Biomechanically Inspired Foldable Leg Mechanism(Based on final project)

This notebook delineates the workflow for the design and manufacturing of a foldable leg mechanism, drawing inspiration from the agility of the jerboa and the energy-efficient locomotion of the ostrich. Utilizing the `foldable_robotics` library, we import CAD geometry, compute hinge parameters, generate laminate layers, and prepare for the first and second cutting passes. Each code cell is supplemented with detailed explanations to elucidate its purpose and the rationale underlying the associated processes.

Import Libraries

```python
import foldable_robotics.dxf as frd
import foldable_robotics as fr
import foldable_robotics.manufacturing as frm
from foldable_robotics.layer import Layer
from foldable_robotics.laminate import Laminate
import foldable_robotics.parts.castellated_hinge2 as frc
import shapely.geometry as sg
```

In this first code cell, we import essential modules from the `foldable_robotics` library and the `shapely` geometry library. These modules provide functions to read DXF files, define laminate layers, compute manufacturing parameters, and work with specialized parts like castellated hinges. Importing `shapely.geometry` allows us to manipulate geometric primitives (points, lines, polygons) for trimming and combining shapes. These imports set up the environment needed to implement the foldable leg mechanism described in the report.

Display and Resolution Settings

```python
fr.display_height=300
```

The `display_height` parameter controls how tall the laminate drawings appear when plotted. Setting `fr.display_height = 300` ensures that all subsequent plots in this notebook render with an appropriate scale for visual inspection. Adjusting display height does not affect the underlying geometry; it simply resizes the images.

```python
fr.resolution = 4
```

Resolution affects how curves are approximated when reading DXF files and when visualizing geometry. Here, `fr.resolution = 4` sets a modest discretization resolution so that curved features are represented with sufficient fidelity while keeping computations efficient. If the design contained many arcs, increasing this value would yield smoother approximations at the cost of computational complexity.

Hinge Width Calculation

```
desired_degrees = 120
thickness = 1
plain_width = frm.plain_hinge_width(desired_degrees,thickness)
plain_width
```

```
1.7320508075688776
```

In this cell we calculate the width of a plain flexure hinge required to achieve a desired rotation. Using the `plain_hinge_width` function from the manufacturing module, we pass a target rotation of 120 degrees and a total laminate thickness of 1 mm. The function returns a width of approximately 1.73 mm, which is derived from trigonometric relations between the hinge gap, material thickness, and rotation angle. Choosing this width ensures that the hinge will bend to 120° without overstressing the material.

Parameter Definitions

```
support_width = 2 # must be larger than hinge width
kerf = .05
is_adhesive = [False,True,False,True,False]
arc_approx = 10
NUM_LAYERS = 5
bridge_thickness = 2
bounding_box_padding = 10
jig_spacing = 10
jig_dia = 5
```

Here we define a set of configuration parameters that govern the rest of the design and manufacturing workflow. `support_width` sets the width of support tabs that hold parts together; it must exceed the hinge width to provide structural integrity. `kerf` captures the laser cutter's beam width so that kerf compensation can be applied to cut paths. `is_adhesive` is a Boolean list indicating which of the five layers will be adhesive; in this case the second and fourth layers bond the rigid and flexible layers together. `arc_approx` controls how circular arcs from the DXF file are approximated by straight segments. `NUM_LAYERS` defines a five-layer laminate with two rigid, two adhesive, and one flexible layer. `bridge_thickness` , `bounding_box_padding` , `jig_spacing` , and `jig_dia` are design values used later for adding support bridges, creating a padded sheet around the design, placing alignment holes, and labeling each layer.

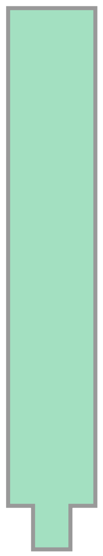
Reading Design Geometry

```python
body_vertices = frd.read_lwpolylines('car.dxf',
layer='body',
arc_approx = arc_approx)
body_vertices
```

```
[[[50.00940537840764, 368.0045980266],
  [93.40940537840761, 368.0045980266],
  [93.4094053784076, 120.0045980266],
  [81.00940537840762, 120.0045980266],
  [81.00940537840762, 98.30459802659999],
  [62.4094053784076, 98.30459802659999],
  [62.4094053784076, 120.0045980266],
  [50.0, 120.0],
  [50.00940537840764, 368.0045980266]]]
```

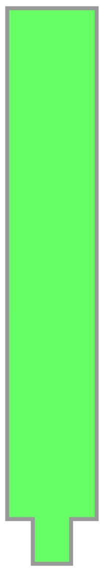
The next step reads the outline of the leg mechanism from the DXF design file. We call `read_lwpolylines` on layer `'body'` to extract lightweight polylines approximating the body outline. Because arcs in the DXF file might need discretization, the `arc_approx` parameter ensures curved segments are represented by a sufficient number of points. The returned `body_vertices` list holds the vertices of each polyline; these coordinates correspond to the 2D profile of the leg drawn in LibreCAD.

```python
body_polygons = [sg.Polygon(item) for item in body_vertices]
body_polygons[0]
```



After loading the vertex lists, we convert each polyline into a `shapely` `Polygon` object. Wrapping the coordinate lists into polygons enables rich geometric operations such as union, intersection, and difference. Displaying the first polygon helps verify that the geometry has been imported correctly before constructing laminate layers.

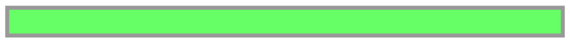
```python
body_layer = Layer(*body_polygons)
body_layer
```



We then create a `Layer` instance from the list of polygons using the `foldable_robotics.layer.Layer` class. A `Layer` represents all of the geometry present in a single laminate layer. At this point the `body_layer` contains the full continuous outline of the leg mechanism drawn in the body layer of the DXF file.

