4): 155–164. DOI:10.1089/soro.2015.0013. URL http://online.liebertpub.com/doi/abs/10.1089/soro.2015.0013.

McEvoy MA and Correll N (2018) Shape-Changing Materials Using Variable Stiffness and Distributed Control. *Soft Robotics* 5(6): 737–747. DOI:10.1089/soro.2017.0147. URL https://www.liebertpub.com/doi/full/10.1089/soro.2017.0147.

Mustaza SM, Elsayed Y, Lekakou C, Saaj CM and Fras J (2018) Dynamic Modeling of Fiber-Reinforced Soft Manipulator: A Visco-hyperelastic Material-based Continuum Mechanics Approach. *Soft Robotics* .

Nakajima K, Hauser H, Li T and Pfeifer R (2015) Information processing via physical soft body. *Scientific Reports* 5: 10487. DOI:10.1038/srep10487. URL https://www.nature.com/articles/srep10487.

Nakajima K, Hauser H, Li T and Pfeifer R (2018) Exploiting the Dynamics of Soft Materials for Machine Learning. *Soft Robotics* DOI:10.1089/soro.2017.0075. URL https://www.liebertpub.com/doi/full/10.1089/soro.2017.0075.

Negrut D and Dyer A (2004) Adams/solver primer. *MSC. Software Documentation, Ann Arbor* .

Patern L, Tortora G and Menciassi A (2018) Hybrid SoftRigid Actuators for Minimally Invasive Surgery. *Soft Robotics* 5(6): 783–799. DOI:10.1089/soro.2017.0140. URL https://www.liebertpub.com/doi/full/10.1089/soro.2017.0140.

Renda F, Boyer F, Dias J and Seneviratne L (2018) Discrete Cosserat Approach for Multisection Soft Manipulator Dynamics. *IEEE Transactions on Robotics* : 1–16DOI:10.1109/TRO.2018.2868815.

Renda F, Cianchetti M, Abidi H, Dias J and Seneviratne L (2017) Screw-Based Modeling of Soft Manipulators With Tendon and Fluidic Actuation. *J. Mechanisms Robotics* 9(4): 041012–041012–8. DOI:10.1115/1.4036579. URL http://dx.doi.org/10.1115/1.4036579.

Renda F and Seneviratne L (2018) A Geometric and Unified Approach for Modeling Soft-Rigid Multi-Body Systems with Lumped and Distributed Degrees of Freedom. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 1567–1574. DOI:10.1109/ICRA.2018.8461186.

Rich SI, Wood RJ and Majidi C (2018) Untethered soft robotics. *Nature Electronics* 1(2): 102.

Rucker C (2018) Integrating Rotations Using Nonunit Quaternions. *IEEE Robotics and Automation Letters* 3(4): 2979–2986. DOI:10.1109/LRA.2018.2849557.

Rucker DC, Jones BA and Webster RJ (2010) A Geometrically Exact Model for Externally Loaded Concentric-Tube Continuum Robots. *IEEE Trans Robot* 26(5): 769–780. DOI:10.1109/TRO.2010.2062570. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3091283/.

Rus D and Tolley MT (2015) Design, fabrication and control of soft robots. *Nature* 521(7553): 467–475. DOI:10.1038/nature14543. URL http://dx.doi.org/10.1038/nature14543.

Sadati S (2017) AutoTMTDyn Software Package. URL https://github.com/hadisdt/AutoTMTDyn. Https://github.com/hadisdt/AutoTMTDyn.

Sadati S and Meghdari A (2017) Singularity-free planning for a robot cat free-fall with control delay: Role of limbs and tail. In: *2017 8th International Conference on Mechanical*

and Aerospace Engineering (ICMAE). pp. 215–221. DOI: 10.1109/ICMAE.2017.8038645.

Sadati S, Naghibi S and Naraghi M (2015) An Automatic Algorithm to Derive Linear Vector Form of Lagrangian Equation of Motion with Collision and Constraint. *Procedia Computer Science* 76: 217–222.

Sadati S, Naghibi SE, Althoefer K and Nanayakkara T (2018a) Toward a Low Hysteresis Helical Scale Jamming Interface Inspired by Teleost Fish Scale Morphology and Arrangement. Livorno, Italy: IEEE, p. 7.

Sadati S, Naghibi SE, Shiva A, Noh Y, Gupta A, Walker ID, Althoefer K and Nanayakkara T (2017a) A Geometry Deformation Model for Braided Continuum Manipulators. *Front. Robot. AI* 4. DOI:10.3389/frobt.2017.00022. URL https://www.frontiersin.org/articles/10.3389/frobt.2017.00022/full.

Sadati S, Naghibi SE, Shiva A, Walker ID, Althoefer K and Nanayakkara T (2017b) Mechanics of Continuum Manipulators, a Comparative Study of Five Methods with Experiments. In: *Towards Autonomous Robotic Systems*, volume 10454. Surrey, UK: Springer International Publishing. ISBN 978-3-319-64106-5 978-3-319-64107-2, pp. 686–702. DOI:10.1007/978-3-319-64107-2_56. URL http://link.springer.com/10.1007/978-3-319-64107-2_56.

Sadati S, Naghibi SE, Walker ID, Althoefer K and Nanayakkara T (2018b) Control Space Reduction and Real-Time Accurate Modeling of Continuum Manipulators Using Ritz and RitzGalerkin Methods. *IEEE Robotics and Automation Letters* 3(1): 328–335. DOI:10.1109/LRA.2017.2743100. URL http://ieeexplore.ieee.org/document/8014482/.

Sadati S, Shiva A, Ataka A, Naghibi SE, Walker ID, Althoefer K and Nanayakkara T (2016) A geometry deformation model for compound continuum manipulators with external loading. IEEE. ISBN 978-1-4673-8026-3, pp. 4957–4962. DOI:10.1109/ICRA.2016.7487702. URL http://ieeexplore.ieee.org/document/7487702/.

Sadati S, Sullivan L, Walker I, Althoefer K and Nanayakkara T (2018c) Three-Dimensional-Printable Thermoactive Helical Interface With Decentralized Morphological Stiffness Control for Continuum Manipulators. *IEEE Robotics and Automation Letters* 3(3): 2283–2290. DOI:10.1109/LRA.2018.2805163. URL http://ieeexplore.ieee.org/document/8288847/.

Sadati SMH, Naghibi SE, Shiva A, Zschaler S, Hauser H, Walker I, Althoefer K and Nanayakkara T (2018d) AutoTMTDyn: A Matlab Software Package to Drive TMT Lagrange Dynamics of Series Rigid- and Continuum-link Mechanisms.

