Proceedings of the ASME 2021
International Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
IDETC-CIE2021
August 17-19, 2021, Virtual, Online

DETC2021-68042

SEQUENTIALLY WORKING ORIGAMI MULTI-PHYSICS SIMULATOR (SWOMPS): A VERSATILE IMPLEMENTATION

Yi Zhu
Department of Civil and Environmental Engineering
University of Michigan
Ann Arbor, Michigan 48109
Email: yizhucee@umich.edu

Evgueni T. Filipov*
Department of Civil and Environmental Engineering
University of Michigan
Ann Arbor, Michigan 48109
Email: filipov@umich.edu

ABSTRACT

This work presents the underlying implementation of a new origami simulator (SWOMPS) that allows for adaptability and versatility with sequential analyses and multi-physical behaviors of active origami systems. The implementation allows for easy updating of origami properties, realistic simulation with multi-physics based actuation, and versatile application of different loadings in arbitrary number and sequence. The presented simulator can capture coupling between multiple origami behaviors including electro-thermo-mechanical actuation, heat transfer, self-stress induced folding, inter panel contact, applied loading forces, and kinematic/mechanical deformations. The simulator contains five different solvers, including three for mechanical loading, one for self-folding, and one for thermal loading. The paper presents details of this code package and uses three practical examples to highlight the versatility and efficiency of the package. Because various loadings and different origami behaviors can be modeled simultaneously and/or sequentially, this simulator is well suited for capturing origami behaviors in practical real-world scenarios. Furthermore, the ability to apply an arbitrary number and sequence of loadings is useful for design, optimization, or system control studies where an unknown set of loads are needed to fold functional active origami. The coded implementation for this simulator and additional examples are made available to encourage future expansions of this work where new sequential and multi-physical behaviors in origami can be explored.

1 Introduction

Active origami systems provide novel methods to build deployable and reconfigurable engineering systems such as metamaterials [1,2], micro-robots [3,4], compact space structures [5], portable shields [6], and more. The folding process of many active origami systems are based on multi-physics actuation mechanisms [7–9] and usually involves sequential loading processes for actuation. For example, one common way to fold active origami is to use thermally or electro-thermally activated creases [4, 10, 11] because they produce high work density [12], and the folding/unfolding of the origami patterns are correlated with the heating/cooling of individual creases. When these active origami systems are actuated in practice, a sequence of mechancial loading and thermal loading will be applied onto the structure. However, current origami simulation methods are based only on the kinematics or mechanics of the system and thus, have difficulty in capturing the coupled multi-physics behaviors important for origami actuation [13–17]. Moreover, current origami simulators are implemented with a single numerical loading method, which is not suitable for simulating active origami in realistic environments where a sequence of complex loading/unloading is applied.

To overcome the above mentioned limitations, our work introduces a new *Sequentially Working Origami Multi-Physics Simulator* (SWOMPS) implementation developed using the object oriented programming (OOP) feature in MATLAB. In general, the OOP feature provides a more readable coding format and gives a more structured package architecture, which makes

*Address all correspondence to this author.

it easy to control and set up the sequential loading of origami. While the new simulator is based on bar and hinge models for origami [15–17] and borrows one nonlinear solver called the modified generalized displacement method from the MERLIN implementation [18–20] it is substantially different from these previous simulators. SWOMPS can capture different and coupled multi-physical behaviors and offers unprecedented versatility for applying an arbitrary number and sequence of loadings onto the system.

The SWOMPS formulation combines several capabilities from our previous work and from other bar and hinge origami simulators (see Fig. 1). The program includes stiffness definitions from [17, 19] and can perform a more advanced meshing of the origami pattern to include compliant creases as proposed in [21, 22]. The simulation framework incorporates one model to capture the heat transfer within origami and can simulate the thermo-mechanically coupled actuation of origami systems [23]. The program also incorporates a model to simulate inter-panel contact, which makes the program suitable for simulating advanced functional origami behavior [24].

The implementation supports five different types of *loading methods*, by which we mean numerical methods that stimulate the system behavior. These loading methods are: (1) Newton-Raphson method, (2) displacement controlled method, (3) modified generalized displacement controlled method (MGDCM) [16, 20], (4) self-folding loading, and (5) thermal loading [23]. The first three loading methods are commonly used mechanical loading methods and can solve the new equilibrium position under applied forces. The self-folding loading solves the new equilibrium location of the origami under a changing stress-free fold angle of different creases. The final, thermal loading method can solve the temperature profile of an active origami under locally applied crease heating, and then can solve for a new equilibrium configuration based on the elevated temperature of active creases. The SWOMPS implementation allows users to create highly customizable loading schemes with arbitrary number and sequence of the provided five loading methods. Furthermore, it is possible to stop a simulation at a specified step and switch between different solvers which provides additional versatility for parametric studies and optimization.

The name 'SWOMPS' is an analogy between how our program can simulate origami and how swamps behave in nature. It is possible to add and remove matter from a swamp to change its composition (properties), and sequential actions can be performed (e.g. changing the ambient temperature) that influence the coupled multi-physics of the system. The code package is made fully open access and published on GitHub at: `https://github.com/zzhuyii/OrigamiSimulator`. This version of the code can enable origami researchers to rapidly simulate active origami systems, optimize their performance, and conceive new designs. We envision that future enhancements of the code will incorporate other multi-physical and sequential behaviors important for origami design.

