Motion Annotation Programs: A Scalable Approach to Annotating Kinematic Articulations in Large 3D Shape Collections

Xianghao Xu[1] David Charatan[1] Sonia Raychaudhuri[2] Hanxiao Jiang[2] Mae Heitmann[1]
Vladimir Kim[3] Siddhartha Chaudhuri[3, 4] Manolis Savva[2] Angel X. Chang[2] Daniel Ritchie[1]

[1]Brown University [2]Simon Fraser University [3]Adobe Research [4]IIT Bombay

Abstract

3D models of real-world objects are essential for many applications, including the creation of virtual environments for AI training. To mimic real-world objects in these applications, objects must be annotated with their kinematic mobilities. Annotating kinematic motions is time-consuming, and it is not well-suited to typical crowdsourcing workflows due to the significant domain expertise required. In this paper, we present a system that helps individual expert users rapidly annotate kinematic motions in large 3D shape collections. The organizing concept of our system is motion annotation programs*: simple, re-usable procedural rules that generate motion for a given input shape. Our interactive system allows users to author these rules and quickly apply them to collections of functionally-related objects. Using our system, an expert annotated over 1000 joints in under 3 hours. In a user study, participants with no prior experience with our system were able to annotate motions 1.5x faster than with a baseline manual annotation tool.*

1. Introduction

3D models of real-world objects are essential for AR and VR, design and advertising, games, and fabrication. They are also increasingly used in synthetic environments for training embodied AI agents [11, 29, 16, 26, 22, 27]. For these applications, 3D assets need to have rich annotations of real-world attributes such as material properties, part compositions, affordances, and kinematics.

The last of these, kinematics, is especially challenging to annotate, since it encapsulates many details: joint types (e.g., hinge vs prismatic), joint parameters (e.g., axes of rotation or translation direction), constraints (e.g., bounds on rotation angle or amount of translation), and indication of which parts are affected by the joint motion. Creating such annotations with simple direct-manipulation interfaces requires minutes per model; this is slow for existing public 3D shape datasets [5] and totally infeasible for large commercial repositories containing millions of assets [21, 4].

Massive data annotation is usually accomplished via crowdsourcing. However, annotating kinematics requires non-trivial domain expertise in 3D user interfaces, geometry, and which joint parameters produce feasible articulations. This makes the task ill-suited for crowd work. Thus, prior efforts employed experts with the appropriate background [25, 27, 30]. However, these efforts use tools for annotating models one by one, which do not scale to large repositories and do not effectively use the expert's time. Our hypothesis is that experts can be more effective by expressing their knowledge via generalizable *rules*. A single rule may not work in all cases (e.g. doors with different hinge structures may need different logic), but a small number of rules can cover a large set of functionally-related joints.

In this paper, we propose *Motion Annotation Programs* (MAPs). MAPs are procedural rules which generate motion annotations when applied to object parts (i.e. the type, axis, and range of motion). We present an interactive system for an expert to author and apply MAPs to *collections* of functionally-related parts. Our interface lets the user inspect and operate on many related parts simultaneously, speeding up annotation. Our system also provides automatic suggestions of what parts are functionally-related enough to be annotated together, and what rule will work best for each part in a functionally-related collection. The system learns from user feedback about these suggestions, making them more relevant over time to further speed up annotation.

We focus on commonplace indoor objects, whose kinematics are governed primarily by prismatic (translational) and revolute (rotational) joints. We evaluate our system by annotating objects in the PartNet Mobility dataset [27], an existing dataset of kinematically-articulated 3D shapes. With our system, an expert user annotated 1170 joints in 174 minutes. We also conduct a user study of our system,

comparing it to an interface for annotating objects one by one. Participants with no prior experience using our system annotated joints more than 1.5x faster using our system.

In summary, our contributions are:

1. Motion Annotation Programs (MAPs), including a domain-specific language for authoring them.
2. An interface for authoring MAPs and visualizing their effects on large collections of related object parts.
3. A learning-based approach for grouping object parts into collections and suggesting MAPs for parts.

Source code for our system is available at https://github.com/brownvc/articulations; a running demo can be found at http://articulations.cs.brown.edu.

2. Related Work

Mobility Analysis of 3D Objects Prior work has looked at analysis of 3D part mobility. [10] analyze static snapshots of articulating objects to predict mobility. [31] co-segment and predict mobility for pairs of input objects. Other work has used mobility analysis to improve 3D shape manipulation [28, 19] or reconstruction [12]. RPM-Net [30] predicts movable parts and their motions from point cloud data. Similarly, [25] proposed a neural network for 3D point cloud mobility prediction. Our work is complementary to these past efforts: we provide a tool which can quickly and scalably create the data needed to train these models.

Annotation of Large-Scale 3D Data Prior work has used experts or crowdsourcing to annotate other properties of 3D data: categories and orientations [5], segmentation [6, 32, 1], and salient points [7]. Even with crowdsourcing, one-by-one annotation can be prohibitively expensive. Thus, prior work explored frameworks to scale this process. PartNet [14] used an expert-defined part hierarchy and asked crowd workers questions to map object sub-parts to nodes in this hierarchy. Yi et al. [32] combine manual labeling, automatic predictions, and user verification to reduce annotation time for part segmentation. SceneNN [15] leverages user interaction to refine automatic segmentations of 3D scene reconstructions. We provide a tool that enables a single expert to annotate large-scale data using procedural rules. This reduces manual effort and the need for quality control mechanisms essential to crowdsourcing.

