Washington State University

Modeling a Soft Robotic Element with Selectable Bending Axes

Relating Tendon Forces to Joint Angles in Static Equilibrium

Emily Allen and Brandon Townsend

Dr. John Swensen

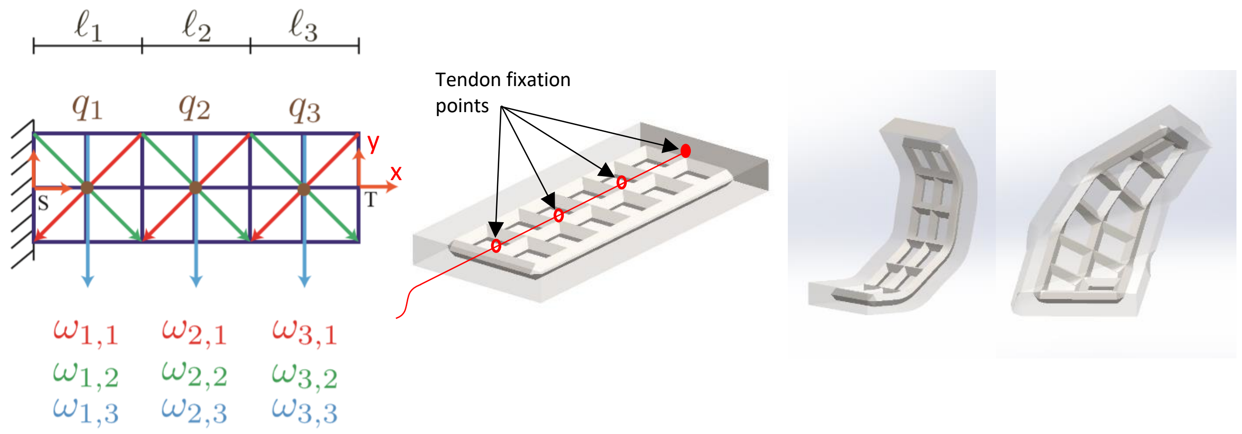
ME 579: Robot Kinematics and Dynamics

December 13, 2018

Table of Contents

1. Introduction

A 3-link soft robotic element has been constructed with an internal rigid skeleton that may be selectively melted to allow bending about specified axes as shown in Figure 1. The element consists of a low-melting-point (LMP) metal lattice encased in silicone rubber with nichrome



ℓ_1
ℓ_2
ℓ_3
q_1
q_2
q_3
y
x
S
T
Tendon fixation points
$\omega_{1,1}$ $\omega_{2,1}$ $\omega_{3,1}$
$\omega_{1,2}$ $\omega_{2,2}$ $\omega_{3,2}$
$\omega_{1,3}$ $\omega_{2,3}$ $\omega_{3,3}$

Figure 1. 3-link soft robotic element with selectable bending axes enabled by LMP metal internal skeleton that may be selectively melted.

heating elements arranged to allow selection of bending axes. The nine selectable bending axes represented by $\omega_{i,j}$, are shown in Figure 1, where *i* refers to the segment number and *j* is the axis direction. For this problem, up to 3 of the 9 bending axes may be selected at once (up to one axis per segment) by localized melting of the lattice. As shown in Figure 2, a tendon is attached to

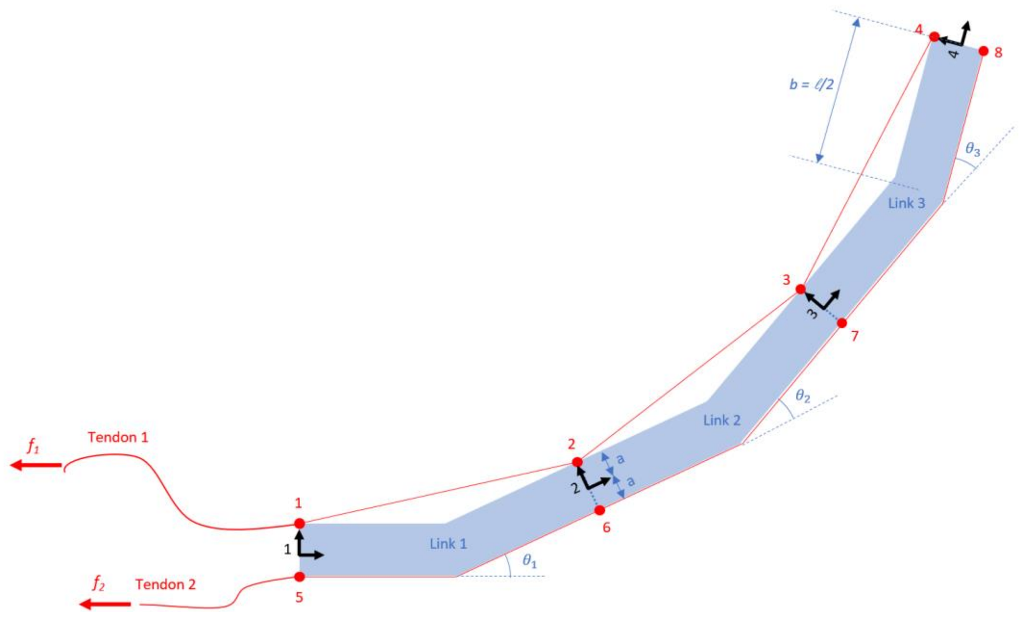


Figure 2. Tendon diagram for 3-link soft robotic element. Red dots indicate locations of tendon fixation loops.

each side of the element to induce bending about the selected compliant axes. The forward kinematics of this element have been constructed using the product of exponentials method to determine the configuration of the piece based on the selected axes and corresponding joint angles [1]. For this project, we want to take this model a step further by relating the forces applied on tendons to the deformation of the piece. This will involve modeling the joints (melted axes) as torsional springs with some constant stiffness. By relating the tendon forces and the element's configuration, we may determine what configurations are possible for any given set of selected axes.

Each set of equations will be derived as a function of ω, the set of selected axes which may include a single axis or up to 3 of the 9 possible bending axes (one axis per segment). Each time that a segment is deformed and then cooled, the reference configuration of the model changes and must be updated. By successively melting different joints and controlling tendon forces, a vast range of configurations may be achieved by this simple element.

Selective melting drastically improves the work space of the device as compared to simultaneous melting by allowing for finer control using only a single tendon. When individual joints may be selected to melt on their own, or even in pairs rather than all three, then the shape of the device may be more precisely controlled.

Simultaneous melting only allows for the bending of all selected joints at once, with each joint experiencing approximately the same angular displacement. This provides both little control and limited workspace. Selective melting allows for configurations and tool tip positions that would not otherwise be achievable by allowing for the manipulation of only one or two joints at once. For example, the tool tip would be able to reach a position 45 degrees up and a distance of $\frac{1}{2}l_1 + l_2 + l_3$ away, and then activate a lever with selective melting. Axis $\omega_{1,3}$ would be melted first, then allowed to cool. Axis $\omega_{3,3}$ would then be melted to activate the lever.

2. Methods

2.1 Extension Functions for Tendons

The torque applied on the joints by the tendons depends on the tendon routing configuration and the axes that have been selected. For example, if the tendon does not lie perpendicular to the selected bending axis, a larger tendon force will be required to achieve the same torque about the joint. These geometric relationships between tendon force and joint torque are derived as a function of each possible axis selection, joint angle, and tendon offset.

The joint torques τ may be directly related to the forces applied to the tendons by developing extension functions for each tendon. This method of analyzing inelastic tendons is described by Murray et al. [2]. This method involves deriving the extension function for each tendon, which expresses the length of the tendon as a function of the joint angles. In our case, since the axis directions may vary, the extension functions are a function of both the joint angles and the selected axes.

For a simple planar problem, developing these extension functions may be done by simply analyzing the geometry. For example, if the axes $\omega_{1,3}, \omega_{2,3}$, and $\omega_{3,3}$, are selected, the geometric relationships may be extracted by inspection of Figure 2. However, when different axes are selected, the problem is no longer planar, and these geometric relations become nontrivial.