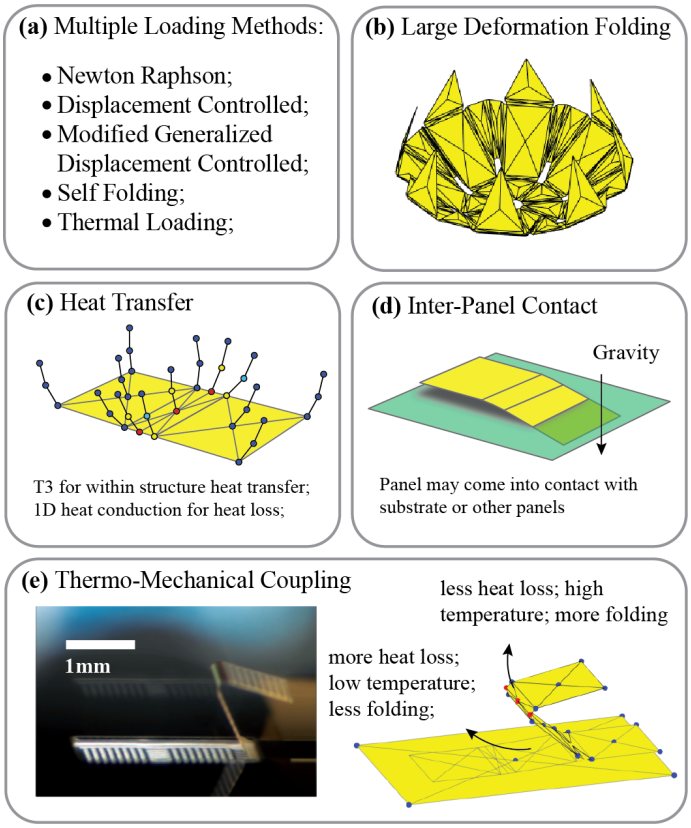

FIGURE 1. Summary of current capabilities within the Sequentially Working Origami Multi-Physics Simulator(SWOMPS) package. This SWOMPS program integrates a number of origami models including a compliant hinge bar and hinge model [21, 22], a origami inter-panel contact model [24], a bar and hinge based heat transfer model [23], as well as the standard bar and hinge model for large deformation loading of origami [16]. These models are verified using FE simulations and physical experiments and the comparisons are presented in the individual references.

The paper is organized as follows: First, we briefly introduce the mechanics and multi-physics simulation framework for electro-thermal origami systems. Next, we use one tutorial example to discuss the SWOMPS implementation, the package formulation, and the major features of the program. Finally, we demonstrate two package execution examples for practical scenarios. Additional supporting material associated with the work such as tutorial videos, tutorial documents, and additional examples can be found on our group website at: `https://drsl.engin.umich.edu/software/swomps-package/` or on GitHub: `https://github.com/zzhuyii/OrigamiSimulator`. Although the paper focuses on highlighting the thermal loading capability of the

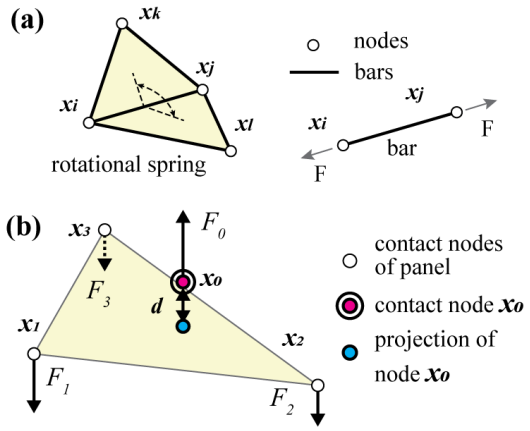

FIGURE 2. Bar and hinge model for origami structures.

SWOMPS program, the program can also be used to study active origami systems under only mechanical loading. A number of additional computation examples for mechanical loading simulations are provided on GitHub.

2 Multi-Physics Simulation for Active Origami

In this section, we introduce the multi-physics simulation framework for active origami systems. The framework is based on the bar and hinge model for origami structures [15–17], which captures the mechanical behavior of an origami using bar elements and rotational spring elements. The principle of stationary potential energy is used to find the static equilibrium position of the origami structure via:

$$\frac{\partial \Pi(x)}{\partial x} = \frac{\partial U_{bar}(x)}{\partial x} + \frac{\partial U_{spr}(x)}{\partial x} - F(x) = 0, \tag{1}$$

where x represents the nodal coordinates, and the potential energies of the bars and springs are U_{bar} and U_{spr} respectively. The term $F(x)$ is the applied loading. The detailed formulation of the two elements can be found in [16] and a detailed study of different meshing for the panels can be found in [17]. The bar and hinge model lumps the degrees of freedom in deformation into the nodes of the origami and represents the structural stiffness of origami using the bar elements and rotational spring elements that connect different nodes. The bar elements are used to capture the extensional and shearing motion of panels while the rotational spring elements are used to model the crease folding and panel bending behaviors (see Fig. 2 (a)). In general, the model is designed to rapidly simulate the global deformation of origami systems and can be orders of magnitude faster than the FE simulations. However these bar and hinge models have limited capability of capturing localized behaviors. The SWOMPS implementation can mesh an origami to have compliant creases which are important for capturing the geometry, mechanics, and actuation of many active origami [21,22]. Figure 1 (b) shows a folding simulation of an origami flower using the bar and hinge model with compliant creases. The compliant creases are developed by adding additional bar elements and rotational spring elements in the crease region that has a distributed width. The derivation of the compliant crase model, and its verification against finite element simulations and physical experiments is presented in [21,22].