Human-in-the-loop machine learning We leverage human feedback to drive a machine learning system. This has been termed 'interactive machine learning' by [8], who use human feedback to train an image foreground segmentation model. Human feedback has also been used for data transformation, visualization, and natural language translation [9]. Active learning is a related methodology that has been adopted for annotating 2D images [2, 3, 18, 23, 24].

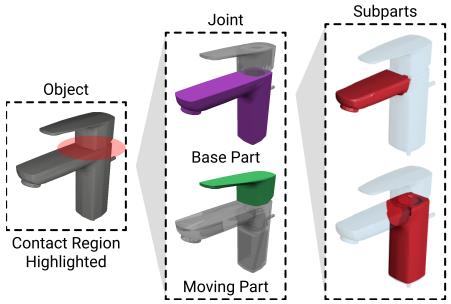


Figure 1. In our system, users annotate kinematic *joints*. A joint consists of a moving part (green) and a base part (purple). A part may be further decomposed into subparts (red).

3. System Overview

This section overviews our motion annotation system: the input data it requires, the annotations it produces, and its three core components: motion annotation programs, an interface for working with them, and a machine learning model for suggesting which programs to use.

Scope We focus on common indoor objects (doors, cabinets, etc.), as these are most necessary for our motivating applications (e.g., AI agent training). Arbitrary kinematic mechanisms, such as robots or gear assemblies, are beyond our scope. Our system supports prismatic (translational) and revolute (rotational) joints, as these are sufficient for virtually all motions in the data we consider.

Input The input to our system is a dataset of part-segmented 3D shapes; such segmentations can be produced manually, semi-automatically [32], or fully automatically [13]. We use these segmentations to organize the data into *joints*. A joint consists of a *moving part* and the *base part* relative to which the moving part moves (Figure 1). Candidate joints are formed from every pair of connected parts; the user decides which candidate joints are mobile. Our system does not require semantic part labels, nor aligned input shapes. More intricate rules can be authored if the input shape parts are further segmented into sub-parts. The dataset we use for our experiments is both aligned and contains subpart segmentations.

Output The output of our system is a set of *kinematic motion annotations*. A motion annotation is a tuple $(\mathcal{J}, \mathcal{T}, \mathbf{a}, \mathbf{c}, \mathbf{r}, p, [r_{\text{min}}, r_{\text{max}}])$, where:

- $\mathcal{J}$ = the joint to which the annotation applies.
- $\mathcal{T} \in \{\text{Rotation}, \text{Translation}\}$ = the motion type.
- $\mathbf{a} \in \mathbb{R}^3$ = the axis of motion.
- $\mathbf{c} \in \mathbb{R}^3$ = the center of rotation (if $\mathcal{T}$ = Rotation).
- $\mathbf{r} \in \mathbb{R}^3$ = the reference point/vector. For rotations, this is a vector perpendicular to $\mathbf{a}$ which defines where

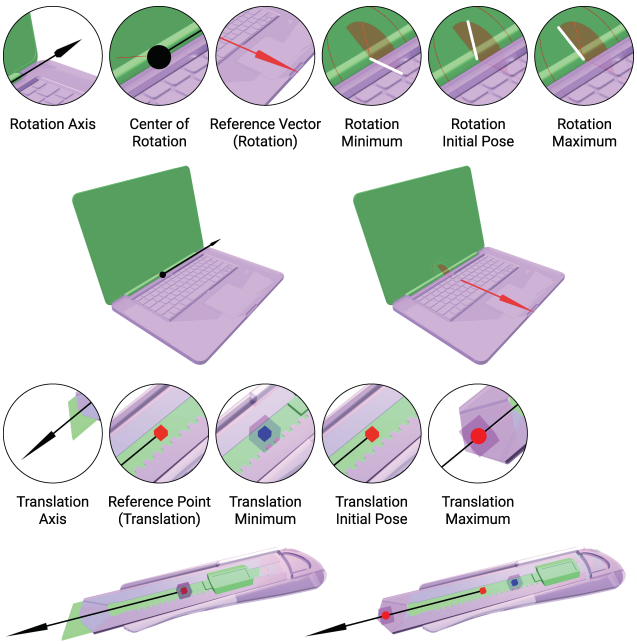


Figure 2. Parameters of our kinematic motion annotations.

$0°$ occurs. For translations, this is a point along the axis of motion at which 0 translation occurs.

- $p \in \mathbb{R}$ = the initial pose of the moving part, relative to the reference point/vector.
- $[r_{\text{min}}, r_{\text{max}}] \in \mathbb{R}^2$ = the range of motion.

Figure 2 illustrates these properties. Given such annotations, a kinematic hierarchy can be created by linking together annotated joints (e.g. where one joint's moving part is another joint's base part). These hierarchies can then be used for e.g. virtual agents to interact with the object.

Motion Annotation Programs Central to our approach, these are the procedural rules that produce kinematic motion annotations for functionally-related joints (Section 4).

Interface An interactive UI for authoring and applying MAPs, centered around building *collections* of functionally-related joints and quickly verifying the effect of MAPs on all joints in the collection (Section 5).

Learning-based Suggestions We provide *learning-based suggestions* of which joints belong in which collections and which rules apply to which joints. The suggestion engine learns from user behavior to improve over time. Our collection-based interface helps here, as each collection learns its own suggestions (Section 6).