Rather than developing complicated, three-dimensional geometric relationships for each bending axis combination, the forward kinematics exponentials may be used to express the length of each tendon for any set of selected axes and joint angles. The length of tendon 1, as shown in Figure 2, is simply the sum of the distances between tendon fixation points 1 and 2, 2 and 3, 3 and 4. These distances are already known from the forward kinematics for this element, which have been previously developed [1]. For example, the coordinates of tendon points 1 and 2 with respect to frame 1 may be expressed in homogeneous coordinates as

$$x_{1,1}(\theta_1, \omega_1) = \begin{bmatrix} 0 \\ 0 \\ a \\ 1 \end{bmatrix}, \tag{1}$$

$$x_{2,1}(\theta_1,\omega_1) = e^{\widehat{\xi_1(\omega_1)}\,\theta_1} g_{1,2_0} \begin{bmatrix} 0 \\ 0 \\ a \\ 1 \end{bmatrix}, \tag{2}$$

where $\xi_1(\omega_1)$ is the twist used to represent the rotation and translation of points due to bending about the selected axis ω_1. The matrix exponential $e^{\widehat{\xi_1}\,\theta_1}$ used to transform points from frame 1 to 2 has been previously developed as a function of ω in the construction of the forward kinematics relationships for this element [1]. The reference configuration $g_{1,2_0}$ is the transformation between frames 1 and 2 when $\theta_1 = 0$. For this case,

$$g_{1,2_0} = \begin{bmatrix} 1 & 0 & 0 & \ell_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

The distance between $x_{1,1}$ and $x_{2,1}$ may then be computed as:

$$d_{1,2}(\theta_1,\omega_1) = \sqrt{\left(x_{1,1}{}^T\, x_{1,1}\right)^2 + \left(x_{2,1}{}^T\, x_{2,1}\right)^2}. \tag{4}$$

Finally, when the joint angles are all positive, the extension function for tendon 1 may then be computed by summing the distances between each tendon fixation point:

$$h_1(\theta,\omega) = d_{1,2} + d_{2,3} + d_{3,4}, \tag{5}$$

where $d_{2,3}$ and $d_{3,4}$ are computed using the matrix exponential for rotation about ω_2 and ω_3.

The extension function for tendon 2 may simply be expressed as

$$h_2(\theta,\omega) = \ell_1 + a\,\theta_1 + \ell_2 + a\,\theta_2 + \ell_3 + a\,\theta_3\,, \tag{6}$$

when the joint angles are all positive. In theory, there are 8 different cases for these extension functions based on different combinations of positive and negative joint angles. For example, if θ_1 and θ_2 are positive while θ_3 is negative, the extension functions would behave differently than if all the joint angles were positive. Thus, extension functions are different for each of the 8 cases of positive/negative joint angle combinations. However, for this project we only have two tendons, so the only possible joint angle combinations are case 1 (all joint angles are positive) and case 8 (all joint angles are negative). For case 8, the joint angles are all negative (i.e. tendon 2 is activated instead of tendon 1), and the extension functions are as follows:

$$h_1(\theta, \omega) = \ell_1 - a\,\theta_1 + \ell_2 - a\,\theta_2 + \ell_3 - a\,\theta_3\,, \tag{7}$$

$$h_2(\theta, \omega) = d_{5,6} + d_{6,7} + d_{7,8}\,. \tag{8}$$

2.2 Coupling Matrix

According to Murray et al. [2], by applying the conservation of energy, the joint torques can be expressed as

$$\tau = P(\theta, \omega) f, \tag{9}$$

where *f* is a vector containing the forces on each tendon:

$$f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}, \tag{10}$$

and where $P(\theta, \omega)$ is the coupling matrix computed from

$$P(\theta, \omega) = \frac{\partial h}{\partial \theta}^{T} (\theta, \omega), \tag{11}$$

where *h* is a vector containing the extension functions for the appropriate case of positive/negative joint angle combinations:

$$h = \begin{bmatrix} h_1(\theta, \omega) \\ h_2(\theta, \omega) \end{bmatrix}. \tag{12}$$

The computation of $P(\theta, \omega)$ is non-trivial. Since the extension functions *h* depend on the joint angles and selected axes, and they involve matrix exponentials and square roots, taking the derivatives for $P(\theta, \omega)$ by hand would be tedious and nearly impossible.

The extension functions were entered into Mathematica for the computation of these derivatives. These derivatives were computed for all 8 cases of positive/negative joint angle combinations.

The resulting coupling matrix for each case, expressed as a function of $\theta, \omega, a, \ell_1, \ell_2$, and ℓ_3, was then converted to MATLAB using the "ToMatlab" package. The Mathematica code for computation of the coupling matrices is shown in Appendix A.

2.3 Estimating Joint Stiffnesses

The joint stiffness modeled by the torsional springs is dependent on the geometry of the element at the joint and the material properties of the elastomer. The torque exerted by the spring (joint) can be expressed as

$$\tau = -k(\theta), \tag{13}$$

where θ is the resulting joint angle relative to the equilibrium configuration, and *k* is the spring constant which may be determined experimentally. The spring constant *k* should be proportional to the elastic modulus of the material and the area moment of inertia of the joint cross-section. *k* may be determined experimentally for a simple geometry and then scaled to find k for other joint geometries (with different area moments of inertia).

For this particular project, there are only two different cross-sections for the nine different allowable bending axes. There is a cross-section for the straight (transverse) axes, and a slightly wider cross-section for the diagonal axes. In the future, the bending stiffnesses for these two types of axes may be measured experimentally. However, for this project, we simply use the following stiffness values:

$$k_{straight} = 30\ N \cdot mm\,, \tag{14}$$

$$k_{diagonal} = 40\ N \cdot mm\,, \tag{15}$$

$$k_{solid} = 10\ 000\ N \cdot mm\,, \tag{16}$$

where k_{solid} is the stiffness of an unmelted joint. These values will need to be replaced with experimental values to ensure that the model matches the physical element.

The appropriate stiffness value ($k_{straight}, k_{diagonal}$, or k_{solid}) is selected in MATLAB based on the input axis selections using if statements. For example, using the conventions we have chosen, k_i represents the joint stiffness for segment *i* of the robotic element. If the input ω_i is 1 or 2, this indicates bending about the axis $\omega_{i,1}$ or $\omega_{i,2}$ which are diagonal axes, so k_i is set to be $k_{diagonal}$. If ω_i is 3, this indicates bending about $\omega_{i,3}$, the transverse axis, so k_i is set to be $k_{straight}$. Finally, if ω_i is 0, this indicates and unmelted joint, so k_i is set to be k_{solid}.

The bending resistance at the joints will appear in the potential energy terms in the derivation of the system dynamics using the Lagrange-Euler method.

2.4 Euler-Lagrange Method to Relate Potential Energy to Joint Torques

The Euler-Lagrange method may be used to develop the equations of motion for the robotic element. Since we are only dealing with the statics of this problem, the higher order terms in this method may be neglected. In other words, we can neglect the effects of kinetic energy and rotational inertia on the system. We will also choose to neglect gravity for this project. By simplifying the problem in this way, the resulting governing equation will take the form:

$$\frac{-\partial \mathcal{L}(q,\omega)}{\partial q} = K(\omega)q = \tau^T, \tag{17}$$

where $\mathcal{L}(q,\omega)$ is the Lagrangian, q is the joint variables, and τ is the joint torques.

Beginning with the equation for the potential energy of the system, we account for the torques generated by the bending at the joints:

$$PE = -\frac{1}{2}k_1\theta_1^2 - \frac{1}{2}k_2\theta_2^2 - \frac{1}{2}k_3\theta_3^2, \tag{18}$$

with k_n being the stiffness of the joint and θ_n being the angular displacement at that joint.

Given that only potential energy is being considered in the system, the Lagrangian is simply

$$\mathcal{L} = \frac{1}{2}(-k_1\theta_1^2 - k_2\theta_2^2 - k_3\theta_3^2). \tag{19}$$

Differentiating the Lagrangian with respect to time, t, yields

$$\frac{\partial \mathcal{L}}{\partial t} = [k_1\theta_1 \quad k_2\theta_3 \quad k_3\theta_3]. \tag{20}$$

Applying the Euler-Lagrange equation to the simplified static model yields the following relationship

$$-\frac{\partial L}{\partial t} = \tau^T. \tag{21}$$

We may then use substitution to write Equation (*20)* as

$$[-k_1\theta_1 \quad -k_2\theta_3 \quad -k_3\theta_3] = \tau^T. \tag{22}$$

By applying the relationship from Equation (*9*), the joint torques may be expressed in terms of the tendon forces and coupling matrix. Combining Equations (*9*) and (*22*) produces the relationship

$$\begin{bmatrix} -k_1(\omega) & 0 & 0 \\ 0 & -k_2(\omega) & 0 \\ 0 & 0 & -k_3(\omega) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = K\,\theta = P(\theta,\omega)f, \tag{23}$$

with k_n being a function of ω.

In our system, we want to be able to determine θ given a particular f and ω. We also hope to be able to determine *f* for a given θ and ω. To determine θ from a given f and ω, we can left-multiply each side of Equation (*23)* by the inverse of the stiffness matrix,

$$K^{-1}(\omega) = \begin{bmatrix} \frac{1}{-k_1(\omega)} & 0 & 0 \\ 0 & \frac{1}{-k_2(\omega)} & 0 \\ 0 & 0 & \frac{1}{-k_3(\omega)} \end{bmatrix}, \tag{24}$$

to yield

$$\theta = \begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\end{bmatrix} = K^{-1}(\omega)P(\theta,\omega)f\ . \tag{25}$$

The joint angles θ cannot be solved for analytically in this expression since the coupling matrix P is a function of θ. Since θ appears on both sides of Equation (*25)*, numerical analysis must be applied. This equation can easily be solved using fixed point iteration to determine the joint angles that result from a given set of selected axes ω and tendon forces *f*.