The model can further consider the effects of inter-panel contact, using an additional potential term in the total potential as:

$$\frac{\partial \Pi(x)}{\partial x} = \frac{\partial U_{bar}(x)}{\partial x} + \frac{\partial U_{spr}(x)}{\partial x} + \frac{\partial U_{contact}(x)}{\partial x} - F(x) = 0, \tag{2}$$

where $U_{contact}$ is the contact potential used to capture the contact and has the form of:

$$U_{contact}(x) = \sum_{i}^{N} U_{contact,i}(d) \tag{3}$$

$$U_{contact,i}(d) = \begin{cases} k_e\{ln[sec(\frac{\pi}{2} - \frac{\pi d}{2d_0})] - \\ \frac{1}{2}(\frac{\pi}{2} - \frac{\pi d}{2d_0})^2\} & (d \le d_0) \\ 0 & (d > d_0), \end{cases} \tag{4}$$

where d_0 is the threshold at which the contact effect is initiated, and d is the distance between a contacting panel and one node from another contacting panel ((see Fig. 2 (b))). This contact potential will generate a large value if the distance d is small, and thus can prevent panels from penetrating each other. The total contact potential is the summation of all contacting pair of nodes and panels considered (represented with subscript i). The details of the contact model are discussed in our previous work [24].

More recently, we proposed a simulation method to capture the heat transfer within active origami [23]. This method captures the heat transfer within active origami using planar triangular thermal elements and simulates the heat transfer between the origami and the surrounding environment using a simplified 1-D heat transfer model. This 1-D heat transfer model is formulated as a chain of nodes and heat conduction bars (see Fig.1 (c)). The thermal conductivity of each bar is derived by preserving the total conductivity of the total substance in the surrounding environment. A detailed formulation of this model can be found in [23].

The work was calibrated using finite element models and was validated with physical micro electro-thermal origami systems fabricated using photolithography-based processes [4] (see Fig.1 (e)).

In summary, the current simulation can simultaneously capture: (1) origami with compliant creases; (2) contact between origami panels; (3) electro-thermo-mechanically coupled actuation of creases; (4) heat transfer within active origami systems; (5) global kinematics/mechanics with large deformation of origami.

3 Using SWOMPS to Simulate an Active Origami

In this section, we will introduce the formulation and various features of the implementation package with the help of a tutorial example. In the example, a single-crease origami system is actuated using electro-thermally active creases. The crease initially has internal residual stresses, the system is subjected to gravitational forces, and self-contact occurs when the crease becomes fully folded.

The origami simulator package is organized using common object oriented programming and contains one major solver class "`OrigamiSolver`" and five loading controller classes. All classes are "`handle`" type classes in Matlab. The major class "`OrigamiSolver`" stores the properties of the origami pattern and contains methods used to solve the loading behavior of the origami systems. The five loading controller classes are used to provide information related to the loading and store pertinent information on the solved behavior of the origami systems such as the displacement history.

Figure 3 summarizes the major steps used to run the package for the analysis of an active origami structure. In the following parts of this section, we will introduce the formulation of the package with the single-crease tutorial example, where we will sequentially apply 3 different loading steps onto the structure. Let's assume that we have set up an instance of the solver class as "`ori=OrigamiSolver`". Then, we can access a property of this class as "`ori.someProperty`" or run a certain method of the class as "`ori.someMethod()`".

3.1 Define geometry

The first step in setting up the simulation is to define the origami geometry. We define "`ori.node0`" to store the nodal coordinates of all nodes and "`ori.panel0`" to store the nodes of each panel. For the pattern shown in Fig. 3 (a), we input the matrix "`ori.node0=[0 0 0; 0 L 0; 0 2L 0; L 0 0; L L 0; L 2L 0]`" for x, y, and z coordinates of the six nodes, and two cell arrays for the panels "`ori.panel0{1}=[1 2 5 4]; ori.panel0{2}=[2 3 6 5];`".

After defining the base origami geometry, we run the method "`ori.Mesh_AnalyzeOriginalPattern()`" to identify all line segments of the system and to generate a corresponding numbering system for them. The line segments include all edges of the panels as shown on Fig. 3 (a), and among the total seven line segments, only line segment 3 is associated with the origami crease. Running this method solves for the number and connectivity of the lines and the solved information is stored as properties within the "`ori`" instance.

We can check the crease numbering of the system using the command "`ori.Plot_UnmeshedOrigami()`", which will display the plot shown in Fig. 3 (a). To render the proper figure for inspection, it is necessary to set up an appropriate plotting range by setting "`ori.displayRange = someValue`". Also, it is possible to generate the picture of the origami from a different direction by setting the azimuth and elevation angles as "`ori.viewAngle1 = someAngle; ori.viewAngle2 = someAngle;`".

3.2 Generate mesh for bar and hinge model

After the origami geometry is specified, the next step is to define the width of the crease and mesh the origami for simulation with the bar and hinge model. We set the width of the creases using "`ori.creaseWidthVec`", which is a vector that stores the associated crease width of each line identified in the previous step. For this example, since only line segment 3 is associated with an origami crease, we can set this vector as "`ori.creaseWidthVec=zeros(ori.oldCreaseNum, 1); ori.creaseWidthVec(3)=W;`".

The package provides two different ways to create the meshed origami structure. The default method is to create the origami meshing using compliant creases, and the second method is to represent the origami creases using a concentrated crease line (see Fig. 3 (b)). To use concentrated creases, the user can specify the command "`ori.compliantCreaseOpen=0`" which would deactivate the compliant crease meshing.

After setting up the width of the crease and determining the proper meshing method, the meshing of the bar and hinge model is generated using "`ori.Mesh_Mesh()`". This method will create the meshed origami system as shown in Fig. 3 (b). Users can use the command "`ori.Plot_MeshedOrigami()`" to check the newly generated bar and hinge model meshing for the origami.

3.3 Define properties

The next step for setting up the simulation is to define the properties of the origami. There are three major groups of properties: mechanical properties, panel contact properties, and thermal properties. Here, we give a summary of how these properties are defined.