4. Motion Annotation Programs

Figure 3 shows a collection of door joints. A simple rule can describe most cases of door motion: the door rotates

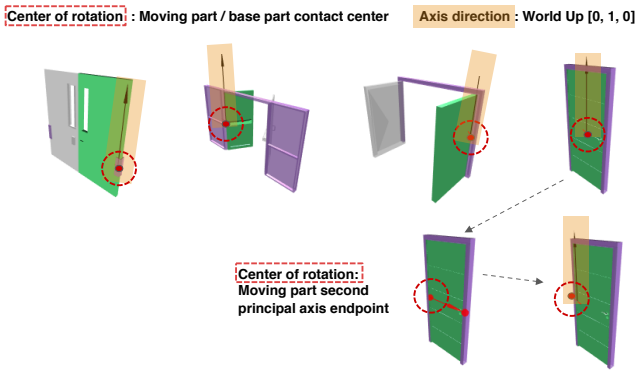


Figure 3. A simple rule (the world up axis) specifies the rotation axes of these doors. One rule gives the center of rotation for the three open doors; a different rule is needed for the closed door.

on the world-up axis, about the contact center between the door and the frame. However, this rule does not produce the correct motion for the rightmost door: it was modeled in a closed pose, so the center of the door-frame contact region does not coincide with the hinge location. Fortunately, another simple rule will do here: the center of rotation is an endpoint of the door's second principal axis. This rule will also work on other closed doors, just as the first rule will work on other open doors. This example highlights the core principle behind MAPs: by writing a few simple rules which each apply to a subset of joints, experts can quickly annotate motions in large 3D shape collections.

Domain-Specific Language To instantiate this principle, we built a domain-specific language (DSL) for authoring MAPs. Our DSL is embedded in Python, which we chose for its popularity within the graphics, vision, and robotics communities. It also offers features that make DSL embedding easier (e.g. operator overloading).

In our DSL, each MAP is expressed as a Python function. We refer to these functions as *rules*. Rules are typed according to the motion type $\mathcal{T}$. They are also further divided into *Axis* vs. *Range* rules, which compute the axis and range of motion, respectively. We separated axis and range rules because we found that this increases the re-usability of individual rules (e.g. an axis rule can be re-used on many joints which exhibit different ranges of motion).

Axis rules An axis rule takes a joint as input and returns the axis direction $\mathbf{a}$ and axis center $\mathbf{c}$ (if $\mathcal{T}$ = Rotation). Figure 4 shows some examples.

Range rules A range rule takes a joint, an axis direction $\mathbf{a}$, and an axis origin $\mathbf{c}$ as input and returns the reference vector/point $\mathbf{r}$, the initial pose p, and the range of motion $[r_{\text{min}}, r_{\text{max}}]$. For rotations, p and $[r_{\text{min}}, r_{\text{max}}]$ represent counter-clockwise rotation from the reference vector. For

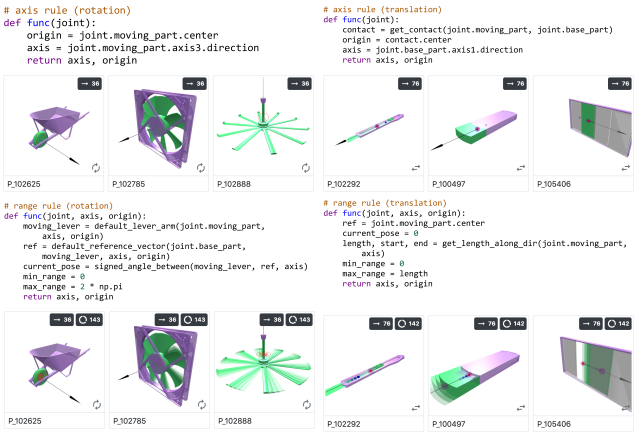


Figure 4. Some example MAP rules. Left: rotation rules (top: an axis rule for wheel-like motions; bottom: a range rule for 360° rotation). Right: translation rules (top: an axis rule for sliding along the longest axis of the base part; bottom: a range rule for letting the moving part translate its full length from its initial pose).

translations, they represent offset along the axis of motion from the reference point. Figure 4 shows some examples.

Data Types The central entities in our system are `Joints`. A `Joint` has a moving `Part` and a base `Part`. A `Part` may have sub-parts, each of which is a `Region` (as are `Parts`, by inheritance). `Region` is the base of our type hierarchy; each `Region` is a patch of geometry with a centroid and principal axes; these can also represent contacts between parts, or arbitrary groupings of parts, subparts, and contacts. Users write code to extracts relevant `Regions` from joints and use their geometric properties to derive motion parameters.

Operators Our DSL provides multiple high-level operators with which to compute motion properties (see the supplemental material for a complete listing):

Selection operators allow selecting salient `Regions` from a joint, from which to extract motion-relevant features. The `get_contact(region1, region2)` operator computes the contact region between any two other regions. Our DSL also provides a jQuery-like API for grouping, sorting, and filtering sets of `Regions` by their properties, allowing the user to zero in on certain geometric features. Such queries are not necessary for most annotations but are nice to have for handling rare "corner cases" (see Figure 5).

Feature extraction operators take `Regions` and extract features that may be relevant for determining part motion. For example, `get_length_along_dir(region, dir)` is useful for determining how far a translating part can move.

