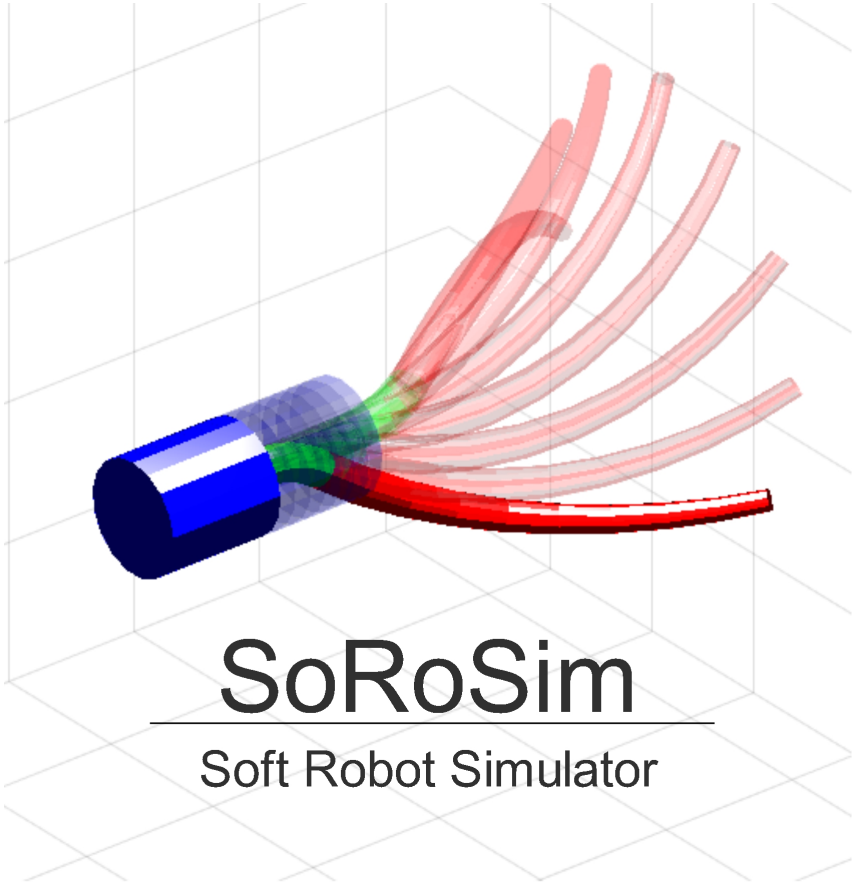
SoRoSim – a MATLAB® Toolbox for Soft and Continuum Robotics Based on the Geometric Variable-strain Approach

v. 2.32

Anup Teejo Mathew[1], Ikhlas Ben Hmida[1], *†

December 13, 2021



Introduction

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

*The manuscript is deposited on arXiv preprint server and can be accessed by this link: https://arxiv.org/abs/2107.05494

†[1] Department of Mechanical Engineering, Khalifa University of Science and Technology, Abu Dhabi, UAE.

Contents

1 Overview of the SoRoSim 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" having similar attributes or that require similar manipulations together under a class. The user can then insert data in the form of 'Properties' associated with that specific object. After an object is created, various functions or 'Methods' can be defined within the class to perform manipulations on the object to yield required results. This type of programming is well-suited for programs that are large, complex and actively updated or maintained [1].

2 Toolbox Structure

The SoRoSim toolbox consists of three classes, namely Link, Twist, and Linkage, that work together to form a powerful simulation tool that employs the geometrical variable strain model to analyze soft, rigid, and hybrid open-chain manipulators.

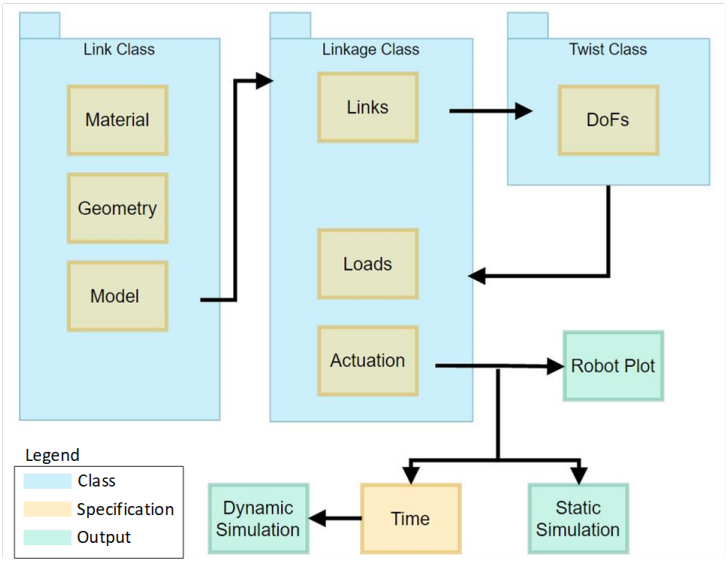


Figure 1: Main components of the toolbox .

2.1 The Link Class

A link is considered as a rigid or a soft body which is attached to a lumped joint at its starting position. The Link class allows the user to construct links that could later be combined to form manipulators. The Link 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 Link 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]
	`cx`, `cy`, `cz`	Coordinates of origin with respect to previous frame for the rigid link or cell element of the same for all divisions of the soft link [m]
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
	`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 Link 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, and coordinates of origin with respect to the previous frame (`cx`, `cy`, and `cz`) are geometric properties of the Link class. The Link 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 Link class element.

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 ($\mathbf{\Sigma}$), and damping matrices ($\mathbf{\Upsilon}$) computed at every significant point of each division. The Link class saves them as cell elements.

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)	$\begin{bmatrix}0&0&0\\0&0&0\\1&0&0\\0&1&0\\0&0&0\\0&0&0\end{bmatrix}$	${}^i\boldsymbol{S}_i^T \boldsymbol{\mathcal{F}}_{J_i}$
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	$\begin{bmatrix} 1 & X & .. & X^{N_1} & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 1 & X & .. & X^{N_2} & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 1 & X & .. & X^{N_3} \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & .. & 0 & 0 & 0 & .. & 0 \end{bmatrix}$	$N_1 + N_2 + N_3 + 3$
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 Linkage class element. The Linkage 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 Linkage 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 Linkage Class

The Linkage class allows the user to combine previously defined Link class elements into a serial open-chain manipulator (linkage). The Linkage 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 Linkage 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 Linkage class into four main categories: General Properties, External Force Properties, Actuation Properties, and Elastic Properties. (Table. 5).