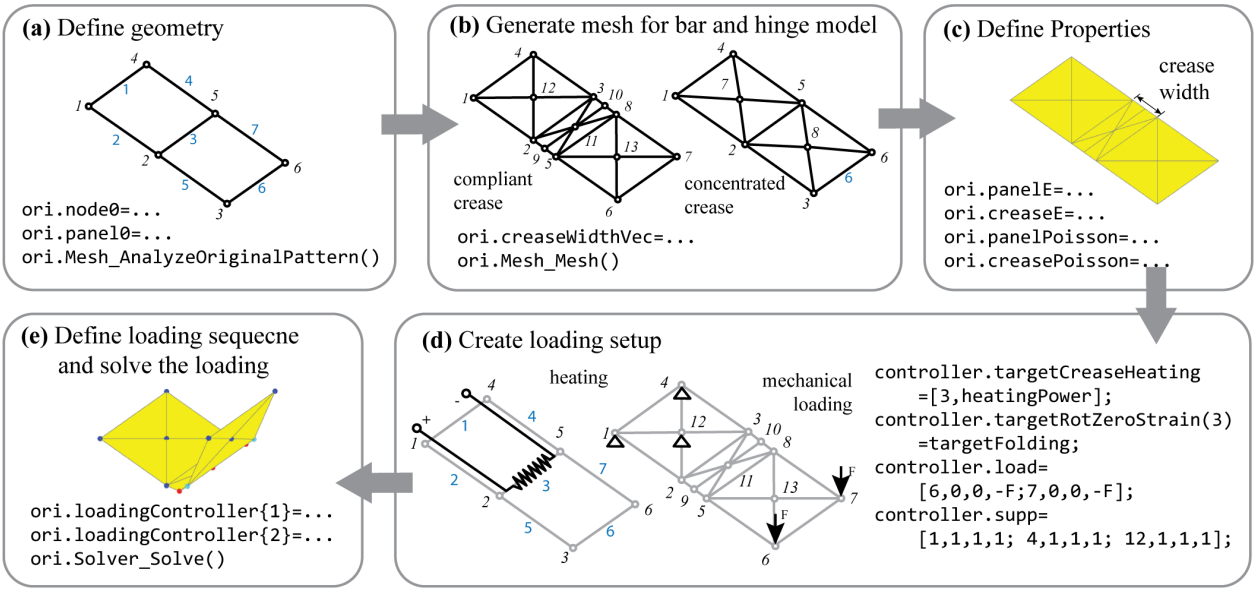

FIGURE 3. A general flowchart for using the SWOMPS package for multi-physics simulation of origami.

Mechanical Properties:

`panelE & creaseE:` These are double precision numbers that store the Young's modulus of the panels and the creases respectively. Similar to other bar and hinge models, SWOMPS assumes the panels to be made with isotropic materials.

`panelPoisson & creasePoisson:` These are double precision numbers that store the Poisson's ratio of the panels and the creases respectively. (Assuming isotropic material)

`panelThickVec:` This is a vector that stores the thickness of the panels. For the single crease example we have "`ori.panelThickVec = [t_panel1; t_panel2]`".

`creaseThickVec:` This is a vector that stores the thickness of creases; we use the numbering system generated in Fig. 3 (a) to refer to the creases. For the single crease example we have "`ori.creaseThickVec(3) = t_crease`".

Panel Contact Properties:

`contactOpen:` This binary variable determines if panel contact is considered or not. Defining a value of 1 means the simulator will consider contact while a value of 0 means the simulator will ignore panel contact.

`ke:` This is a double variable that stores the contact potential scaling factor. A larger value will give higher contacting forces and a more abrupt transition after the panel contact is initiated. The simulator is not sensitive to the selection of this value when considering convergence.

`d0edge & d0center:` These are double precision numbers that store the contact initiation thresholds for the edges and center of the panel respectively. Usually, it is best to use values that are close to the real thickness of the panel.

A detailed study of the contact related parameters are presented in [24].

Thermal Properties:

`panelThermalConductVec:` This variable stores the thermal conductivity of the panels. For the single crease example we define "`ori.panelThermalConductVec = [K_panel1; K_panel2]`".

`creaseThermalConduct & envThermalConduct:` These are two double precision numbers that store the thermal conductivity of the creases and the surrounding environment respectivley.

`t2RT:` This is a double precision number that represents the distance (or environmental matter thickness) at which we reach the room temperature as measured from the surface of the origami structure.

`envLayer:` This integer variable stores the total number of discretization used to formulate the bar and hinge heat transfer model for capturing the heat loss between the structure and the surrounding environment. A value of 10 is recommended.

`thermalDissipation:` This is an double precision number that stores the heat dissipating angle for capturing the heat loss between the structure and the surrounding environment. A value of $20\pi/180$ is recommended for generic situations. A larger value will generate a higher rate of heat loss to the surrounding environment.

`RT:` This is a double precision number defining the room temperature of the surrounding environment.