5. Collection-based Interface

Given our DSL, we need an interface for authoring programs and applying them to joints. Each authored rule can

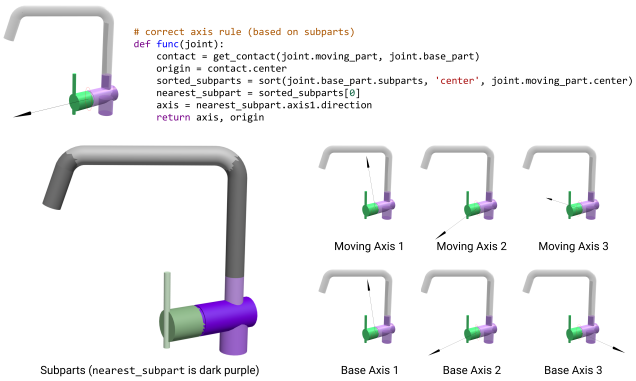


Figure 5. An example of a joint that requires subpart selectors to annotate correctly. No world space axis, nor any principal axis of the moving or base parts, aligns with the correct axis of rotation. The principal axis of one of the subparts does, however.

apply to multiple, functionally-related joints. Thus, the interface allows: (1) creating collections of functionally-related joints, and (2) authoring and applying rules to entire collections of joints at once. This section describes our interface; the supplement includes a demonstration video.

Creating Joint Collections Figure 6 shows our interface. In the Collections Panel (3) the user creates collections of related joints to be annotated. The Dataset Panel (2) shows the database of objects begin annotated; each tile represents a group of object parts. The Top Bar (1) allows filtering, sorting, and grouping parts to help users zero in on parts of interest. Clicking on a group tile expands it into a Shelf of parts contained within that group (4); this two-level hierarchy supports efficient browsing. Clicking on a collection in (3) opens the Collection Inspector (5), revealing the joints with that collection. Joints are color-coded to speed visual verification: green for moving parts, purple for base parts, grey for other static parts. Clicking on a part within (4) brings up a menu of possible base parts for that part; selecting one adds the resulting joint to the currently-open collection. Finally, (5) contains a button users can click to request suggestions of joints to add to this collection. The system delivers those in the Suggestions Panel (6); the user can individually accept or reject any of these suggestions.

Authoring & Applying Rules Once the user has added joints to a collection, they can begin annotating those joints using MAPs. The user can open the Rule and Motion Editor (Figure 7) by clicking on the tab by the same name near the upper-left corner. The Rule List (1) shows the rules the user has added to this collection; these are the rules available to be applied to joints and considered for automatic suggestions. The Rule Editor (2) is a text editor which shows and allows editing of rule source code. The Motion Viewer (3) shows the joints in the collection as interactive 3D tiles. The

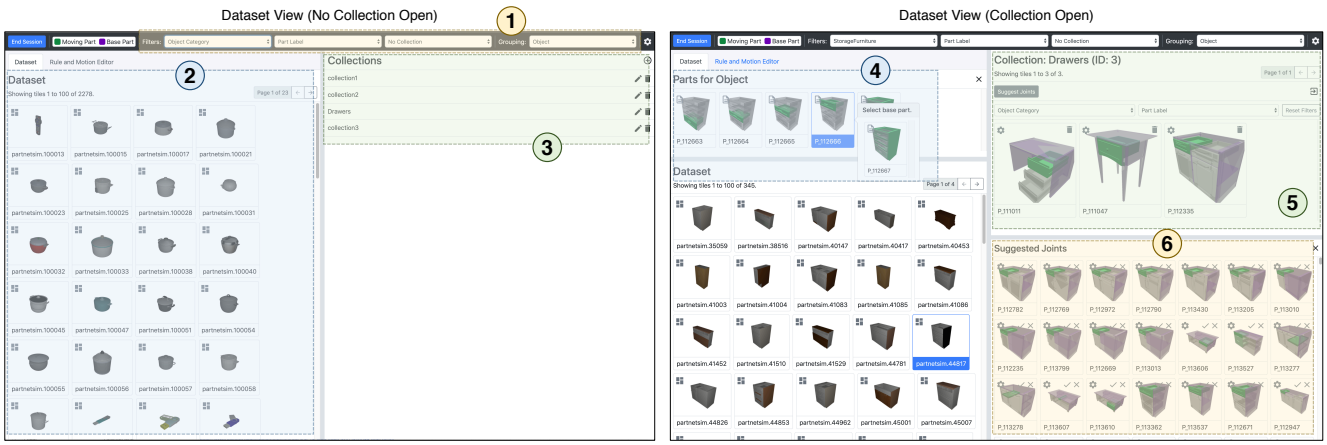


Figure 6. Our interface for building related joint collections. (1) Users can filter, group, and sort to navigate the dataset (2) more quickly. Users create collections (3) to which they add related joints (4, 5). Users can also request automatic suggestions of new joints to add (6).

Figure 7. Our interface for authoring and applying rules. Users can create new rules or import existing rules from a library (1). Rules are editable Python programs (2). Users can visualize the effect of rules on multiple joints simultaneously (3).

user can click and drag on these joint tiles to manipulate the camera; this manipulation propagates to all joints in the collection, keeping their viewpoints synchronized. With this interface, the user can create rules, edit rules, and request suggestions for rules to apply to joints in a collection.

6. Learning-based Suggestions

Here we describe how suggestions are computed using models trained on user feedback. The main challenge is the limited training data, as the models must provide useful suggestions without much user input. Grouping joints into collections enables us to train separate learners per group.

Joint Suggestions Each collection trains a function f_{joint} for suggesting new joints to add to the collection. f_{joint} takes as input a d-dimensional feature representation of a joint (see below) and returns a non-negative scalar score indicating the probability that the joint belongs to the collection. When the user requests suggestions, the system evaluates f_{joint} on all joints in the Dataset Panel, ranks them by their predicted score, and displays them in the Suggestion Panel.