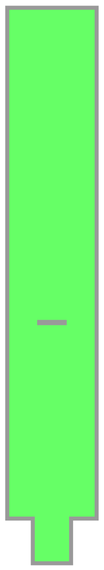
Handling Holes and Cuts

```
hole_vertices = frd.read_lwpolylines('car.dxf',
layer='holes',
arc_approx = arc_approx)
hole_layer = Layer(*[sg.Polygon(item) for item in hole_vertices])
hole_layer
```



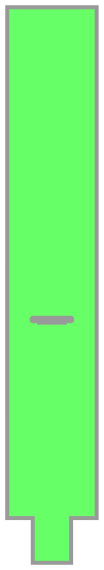
Additional features such as through-holes or slots are read from the DXF's `'holes'` layer. By converting the hole polylines to polygons and then to a `Layer`, we obtain shapes that represent material to be removed. Holes may serve as clearance features, attachment points, or part of hinge mechanisms.

```
body_layer -= hole_layer
body_layer
```



We subtract the `hole_layer` from the `body_layer` to remove material where holes are located. This operation creates cutouts in the body layer corresponding to the holes defined in the CAD model. Visualizing the updated `body_layer` ensures the holes have been correctly carved out.

```
cut_vertices = frd.read_lwpolylines('car.dxf',
                                    layer='cuts',
                                    arc_approx=arc_approx)
cut_layer = Layer(*[sg.LineString(item) for item in cut_vertices])
cut_layer
cut_layer <<=.5
cut_layer
body_layer -= cut_layer
body_layer
```



Cuts such as notches and clearance slots are defined in the DXF’s `'cuts'` layer. Here we read those polylines, convert them into a `Layer` of lines, and then offset (`<<=.5`) each line by 0.5 mm to give them finite width. Subtracting the resulting `cut_layer` from `body_layer` trims away the extra material at joint regions so that the hinges can fold freely. Offsetting by half the desired width ensures the cuts will remove the correct amount of material on both sides of the line.

Processing Joint Lines

```
joint_vertices = frd.read_lines('car.dxf', layer='joints')
joint_vertices
```

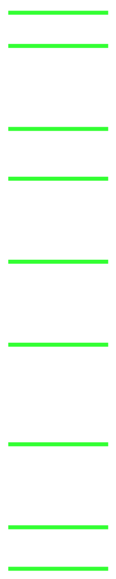
```
[]
```

The `'joints'` layer of the DXF contains straight lines indicating where hinge axes should be placed. We first read these as lists of points using `read_lines`. Printing `joint_vertices` helps confirm the number and approximate positions of joints before further processing.

```
joint_vertices = frd.read_lwpolylines('car.dxf',layer='joints',arc_approx =
arc_approx)
l = sg.LineString(joint_vertices[0])
l
```

An alternative call to `read_lwpolylines` reads the same joint lines and wraps them into a `LineString`. Converting to a `LineString` allows us to perform geometric set operations and plotting. This cell is essentially a sanity check to inspect a single joint's geometry.

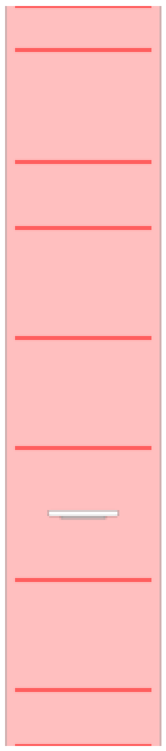
```
joint_lines_original_layer = Layer(*[sg.LineString(item) for item in
joint_vertices])
joint_lines_original_layer
```



To manage all joint lines collectively, we construct a `Layer` composed of `LineString` objects. The resulting `joint_lines_original_layer` represents all hinge lines before trimming and is used as input for subsequent intersection operations.

```
body_layer.plot()
joint_lines_original_layer.plot()
```

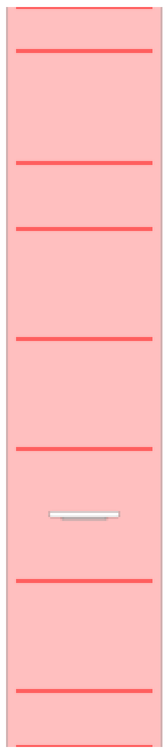
```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
```



Visual inspection is crucial for debugging the geometry. We plot the current `body_layer` and overlay the raw joint lines using the `plot()` method. The plot shows whether joint lines cross the body outline correctly and highlights any misalignments.

```
joint_lines_modified_layer = joint_lines_original_layer & body_layer
body_layer.plot()
joint_lines_modified_layer.plot()
```

```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
```



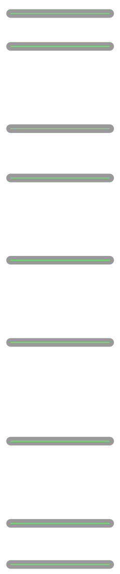
Only the portions of the joint lines that lie within the body should be retained. We compute the intersection of `joint_lines_original_layer` and `body_layer`, storing the result in `joint_lines_modified_layer`. Plotting the trimmed lines alongside the body verifies that the joint axes now perfectly span the width of the leg profile.

```
modified_joint_vertices = [list(item.coords) for item in
joint_lines_modified_layer.geoms]
modified_joint_vertices
```

```
[[(53.1094053784076, 346.3045980266), (90.30940537840759, 346.3045980266)],
 [(53.1094053784076, 358.7045980266), (90.30940537840759, 358.7045980266)],
 [(53.1094053784076, 315.3045980266), (90.30940537840759, 315.3045980266)],
 [(53.1094053784076, 296.7045980266), (90.30940537840759, 296.7045980266)],
 [(53.1094053784076, 265.7045980266), (90.30940537840759, 265.7045980266)],
 [(53.10940537840759, 234.7045980266), (90.30940537840759, 234.7045980266)],
 [(53.10940537840759, 197.5045980266), (90.30940537840758, 197.5045980266)],
 [(53.10940537840759, 166.5045980266), (90.30940537840758, 166.5045980266)],
 [(53.10940537840759, 151.0045980266), (90.30940537840758, 151.0045980266)]]
```

Once trimmed, we extract the coordinates of each segment in `joint_lines_modified_layer`. The list `modified_joint_vertices` will later be used to calculate the width and placement of hinge cutouts. Capturing these coordinates ensures we have precise endpoints for each hinge slot.

```
simple_joint_layer = joint_lines_modified_layer << plain_width/2
simple_joint_layer
```



We create a `simple_joint_layer` by offsetting the trimmed joint lines by half the hinge width (`plain_width/2`). This operation produces narrow rectangular regions around each joint line, representing the areas of the body that will be removed to form flexible hinges. Offsetting by half the hinge width ensures the final hinge slot has the correct total width.