To find the forces required to produce a desired set of joint angles, we simply left multiply Equation (*23*) by P-inverse to obtain:

$$f = P^{-1}(\theta,\omega)\begin{bmatrix}-k_1(\omega) & 0 & 0\\ 0 & -k_2(\omega) & 0\\ 0 & 0 & -k_3(\omega)\end{bmatrix}\begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\end{bmatrix} = P^{-1}(\theta,\omega)\ K\ \theta. \tag{26}$$

3. Results/Discussion

3.1 Configuration Computation

By activating different heating elements and applying different tendon forces, a wide variety of configurations may be achieved. A relationship has been developed that enables computation of the joint angles that result from application of a given tendon force. For example, Figure 3 shows the computed equilibrium configuration when a 5-Newton force is applied to the upper tendon, and the heating elements are activated to allow bending along axes $\omega_{1,1}$ (diagonal axis on first

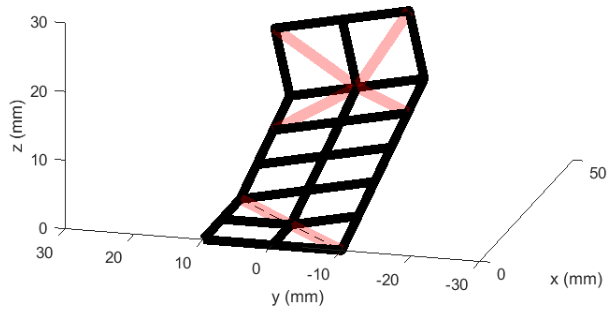


Figure 3. Configuration of robotic element resulting from 5N force applied to upper tendon while axes $\omega_{1,1}$ and $\omega_{3,3}$ are melted.

link) and $\omega_{3,3}$ (transverse axis on third link). Other unique configurations may be achieved by applying different tendon forces and selectively melting different axes along the lattice, as seen in Figures 4 and 5.

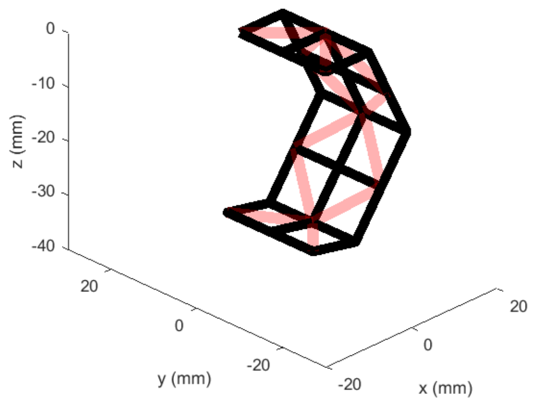


Figure 4. Configuration of robotic element resulting from 4.5N force applied to lower tendon while axes $\omega_{1,3}$, $\omega_{2,3}$ and $\omega_{3,3}$ are melted.

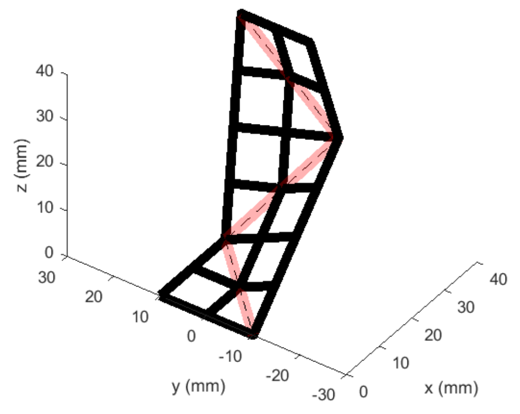


Figure 5. Configuration of robotic element resulting from 9N force applied to upper tendon while axes $\omega_{1,1}$, $\omega_{2,2}$ and $\omega_{3,1}$ are melted.

Various configurations may be used to perform intelligent tasks. For example, applying a 22-Newton force to the upper tendon while melting parallel diagonal axes could allow the robot to grab a pencil as shown below in Figure 6.

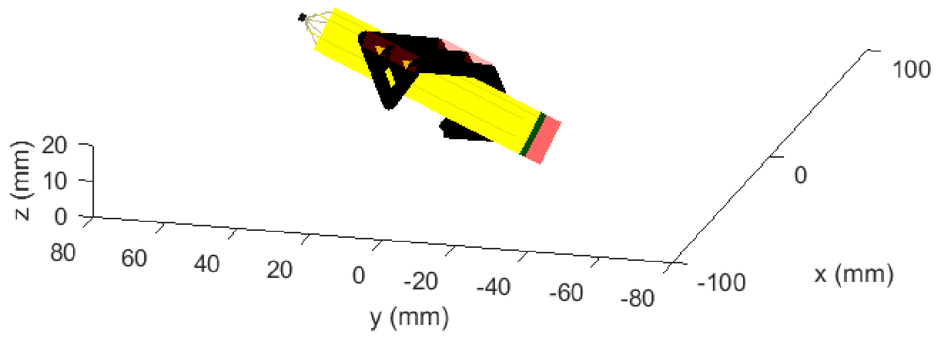


Figure 6. Robot configured to grab pencil by applying 22N force to upper tendon while axes $\omega_{1,1}$, $\omega_{2,1}$ and $\omega_{3,1}$ are melted.

3.2 Workspace Limitations

It is interesting to note that Equation (*25*) fails to compute realistic joint angles if a joint angle exceeds 90° along a diagonal axis. At first glance, this may appear to be a computational error, but in reality, this computational limitation perfectly represents a physical limitation. Careful inspection of the tendon routing shown in Figure 1 and Figure 2 reveals the reason for this limitation. For this particular arrangement of tendon fixation points, the tendon length across a diagonal joint reaches a minimum when the joint angle is 90°. Bending a diagonal joint beyond 90° would actual require elongation of the tendon, so the physical model would never actually bend more than 90° along a diagonal axis. However, along the transverse axes, the physical element can bend beyond 90°, up to the point of folding on itself. This concept is again modeled perfectly by the mathematical model, which allows bending beyond 90° along transverse axes.

3.3 Force Computation

The ability to compute the tendon forces required to achieve a configuration is desirable for implementing advanced control and maneuverability with the soft robotic element. The relationship from Equation (*26*) enables this computation. For example, the configuration shown in Figure 3 was achieved by applying a 5-Newton force to the upper tendon while melting axes $\omega_{1,1}$ and $\omega_{3,3}$. Applying Equation (*25*) revealed that this load/axis combination results in joint angles of 17.3°, 0.1°, and 79.3°, respectively. We can then test the validity of the force computation function by solving this problem backwards and comparing the force results with the original input force values. To do this, we use 17.3°, 0.1°, and 79.3° as the θ input in Equation (*26*), along with the axis selection $\omega_{1,1}$ and $\omega_{3,3}$ to determine the forces needed to achieve these joint angles. The forces computed should theoretically be 5 Newtons on the upper tendon and 0 Newtons on the lower tendon. Application of Equation (*26*) with these input values indicated that forces of 4.998 N and -0.002 N on the upper and lower tendons are required to achieve the particular configuration shown in Figure 3. These results are not perfect, but they show that Equation (*26*) is useful for calculating the tendon forces required to achieve a desired set of joint angles with reasonable accuracy.

3.4 Joint Space Limitations

It is important to note that in the case of this 3-segment element with 2 opposing tendons, not all joint angle combinations are possible. The joint space is limited, and the required force computation only works when the desired joint angles lie within the allowable joint space. This detail is evidenced by both the physical model and the mathematics behind the force computation.

From the physical model, we can intuitively see that the angles of the 3 joints cannot be controlled independently with the use of only one tendon. Here we consider only one tendon since it would be counter-productive to pull both tendons simultaneously. Increasing the tendon tension increases the joint angles of all three joints simultaneously in a specific ratio that is proportional to the relative bending stiffnesses of the three joints. In order to compute the tendon force required to achieve a desired set of joint angles, these joint angles must follow the proper ratio proportional to their bending stiffnesses. Otherwise, no tendon force will be able to cause joint angles that do not follow this ratio. For example, no tendon force will be able to cause bending in only the first joint if three joints are melted.

Mathematically, this concept is demonstrated in the form of the coupling matrix P. Computing the required tendon forces involves multiplying by the inverse of P, as seen in Equation (*26*). Recall that P is a 3 x 2 matrix in this case. As such, the force computation only works when

$$\theta \in \mathrm{Image}(K^{-1}(\omega)P(\theta,\omega)). \tag{27}$$

However, the restriction on θ for the force computation is actually even more stringent due to the tendon antagonism. Since one of the tendon forces is always zero, θ must be a scalar multiple of a single column of $(K^{-1}(\omega)P(\theta,\omega))$ for computation of the tendon forces to be possible.