Sadati SMH and Williams T (2018) Toward Computing with Spider Webs: Computational Setup Realization. In: *Biomimetic and Biohybrid Systems*, Lecture Notes in Computer Science. Springer, Cham. ISBN 978-3-319-95971-9 978-3-319-95972-6, pp. 391–402. DOI:10.1007/978-3-319-95972-6_43. URL http://link.springer.com/chapter/10.1007/978-3-319-95972-6_43.

Shiva A, Sadati S, Noh Y, Fras J, Ataka A, Wurdemann HA, Hauser H, Walker ID, Nanayakkara T and Althoefer K (2018) Elasticity vs. Hyperelasticity Considerations in Quasi-Static Modelling of a Soft Finger-like Robotic Appendage for Real-time Position & Force Estimation | Request PDF. *Soft Robotic Journal* (accepted, under print). URL https://www.researchgate.net/publication/328583306_Elasticity_vs_Hyperelasticity_Considerations_in_Quasi-Static_Modelling_of_a_Soft_Finger-like_Robotic_Appendage_for_Real-time_Position_Force_Estimation.

Skeel RD and Berzins M (1990) A Method for the Spatial Discretization of Parabolic Equations in One Space Variable. *SIAM Journal on Scientific and Statistical Computing* 11(1): 1–32. DOI:10.1137/0911001. URL http://epubs.siam.org/doi/10.1137/0911001.

Sutar MK and Pathak PM (2017) Bond Graph Modelling and Control of Hyper-Redundant Miniature Robot for In-Vivo Biopsy. In: *Bond Graphs for Modelling, Control and Fault Diagnosis of Engineering Systems*. Springer, Cham. ISBN 978-3-319-47433-5 978-3-319-47434-2, pp. 451–495. DOI:10.1007/978-3-319-47434-2_13. URL https://link.springer.com/chapter/10.1007/978-3-319-47434-2_13.

Takano R, Mochiyama H and Takesue N (2017) Real-time shape estimation of Kirchhoff elastic rod based on force/torque sensor. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE. ISBN 978-1-5090-4633-1, pp. 2508–2515. DOI:10.1109/ICRA.2017.7989292. URL http://ieeexplore.ieee.org/document/7989292/.

Thieffry M, Kruszewski A, Duriez C and Guerra Tm (2018a) Control Design for Soft Robots based on Reduced Order Model. *IEEE Robotics and Automation Letters* : 1–1DOI:10.1109/LRA.2018.2876734. URL https://ieeexplore.ieee.org/document/8496833/.

Thieffry M, Kruszewski A, Guerra TM and Duriez C (2018b) Reduced Order Control of Soft Robots with Guaranteed Stability. In: *European Control Conference ECC18*. Limassol, Cyprus. URL https://hal.archives-ouvertes.fr/hal-01747433.

Thuruthel TG, Ansari Y, Falotico E and Laschi C (2018a) Control Strategies for Soft Robotic Manipulators: A Survey. *Soft Robotics* 5(2): 149–163. DOI:10.1089/soro.2017.0007. URL https://www.liebertpub.com/doi/10.1089/soro.2017.0007.

Thuruthel TG, Falotico E, Renda F and Laschi C (2018b) Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators. *IEEE Transactions on Robotics* : 1–11DOI:10.1109/TRO.2018.2878318.

Till J and Rucker DC (2017) Elastic Stability of Cosserat Rods and Parallel Continuum Robots. *IEEE Transactions on Robotics* 33(3): 718–733. DOI:10.1109/TRO.2017.2664879.

Trivedi D, Lotfi A and Rahn CD (2008) Geometrically Exact Models for Soft Robotic Manipulators. *IEEE Transactions on Robotics* 24(4): 773–780. DOI:10.1109/TRO.2008.924923.

Tunay I (2013) Spatial continuum models of rods undergoing large deformation and inflation. *IEEE Transactions on Robotics* 29(2): 297–307. DOI:10.1109/TRO.2012.2232532. 00000.

Walker ID, Choset H and Chirikjian GS (2016) Snake-Like and Continuum Robots. In: *Springer Handbook of Robotics*. Cham: Springer International Publishing, pp. 481–498. URL http://link.springer.com/10.1007/978-3-319-32552-1_20.

Webster RJ and Jones BA (2010) Design and Kinematic Modeling of Constant Curvature Continuum Robots: A Review. *The International Journal of Robotics Research* 29(13): 1661–1683. DOI:10.1177/0278364910368147. URL http://ijr.sagepub.com/cgi/doi/10.1177/0278364910368147.

Wehner M, Truby RL, Fitzgerald DJ, Mosadegh B, Whitesides GM, Lewis JA and Wood RJ (2016) An integrated design and fabrication strategy for entirely soft, autonomous robots. *Nature* 536(7617): 451–455. DOI:10.1038/nature19100. URL http://www.nature.com/nature/journal/v536/n7617/full/nature19100.html.

Wisse M and Linde RQvd (2007) *Delft Pneumatic Bipeds*. Springer Science & Business Media. ISBN 978-3-540-72807-8. Google-Books-ID: NE8CmUMdG38C.

Appendices

1. High Level Language

Table 10. HLL inputs' symbol, definition, type, dimension, proposed unit in SI, default values, and user options. n_m, n_d, n_t, n_h, n_q, n_r, n_j, n_l are the number of bodies, repeted bodies in a mesh geometry, concecuitive relative transformtions, DOF definitions, states (generalized coordinates), polynomial order in reduced order model, joints/connections, and external loads in a system, respectively. i is a general counter, [ ] is an empty variable, - is for not defined, SBD means "Should Be Defined", ROM is for continuum body Reduced Order Model, MK for joint on main kinematic chain, and $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ are for Natural, Integer and Real number sets, respectiely. Different possible presentations of each variable are discussed in extra rows shaded with dark gray (continued in Table 11).

