Supplemental Material
AutoConnect: Computational Design of 3D-Printable Connectors

Yuki Koyama[1,2] Shinjiro Sueda[1,3] Emma Steinhardt[1] Takeo Igarashi[2] Ariel Shamir[1,4] Wojciech Matusik[5]

[1]Disney Research Boston [2]The University of Tokyo [3]Cal Poly [4]IDC Herzliya [5]MIT

Abstract

This supplemental material provides additional information regarding the mechanical holders and free-from holders. We include all the necessary information to create the mechanical holder database for AutoConnect: the details of the measured grip strength data for the six mechanical holders we used and working pseudocode. This database can be easily extended by dedicated end-user communities or companies by adding other holder definitions along with measured force data.

1 Mechanical Holder Details

Each mechanical holder is designed for a specific type of *standard shapes* and is parameterized with a few control parameters. In our current implementation, we consider four types of standard shapes as shown in Fig. 1. Some of standard shapes include parameters representing their geometric structure such as the *diameter* in a cylinder. As demonstrated in the paper, these four types of shapes cover a large set of possible shapes and applications.

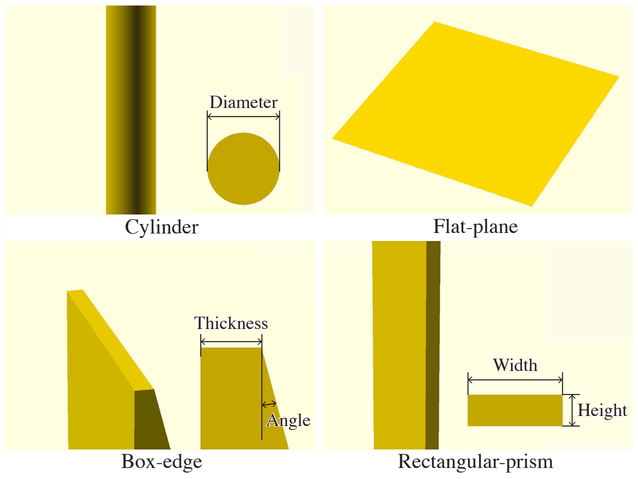


Figure 1: *The definitions of the standard shapes.*

The control parameters of the mechanical holders are used to balance the functionality (*e.g.,* how tightly they attach to the target object) and the volume (*i.e.,* material consumptions). In AutoConnect, the values of the control parameters are computed automatically by solving an optimization problem. Table 1 shows the list of our mechanical holders.

Table 1: *Mechanical holders in our database.*

Holder name	Standard shape	#Parameters
Snap Pipe Clamp	Cylinder	4
Strap Pipe Clamp	Cylinder	1
Cam Pipe Clamp	Cylinder	3
Suction Cup	Flat-plane	3
Toggle Clamp	Box-edge	3
Box Clamp	Rectangular-prism	4

All mechanical holders are designed using *Constructive Solid Geometry* (CSG), resulting in a CSG-tree representation for each holder. Other parametric representations can be used in our implementation as well. A CSG-tree can be converted into a printable, watertight surface mesh by using existing libraries such as CGAL [CGAL]. We include working pseudocode in OpenSCAD grammar [OPENSCAD] of the CSG-trees of all holders in Listing 3 to Listing 8 the end of this document. Listing 1 shows some common utility functions and modules, and Listing 2 is code to add *fillets* to corners, which is a standard technique to strengthen the structure by avoiding stress concentration (Fig. 2).

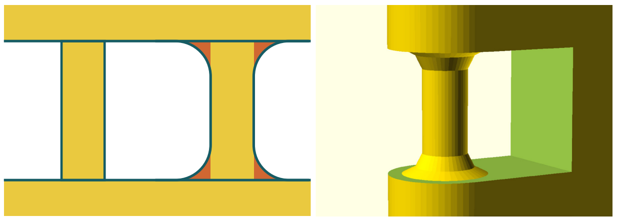


Figure 2: *Left: comparison of a model with and without fillets (the orange parts). Right: example of using fillet in a 3D model.*

Table 2: *Mechanical Holder Parametrization.*

Holder type	Parameter	Expected range
Snap Pipe Clamp	Target diameter [mm]	$[20, 40]$
	Closeness	$[0, 1]$
	Thickness	$[0, 1]$
	Width	$[0, 1]$
Strap Pipe Clamp	Target diameter [mm]	$[20, 40]$
Cam Pipe Clamp	Target diameter [mm]	$[20, 40]$
	Thickness	$[0, 1]$
	Width	$[0, 1]$
Suction Cup	Height	$[0, 1]$
	Radius	$[0, 1]$
	Scale	$[0, 1]$
Toggle Clamp	Target thickness [mm]	$[20, 40]$
	Target angle [deg]	$[0, 30]$
	Contact area and thickness	$[0, 1]$
Box Clamp	Target rectangle width [mm]	$[50, 70]$
	Target rectangle height [mm]	$[15, 25]$
	Thickness	$[0, 1]$
	Width	$[0, 1]$

Details of Each Holder Type In Table 2, we describe the parameters defined for each mechanical holder. Some parameters such as "target diameter" are determined by the geometry to be held, and the other parameters are automatically optimized by AutoConnect. We show our mechanical holder appearances with various parameter settings in Fig. 4. We provide the working pseudocode of all of the holders at the end of this documents.

Snap Pipe Clamp The snap pipe clamp snaps to cylindrical pipes by frictional force. As this frictional force is generated by its elasticity, parameters that affect the elasticity also affects its grip strength.

Strap Pipe Clamp The strap pipe clamp can be attached to cylindrical pipes by utilizing a (not-printed) strap. The printed part has a hole through which a strap can pass. After several measurements we found that this holder can be considered infinitely strong in practice. Thus, we decided to omit the degrees of freedom in its design (such as thickness) other than the target cylinder diameter.

Cam Pipe Clamp The cam pipe clamp has three components: the base part, the lever part, and the link part, which define the open/close states (see Fig. 3). These parts can be printed as an assembly by using inkjet-style 3D printers, and there is no need to manually assemble them. In a closed state, this holder generates pressure force on the pipe because the diameter becomes slightly smaller than the diameter of the target pipe.

Suction Cup The suction cup sticks to a smooth flat surface such as a window glass. We designed this suction cup inspired by [Thanh-Vinh et al. 2011]. After several experiments, we found that a commercial rubber-like material (*TangoBlack+* from Stratasys, Ltd.) with the *glossy finish* work well to achieve high grip strength.

Toggle Clamp The toggle clamp has a linkage mechanism in its lever part, which provides a strong pressure force when it is closed. In contacting areas, rubber-like materials (displayed as black parts) are placed so that it generates strong friction when attached.

Box Clamp The box clamp clamps rectangular-prisms simply by frictions.

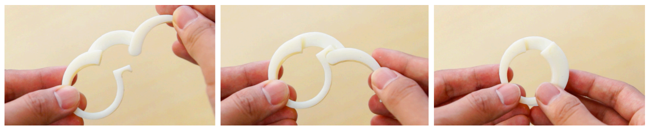


Figure 3: *The mechanism of the cam pipe clamp.*

2 Measured Force Data

2.1 Number of Data Points

For each mechanical holder we sample the parametric space, produce and print actual holders and measure their grip strength. For a mechanical holder that has n design parameters, we consider $1+2n+2^n$ samples, or *parameter sets*, consisting of 1 default parameter set (*e.g.,* $(0.5, 0.5, \ldots, 0.5)$), $2n$ settings where only one parameter has minimium or maximum values, while the other have default values (*e.g.,* $(1.0, 0.5, \ldots, 0.5)$), and 2^n settings where all parameters have either minimimum or maximum values (*e.g.,* $(1.0, 0.0, \ldots, 0.0)$). For instance, for the snap pipe clamp that has 4 parameters, we print $1 + 2 \cdot 4 + 2^4 = 25$ clamps and measure each clamp's grip strength.