3.5 Effects of Successive Melting

By successively melting joints, a larger number of more complex configurations are made possible. Notably, it allows for the combination of movements which require both the top and bottom tendons to be pulled. As seen in Figures 7 and 8, a single joint was melted to achieve a position by pulling the top and bottom tendons, respectively. With successive melting, these configurations can be combined to achieve the configuration shown in in Figure 9. This combined configuration is not achievable with simultaneous melting, as both the top and bottom tendons must be pulled to achieve the final form. If both axes were melted simultaneously and each tendon pulled, then coupling would occur.

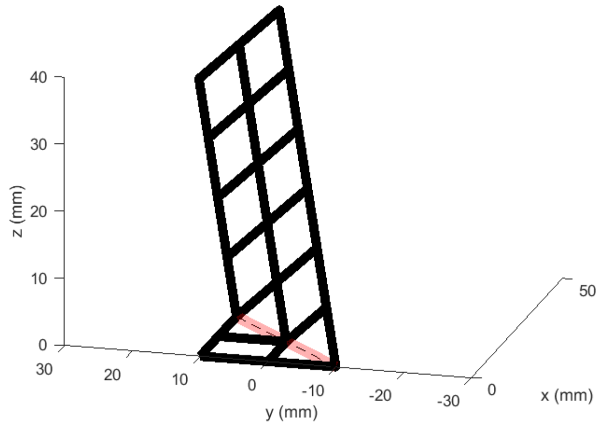


Figure 7. Configuration achieved by individually melting joint $\omega_{1,1}$ and pulling the upper tendon.

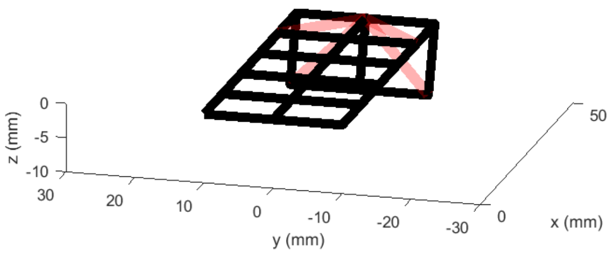


Figure 8. Configuration achieved by individually melting joint $\omega_{3,3}$ and pulling the lower tendon.

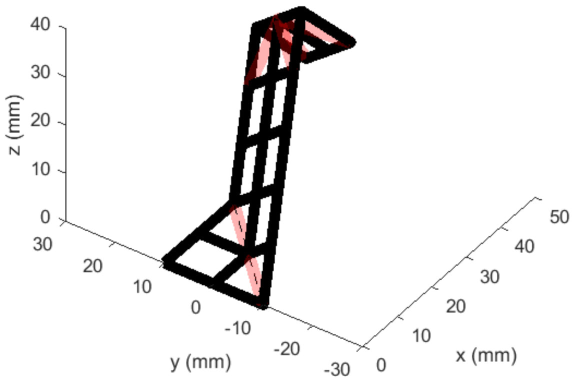


Figure 9. Configuration achieved by successively melting and deforming joints $\omega_{1,1}$ and $\omega_{3,3}$.

4. Conclusion

Ultimately, a relationship has been obtained to relate tendon forces to resulting joint angles in the static equilibrium of a 3-segment soft robotic element with selectable axes. With some limitations, this relationship can be used both ways: to compute joint angles given tendon forces and selected axes, or to compute necessary tendon forces to achieve a desired set of joint angles for a given set of selected axes.

Using Equations *(25)* and (*26*), the joint angles are related to the tendon forces required to produce a desired position using the coupling matrix. The derivation of the application of the coupling matrix is detailed in chapter 4 section 4.1 of A Mathematical Introduction to Robotic Manipulation [2].

While we approximated joint stiffnesses for the sake of simplicity, the true stiffness values of the joints would be determined experimentally. By melting the respective joint and causing angular displacement using a force meter oriented perpendicular to the body of the displaced segment, we could then plot force vs. displacement to obtain the stiffness of the joint.

The next step for this project would be to incorporate the dynamics of the system. For this static model, we have neglected the kinetic energy and rotational inertia components. Including these elements in the model will allow us to implement torque control in the future so that we may accurately control the movement of the element and achieve desired trajectories.

In the future, it may be helpful to develop an intelligent method of determining which axes need to be melted in order to achieve a desired trajectory. This may be done by determining which set of axes make the desired trajectory lie within the image of the spatial manipulator jacobian. However, for this work the selected axes will be a simple input.

Once these relationships are developed, they may then be applied to larger scale robotic elements that include more segments or even three-dimensional structures. We have seen in this project the unique configurations enabled by the selective melting of a small soft robotic element. With just one set of tendons and 3 joints, a wide variety of movements and configurations may be achieved, as evidenced by the ability to grasp a pencil, curl into a tight ball or extend out as a rigid arm. By applying these concepts into higher dimensional structures, highly dexterous movements even mimicking human gestures may be achieved.

References

[1] E. A. Allen and J. P. Swensen, "Directional Stiffness Control Through Geometric Patterning and Localized Heating of Field's Metal Lattice Embedded in Silicone," *Actuators,* vol. 7, no. 4, p. 80, 2018.

[2] R. M. Murray, Z. Li and S. S. Sastry, A mathematical introduction to robotic manipulation, CRC press, 1994.

Appendix A. Mathematica Computation of Coupling Matrix

```
ClearAll["Global`*"]

(*Points along bending axes*)
```

$q1_1 = \left\{\frac{L1}{2}, 0, 0\right\};$

$q2_2 = \left\{\frac{L2}{2}, 0, 0\right\};$

$q3_3 = \left\{\frac{L3}{2}, 0, 0\right\};$

```
(*Axis directions*)
w1 = {w1x, w1y, 0};
w2 = {w2x, w2y, 0};
w3 = {w3x, w3y, 0};

(*Twists*)
```

$c1 = -w1 \times q1_1;$

$c2 = -w2 \times q2_2;$

$c3 = -w3 \times q3_3;$

$$xsi1 = \begin{pmatrix} c1[[1]] \\ c1[[2]] \\ c1[[3]] \\ w1[[1]] \\ w1[[2]] \\ w1[[3]] \end{pmatrix};$$

$$xsi2 = \begin{pmatrix} c2[[1]] \\ c2[[2]] \\ c2[[3]] \\ w2[[1]] \\ w2[[2]] \\ w2[[3]] \end{pmatrix};$$

$$xsi3 = \begin{pmatrix} c3[[1]] \\ c3[[2]] \\ c3[[3]] \\ w3[[1]] \\ w3[[2]] \\ w3[[3]] \end{pmatrix};$$

```
(*Reference Configurations*)
```

$$g12_0 = \begin{pmatrix} 1 & 0 & 0 & L1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

$$g23_0 = \begin{pmatrix} 1 & 0 & 0 & L2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

$$g34_0 = \begin{pmatrix} 1 & 0 & 0 & L3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

```
(*Skew function*)
```

$$SK[x\_] := \begin{pmatrix} 0 & -x[[6]] & x[[5]] & x[[1]] \\ x[[6]] & 0 & -x[[4]] & x[[2]] \\ -x[[5]] & x[[4]] & 0 & x[[3]] \\ 0 & 0 & 0 & 0 \end{pmatrix};$$

```
xsi1hat = SK[xsi1];
xsi2hat = SK[xsi2];
xsi3hat = SK[xsi3];

(*Matrix exponentials*)
ex1 = MatrixExp[xsi1hat * theta1];
ex2 = MatrixExp[xsi2hat * theta2];
ex3 = MatrixExp[xsi3hat * theta3];

(*Coordinates of First Tendon Attachment Points*)
x11 = {0, 0, a, 1};
x22 = {0, 0, a, 1};
x33 = {0, 0, a, 1};
x55 = {0, 0, -a, 1};
x66 = {0, 0, -a, 1};
x77 = {0, 0, -a, 1};

(*Coordinates of Second Tendon Attachment Points*)
```

$x21 = ex1.g12_0.x11;$

$x32 = ex2.g23_0.x22;$

$x43 = ex3.g34_0.x33;$

$x65 = ex1.g12_0.x55;$

$x76 = ex2.g23_0.x66;$

$x87 = ex3.g34_0.x77;$

```
(*Extension Function Top Tendon*)
(*Evaluate 8 different joint angle cases*)
(*Case1 +++ *)
```

$h11 = \sqrt{Total[(x21 - x11)^2]} + \sqrt{Total[(x32 - x22)^2]} + \sqrt{Total[(x43 - x33)^2]};$

$h21 = 6 b + a (theta1 + theta2 + theta3);$

`(*Case2 ++- *)`

$h12 = \sqrt{Total[(x21 - x11)^2]} + \sqrt{Total[(x32 - x22)^2]} + 2 b - a\, theta3;$

$h22 = 4 b + a (theta1 + theta2) + \sqrt{Total[(x87 - x77)^2]};$

`(*Case3 +-+ *)`

$h13 = \sqrt{Total[(x21 - x11)^2]} + 2 b - a\, theta2 + \sqrt{Total[(x43 - x33)^2]};$