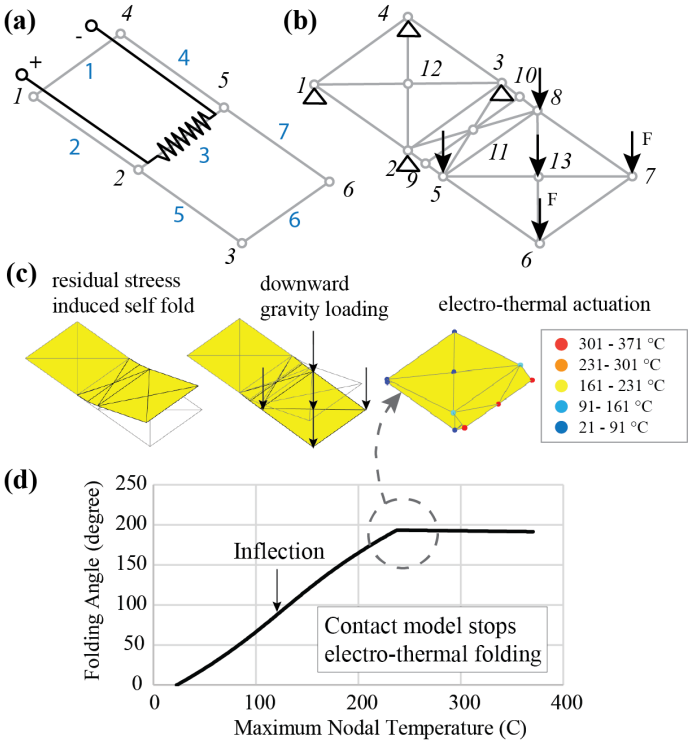

FIGURE 4. Simulation of a single crease folding under an applied electro-thermal actuation. (a) Geometry and node numbering of the input unmeshed origami; (b) Geometry of the meshed origami with support and gravity loading (The gravity nodes are only applied to the suspended panels); (c) This example considers three loading effects including: self-folding induced by a residual stress, gravity loading of one panel, and electro-thermal actuation. The three loading steps are applied sequentially by stacking different loading controllers; (d) The folding motion stops when inter-panel contact occurs, even though we keep increasing the heating power applied to the crease.

A detailed study on the parameters related to the thermal simulation are presented in [23].

3.4 Create loading setup

After defining the properties of the origami system, we can prescribe loading steps for the analysis. The solver allows users to create a loading scheme with an arbitrary number and sequence of loading methods chosen from the five provided here (additional loading methods can be introduced for SWOMPS in the future). For this tutorial example, we demonstrate three of the loading methods. Consider the structure will first experience self-folding induced by the residual stresses, then gravitational forces are applied onto the same structure, and finally a thermal loading is added for actuation. To prepare the simulation, we first need to define the three loading steps individually and then group them to create a loading sequence.

To apply the residual stress which induces self-folding, we first create a self-folding controller instance as: "`selfFold=ControllerSelfFolding`". Then, we define the target stress-free folding angle of crease 3 using:

```
"selfFold.targetRotZeroStrain=
pi*ones(ori.oldCreaseNum,1);
selfFold.targetRotZeroStrain(3)=
pi+20/180*pi;",
```

assuming that crease 3 will develop 20 degrees of folding generated by residual stresses. After defining the target self-folding, we define the support information as: "`selfFold.supp=[1 1 1 1; 2 1 1 1; 3 1 1 1; 4 1 1 1];`" where we are restricting the x, y, and z directions of node 1 to 4 as shown in Fig. 4 (b). When using the package for other systems, it is possible to release support in the designated directions by changing the 1 to a 0. The package also supports the use of non-rigid supports, where a linear spring can be inserted at the support point. This non-rigid support can be turned on by setting "`selfFold.nonRigidSupport=1`".

When setting up the analysis, it is possible to change other loading solver parameters such as the total number of incremental steps `increStep`, the tolerance threshold for solving equilibrium `tol`, and the maximum iterations per increment `iterMax`. If these parameters are not specified, default values of `increStep=50`, `tol=1E-5`, and `iterMax=30` are used.

To apply the gravitational loading we initiate an instance of the Newton-Raphson solver as: "`nr=ControllerNRLoading;`". To load the structure, we define the loads as:

```
"nr.load=[5,0,0,-loadMag;
6,0,0,-loadMag; 7,0,0,-loadMag;
8,0,0,-loadMag; 13,0,0,-2*loadMag;];".
```

This matrix will create a downward (-z direction) loading with a total magnitude of `6*loadMag` acting on the panel. The load is distributed across the panel evenly, such that the center node will have double the load value of the edge nodes. In this example we do not add gravity on the crease or the restrained panel, though those gravity loads can be applied in a similar fashion. The support conditions are specified in an identical way as with the self-folding loading. Similarly, we can change other loading information (e.g. `increStep`, `tol`, `iterMax`, etc.) based on our needs.

We can finally create an instance of the electro-thermal loading controller as: "`thermal=ControllerThermalLoading;`". The SWOMPS simulator assumes that the electro-thermal creases are made from bimaterial morphs which are common in many

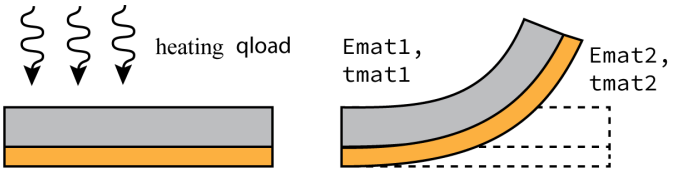

FIGURE 5. Properties used to capture the thermally induced folding of bimaterial morph actuators for active origami.

active origami [4, 7, 10]. These creases fold because of different thermal expansion in the two layers of materials (see Fig. 5). Other thermally active creases could be embedded in SWOMPS by modifying the embedded thermo-mechanical model. The first step in specifying electo-thermal loading of the bimorph creases is to define the target heating power input as:

```
thermal.targetCreaseHeating=[3,qload];,
```

where we apply heating power with a magnitude of `qload` to crease 3 of the origami. We define the actuator properties within the thermal loading controller based on Fig. 5 as:

```
“thermal.deltaAlpha
=zeros(ori.oldCreaseNum,1);
thermal.deltaAlpha(3)=someValue;
thermal.Emat1=someValue;
thermal.Emat2=someValue;
thermal.tmat1=someValue;
thermal.tmat2=someValue;];”.
```

where “`thermal.deltaAlpha`” stores the difference between the thermal expansion coefficients (α) of the two materials, “`thermal.Emat1`” and “`thermal.Emat2`” store the Young’s moduli of the two layers, and “`thermal.tmat1`” and “`thermal.tmat2`” store the thickness of the two layers. After setting up the loading, we define the supports and change the loading parameters based on our needs.