Generating Hinge Holes

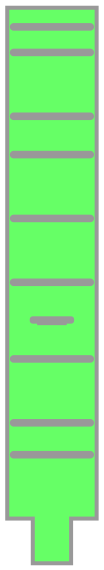
```
hole,dummy = frm.calc_hole(modified_joint_vertices,plain_width/2)
fr.my_line_width=0
holes = hole.to_laminate(NUM_LAYERS)
holes<<=.5
holes
```

```
<Figure size 640x480 with 0 Axes>
```

The call to `frm.calc_hole` computes the circular or elliptical reinforcement holes that are often added at the ends of hinges to distribute stress. Passing in the list of modified joint vertices and half the hinge width yields hole geometry sized to fit at each hinge. We then convert these holes into a multi-layer laminate (`holes.to_laminate(NUM_LAYERS)`) so that the holes propagate through all layers. Finally, we offset (`<<=.5`) the holes slightly to account for kerf and to ensure proper clearance.

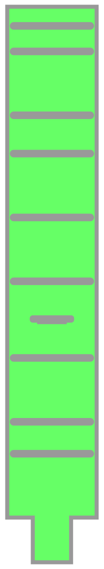
Constructing Laminate Layers

```
rigid_layer = (body_layer - simple_joint_layer)
rigid_layer
```



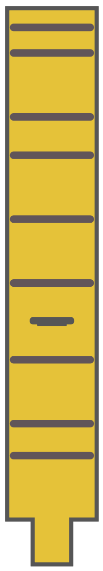
The `rigid_layer` is formed by subtracting the hinge slots (`simple_joint_layer`) from the main `body_layer`. This operation leaves only the solid portions of the leg that will remain rigid, while the removed regions correspond to flexible joints. Inspecting `rigid_layer` confirms that the joint cutouts appear in the correct locations.

```
adhesive_layer = rigid_layer & body_layer
adhesive_layer
```



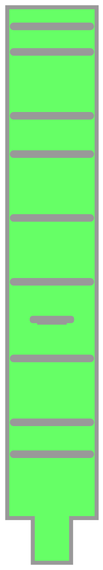
Adhesive layers sit between rigid and flexible layers to bond them together. We compute the `adhesive_layer` by taking the intersection of `rigid_layer` and `body_layer`. This produces a shape exactly matching the rigid parts, ensuring adhesive is applied only where rigid layers overlap the flexible core.

```
ideal_final_device = Laminate(rigid_layer,adhesive_layer, body_layer,
adhesive_layer,rigid_layer)
ideal_final_device
```



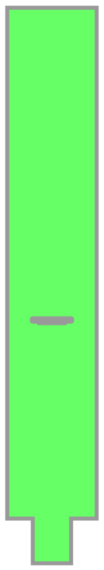
Here we assemble the ideal laminate stack. Using the `Laminate` class, we order the layers as rigid, adhesive, flexible (body), adhesive, and rigid. This five-layer structure mirrors the recommended laminate configuration for foldable mechanisms. Visualizing `ideal_final_device` provides a composite view of the design before adding supports and bridges.

```
ideal_final_device[0]
```



Accessing `ideal_final_device[0]` extracts the top rigid layer for inspection. Viewing individual layers helps verify that the laminate stack is organized correctly.

```
ideal_final_device[2]
```



Similarly, `ideal_final_device[2]` displays the middle flexible layer. Examining the flexible layer alone confirms that the hinge slots and holes have been properly applied.

Adding Support Bridges

```
bridges = frd.read_lines('car.dxf', layer='bridge')
bridges
```

```
[]
```

Many laminate designs require temporary bridges to hold scrap pieces in place during manufacturing. We read bridge lines from the `'bridge'` layer of the DXF file using `read_lines`. These coordinates indicate where narrow bridges will connect the rigid parts to scrap material.

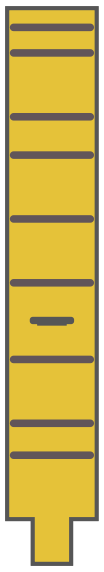
```
bridges_layer = Layer(*[sg.LineString(item) for item in bridges])
bridges_layer <<= bridge_thickness
bridges_layer
```

We convert the bridge lines into a `Layer` of `LineString` objects and then offset them by the specified `bridge_thickness`. Offsetting gives the bridges finite width so they can physically connect pieces. The resulting `bridges_layer` captures all support bridges.

```
bridges_lam =
Laminate(bridges_layer,bridges_layer,Layer(),bridges_layer,bridges_layer)
bridges_lam
```

Next we create `bridges_lam`, a laminate where bridges are present in the top, second, fourth, and fifth layers but not the flexible middle layer. Omitting bridges from the center layer prevents them from interfering with the hinge motion while still holding the outer layers together.

```
supported_actual_device = ideal_final_device | bridges_lam
supported_actual_device
```



To form the `supported_actual_device`, we perform a union (`|`) of the ideal device and the bridge laminate. This composite geometry includes both the designed mechanism and the temporary bridges needed for fabrication.

Removing Extra Material and Final Device

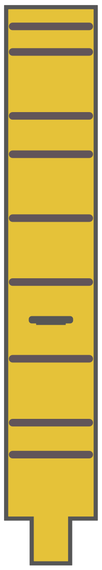
```
diff = supported_actual_device - ideal_final_device
removal = frm.cleanup(diff, .1)
removal
```

We calculate the difference between the supported and ideal devices to isolate only the bridge structures. Passing this difference to `frm.cleanup` with a small threshold removes tiny slivers of geometry that could cause issues during manufacturing. The resulting `removal` layer represents the bridge material that must be removed after the first pass of cutting.

```
removal = frm.keepout_laser(removal)
removal
```

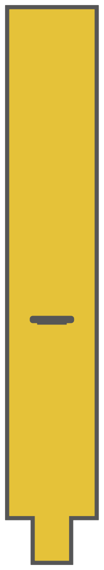
Applying `frm.keepout_laser` to `removal` expands the removal regions to account for the laser kerf and ensures that no adhesive is cut unintentionally. This step generates a keep-out area around each removal region.

```
actual_final_device = ideal_final_device- holes - removal
actual_final_device
```



We compute `actual_final_device` by subtracting both the hinge reinforcement holes and the removal regions from the ideal device. The resulting laminate geometry represents the final shape of the leg mechanism after all temporary supports have been removed. This is the geometry that will function as the leg.

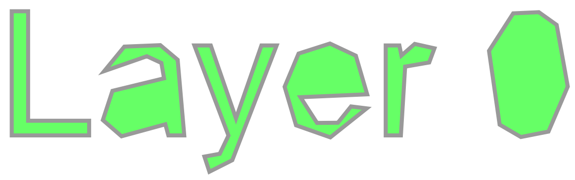
```
keepout = frm.keepout_laser(actual_final_device)
keepout
```



The keep-out region of the final device is calculated with `frm.keepout_laser(actual_final_device)`. Keep-out zones define areas where the laser should not cut during the second pass to prevent damaging the laminate or adhesives.