2.3.1 General Properties

The General Properties of the Linkage class include numbers such as, the total number of links combined (`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

Type	Name	Description
General Properties	N	Total number of links in series
	ntot	Total number of pieces
	ndof	Total degrees of freedom of linkage
	n_sig	Total number of points at which the computation is performed (significant points)
	VLinks	Vector of Link class elements of all links from the base to the tip of the linkage
	Vtwists	Vector of Twist class elements of all pieces from the base to the tip of the linkage
	Ltot	Total length of the linkage
	g_ini	Initial transformation matrix of the linkage (default value: $I(4)$)
	g0	Cell element of the initial transformation matrix of each piece
	q_scale	$(ndof \times 1)$ array of multipliers for each joint coordinate
External Force Properties	Gravity	Value is 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
	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
	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 5: Properties of Linkage Class

(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 Linkage class arranges the Link class elements and the Twist class elements as vectors (class arrays) and saves them as General Properties, namely VLinks and Vtwists. The properties of the Link and the Twist classes can be accessed through these properties. For example, to access the reference strain vector of the second piece of the linkage, the syntax is S1.Vtwists(2).xi_starfn while to access the color of the second link of the linkage, the syntax is S1.VLinks(2).color, where S1 is the name of the Linkage class element.

The linkage class computes the total length of the linkage (Ltot) by adding the lengths of each link. It also falls under the category of General Properties. The initial transformation matrix (g_ini) specifies the location and orientation of the starting frame of the linkage relative to the global frame. Similar to g_ini for the linkage, each piece has an initial transformation matrix. The Linkage class saves these transformation matrices as cell elements, called g0. Each element of g0 (g0{piece_number}) corresponds to the initial transformation matrix of a piece relative to the previous frame. We include g_ini and g0 in the General Properties of the Linkage class. 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, is also a General Property of the Linkage class.

2.3.2 External Force Properties

We organize all properties used to apply external forces or moments on the linkage as External Force Properties. The Linkage 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 Linkage class uses a property called PointForce to enable or disable the application of point forces and moments. np, Fp_loc, and , Fp_vec are Linkage 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 Link. 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 Linkage, 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 Linkage 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.

2.3.3 Actuation Properties

We classify all properties used to define actuators on the linkage as Actuation Properties. The Linkage 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 Linkage saves the number of soft actuators using a property called `n_sact`. To compute the generalized actuation matrix for the soft piece the Linkage 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 Linkage 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 Linkage 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 Linkage 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.

2.3.4 Elastic and Other Properties

Constant parameters such as generalized stiffness matrix and generalized damping matrix are pre-computed and saved as properties of the Linkage, 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 Linkage 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.

2.3.5 Methods of Linkage Class

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

Apart from these, we pack the Linkage 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 Linkage class in Table. 6.

Name and syntax	Description
`S.findg0`	To get the initial transformation matrix of all pieces
`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.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 6: Methods of the Linkage Class. S is the name of the Linkage.

3 Using the Toolbox

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

3.1 Toolbox Installation

The toolbox is made available on more than one platforms to facilitate an easy installation process. Users can find the toolbox on GitHub as well as MATLAB Central file exchange.

The toolbox can be accessed on GitHub through the link below:
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 2 below.

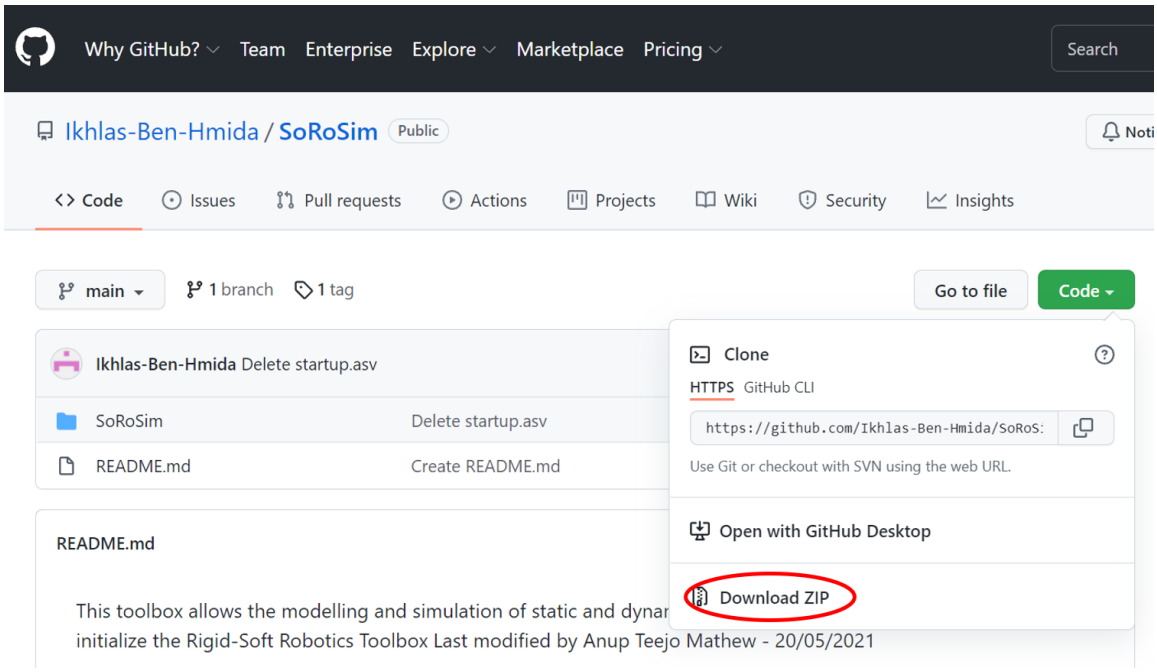


Figure 2: Download page on MATLAB central file exchange.

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 Toolbox from the drop-down menu as shown in figure 3 below.