The strap pipe clamp is an exception. This clamp has almost infinitely strong grip strength in a realistic scenario (we applied over 400N force several times, but it never moved). Thus, we do not gather any data for this type of clamp. Note that the strap pipe clamp is parameterized by only one *geometrically-deterministic* parameter (the diameter of the target pipe) and thus there is no need to optimize the design according to the grip strength.

2.2 Measurement Settings

For the cylindrical holders we use two types of pipes: aluminum polished pipes and printed plastic pipes. For each type we use three diameters: $20.0\ \text{mm}$, $30.0\ \text{mm}$, and $40.0\ \text{mm}$ (see Fig. 5). For the flat-plane holder (*i.e.,* suction cup), we use a glass plate. For the rectangular-prism holder (*i.e.,* box clamp) and the box-edge holder (*i.e.,* toggle clamp), we use printed prisms and printed boxes.

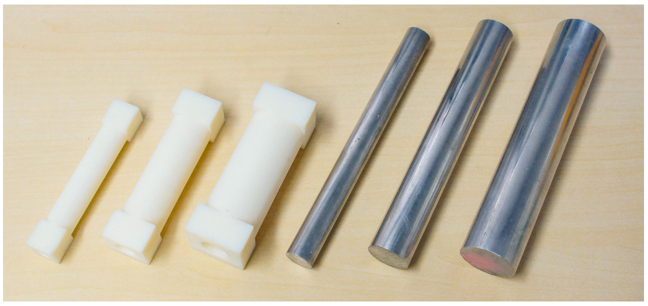


Figure 5: *Plastic and metal pipes used in our measurement.*

Both holders and the standard shapes used for measurements are printed with a material called *Endur*, which is Simulated Polypropylene, except for the suction cups and contact areas in toggle clamps (see Table 3). For those two, we use the meterial called *TangoBlack+*, which is Rubber-like material. All holders are printed using a commercial inkjet 3D printer, *Objet Connex 500*.

Table 3: *Printing settings for the measurement.*

Holder name	Material	Finish
Snap Pipe Clamp	Endur	Matt
Strap Pipe Clamp	Endur	Matt
Cam Pipe Clamp	Endur	Matt
Suction Cup	TangoBlack+	Glossy
Toggle Clamp	Endur / TangoBlack+	Matt
Box Clamp	Endur	Matt

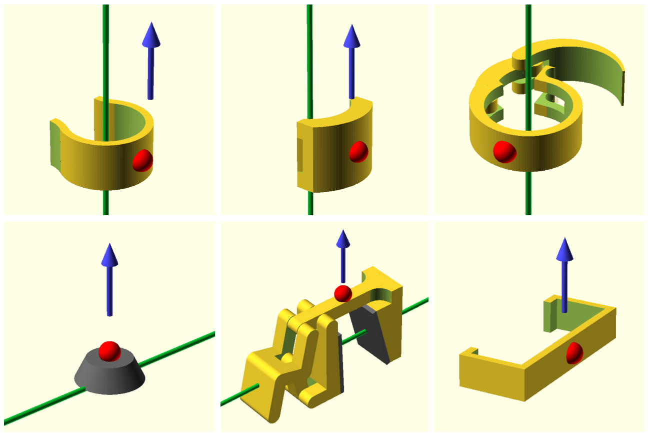
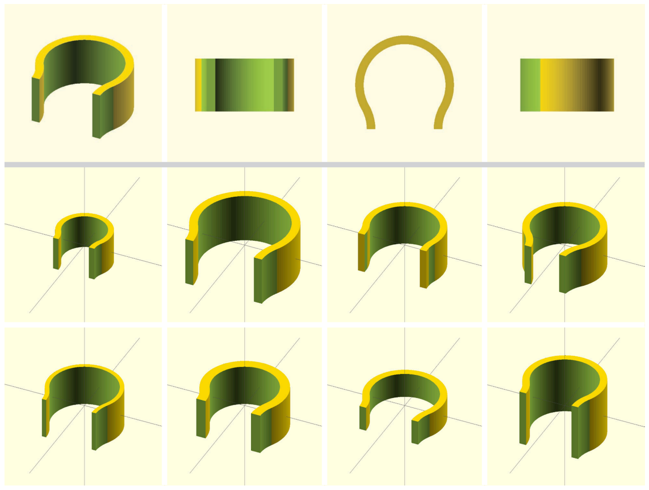
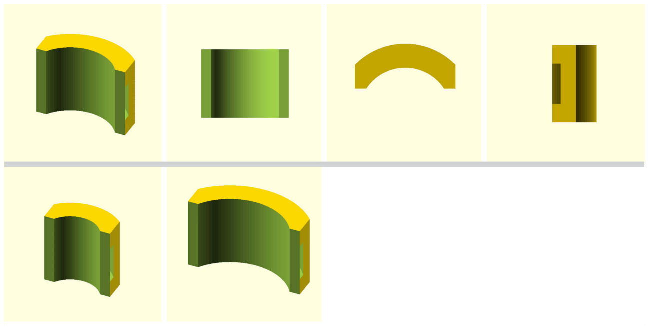
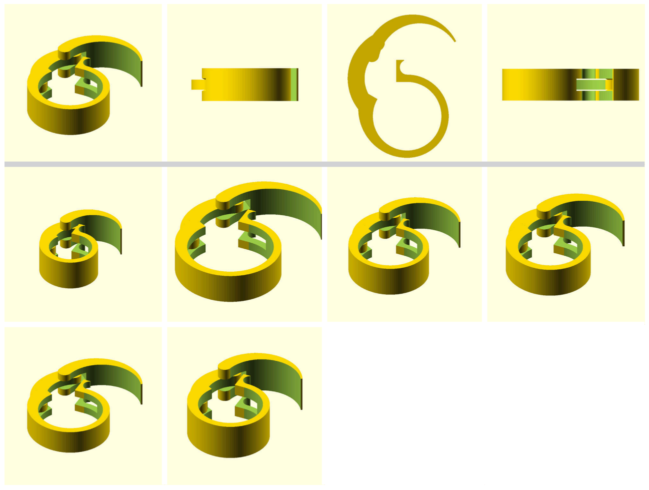


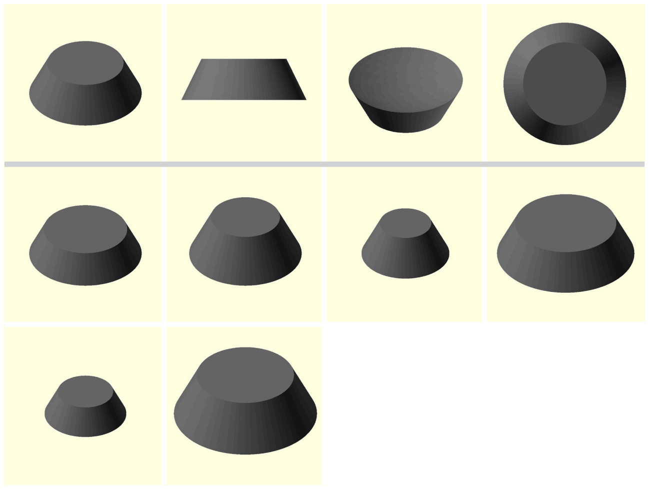
Figure 6: *Pull directions and the centers of rotation in our measurement. The red points indicate the position where we applied pulling force. The blue directions are used for measuring the translational force. The green lines indicate the observed center axes of rotation.*