$h23 = 2 b + a\, theta1 + \sqrt{Total[(x76 - x66)^2]} + 2 b + a\, theta3;$

`(*Case4 -++ *)`

$h14 = 2 b - a\, theta1 + \sqrt{Total[(x32 - x22)^2]} + \sqrt{Total[(x43 - x33)^2]};$

$h24 = \sqrt{Total[(x65 - x55)^2]} + 2 b + a\, theta2 + 2 b + a\, theta3;$

`(*Case5 +-- *)`

$h15 = \sqrt{Total[(x21 - x11)^2]} + 2 b - a\, theta2 + 2 b - a\, theta3;$

$h25 = 2 b + a\, theta1 + \sqrt{Total[(x76 - x66)^2]} + \sqrt{Total[(x87 - x77)^2]};$

`(*Case6 -+- *)`

$h16 = 2 b - a\, theta1 + \sqrt{Total[(x32 - x22)^2]} + 2 b - a\, theta3;$

$h26 = \sqrt{Total[(x65 - x55)^2]} + 2 b + a\, theta2 + \sqrt{Total[(x87 - x77)^2]};$

`(*Case7 --+ *)`

$h17 = 4 b - a (theta1 + theta2) + \sqrt{Total[(x43 - x33)^2]};$

$h27 = \sqrt{Total[(x65 - x55)^2]} + \sqrt{Total[(x76 - x66)^2]} + 2 b + a\, theta3;$

`(*Case8 --- *)`

$h18 = 6 b - a (theta1 + theta2 + theta3);$

$h28 = \sqrt{Total[(x65 - x55)^2]} + \sqrt{Total[(x76 - x66)^2]} + \sqrt{Total[(x87 - x77)^2]};$

```
(*All Cases*)
h1 = If[theta1 > 0, Sqrt√Total[(x21 - x11)^2] , 2 b - a theta1] +
   If[theta2 > 0, √Total[(x32 - x22)^2] , 2 b - a theta2] +
   If[theta3 > 0, √Total[(x43 - x33)^2] , 2 b - a theta3];
h2 = If[theta1 < 0, √Total[(x65 - x55)^2] , 2 b + a theta1] +
   If[theta2 < 0, √Total[(x76 - x66)^2] , 2 b + a theta2] +
   If[theta3 < 0, √Total[(x87 - x77)^2] , 2 b + a theta3];

(*Coupling Matrix for 8 Cases*)
```

$$p1 = \begin{pmatrix} \partial_{theta1} h11 & \partial_{theta1} h21 \\ \partial_{theta2} h11 & \partial_{theta2} h21 \\ \partial_{theta3} h11 & \partial_{theta3} h31 \end{pmatrix};$$

$$p2 = \begin{pmatrix} \partial_{theta1} h12 & \partial_{theta1} h22 \\ \partial_{theta2} h12 & \partial_{theta2} h22 \\ \partial_{theta3} h12 & \partial_{theta3} h32 \end{pmatrix};$$

$$p3 = \begin{pmatrix} \partial_{theta1} h13 & \partial_{theta1} h23 \\ \partial_{theta2} h13 & \partial_{theta2} h23 \\ \partial_{theta3} h13 & \partial_{theta3} h33 \end{pmatrix};$$

$$p4 = \begin{pmatrix} \partial_{theta1} h14 & \partial_{theta1} h24 \\ \partial_{theta2} h14 & \partial_{theta2} h24 \\ \partial_{theta3} h14 & \partial_{theta3} h34 \end{pmatrix};$$

$$p5 = \begin{pmatrix} \partial_{theta1} h15 & \partial_{theta1} h25 \\ \partial_{theta2} h15 & \partial_{theta2} h25 \\ \partial_{theta3} h15 & \partial_{theta3} h35 \end{pmatrix};$$

$$p6 = \begin{pmatrix} \partial_{theta1} h16 & \partial_{theta1} h26 \\ \partial_{theta2} h16 & \partial_{theta2} h26 \\ \partial_{theta3} h16 & \partial_{theta3} h36 \end{pmatrix};$$

$$p7 = \begin{pmatrix} \partial_{theta1} h17 & \partial_{theta1} h27 \\ \partial_{theta2} h17 & \partial_{theta2} h27 \\ \partial_{theta3} h17 & \partial_{theta3} h37 \end{pmatrix};$$

$$p8 = \begin{pmatrix} \partial_{theta1} h18 & \partial_{theta1} h28 \\ \partial_{theta2} h18 & \partial_{theta2} h28 \\ \partial_{theta3} h18 & \partial_{theta3} h38 \end{pmatrix};$$

```
Directory[]
C:\
   f1 = OpenWrite["P1.m"]
OuputStream[P1.m, 4]
WriteMatlab[p1, f1, "P1"]
Close[f1]
P1.m

Directory[]
C:\
   f2 = OpenWrite["P2.m"]
OuputStream[P2.m, 4]
WriteMatlab[p2, f2, "P2"]
Close[f2]
P2.m

Directory[]
C:\
   f3 = OpenWrite["P3.m"]
OuputStream[P3.m, 4]
WriteMatlab[p3, f3, "P3"]
Close[f3]
P3.m
```

```
Directory[]
C:\
   f4 = OpenWrite["P4.m"]
OuputStream[P4.m, 4]
WriteMatlab[p4, f4, "P4"]
Close[f4]
P4.m

Directory[]
C:\
   f5 = OpenWrite["P5.m"]
OuputStream[P5.m, 4]
WriteMatlab[p5, f5, "P5"]
Close[f5]
P5.m

Directory[]
C:\
   f6 = OpenWrite["P6.m"]
OuputStream[P6.m, 4]
WriteMatlab[p6, f6, "P6"]
Close[f6]
P6.m

Directory[]
C:\
   f7 = OpenWrite["P7.m"]
OuputStream[P7.m, 4]
WriteMatlab[p7, f7, "P7"]
Close[f7]
P7.m

Directory[]
C:\
   f8 = OpenWrite["P8.m"]
OuputStream[P8.m, 4]
WriteMatlab[p8, f8, "P8"]
Close[f8]
P8.m
```

Appendix B. MATLAB Functions

```
function ME579Project
```

B-1. Constant Parameters

```
global L1 L2 L3 w u q1 q2 q3 a b kdiag kstraight ksolid

% Segment dimensions
L1 = 20;    % Length of Link 1
L2 = 20;    % Length of Link 2
```

```
L3 = 20;    % Length of Link 3
w = L1;     % Width of links
u = 5;      % Thickness of links
a = 2.5;    % 1/2 thickness of links
b = L1/2;   % 1/2 length of links

% Joint Stiffnesses (N-mm/radian)
kdiag = 40;       % Stiffness of diagonal joint
kstraight = 30;   % Stiffness of transverse joint
ksolid = 10000;   % Stiffness of unmelted joint

% Point on twist axes
q1 = [L1/2;0;0];       % Center point of Link 1
q2 = [L1+L2/2;0;0];    % Center point of Link 2
q3 = [L1+L2+L3/2;0;0]; % Center point of Link 3
```

B-2. Stiffness Matrix Function

```
% Accepts inputs of the form omega = [omega1; omega2; omega3]
% where omega1 is axis selected for Link 1, omega2 for Link 2, etc.
% and where omega values may be 1,2,3, or 0.
% omega = 1       ->   transverse axis
% omega = 2 or 3  ->   diagonal axis
% omega = 0       ->   no melted axis

% K matrix is of a form such that
% K*[theta1; theta2; theta3] = Tau,
% where Tau is a vector of torques exerted by each joint due to bending
% stiffness.

    function K = stif(omega)

        if omega(1) == 1 || omega(1) ==2; k1 = kdiag;
        elseif omega(1) == 3; k1 = kstraight;
        else; k1 = ksolid;
        end

        if omega(2) == 1 || omega(2) ==2; k2 = kdiag;
        elseif omega(2) == 3; k2 = kstraight;
        else; k2 = ksolid;
        end

        if omega(3) == 1 || omega(3) ==2; k3 = kdiag;
        elseif omega(3) == 3; k3 = kstraight;
        else; k3 = ksolid;
        end

        K = [-k1 0 0; 0 -k2 0; 0 0 -k3];
    end
```