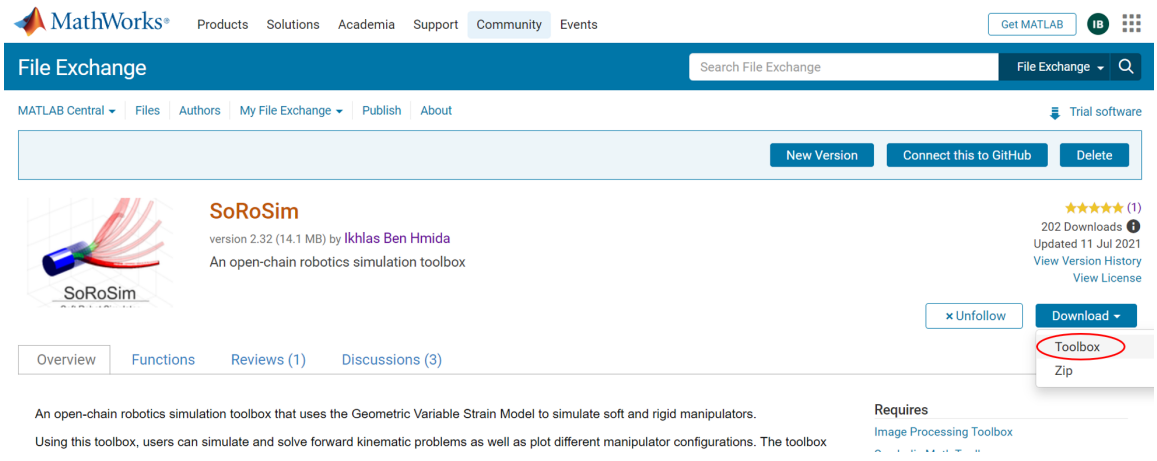


Figure 3: Download page on MATLAB central file exchange.

After the download is complete run the SoRoSim.mltbx file. This will open the Add-on manager window, shown in figure 4, after agreeing to the license agreement the toolbox will be automatically installed and ready to use.

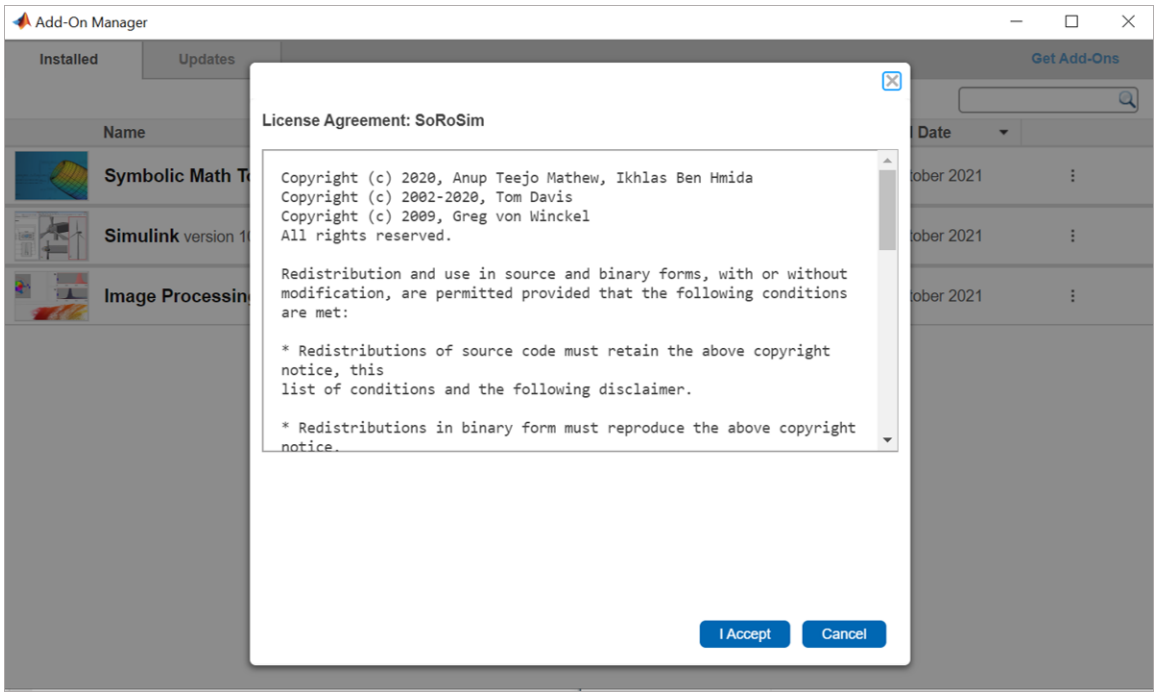


Figure 4: Toolbox license agreement

the startup.m file, that can be found among the toolbox folders in MATLAB folder directory, 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 (see figure 5).

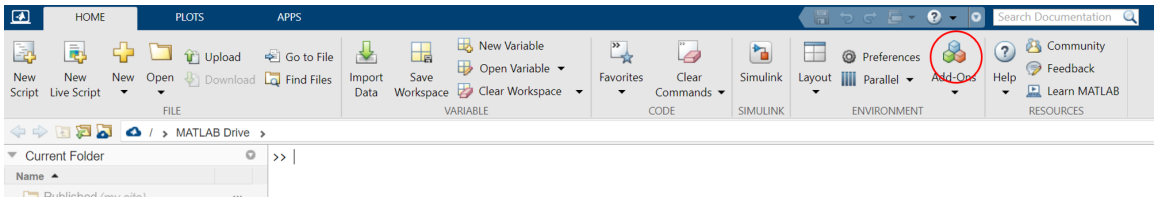


Figure 5: Add-on icon in MATLAB's home tab

The Add-On Explorer opens and displays the list of available add-ons. The toolbox can be found by typing "SoRoSim" in the search bar. figure 5 demonstrates this process. Once the toolbox is selected click on Add to install as shown in figure 6.