Symbol	Definition	Type [Dimensions]	Default [Unit]	Input Options
par	control param.s & var.s	struct	-	SBD
$par.anim$	animation control	boolean	1	-, [ ], 0: off, 1: on
$par.mov$	save video control	boolean	0	-, [ ], 0: off, 1: on
$par.derive$	derivation control	boolean	1	-, [ ], 0: off, 1: on
$par.fun$	derived function type	integer	1	-, [ ],...
				1: single Matlab structures for each element,...
				2: separate Matlab functions for each element,...
				3: Matlab functions for each EOM matrix/vector,...
				4: similar to (3) but in C language
$par.mex$	C-mex file generation	boolean	1	-, [ ], 0: off, 1: on
$par.modal$	linear modal analysis	$\in [0, 1, 2]$	1	-, [ ],...
				0: off, 1: use with Matlab function, 2: use C-mex function
$par.equil$	solve equilibrium point	$\in [0, 1, 2]$	1	-, [ ],...
				0: off, 1: use with Matlab function, 2: use C-mex function
$par.dyn$	dynamic simulation	$\in [0, 1, 2]$	1	-, [ ],...
				0: off, 1: use with Matlab function, 2: use C-mex function
$par.nint$	spatial integration steps	integer	50	-, $\mathbb{N}$
$world$	system general properties	struct	-	-, [ ]
$world.g$	gravity vector	vector $[1 \times 3]$	[0,0,0] [m/s^2]	-, [ ], $\mathbb{R}_{[1\times3]}$
$body(i_m)$	inertial elements	struct $[1 \times n_m]$	-	SBD
$body(i_m).m$	body mass	double $[n_d \times 1]$	1e-9 [Kg]	-, [ ],...
				$\mathbb{R}$: in a mesh: same value for all elements,...
				$\mathbb{R}_{[n_d\times1]}$: individual values for each mesh element
$body(i_m).I$	body inertia	double $[3 \times 3 \times n_d]$	1e-9 [Kg.m^2]	-, [ ],...
				$\mathbb{R}_{[3\times3]}$: same value for all elements in a mesh,...
				$\mathbb{R}_{[3\times3\times n_d]}$: individual values for each mesh element
$body(i_m).l\_com$	COM local pos. vec.	double $[n_d \times 3]$	[0,0,0] [m]	-, [ ],...
				$\mathbb{R}_{[1\times3]}$: same value for all elements in a mesh,...
				$\mathbb{R}_{[n_d\times3]}$: individual values for each mesh element
$body(i_m).tip$	tip local pos. vec.	double $[n_d \times 3]$	2×l_com [m]	-, [ ],...
				$\mathbb{R}_{[1\times3]}$: same value for all elements in a mesh,...
				$\mathbb{R}_{[n_d\times3]}$: individual values for each mesh element
$joint(i_j)$	connections & actuation lines	struct $[1 \times n_j]$	-	SBD
$joint(i_j).rom$	ROM details	struct	[ ]	if ROM- $.l$ (SBD),...
				else- -, [ ]
$... .rom.order$	ROM polynomial order	integer	-	if ROM- $\mathbb{N}$ (SBD),...
				else- -, [ ]
$joint(i_j).first$	first connecting body	double $[1 \times 2\text{or}(n_d + 1)]$	[0,0]	-, [ ],...
				$\mathbb{N}$: body number, if MK- single mass, else- attached to single body or to 1st element in a mesh,...
				$[\mathbb{N}, \mathbb{N}\text{or}\mathbb{R}]_{[1\times2]}$: body number $first(1)$, attached to element number $first(2)$ ($\in \mathbb{N}$) in a mesh or...
				or axial length $s = first(2)$ ($\in \mathbb{R}$) in a ROM body,...
				$\mathbb{Z}_{[1\times(n_d+1)]}$: body number $first(1)$, if MK- creates a mesh with n_d bodies,...
				else- attached to elements with number $first(2 : ...)$ in a mesh
$joint(i_j).second$	second connecting body	integer $[1 \times 2\text{or}(n_d + 1)]$	[0,1]	same as $.first$ except...
				$[\mathbb{N}, \mathbb{N}\text{or}\mathbb{R}]_{[1\times2]}$: body number $second(1)$, if MK- creates ROM continuum rod with length $second(2)$ ($\in \mathbb{R}$)
$joint(i_j).tr(i_t)$	local transformations to 1st body... contact point location/orientation	struct $[1 \times n_t]$	-	-, [ ], $.trans$, $.rot$
$... .tr(i_t).trans$	translation vector	double $[1 \times 3]$	[0,0,0][m]	-, [ ],...
				$\mathbb{R}_{[1\times3]}$: set elements to inf to create a DOF,...
				if ROM- $\mathbb{R}_{[1\times3]}$ represents the local strain vector

Table 11. ...continued from Table 10 (continued in Table 12).