B-3. Static Configuration Function

```
% Computes joint angles resulting from forces applied to tendons.
% Accepts force input in the form f = [f1;f2], where f1 and f2 are the
% forces applied to tendons 1 and 2.
% Accepts axis selection in the form omega = [omega1;omega2;omega3],
% where omega1 is axis selected for Link 1, omega2 for Link 2, etc.
% and where omega values may be 1,2,3, or 0.
% omega = 1       ->   transverse axis
% omega = 2 or 3  ->   diagonal axis
% omega = 0       ->   no melted axis

    function theta = staticconfig(f,omega)
        L = [L1,L2,L3];     % Link lengths
        [m,n] = size(f);
        if n>m; f = f';end  % Transpose f if necessary

        % Select appropriate case and theta_guess
        if f(1)>=f(2)                        % Case 1
            c = 1;
            theta = max(f)/20*[1;1;1];       % Approximate theta's
        else                                 % Case 1
            c = 8;
            theta = max(f)/20*[-1;-1;-1]; % Approximate theta's
        end

        if omega(1)==0; theta(1) = 0;        % Set theta1 = 0 if link 1 is unmelted
        elseif omega(2)==0; theta(2) = 0; % Set theta2 = 0 if link 2 is unmelted
        elseif omega(3)==0; theta(3) = 0; % Set theta3 = 0 if link 3 is unmelted
        end

        % Use fixed point iteration to find theta's:

        K = stif(omega);                     % Stiffness matrix
        err = 1;                             % Initial error
        itercount = 0;                       % Initialize iteration counter
        while err>0.001

            itercount = itercount+1;         % Count iterations
            thetaold = theta;                % Store old theta values

            % Case 1
            if c==1
                theta = inv(K)*(P1(theta,omega,a,u,L)*f);
            end

            % Case 8
            if c==8
                theta = inv(K)*(P8(theta,omega,a,u,L)*f);
            end

            err = max(abs(theta-thetaold));% Compute error between iterations
            % Stop computation after 50000 iterations
            if itercount>=50000
                err = 0;
                fprintf('Computation terminated after 50000 iterations\n');
            end

        end
    end
```

B-4. Required Force Function

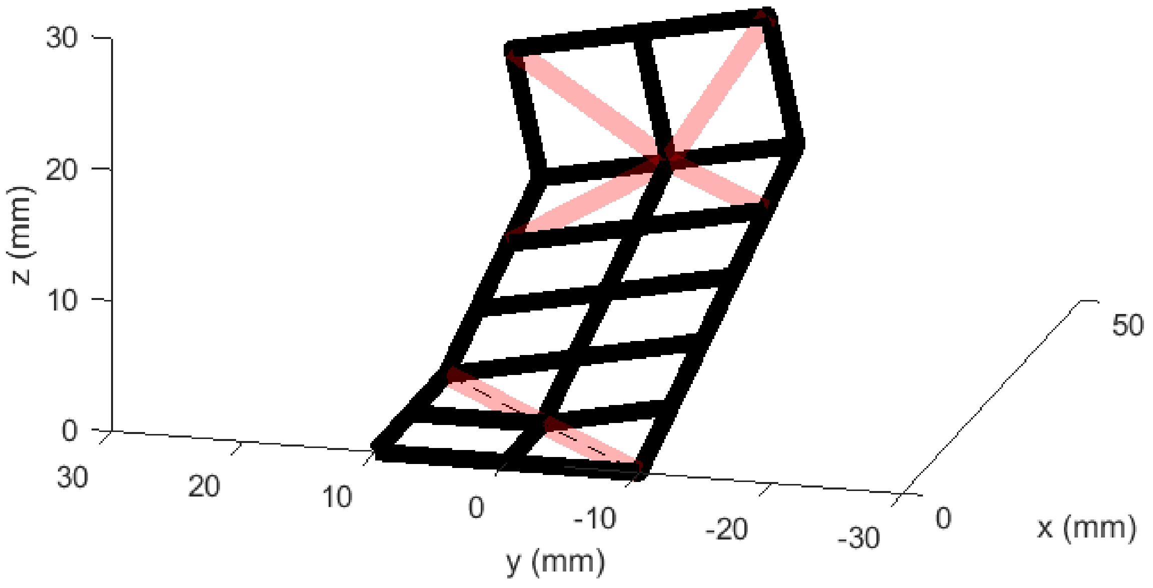
```
% Computes tendon forces required to achieve desired joint angles.
% Accepts joint angles of the form theta = [theta1;theta2;theta3]
% Accepts selected axes in the form omega = [omega1;omega2;omega3],
% where omega1 is axis selected for Link 1, omega2 for Link 2, etc.
% and where omega values may be 1,2,3, or 0.
% omega = 1       ->   transverse axis
% omega = 2 or 3  ->   diagonal axis
% omega = 0       ->   no melted axis

    function f = fr(theta,omega)
        L = [L1,L2,L3];             % Link lengths
        K = stif(omega);            % Stiffness matrix
        [m,n] = size(theta);
        if n>m; theta = theta';end  % Transpose theta if necessary

        % select appropriate case
        if sum(theta)>=0; c = 1;
        else; c = 8;
        end

        if c == 1      % Case 1 - theta's are positive
            f = pinv(P1(theta,omega,a,u,L))*K*theta;
        elseif c == 8  % Case 2 - theta's are negative
            f = pinv(P8(theta,omega,a,u,L))*K*theta;
        end

    end
```

B-5. General Demonstration

```
% Demonstrate the configuration of the robotic element when a 5N force is
% applied to tendon1 while axes omega(1,1) and omega (3,3) are activated.

ftest = [5,0];                           % forces
omegatest = [1,0,3];                     % bending axes
thetatest = staticconfig(ftest,omegatest) % compute resulting joint angles
testplot = config(thetatest,omegatest);   % plot resulting configuration
```

```
% Convert resulting joint angles back to tendon forces to show that
% required tendon forces may be calculated for given joint angles.

% Required forces should be ~[5,0]

testforces = fr(thetatest,omegatest)      % compute forces required to achieve desired joint
angles
```

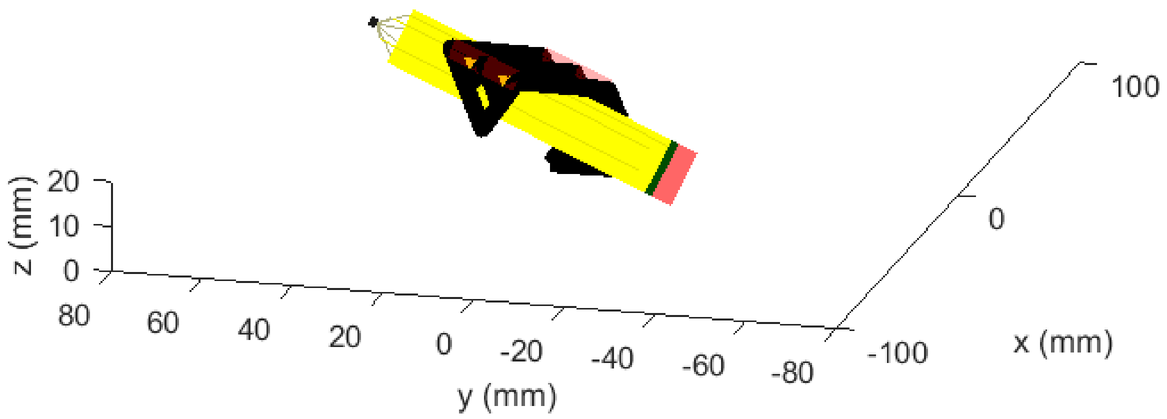
```
thetatest =

    0.3018
    0.0013
    1.3838


testforces =

    4.9980
   -0.0019
```



B-6. Pencil Grab Demonstration

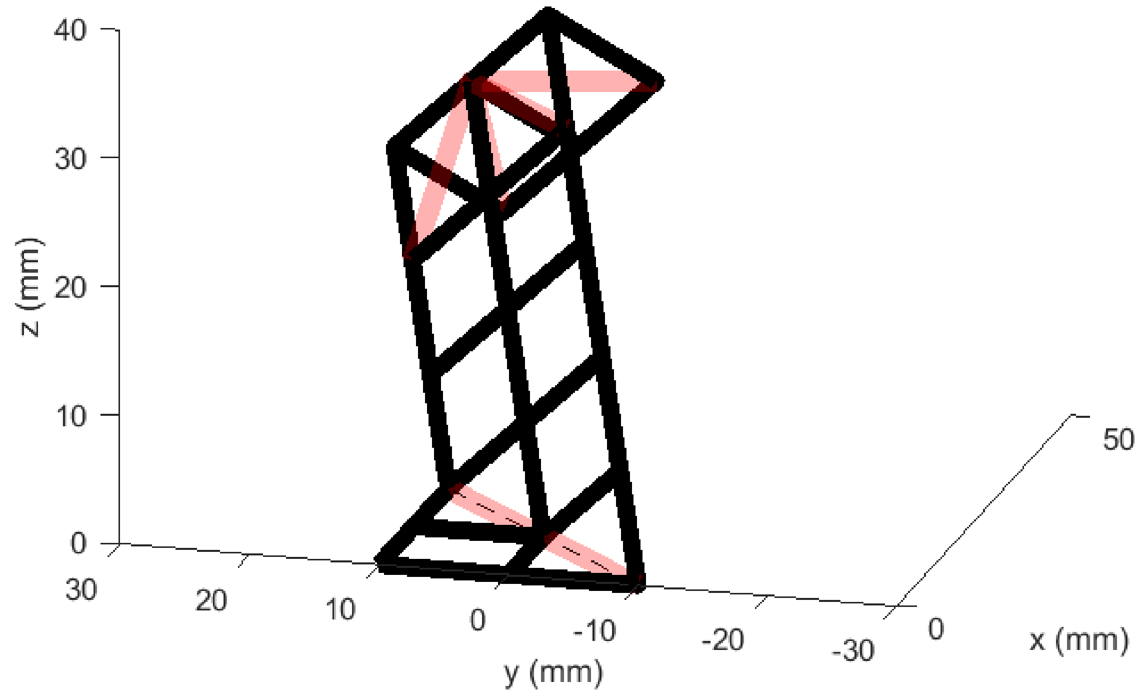
```
% Demonstrate the dexterity of the element by wrapping it around a pencil
% using a 22N force applied to tendon 1.

ftest2 = [22,0];                                % forces
omegatest2 = [1,1,1];                           % bending axes
thetatest = staticconfig(ftest2,omegatest2)     % compute resulting joint angles
[testplot,ptest] = config(thetatest,omegatest2); % plot resulting configuration
testpenc = pencilplot(ptest,omegatest2);        % plot pencil
```

```
thetatest =

    1.5594
    1.5594
    1.5594
```



