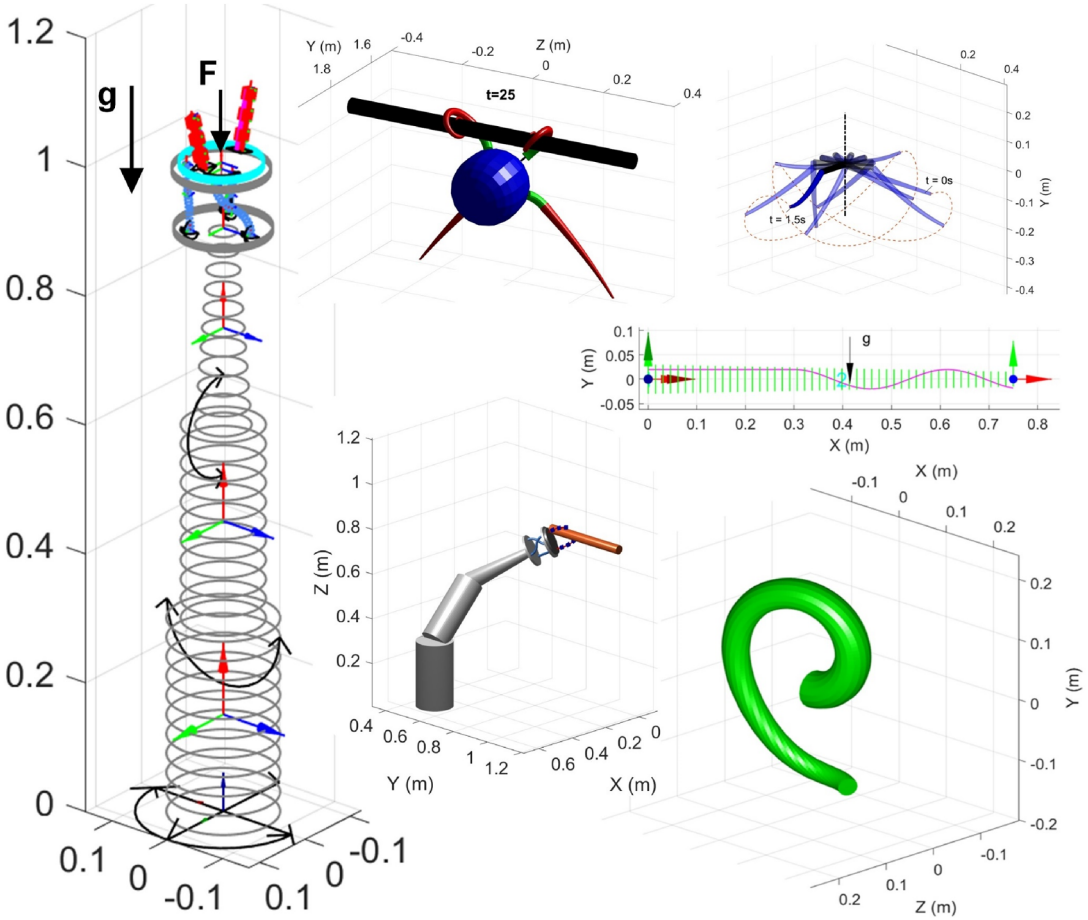
SoRoSim – a MATLAB® Toolbox for Hybrid Soft and Rigid Robotics Based on the Geometric Variable-strain Approach

v3.0

Anup Teejo Mathew, Ikhlas Ben Hmida and Federico Renda

March 16, 2022



SoRoSim

Soft Robot Simulator

Introduction

SoRoSim (Soft Robot Simulator) is a user-friendly MATLAB toolbox that uses the Geometric Variable Strain (GVS) approach to provide a unified framework for the modeling, analysis, and control of soft, rigid, and hybrid robots. The toolbox can be used to analyze open-, closed- and branched structures and allows the user to model many different exernal loading and actuation scenarios. Soft link should be modelled as a Cosserat rod, a 1D, slender rods accounting for bend, twist, stretch, and shear. While the ridig link can have any shape. MATLAB GUI assists creation of links, their assembly, assignment of DoFs, and the application of external and actuation forces.

Using a hybrid soft-rigid manipulator example, this document demonstrates how the user can create SorosimLink and SorosimLinkage class elements and perform static and dynamic analysis of the manipulator. The document also presents a detailed description of the properties and methods of SorosimLink, Twist, and SorosimLinkage classes. We write the Properties, Methods, and MATLAB command window syntaxes in `typewriter font` to distinguish them from the rest.

Contents

1 Overview of the SoRoSim Toolbox

The SoRoSim toolbox was developed using MATLAB as it provides a vast library of functions for mathematical computations and allows the developer to accurately model and analyze the mechanical behaviour of the system. Users can also make use of the various add-ons and toolboxes to analyze different aspects of the robotics systems developed by the toolbox.

The programming approach used in creating this toolbox is Object-Oriented Programming (OOP). This approach entails software design around data, or objects, rather than functions and logic. In OOP, the developer groups 'Objects' with similar attributes or requiring similar manipulations together under a 'Class.' The user can then insert data in the form of 'Properties' associated with that specific object. Class 'Methods' can perform manipulation or analysis of the object. OOP comes with many advantages as it provides a well-structured map of the program and allows easy access and adjustment to object-specific data. OOP also allows reusability, once an object is defined and saved in MATLAB's workspace the user can perform multiple manipulations without having to redefine the object, which makes this approach well-suited for programs that are large, complex and actively updated or maintained [1].

The Geometric Variable Strain (GVS) approach was introduced by Renda et al.[2] in statics and Boyer et al.[3] in dynamics. It is based on a strain parametrization of the soft links represented by Cosserat rods. This model is also geometrically-exact and generalizes the geometric theory of rigid robotics [4] to hybrid systems of soft and rigid links with multidimensional joints, externally applied point and distributed forces, as well as distributed actuation forces [5]. An extension of the GVS approach to closed-chain and branched soft robots was also presented in [6].

A new nested quadrature computational scheme is used to implement the toolbox. The toolbox is validated by comparing with published results. The SoRoSim toolbox consists of three main classes: SorosimLink, Twist, and SorosimLinkage. These classes work together to form a powerful analytical tool which employs the GVS model to statically and dynamically analyze soft, rigid, and hybrid robotic structures. The next sections will give guidelines for new users.

2 Toolbox Structure

The SoRoSim toolbox consists of three classes, namely SorosimLink, Twist, and SorosimLinkage, that work together to form a powerful simulation tool that employs the geometrical variable strain model to analyze soft, rigid, and hybrid robotic structures. Figure 1 shows the overview of SorosimLink and SorosimLinkage creation. This section will give details on the Properties and Methods of SoRoSim classes.

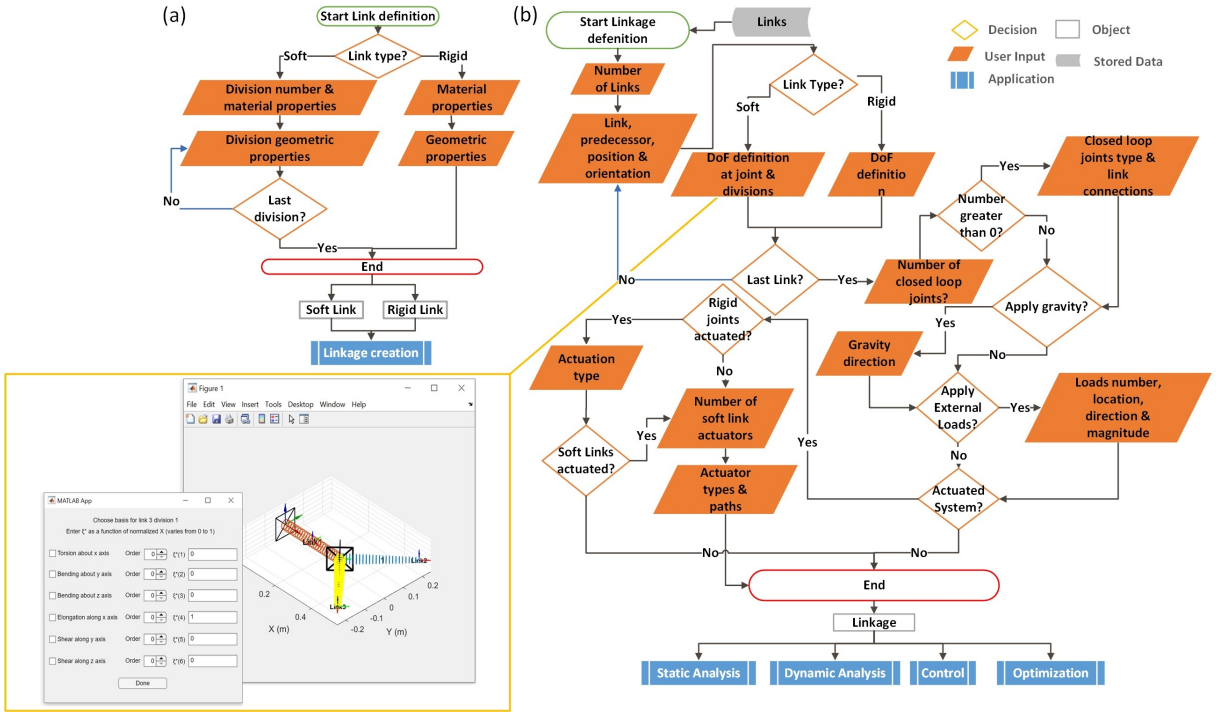


Figure 1: Toolbox workflow: (a) Creation of SorosimLink, (b) Creation of SorosimLinkage. The inset shows a sample GUI.

2.1 The SorosimLink Class

A link is considered as a rigid or a soft body which is attached to a lumped joint at its starting position. The SorosimLink class allows the user to construct links that could later be combined to form manipulators. The SorosimLink class makes use of 'Properties' to define link parameters. Some properties are user-defined, some are default values, while others are automatically computed. An element of the SorosimLink class is an OOP object derived from handle. It is always passed as reference, i.e., MATLAB allows a function to make changes to the variable in-place without the need for a copy. We divide the properties associated with this class into three categories: General Properties, Geometric Properties, and Material Properties (Table. 1).

Type	Name	Description
General Properties	`jointtype`	Jointtype of the link: 'R' for revolute, 'P' for prismatic, 'H' for helical, 'U' for universal, 'C' for cylindrical, 'A' for planar, 'S' for spherical, 'F' for free motion, and 'N', for fixed joint
	`linktype`	Type of link: 's' for soft body, 'r' for rigid body
	`CS`	Cross sectional shape: 'C' for circular and 'R' for rectangular
	`npie`	1 for rigid link and 1+number of divisions for soft link
	`nGauss`	Cell element with of number of Gauss quadrature points for each division (including the start and end point) of a soft link
	`Xs`	Cell element of corresponding Gauss quadrature points computed between 0 and 1
	`Ws`	Cell element of corresponding weights
Geometric Properties	`lp`	Cell element of piece-wise lengths of each divisions of the soft link [m]
	`L`	Total length of the link [m]
	`r`	Radius of the rigid link or cell element of the same for all divisions of the soft link. Saved as a function of `X1 = X/L` for rigid link or `X1 = X/lp{division_number}` for soft divisions [m]
	`h`	Height of the rigid link or cell element of the same for all divisions of the soft link (as a function of `X1`) [m]
	`w`	Width of the rigid link or cell element of the same for all divisions of the soft link (as a function of `X1`)[m]
	`a`	Semi-major axis as a function of `X1` [m]
	`b`	Semi-minor axis as a function of `X1` [m]
	`gi`	Transformation from the joint to the center of mass for ridig link to center of area for soft link
	`gf`	Transformation from center of mass to tip reference point for ridig link from center of area for soft link to tip reference point
Material Properties	`E, Poi, G, Eta`	Young's modulus [Pa], Poisson's ratio [-], shear modulus [Pa], and material damping [Pa.s] of the soft link
	`Kj`	Joint Stiffness Matrix
	`Rho`	Density of the link $[kg/m^3]$
	`Ms`	Screw inertial matrix of the rigid link or cell element of cross sectional screw inertial matrix for all divisions of the soft link
	`Es`	Cell element of cross sectional screw stiffness matrix for all divisions of the soft link
	`Gs`	Cell element of cross sectional screw damping matrix for all divisions of the soft link

Table 1: Properties of SorosimLink Class

Properties that defines the general characteristic of the link are grouped together as General Properties. These properties include the type of lumped joint (`jointtype`), body type (`linktype`), cross sectional shape (`CS`), number of pieces on the link (`npie`), and Gauss quadrature parameters (`nGauss`, `Xs`, and `Ws`). For a rigid link, `npie` is 1, which corresponds to the joint, while for a soft link, `npie` is equal ro the total number of divisions plus 1. `nGauss{division_number}` indicates the total number of points on a soft division at which the toolbox performs computations (significant points). It includes the start and endpoints of the division in addition to the Gauss quadrature points. `Xs` and `Ws` are corresponding

normalized positions and weights.