Adding Layer Labels and Alignment Holes

```
layer_id = frm.build_layer_numbers(NUM_LAYERS,
text_size=jig_dia)
layer_id = layer_id.simplify(.2)
layer_id[0]
```



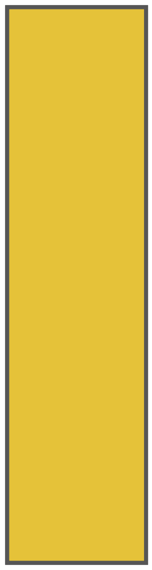
To help identify layers during assembly, we generate small numerical labels. The `build_layer_numbers` function creates tiny text shapes for each layer using the specified `text_size`. Simplifying these shapes (`simplify(.2)`) reduces their complexity. Displaying `layer_id[0]` shows the label on the first layer.

```
(x1,y1),(x2,y2) = actual_final_device.bounding_box_coords()
w1,h1 = actual_final_device.get_dimensions()
w2 = round(w1/jig_spacing)*jig_spacing+jig_spacing+support_width
h2 = round(h1/jig_spacing)*jig_spacing+jig_spacing+support_width
x1 -= (w2-w1)/2
y1 -= (h2-h1)/2
x2 += (w2-w1)/2
y2 += (h2-h1)/2
points = []
points.append(sg.Point(x1,y1))
points.append(sg.Point(x2,y1))
points.append(sg.Point(x1,y2))
points.append(sg.Point(x2,y2))
alignment_holes_layer = Layer(*points)
alignment_holes_layer<<=(jig_dia/2)
alignment_holes=alignment_holes_layer.to_laminate(NUM_LAYERS)
alignment_holes
```

Alignment holes ensure that the five laminate layers line up correctly during assembly. We compute the bounding box of the `actual_final_device`, round its dimensions to the nearest multiple of `jig_spacing`, and calculate corner positions (`points`). Converting these points into small circular holes (`<<=(jig_dia/2)`) and then into a laminate replicates the holes across all layers. These alignment holes will later be cut through the sheet and used with mechanical jigs during assembly.

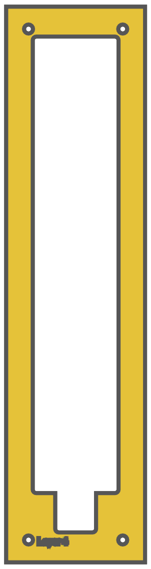
Defining Sheet and Scrap Sections

```
sheet_layer = (alignment_holes_layer<<bounding_box_padding).bounding_box()
sheet=sheet_layer.to_laminate(NUM_LAYERS)
sheet
```



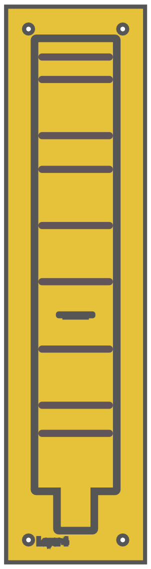
We create a `sheet_layer` by expanding the alignment holes layer by `bounding_box_padding` and taking its bounding box. The resulting sheet defines the rectangular material sheet on which all parts will be cut. Converting this to a laminate (`sheet`) makes it easy to combine with other laminate objects.

```
removable_scrap = frm.calculate_removable_scrap(
actual_final_device,sheet,support_width,is_adhesive)
web = removable_scrap-alignment_holes-layer_id.translate(x1+jig_dia,y1-jig_dia/2)
web
```



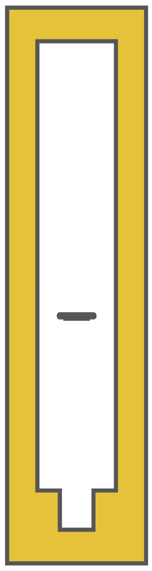
Scrap that will be removed during cutting must sometimes be supported by a web of material. The function `frm.calculate_removable_scrap` computes this removable material given the actual device, the sheet, support width, and adhesive pattern. We then subtract the alignment holes and translated layer labels from the scrap to form `web`, ensuring those features remain unaffected. The `web` acts as a lattice that holds individual parts together until they are manually separated.

```
(web|actual_final_device)
```



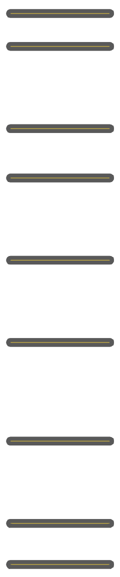
Taking the union of the scrap web and the actual device visualizes how the final mechanism is embedded in the supporting material. This combined view helps confirm that the support web adequately surrounds the device.

```
second_pass_scrap = sheet-keepout
second_pass_scrap
```



The `second_pass_scrap` represents all the material outside the keep-out region that will be removed in the second cutting pass. It is computed by subtracting the keep-out layer from the sheet.

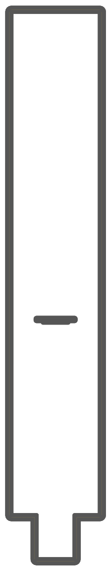
```
first_pass_scrap = sheet - second_pass_scrap - actual_final_device
first_pass_scrap = frm.cleanup(first_pass_scrap,.00001)
first_pass_scrap
```



First-pass scrap consists of the sheet material minus both the second-pass scrap and the actual device. Cleaning up this shape using `frm.cleanup` removes tiny artifacts, leaving only meaningful scrap that should be removed in the first pass. Separating cutting into two passes allows delicate hinges and bridges to be cut while maintaining overall structural integrity.

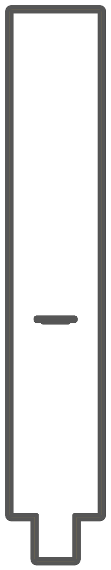
Supports and Combining Design

```
support =
frm.support(actual_final_device,frm.keepout_laser,support_width,support_width/2)
support
```



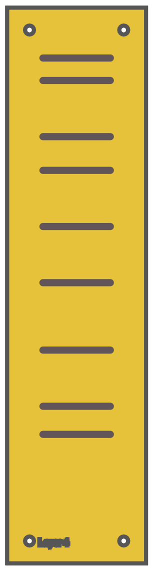
Support structures are generated to hold the device in place during both cutting passes. The `frm.support` function creates lines of support at a specified width (`support_width`) and spacing (`support_width/2`) that avoid interfering with the device geometry. These supports distribute stresses and ensure the device does not fall out of the sheet prematurely.

```
support | bridges_lam
```



Overlaying the support structures with the bridge laminate shows how both features work together. Visualizing `support | bridges_lam` allows us to inspect whether supports and bridges overlap as intended.