Symbol	Definition	Type [Dimensions]	Default [Unit]	Input Options
... $.tr(i_t).rot$	rotation vector	double $[1 \times 2-4]$	[0,0] [-,rad]	-, [ ],...
	$\mathbb{R}_{[1\times2]}$: consist of rotation axis & rotation angle respectively, set the 2nd element inf for rotational DOF,...			
	$\mathbb{R}_{[1\times3]}$: local bendings/twist angle vector (use for VC and with ROM),...			
	$\mathbb{R}_{[1\times4]}$: rotation expressed with a quaternion, set any of the 2nd-4th to inf form a DOF. If so, Q_0 is set to $\sqrt{Q_\rho Q_\rho^\top}$			
Only for spring/damper/input/constraint connection without any DOF:				
$joint(i_j).tr2nd(i_{t2})$	local transformations to 2nd body... contact point location/orientation... (not needed for MK)	struct $[1 \times n_t]$	-	-, [ ], $.trans$, $.rot$
... $.tr2nd(i_{t2}).trans$	translation vector	double $[1 \times 3]$	[0,0,0] [m]	-, [ ],...
	$\mathbb{R}_{[1\times3]}$: Don't set elements inf			
... $.tr(i_{t2}).rot$	rotation vector	double $[1 \times 2-4]$	[0,0] [-,rad]	-, [ ],...
	$\mathbb{R}_{[1\times2]}$: consist of rotation axis & rotation angle respectively, don't set elements inf,...			
	$\mathbb{R}_{[1\times4]}$: rotation expressed with a quaternion, don't set elements inf			
Only for MK joints with DOF:				
$joint(i_j).dof(i_h)$	properties of each inf (DOF)... in all instances of $joint(i_j).tr$	struct $[1 \times n_d]$	-	-, [ ],... $.equal2$, $.init$, $.spring$, $.damp$, $.input$
... $.dof(i_h).equal2$	geometric constrain between DOFs	integer $[n_d \times 3]$	[ ]	-, [ ],...
	$\mathbb{N}_{[1\times2]}$: equal to DOF number $equal2(2)$ of joint number $equal2(1)$,...			
	$\mathbb{N}_{[1\times3]}$: equal to DOF number $equal2(3)$ in mesh number $equal2(2)$ of joint number $equal2(1)$,... not applicable to ROM			
... $.dof(i_h).init$	DOF initial value in simulations	double $[n_d \times \mathbb{N}]$	0 [m]or [rad]	-, [ ],...
	$\mathbb{R}$: initial value for the DOF, if mesh or ROM- same for all of mesh instances or polynomial coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: initial value of DOF for each element in mesh,...			
	$\mathbb{R}_{[1\times\mathbb{N}]}$: initial value of DOF for any number ($\mathbb{N}$) of points along the polynomial in ROM			
... $.dof(i_h).init\_s$	Backbone position for ROM... initial values	double $[1 \times \mathbb{N}]$	$(1:n_r)l/n_r[m]$	-, [ ], $\mathbb{R}_{[1\times\mathbb{N}]}$
... $.dof(i_h).spring$	Spring parallel to the DOF	struct	[ ]	-, [ ],... $.coeff$, $.init$, $.compr$
... $.spring.coeff$	DOF spring coefficient	double $[n_d \times 1]$	0 [N/m] or [Nm/rad]	-, [ ],...
	$\mathbb{R}$: coefficient of the DOF spring and if mesh or ROM- same for all of its instances in the mesh or ROM coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: coefficient of the DOF springs of each element in mesh			
... $.spring.init$	DOF spring initial (resting) value	double $[n_d \times 1]$	0 [m] or [rad]	-, [ ],...
	$\mathbb{R}$: initial value for the DOF spring and if mesh or ROM- same for all of its instances in the mesh or ROM coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: initial value for DOF springs of each element in mesh... set each element to nan to automatically find that initial value based on system DOFs initial condition			
... $.spring.compr$	DOF spring compression ratio	double $[n_d \times 1]$	1	-, [ ],...
	$\mathbb{R} \in (0,1]$: compression ratio for the DOF spring and if mesh or ROM- same for all of its instances or coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: compression ratio for DOF springs of each element in mesh			
... $.dof(i_h).dir$	DOF active direction	integer $[n_d \times 1]$	0	-, [ ],...
	-1: compression only (e.g. soft contact), 1: elongation only (e.g. rope), 0: both directions (e.g. regular spring)			
... $.dof(i_h).damp$	Viscous damping parallel to DOF	struct	[ ]	-, [ ], $.visc$, $.pow$
... $.damp.visc$	DOF viscous damping coefficient	double $[n_d \times 1]$	0 [Ns/m] or [Nms/rad]	-, [ ],...
	$\mathbb{R}$: coefficient of the DOF damping and if mesh or ROM- same for all of its instances in the mesh or ROM coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: coefficient of the DOF damping of each element in mesh			
... $.damp.pow$	DOF viscous damping power	double $[n_d \times 1]$	1	-, [ ],...
	$\mathbb{R}$: power value for the DOF non-Newtonian viscous damping and if mesh or ROM- same for all of its instances in... the mesh or ROM coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: power value for DOF damping of each element in mesh			
... $.dof(i_h).input$	direct load (input) on DOF	double $[n_d \times 1]$	0 [N] or [Nm]	-, [ ],...
	$\mathbb{R}$: value of the DOF input and if mesh or ROM- same for all of its instances in the mesh or ROM coefficients,...			
	$\mathbb{R}_{[n_d\times1]}$: value of the DOF input of each element in mesh			
Only for ROM or spring/damper/input connection without any DOF (not for constrains):				
$joint(i_j).dir$	joint active direction	integer $[n_d \times 1]$	0	-, [ ],...
	-1: compression only (e.g. soft contact), 1: elongation only (e.g. rope), 0: both directions (e.g. regular spring)... ignored for constraints (when $fixed$ field is present)			
$joint(i_j).xaxis$	joint x-axis direction w.r.t. local frame	integer $[n_d \times 3]$	[ ] [m]	-, [ ], $\mathbb{R}_{[n_d\times3]}$
$joint(i_j).spring$	spring link	struct	[ ]	-, [ ], ,... $.coeff$, $.init$, $.compr$
... $.spring.coeff$	joint spring coefficient	double $[n_d \times 1\text{or}6]$	0 [N/m] & [Nm/rad]	-, [ ],...
	$\mathbb{R}$: coefficient of the link spring and if beam or ROM- same for all the directions,...			
	$\mathbb{R}_{[1\times6]}$: stiffness of the beam or ROM direction (strains and curvature/torsion),...			
	$\mathbb{R}_{[n_d\times1\text{or}6]}$: coefficient(s) of the joint/beam springs for each element in mesh			

Table 12. ...continued from Table 11.