B-7. Successive Melting Demonstration

```
% Demonstrate a configuration made possible by successive melting/cooling
% of joints.

% Step 1 - Apply 15N force to tendon 1 to cause bending about omega(1,1)
f1 = [15,0];                         % forces
omega1 = [1,0,0];                    % bending axes
theta1 = staticconfig(f1,omega1);    % compute resulting joint angles
% plotstep1 = config(theta1,omega1); % plot resulting configuration

% Step 2 - Apply 6N force to tendon 2 to cause bending about omega(3,3)
f2 = [0,6];                          % forces
omega2 = [0,0,3];                    % bending axes
theta2 = staticconfig(f2,omega2);    % compute resulting joint angles
% plotstep2 = config(theta2,omega2); % plot resulting configuration

% Resulting shape - combine results from step 1 and step 2
plotresult = config([theta1(1),theta1(2),theta2(3)],[omega1(1),omega1(2),omega2(3)])
```



B-8. Plot Configuration Function

```
    function [configplot,p] = config(theta,omega)
        % Axes selection
        A1 = omega(1);
        A2 = omega(2);
        A3 = omega(3);

        % Joint angles
        t1 = theta(1);
        t2 = theta(2);
        t3 = theta(3);

        % Set joint angle to 0 if axis is not melted
        if omega(1) == 0; A1 = 3; t1 = 0; end
        if omega(2) == 0; A2 = 3; t2 = 0; end
        if omega(3) == 0; A3 = 3; t3 = 0; end

        % Axis lengths
        if A1==3; LA1 = w; else; LA1 = sqrt(L1^2+w^2); end
        if A2==3; LA2 = w; else; LA2 = sqrt(L2^2+w^2); end
        if A3==3; LA3 = w; else; LA3 = sqrt(L3^2+w^2); end

        % Twist axis directions
        if A1==3; w1 = [0; -1; 0]; elseif A1==1; w1 = [-1; -1; 0]./sqrt(2); else; w1 = [1; -1;
0]./sqrt(2); end
        if A2==3; w2 = [0; -1; 0]; elseif A2==1; w2 = [-1; -1; 0]./sqrt(2); else; w2 = [1; -1;
0]./sqrt(2); end
        if A3==3; w3 = [0; -1; 0]; elseif A3==1; w3 = [-1; -1; 0]./sqrt(2); else; w3 = [1; -1;
0]./sqrt(2); end

        % Twists
        x1 = [cross(-w1,q1);w1];
        x2 = [cross(-w2,q2);w2];
        x3 = [cross(-w3,q3);w3];

        % Reference configuration
        gST0 = [1 0 0 L1+L2+L3;0 1 0 0;0 0 1 0;0 0 0 1];
        gSU0 = [1 0 0 L1+L2+L3/2;0 1 0 0;0 0 1 0;0 0 0 1];
        gSV0 = [1 0 0 L1+L2/2;0 1 0 0;0 0 1 0;0 0 0 1];
        gSW0 = [1 0 0 L1/2;0 1 0 0;0 0 1 0;0 0 0 1];

        % Skew operation
        sk4 = @(z) [0 -z(6) z(5) z(1);z(6) 0 -z(4) z(2);-z(5) z(4) 0 z(3);0 0 0 0];
        skx1 = sk4(x1);
        skx2 = sk4(x2);
        skx3 = sk4(x3);

        % Matrix exponential
        exp1 = expm(skx1*t1);
        exp2 = expm(skx2*t2);
        exp3 = expm(skx3*t3);
        % Product of exponentials
        gST = exp1*exp2*exp3*gST0;
        gSU = exp1*exp2*exp3*gSU0;
        gSV = exp1*exp2*gSV0;
        gSW = exp1*gSW0;

        % Define lattice points
        p = zeros(4,21);
        p(4,:) = ones(1,21);
        p(1:3,1:3) = [0, 0, 0;w/2, 0, -w/2;0, 0, 0];
        if A1~=2; p(1:3,4) = [L1/2; w/2; 0]; else; p(:,4) = gSW*[0;w/2;0;1]; end
        p(1:3,5) = [L1/2; 0; 0];
        if A1~=1; p(1:3,6) = [L1/2; -w/2; 0]; else; p(:,6) = gSW*[0;-w/2;0;1]; end
        if A1==1; p(1:3,7) = [L1; w/2; 0]; else; p(:,7) = gSW*[L1/2;w/2;0;1]; end
        p(:,8) = gSW*[L1/2; 0; 0; 1];
        if A1==2; p(1:3,9) = [L1; -w/2; 0]; else; p(:,9) = gSW*[L1/2;-w/2;0;1]; end
        if A2~=2; p(:,10) = gSW*[L1/2+L2/2; w/2; 0; 1]; else; p(:,10) = gSV*[0; w/2; 0; 1]; end
        p(:,11) = gSW*[L1/2+L2/2; 0; 0; 1];
        if A2~=1; p(:,12) = gSW*[L1/2+L2/2; -w/2; 0; 1]; else; p(:,12) = gSV*[0; -w/2; 0; 1]; end
        if A2==1; p(:,13) = gSW*[L1/2+L2; w/2; 0; 1]; else; p(:,13) = gSV*[L2/2; w/2; 0; 1]; end
        p(:,14) = gSV*[L2/2; 0; 0; 1];
        if A2==2; p(:,15) = gSW*[L1/2+L2; -w/2; 0; 1]; else; p(:,15) = gSV*[L2/2; -w/2; 0; 1];
end
        if A3~=2; p(:,16) = gSV*[L2/2+L3/2; w/2; 0; 1]; else; p(:,16) = gSU*[0; w/2; 0; 1]; end
        p(:,17) = gSV*[L2/2+L3/2; 0; 0; 1];
        if A3~=1; p(:,18) = gSV*[L2/2+L3/2; -w/2; 0; 1]; else; p(:,18) = gSU*[0; -w/2; 0; 1]; end
        if A3==1; p(:,19) = gSV*[L2/2+L3; w/2; 0; 1]; else; p(:,19) = gSU*[L3/2; w/2; 0; 1]; end
        p(:,20) = gSU*[L3/2; 0; 0; 1];
        if A3==2; p(:,21) = gSV*[L2/2+L3; -w/2; 0; 1]; else; p(:,21) = gSU*[L3/2; -w/2; 0; 1];
end

        p = p(1:3,:);

        % Reorder points for plotting
        borderpts = [p(1:3,1:3),p(1:3,6),p(1:3,9),p(1:3,12),p(1:3,15),...
            p(1:3,18), p(1:3,21),p(1:3,20),p(1:3,19),p(1:3,16),p(1:3,13),...
            p(1:3,10), p(1:3,7),p(1:3,4),p(1:3,1)];
        yline1 = p(1:3,4:6);
        yline2 = p(1:3,7:9);
        yline3 = p(1:3,10:12);
        yline4 = p(1:3,13:15);
        yline5 = p(1:3,16:18);
        xline = [p(1:3,2),p(1:3,5),p(1:3,8),p(1:3,11),p(1:3,14),p(1:3,17),p(1:3,20)];

        if A1==1; axis1 = [p(1:3,7),p(1:3,5),p(1:3,3)];
        elseif A1==2; axis1 = [p(1:3,1),p(1:3,5),p(1:3,9)];
        else; axis1 = p(1:3,4:6);
        end

        if A2==1; axis2 = [p(1:3,13),p(1:3,11),p(1:3,9)];
        elseif A2==2; axis2 = [p(1:3,7),p(1:3,11),p(1:3,15)];
        else; axis2 = p(1:3,10:12);
        end
        if A3==1; axis3 = [p(1:3,19),p(1:3,17),p(1:3,15)];
        elseif A3==2; axis3 = [p(1:3,13),p(1:3,17),p(1:3,21)];
        else; axis3 = p(1:3,16:18);
        end

        % Diagonal Lines
        h1 = [p(1:3,7),p(1:3,5),p(1:3,3)];
        h2 = [p(1:3,1),p(1:3,5),p(1:3,9)];
        h3 = [p(1:3,13),p(1:3,11),p(1:3,9)];
        h4 = [p(1:3,7),p(1:3,11),p(1:3,15)];
        h5 = [p(1:3,19),p(1:3,17),p(1:3,15)];
        h6 = [p(1:3,13),p(1:3,17),p(1:3,21)];

        configplot = figure('Name','Case2','NumberTitle','Off');
        figure(configplot);
        hold on
        plot3(borderpts(1,:),borderpts(2,:),borderpts(3,:),'k','lineWidth',6);
        plot3(yline1(1,:),yline1(2,:),yline1(3,:),'k','lineWidth',6);
        plot3(yline2(1,:),yline2(2,:),yline2(3,:),'k','lineWidth',6);
        plot3(yline3(1,:),yline3(2,:),yline3(3,:),'k','lineWidth',6);
        plot3(yline4(1,:),yline4(2,:),yline4(3,:),'k','lineWidth',6);
        plot3(yline5(1,:),yline5(2,:),yline5(3,:),'k','lineWidth',6);
        plot3(xline(1,:),xline(2,:),xline(3,:),'k','lineWidth',6);
        plot3(axis1(1,:),axis1(2,:),axis1(3,:),'k--');
        plot3(axis2(1,:),axis2(2,:),axis2(3,:),'k--');
        plot3(axis3(1,:),axis3(2,:),axis3(3,:),'k--');
        if omega(1)~=0
            if A1~=2; plot3(h1(1,:),h1(2,:),h1(3,:),'color',[1,0,0,0.3],'lineWidth',7); end
            if A1~=1; plot3(h2(1,:),h2(2,:),h2(3,:),'color',[1,0,0,0.3],'lineWidth',7); end
        end
        if omega(2)~=0
            if A2~=2; plot3(h3(1,:),h3(2,:),h3(3,:),'color',[1,0,0,0.3],'lineWidth',7); end
            if A2~=1; plot3(h4(1,:),h4(2,:),h4(3,:),'color',[1,0,0,0.3],'lineWidth',7); end
        end
        if omega(3)~=0
            if A3~=2; plot3(h5(1,:),h5(2,:),h5(3,:),'color',[1,0,0,0.3],'lineWidth',7); end
            if A3~=1; plot3(h6(1,:),h6(2,:),h6(3,:),'color',[1,0,0,0.3],'lineWidth',7); end
        end
        daspect([1 1 1]);
        ylim([-30 30]);
        xlabel('x (mm)');
        ylabel('y (mm)');
        zlabel('z (mm)');
        view(-75,17);

        set(configplot,'Units','Inches');
        pos = get(configplot,'Position');
        set(configplot,'PaperPositionMode','Auto','PaperUnits','Inches','PaperSize',[pos(3),
pos(4)]);
        print(configplot,'Robot Configuration','-dpdf','-r0');

    end
```