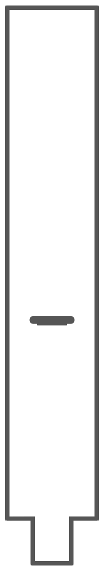
```
supported_design = web|actual_final_device|support| bridges_lam
supported_design
```



The `supported_design` is the union of the scrap web, the actual device, the support structures, and the bridges. This composite laminate is what will be sent to the laser cutter for the first pass. It contains all features necessary to build the device while maintaining connectivity and support.

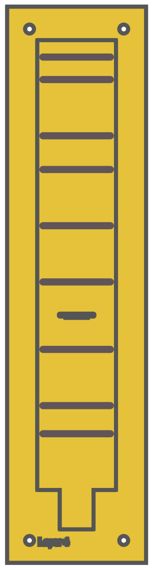
Kerf Compensation and Remaining Parts

```
cut_material = (keepout<<kerf)-keepout
cut_material
```



Kerf compensation is applied to the keep-out region to determine the actual material that will be removed. Offsetting the keep-out layer by the kerf amount (`keepout << kerf`) and subtracting the original keep-out yields `cut_material` , which represents the laser's path width. Incorporating kerf compensation ensures the final parts have accurate dimensions.

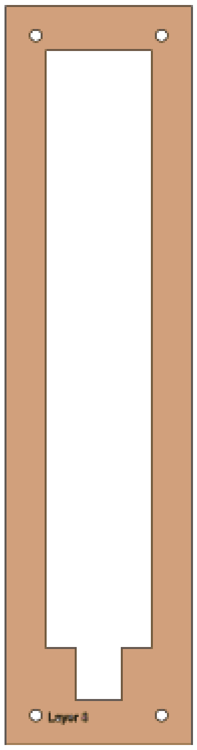
```
remaining_material = supported_design-cut_material
remaining_material
```

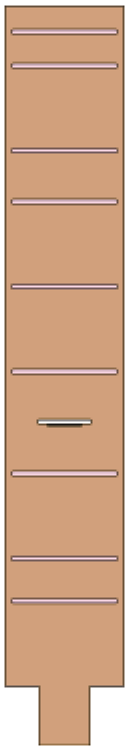


The `remaining_material` is calculated by subtracting the cut material from the supported design. This leaves only the pieces of material that will remain attached after cutting, including both the device and any leftover scrap.

```
remaining_parts = frm.find_connected(remaining_material,is_adhesive)
for item in remaining_parts:
    item.plot(new=True)
```

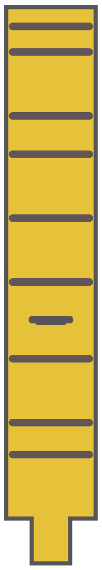
```
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
```





To identify separate scraps, we call `frm.find_connected` on the remaining material. This function groups connected pieces based on whether they are adhesive or rigid. Plotting each `item` in `remaining_parts` reveals the different regions that persist after cutting and helps us locate the piece corresponding to the device.

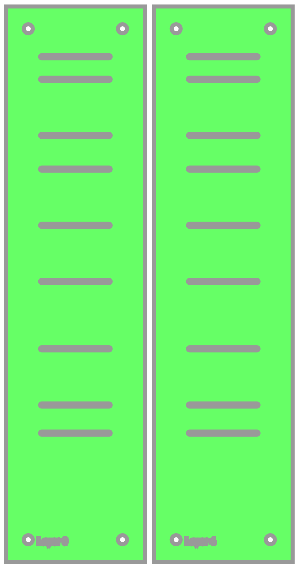
```
test_part=actual_final_device>>1
for result in remaining_parts:
    if not (result&test_part).is_null():
        break
result
```



We shift (`>>`) the actual device by one layer to aid in selecting the correct piece from `remaining_parts`. By checking which remaining piece intersects the shifted device, we isolate the geometry corresponding to the device. This step ensures that we select the correct connected component for further processing.

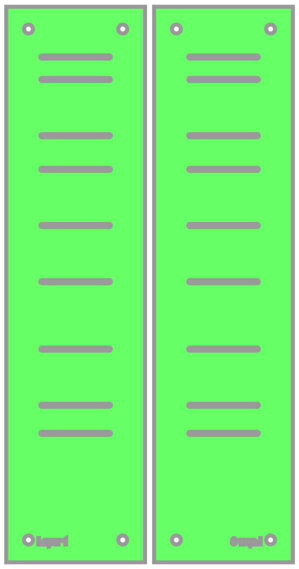
Preparing Layout for First Pass

```
w,h = supported_design.get_dimensions()
p0,p1 = supported_design.bounding_box_coords()
rigid_layer = supported_design[0] | (supported_design[-1].translate(w+5,0))
rigid_layer
```



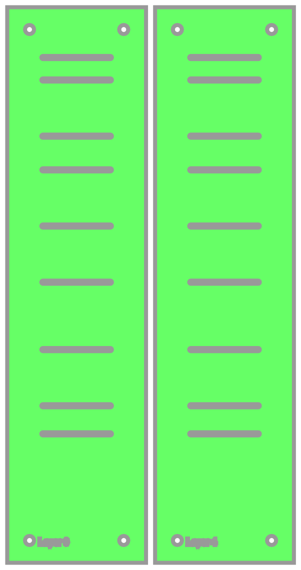
The next few cells prepare the layout for the first cutting pass by positioning the rigid layers side-by-side. We extract the dimensions (`w`, `h`) and bounding box (`p0`, `p1`) of the supported design. We then create a new `rigid_layer` as the union of the first and last layers, translating one copy horizontally (`translate(w+5, 0)`) to separate them. This arrangement makes it easier to cut and assemble the laminate.

```
l4 = supported_design[3].scale(-1,1)
p2,p3 = l4.bounding_box_coords()
l4 = l4.translate(p0[0]-p2[0]+w+5,p0[1]-p2[1])
adhesive_layer = supported_design[1] | l4
adhesive_layer
```



In similar fashion, we arrange the adhesive layers. We take the fourth layer of `supported_design`, mirror it horizontally (`scale(-1,1)`), compute its bounding box (`p2`, `p3`), and translate it to align next to the existing adhesive layer. Uniting this mirrored adhesive layer with the second layer yields the final adhesive layer arrangement for cutting.