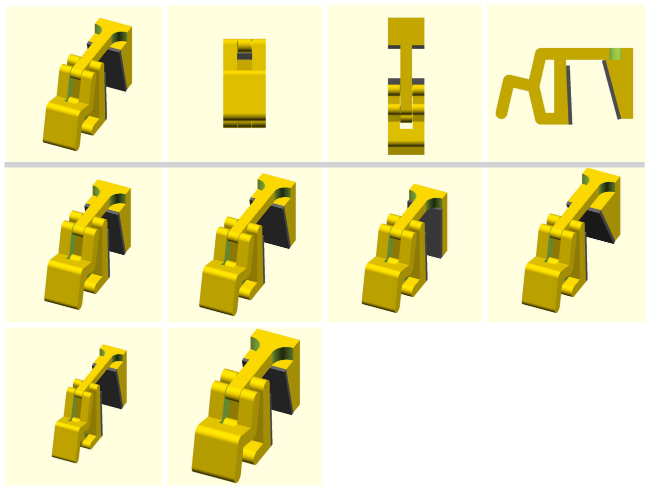
To measure the forces, we use a force meter whose resolution is 0.050 kg-weight. When printing the holders for measurement, we add a small hook at a specific point (the red points in Fig. 6), to


(a) Snap pipe clamp


(b) Strap pipe clamp


(c) Cam pipe clamp


(d) Suction cup


(e) Toggle clamp

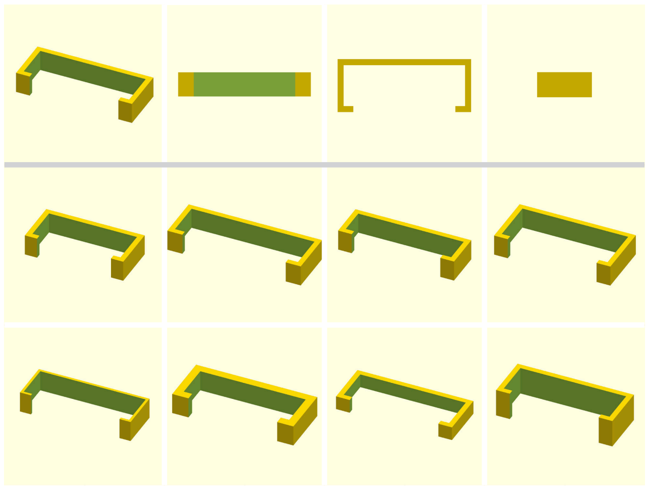

(f) Box clamp

Figure 4: *Shapes of mechanical holders with various parameters. The top row of each holder shows the shape with the default parameter setting. For the other rows, we show the holders with one-parameter-increased/decreased settings.*

attach the force meter. Starting with a small force, we pull the holder and increase the force gradually. When the mechanical holder moves, we record the maximum force displayed by the force meter before the movement. We repeat this procedure at least 3 times for each printed holder[1], and then average the three measurements. It typically takes around 2 hours to obtain the complete data set for all data points of a holder type.

2.3 Data Tables

After pulling each holder several times, we can observe the weakest translational movement direction and/or the weakest rotational movement direction for each holder type. In Fig. 6, the blue arrows indicate the translational movement directions and thus are used for measuring forces, while the green lines are the centers of the rotational movements, and thus used for measuring torques. Note that the box clamp did not move rotationally, and therefore we did not measure any torques for this holder. Also, for the cam pipe clamp, the forces to translationally move the holder were considerably stronger than the forces to rotationally move it. Thus, we did not collect data for the cam clamp. Tables 4–8 contain all the data for all holders obtained by our pulling measurements in these directions. We use this data to estimate the grip strength for all parameter settings using interpolation (as described in the paper).

2.4 Validation

We performed a simple validation for the snap pipe clamp that contains 4 parameters. We printed an additional set of snap pipe clamps that we call *testing set*, and measured the forces on them using the same procedure as for the sample set. The size of the original sampling data set is 25, and we used $N = 11$ clamps for testing. The parameters values for the testing set were random, while the diameter parameter is rounded to 20.0, 30.0, or 40.0. Let $\mathbf{x}_1, \ldots, \mathbf{x}_N$ be the testing parameters, $G(\mathbf{x}_1), \ldots, G(\mathbf{x}_N)$ the force/torque estimated from the data table, and $G_1, \ldots, G_N$ the actual measured force/torque. The root mean square (RMS) error of our estimation is

$$\mathrm{RMS}^{\mathrm{force}}_{\mathrm{metal}} = \sqrt{\frac{\sum_{i=1}^{N}\left(G(\mathbf{x}_i) - G_i\right)^2}{N}} = 2.28\ [\mathrm{N}] \tag{1}$$

for the forces with metal pipes. In the same way, we obtain

$$\mathrm{RMS}^{\mathrm{torque}}_{\mathrm{metal}} = 0.0395\ [\mathrm{N}\cdot\mathrm{m}], \tag{2}$$
$$\mathrm{RMS}^{\mathrm{force}}_{\mathrm{plastic}} = 1.65\ [\mathrm{N}], \tag{3}$$
$$\mathrm{RMS}^{\mathrm{torque}}_{\mathrm{plastic}} = 0.0373\ [\mathrm{N}\cdot\mathrm{m}]. \tag{4}$$

As this estimation is not very accurate, we use a large safety factor, which is typically a rough value (*e.g.,* $s = 5$ or $s = 10$) and found this to be sufficient for our settings, assuming the objects connected are not exceedingly heavy. Note that when necessary, better accuracy can be obtained by adding data sample points to the tables.

2.5 Friction Consideration

Many pipes in our daily lives are made of metal (*e.g.,* desk legs), and therefore we measured the grip strength of the pipe holders using metal pipes as well as plastic. Fig. 7 shows the comparison between grip strength of plastic and metal pipes. As the friction of metal is smaller than that of plastic, the grip strength is smaller for metal pipes. In AutoConnect we encourage users to use the *worst-case* data set in the available data, which in this case is the measurements taken with metal pipes. In the future, it is possible to interpolate these two data sets to model different frictional surfaces for more accurate estimation of grip strength.

[1]For a small number of suction cups, the hooks were broken before completing this repetition. In this case, we could repeat the experiment only one or two times.

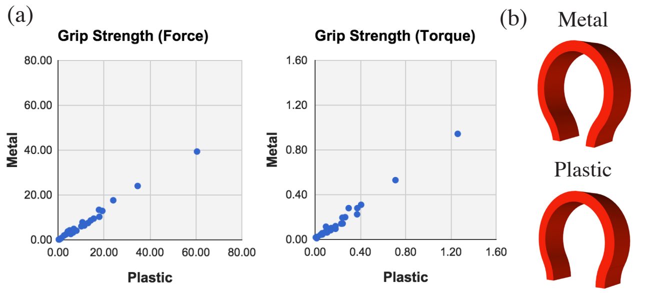


Figure 7: *(a) Comparisons of the measured data with plastic vs. metal pipes for snap pipe clamps. Metal pipes have less friction, and thus the observed grip strength is weaker. (b) Optimization with the two different data-sets: although we specify the same target grip strength, different shapes are obtained.*

3 Free-Form Holder Details

3.1 Contact Displacement Function

To define holdability, we require the 6-dimensional rigid motion (or twist)

$$\phi = \begin{pmatrix}\omega \\ \nu\end{pmatrix} \in \mathbb{R}^6, \tag{5}$$

where $\omega \in \mathbb{R}^3$ is the rotational velocity, and $\nu \in \mathbb{R}^3$ is the translational velocity expressed in body local coordinates.[2] Let p be a surface point with position $\mathbf{x}$ and normal $\mathbf{n}$. Like ϕ, both $\mathbf{x}$ and $\mathbf{n}$ are expressed in body local coordinates. Given p and ϕ, we can evaluate how much the surface point p moves along its normal when the object is moved infinitesimally by ϕ. We call this the contact blockage of p given ϕ:

$$b(p, \phi) = \max(\mathbf{n}^T\Gamma\phi, 0), \tag{6}$$

where $\Gamma = ([\mathbf{x}]^T\ I)$ is the 3×6 matrix that transforms a spatial velocity, ϕ, to a point velocity. The 3×3 skew symmetric matrix, $[\mathbf{x}]$, is the cross product matrix, such that $[\mathbf{x}]\mathbf{a} = \mathbf{x} \times \mathbf{a}$, and I is the 3×3 identity matrix. If b is positive, then p moves along its normal. Thus, this surface point p blocks the object from moving in the direction ϕ.