After defining the three loading controllers (i.e. separate instances), we can apply the loading controllers in series to our system “`ori`” as:

```
ori.loadingController{1}=
{"SelfFold",selfFold};
ori.loadingController{2}=
{"NR",nr};
ori.loadingController{3}=
{"ThermalLoading",thermal};
ori.Solver_Solve()
```

The first three commands will create a 3-sequence loading process where the residual-stress (and resulting self-folding) is applied first, the gravity second, and electro-thermal actuation last. The last line of of code will run the solver. Figure 4 (c) shows the results of the loading scheme. The origami first folds up due to the residual stress, and then folds back downward due to the gravity loading. Finally, the origami is heated, causing the origami panel to fold just past 180 degrees (possible with a compliant crease) when the crease temperature increases to 240°C. The solver can capture the inter-panel contact, so while the crease temperature increases, no further folding is obtained (see Fig. 4 (d)). When simulating the electro-thermal actuation, the solver maintains and integrates all previously applied loading as needed (i.e. self-stress and gravity). For example, for fold angles below 90° gravity pushes the system towards the initial flat state, while after 90° the applied gravity assists in the folding. Due to this gravity effect, we see the slightly different curvatures and an inflection point in the fold angle vs temperature curve (around 90° in Fig. 4 (d)).

The capability of stacking different loading controllers to solve the origami system sequentially is useful for enabling versatile and/or multi-physical loading. For example, with the stacking capability, users can switch between different mechanical loading solvers at a particular loading step so they can explore the multi-stable or bifurcation behavior in a nonlinear structure. Additionally, one can study the behavior of an origami under a combination of different loading effects, which is demonstrated by the situation in this particular example. This 3-step loading process resembles a realistic application scenario where the origami device experience a variety of physical effects observed in practical systems [4].

3.5 Continuing loading

Another handy functionality of the solver is that it allows users to continue the loading process based on the existing configuration of the origami. This capability can be activated by setting “`ori.continuingLoading=1`”. When continuing loading is activated, the solver will skip the initializing steps and resume the analysis based on the current origami configuration. Suppose for example that we want to double the gravitational load after running all of the above simulations; then we would simply run the following command:

```
ori.continuingLoading=1;
ori.loadingController={};
ori.loadingController{1}="NR",nr;
ori.Solver_Solve;
```

The above code will place another gravitational loading process on to the system and will find the new equilibrium position with the double gravitational loading. This functionality can be useful for creating external incremental loops as will be demonstrated in the example section. The sample code for this tutorial simulation can be found in “`Example01_SingleCrease.m`” in the code package on GitHub. A tutorial document associated with this example can be found on our group website at `https://drsl.`

engin.umich.edu/software/swomps-package/.

3.6 Retrieve Structure Response

After running the entire analysis, the SWOMPS program also allows users to retrieve the structures responses. The structure loading histories are stored in each instance of the loading controller class. The provided loading history includes: displacement history (`Uhis`), nodal force history (`FnodalHis`), bar strain history (`barExHis`), bar stress history (`barSxHis`), rotational spring moment history (`sprMHis`), and rotational spring angle history (`sprRotHis`).

4 Application Examples

In this section, we present two application examples to further demonstrate how to use the implementation package and to highlight the efficiency and capability of the implementation. *Readers can find additional examples on GitHub*.

4.1 Determine the heating for folding to a right angle

In this example, we demonstrate how we can use the "`continuingLoading`" function to create external loading increment to solve a realistic problem. Suppose that we want to find the heating power needed to fold the single-crease pattern to a right angle. Then, we can set up the simulation as:

```
ori.continuingLoading=1;
thermal.targetCreaseHeating=[3,deltaQ];
thermal.thermalStep=1;
...
for i=1:maxAllowedStep
   ori.loadingController{1}=
   {"ThermalLoading",thermal};
   ori.Solver_Solve();
   functions_for_foldAngle(ori);
   if foldAngle>pi/2
      break;
   end
end
```

In the above code, we set up a 1-increment loading controller such that each time the target heating is increased by "`deltaQ`". This 1-increment loading controller is applied using the external "`for`" loop. Within every increment of the "`for`" loop, we let the package solve for a new equilibrium configuration of the the origami under the increased heating, calculate the folding angle, and check to see if the angle has reached 90 degrees. When the origami is folded to a right angle, the code will break the external loop and the total sum of the "`deltaQ`" is the heating energy required to fold the system to a right angle.

By setting "`ori.continuingLoading=1;`", the package will skip all of the initialization steps when running the external "`for`" loop, allowing the solver to compute a new equilibrium configuration at the present increment based on the equilibrium configuration from the previous step. Instead, if we were to initialize the problem again, we would wipe out the loading history and restart the loading from the "flat" unloaded state. This example demonstrates that the "`continuingLoading`" function gives the users substantial flexibility to customize the code for solving practical problems where the loading type and target final state are known, but the total loading and process are unknown. The sample code can be found in "`Example02_SingleCrease2.m`". This type of capability would be particularly useful for building and running control algorithms for active origami systems.