B-9. Pencil Plotting Function

```
    function pp = pencilplot(p,omega)
        cent = (p(:,16) + p(:,4))./2;  % coordinates of pencil center
        A1 = omega(1);                 % selected axis for Link 1

        if A1==3 w1 = [0; -1; 0]; elseif A1==1; w1 = [-1; -1; 0]./sqrt(2); else; w1 = [1; -1;
0]./sqrt(2); end

        dir = w1;                      % pencil direction
        R = 5;                         % pencil radius (mm)
        plen = 110;                    % pencil straight length (mm)
        elen = 8;                      % eraser length (mm)
        blen = 3;                      % band length (mm)
        tlen = 10;                     % tip length (mm)
        llen = 3;                      % lead length (mm)

        % hexagon parameters
        r = sqrt(3)*R/2;
        t = r;

        % hexagon vertices around center
        centa = cent+[R/sqrt(2),-R/sqrt(2),0]';
        centb = cent+[t/2/sqrt(2),-t/2/sqrt(2),r]';
        centc = cent+[-t/2/sqrt(2),t/2/sqrt(2),r]';
        centd = cent-[R/sqrt(2),-R/sqrt(2),0]';
        cente = cent-[t/2/sqrt(2),-t/2/sqrt(2),r]';
        centf = cent-[-t/2/sqrt(2),t/2/sqrt(2),r]';

        % points at ends of each hexagon line
        pptsa = zeros(3,2);
        pptsa(:,1) = centa+plen/2*dir;
        pptsa(:,2) = centa-plen/2*dir;
        pptsb = zeros(3,2);
        pptsb(:,1) = centb+plen/2*dir;
        pptsb(:,2) = centb-plen/2*dir;
        pptsc = zeros(3,2);
        pptsc(:,1) = centc+plen/2*dir;
        pptsc(:,2) = centc-plen/2*dir;
        pptsd = zeros(3,2);
        pptsd(:,1) = centd+plen/2*dir;
        pptsd(:,2) = centd-plen/2*dir;
        pptse = zeros(3,2);
        pptse(:,1) = cente+plen/2*dir;
        pptse(:,2) = cente-plen/2*dir;
        pptsf = zeros(3,2);
        pptsf(:,1) = centf+plen/2*dir;
        pptsf(:,2) = centf-plen/2*dir;

        % points at ends of pencil body
        ppts = zeros(3,2);
        ppts(:,1) = cent + plen/2*dir;
        ppts(:,2) = cent - plen/2*dir;
```

```
% points at ends of eraser
epts = zeros(3,2);
epts(:,1) = cent + (plen/2+blen+elen)*dir;
epts(:,2) = cent + (plen/2+blen)*dir;

% points at ends of eraser band
bpts = zeros(3,2);
bpts(:,1) = cent + (plen/2+blen)*dir;
bpts(:,2) = cent + (plen/2)*dir;

% points at ends of pencil lead
lpts = zeros(3,2);
lpts(:,1) = cent + (-plen/2-tlen-llen)*dir;
lpts(:,2) = cent + (-plen/2-tlen)*dir;

% points at ends of pencil tip lines
tptsa = zeros(3,2);
tptsa(:,1) = cent-(plen/2+tlen)*dir;
tptsa(:,2) = centa-plen/2*dir;
tptsb = zeros(3,2);
tptsb(:,1) = cent-(plen/2+tlen)*dir;
tptsb(:,2) = centb-plen/2*dir;
tptsc = zeros(3,2);
tptsc(:,1) = cent-(plen/2+tlen)*dir;
tptsc(:,2) = centc-plen/2*dir;
tptsd = zeros(3,2);
tptsd(:,1) = cent-(plen/2+tlen)*dir;
tptsd(:,2) = centd-plen/2*dir;
tptse = zeros(3,2);
tptse(:,1) = cent-(plen/2+tlen)*dir;
tptse(:,2) = cente-plen/2*dir;
tptsf = zeros(3,2);
tptsf(:,1) = cent-(plen/2+tlen)*dir;
tptsf(:,2) = centf-plen/2*dir;

% pencil body
pp = plot3(ppts(1,:),ppts(2,:),ppts(3,:),'y','lineWidth',20);
% pencil lines
plot3(pptsa(1,:),pptsa(2,:),pptsa(3,:),'Color',[0.9,0.9,0]);
plot3(pptsb(1,:),pptsb(2,:),pptsb(3,:),'Color',[0.9,0.9,0]);
plot3(pptsc(1,:),pptsc(2,:),pptsc(3,:),'Color',[0.9,0.9,0]);
plot3(pptsd(1,:),pptsd(2,:),pptsd(3,:),'Color',[0.9,0.9,0]);
plot3(pptse(1,:),pptse(2,:),pptse(3,:),'Color',[0.9,0.9,0]);
plot3(pptsf(1,:),pptsf(2,:),pptsf(3,:),'Color',[0.9,0.9,0]);
% eraser
plot3(epts(1,:),epts(2,:),epts(3,:),'Color',[1,0.4,0.4],'lineWidth',20);
% eraser border
plot3(bpts(1,:),bpts(2,:),bpts(3,:),'Color',[0,0.3,0],'lineWidth',20);
% lead
plot3(lpts(1,:),lpts(2,:),lpts(3,:),'Color',[0.1,0.1,0.1],'lineWidth',3);
ylim([-80,80])
% tip lines
plot3(tptsa(1,:),tptsa(2,:),tptsa(3,:),'Color',[0.7,0.7,0.5]);
```

```
        plot3(tptsb(1,:),tptsb(2,:),tptsb(3,:),'Color',[0.7,0.7,0.5]);
        plot3(tptsc(1,:),tptsc(2,:),tptsc(3,:),'Color',[0.7,0.7,0.5]);
        plot3(tptsd(1,:),tptsd(2,:),tptsd(3,:),'Color',[0.7,0.7,0.5]);
        plot3(tptse(1,:),tptse(2,:),tptse(3,:),'Color',[0.7,0.7,0.5]);
        plot3(tptsf(1,:),tptsf(2,:),tptsf(3,:),'Color',[0.7,0.7,0.5]);
    end
```

```
end
```

Published with MATLAB® R2016b