Properties that are related to the geometry of the link fall under Geometric Properties. Properties such as the piece-wise length of soft link divisions (`lp`), total length of the link (`L`), radius (`r`) of a link with a circular cross section, width (`w`) and height (`h`) of a link with a rectangular cross section, semi-major (`a`) and semi-minor (`b`) axis length of a link with an ellipsoidal cross section, and initial (`gi`) and final transformation (`gf`) matrices are geometric properties of the SorosimLink class. Figure 2 shows the definitions of `gi` and `gf` for rigid and soft links. The SorosimLink class saves the radius, height, and width as functions of `X1`, where `X1` varies from 0 to 1. For a rigid link, the Geometric Properties are saved as numerical values, while for the soft link they (with the exception of `L`) are saved as cell elements with a numerical value for each division of the link. For instance, to access the radius of a rigid link, the syntax is `L1.r` ↵ while for a soft link, it is `L1.r{division_number}` ↵, where `L1` is the name of the SorosimLink class element.

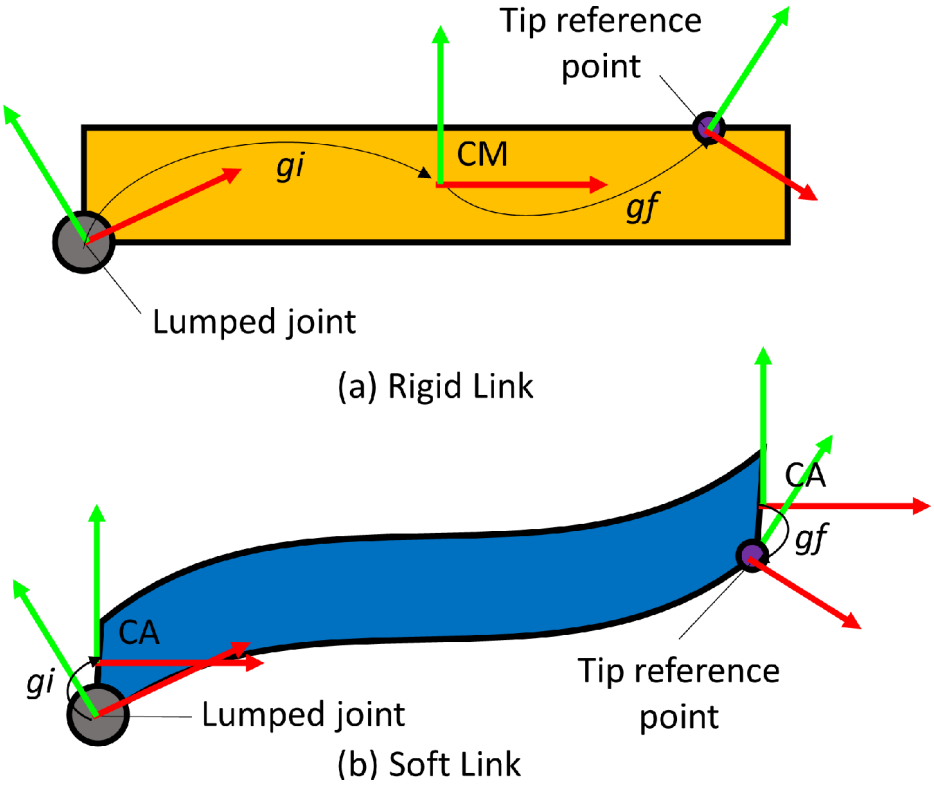


Figure 2: Definition of `gi` and `gf` for rigid (a) and soft (b) links

Material Properties include the employed material's Young's modulus (`E`), Poisson's ratio (`Poi`), density (`Rho`), and material damping (`Eta`). For a rigid link, only the value of `Rho` is significant. Using these values along with the Geometrical Properties of the link, the program computes the shear modulus (`G`), the screw inertia matrix (`Ms`), screw stiffness matrix (`Es`), and screw damping matrix (`Gs`). These derived quantities are also saved as the Material Properties of the link. For a rigid link, `Ms` represents the screw stiffness matrix of the whole body ($\mathcal{M}$). In contrast, for a soft link, `Ms`, `Es`, and `Gs` represent the cross-sectional inertia ($\bar{\mathcal{M}}$), stiffness (Σ), and damping matrices (Υ) computed at every significant point of each division. The SorosimLink class saves them as cell elements. The property called (`Kj`) saves the

value of joint stiffness. For a 1D joint it is a 1×1 scalar, for 2D joints it is 2×2 matrix and so on.

The Link class also includes Plot Properties that are used to plot the link. They include the number of cross sections (per division) of the link (`n_l`), the number of radial points (`n_r`, applicable only for a circular cross section), and the color of the link (`color`). The user can choose a color that follows the MATLAB color code. The default values of `n_l`, `n_r` and `color` are '10', '18', and 'b' (blue) respectively.

2.2 The Twist Class

We define a 'piece' as a joint or a division of the soft link within the open-chain manipulator. The Twist class is used to specify the allowable degrees of freedom (DoFs) of each piece. For lumped joints (rigid joints), the Twist class creates a base matrix (`B`, as shown in Table. 2) with '1 's' and '0 's', where '1' indicates that a particular DoF is allowed. `B` is a property of the twist class.

DoF	Joint	Base ($\boldsymbol{B}$)	Generalized Force (τ_i)
0	Fixed	-	-
1	Revolute (about x axis)	$\begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\p\\0\\0\end{bmatrix}$	$\boldsymbol{B}_{\boldsymbol{\xi}_i}^T \boldsymbol{\mathcal{F}}_{J_i}$
	Prismatic (along x axis)		
	Helical (along x axis)		
2	Universal (constrained about x axis)	$\begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix} \& \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix}$	$\boldsymbol{B}_{\boldsymbol{\xi}_i}^T \boldsymbol{\mathcal{F}}_{J_i}$
	Cylindrical (about x axis)	$\begin{bmatrix}1&0\\0&0\\0&0\\0&1\\0&0\\0&0\end{bmatrix}$	${}^i\boldsymbol{S}_i^T \boldsymbol{\mathcal{F}}_{J_i}$
3	Planar (x-y plane)	$\begin{bmatrix}0&0&0\\0&0&0\\1&0&0\\0&1&0\\0&0&1\\0&0&0\end{bmatrix}$	${}^i\boldsymbol{S}_i^T \boldsymbol{\mathcal{F}}_{J_i}$
	Spherical	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&0\\0&0&0\\0&0&0\end{bmatrix}$	${}^i\boldsymbol{S}_i^T \boldsymbol{\mathcal{F}}_{J_i}$
6	Free Motion	$\boldsymbol{I_6}$	${}^i\boldsymbol{S}_i^T \boldsymbol{\mathcal{F}}_{J_i}$

Table 2: Examples of base matrices of lumped joints

For a division of a soft link (distributed system), there are six modes of deformation: torsion about x axis, bending about y axis, bending about z axis, elongation about x axis, shear about y axis and shear about z axis. We use a property of Twist class, namely `Bdof`, to specify the allowable modes. `Bdof` is a 6×1 array of '1 's' and '0 's', where 1 indicates that a particular deformation mode is allowed and 0 indicates that it is not. Another property called `Bodr`, which is also a 6×1 array, specifies the order of polynomials that are used to fit the strains corresponding to each deformation mode. Using `Bdof` and `Bodr`, the Twist class computes the base matrix (`B`) of a soft division at every Gauss quadrature points (`Xs`) of the division. Some of the most commonly used Base matrix for a distributed system are shown in Table. 3.

Beam	Base ($\boldsymbol{B}$)	DoF
Linear Spring	$\begin{bmatrix} 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 \\ 1 & X & .. & X^{N_4} \\ 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 \end{bmatrix}$	$N_4 + 1$
Planar Constant Curvature	$\begin{bmatrix} 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 1 & X & .. & X^{N_3} & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 1 & X & .. & X^{N_4} \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \end{bmatrix}$	N_3+N_4+2
Inextensible Constant Curvature	$\begin{bmatrix} 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 1 & X & .. & X^{N_2} & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 1 & X & .. & X^{N_3} \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \end{bmatrix}$	$N_1 + N_2 + N_3 + 3$
Kirchhoff-Love	$\begin{bmatrix} 1 & X & .. & X^{N_1} & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 1 & X & .. & X^{N_2} & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 1 & X & .. & X^{N_3} & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 1 & X & .. & X^{N_4} \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \end{bmatrix}$	$N_1 + N_2 + N_3+N_4+4$

Table 3: Examples of base matrices of distributed joints (soft piece). N_1, N_2, N_3, and N_4 are orders of polynomials used to fit the strains associated with different deformation modes. The generalized force ((τ_i)) of a distributed joint is given by, $\int_0^{L_i} \boldsymbol{B}_{\boldsymbol{\xi}_i}^T \boldsymbol{\mathcal{F}}_{a_i} dX_i$

.

Properties namely `B_Z1` and `B_Z2` are base matrices that are evaluated at points associated with the fourth-order Zanna's collocation method. The Twist class saves the reference strain vector corresponding to a piece ($\boldsymbol{\xi}^*(X)$) as a property namely `xi_starfn`. Another property called, `xi_star`, saves the value of $\boldsymbol{\xi}^*(X)$ at points associated with Gauss quadrature and Zanna's collocation method. Finally, the total number of joint coordinates required to specify the state of a piece is saved as a property called `dof`.

The Twist class is called during the creation of the SorosimLinkage class element. The SorosimLinkage class collects the Twist class elements of each piece and saves them together as a Twist vector. Hence, the Twist class elements are accessed through the SorosimLinkage class element. Table. 4 summarizes the properties of Twist class.

Name	Description
`Bdof`	6×1 array specifying the allowable DoFs of a soft piece: 1 if allowed 0 if not
`Bodr`	6×1 array specifying the order of allowed DoFs: (0 if constant, 1 if linear, 2 if quadratic,...)
`B`	Base matrix calculated at lumped joints or array of of base matrices computed at every significant points of a soft piece
`B_Z1`, `B_Z2`	Base matrices calculated at points corresponding to the fourth order Zanna's collocation method
`xi_starfn`	Reference strain vector of the piece as a function of X
`xi_star`	6×1 reference strain vector at the lumped joint or column array of (6×3) matrices with reference strain vectors computed at Gauss quadrature and Zanna collocation points
`dof`	Total degrees of freedom of the piece

Table 4: Properties of Twist Class

2.3 The SorosimLinkage Class

The SorosimLinkage class allows the user to combine previously defined SorosimLink class elements into a open-, closed-, and branched- chain systems (linkage). The SorosimLinkage class allows the user to apply external or actuation loads to the linkage. There are in-built functions to apply external loads such as gravity and point force/moment. The user can also add custom external wrenches on the linkage to simulate other types of external forces such as fluid interaction or contact forces.

The class consists of in-built functions to actuate all kinds of rigid joints as well as soft links. The user can actuate the joints by specifying the value of joint coordinates as a function of time or by entering the joint wrenches. Hence, the toolbox can simulate a mixed forward-inverse model for linkages with rigid joints. Built-in functions are available on the toolbox to allow users to add thread-like actuators on a soft link. We can use them to simulate actuation forces due to cables embedded inside the soft link, pneumatic actuation, and soft gripper actuation. In addition, the user can apply custom actuator strengths or custom actuation wrenches on the soft links to simulate closed-loop control and other types of actuators.