4.2 Sequential loading of an origami gripper

In this subsection, we use an origami gripper example to further demonstrate the capability and efficiency of the simulation package to perform sequential analyses (see Fig. 6). In this 3-step sequential loading, we first apply thermal heating to fold the origami gripper to its functioning state. Then, forces are applied onto the gripper tip to open it up, simulating the interaction with an object that may be expanding. The opened gripper is then closed by applying additional heating. As demonstrated in Fig. 6 the package can accurately simulate this behavior and the wall clock time for running the entire 3-step simulation is around 65 seonds on a desktop computer with an i9-10900KF processor. This example highlights the efficiency of the SWOMPS package when the simulation involves multi-physical behaviors.

This example also demonstrates another capability of the package, which is to capture the coupling between thermal and mechanical effects in the actuator creases. The origami gripper is placed on top of a substrate, which acts like a thermal boundary. In most situations, the substrate will remain at room temperature because it contains a much larger volume of material than the origami and thus the temperature elevation of the substrate can be neglected. Because of the presence of the substrate, the creases that are closer to the substrate will dissipate more heating power and thus remain at a lower temperature. From Fig. 6 we see that, although identical heating power is applied to all creases, there are different temperatures at the different creases due the presence of the substrate. Creases that are further away from the substrate develop higher temperatures, whereas those near the bottom develop lower temperatures. Although the creases develop different temperatures and thus different folding angles due to the thermo-mechanical coupling, this origami gripper can still function as designed because the origami pattern has one-degree-of-freedom kinematics. The sample code can be found in "`Example03_Gripper.m`" on GitHub.

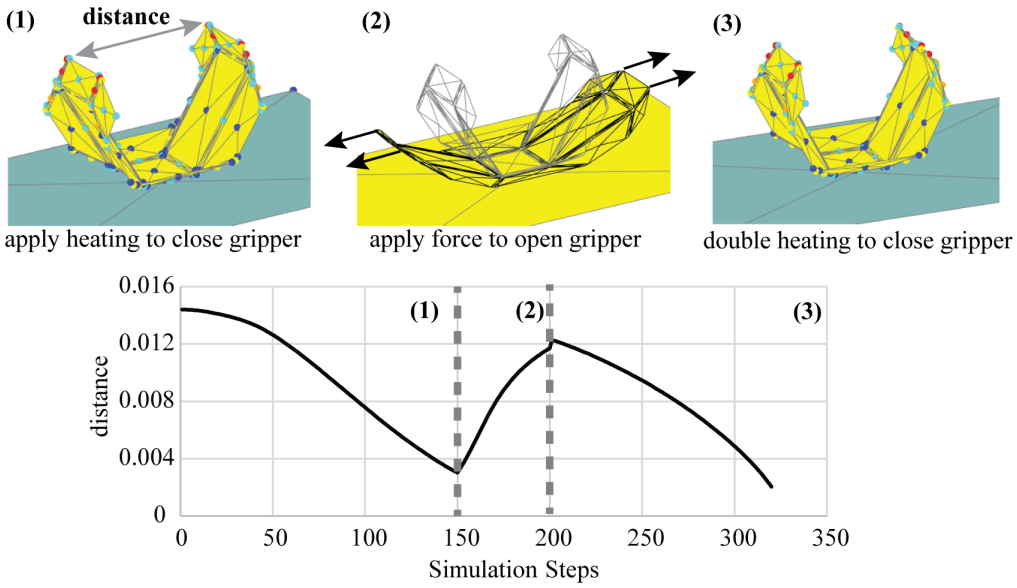

FIGURE 6. Sequential loading of an origami gripper. First, heating is applied to close the gripper. Then, forces are applied to open the gripper. Finally, heating is doubled to offset the effects of the applied forces.

5 Supplementary Materials

The supplementary material associated with this publication can be found on GitHub at `https://github.com/zzhuyii/OrigamiSimulator` or from our group website at `https://drsl.engin.umich.edu/software/swomps-package/`. The supplementary material associated with this work includes the following items:

(1) Simulation code package;
(2) Tutorial video;
(3) Tutorial document;
(4) Additional examples;
(5) Summary document for the examples.

A number of additional examples are provided on GitHub, including both the mechanical and thermal loading of various origami systems. Readers are referred to the GitHub page if further examples are needed.

6 Concluding Remarks and Future Development

In this work, we presented a MATLAB implementation of the new Sequentially Working Origami Multi-Physics Simulator (SWOMPS) that allows for versatile analysis of activate origami structures. This simulation framework can solve various behaviors in active origami including thermo-mechanically coupled actuation, large deformation folding, heat transfer, inter-panel contact, self-folding due to internal stresses, and applied mechanical loading (e.g. gravity). The implementation includes five different loading methods, including three for mechanical loading, one for self-folding, and one for thermal loading. SWOMPS offers unprecedented versatility for applying an arbitrary number and sequence of these five loading methods so that they can simulate complex sequential loading of active origami systems. The code package, supplementary information, and tutorial resources are made available for direct use or for future enhancements.

This paper also provides guidance on how to use the implementation for simulating realistic behaviors of active origami systems. We demonstrate how the origami system can be meshed, how its properties are defined, and how to set up different loading methods for the analyses in detail. Three examples, with the associated codes, are presented to show the simulator's capabilities and to guide readers on how to use the package. A number of additional examples are provided as supplementary material for further reference.

We envision that future extensions of SWOMPS will enable it to capture new origami behaviors such as magnetic actuation, optical actuation, material plasticity. The package provides an accessible platform for easy addition of these functions. For example, we can readily use the continuing loading features to sequentially update the properties (stress-free angle) of the origami to model plastic folding of creases. We believe the proposed SWOMPS package can enable researchers to design new origami patterns, study origami behaviors, and optimize origami performances more efficiently.

7 Acknowledgement

The authors acknowledge support from Defense Advanced Research Project Agency (DARPA) Grant D18AP00071 and the National Science Foundation (NSF) Grant #2054148. The paper reflects the views and position of the authors, and not necessarily those of the funding entities.