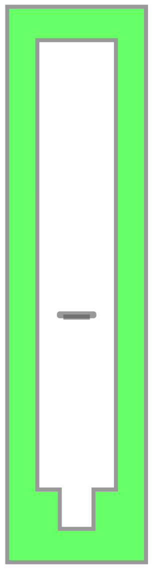
```
first_pass = Laminate(rigid_layer,adhesive_layer,supported_design[2])
first_pass.export_dxf('first_pass2')
first_pass[0]
```



The `first_pass` laminate is constructed by stacking the rearranged rigid and adhesive layers with the flexible layer from `supported_design`. We then export this laminate to a DXF file named `'first_pass2'` using `export_dxf`. Displaying `first_pass[0]` lets us examine the top layer of the first-pass layout and confirm that all parts are correctly positioned.

Final Cut Path and Scaling

```
final_cut = sheet - keepout
final_cut = final_cut[0]
final_cut.export_dxf('final_cut')
final_cut
```



After planning the first pass, we compute the `final_cut` path needed to free the device from the sheet in the second pass. We subtract the keep-out region from the sheet and take only the first layer of the result for cutting. Exporting this shape as `'final_cut'` generates a DXF containing the outer contour that will separate the finished device from the surrounding material.

```
final_cut_scaled=first_pass[0].scale(1/25.4,1/25.4)
(x,y),(_,_) = final_cut_scaled.bounding_box_coords()
final_cut_scaled = final_cut_scaled.translate(-x,-y)
final_cut_scaled.bounding_box_coords()
```

```
[(np.float64(0.0), np.float64(0.0)),
 (np.float64(6.229583118745245), np.float64(12.07148447275845))]
```

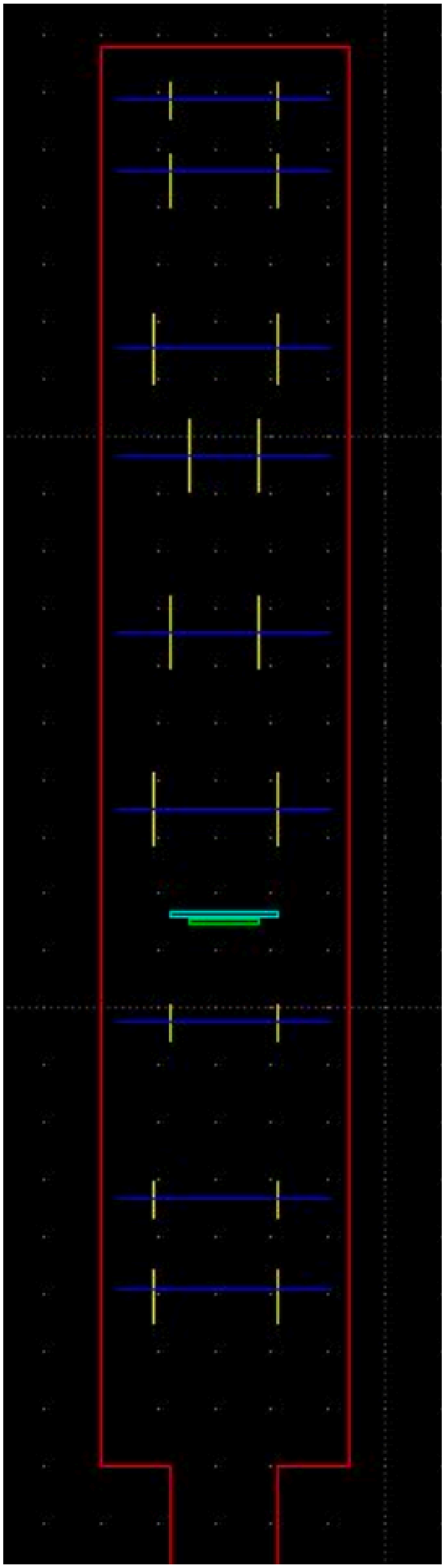
To prepare the cut path for fabrication, we scale it from millimeters to inches (1 inch ≈ 25.4 mm) by dividing by 25.4. We then translate the scaled geometry to the origin by subtracting its minimum coordinates (`x` , `y`) from all points. These transformations position the design correctly for the manufacturing equipment.

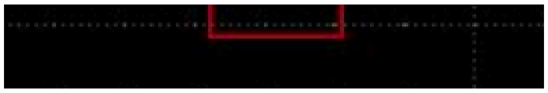
```
final_cut_scaled.exteriors()
```

```
[[(3.016366362522229, 0.0),
  (0.0, 0.0),
  (0.0, 12.07148447275845),
  (3.016366362522229, 12.07148447275845),
  (3.016366362522229, 0.0)],
 [(6.229583118745245, 0.0),
  (3.213216756223016, 0.0),
  (3.213216756223016, 12.07148447275845),
  (6.229583118745245, 12.07148447275845),
  (6.229583118745245, 0.0)]]
```