We use the SorosimLinkage class to perform static and dynamic simulations. During the simulation, we perform the computations of soft pieces after normalizing the piece. The normalization process starts with transforming the length of a soft piece to 1 unit. Consequently, for the computation of the piece, we scale all quantities with length dimension using the original length of the piece. Hence, internally, the toolbox performs the computation of a soft piece using scaled quantities. Once the simulation is completed, the joint coordinates of soft pieces are scaled back to their original dimensions. The normalization of soft pieces avoids poorly scaled matrices such as the base matrix. This allows faster static solutions and stable dynamic simulations.

We divide the properties of the SorosimLinkage class into five main categories: General Properties, Closed-loop Properties External Force Properties, Actuation Properties, and Elastic Properties. (Tables 5 and 6).

Type	Name	Description
General Properties	N	Total number of links in series
	ndof	Total degrees of freedom of linkage
	nsig	Total number of points at which the computation is performed (significant points)
	VLinks	Vector of all unique links (obtained from user input)
	LinkIndex	$(N \times 1)$ array of indices corresponding to each links. ith Link = Tr.VLinks(LinkIndex(i))
	CVTwists	Cell element of Twist vectors for each link
	iLpre	$(N \times 1)$ array corresponding to the Link index of the Link to which the ith Link is connected
	g_ini	$(4N \times 4)$ Fixed initial transformation matrices of Links wrt to the tip of its previous link
	q_scale	$(ndof \times 1)$ array of multipliers for each joint coordinate
Closed-loop Properties	nCLj	Total number of closed loop joints
	iACL	$(nCLj \times 1)$ array corresponding to the index of Link A
	iCLB	$(nCLj \times 1)$ array corresponding to the index of Link B
	VTwistsCLj	$(nCLj \times 1)$ array of Twist vectors corresponding to each closed loop joint
	gACLj	$(nCLj \times 1)$ cells of fixed transformation from the tip of Link A to the close loop joint
	gBCLj	$(nCLj \times 1)$ cells of fixed transformation from the tip of Link B to the close loop joint
	CLpre-compute	Struct element which contains pre-computed BpCLj (cell element of constrain basis), i_sigA (array of significant index corresponding to A), i_sigB (array ofsignificant index corresponding to B), and nCLp (total number of constraints)
	T_BS	Baumgarte stabilization constant. Lower the value stricter the constrain.

Table 5: General and Closed-loop Properties of SorosimLinkage Class

2.3.1 General Properties

The General Properties of the SorosimLinkage class include numbers such as, the total number of links (N), total number of pieces within the linkage (ntot), total number of joint coordinates to represent the state of the linkage (ndof), and the total number of significant points on the linkage at which the toolbox performs computations in a static or a dynamic simulation (n_sig). We consider joints, the center of mass of rigid links, and the significant points of soft links as the significant points of the linkage.

The SorosimLinkage class arranges the SorosimLink class elements and the Twist class elements as vectors (class arrays) and cell vectors and saves them as General Properties, namely VLinks and CVTwists. VLinks is a vector of all unique SorosimLinks. LinkIndex correspond to the the index of i^{th} link. Hence, the i^{th} SorosimLink is VLinks(LinkIndex(i)). The properties of the SorosimLink and the Twist classes can be accessed through these properties. For example, to access the reference strain vector of the second division of the first link, the syntax is S1.CVTwists1(3).xi_starfn (the first element of S1.CVTwistsi is dedicated to the Twist of the lumped joint of i^{th} link) while to access the color of the second link of

the linkage, the syntax is `S1.VLinks(LinkIndex(2)).color`, where `S1` is the name of the SorosimLinkage class element.

The property (`iLpre`) corresponds to the link number of the Link to which the i^{th} Link is connected. (see Figure 3) The initial transformation matrix (`g_ini`) specifies the location and orientation of the starting frame of each link wrt its previous link given by `iLpre`.

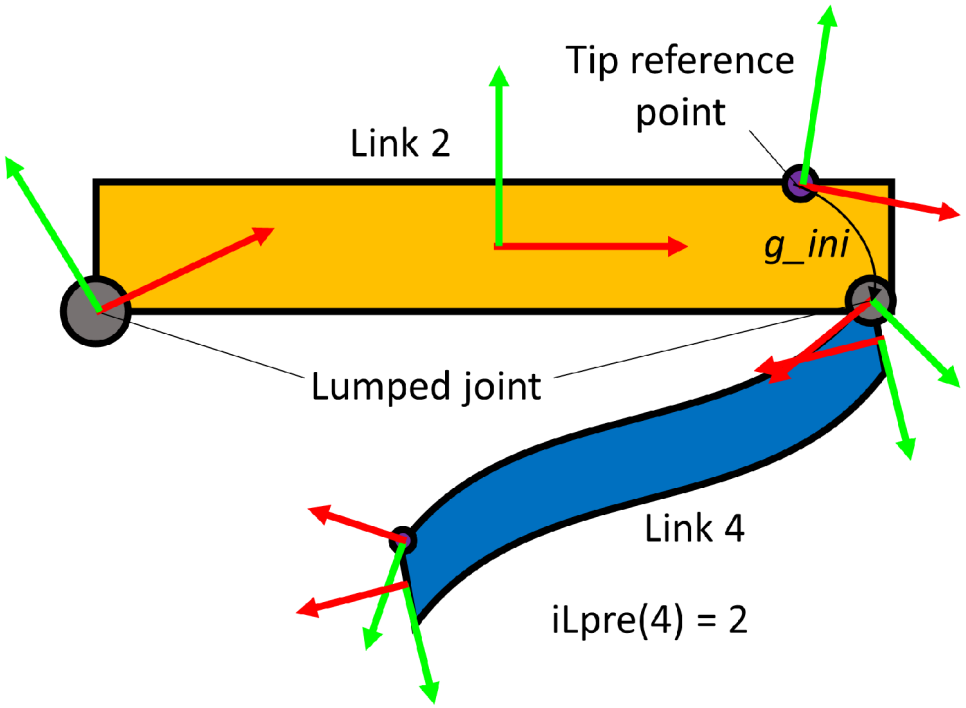


Figure 3: Link assembly: Definition of `g_ini` and `iLpre`

Finally, the scaling factor of joint coordinates (`q_scale`), which is used to obtain the original joint coordinates after the static or the dynamic simulation.

2.3.2 Closed-loop Properties

We group all properties used to define closed-loop joints on the linkage as Closed-Loop Properties. A closed-loop joint (CLJ) is a joint between the tip reference points of two links. The tip of link A is connected to that of link B via a rigid joint. The schematic of the CLJ is shown in Figure 4.

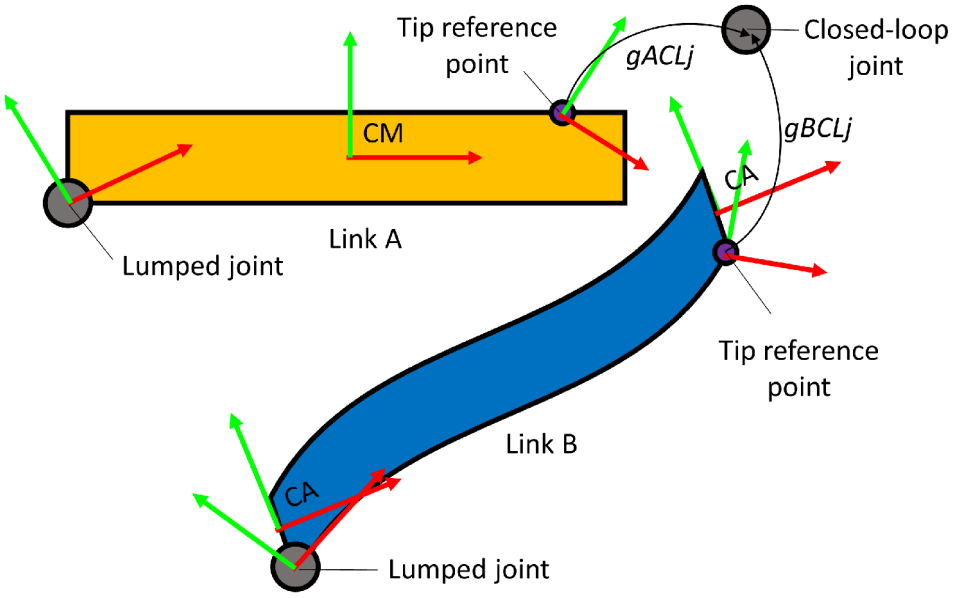


Figure 4: Closed-loop joint: Definition of `gACLj` and `gBCLj`

The SorosimLinkage class use properties called `nCLj` to define the number of closed-loop joints. Properties called `iACL` and `iCLB` indicate indices of links A and B. `VTwistsCLj` contains twist information about the CLJ. `gACLj` and `gBCLj` are transformation matrices from the tip reference points of A and B to the closed-loop joints. `CLprecompute` is a struct element that saves precomputed quantites such as `BpCLj` (cell element of constrain basis), `i_sigA` (array of significant index corresponding to A), `i_sigB` (array ofsignificant index corresponding to B), and `nCLp` (total number of constraints).

External Force Properties	`Gravity`	logical 1 if gravity is present, 0 if not
	`G`	Gravity wrench vector ($[0, 0, 0, g_x, g_y, g_z]^T$, where g is gravity vector)
	`PointForce`	Value is 1 if the linkage is subject to point force/moment, 0 if not
	`Follower-Force`	logical 1 if point force/moment is a follower force (local frame) and 0 if it is wrt global frame
	`np`	Total number of point forces/moments
	`Fp_loc`	Cell element of piece numbers corresponding to the location of point forces/moments
	`Fp_vec`	Cell element of wrench vectors
	`CEFP`	Value is 1 if custom external force is present, 0 if not (default value: 0)
	`M_added`	Added Mass (used for external force that depend on $\dot{\eta}$)
Actuation Properties	`Actuated`	Value is 1 if the linkage is actuated, 0 if not
	`nact`	Total number of actuators
	`Bqj1`	Generalized actuation matrix of one dimensional joints
	`n_jact`	Total number of lumped joint actuators
	`i_jact`	Index of links who's joints are actuated
	`i_jactq`	Index of joint coordinates of all active joints
	`Wrench-Controlled`	$(n\_jact \times 1)$ array of '1 's' and '0 's', where 1 indicates that the joint is controlled by wrench and 0 indicates that it is controlled by the joint coordinate(s).
	`n_sact`	Total number of soft link actuators
	`dc`	$(n\_sact \times N)$ cell element of local cable positions $[0, y_p, z_p]^T$ at significant points of all active soft divisions
	`dcp`	Cell element of space derivatives of dc (with respect to x)
	`Sdiv`, `Ediv`	$(n\_sact \times N)$ cell element of division numbers corresponding to the start and the end position of an actuator on a link
	`Inside`	Value is 1 if the actuator is fully inside the linkage, 0 if not
	`CablePoints`	Struct with cell elements (`Cy_fn` and `Cz_fn`) of parameterized functions corresponding to the y and z coodinates of the cable
	`CAP`	Value is 1 if custom actuation is present, 0 if not (default value: 0)
	`CAS`	Value is 1 for applying a custom actuator strength (default value: 0)
Elastic Properties	`K`	Generalized stiffness matrix of the linkage
	`Damped`	Value is 1 if the soft links are elastically damped 0 if not
	`D`	Generalized damping matrix of the linkage
	`CP1, CP2, CP3`	Constant custom properties of linkage

Table 6: External Force, Actuation, and Elastic Properties of SorosimLinkage Class

2.3.3 External Force Properties

We organize all properties used to apply external forces or moments on the linkage as External Force Properties. The SorosimLinkage class use properties called `Gravity` and `G` to define distributed external forces such as gravity. We can enable or disabled the gravity by typing `S1.Gravity=1` ↵ or `S1.Gravity=0` ↵. To update the magnitude and direction of the gravity, we can change the gravity wrench vector (`G`).

Similar to `Gravity`, the SorosimLinkage class uses a property called `PointForce` to

enable or disable the application of point forces and moments. `np`, `Fp_loc`, and , `Fp_vec` are SorosimLinkage class properties that determine the number of point forces, cell element of piece numbers corresponding to the application of point wrenches, and cell element with the magnitudes of point wrenches. Note that if we choose a rigid piece, the program will apply the point wrench at the center of mass provided by the `cx` of the SorosimLink. While, if we choose a soft piece, the program will apply the same at the tip of the piece. Hence, the user needs to divide the soft link appropriately to adjust the point of application of the wrench. During the creation of a SorosimLinkage, the program prompts users to enter the value of these properties.