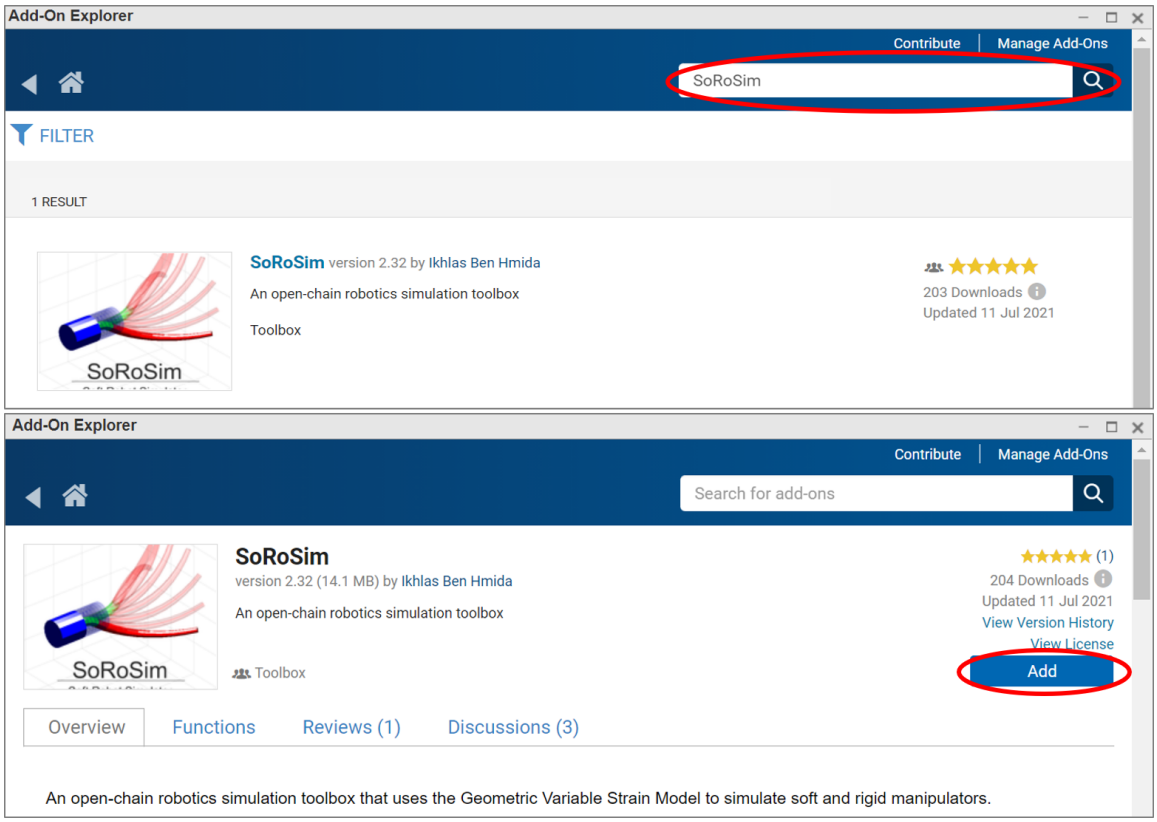


Figure 6: Toolbox installation from MATLAB's add-on explorer.

After the toolbox is installed, MATLAB automatically manages the MATLAB 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 Link Creation

To create a Link, the syntax is `LinkName = Link` ⏎. Then the user needs to follow the dialog boxes that appear. Default values are provided for all properties if the user do not wish to change. Figure 7 shows the step by step process for creating a soft and a rigid link (`L1` and `L2`). Once a Link is created, the 'dot' indexing allows to access or modify its properties. For instance, to change the color of the rigid link `L2` to red, the syntax is `L2.color='r'` ⏎. Changing a property of the Link class will affect all its dependent properties. For example, a change of the radius of the second division of a soft link `L1 by L1.r{2}=new_value` ⏎ will update the values of `L1.Ms{2}`, `L1.Es{2}`, and `L1.Gs{2}`.

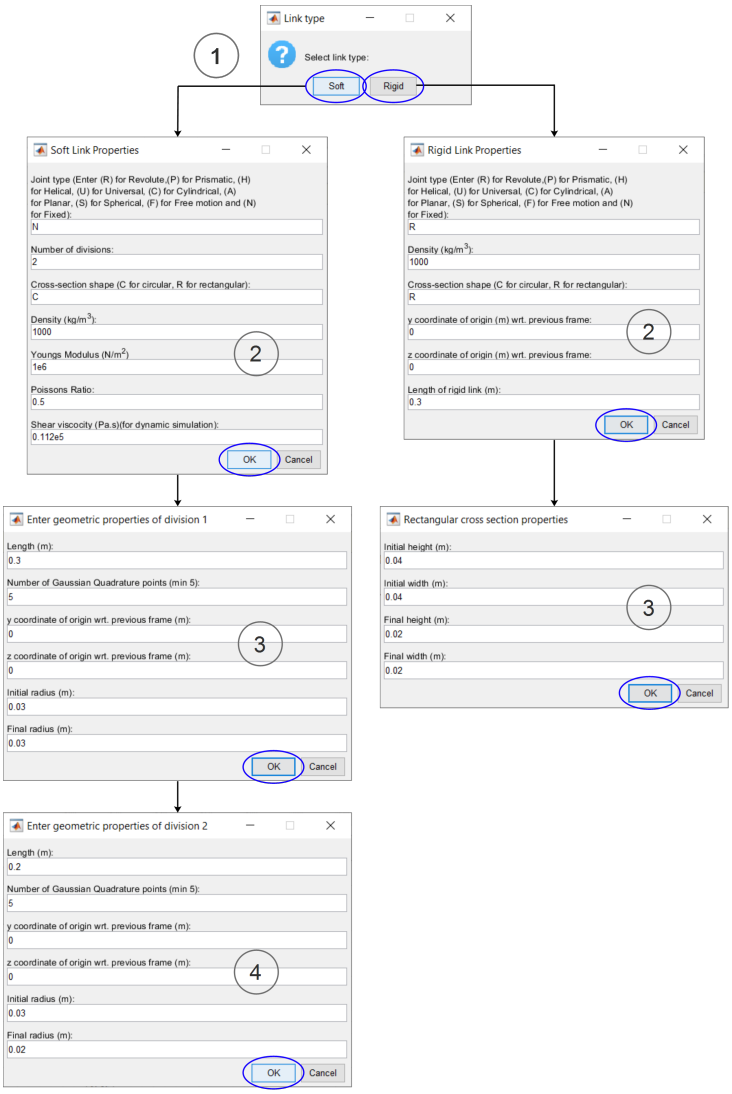


Figure 7: Steps for creating a soft link with a fixed joint, circular cross section, and 2 divisions (1→ 2 → 3 → 4) and a rigid link with a revolute joint and rectangular cross section (1 → 2 → 3).