Symbol	Definition	Type [Dimensions]	Default [Unit]	Input Options
... $.spring.init$	joint spring initial (resting) value	double $[n_d \times 1\text{or}6]$	0 [m] & [rad]	-, [ ],...
				$\mathbb{R}$: initial value for the link spring and if beam or ROM- same for all the directions,...
				$\mathbb{R}_{[1\times6]}$: stiffness of the beam or ROM directions (strains and curvature/torsion),...
				$\mathbb{R}_{[n_d\times1\text{or}6]}$: initial value(s) for joint/beam springs of each element in mesh,...
				set each element to nan to automatically find that initial value based on system DOFs' initial condition (not for ROM)
... $.spring.compr$	spring compression ratio	double $[n_d \times 1\text{or}6]$	1	-, [ ],...
				$\mathbb{R} \in (0, 1]$: link spring compression ratio and if mesh or ROM- same for all the directions,...
				$\mathbb{R}_{[1\times6]}$: link spring compression ratio for the beam or ROM directions (strains and curvature/torsion),...
				$\mathbb{R}_{[n_d\times1\text{or}6]}$: link spring compression ratio of each element in mesh
$joint(i_j).damp$	Viscous damping connection	struct	[ ]	-, [ ], $.visc$, $.pow$
... $.damp.visc$	link viscous damping coefficient	double $[n_d \times 1\text{or}6]$	0 [Ns/m] & [Nms/rad]	-, [ ],...
				$\mathbb{R}$: coefficient of the link viscous damping and if beam or ROM- same for all the directions,...
				$\mathbb{R}_{[1\times6]}$: damping of the beam or ROM directions (strains and curvature/torsion),...
				$\mathbb{R}_{[n_d\times1\text{or}6]}$: damping coefficient of each element in mesh
... $.damp.pow$	link viscous damping power	double $[n_d \times 1\text{or}6]$	1	-, [ ],...
				$\mathbb{R}$: power value for the link viscous damping and if beam or ROM- same for all the directions,...
				$\mathbb{R}_{[1\times6]}$: damping power value of the beam or ROM directions (strains and curvature/torsion),...
				$\mathbb{R}_{[n_d\times1\text{or}6]}$: power value for the damping of each element in mesh
$joint(i_j).input$	load (input) along the joint	double $[n_d \times 1\text{or}6]$	0 [N] & [Nm]	-, [ ],...
				$\mathbb{R}$: value of the input load along the joint and if beam or ROM- same for all the directions,...
				$\mathbb{R}_{[1\times6]}$: input value of the beam or ROM directions (strains and curvature/torsion),...
				$\mathbb{R}_{[n_d\times1\text{or}6]}$: input value of the joint for each element in mesh
$joint(i_j).fixed$	constraint directions	boolean $[n_d \times 6]$	0	-, [ ],...
				$[1 \times 6]$ boolean vector: constraint directions that are set to 1,...
				$[n_d \times 3]$ boolean matrix: different constraint directions for each element in a mesh,...
				ignored for ROM (when rom filed is present)
$joint(i_j).control$	constraint desired acceleration	double $[n_d \times 6]$	0	-, [ ],...
				0: fixed constraint,...
				$\mathbb{R}$: same constraint control for all the $fixed$ directions,...
				$\mathbb{R}_{[1\times6]}$: different controls for each of the $fixed$ directions,...
				$\mathbb{R}_{[n_d\times6]}$: different constraint controls for each element in a mesh,...
				ignored for ROM (when rom filed is present)
$mesh$	a mesh element	struct $[1 \times 1]$	-	SBD
$mesh.file\_name$	mesh file name	String	-	-
$mesh.body$	body assigning to nodes	struct $[1 \times 1]$	-	same as $body$
$mesh.joint$	joints defining DOFs & links joints	struct $[1 \times 2]$	-	same as $joint$,...
				$joint(1)$: joint element defining each node DOF,...
				$joint(2)$: joint element assigning to each line
$exload(i_l)$	external loads	struct $[1 \times n_l]$	-	SBD
$exload(i_l).exbody$	exerting point body	double $[1 \times (n_d + 1)]$	-	...
				$\mathbb{R}$: exerting body number, on 1st element,...
				$\mathbb{R}_{[1\times2]}$: exerts at $body(2)$ element,...
				$\mathbb{R}_{[1\times(n_d+1)]}$: exerts at $body(2:...)$ elements
$exload(i_l).refbody$	reference body	double $[1 \times (n_d + 1)]$	[0,1]	-, [ ],...
				$\mathbb{R}$: reference body number, uses 1st element for mesh or tip for ROM,...
				$\mathbb{R}_{[1\times2]}$: always uses $refbody(2)$ element for mesh or at $s = refbody(2)$ for ROM,...
				$\mathbb{R}_{[1\times(n_d+1)]}$: uses $refbody(2:...)$ elements for mesh or at $s = refbody(2:...)$ for ROM
$exload(i_l).tr(i_t)$	local transformations to contact... point location/orientation	struct $[1 \times n_t]$	-	-, [ ], $.trans$, $.rot$
... $.tr(i_t).trans$	translation vector	double $[1 \times 3]$	[0,0,0] [m]	-, [ ], $\mathbb{R}_{[1\times3]}$
... $.tr(i_t).rot$	rotation vector	double $[1 \times 2\text{or}4]$	[0,0] [-,rad]	-, [ ],...
				$\mathbb{R}_{[1\times2]}$: consist of rotation axis & rotation angle respectively,...
				$\mathbb{R}_{[1\times4]}$: rotation expressed with a quaternion $[Q_0, Q_\rho]$
$exload(i_l).ftau$	load vector	double $[n_d \times 6]$	[0,0,0,0,0,0] [N,Nm]	-, [ ],...
				$\mathbb{R}_{[1\times3]}$: f_l,...
				$\mathbb{R}_{[1\times6]}$: $[f, \tau]_l$,...
				$\mathbb{R}_{[n_d\times6]}$: separate load for each instance

2. Algorithms

Algorithm 1: Algorithm for system declaration in "*system.m*". See Table 10-12.

Result: System model, simulation & analysis
$par.anim, .mov, .derive, .fun, .mex, .equil, .dyn, .nint$; modeling parameters
if $par.derive$ **then**
  $syms, par.sym$; symbolic structural parameters
end
$par.var$; default structural parameters
$world.g, body, joint, mesh, exload$; system geometry
Default routines:
if $par.derive$ **then**
  $tmt\_eom\_derive(...)$; EOM derivation
end
if $par.equil$ **then**
  $equil(...)$; solve for equilibrium
end
if $par.dyn$ **then**
  $modal(...)$; system linear modal analysis
end
if $par.mov$ **then**
  Set video recording parameters
end
for dynamic simulation intervals **do**
  if $par.dyn$ **then**
    $dyn\_sym(...)$; solve system dynamics
  end
  if $par.anim$ **then**
    $anim(...)$; simple animation & record video
  end
end
if $par.mov$ **then**
  Close recorded video file
end
$post\_proc(...)$; post-processing the results
save('code/results.mat'); save results in a file

Algorithm 2: Algorithm for importing "IGES" file geometries, using Matlab igesToolBox toolbox, in "*mesh_import.m*".

Data: $body, joint, mesh, par$
Result: $mesh\_body, mesh\_joint$; mesh elements
n_m; find current number of *body* instances
$[mesh.body, mesh.joint] = check(par, [], mesh.joint, mesh.joint, [], [])$; Default values for mesh *body* & $joints$
Import mesh file: $makeIGESmex()$; compile the c-files
$lines$; load line elements from IGES-file and transform them based on $mesh.tr$
for $i = 1$:**numel**$(lines)$; find nodes and links **do**
  i_j, i_{jl}; number of DOF & linking $joints$
  $link\_joint(i_{j_l}) = mesh.joint(2)$; assign link joints & default values
  Find new nodes:
  if new node **then**
    $i_p = i_p + 1$; number of points
    $points(i_p, :) = [lines(i).p1(1:3), 0, i_{jl}]$; update point sets
  else
    if new mass **then**
      $i_b = i_b + 1;\ i_j = i_j + 1$; number of bodies and DOF joints
      Set node mass and DOF properties:
      $mesh\_body(i_b) = mesh.body$;
      $mesh\_joint(i_j) = mesh.joint(1)$;
      $mesh\_joint(i_j).first|second = i_b + n_b$; link the $joint$ to the new *body* index
      $points(i_p, 4) = i_b + n_b$; record the node *body* index
    else
      $link\_joint(i_{j_l}).first|second = i_b + n_b$; update linking $joint$ info.
    end
  end
  Similar procedure for the links' 2nd end.
end
$mesh\_joint = [mesh\_joint, link\_joint]$; concatenating the imported DOF & linking $joints$
$plotIGES(lines)$; plot the IGES object and label the nodes

Algorithm 3: Algorithm for system parameters check and default values in "*check.m*".