Rule Suggestions Each collection trains a function f_{axis} for suggesting axis rules for joints. f_{axis} takes as input a d-dimensional joint feature vector, a 3-dimensional axis direction, a 3-dimensional axis origin, and a binary indicator of motion type (translation vs. rotation), and returns a scalar score indicating the probability that the axis is a correct annotation for the given joint. When the user requests an axis suggestion for a joint, the system executes all axis rules in the collection on that joint, evaluates f_{axis} on their outputs, and returns the one with the highest score.

Range rule suggestion works analogously to axis rule suggestion. Each collection trains a function f_{range} for suggesting range rules. This function expects the same inputs as f_{range}, plus a 2-dimensional vector encoding the lower and upper bounds of the motion range and a 3-dimensional reference vector for these bounds. As with axis suggestion, the system executes all available range rules on a joint and returns the one with the highest score.

Joint Feature Representation The first input to each of the above learned functions f is a feature descriptor of a joint. We train a joint autoencoder whose bottleneck layer becomes our feature representation. This representation is trained without supervision, prior to any user annotations; it just requires part-segmented 3D models from which to construct candidate training joints. We use a point cloud autoencoder [17]; Figure 8 illustrates our point cloud representation. In addition to position, each point also has (a)

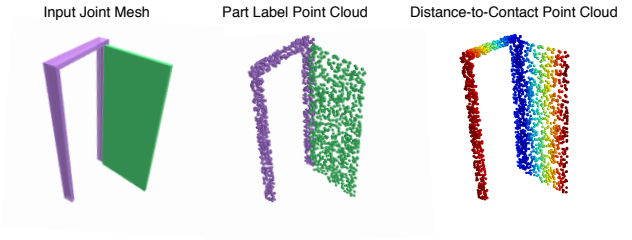


Figure 8. The point cloud representation we use for pre-training a joint feature representation. Each point has a base part vs. moving part label and a distance to the nearest point in the other part.

a binary label indicating whether the point is sampled from the base part or the moving part and (b) a scalar value indicating the distance to the closest point on the other part. This second feature communicates which points are closest to the moving/base contact region.In the supplemental, we analyze the structure of the learned joint feature space.

Learning Model We explored two options for learning each f from very few labeled examples: a *discriminative* approach using a random forest binary classifier (output score = classifier confidence) and a *generative* approach using a Gaussian kernel density estimator (output score = probability density). In both cases, we concatenate all inputs into a single vector that is fed to the model.

The generative approach works best with very few examples but does not support negative examples (it estimates the distribution from which positive examples are drawn). Thus, it does not respond to negative feedback. The discriminative approach requires more examples to produce useful suggestions and requires both positive and negative examples. Thus, we use a hybrid approach. When *both positive & negative examples* are available, we train a discriminative model. With *positive examples only*, we train a generative model. With *negative examples only*, we train a generative model and return the *negative* probability density of the input point as the scalar score (i.e. prefer lower-probability points).

Training Examples For joint suggestions, positive examples are (a) joints that the user adds to a collection and (b) suggested joints that the user accepts. Negative examples are suggested joints that the user rejects. For rule suggestions, positive examples are (a) rules that the user assigns to joints and (b) suggested rules that the user accepts. Negative examples are suggested rules that the user rejects.

Timing Joint suggestion takes less than 1 second for large number of joints; rule suggestion takes less than 4 seconds for reasonably large number of joints and rules (refer to supplemental for details). These runtimes are sufficiently fast to support interactive usage.

			Rules			
			Axis		*Range*	
Time	**Joints**	**Collections**	**New**	**Lib**	**New**	**Lib**
174 mins	1170	20	2	16	10	4

Table 1. Using our system, an expert user was able to annotate over 1,000 joints in under three hours. They used 20 collections and 32 rules (18 axis, 14 range). For axis annotation, library rules mostly sufficed; motion ranges required more custom rules.

7. Results & Evaluation

We evaluate our system both with an expert user and with new users previously unfamiliar with MAPs.

Implementation Details We implemented our motion annotation system as a web application (React + WebGL frontend; node.js + Python backend). For our experiments, we ran this implementation on AWS EC2 instances with 8 Xeon E5-2686 (2.30GHz) with 32 GB RAM.

Data We use the PartNet-Mobility dataset [27], the current state-of-the-art dataset for kinematically-articulated 3D shapes ($14,068$ annotated articulated joints, $2,346$ objects, 46 object categories). We choose a dataset with existing kinematic motion annotations so that we have ground-truth annotations to which to compare the ones produced by our system. This dataset was extended from PartNet, which provides hierarchical part segmentations. We convert these part hierarchies into two-level part + subpart segmentations by (a) using the motion part annotations from PartNet-Mobility as top-level parts and (b) using the the lowest level of the hierarchy provided by PartNet-Mobility as subparts.

Re-annotation Study First, an expert user (one of the authors) re-annotated objects from the PartNet-Mobility dataset. Table 1 shows some statistics from this re-annotation effort. Our expert annotated 1170 joints in 174 minutes, an average rate of 6.72s joint/minute.

Figure 9 shows some collections created by our expert user as well as statistics of all collections. Joints from the same object category often have a high degree of shape similarity, so that they can be grouped together into one collection. Within each collection, typically a few rules suffices to annotate a large number of joints.