3.3 Linkage Creation

The syntax for creating a Linkage is `LinkageName = Linkage(L1,L2,...)` ⏎, where `L1`, `L2`, etc. are the names of links that are connected in series from the base to the tip. To create a linkage using the soft (`L1`) and the rigid links (`L2`) created in the previous section, the syntax is `S1 = Linkage(L1,L2)` ⏎, where `S1` is the name of the Linkage. This is followed by a set of dialog boxes and graphical user interfaces (GUIs) to obtain the properties of the Linkage class. The toolbox computes all General Properties of the Linkage except `Vtwists`, `ndof`, and `q_scale` using the Link class elements (here `L1` and `L2`). User input is required to define these properties.

3.3.1 Twist Class Definition

The first set of GUIs obtains the Twist class element for each piece on the linkage (Figure. 8): two soft divisions of `L1` and the revolute joint of `L2` for this example. We enable all three rotational modes of deformation for the soft pieces (steps 1 and 2). For the first piece, we use a linear strain (order 1), and for the second, we keep constant strain (order 0). We also change the reference strain twist vectors for both the soft pieces, as shown in the figure (steps 1 and 2). The user may also write the reference strain vector as a function of X. For the revolute joint, we choose the y-axis as the axis of rotation (step 3).

The image of the linkage will update based on the changes made by the user. Once the Linkage class collects all the Twist class elements for each piece, an image of the final geometry of the link with a dialog box asking to confirm the geometry will appear (step 4). The GUI will repeat the previous steps if the user chooses 'No'.

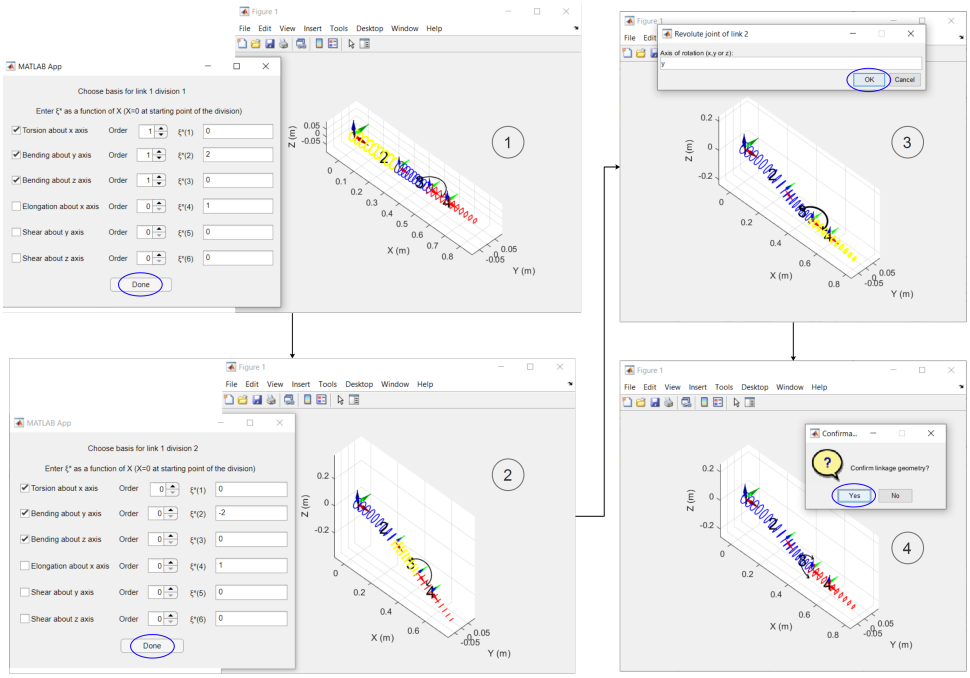


Figure 8: Steps for obtaining the Twist class elements during the Linkage class creation (1$\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 4). Dialog boxes will appear for each piece, from the start to the end of the linkage. The GUI will highlight the corresponding piece at the same time. The numbers shown on the linkage image corresponds to the piece numbers. Red, green, and blue arrows indicate x, y, and z axes, respectively.

3.3.2 External Force Definition

Once the user confirms the geometry, the second set of GUIs appears (Figure. 9). Their purpose is to obtain the external forces or moments on the linkage. For this example, we enable `Gravity` in step 1 and choose '-y' as the gravity direction in step 2. The toolbox assigns `G` with the common value of acceleration due to gravity ($9.81 m/s^2$) in the direction chosen by the user. The following images of the linkage will show the direction of gravity.

In steps 3 and 4 we enable `PointForce` and choose to apply 1 point wrench (`np` =1). In step 5, we apply a point force of 5 N in the direction of the local y-axis at $X = 0.3$ m of the first link by choosing the piece number corresponding to the point of application of force/moment as '2'. Note that we can apply the point force at X = 0.3 m as the length of the first division of

the link was 0.3 m. The Linkage 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). The following images will display the point force applied.

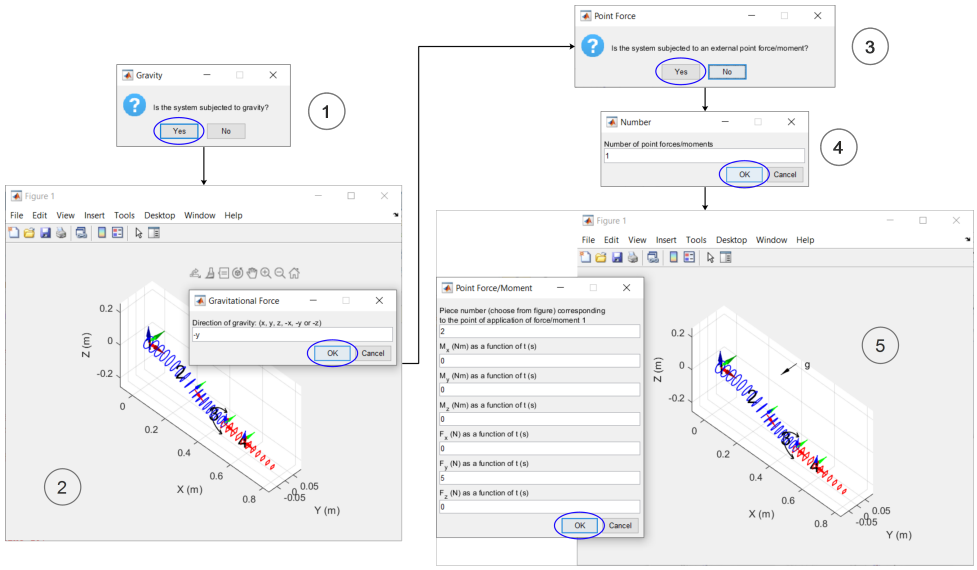


Figure 9: Steps for obtaining the External Force Properties of the Linkage class (1→ 2 → 3 → 4 → 5). Dialog boxes will appear for applying gravity, followed by point wrenches.