3.2 Finding Intrinsic Free Motions

We start with an empty set $\mathcal{F}$ of free motions, and incrementally build this set by iteratively solving the shell blockage equation (Eq. 3 in the paper) subject to the constraint set $\mathcal{C}$, which is also initially empty. Each new free motion we find adds the corresponding constraint to $\mathcal{C}$ of the form Eq. 6 in the paper, so that motion directions close to the newly found direction are no longer considered. Once we achieve holdability, the algorithm terminates, returning a set of intrinsic free motions of the input mesh. The thresholding parameter (*e.g.,* $\epsilon = 0.01$) is necessary to be robust against noise in the input mesh and inaccuracy due to discretization. For our examples,

[2]More precisely, $\phi \in se(3)$, the Lie algebra of SE(3), the special Euclidean group in 3 dimensions.

Table 4: *Snap pipe clamp.*

Surface	$x[0]$	$x[1]$	$x[2]$	$x[3]$	Force [N]	Torque [N · m]
Plastic	30.0	0.5	0.5	0.5	10.14	0.18
	20.0	0.0	0.0	0.0	0.82	0.01
	20.0	0.0	0.0	1.0	3.43	0.06
	20.0	0.0	1.0	0.0	5.07	0.10
	20.0	0.0	1.0	1.0	19.29	0.24
	20.0	1.0	0.0	0.0	1.47	0.02
	20.0	1.0	0.0	1.0	5.89	0.06
	20.0	1.0	1.0	0.0	11.45	0.14
	20.0	1.0	1.0	1.0	24.03	0.27
	40.0	0.0	0.0	0.0	0.49	0.01
	40.0	0.0	0.0	1.0	4.25	0.10
	40.0	0.0	1.0	0.0	10.63	0.30
	40.0	0.0	1.0	1.0	34.66	0.71
	40.0	1.0	0.0	0.0	0.33	0.02
	40.0	1.0	0.0	1.0	6.54	0.14
	40.0	1.0	1.0	0.0	17.99	0.41
	40.0	1.0	1.0	1.0	60.50	1.26
	30.0	0.5	0.5	0.0	5.56	0.11
	30.0	0.5	0.5	1.0	15.53	0.24
	30.0	0.5	0.0	0.5	2.62	0.05
	30.0	0.5	1.0	0.5	17.82	0.37
	30.0	0.0	0.5	0.5	8.01	0.18
	30.0	1.0	0.5	0.5	13.08	0.23
	20.0	0.5	0.5	0.5	6.87	0.09
	40.0	0.5	0.5	0.5	14.22	0.37
Metal	30.0	0.5	0.5	0.5	6.05	0.12
	20.0	0.0	0.0	0.0	0.33	0.01
	20.0	0.0	0.0	1.0	2.45	0.04
	20.0	0.0	1.0	0.0	4.25	0.06
	20.0	0.0	1.0	1.0	12.92	0.14
	20.0	1.0	0.0	0.0	0.82	0.02
	20.0	1.0	0.0	1.0	4.09	0.06
	20.0	1.0	1.0	0.0	6.38	0.08
	20.0	1.0	1.0	1.0	17.66	0.20
	40.0	0.0	0.0	0.0	0.16	0.02
	40.0	0.0	0.0	1.0	3.76	0.11
	40.0	0.0	1.0	0.0	7.85	0.28
	40.0	0.0	1.0	1.0	24.03	0.53
	40.0	1.0	0.0	0.0	0.16	0.02
	40.0	1.0	0.0	1.0	3.27	0.10
	40.0	1.0	1.0	0.0	10.30	0.31
	40.0	1.0	1.0	1.0	39.40	0.94
	30.0	0.5	0.5	0.0	2.62	0.06
	30.0	0.5	0.5	1.0	9.48	0.20
	30.0	0.5	0.0	0.5	1.96	0.04
	30.0	0.5	1.0	0.5	13.41	0.22
	30.0	0.0	0.5	0.5	4.09	0.10
	30.0	1.0	0.5	0.5	7.52	0.14
	20.0	0.5	0.5	0.5	4.91	0.06
	40.0	0.5	0.5	0.5	8.67	0.28

Table 5: *Cam pipe clamp.*

Surface	$x[0]$	$x[1]$	$x[2]$	Force [N]	Torque [N · m]
Plastic	30.0	0.5	0.5	-	0.28
	40.0	0.5	0.5	-	0.63
	30.0	1.0	0.5	-	0.30
	20.0	0.0	1.0	-	0.21
	40.0	0.0	1.0	-	0.68
	30.0	0.5	1.0	-	0.45
	20.0	1.0	1.0	-	0.29
	40.0	1.0	1.0	-	0.94
	20.0	0.0	0.0	-	0.24
	40.0	0.0	0.0	-	0.38
	30.0	0.5	0.0	-	0.26
	20.0	1.0	0.0	-	0.13
	40.0	1.0	0.0	-	0.64
	30.0	0.0	0.5	-	0.36
	20.0	0.5	0.5	-	0.20
Metal	30.0	0.5	0.5	-	0.04
	40.0	0.5	0.5	-	0.15
	30.0	1.0	0.5	-	0.05
	20.0	0.0	1.0	-	0.09
	40.0	0.0	1.0	-	0.17
	30.0	0.5	1.0	-	0.14
	20.0	1.0	1.0	-	0.16
	40.0	1.0	1.0	-	0.14
	20.0	0.0	0.0	-	0.02
	40.0	0.0	0.0	-	0.11
	30.0	0.5	0.0	-	0.06
	20.0	1.0	0.0	-	0.03
	40.0	1.0	0.0	-	0.10
	30.0	0.0	0.5	-	0.07
	20.0	0.5	0.5	-	0.07

Table 6: *Suction cup.*

Surface	$x[0]$	$x[1]$	$x[2]$	Force [N]	Torque [N · m]
Glass	0.5	0.5	0.5	22.8	0.16
	1.0	0.5	0.5	24.0	0.19
	0.5	1.0	0.5	43.7	0.35
	0.0	0.0	1.0	19.5	0.15
	1.0	0.0	1.0	19.7	0.21
	0.5	0.5	1.0	37.3	0.50
	0.0	1.0	1.0	22.7	0.27
	1.0	1.0	1.0	43.9	0.74
	0.0	0.0	0.0	4.6	0.02
	1.0	0.0	0.0	6.4	0.04
	0.5	0.5	0.0	13.1	0.08
	0.0	1.0	0.0	18.1	0.11
	1.0	1.0	0.0	16.7	0.15
	0.5	0.0	0.5	11.0	0.08
	0.0	0.5	0.5	15.0	0.11

Table 7: *Toggle clamp.*

Surface	$x[0]$	$x[1]$	$x[2]$	Force [N]	Torque [N · m]
Plastic	30.0	15.0	0.5	24.0	0.42
	40.0	15.0	0.5	24.4	0.52
	30.0	30.0	0.5	11.8	0.28
	20.0	0.0	1.0	24.0	0.47
	40.0	0.0	1.0	31.1	0.82
	30.0	15.0	1.0	43.7	1.11
	20.0	30.0	1.0	47.9	1.01
	40.0	30.0	1.0	21.8	0.59
	20.0	0.0	0.0	11.3	0.19
	40.0	0.0	0.0	15.9	0.12
	30.0	15.0	0.0	13.4	0.17
	20.0	30.0	0.0	8.8	0.17
	40.0	30.0	0.0	5.1	0.10
	30.0	0.0	0.5	13.1	0.21
	20.0	15.0	0.5	23.2	0.32