Figure 10 analyzes our expert's rule usage. Most collections require few rules to annotate. There are some "custom" rules created for only one collection, and some general-purpose rules used across collections. The most-used axis rule was putting the world-up axis through the center of the moving part (mostly due to keyboard keys, of which there are many in PartNet-Mobility).

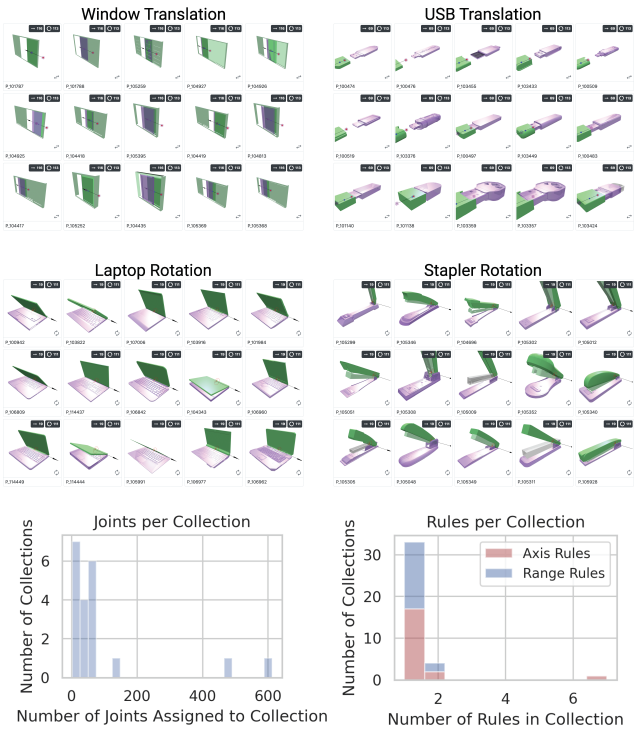


Figure 9. A sample of the joint collections created by our expert user, along with representative joints for each one, plus histograms of the number of joints per collection and number of rules used per collection. See the supplemental material for more.

User Study We also conducted a user study to gauge the performance of our system when operated by other users.

Experimental task. Participants were tasked with annotating articulated joints from a subset of PartNet-Mobility. They were instructed to try to annotate as many joints as possible in the allotted time (one hour). We chose a subset of objects to annotate that covered a range of motion types, to exercise our system's capabilities: hinge motions (16 Door + 64 StorageFurniture objects), swiveling motions (17 Knife + 12 USB + 10 Fan objects), and retracting motions (9 Knife + 11 USB objects). The total number of objects (139) was chosen such that participants would have enough work for at least an hour (based on the performance of our expert user) but would not feel overwhelmed.

Experimental conditions. Our study had two experimental conditions. In the *Ours* condition, participants used our system to annotate joints. In the *Manual* condition, participants used an alternative interface for manually annotating objects one at a time by directly adjusting axis, center, and range of motion values (see supplement for a video).

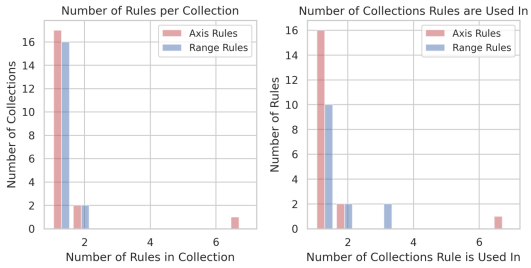
Participants. We recruited 16 participants (8 per condition). Participants were university students and staff with Python and computer graphics experience. Participants first watched a short tutorial video demonstrating their assigned interface. They then practiced using this interface on a different subset of PartNet-Mobility. Finally, they were given one hour to perform annotations. Afterwards, participants filled out a brief exit survey.

Number of annotations. Figure 11 plots the total number of joints annotated per minute between the two conditions, as well the number which are also annotated in the ground truth PartNet-Mobility data. This number is smaller than the total number, as users may annotate joints that were not annotated in PartNet-Mobility. This can happen due to an ambiguous choice of base part (e.g. does the cap of a USB stick move with respect to the stick, or does the stick move with respect to the cap) or due to the participant interpreting the joint differently than the original PartNet-Mobility annotator (e.g. viewing a vertically-opening cabinet door as a cover that slides up and down). Participants using our system annotated more than 1.5x times as many joints than those using the manual baseline. Dotted references lines shows the annotation rates of our expert user on the same set of joints. There is a learning curve associated with our system, such that more familiarity leads to significant performance increase. In contrast, the expert only performs slightly better than the participants with the manual system, suggesting that this interface has limited peak performance regardless of experience or familiarity.

Annotation accuracy. We evaluate accuracy via three metrics: 1) *Axis direction error:* angular difference between the annotated and ground truth axis direction, in degrees. 2) *Center of rotation error:* distance from the annotated center of rotation to the line defined by the ground truth rotation axis + center, as percentage of the object's bounding box diagonal length. 3) *Range error:* the intersection over union (IoU) of the annotated and ground truth motion ranges. Figure 12 plots these metrics for both conditions. Annotations produced by both interfaces have less than 1° of axis direction error, indicating that this task is fairly easy. Annotating the center of rotation is more difficult, though easier in our system. Both conditions produced a wide variety of motion range errors. This is not surprising, as motion range is more ambiguous (e.g., how far does a door swing open?).

Suggestions. 6/8 participants used joint suggestions; 5/8 participants used rule suggestions. Acceptance rates were low, as the user typically accepts the few good suggestions for their use case and then rejects the rest. The supplement provides a more detailed breakdown of suggestion usage.