Data: $par, world, body, joint, exload, mesh$
Result: $s, world, body, joint, exload, par$; System structural parameters
syms s; axial length variable for ROM elements
$mesh\_import(...)$; assign bodies and DOF joints and links to a mesh file
par; set default values for undefined modeling parameters
$world$; set default values for $world.g$
for $i_b = 1 : n_b$;number of bodies **do**
  for $i_j = 1 : n_j$;number of joints **do**
    if $joint.second = i_b$;is in MK **then**
      n_d = **numel**($joint(i_j).first$) - 1; number of mesh (repeated similar) elements
      $body$; set default values for $body$, with dimension n_d
    end
  end
end
for $i_j = 1 : n_j$;number of bodies **do**
  $joint.first, .second, .rom$; set default values
  for $i_b = 1 : n_b$;number of joints **do**
    if $joint.second = i_b$;is in MK **then**
      mainkin = 1; MK index
    end
  end
  if numel($joint.first, .second$) $= 2$; series of links **then**
    $joint(i_j).first, .second(2 : joint(i_j).first, .second(2) + 1) = 1 : joint(i_j).second(2)$;
  end
  $n_d =$**max**([**numel**($joint(i_j).first$), ...
  numel($joint(i_j).second$)]); number of mesh elements
  $joint(i_j).tr.trans, .rot$; Default values
  for $i_1 = 1$:**numel**($joint(i_j).tr$) **do**
    $rotrans = [joint(i_j).tr(i_1).trans, joint(i_j).tr(i_1).rot]$; transformation vector
    for $i_2 = 1$:**numel**($rotrans$) **do**
      if isinf($rotrans(i_2)$); DOFs **then**
        $n_q = n_q + 1$; Number of DOFs
        $joint(i_j).dof(nq)$; DOF Default values, with dimension n_d
        if $joint(i_j).rom$; ROM geometries **then**
          for $i_r = 1 : joint(i_j).second(2)$; ROM order **do**
            $i_{rc} = i_r$;
            if $i_2 < 4$; x,y,z Boundary Condition **then**
              $i_{rc} = i_r + 1$; corrected order
            end
            $S = [S, s_{rc}^{i_r}]$; vector of shape functions
          end
          $joint(i_j).dof(nq).init_s$ default values;
          for $i_r = 1$:**numel**($joint(i_j).dof(n_q).init$); axial positions **do**
            $s_0 = joint(i_j).dof(n_q).init_s(i_s)$;
            $S_0 = [S_0$;**subs**$(S, s, s_0)]$; axial position vector
            if $i_2 = 3$; z boundary condition correction **then**
              $b_0 = [b_0, s_0]$; z boundary condition correction
            else
              $b_0 = [b_0, 0]$;
            end
          end
          $joint(i_j).dof(n_q).init =$**inv**$(S_0)(joint(i_j).dof(n_q).init - b_0)$; inverse problem for initial values
          $joint(i_j).dof(n_q).s = S$; shape function vector
        end
      end
    end
  end
  if $joint(i_j).tr2nd$; links other than MK **then**
    $joint(i_j).tr2nd$; default values
    $joint(i_j).dir, .xaxis, .spring, .damp, .input, .fixed$; default values for axial or beam links and constraints, with dimension n_d
  end
  if $joint(i_j).rom$; ROM elements **then**
    $joint(i_j).spring, .damp, .input$; default values for ROM links
  end
end
for $i_l = 1 : n_l$;number of external loads **do**
  $n_d =$**numel**($exload(i_l).body$); number of mesh elements
  $exload(i_l).exbody, .refbody, .tr, .ftau$; default values, with dimension n_d
end
Handle $mesh$ for $mesh\_import.m$ module:
$mesh.file\_name, .tol, .body, .joint, .tr$; default values & error check

Algorithm 4: Algorithm for deriving TMT EOM in "*tmt_eom_derive.m*". Continues in Algorithm 5.

Data: $par, world, body, joint, exload, mesh$; system geometry and modeling parameters
Result: $q, u, lambda, par$ system states & saving derivations to HDD
$[body\_mesh, joint\_mesh]$ =**mesh_import**$(mesh, par)$; import $mesh$ geometries from a CAD-file
Concatenating all the system $joint$ and $body$ elements
$[s, world, body, joint, exload, par, symbols]$ =**check**$(par, world, body, joint, exload, mesh)$; parameters' check & default values
Deriving state vector and local transformations:
for $i_j = 1 : n_j$ all joints **do**
  for $n_d = 1$:**max**([**numel**$(joint(i_j).first)$,**numel**$(joint(i_j).second)$]) $- 1$ mesh elements **do**
    for $i_3 = 1$:**size**$(joint(i_j).tr)$ number of transformations **do**
      $rotrans$ =**sym**$([joint(i_j).tr(i_3).transjoint(i_j).tr(i_3).rot])$; transformational vector
      for $i_2 = 1$:**numel**$(rotrans)$ **do**
        if $rotrans(i_2) ==$ inf DOF element **then**
          if ROM **then**
            $n_r = joint(i_j).second(2)$; ROM order
          else
            $n_r = 1$;
          end
          for $i_r = 1 : n_r$ **do**
            if $joint(i_j).dof(i_r).equal2$ geometric constraint **then**
              $q, \dot{q}, q_0$; form states and initial value vectors
            else
              Repeat previous states based on the constraint
            end
          end
          $sprdmp(i_q).Tt, .kx, .dl, .init, .in, .k\_mat, .vd\_mat, .dir$; compliant element terms for i_q state: transpose of transformation matrix Tt, elastic term kx, change of state dl, initial value $init$, direct input in, linear stiffness matrix k_mat, linear viscous damping matrix vd_mat, and active direction dir
          $fj\_k, fj\_vd, fj\_in, fj\_sdi, fj\_k\_mat, fj\_vd\_mat$; collecting terms for all states
          if ROM **then**
            $rotrans(i_2) = q(\text{end} - i_r + 1 : \text{end})joint(i_j).dof.s$; from ROM series
            if $i_2 = 3$; z boundary condition **then**
              $rotrans(i_2) = rotrans(i_2) + s$;
            end
          else
            $rotrans(i_2) = q(i_q)$; regular states
          end
        end
      end
      $joint(i_j).Q(i_d).loc = joint(i_j).Q(i_d).loc \times Q(rotrans)$; forming local quaternions
      $joint(i_j).TQ(i_d).loc = joint(i_j).TQ(i_d).loc \star \Xi(rotrans)$; local transformation quaternions pair
    end
    if $joint(i_j).tr2nd$ **then**
      $joint(i_j).Q2nd(i_d).loc, .TQ2nd(i_d).loc$; local rotation & transformation for 2nd end of a joint similar to above
    end
  end
end

Algorithm 5: Continoued from Algorithm 4. Continues in Algorithm 6.