3.3.3 Actuation Definition

The third set of GUIs is for collects the Actuation Properties of the Linkage class (Figure. 10 and Figure .11). We enable `Actuated` in the first step (Figure. 10). Consequent prompts are for setting all the rigid joints on the linkage as active or passive. 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 prompts for this in step 3. The Linkage 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.

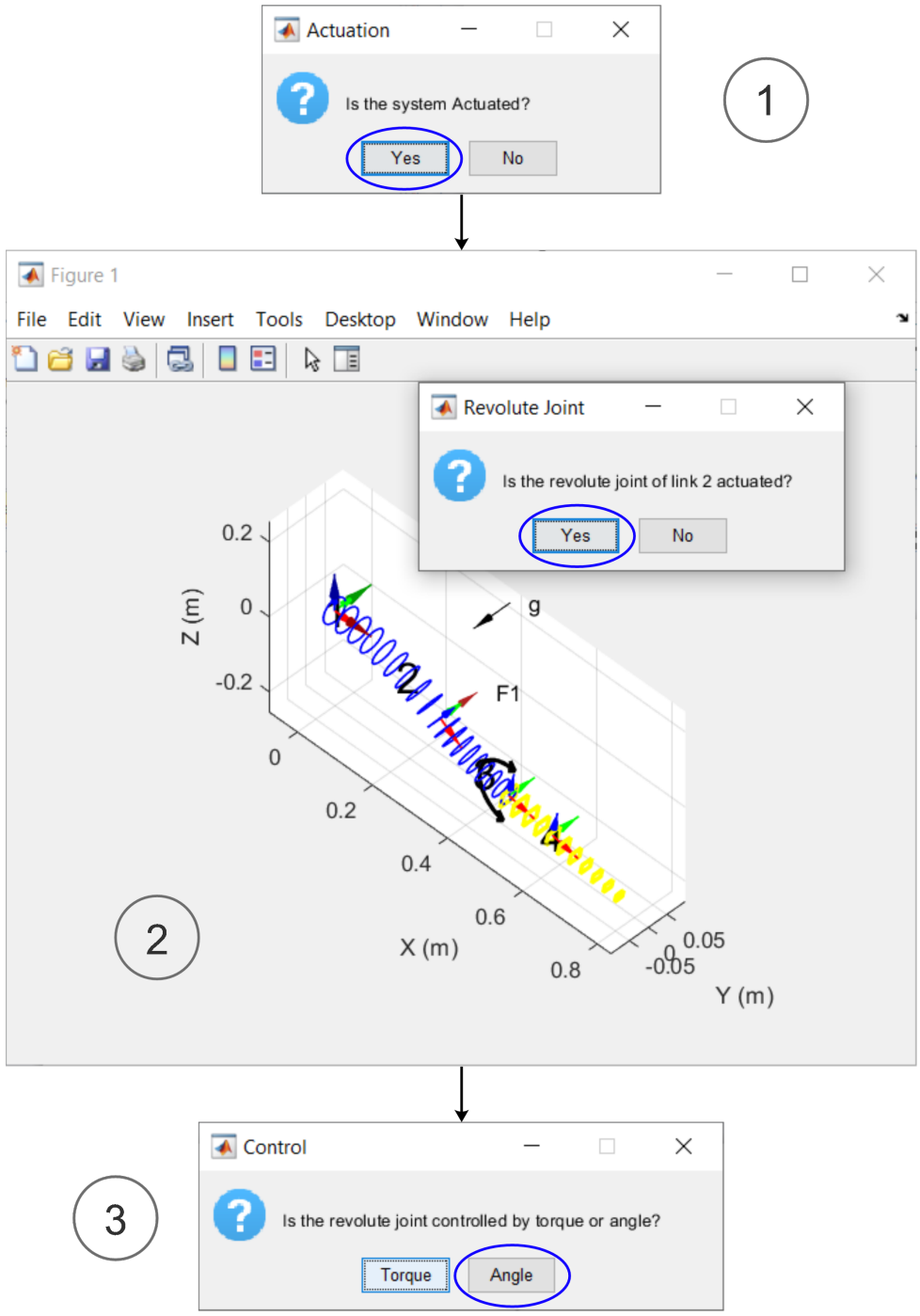


Figure 10: Steps for obtaining the Actuation Properties associated with the rigid joints of the Linkage class (1→ 2 → 3).

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 .11). 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 the helical path and enter the parameters as shown in step 8. The actuator path, according to the user

input, will be displayed on the linkage image (step 9). The user may change the actuator path by repeating steps 7 and 8. When step 9 is finished, the toolbox updates the rest of Actuation Properties.