Table 8: *Box clamp.*

Surface	$x[0]$	$x[1]$	$x[2]$	$x[3]$	Force [N]	Torque [N · m]
Plastic	60.0	20.0	0.5	0.5	8.18	-
	50.0	15.0	0.0	0.0	0.82	-
	50.0	15.0	0.0	1.0	0.98	-
	50.0	15.0	1.0	0.0	6.70	-
	50.0	15.0	1.0	1.0	8.01	-
	50.0	25.0	0.0	0.0	1.14	-
	50.0	25.0	0.0	1.0	4.25	-
	50.0	25.0	1.0	0.0	8.99	-
	50.0	25.0	1.0	1.0	15.70	-
	70.0	15.0	0.0	0.0	4.41	-
	70.0	15.0	0.0	1.0	0.33	-
	70.0	15.0	1.0	0.0	8.34	-
	70.0	15.0	1.0	1.0	15.37	-
	70.0	25.0	0.0	0.0	2.62	-
	70.0	25.0	0.0	1.0	4.25	-
	70.0	25.0	1.0	0.0	11.45	-
	70.0	25.0	1.0	1.0	17.99	-
	60.0	20.0	0.5	0.0	7.68	-
	60.0	20.0	0.5	1.0	9.48	-
	60.0	20.0	0.0	0.5	3.60	-
	60.0	20.0	1.0	0.5	12.10	-
	60.0	15.0	0.5	0.5	8.01	-
	60.0	25.0	0.5	0.5	8.01	-
	50.0	20.0	0.5	0.5	10.14	-
	70.0	20.0	0.5	0.5	7.19	-

Algorithm 1 Computing intrinsic free motions

```
1:  $\mathcal{F} \leftarrow$ empty set of free motions;
2:  $\mathcal{C} \leftarrow$ empty set of constraints;
3:  $\mathcal{T} \leftarrow$ all triangles from mesh;
4:  loop
5:      Solve: $H \leftarrow \min_\phi B(\mathcal{T}, \phi)$ s.t. $\mathcal{C}$;
6:      $\phi^{\min} \leftarrow$ the optimal argument;
7:      if $H < \epsilon$ then
8:          $\mathcal{F} \leftarrow \mathcal{F} \cup \{\phi^{\min}\}$;
9:          $\mathcal{C} \leftarrow \mathcal{C} \cup \{\phi \cdot \phi^{\min} < \alpha\}$;
10:     else
11:         return $\mathcal{F}$;
12:     end if
13: end loop
```

this algorithm takes less than one second to find the intrinsic free motions.

3.3 Splitting

To verify that every part of the holder can be attached to and detached from the object, we first ensure that the local holdability of each holder part is zero, and then ensure that there is an intersection-free motion path for each part. To find such an intersection-free path, we use the information of the optimal twist motion ϕ_k that satisfies $H(\mathcal{T}_k, \phi_k) = 0$, where $\mathcal{T}_k$ is the sets of triangles for the split parts so that $\mathcal{T} = \bigcup \mathcal{T}_k$. Then, we run a dynamic rigid body simulation, where every holder part and the target object are assumed to be rigid, with virtual spring forces that pull each holder part towards their computed twist motion ϕ_k. If the global intersection check fails, the system rejects the current cut plane, and a different one is chosen either automatically (if symmetry planes are used) or by the user.

3.4 Holder Variations

We show various free-form holders generated during the computation of a diverse set of suggestions in Fig. 8. These variations come from the bias parameters for shell computation and the seed point positions. As shown in the figure, some holders are visually very similar; this indicates the need of our algorithm of selecting a small number of diverse, distinguishable holders from these variations.

References

CGAL. Computational Geometry Algorithms Library. `http://www.cgal.org`.

OPENSCAD. The Programmers Solid 3D CAD Modeller. `http://www.openscad.org/`.

THANH-VINH, N., TAKAHASHI, H., KAN, T., NODA, K., MATSUMOTO, K., AND SHIMOYAMA, I. 2011. Micro suction cup array for wet/dry adhesion. In *Micro Electro Mechanical Systems (MEMS), 2011 IEEE 24th International Conference on*, 284–287.

```
function solve_quad_positive(a, b, c) = (- b + sqrt(b * b - 4.0 * a * c)) /
      (2.0 * a);
function solve_quad_negative(a, b, c) = (- b - sqrt(b * b - 4.0 * a * c)) /
      (2.0 * a);
function sqr(x) = x * x;
function len(x, y) = sqrt(sqr(x) + sqr(y));

module prism(s, theta1, theta2, h) {
  linear_extrude(height = h, center = true) {
    polygon(points = [[0.0, 0.0], [s * cos(theta1), s * sin(theta1)], [s *
       cos(theta2), s * sin(theta2)]], paths=[[0, 1, 2]]);
  }
}
```

Listing 1: *Utility functions.*

```
module cylindrical_fillet(x, s, r) {
  rot = s ? 0.0 : 180.0;
  translate(x) rotate(rot, [1.0, 0.0, 0.0]) {
    cylinder(r1 = r + 0.90 * r, r2 = r, h = 0.45 * r, center = false);
    cylinder(r1 = r + 0.45 * r, r2 = r, h = 0.90 * r, center = false);
  }
}

module cylinder_with_fillets(x, r, h, tol) {
  translate(x) {
    cylindrical_fillet([0.0, 0.0, - 0.5 * h],  true, r + tol * 1.4142 * 0.5);
    cylindrical_fillet([0.0, 0.0, + 0.5 * h], false, r + tol * 1.4142 * 0.5);
    cylinder(r = r + tol, h = h, center = true);
  }
}
```

Listing 2: *Fillet.*

```
//////////////////////
/* Basic parameters */
//////////////////////

inDia = x[0] * 0.9;
opLoc = 0.7 + (x[1] - 0.5) * 0.4;
thick = 1.20 + inDia * 0.10 * (0.2 + x[2] * 0.8);
depth = 8.0 + inDia * (0.1 + x[3] * 0.5);

///////////////
/* Utilities */
///////////////

function f_1(a, b, r1, r2) = - sqr(a) - sqr(b) - 2.0 * r1 * r2 - sqr(r2);
function f_2(a, b)         = 4.0 * (sqr(a) + sqr(b));
function f_3(a, c)         = 4.0 * a * c;
function f_4(b, c, r1, r2) = sqr(c) - 4.0 * sqr(b) * sqr(r1 + r2);
function solve_tangent_circle_center(p, r1, r2) = solve_quad_positive(f_2(p[0],
      p[1]), f_3(p[0], f_1(p[0], p[1], r1, r2)), f_4(p[1], f_1(p[0], p[1], r1,
      r2), r1, r2));

/////////////////////////
/* Internal parameters */
/////////////////////////

t  = thick;
l  = 0.15 * inDia;
d  = depth;
a  = 0.5 * inDia * opLoc;
r1 = 0.5 * inDia;
r2 = r1 + t;
y1 = - a;
x1 = sqrt(sqr(r1) - sqr(y1));
x4 = x1 + t;
x3 = x1;
x2 = x4;
y2 = - sqrt(sqr(r2) - sqr(x2));
y4 = y2 - l;
y3 = y4;
x5 = solve_tangent_circle_center([x4, y4], r1, r2);
```

```
y5 = - sqrt(sqr(r1 + r2) - sqr(x5));
r4 = len(x5 - x3, y5 - y3);
r3 = r2;
x6 = solve_tangent_circle_center([x3, y3], r3, r1);
y6 = - sqrt(sqr(r3 + r1) - sqr(x6));
r6 = len(x6 - x1, y6 - y1);
theta1 = acos(- x5 / len(x5, y5));
theta2 = acos(- (x5 - x3) / len(x5 - x3, y5 - y3));
theta3 = acos(- x6 / len(x6, y6));
theta4 = acos(- (x6 - x3) / len(x6 - x3, y6 - y3));

//////////
/* Misc. */
//////////

$fn = 120;
e   = 0.1;
i   = 1000.0;

/////////////
/* Modules */
/////////////

module base() {
  difference() {
    difference() {
      union() {
        cylinder(r = r2, h = d, center = true);
        translate([0.0, 0.5 * y4, 0.0]) cube(size = [x4 * 2.0, - y4, d], center = true);
      }
      cylinder(r = r1, h = d + e, center = true);
    }
    translate([0.0, 0.5 * (y3 - i), 0.0]) cube(size = [x3 * 2.0, - y3 + i, d + e], center = true);
  }
}

module out(s) {
  scale([s, 1.0, 1.0]) {
    intersection() {
      difference() {
        translate([x5, y5, 0.0]) cylinder(r = r4, h = d, center = true);
        translate([x5, y5, 0.0]) cylinder(r = r1, h = d + e, center = true);
      }
      translate([x5, y5, 0.0]) prism(i, theta1, theta2, d);
    }
  }
}

module in(s) {
  scale([s, 1.0, 1.0]) {
    intersection() {
      difference() {
        translate([x6, y6, 0.0]) cylinder(r = r6 + e, h = d + e, center = true);
        translate([x6, y6, 0.0]) cylinder(r = r3, h = d + 2.0 * e, center = true);
      }
      translate([x6, y6, 0.0]) prism(i, theta3, theta4, d + e);
    }
  }
}

module snap_pipe_clamp() {
  difference() {
    union() {
      base();
      out(1.0);
      out(-1.0);
    }
    union() {
      in(1.0);
      in(-1.0);
      translate([0.0, y4 - i / 2.0, 0.0]) cube(size = [i, i, d + e], center = true);
    }
  }
}
```