Apart from these, the user can also apply a customized external wrench to the linkage. To do that, firstly, the user needs to enable the custom external force property called, `CEFP` by typing, `S1.CEFP=1` ↵. The default value of `CEFP` is 0. Secondly, the user needs to modify a MATLAB function given by 'CustomExtForce.m' inside a folder, namely 'Custom.' The MATLAB function CustomExtForce.m lets the user apply a customized external wrench as a function of the SorosimLinkage class element, time, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{g}$, $\boldsymbol{J}$, $\boldsymbol{\eta}$, and $\dot{\boldsymbol{J}}$. Here, $\boldsymbol{g}$, $\boldsymbol{J}$, $\boldsymbol{\eta}$, and $\dot{\boldsymbol{J}}$ are transformation matrices, Jacobians, screw velocities, and time derivatives of Jacobians evaluated at every significant point. Using the MATLAB function, the user can apply a point wrench on rigid links and a distributed wrench on soft links. The user can also modify a property called added mass (`M_added`) to simulate an external wrench that depends on $\dot{\boldsymbol{\eta}}$. In section 6, we show an example of how we can use the customized external force to model the underwater propagation of a flagellate soft robot.

2.3.4 Actuation Properties

We classify all properties used to define actuators on the linkage as Actuation Properties. The SorosimLinkage class uses a property called `Actuated` to enable or disable actuation. Another property called, `nact` saves the number of actuators on the linkage. To define the actuation of lumped joints, the class uses properties such as `Bqj1`, `n_jact`, `n_jact`, `i_jactq`, and `WrenchControlled`. `Bqj1` is the generalized actuation matrix of all joints with a single degree of freedom. The property, `n_jact` saves the total number of joint actuators. `i_jact` saves the indices of links whose joints are actuated and `i_jactq` saves the indices of joint coordinates corresponding to the lumped joints that are actuated. The property called, `WrenchControlled` is a $(n\_jact \times 1)$ array that determines whether a wrench or joint coordinates controls the rigid joint. Using this property the toolbox can simulate mixed forward-inverse model for linkages with rigid joints.

The SorosimLinkage saves the number of soft actuators using a property called `n_sact`. To compute the generalized actuation matrix for the soft piece the SorosimLinkage class uses properties such as, `dc`, `dcp`, `Sdiv`, `Ediv`, and `Inside`. `dc` is a $(n\_sact \times N)$ is cell-matrix whose elements are cell arrays for a particular actuator and a link. They include the local positions of the actuator path in the format $[0, y_p, z_p]^T$ at every significant point of each soft division. `dcp` saves the derivative of local cable positions in a similar way. The user can choose the actuator path from the list: 'constant', 'oblique', 'helical', and 'custom'. `Sdiv` and `Ediv` are also cell matrices with the data of division numbers corresponding to the start and the end position of an actuator on a link. The property called, `Inside` determines whether the actuator is fully embedded inside the linkage or is partially outside. The value of this property is 1 if the actuator is fully inside and 0 if it is not. The toolbox uses the later case to simulate the actuation of soft grippers. We show an example of the simulation of a soft gripper in section 3.1. The toolbox estimates the value of all these parameters based on user inputs during the SorosimLinkage creation.

The toolbox prompts for the value of the actuator strength before the static or dynamic simulation. However, the user may apply a customized actuator strength on joint and soft actuators. To do this, the user needs to enable the custom actuator strength by changing the value of property called `CAS` to 1. `CAS` has a default value of 0. Subsequently, the user needs to modify a MATLAB function given by 'CustomActuatorStrength.m' inside the folder, namely 'Custom.' In the file, the user has access to the SorosimLinkage class element, time, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{g}$, $\boldsymbol{J}$, $\boldsymbol{\eta}$, and $\dot{\boldsymbol{J}}$. This opens up the possibility of simulating a closed-loop control for the linkage.

In addition to this, the user can also add a customized actuation on soft links. An Actuation Property called `CAP` is used to enable (value 1) or disable (value 0) customized actuation. `CAP` has a default value of 0. The process is similar to adding the customized external wrench. The user needs to modify a MATLAB function given by 'CustomActuation.m' inside the folder, namely 'Custom.' Similar to the 'CustomExtForce.m' and 'CustomActuatorStrength.m', in the file 'CustomActuation.m', the user has access to the SorosimLinkage class element, time, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{g}$, $\boldsymbol{J}$, $\boldsymbol{\eta}$, and $\dot{\boldsymbol{J}}$. Using the MATLAB function and these inputs, the user can simulate other kinds of soft actuators.

2.3.5 Elasticity and Other Properties

Constant parameters such as generalized stiffness matrix and generalized damping matrix are pre-computed and saved as properties of the SorosimLinkage, namely, `K` and `D`. The class uses a property called `Damped` to enable (value 1) or disable (value 0) the elastic damping of soft links. The default value of `Damped` is 1.

We reserve three customizable properties, `CP1`, `CP2`, and `CP3`, which can be used to save constant properties of the linkage for applying a custom external force or actuation. The SorosimLinkage class also has two other properties, namely `CablePoints` and `PlotParameters`. The class uses `CablePoints` for plotting the actuator path, and `PlotParameters` is a struct element of MATLAB with parameters for changing the output plot of the simulation. The user can access the plot parameters by typing `S1.PlotParameters` ↵ and can change the value of its fields according to the formats displayed.

2.3.6 Methods of SorosimLinkage Class

Methods of a class use the properties of the class element to generate meaningful results. The SorosimLinkage class uses methods `statics` and `dynamics` to perform static and dynamic simulations. Once we create a SorosimLinkage (S1), the syntax for using these methods are `S1.statics` and `S1.dynamics`.

Apart from these, we pack the SorosimLinkage class with several other methods. For instance, we can use a method called `FwdKinematics` to obtain the transformation matrices at every significant point of the linkage for a given value $\boldsymbol{q}$. The syntax is `S1.FwdKinematics(q)`, where `q` is a joint coordinate vector. The output is `n_sig` transformation matrices that are arranged as column arrays. Similary, to obtain the generalized coriolis matrix the syntax is `S1.GeneralizedCoriolisMatrix(q,qd)`, where `q` and `qd` are $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. `S1.plotq(q)` generates the shape of the open-chain manipulator for a given value of $\boldsymbol{q}$. We provide the complete list of methods of the SorosimLinkage class in Table. 7.

Name and syntax	Description
`S.FwdKinematics(q)`	To get the transformation matrix at every significant points (arranged as column array)
`S.Jacobian(q)`	To get the Jacobian at every significant points (arranged as column array)
`S.Jacobiandot(q,qd)`	To get the derivative of Jacobian at every significant points (arranged as column array)
`S.ScrewStrain(q)`	To get the screw strain at every significant points (arranged as column array)
`S.ScrewVelocity(q,qd)`	To get the screw velocity at every significant points (arranged as column array)
`S.findD`	To compute and get the generalized damping matrix
`S.findK`	To compute and get the generalized stiffness matrix
`S.ActuationMatrix(q)`	To get the generalized actuation matrix (customized actuation is not included)
`S.GeneralizedMass-Matrix(q)`	To get the generalized mass matrix
`S.GeneralizedCoriolis-Matrix(q,qd)`	To get the generalized Coriolis matrix
`S.GeneralizedExternal-Force(q)`	To get the generalized external force (customized external force is not included)
`S.dynamics`	For the dynamic simulation
`S.statics`	For the static equilibrium simulation
`S.plotq0`	To plot the free body diagram of the linkage
`S.plotq(q)`	To plot the state of the linkage for a given value of the joint coordinate
`S.plotqqd(t,qqd)`	To get dynamic simulation video output for a given t (time array) and qqd (array of joint coordinates and their time derivatives)

Table 7: Methods of the SorosimLinkage Class. S is the name of the SorosimLinkage.

3 Using the Toolbox

This section will help guide the user through the installation process and getting started with the toolbox by going through the steps to create a hybrid closed-chain manipulator as well as running two simulations, a static and dynamic one.

3.1 Toolbox Installation

The toolbox is freely available for users and can be downloaded through GitHub as well as MATLAB Central file exchange. The toolbox can be accessed on GitHub through this link: https://github.com/Ikhlas-Ben-Hmida/SoRoSim The GitHub Repository "SoRoSim" will be displayed. Click on the "Code" button and select "Download Zip" from the drop-down menu as shown in figure 5 below.

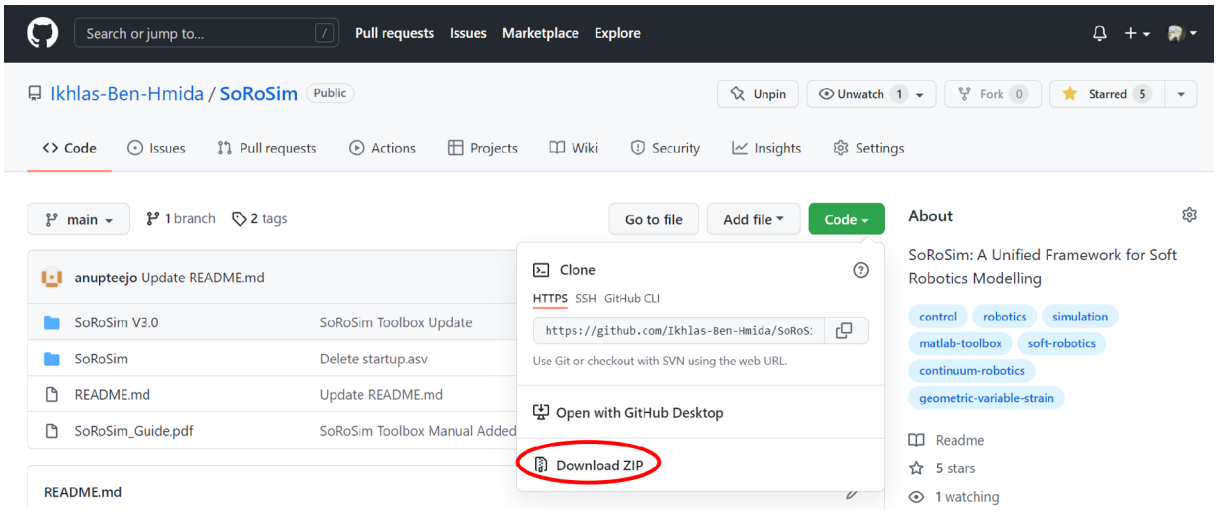


Figure 5: Repository page on GitHub. Files shown in the image may differ slightly depending on the current version in the repository.

Extract the folders within the "SoRoSim-main.zip" to a location of your choice and open the SoRoSim Folder in MATLAB. Run the startup.m file to ensure all necessary files are added to your path. To download the toolbox from MATLAB Central's File Exchange follow this link: https://www.mathworks.com/matlabcentral/fileexchange/83038-sorosim The toolbox can be downloaded by clicking the download button and selecting either "Toolbox" or "Zip" from the drop-down menu as shown in figure 6 below.

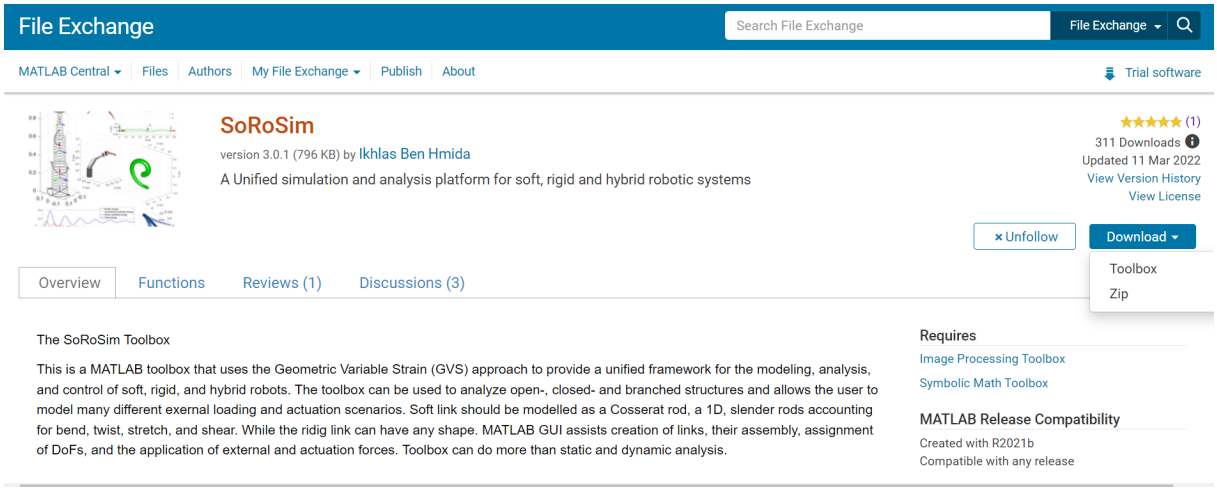


Figure 6: Download page on MATLAB central file exchange.