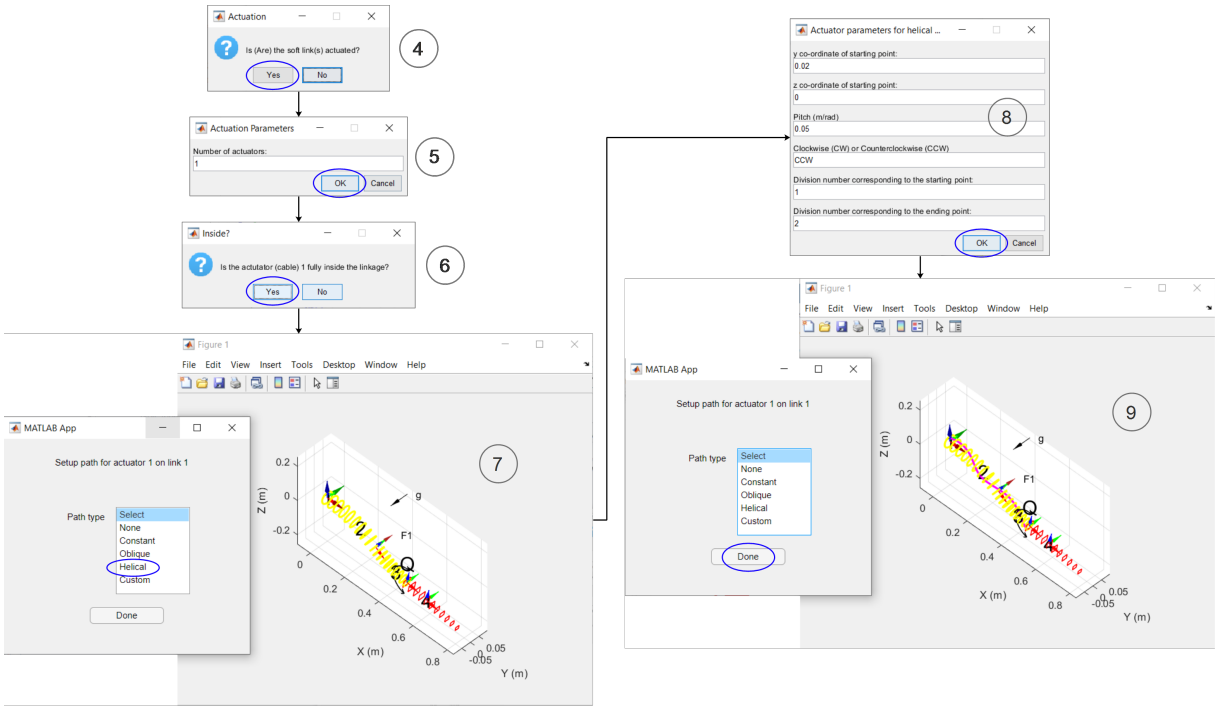


Figure 11: Steps for obtaining the Actuation Properties of soft links of the Linkage class (4 $\rightarrow$ 5 $\rightarrow$ 6 $\rightarrow$ 7 $\rightarrow$ 8 $\rightarrow$ 9).

In the final stage of the Linkage creation, if there are any revolute or prismatic joints within the linkage, dialog boxes asking the linear stiffness values of these joints will appear as shown in step 1 in Figure. 12. These values are used to update the generalized stiffness matrix `K` of the linkage. A revolute joint with a positive stiffness will act like a torsional spring, while a prismatic joint with a positive stiffness will act like a linear spring. For this example we keep the stiffness value of the revolute joint as 0 Nm/rad. Once the Linkage creation is complete, the toolbox displays its final image.

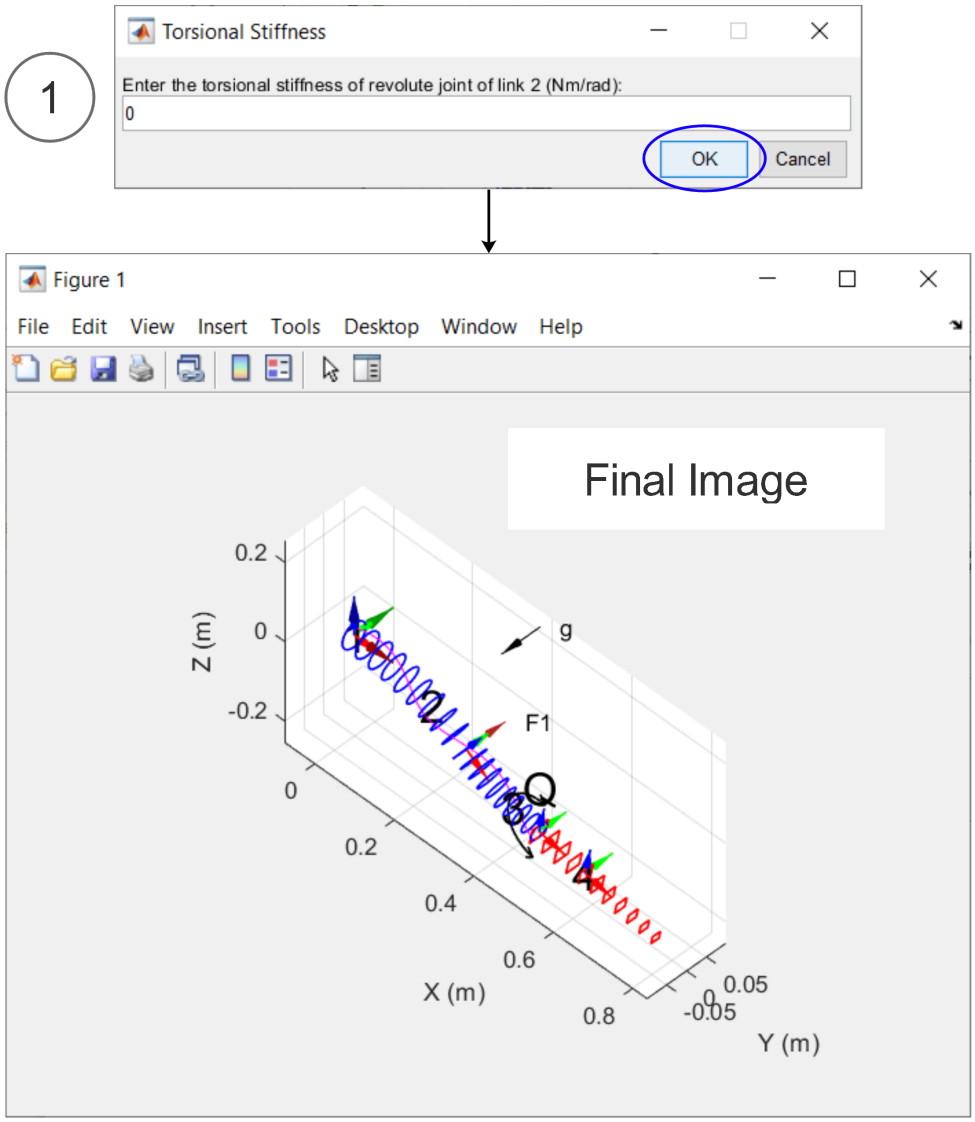


Figure 12: Step for entering the stiffness value of revolute and prismatic joints and the final image of the linkage.

We can perform static and dynamic simulations of the linkage using the Linkage 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 Linkage we created, the first prompt is to enter the value of angle of the revolute joint of link 2 in radians, and the second is to enter the strength of the soft actuator 1 in Newtons. As an example, we enter '`-pi/2`' as the revolute joint angle and '`-200`' as the strength of the actuator. 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 user enters 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. 13 compares the equilibrium state of the linkage with its reference

state. 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'.