Listing 3: *Snap Pipe Clamp.*

```
/////////////////////////
/* Internal parameters */
/////////////////////////

d      = x[0];
s      = 25.4;
p      = 0.55;
w      = s * 1.000;
t      = s * 0.125;
r      = 0.5 * d;
h_open = s * 0.035 + 0.5;
w_open = s * 0.5 + 0.5;
r_0    = r;              // innermost radius
r_1    = r_0 + t;
r_2    = r_1 + h_open;
r_3    = r_2 + t;        // outermost radius
x_b    = sqrt(r_1 * r_1 - r * r * p * p);

//////////
/* Misc. */
//////////

$fn    = 120;
eps    = 0.1;

/////////////
/* Modules */
/////////////

module strap_pipe_clamp() {
    intersection() {
        difference() {
            difference() {
                difference() {
                    cylinder(h = w, r = r_3, center = true);
                    cylinder(h = w + eps, r = r_0, center = true);
                }
                difference() {
                    cylinder(h = w_open, r = r_2, center = true);
                    cylinder(h = w_open + eps, r = r_1, center = true);
                }
            }
            cube(size=[r_3 * 2.0 + eps, r * p * 2.0, w + eps], center = true);
        }
        translate([0.0, 0.5 * (r_3 + r * p), 0.0]) cube(size=[x_b * 2.0, r_3 - r * p + eps, w + eps], center = true);
    }
}
```

Listing 4: *Strap Pipe Clamp.*

```
/////////////////////////
/* Internal parameters */
/////////////////////////

r_i = x[0] * 0.5;
a   = 2.0 + (x[0] / 30.0) * 1.2 + 2.0 * 0.4 * (x[1] - 0.5);
t   = 4.5 + 2.0 * 1.0 * (x[2] - 0.5);
tol = 0.02 * 25.4;
g   = 60.0;
c   = 3.0;
r_1 = r_i;
r_2 = r_i + a;
r_3 = r_i + 2.0 * a;
w   = 3.0 * t + 2.0 * tol;
tt0 = tan(g + 15.0 + 90.0);
A0  = 1.0 + sqr(tt0);
B0  = - 2.0 * a * tt0;
C0  = sqr(a) - sqr(r_2);
x0  = solve_quad_negative(A0, B0, C0);
y0  = tt0 * x0;
tt1 = tan(75.0 + c);
A1  = 1.0 + sqr(tt1);
B1  = - 2.0 * a * tt1;
C1  = sqr(a) - sqr(r_2);
x1  = solve_quad_positive(A1, B1, C1);
y1  = tt1 * x1;
tt2 = tan(75.0);
A2  = 1.0 + sqr(tt2);
B2  = - 2.0 * a * tt2;
C2  = sqr(a) - sqr(r_2);
x2  = solve_quad_positive(A2, B2, C2);
y2  = tt2 * x2;

//////////
/* Misc. */
//////////

$fn  = 120;
eps  = 0.010;
inf  = 500.0;

////////////
/* Modules */
////////////

module x_neg() {
  translate([- inf / 2.0, 0.0, 0.0]) cube(size=inf, center=true);
}

module x_pos() {
  translate([+ inf / 2.0, 0.0, 0.0]) cube(size=inf, center=true);
}

module ring() {
  module uni() {
    intersection() {
      difference() {
        translate([0.0, a, 0.0]) cylinder(r = r_3, h = w, center = true);
        union() {
          cylinder(r = r_1, h = w + eps, center = true);
          negative();
        }
      }
      x_neg();
    }
    ring_pos();
    claw();
  }

  module negative() {
    r = inf;
    linear_extrude(height = w + eps, center = true, convexity = 5) {
      polygon(points = [[0.0, 0.0], [eps, r], [- r * sin(g), r * cos(g)]], paths = [[0, 1, 2]]);
    }
  }

  module cut() {
    r = inf;
    prism(r, 60.0, 120.0 + g + 15.0, t + 2.0 * tol);
  }

  module claw() {
    intersection() {
      difference () {
        translate([0.0, r_1 + 0.5 * 3.0 * a, 0.0]) cube(size=[3.0 * a, 3.0 * a, w], center=true);
        union() {
          translate([0.0, r_1 + 0.5 * 3.0 * a + 0.5 * a, 0.0]) cylinder(r = a, h = w + eps, center = true);
          translate([0.0, r_i + 3.0 * a, 0.0]) cube(size=[0.8 * a, 0.8 * a, w + eps], center=true);
        }
      }
      x_neg();
    }
  }

  module pipe() {
    h = t + 2.0 * tol;
    cylinder_with_fillets([x0, y0, 0.0], 0.5 * a, h + eps, 0.0);
  }

  module ring_pos() {
    intersection() {
      difference() {
        cylinder(r = r_2, h = w, center = true);
        cylinder(r = r_1, h = w + eps, center = true);
      }
      x_pos();
    }
  }

  difference() {
    uni();
    cut();
  }
  pipe();
}

module lever() {
  r = inf;
  module pos() {
    translate([0.0, r_2 + a, 0.0]) cylinder(r = a, h = w, center = true);
    intersection() {
      difference() {
        translate([0.0, a, 0.0]) cylinder(r = r_3, h = w, center = true);
        cylinder(r = r_2, h = w + eps, center = true);
      }
      prism(r, -45.0, 90.0, w + eps);
    }
  }

  module neg() {
    r = inf;
    prism(r, 60.0, 120.0, t + 2.0 * tol);
  }

  module pipe() {
    h = t + 2.0 * tol;
    cylinder_with_fillets([x2, y2, 0.0], 0.5 * a, h + eps, 0.0);
  }

  difference() {
    pos();
    neg();
  }
  pipe();
}

module link() {
  r = inf;

  module circles_pos() {
    translate([x1, y1, 0.0]) cylinder(r = a + tol, h = t, center = true);
    translate([x0, y0, 0.0]) cylinder(r = a + tol, h = t, center = true);
  }
  module base () {
    intersection() {
      difference() {
        translate([0.0, a, 0.0]) cylinder(r = r_3 + tol, h = t, center = true);
        translate([0.0, a, 0.0]) cylinder(r = r_i - tol, h = t + eps, center = true);
      }
      prism(r, 75.0 + c, g + 105.0, t + eps);
    }
  }
  module circles_neg() {
    h = t + 2.0 * tol;
    cylinder_with_fillets([x1, y1, 0.0], 0.5 * a, h, tol);
    cylinder_with_fillets([x0, y0, 0.0], 0.5 * a, h, tol);
  }

  difference() {
    union () {
      circles_pos();
      base();
    }
    circles_neg();
  }
}

module cam_clamp() {
  ring();
  translate([x0, y0, 0.0]) rotate(45.0, [0.0, 0.0, 1.0]) translate([-x0, -y0, 0.0]) union () {
    link();
    rotate(c, [0.0, 0.0, 1.0]) translate([x2, y2, 0.0]) rotate(30.0, [0.0, 0.0, 1.0]) translate([- x2, - y2, 0.0]) lever();
  }
}
```