You can either select "Zip", and follow the same instructions used for the GitHub zip file download Or choose to download as a "Toolbox". For the latter case, a "SoRoSim.mltbx" file will be downloaded. Once the download is complete, run the file. This will open the "Add-on Manager" window, shown in figure 7, after agreeing to the license agreement the toolbox will be automatically installed and ready to use.

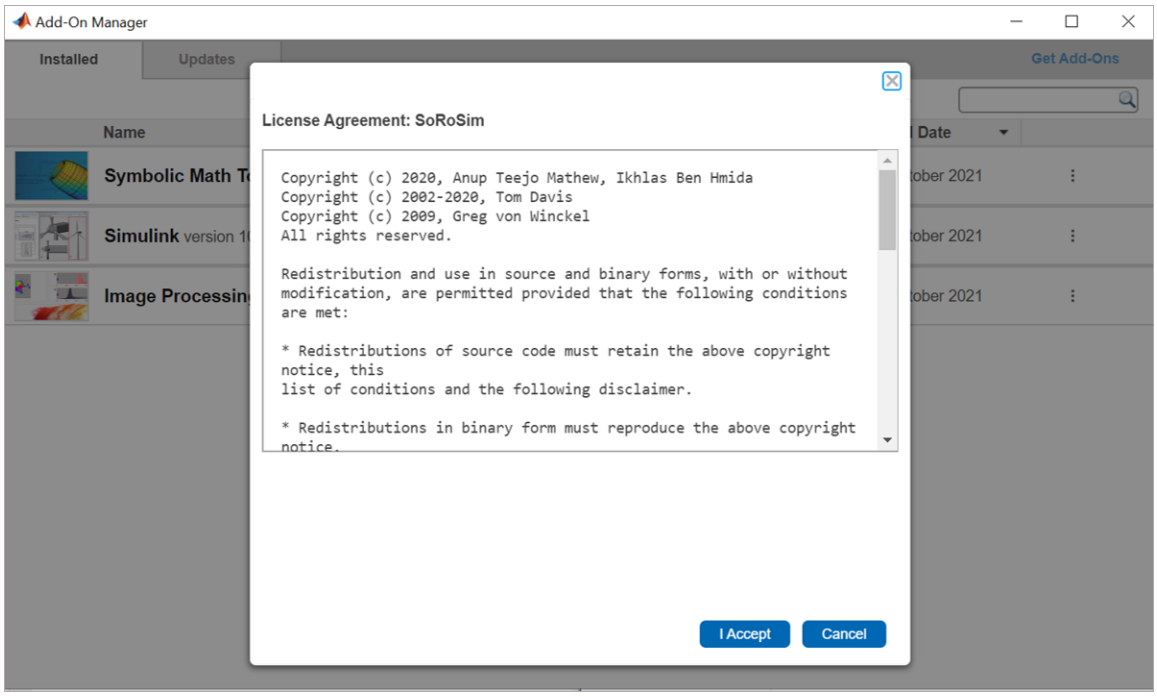


Figure 7: Toolbox license agreement

the startup.m file, that can be found among the toolbox folders, should be run to ensure the toolbox folders are added to the user's MATLAB path. This file also contains description of the classes and their methods should the user need to refer to it. Alternatively, the user could directly download the toolbox as an add-on from within MATLAB. This is done by clicking on the Add-ons icon located in the home tab at the top of the MATLAB window. The Add-On Explorer window will show a list of available add-ons. The toolbox can be found by typing "SoRoSim" in the search bar. Once the toolbox is selected click on Add to install as shown in figure 8.

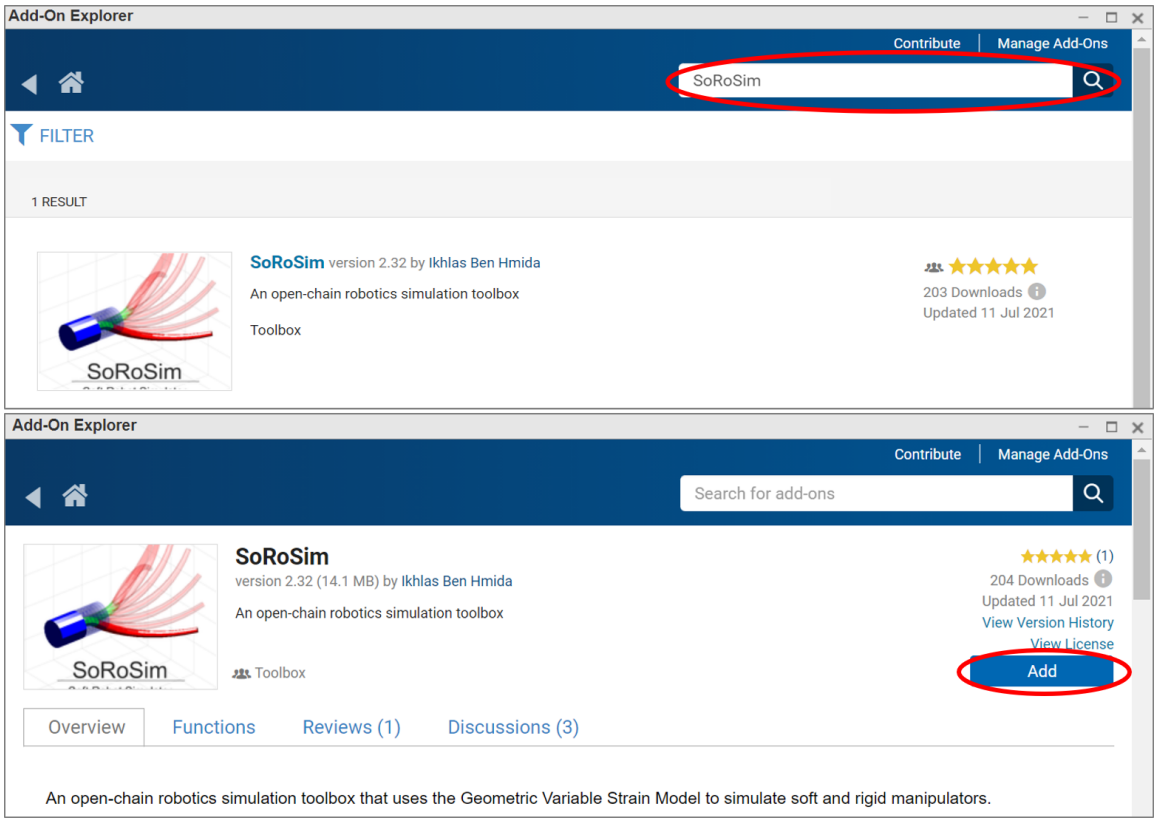


Figure 8: Toolbox installation from MATLAB's add-on explorer.

After the toolbox is installed, MATLAB automatically manages the folders path. This lets the user start using the toolbox without adjusting the desktop environment and eliminates the need to run the startup.m file. Please note that in order to use the SoRoSim toolbox, the user should have the Symbolic Math Toolbox, Optimization Toolbox, and the Image Processing Toolbox installed as well. These toolboxes can be downloaded as described above as well.

3.2 SorosimLink Creation

To create a link, the syntax is `LinkName = SorosimLink` ⏎. The user can then follow the dialog boxes that appear and answer questions to specify link properties. Default values are provided for all properties if the user does not wish to customize the link. Figure 9 shows the step by step process for creating a rigid and a soft link (`L1` and `L2`). Once a link is created, 'dot' indexing allows access and modification to its properties. For instance, to change the color of the rigid link `L1` to red, the syntax is `L1.color='r'` ⏎. Changing a property of the SorosimLink class will affect all its dependent properties. For example, a change of the radius of the second division of the soft link `L1` by `L2.r{2}=@(X1) function of X1` ⏎ will update the values of `L2.Ms{2}`, `L2.Es{2}`, and `L2.Gs{2}`.

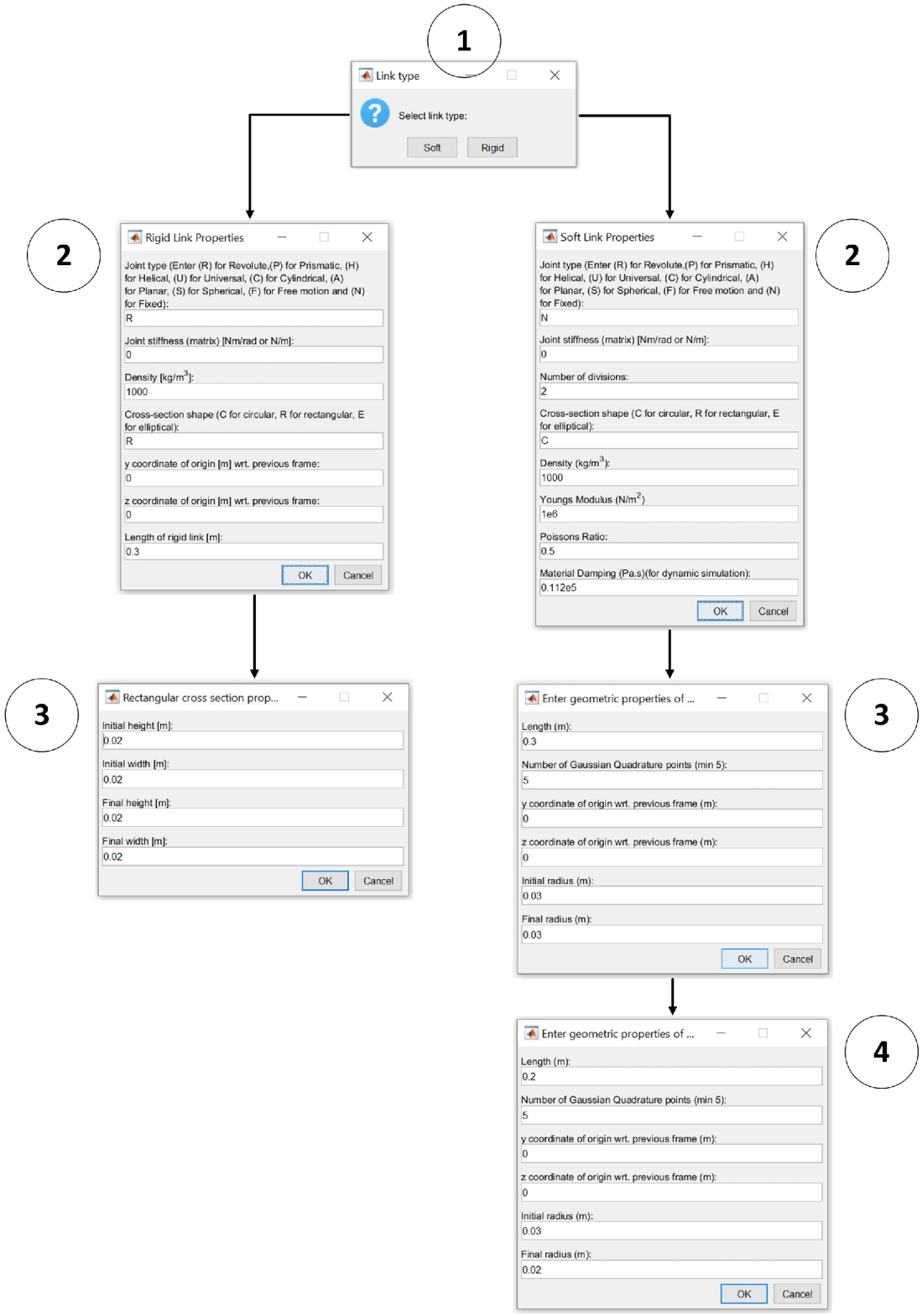


Figure 9: Steps for creating a soft link with a fixed joint, circular cross section, and 2 divisions ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4$) and a rigid link with a revolute joint and rectangular cross section ($1 \rightarrow 2 \rightarrow 3$).