Deriving inertial terms:
for $i_m = 1 : n_m$; number of bodies **do**
  for $i_j = 1 : n_j$; all joints **do**
    if $joint(i_j).second(1) = i_m$; MK joint **then**
      for $i_d = 1$:**numel**$(joint(i_j).first) - 1$; number of mesh elements **do**
        $n_{md} = i_{md} + 1$; all body counter
        if ROM **then**
          Q, TQ =**subs**$(body(joint(i_j).first(1)).Q, .TQ(1).abs, s, joint(i_j).first(2))$; specific axial location
        end
        $joint(i_j).Q, .TQ(i_d).abs$; deriving absolute rotations & transformations
        r_jtips; $[n_{mb} \times 6]$ matrix of bodies both ends positions
        $body(i_m).Q, .TQ(i_d).loc, .abs$; body kinematics
        $mass(i_{md}).v\_com, .omega$; deriving COM velocities
        $mass(i_{md}).M, .T, .Dd, .fg$; deriving EOM inertial terms
        M, T, Dd, fgv; collecting terms for all bodies
      end
      break; jump to next body
    end
  end
end
Compliant links & constraints
for $i_m = 0 : n_m$ number of bodies **do**
  for $i_j = 1 : n_j$ all joints **do**
    if $joint(i_j).second(1) = i_m$ **and** $joint(i_j).tr2nd$) double ended link **then**
      for $i_d = 1$:**max**([**numel**$(joint(i_j).first)$,**numel**$(joint(i_j).second)$]) $- 1$ number of mesh elements **do**
        if ROM **then**
          Q, TQ =**subs**$(body(joint(i_j).first(1)).Q, .TQ, .Q2nd, .TQ2nd(1).abs, s, joint(i_j).first(2))$; specific axial location
        end
        $joint(i_j).Q, .TQ, .Q2nd, .TQ2nd(i_d).abs$; deriving absolute rotations & transformations
        compliant and constraint elements:
        $i_{jd} = i_{jd} + 1$; all joint counter
        $r_k s$; $[n_{jd} \times 6]$ matrix of links' both ends positions
        if numel$(joint(j_count).spring.coeff(n_mesh, :)) = 1$ **then**
          Axial elements:
          l_{j0}; initial length based on input parameters or initial geometry
          $sprdmp(i_{jd}).Tt, .kx, .dl, .init, .in, .k\_mat, .vd\_mat$; compliant element terms for axial element
        else
          Beam elements:
          r_{j0}, Q_{j0}; rotate initial position/orientation vector to beam frame $xaxis$
          $rQ_{j0} = [r_{j0}, Q_{j0}]$; initial position/orientation vector based on input parameters or initial geometry
          $sprdmp(i_{jd}).Tt, .kx, .dl, .init, .in, .k\_mat, .vd\_mat$; compliant element terms for beam
        end
        $sprdmp(i_{jd}).dir$; active directions
        $fj\_k, fj\_vd, fj\_in, fj\_sdi, fj\_k\_mat, fj\_vd\_mat$; collecting terms for all links
        Constraints:
        for compliant directions (3 or 6) **do**
          if $joint(i_{jd}).fixed$; fixed direction **then**
            $i_{cd} = i_{cd} + 1$; constraint counter
            $\lambda, cnst(i_{cd}).T, .D, .in$; constraints Lagrangian and EOM terms
            Tcn, Dcn, Ccn; Collect terms for all constraints
          end
        end
      end
    end
    if ROM **then**
      for $i_d = 1$:**max**([**numel**$(joint(i_j).first)$,**numel**$(joint(i_j).second)$]) $- 1$ number of mesh elements **do**
        $i_{jd} = i_{jd} + 1$; all joint counter
        dr_{jd}, dQ_{jd}; differential position and orientation vectors
        $sprdmp(i_{jd}).Tt, .kx, .dl, .init, .in, .k\_mat, .vd\_mat, .dir$; compliant element terms for beam
        $fj\_k, fj\_vd, fj\_in, fj\_sdi, fj\_k\_mat, fj\_vd\_mat$; collecting terms for all links
      end
    end
  end
end

Algorithm 6: Continoued from Algorithm 5.

External loads:
for $i_l = 1 : n_l$; number of bodies **do**
  for $i_d = 1$:**numel**$(exload(i_j).exbody) - 1$; number of mesh elements **do**
    $n_{ld} = i_{ld} + 1$; all body counter
    $exload(i_j).Q(i_d).loc, exload(i_j).TQ(i_d).loc$; local rotations & transformations
    if $exload(i_l).exbody =$ ROM **then**
      $Q, TQ =$**subs**$(body(joint(i_j).first(1)).Q, .TQ(1).abs, s, joint(i_j).first(2))$; specific axial location
    end
    $exload(i_j).Q, .TQ(i_d).abs$; deriving absolute rotations & transformations
    r_ef; $[n_{mb} \times 3]$ matrix of loads' exerting point positions
    $ftau\_abs$; $[1 \times 6]$ transforming load vector from $exload(i_l).refbody$ frame to reference frame
    $loads(i_{md}).Tt, ftau$; deriving terms for external load action
    $Ttef, ftau\_ef$; collecting terms for all external loads
  end
  break; jump to next body
end
$save\_func(...)$; pass derivation for optimization and forming Matlab, C-mex & C functions

Algorithm 7: Algorithm for optimizing derived EOM and saving on HDD, in "*save_func.m*" & "*save_mex.m*".

Data: EOM terms
Result: Saved functions on HDD
$vars\_mex =$ [**zeros**(1,**numel**$(par.sym)$),**zeros**(1,**numel**$([q\lambda u])$), 0]; function input format
switch $par.fun$ **do**
  case 1: Matlab function for $body, sprdmp, cnst, loads$ structures
  case 2: Matlab function for separate terms in $body, sprdmp, cnst, loads$
  case 3: Matlab function for collective terms
  case 4: C function for collective terms
end
$save\_mex(...)$; generate C-mex files from above functions. The file has the same structure as above.

Algorithm 8: Algorithm for forming full system EOM and saving on HDD, in "*save_eom_mex.m*". Similar algorithms are used for static equilibrium (without temporal differential terms in the same file) and modal analysis (calling functions for $fj\_k\_mat, fj\_vd\_mat, k\_mat, vd\_mat$ in $save\_modal\_mex.m$ file) codes.