Listing 5: *Cam Pipe Clamp.*

```
/////////////////////////
/* Internal parameters */
/////////////////////////

d  = x[0] * 1.20 + 0.40;
r  = x[1] * 2.00 + 3.575;
s  = 25.4 * (x[2] * 0.06 + 0.08);
h  = d + 2.0;
q  = 63.9;
sd = s * d;
sh = s * h;
sr = s * r;
R  = (sd * sd + sr * sr) / (2.0 * sd);
y0 = (sd * sd - sr * sr) / (2.0 * sd);
x0 = sr - (sh / tan(q));

///////////
/* Misc. */
///////////

$fn = 90;

/////////////
/* Modules */
/////////////
module suction_cup() {
  difference() {
    cylinder(r1 = sr, r2 = x0, h = sh);
    translate([0.0, 0.0, y0]) sphere(r = R);
  }
}
```

Listing 6: *Suction Cup.*

```
/////////////////////////
/* Internal parameters */
/////////////////////////

target_space     = x[0];
target_angle     = x[1];
area_factor      = x[2];
rubber_thickness = 0.125 * 25.4;
tol              = 0.5;
angle            = 2.5;
arm_length       = 55.0 + 2.0 * 5.0 * (area_factor - 0.5);
depth            = 10.0 + 2.0 * 3.0 * (area_factor - 0.5);
w                = 10.0 + 2.0 * 3.0 * (x[2] - 0.5);
alpha            = arm_length - 0.5 * w;
beta             = alpha - 0.5 * w;
gamma            = beta * tan(target_angle);
space            = target_space - 0.060 * 25.4 + 2.0 * rubber_thickness + gamma;
overcenter       = 0.25 * 25.4;
piv_dia          = 0.5 * w;
h1               = overcenter;
h2               = h1 + 0.5 * w;
pivot_space      = h2 + 0.5 * w;
base1y           = space + 0.5 * w;
base2y           = base1y + pivot_space;
top_depth        = 3.0 * depth + 2.0 * tol;
L                = len(0.5 * arm_length, h1);
cylinder_radius  = piv_dia * 0.5 - tol;

///////////
/* Misc. */
///////////

inf = 1000.0;
ep  = 0.030;
$fn = 60;

/////////////
/* Modules */
/////////////

module base()
{
  a  = arm_length;
  y1 = base1y;
  y2 = base2y;
  d  = depth;
  r  = cylinder_radius;

  module positive() {
    tmp = (a + w / 2) / 2 - w / 2;
    translate([tmp, 0.0, 0.0])  cube(size=[a + w / 2.0, w, top_depth], center=true);
    translate([0, -w/2-y2/2,0]) cube(size=[w, y2 + ep, d], center=true);
    translate([0, -y2-w/2, 0])  cylinder(r=w/2, h=d, center=true);
  }

  module negative() {
    cylinder_with_fillets([0,-y2-w/2,0], r, d + 2.0 * tol, tol);
    cylinder_with_fillets([0,-y1-w/2,0], r, d + 2.0 * tol, tol);
  }

  module anglePart() {
    eps = 1e-5;
    linear_extrude(height = top_depth, center = true, convexity = 5) {
      polygon(points = [[0.5 * w - eps, - 0.5 * w + eps], [a, - 0.5 * w + eps], [0.5 * w - eps, - 0.5 * w - tan(target_angle) * (a - 0.5 * w)]], paths = [[0, 1, 2]]);
    }
  }

  module f() {
    difference() {
      translate([0.0, - w / 2.0 - d / 2.0, d]) cube(size=[w, d + ep, d + ep], center=true);
      translate([0.0, - w / 2.0 - d, 1.5 * d]) rotate(90.0, [0.0, 1.0, 0.0]) cylinder(r=d, h=w+ep,center=true);
    }
    difference() {
      translate([0.0, - w / 2.0 - d / 2.0, - d]) cube(size=[w, d + ep, d + ep], center=true);
      translate([0.0, - w / 2.0 - d, - 1.5 * d]) rotate(90.0, [0.0, 1.0, 0.0]) cylinder(r=d, h=w+ep,center=true);
    }
  }

  difference() {
    positive();
    negative();
  }
  f();
  if (target_angle > 1e-04) anglePart();
}

module arm()
{
    l = arm_length;
    d = depth;

    module p1() {
        cylinder(r=w/2,h=top_depth,center=true);
        translate([l/2+0.25*w,0,0]) cube(size=[l+0.5*w,w,top_depth],center=true);
        translate([l,-pivot_space/2,0]) cube(size=[w, pivot_space,top_depth],center=true);
        translate([l,-pivot_space,0]) cylinder(r=w/2,h=top_depth,center=true);
    }

  module p2() {
    tmp1 = d+2*tol;
    tmp2 = (pivot_space-w/2)*2;
    cube(size=[w*2,inf,tmp1],center=true);
    translate([0,-pivot_space,0]) cube(size=[inf,tmp2-ep,tmp1],center=true);
  }

  module p3() {
    r = cylinder_radius;
    cylinder_with_fillets([0.0, 0.0, 0.0], r, d + 2.0 * tol + ep, 0.0);
    cylinder_with_fillets([l,-pivot_space,0], r, d + 2.0 * tol + ep, 0.0);
  }

  module prism() {
    x1 = w;
    x2 = arm_length + 0.5 * w;
    dy = tan(angle) * (arm_length - 0.5 * w);
    linear_extrude(height = top_depth, center = true)
      polygon(points=[[x1, 0.5 * w], [x2, 0.5 * w], [x2, 0.5 * w + dy]]);
  }

  difference() { p1(); p2(); }
  p3();
  prism();
}

module lever()
{
    d = top_depth;

    module p1() {
        tmp = h2 + w;
        cylinder(r=w/2,h=d,center=true);
        translate([0.5*L,0,0]) cube(size=[L,w,d],center=true);
        translate([L,0,0]) cylinder(r=w/2,h=d,center=true);
        translate([L,-tmp/2,0]) cube(size=[w,tmp,d],center=true);
        translate([L,-tmp,0]) cylinder(r=w/2,h=d,center=true);
        translate([1.5*L,-tmp,0]) cube(size=[L,w,d],center=true);
        translate([2.0*L,-tmp,0]) cylinder(r=w/2,h=d,center=true);
    }

  module p2() {
    cube(size=[inf,h2*2,depth+2*tol],center=true);
  }

  module p3() {
    cylinder_with_fillets([0.0, 0.0, 0.0], cylinder_radius, depth + 2.0 * tol + ep, 0.0);
    cylinder_with_fillets([  L, 0.0, 0.0], cylinder_radius, depth + 2.0 * tol + ep, 0.0);
  }

  difference() { p1(); p2(); }
  p3();
}

module link()
{
    module p1() {
        cylinder(r=w/2,h=depth,center=true);
        translate([L/2,0,0]) cube(size=[L,w,depth],center=true);
        translate([L,0,0]) cylinder(r=w/2,h=depth,center=true);
    }

  module p2() {
    cylinder_with_fillets([0, 0, 0], cylinder_radius, depth + 2.0 * tol, tol);
    cylinder_with_fillets([L, 0, 0], cylinder_radius, depth + 2.0 * tol, tol);
  }

  difference() { p1(); p2(); }
}

module rubber_top()
{
  length = (arm_length - 0.5 * w) / cos(target_angle);
  cube(size=[top_depth, rubber_thickness, length], center=true);
}

module rubber_bottom()
{
  length = (1.0 / cos(angle)) * (arm_length - 0.5 * w);
  cube(size=[top_depth, rubber_thickness, length], center=true);
}

module assemble()
{
  t  = space;
  x  = - w / 2.0;
  y1 = w/2 + t/2;
  y2 = -w/2 - t/2;
  y3 = -base2y+t/2;
  q  = + acos(arm_length / (2.0 * L)); // positive: open, negative: close
  translate([x, y1, 0]) base();
  translate([x, y2, 0]) arm();
  translate([x, y3, 0]) rotate(- q, [0.0, 0.0, 1.0]) lever();
  translate([x + arm_length, y3, 0]) rotate(180.0 + q, [0.0, 0.0, 1.0]) link();
}

module rubbers()
{
  {
    p1_y = - 0.5 * space;
    p1_z = - w + 0.5 * w;
    p2_y = p1_y + tan(angle) * (arm_length - 0.5 * w);
    p2_z = - arm_length;
    pc_y = (p1_y + p2_y) * 0.5;
    pc_z = (p1_z + p2_z) * 0.5;
    translate([0.0, pc_y, pc_z]) rotate(angle, [1.0, 0.0, 0.0]) translate([0.0, 0.5 * rubber_thickness, 0.0]) rubber_bottom();
  }
  {
    l = (arm_length - 0.5 * w) / cos(target_angle);
    y1 = space * 0.5;
    z1 = - (arm_length - 0.5 * w);
    y2 = y1 - l * sin(target_angle);
    translate([0.0, (y1 + y2) * 0.5, 0.5 * z1]) rotate(target_angle, [1.0, 0.0, 0.0]) translate([0.0, - 0.5 * rubber_thickness, 0.0]) rubber_top();
  }
}

module toggle_clamp_hard() {
  translate([0.0, 0.5 * gamma, 0.5 * w]) rotate(90.0, [0.0, 1.0, 0.0]) assemble();
}

module toggle_clamp_soft() {
  translate([0.0, 0.5 * gamma, 0.5 * w]) rubbers();
}
```