Qualitative Feedback. Participants who used our system found the overall design of the interface intuitive, including the visualizations of joints and motions. Many commented positively on the joint suggestions, noting that the suggested joints are usually the ones they want. Participants were mixed on rule suggestions: some preferred manually assigning rules to joints, as the number of joints to annotate was not that large. Rule suggestions may become more valuable as the size of the dataset increases. Participants




```python
# most-used axis rule (used for 473 joints)
def func(joint):
    origin = joint.moving_part.center
    axis = [0, 1, 0]
    return axis, origin

# most-used range rule (used for 517 joints)
def func(joint, axis, origin):
    ref = joint.moving_part.center
    current_pose = 0
    length, start, end = get_length_along_dir(
        joint.moving_part, axis)
    min_range = -0.5 * length
    max_range = 0
    return ref, current_pose, min_range, max_range
```

```python
# least-used axis rule (used for 6 joints)
def func(joint):
    origin = joint.moving_part.axis2.bottom
    axis = [1, 0, 0]
    return axis, origin

# least-used range rule (used for 11 joints)
def func(joint, axis, origin):
    ref = joint.moving_part.center
    current_pose = 0
    min_range = 0
    max_range = 1
    return ref, current_pose, min_range, \
        max_range
```

Figure 10. Analyzing the rules used by our expert using histograms of the number of joints to which each rule applies and the number of collections in which each rule is used. We also highlight two frequently-used rules and two rarely-used rules.

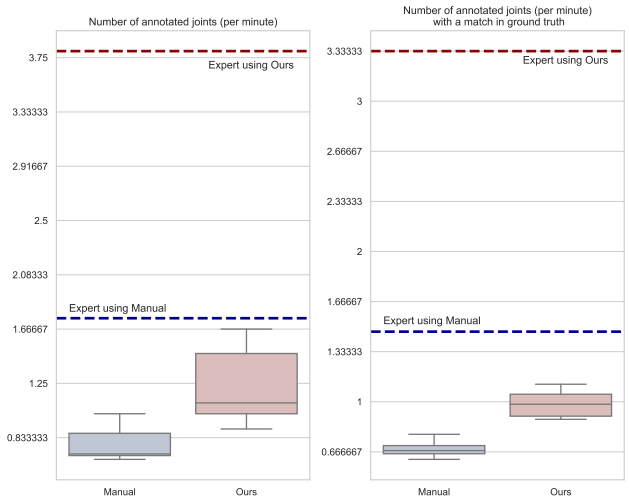


Figure 11. Box-and-whisker plots for the annotation rates of participants in our user study (joints/minute). We report the total number of joints annotated as well as the number with a matching annotation in the PartNet-Mobility data. The performance of our expert user with each interface is shown using dashed lines.

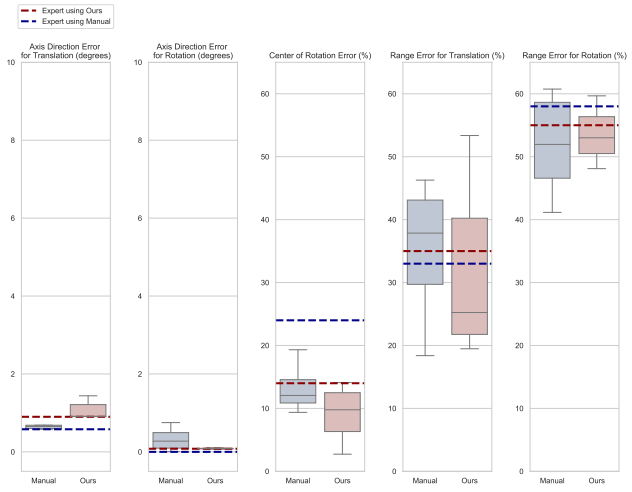


Figure 12. Box-and-whiskers plots for annotation errors made by participants in our user study. The performance of our expert user with each interface is shown using dashed lines.

were split on the ease-of-use of writing rules. Half felt that the DSL was well-designed and easy-to-learn, and that the rule library was well-organized and useful. The other half felt that mastering this system in an hour was too tall an order. One such participant felt confident they could become an expert user given a few more hours.

8. Conclusion & Future Work

We presented Motion Annotation Programs, a new approach for scalably annotating kinematic motions in 3D shape collections. We designed a DSL for authoring MAPs, an interface for applying them to collections of joints and verifying the results, and learning-based suggestions for further speeding up annotation. We evaluated the approach by reproducing motion annotations from PartNet-Mobility and by showing that previously-untrained users could annotate 1.5x faster than manual annotation using our system.

We could combine our approach with the ease-of-use by synthesizing rules from example manual annotations. In many cases, this would reduce to finding a library rule that agrees with the examples. In general, one could synthesize a new rule by searching a reduced space of possible programs with a constraint-based program synthesizer [20].

Our work focused on making a single expert annotator as efficient as possible. One could also explore whether multiple experts can work together to be even more efficient.

Finally, annotation via expert-authored rules could be applied to segmenting 3D scans, constructing hierarchical object decompositions, and more. Our work points the way toward a new "3D data programming" that will help scale 3D data annotation to the future demands of AI systems.

Acknowledgments

We thank the anonymous reviewers for their helpful suggestions; we also thank the participants in our user study for volunteering their time. Michael Cosgrove, Grishka Barboy, and Hameed Abdul-Rashid helped with the development of an earlier version of our system. Angel X. Chang is supported by the Canada CIFAR AI Chair program. Manolis Savva is supported by an NSERC Discovery Grant.