3.3 SorosimLinkage Creation

The syntax for creating a Linkage is `LinkageName = SorosimLinkage(LinkName1,LinkName2,` ↵, where `LinkName1`, `LinkName2`, etc. are the names of links previously created and saved in the workspace. To create a linkage using the rigid (`L1`) and soft links (`L2`) created in the previous section, the syntax is `S1 = SorosimLinkage(L1,L2)` ↵, where `S1` is the name of the linkage. This is followed by a set of dialog boxes and graphical user interfaces (GUIs) that collect the user input needed to define the linkage structure and loads. The toolbox computes all General Properties of the linkage except `Vtwists`, `ndof`, and `q_scale` using the SorosimLink class elements (here `L1` and `L2`).

3.3.1 Twist Class Definition

The user first enters the total number of links that linkage is made up of since Links could be repeated within the linkage. After the number of links is set the user needs to define the position and orientation of each link to create open-chain, closed-chain, or branched linkages. Step 1 in figure 10 Shows the definition of the position and orientation of the first link `L1` with the help of the toolbox GUIs. A tentative plot of the link helps the user visualize the link in space. The user can test and try different values by clicking on the "Apply" button and seeing the updated plot of the current input values. We set the position of `L1` to be the origin $(0, 0, 0)$ and we keep it's orientation as the default orientation. Once the user is satisfied with their choice, pressing "Done" moves to the next step which is the twist definition.

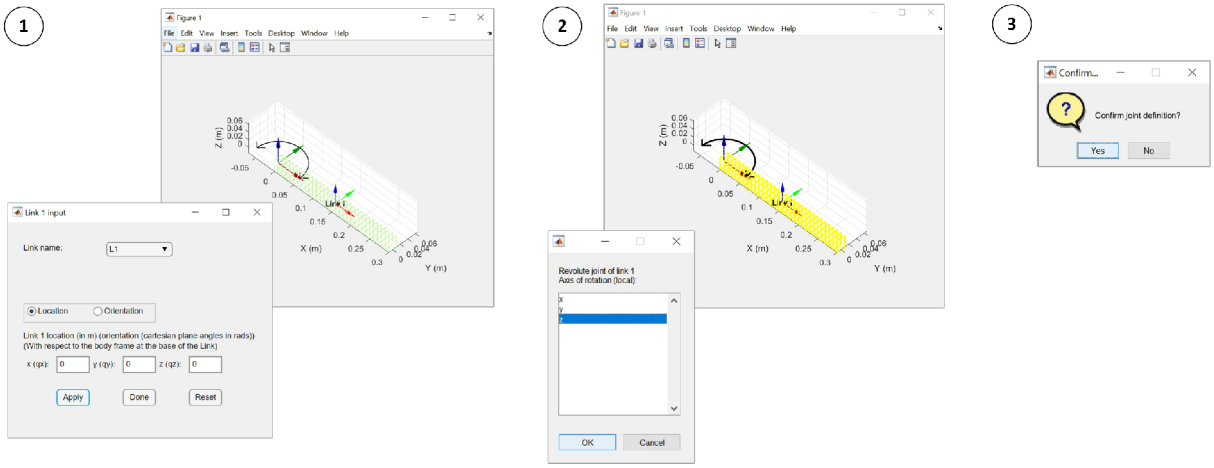


Figure 10: Step (1) shows the process of positioning and orienting the rigid link in space. Step (2) demonstrates the definition of the rotation axis for the lumped revolute joint. Step (3) Confirmation of the lumped joint definition.

The twist of rigid links is done by the Twist class and is defined by setting the axis of their active lumped joints. In this case, the rotation of the revolute joint needs to be defined. This is done by selecting the axis from a list as shown in step 2 of figure 10. The list changes appropriately based on the lumped joint connected. We choose the z-axis as the axis around which the revolute joint of `L1` rotates and select "Yes" (Step 3 in figure 10) to confirm our axis selection. The GUI will repeat Step 2 if the user chooses 'No'.

The user is then asked to do the same for the remaining links of the system. The second link we select is `L2`. As `L2` is not the first link in our linkage we have the option to select its

predecessor (`L1`, the first link, was connected to the ground by default). We can either connect it to the ground (Link 0) or Link 1, which is the the previous link we defined (`L1`). We choose Link 1 (`L1`) as its predecessor and give it a positive rotation about the y-axis as shown in step 1 of figure 11. After selecting "Done" we can now choose the active DoFs for the first division (highlighted in yellow) of the soft link.

For the first division we enable bending about the y and z-axis and set the strain order to 1. The user can define a fixed reference strain for soft link divisions as numerical values or functions of X. In our case we leave the reference strain as the default non-deformed value. The same DoFs are enabled for the second division (Step 3) but the strain order used is 0. Once the SorosimLinkage class collects all the Twist class input for each piece, we confirm our twist selection (step 4).

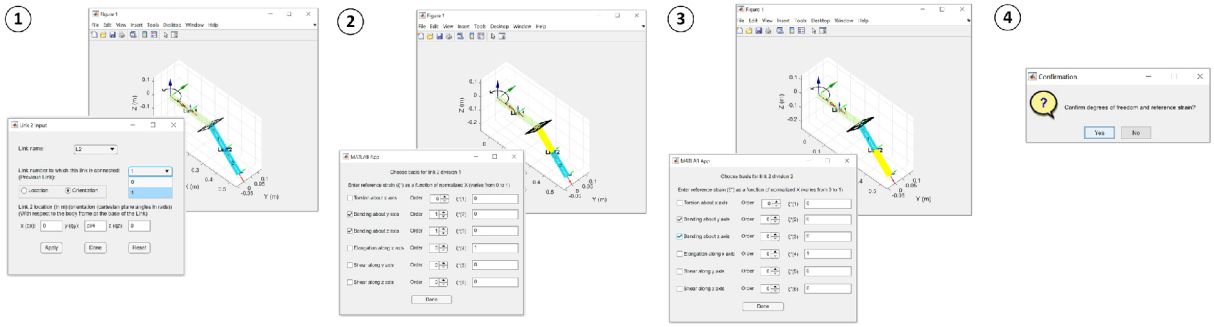


Figure 11: Step (1) setting predecessor link and orientation of `L2`. Step (2) enabling active DoFs and setting their order for the first link division. Step (3) Step (2) enabling active DoFs and setting their order for the second link division. Step (4) Confirmation of the soft link DoFs and fixed strain definition.

The same process is repeated for links 3, 4 and 5 as shown in figure 12 to form a 5-link structure.

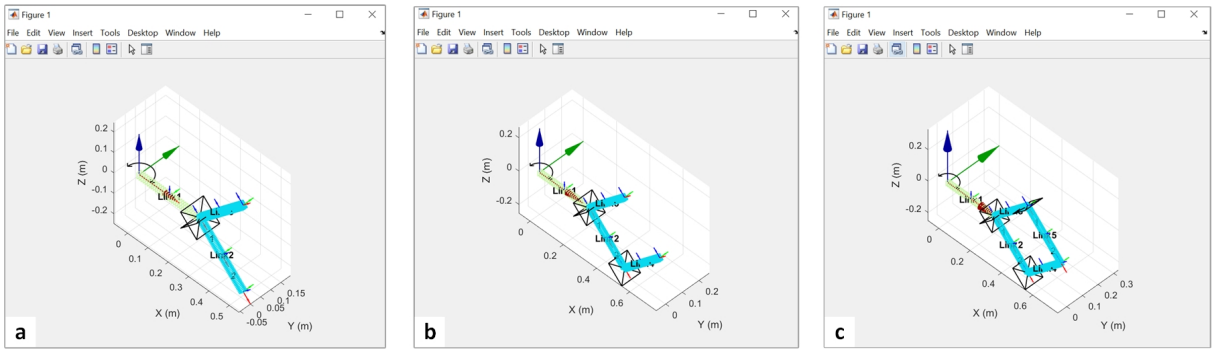


Figure 12: (a) Linkage after adding link 3. (b) Linkage after adding link 4. (c) Linkage after adding link 5.

3.3.2 Closed-Loop Joint Defenition

In order to model linkages that have closed-loop or constrained structures the toolbox uses closed-loop joints. These Closed-Loop Joints (CLJ) connect the tips of two links to constrain their motion based on user input. Figure **??** shows the steps of this process as applied to our

example. We are first asked to enter the number of CLJs present in the system. A default value of 0 is shown which can be changed appropriately. For our case the needed number of CLJs is 1. A series of GUIs then helps us to select the type of joints constraining the links. In our example we define a fixed joint that connects the tips of links 4 and 5 to form a closed loop. We first assign a letter to each of the two links to be connected, either A or B. The tip of Link A will be connected to B with respect to the body frame of Link A. After assigning the links a letter, we select the type of lumped joint connecting them from a drop-down menu as shown in step 2 of figure 13 The joint is now shown in the linkage diagram as a dashed line fixed joint at the tip of links 4 and 5.

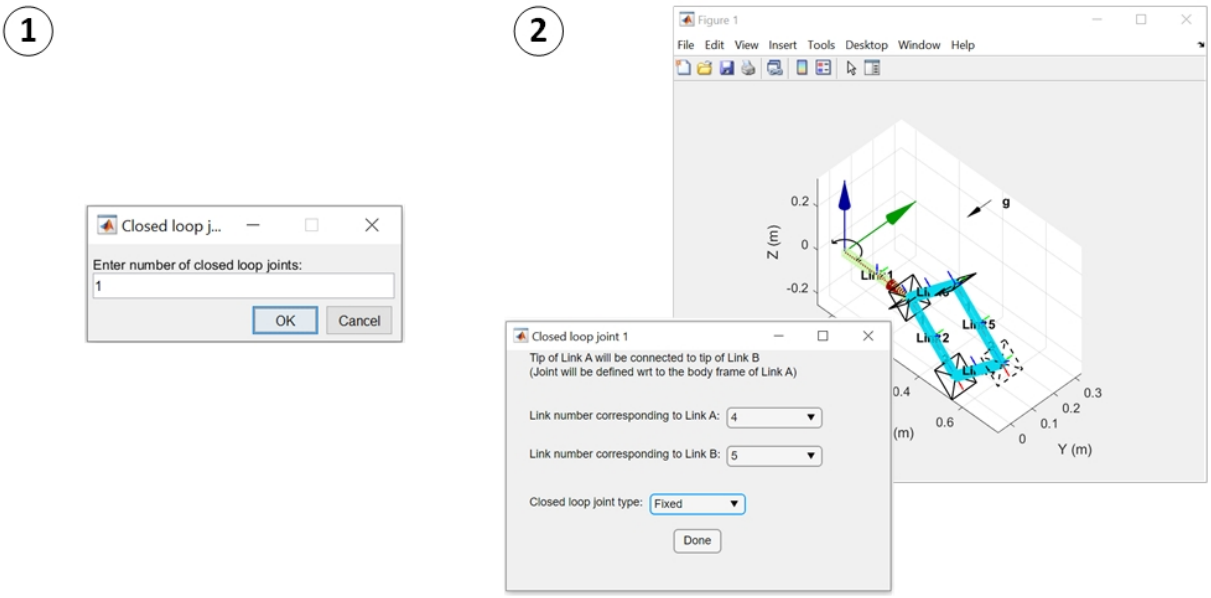


Figure 13: Step (1) Entering the number of CLJs. Step (2) Defining CLJ specifications.

3.3.3 External Force Definition

The next step in the linkage creation is the definition of external forces acting on the system. For this example, we show how to define a gravitational load as well as a follower force. Steps 1 and 2 in figure 14 show the process of defining the gravitational force. Gravity is enabled `Gravity` by answering "Yes" to the prompt in step 1 and we set the direction to be '-y' in step 2. An arrow pointing in the direction chosen will be shown on the updated linkage plot. The toolbox creates a force vector with a magnitude of the acceleration due to gravity ($9.81 m/s^2$) in the direction chosen by the user. This vector is then assigned to the property `G`.