Listing 7: *Toggle Clamp.*

```
/////////////////////////
/* Internal parameters */
/////////////////////////

w    = x[0];
h    = x[1];
t    =  3.0 + 2.0 * 1.5 * (x[2] - 0.5);
d    = 12.0 + 2.0 * 4.0 * (x[3] - 0.5);
l    = w * 0.08;
tol  = 0.01 * 25.4;
inW  = w + tol;
inH  = h + tol;
outW = inW + 2 * t;
outH = inH + 2 * t;
w2   = outW - 2.0 * (l + t);

//////////
/* Misc. */
//////////

ep   = 0.1;

/////////////
/* Modules */
/////////////

module box_clamp() {
  difference() {
    difference() {
      cube(size=[outW, outH, d   ], center = true);
      cube(size=[ inW,  inH, d+ep], center = true);
    }
    translate([0.0, - (outH / 4.0 + ep * 2.0), 0.0]) cube(size=[w2, outH / 2.0 + ep, d + ep], center=true);
  }
}
```

Listing 8: *Box Clamp.*

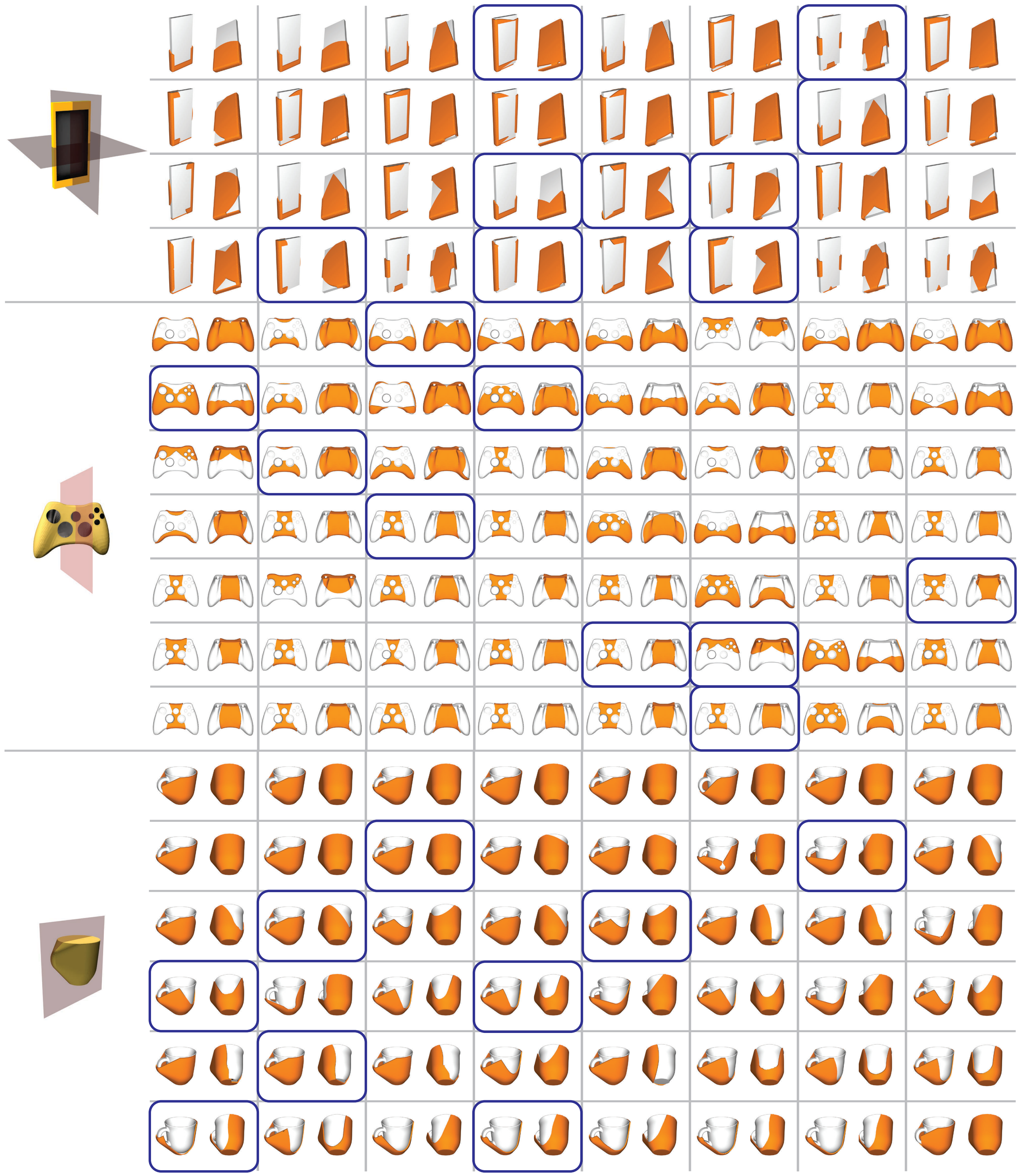


Figure 8: *Variations of generated free-form holders. The holders annotated with boxes are the holders selected by our algorithm of selecting a diverse set holders. Top row: a rectangular mobile phone, where the display part is specified as a constrained (uncoverable) region, and the top direction is specified as a free direction. Middle row: game controller. Bottom row: a mug, where the convex hull is used for shell computation and the top direction is specified as a free-direction.*