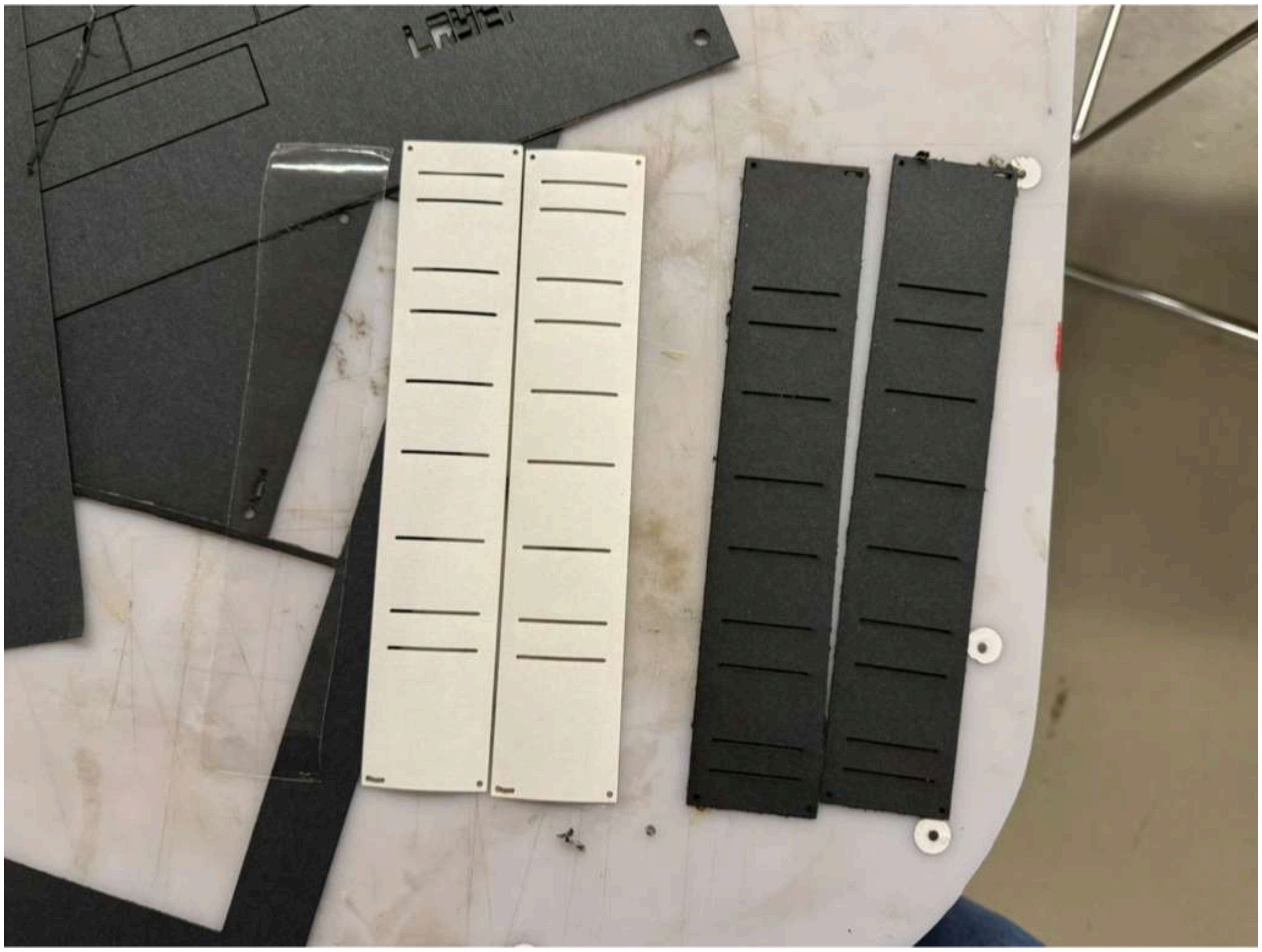
Finally, retrieving `final_cut_scaled.exteriors()` gives us the exterior boundaries of the scaled final cut. These boundary coordinates can be used directly by CAD/CAM software to drive the laser cutter during the second pass.

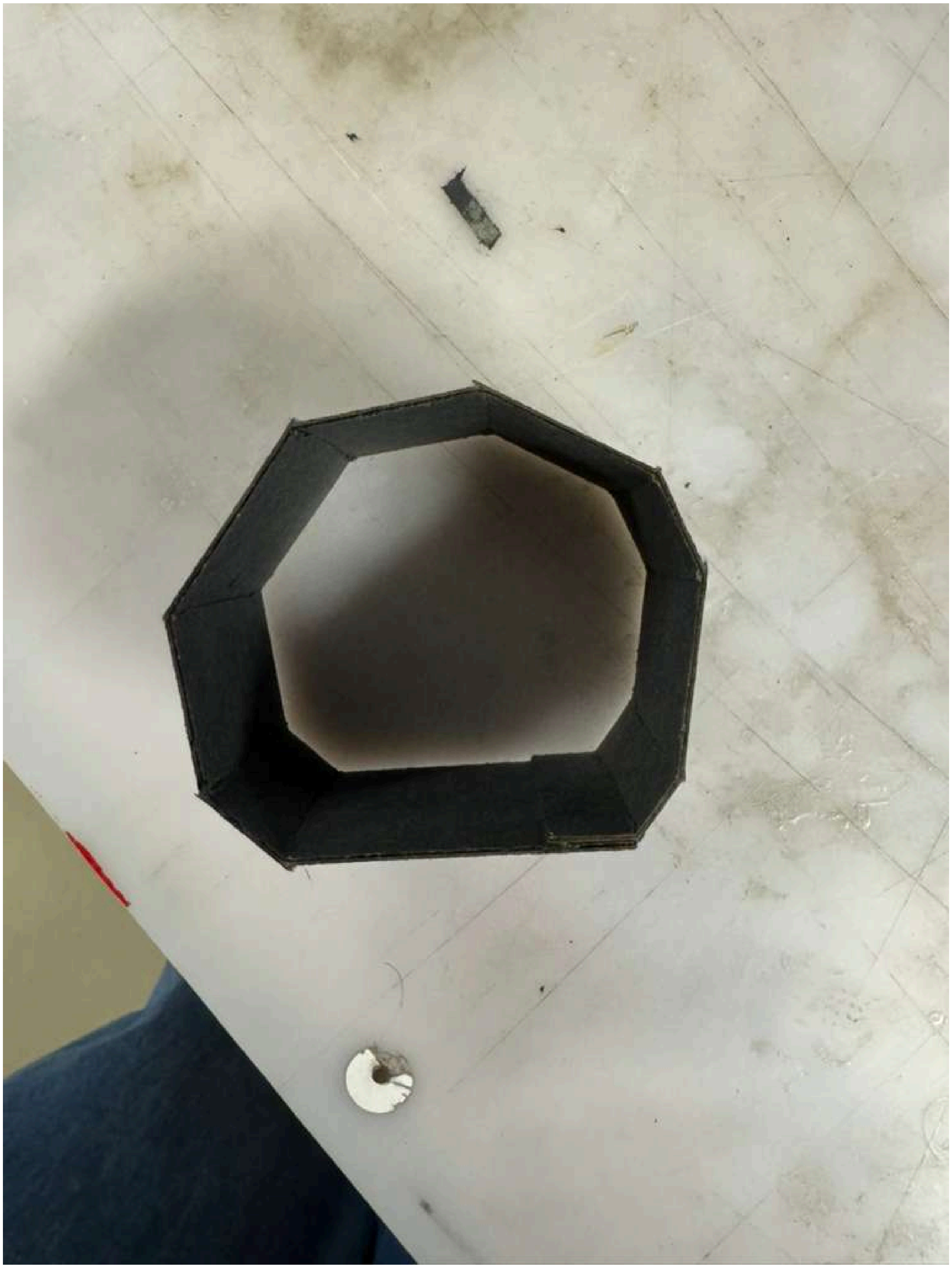
LibreCAD Model





Cutout material and the parallel mechanism





Design Evaluation and Reflection

Did the Design Perform as Expected?