In steps 3 to 6 we enable the `PointForce` property and choose to apply 1 point wrench (`np` =1). In step 5, we define the wrench to be a follower. This means that the force direction remains perpendicular to the surface of the body regardless of its deflection and orientation. We then apply a point force of 5 N in the direction of the local y-axis at the end of the second division of link 5 by setting the division number corresponding to the point of application of the force/moment as '2'. The SorosimLinkage class updates the value of `Fp_loc` and `Fp_vec` in this step. The user can also apply a time-varying point wrench by entering the wrench components as a function of `t` (time).

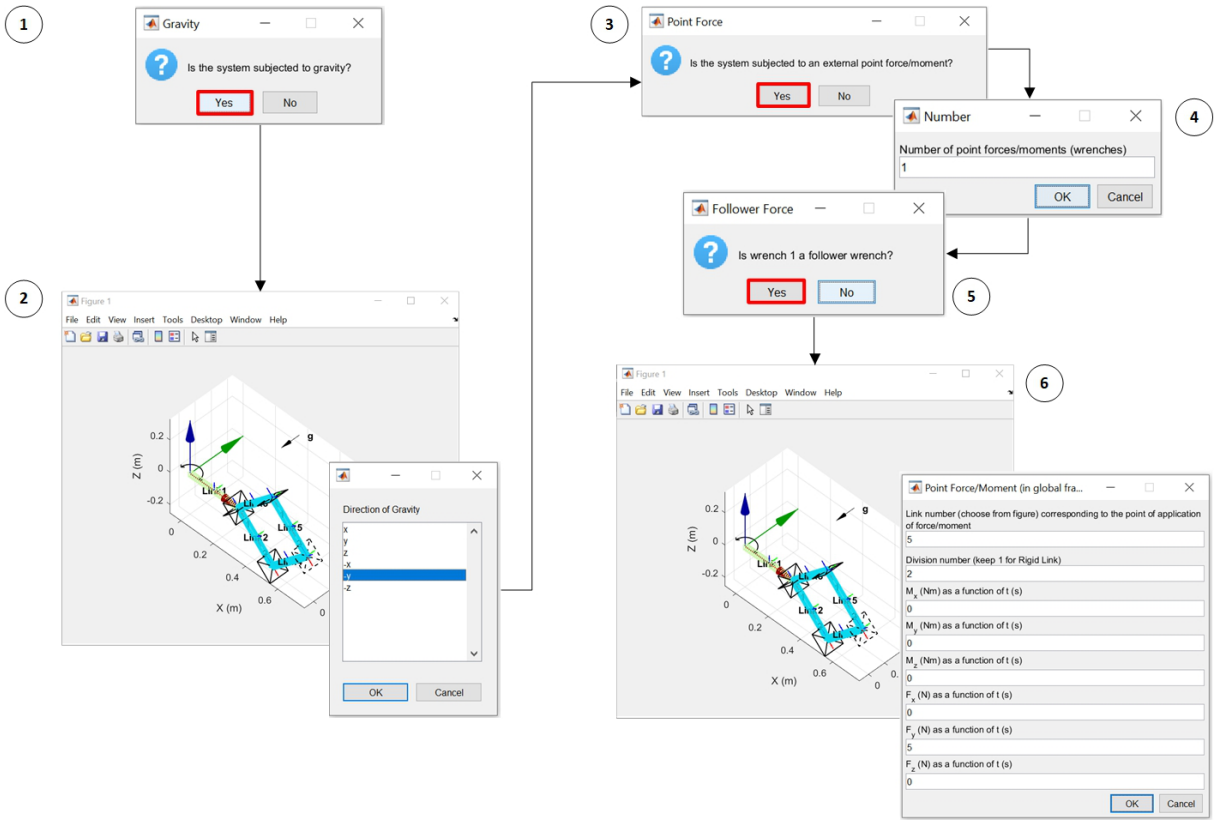


Figure 14: Steps for obtaining the External Force Properties of the SorosimLinkage class (1→ 2 → 3 → 4 → 5 → 6). Dialog boxes will appear for applying gravity, followed by GUIs forr point wrenches definition.

3.3.4 Actuation Definition

The fourth set of GUIs collects the Actuation Properties of the SorosimLinkage class (Figure. 15 and Figure .16). We enable the property `Actuated` in the first step (Figure. 15) by selecting "Yes" when asked if the system is actuated. Consequent prompts help us in setting the rigid joints in the system as active or passive and deciding on the method of their control. The GUI will highlight the links corresponding to the joint as shown in step 2. There are two ways of actuating a rigid joint, either by joint wrenches or by joint coordinates. The toolbox collects input for this in step 3. The SorosimLinkage class updates the values of `Bqj1`, `n_jact`, `n_jact`, `i_jactq`, and `WrenchControlled` in this step. For this example, we actuate the revolute joint of the rigid link by joint coordinates. The toolbox displays the image of the linkage with the letter 'W' on the joints that are wrench controlled and the letter 'Q' on the joints controlled by joint coordinates.

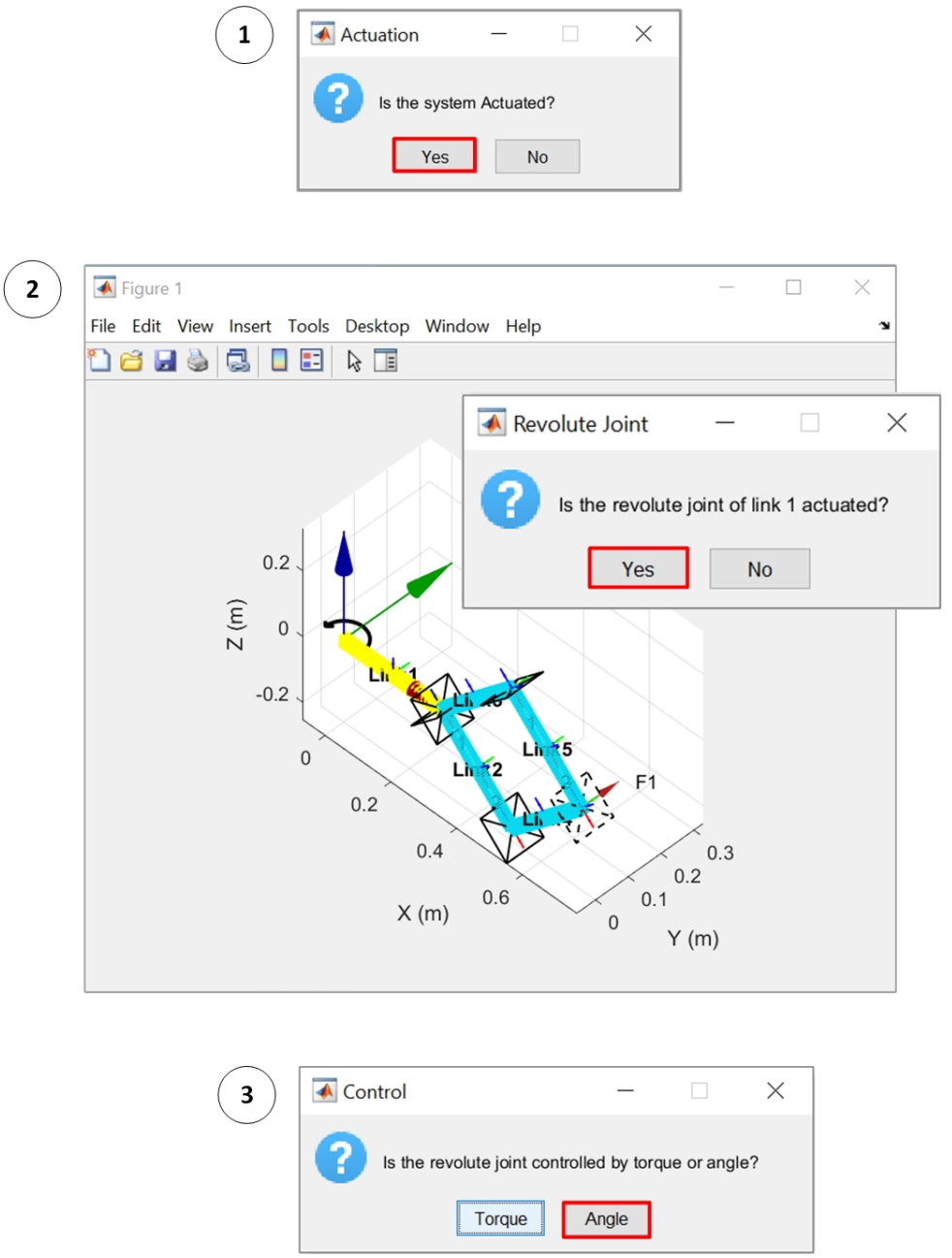


Figure 15: Steps for obtaining the Actuation Properties associated with the rigid joints of the SorosimLinkage class (1→ 2 → 3).

After every rigid joint, if there are any soft links in the linkage, the GUI asks whether the soft links are actuated (step 4 in Figure .16). After the number of soft actuators (`n_sact`) is updated in step 5 (for this case `n_sact` = 1), the GUI prompts the user to choose whether an actuator is entirely inside the linkage or not (step 6). After this, the user needs to choose the actuator path for each actuator on each link from the list: 'None', 'Constant', 'Oblique', 'Helical', and 'Custom' (step 7). The option 'None' is to disable the actuation of the link, while the option 'custom' is to enter the actuator path as a function of X. For this demonstration, we choose an oblique path and enter the parameters as shown in step 8. The actuator path is plotted based on the user input and is displayed on the linkage image (step 9). The user may change the actuator path by repeating steps 7 and 8. Steps 7, 8 and 9 are repeated for all the soft links in the system. In this example the same actuator path is created for all the soft links. Once all

the actuator paths are defined, the toolbox updates the rest of Actuation Properties.

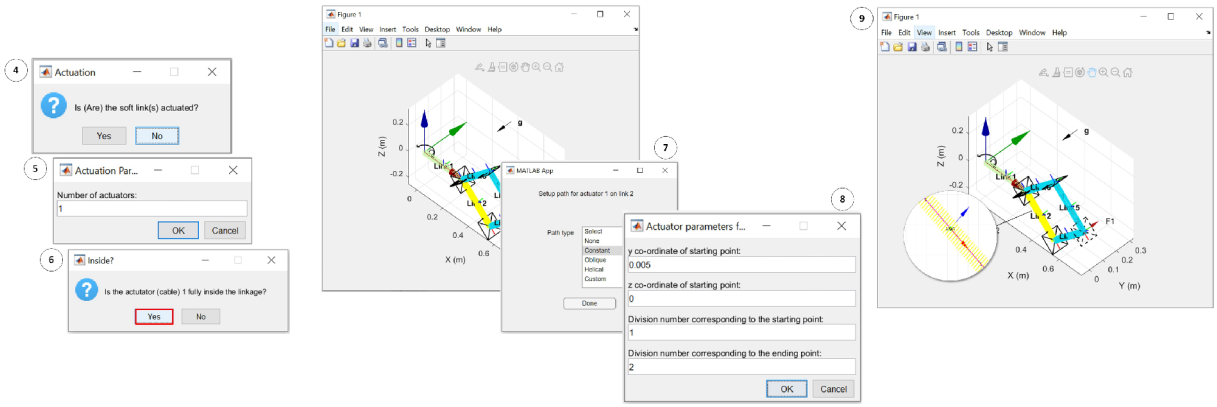


Figure 16: Steps for obtaining the Actuation Properties of soft links of the SorosimLinkage class ($4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$).

Once the SorosimLinkage creation is complete, the toolbox displays its final image showing all the forces, joint types and actuators and other annotations in the system.