REFERENCES

[1] Schenk, M., and Guest, S. D., 2013. "Geometry of miura-folded metamaterials". *PNAS,* ***110***(9), pp. 3276–3281.

[2] Fang, H., Chu, S.-C. A., Xia, Y., and Wang, K.-W., 2018. "Programmable self-locking origami mechanical metamaterials". *Advanced Materials,* ***30***(1706311).

[3] Jager, E. W. H., Inganas, O., and Lundstrom, I., 2000. "Microrobots for micrometer-size objects in aqueous media: Potential tools for single-cell manipulation". *Science,* ***288***, pp. 2335–2338.

[4] Zhu, Y., Mayur, B., Oldham, K. R., and Filipov, E. T., 2020. "Elastically and plastically foldable electro-thermal micro-origami for controllable and rapid shape morphing". *Advanced Functional Material,* ***30***(2003741).

[5] Lang, R. J., Magleby, S., and Howell, L., 2016. "Single degree-of-freedom rigidly foldable cut origami flashers". *Journal of Mechanisms and Robotics,* ***8***(031005).

[6] Seymour, K., Burrow, D., Avila, A., Bateman, T., Morgan, D. C., Magleby, S. P., and Howell, L. L., 2018. "Origami based deployable ballistic barrier". In Origami7, Vol. 3, pp. 763–778.

[7] Na, J.-H., Evans, A. A., Bae, J., Chiappelli, M. C., Santangelo, C. D., Lang, R. J., Hull, T. C., and Hayward, R. C., 2015. "Programming reversibly self-folding origami with micropatterned photo-crosslinkable polymer trilayers". *Advanced Materials,* ***27***, Oct., pp. 79–85.

[8] Bassik, N., Stern, G. M., and Gracias, D. H., 2009. "Microassembly based on hands free origami with bidirectional curvature". *Applied Physics Letters,* ***95***(091901).

[9] Breger, J. C., Yoon, C., Xiao, R., Kwag, H. R., Wang, M. O., Fisher, J. P., Nguyen, T. D., and Gracias, D. H., 2015. "Self-folding thermo-magnetically responsive soft microgrippers". *ACS Applied Materials and Interfaces,* ***7***(5), Jan., pp. 3398–3405.

[10] Felton, S., Tolley, M., Demaine, E., Rus, D., and Wood, R., 2014. "A method for building self-folding machines". *Science,* ***345***(6197), pp. 644–646.

[11] An, B., Miyashita, S., Ong, A., Tolley, M. T., Demaine, M. L., Demaine, E. D., Wood, R. J., and Rus, D., 2018. "An end-to-end approach to self-folding origami structures". *IEEE Transactions on Robotics,* ***34***(6), Dec., pp. 1409–1424.

[12] Yang, X., Chang, L., and Perez-Arancibia, N. O., 2020. "An 88-milligram insect-scale autonomous crawling robot driven by a catalytic artificial muscle". *Science Robotics,* ***5***(eaba0015).

[13] Tachi, T., 2009. "Simulation of rigid origami". In Origami 4, R. J. Lang, ed., Decmber 8-10 2015, Caltech, Pasadena, CA, CRC Press., pp. 175–187.

[14] Tachi, T., 2009. "Generalization of rigid foldable quadrilateral mesh origami". *Journal of the International Association for Shell and Spatial Structures,* ***50***, pp. 173–179.

[15] Schenk, M., and Guest, S. D., 2010. "Origami folding: A structural engineering approach". In Origami5, Proceedings of 5OSME, July 13-17, 2010, Singapore, CRC press, pp. 293–305.

[16] Liu, K., and Paulino, G., 2017. "Nonlinear mechanics of non-rigid origami: an efficient computational approach". *Proceedings of Royal Society A,* ***473***(20170348).

[17] Filipov, E., Liu, K., Schenk, M., and Paulino, G., 2017. "Bar and hinge models for scalable analysis of origami". *International Journal of Solids and Structures,* ***124***(1), Oct., pp. 26–45.

[18] Liu, K., and Paulino, G., 2016. "Merlin: A matalab implementation to capture highly nonlinear behavior of non-rigid origami". In Proceedings of IASS Annual Symposium, September 26-30, 2016, Tokyo, Japan. International association for shell and spatial structures, pp. 1–10.

[19] Liu, K., and Paulino, G., 2018. "Highly efficient structural analysis of origami assemblages using the merlin2 software". In Origami 7, the 7th International Meeting on Origami in Science, Mathematics and Education (7OSME), September 5-7, 2018, Oxford University, United Kingdom. Tarquin.

[20] Leon, S. E., Lages, E. N., DeAraujo, C. N., and Paulino, G. H., 2014. "On the effect of constraint parameters on the generalized displacement control method". *Mechanics Research Communications,* ***56***, pp. 123–129.

[21] Zhu, Y., and Filipov, E., 2019. "Simulating compliant crease origami with a bar and hinge model". In IDETC/CIE, no. DETC2019-97119, August 16-19, 2019, Anaheim, CA, USA. ASME.

[22] Zhu, Y., and Filipov, E. T., 2020. "A bar and hinge model for simulating bistability in origami structures with compliant creases". *Journal of Mechanisms and Robotics,* ***12***(021110).

[23] Zhu, Y., and Filipov, E. T., 2021. "Rapid multi-physics simulation for electro-thermal origami systems". Accepted manuscript at International Journal of Mechanical Science.

[24] Zhu, Y., and Filipov, E., 2019. "An efficient numerical approach for simulating contact in origami assemblages". *Proceedings of the Royal Society A,* ***475***(20190366).