The constructed prototype exhibited a high degree of fidelity to the intended design specifications. The hinge bands, designed with an approximate width of **1.73 mm** to facilitate a **120°** rotation at a **1 mm** laminate thickness, operated as anticipated, enabling a seamless and repeatable motion without any binding issues. Furthermore, the five-layer laminate configuration successfully preserved the continuity of the flexible core across the joints. The incorporation of alignment holes and support tabs proved effective in achieving precise registration during both the cutting and assembly processes. A minor deviation noted was the excessive adhesive coverage in the hinge zone during the initial stages of fabrication; this issue will be addressed in the comprehensive build by implementing masking techniques for the adhesive layers.

---

(1) Scaling Values

In order to ensure dimensional accuracy in the modeling of geometric features, it is crucial to adopt a metric unit system; specifically, measurements should be conducted in millimeters, as outlined in the referenced workflow materials. This approach can be validated through the utilization of a 10 mm calibration square, which serves as a reliable means of confirming the precision of dimensions post-DXF import.

The hinge-width calculation is intricately linked to both the laminate thickness and the intended range of motion. For instance, a laminate with a total thickness of 1 mm and a desired fold angle of 120° necessitates a hinge width of 1.73 mm. It is imperative to proportionally increase this width for laminates of greater thickness or for broader fold angles, adhering to the relationship defined by the equation (w = t \tan(\theta/2)).

Additionally, the calibration of kerf compensation must be approached experimentally; the nominal kerf value of 0.05 mm should be fine-tuned according to the specific characteristics of the cutting tool employed. Inadequate kerf values that are undersized may result in the undesirable fusion of parts.

During the process of downscaling or miniaturizing a design, it is essential to maintain the stability of the parts by ensuring that the widths of bridges and supports are greater than twice the kerf. This parameter is critical for sustaining structural integrity throughout the usage and application of the design.

---

(2) Kinematic Adjustments

To improve the biomimetic performance of robotic legs, several design parameters can be optimized.

Firstly, the **segment lengths** of the leg can be strategically adjusted. Extending the distal link is reminiscent of the energy-efficient locomotion observed in ostriches, as it enhances energy return during movement. Conversely, employing a lighter mid-link, inspired by the jerboa, can facilitate rapid transitions between stance and swing phases, allowing for more agile movements.

Secondly, the **hinge stiffness** is another critical parameter that can be fine-tuned. This can be achieved by altering hinge length or the thickness of the flex layers. Longer or thinner hinges tend to create a more compliant motion, enabling smoother movement, while shorter or thicker hinges contribute to increased stiffness and enhanced load-bearing capabilities.

Finally, the introduction of **passive coupling** mechanisms—such as employing springs or linkages between joints—promotes synchronized motion among adjacent segments. This design approach mimics the energy transfer mechanisms of natural runners, akin to the tendon-like interactions that facilitate efficient movement in biological organisms.

---

(3) Geometric Integration

In the context of integrating a proximal link into a larger robotic system, several design considerations are paramount to ensure functionality and compatibility. The incorporation of standardized mounting holes, such as those with M2–M3 spacing, is essential for facilitating secure attachment and interoperability with various components. Additionally, the inclusion of flat reference surfaces promotes accurate fixture alignment, thereby enhancing the overall assembly process.

Furthermore, the design can be optimized for tendon-driven actuation by integrating cable-routing channels or small eyelets near the joints. This feature will not only streamline the actuation mechanism but also contribute to a more organized and efficient system layout.

To enhance usability, it is advisable to include mirror symmetry options for both left and right limbs. Clear alignment markers should be implemented to assist assemblers in achieving precise configurations. Such design considerations will significantly contribute to the robustness and adaptability of the robotic system in diverse applications.

---

(4) Manufacturing Considerations

Adhesive masking plays a critical role in the structural integrity of hinge regions within composite materials. It is essential to remove adhesive layers from these areas to mitigate the risks of stiffness and fatigue failure. This can be effectively accomplished by customizing adhesive sheets to conform to the outlines of the rigid layers.

Furthermore, implementing stress relief features, such as rounded end-reliefs (small circular cutouts) at the ends of hinges, is necessary for uniform stress distribution. This design element not only enhances the performance of the hinge but also contributes to an increased cycle life.

When it comes to support optimization, employing fewer and thicker bridging elements is advisable to enhance stability. It is also crucial to ensure that the width of these supports exceeds the kerf by an appropriate safety margin, which prevents structural integrity issues during the cutting process.

In terms of cut sequencing, adopting a two-pass fabrication strategy is recommended. This involves performing internal features and hinge bands in the initial pass, followed by perimeter release in the subsequent pass. Such a methodology ensures the preservation of registration and the integrity of the components throughout the fabrication process.

Lastly, scalability considerations should be taken into account for prototyping purposes. A single-layer flexible hinge variant can be produced for rapid prototyping, whereas the full five-layer laminate structure remains superior for applications requiring enhanced durability and biomechanical fidelity.

Summary

The current prototype successfully achieves the main design objectives: predictable hinge motion, accurate registration, and manufacturability within the limits of the laminate process. Future iterations will focus on refining hinge scaling, enhancing joint coupling for dynamic gait reproduction, and improving mounting geometry for better system integration. Together, these modifications will enable more efficient energy storage and agile multi-gait motion, aligning more closely with the agility of the jerboa and the tendon-mediated efficiency of the ostrich as described in the biomechanical literature.

Conclusion

This notebook comprehensively outlines the complete workflow for foldable robotics as detailed in the accompanying report. It begins with the formulation of bio-inspired design objectives, drawing insights from the agility of the jerboa and the efficient locomotion of the ostrich. Using the methodologies discussed in Chapters 52–54, a CAD model is transformed into a multi-layer laminate structure.

The workflow involves the importation of DXF geometry, computation of hinge parameters, definition of support structures, and preparation of cutting layouts for both first and second passes. By incorporating elements such as holes, hinges, bridges, alignment features, and kerf compensation, a manufacturable design for a foldable leg mechanism is achieved. The notebook is meticulously organized into sections, with detailed explanations that elucidate the rationale behind each step. This structure effectively bridges the connection between the code, the foundational biomechanical inspiration, and the fabrication requirements.