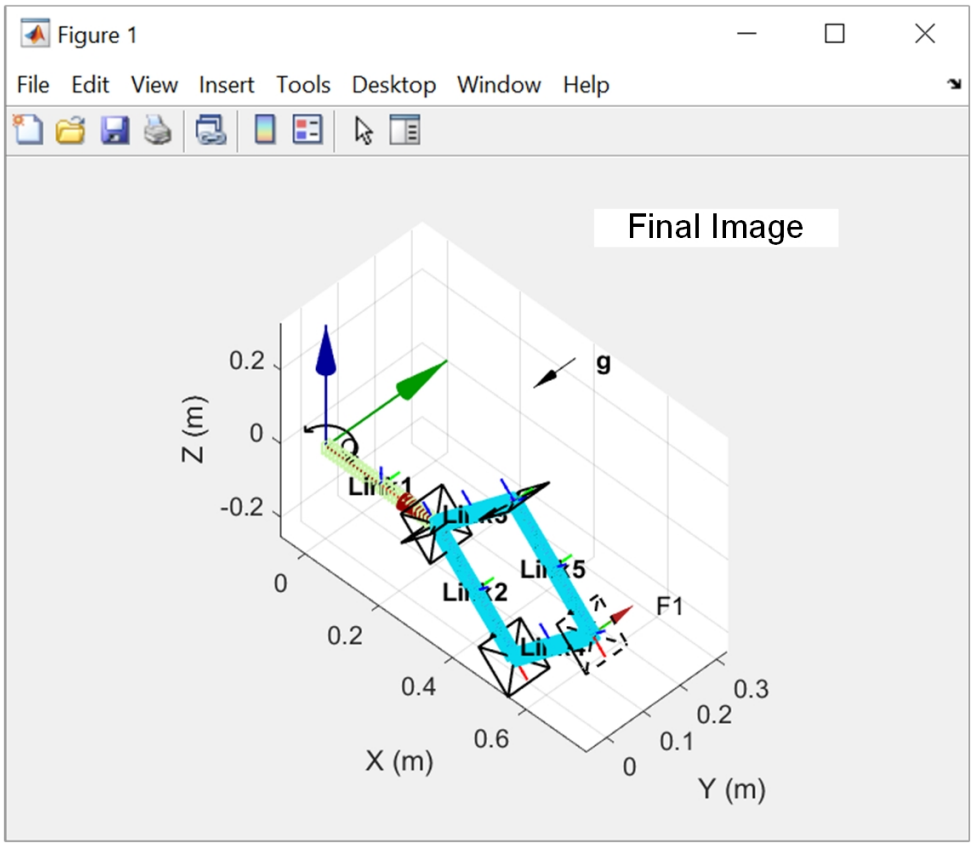


Figure 17: Step for entering the stiffness value of revolute and prismatic joints and the final image of the linkage.

We can perform static and dynamic simulations of the linkage using the SorosimLinkage class element (`S1`) we created. We demonstrate these in the next section.

3.4 Static and Dynamic Simulation

The syntax to perform a static equilibrium simulation is `S1.statics` ↵. Followed by this, the toolbox prompts the user to enter the actuation parameters. For the SorosimLinkage we created, the first prompt is to enter the value of angle of the revolute joint of link 1 in radians, and the second is to enter the strength of the soft actuator 1 in Newtons. As an example we enter '`-pi/4`' for the revolute angle and set the actuation input as '`-50`'. A negative value of actuator strength implies that the actuator is acting like a cable that applies a tension. This is followed by a dialog box asking for the initial guess of the simulation. The initial guess includes the guess value of joint coordinates and the wrenches of joints which are controlled by joint coordinates.

Once the we enter the initial guess (default values in this case), the toolbox estimates the equilibrium state of the linkage using the nonlinear solver of MATLAB called, 'fsolve' and displays the output. Figure 18b shows the final configuration as compared to the reference configuration 18a. In figure 18c we see the static solution when the revolute angle input is '`-pi/12`'. The point force is suppressed for both cases. In 18d we see the effect of the point force on the linkage. The toolbox also saves the results of the static simulation, value of joint coordinates (with a variable name 'q'), and actuator forces (with a variable name 'u'), in a MATLAB file called 'StaticsSolution.mat'.

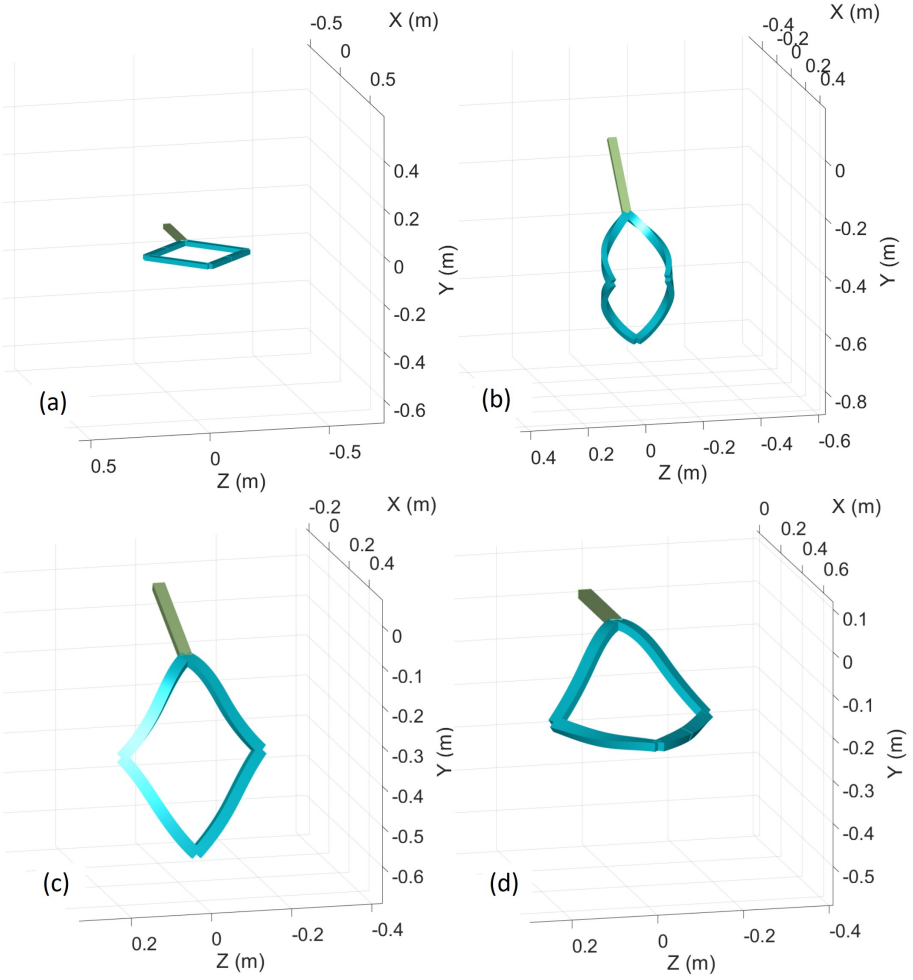


Figure 18: The static equilibrium state of the linkage at the reference state (a) and under different loading conditions(b), (c) and (d)

Similarly, to perform the dynamic simulation of the linkage, the syntax is `S1.dynamics` ↵. Subsequently, the toolbox prompts the user to enter the actuation parameters for the dynamic simulation as a function of 't' (time). As an example, we can input the revolute joint angle as a periodic function (`pi*t/2`). After this, a dialog box pops up asking for the initial condition of the simulation, which includes the initial value of joint coordinates and their time derivatives, and the simulation time. We simulate this example by keeping the default values. The user is asked of plotting while solving is required. If "Yes" is selected a plot box showing the time and the current configuration of the linkage appears while the problem is being solved.

The toolbox uses the differential solver of MATLAB called ode45 to solve dynamic problems. Once the dynamic simulation is complete, another dialog box appears and asks if the user needs the output video of the simulation. If the user chooses 'Yes', the toolbox plots the linkage states and generates an output video in '.avi' format. The toolbox saves the video as 'Dynamics.avi' on the folder from which we run the simulation. It also saves the dynamic simulation results in a MATLAB file called 'DynamicsSolution.mat'. The MATLAB file will have a $(N_{dyn} \times 1)$ array of time (with a variable name 't') and a $(N_{dyn} \times 2ndof)$ matrix of joint coordinates and their derivatives (with a variable name 'qqd', which represents q and $\dot{q}$). Here, N_{dyn} is the number of elements in 't' and $ndof$ is the degrees of freedom of the linkage. Figure. 19 shows the states of the linkage at different times of the dynamic simulation.

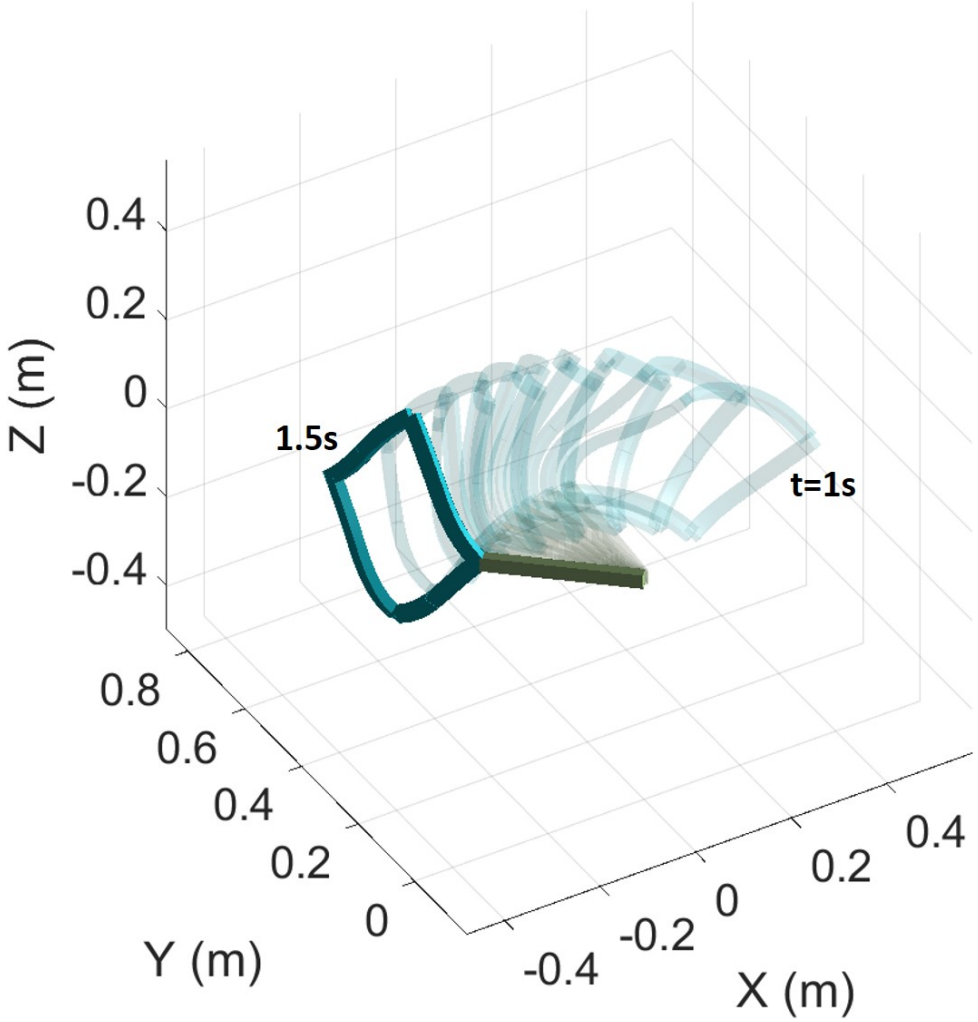


Figure 19: The states of the linkage between t=1 to 1.5s as the revolute joint rotates the system about the z axis.

References

[1] G. Blaschek, "Principles of object-oriented programming," in *Object-Oriented Programming*. Springer, 1994, pp. 9–90.

[2] F. Renda, C. Armanini, V. Lebastard, F. Candelier, and F. Boyer, "A geometric variable-strain approach for static modeling of soft manipulators with tendon and fluidic actuation," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4006–4013, 2020.

[3] F. Boyer, V. Lebastard, F. Candelier, and F. Renda, "Dynamics of continuum and soft robots: A strain parameterization based approach," *IEEE Transactions on Robotics*, pp. 1–17, 2020.

[4] R. Murray, Z. Li, and S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Taylor & Francis, Boca Raton, USA, 1994.

[5] F. Renda and L. Seneviratne, "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)*, May 2018, pp. 1567–1574.

[6] C. Armanini, I. Hussain, M. Z. Iqbal, D. Gan, D. Prattichizzo, and F. Renda, "Discrete cosserat approach for closed-chain soft robots: Application to the fin-ray finger," *IEEE Transactions on Robotics*, pp. 1–10, 2021.