Data: $par, analysis$; system parameters and analysis type
Result: Updated par & Saved function for EOM on HDD
Initialize: remove figure handles and symbolic parameters from par
Write to file write the following in a Matlab & C-mex function file
  function $\dot{X} = EOM(t, X, par)$ function file header: $X = [q, \lambda, \dot{q}, \dot{\lambda}]$
  $[z, par] = int\_midstep(t, X, par)$; update z & par in each integration mid-step if needed
  $f_{k_q}, f_{v_q}, f_{u_q}$; call functions for states compliance
end
for $i = 1 : n_m$; all masses **do**
  Write to file
    for $s = s_0 : par.rom.mass(i) - s_0$; **do**
      $[M, T_m, D, f_\sigma] = massF_i(par.var, X, s)$; call function
      $\bar{M} = \bar{M} + T_m^\top \, M \, T_m \, \mathrm{d}s$; inertial terms
      $d_m = d_m + T_m^\top \, M \, (-D * \dot{q} + f_\sigma) \, \mathrm{d}s$;
    end
  end
end
for $i = n_q + 1 : n_{jd}$; all compliant links **do**
  Write to file
    for $s = s_0 : par.rom.mass(i) - s_0$; **do**
      $[T_k, k, v, u] = sprdmpF_i(par.var, X, s)$; call function
      $w_v = w_v + T_k^\top \, v \, \mathrm{d}s$;
      $w_k = w_k + T_k^\top \, k \, \mathrm{d}s$;
      $w_u = w_u + T_k^\top \, u \, \mathrm{d}s$;
    end
  end
end
for $i = 1 : n_l$; all external loads **do**
  Write to file
    $[T_l, f_l] = loadsF_i(par.var, X, 0)$; call function
    $w_l = w_l + T_l^\top * f_l$;
    end
  end
end
for $i = 1 : n_c$; all constraints **do**
  Write to file
    $[T_{ci}, D_c, in] = cnstF_i(par.var, X, 0)$; call function
    $T_c = [T_c, T_{ci}]$; $d_c = [d_c, -D_c \, \dot{q} + u_c]$; **end**
  end
end
Forming EOM to file
Write:
  $\bar{M} = [\bar{M}, T_c^\top; T_c, \mathbf{zeros}(n_c, n_c)]$;
  $d = [(d_m + f_{k_q} + f_{v_q} + f_{u_q} + w_k + w_u + w_v + w_l); d_c]$;
  $[\ddot{q}, \ddot{\lambda}] = \mathbf{inv}(\bar{M}) \, d$;
  $\dot{X} = [\dot{q}, \dot{\lambda}, \ddot{q}, \ddot{\lambda}]$;
end
Write EOM function to file "$EOM\_eq.m$" & "$EOM.m$";

Algorithm 9: Algorithm for static equilibrium, in "*equil.m*".

Data: par; system parameters
Result: X_{0_e}, par; equilibrium point & updated system parameters
$nqF(par.var)$; recall model states, initial values, & number of different elements
X_0; states initial guess
$save\_eom\_mex(par, 0)$; generate static equilibrium EOM
if $par.equil == 1$; **then**
  $equilfun = @EOM\_eq$; use Matlab function
else
  $equilfun = @EOM\_eq\_mex$; use C-mex function
end
$X_{0_e} = \mathbf{fsolve}(equilfun, X_0)$; use Matlab "fsolve" to find equilibrium point

Algorithm 10: Algorithm for linear modal analysis, in "*modal.m*".

Data: X, par; system states & parameters
Result: $\Phi_\omega, M_\omega, K_\omega, V_\omega, \omega, \eta_\omega$; terms of modal state space & natural frequencies
$nqF(par.var)$; recall model states, initial values, & number of different elements
X; current states
$save\_modal\_mex(par)$; generate function for modal analysis
if $par.modal == 1$; **then**
  $modalfun = @EOM\_modal$; use Matlab function
else
  $modalfun = @EOM\_modal\_mex$; use C-mex function
end
$[\bar{M}, \bar{K}, \bar{V}] = modalfun(X)$; EOM linearized terms in state space
$[\Phi_\omega, \omega] = -$**eig**(**inv**$(\bar{M})\ \bar{K})$; undamped modal analysis
$M_\omega, K_\omega, V_\omega, \omega, \eta_\omega$; Solve for natural frequencies, EOM terms in modal space, and modal damping ratio
Plot the mode shapes

Algorithm 11: Algorithm for dynamic simulation, in "*dyn_sim.m*".

Data: X_0, par; system initial states & parameters
Result: t, X, par; dynamic simulation results & updated system parameters
$nqF(par.var)$; recall model states, initial values, & number of different elements
X_0; form states initial vector if not provided from static equilibrium analysis
$save\_eom\_mex(par, 1)$; generate EOM for dynamic simulation
if $par.dyn == 1$; **then**
  $odefun = @EOM$; use Matlab function
else
  $odefun = @EOM\_mex$; use C-mex function
end
$[t, X] =$**ode15s** or **ode113**$(odefun, X_0)$; use Matlab ODE numerical solvers for dynamic simulation

Algorithm 12: Algorithm for plotting a simple animation, in "*anim.m*".

Data: t, X, par; simulation time steps, system states & parameters
Result: $r\_anim, rjtip, rks, par$; structure & matrix variable for links' tip position, & updated system parameters
$par.n\_mass_anim, par.n\_ks\_anim$; determine number of bodies and compliant links
if ROM link presents **then**
  $par.n\_animpoints = 50$; points plotting along a link
else
  $par.n\_animpoints = 2$;
end
for t; for all time steps **do**
  for $par.n\_animpoints$; plotting points **do**
    $rksF\_mex(...), rjtipF\_mex(...)$; bodies and links positions **for** $par.n\_mass_anim, .n\_ks\_anim$; all bodies and links **do**
      $r\_anim.mass, .sprdmp$; store the links' tips in a structure
      $rjtip, rks$; store the links' tips in a separate matrices
    end
  end
end
Determine plotting area
for t; for all time steps **do**
  plot3$(par.n\_mass_anim)$; plot bodies with solid line
  plot3$(par.n\_ks\_anim)$; plot compliant links with dashed line
end

3. STIFF-FLOP Experimental Setup

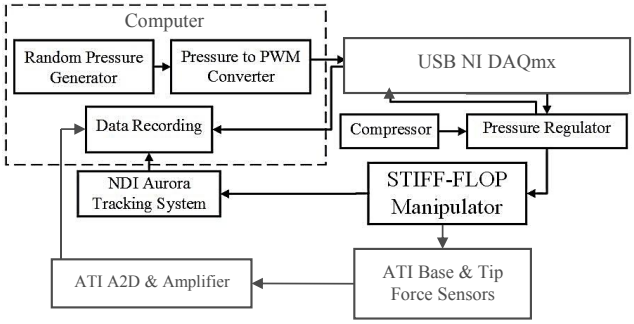


Figure 8. Control diagram for a STIFF-FLOP module.

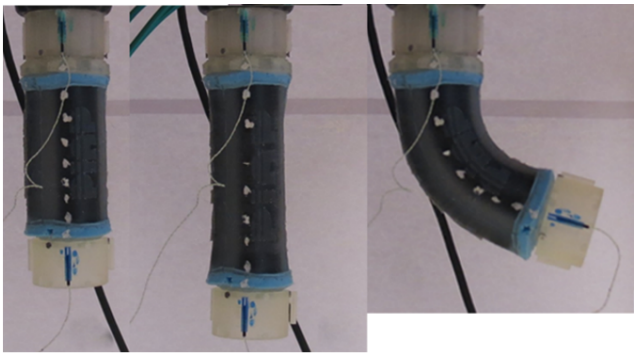


Figure 9. Axial and side bending of a pneumatically actuated STIFF-FLOP continuum appendage. The system has 3 DOFs via three internal actuation chambers.