References

[1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[2] S. Branson, P. Perona, and S. Belongie. Strong supervision from weak annotation: Interactive training of deformable part models. In *Proc. of International Conference on Computer Vision (ICCV)*, Barcelona, 2011. 2

[3] S. Branson, G. Van Horn, C. Wah, P. Perona, and S. Belongie. The ignorant led by the blind: A hybrid human–machine vision system for fine-grained categorization. *IJCV*, 108(1-2):3–29, 2014. 2

[4] CGTrader. CGTrader - 3D Models for VR / AR and CG Projects. https://www.cgtrader.com/, 2020. Accessed: 2020-05-22. 1

[5] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University, 2015. 1, 2

[6] X. Chen, A. Golovinskiy, and T. Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), Aug. 2009. 2

[7] X. Chen, A. Saparov, B. Pang, and T. Funkhouser. Schelling points on 3D surface meshes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, Aug. 2012. 2

[8] J. A. Fails and D. R. Olsen, Jr. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 39–45, 2003. 2

[9] J. Heer. Agency plus automation: Designing artificial intelligence into interactive systems. *Proceedings of the National Academy of Sciences*, 116(6):1844–1850, 2019. 2

[10] R. Hu, W. Li, O. Van Kaick, A. Shamir, H. Zhang, and H. Huang. Learning to predict part mobility from a single static snapshot. *ACM Transactions on Graphics (TOG), Proc. SIGGRAPH Asia*, 36(6):227, 2017. 2

[11] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv:1712.05474*, 2017. 1

[12] H. Li, G. Wan, H. Li, A. Sharf, K. Xu, and B. Chen. Mobility fitting using 4d ransac. *Comput. Graph. Forum*, 2016. 2

[13] T. Luo, K. Mo, Z. Huang, J. Xu, S. Hu, L. Wang, and H. Su. Learning to group: A bottom-up framework for 3d part discovery in unseen categories. In *International Conference on Learning Representations*, 2020. 2

[14] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[15] D. T. Nguyen, B.-S. Hua, L.-F. Yu, and S.-K. Yeung. A robust 3d-2d interactive tool for scene segmentation and annotation. In *TVCG*, 2017. 2

[16] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. VirtualHome: Simulating household activities via programs. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8494–8502, 2018. 1

[17] C. Qi. pointnet-autoencoder. https://github.com/charlesq34/pointnet-autoencoder, 2020. Accessed: 2020-05-22. 5

[18] O. Russakovsky, L.-J. Li, and L. Fei-Fei. Best of both worlds: human-machine collaboration for object annotation. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[19] A. Sharf, H. Huang, C. Liang, J. Zhang, B. Chen, and M. Gong. Mobility-trees for indoor scenes manipulation. In *Computer Graphics Forum*, volume 33, pages 2–14. Wiley Online Library, 2014. 2

[20] A. Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California at Berkeley, USA, 2008. 8

[21] Turbosquid. Turboquid: 3D Models for Professionals. https://www.turbosquid.com/, 2020. Accessed: 2020-05-22. 1

[22] Y. Urakami, A. Hodgkinson, C. Carlin, R. Leu, L. Rigazio, and P. Abbeel. DoorGym: A scalable door opening environment and baseline agent. *arXiv preprint arXiv:1908.01887*, 2019. 1

[23] A. Vezhnevets, J. M. Buhmann, and V. Ferrari. Active learning for semantic segmentation with expected change. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3162–3169, 2012. 2

[24] S. Vijayanarasimhan and K. Grauman. Multi-level active prediction of useful image annotations for recognition. In *NIPS*, pages 1705–1712, 2008. 2

[25] X. Wang, B. Zhou, Y. Shi, X. Chen, Q. Zhao, and K. Xu. Shape2Motion: Joint analysis of motion parts and attributes from 3D shapes. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8876–8884, 2019. 1, 2

[26] F. Xia, C. Li, K. Chen, W. B. Shen, R. Martín-Martín, N. Hirose, A. R. Zamir, L. Fei-Fei, and S. Savarese. Gibson env v2: Embodied simulation environments for interactive navigation. Technical report, Stanford University, 6 2019. 1

[27] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, L. Yi, A. X. Chang, L. J. Guibas, and H. Su. SAPIEN: A simulated part-based interactive environment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1, 6

[28] W. Xu, J. Wang, K. Yin, K. Zhou, M. van de Panne, F. Chen, and B. Guo. Joint-aware manipulation of deformable models. In *SIGGRAPH*, 2009. 2

[29] C. Yan, D. Misra, A. Bennnett, A. Walsman, Y. Bisk, and Y. Artzi. CHALET: Cornell house agent learning environment. *arXiv:1801.07357*, 2018. 1

[30] Z. Yan, R. Hu, X. Yan, L. Chen, O. Van Kaick, H. Zhang, and H. Huang. RPM-Net: recurrent prediction of motion and parts from point cloud. *ACM Transactions on Graphics (TOG), Proc. SIGGRAPH Asia*, 38(6):240, 2019. 1, 2

[31] L. Yi, H. Huang, D. Liu, E. Kalogerakis, and L. G. Hao S and. Deep part induction from articulated object pairs. *ACM Transactions on Graphics (TOG), Proc. SIGGRAPH Asia*, 37(6):209, 2019. 2

[32] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016. 2