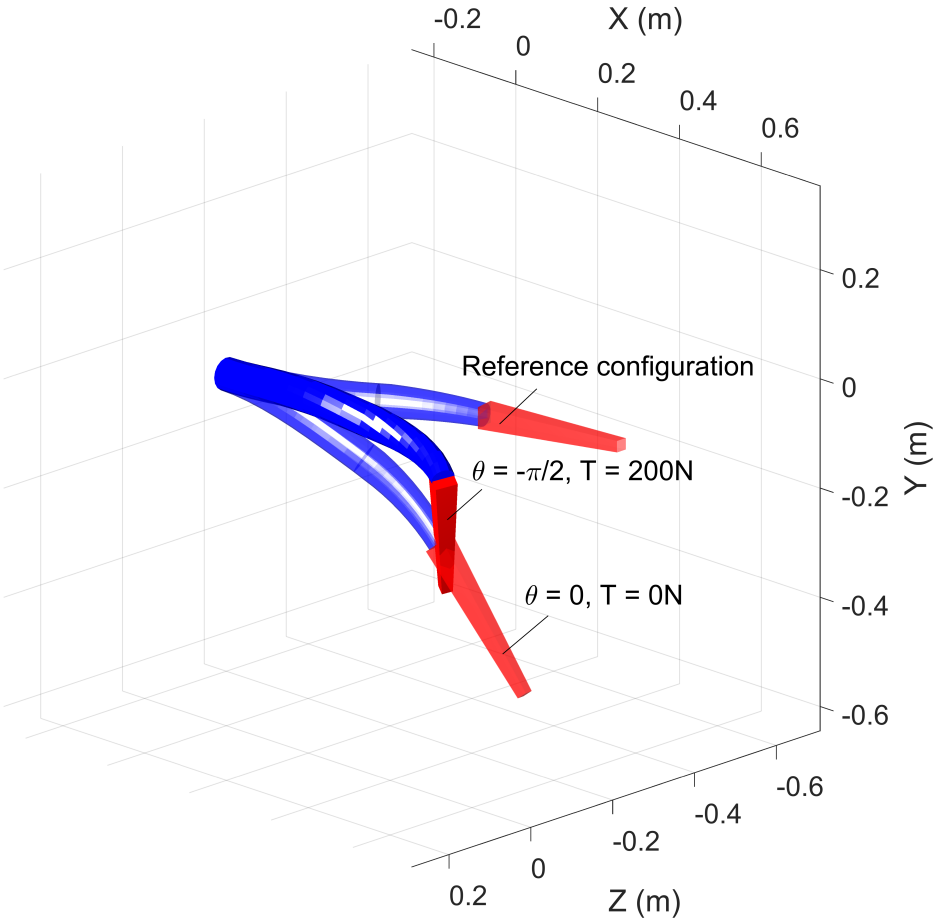


Figure 13: The static equilibrium state of the linkage when the revolute joint angle ($\boldsymbol{\theta}$) is $-\boldsymbol{\pi}/2$ and the cable tension is 200 N. The reference configuration and the equilibrium state of the Linkage when $\boldsymbol{\theta} = 0$ and $T = 0N$ are also plotted for comparison.

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/2*sin(2*pi*t)`) and the actuator strength as a ramp input that goes from 0 to -200 N in one second and stays at -200 N after that (`-200*t + (200*t-200)*heaviside(t-1)`). 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 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. 14 shows the states of the linkage at different times of the dynamic simulation.

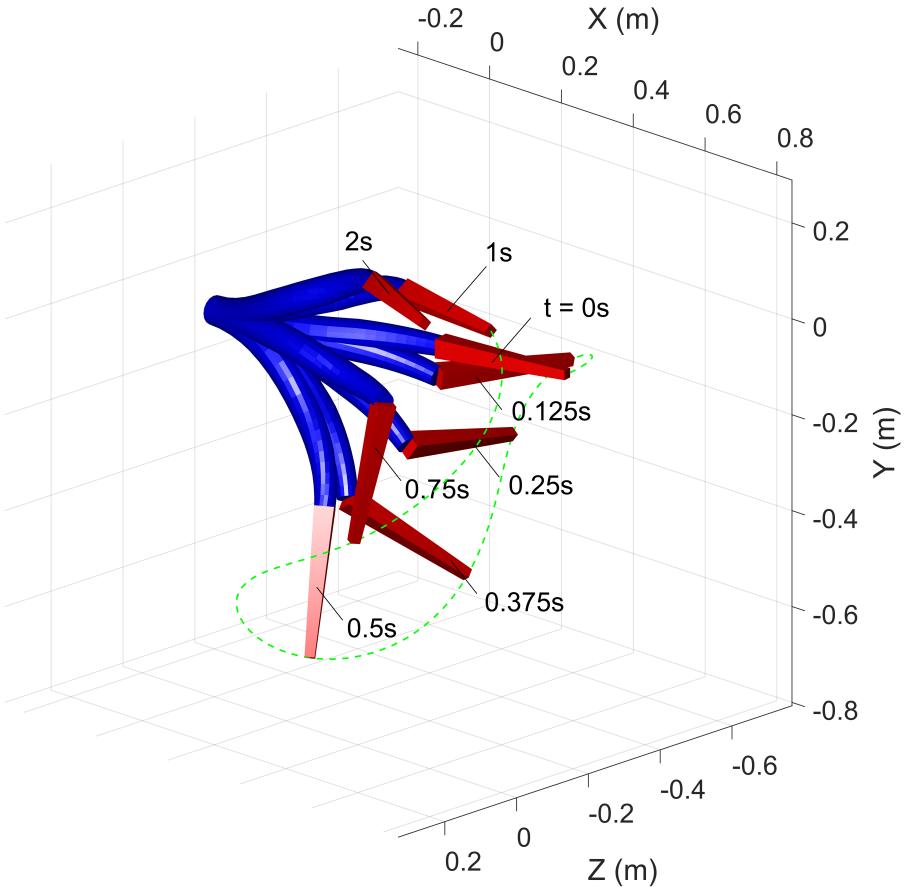


Figure 14: The states of the linkage at different times of the dynamic simulation (t=0s, 0.125s, 0.25s, 0.375s, 0.5s, 0.75s, 1s, and 2s) when the revolute joint angle, $\boldsymbol{\theta} = (\boldsymbol{\pi}/2)sin(2\boldsymbol{\pi}t)$ and the cable tension is a ramp input that reaches 200 N at 1 second and stays at 200 N after 1 second. The trajectory of the tip of the rigid link during the time, t=0 to 1s is indicated as a green dashed line.

References

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