AutoURDF: Unsupervised Robot Modeling from Point Cloud Frames Using Cluster Registration

Supplementary Material

Overview. The supplementary material is structured into the following sections:

- Data Collection A: This section describes the synthetic data generation process and real-world datasets used for evaluating the proposed method;
- Method Details B: This section provides a detailed explanation of the topology inference algorithm, model architecture, and comparison between pose representations;
- Additional Experiments C: This section presents experiments analyzing the impact of varying frames, cameras, clusters, noise, and parameters, along with supplementary visualization results.

A. Data Collection

A.1. Synthetic Data

We simulate the robots and point cloud scanning process with Pybullet [9]. The robot is controlled by randomly sampled motor angle sequences, and the corresponding point cloud frames are captured during this process. In the simulation, we collect 5 video sequences per robot, with each video containing 10 frames of point cloud data. To simulate real-world conditions, random positional noise and per-point noise are added to the point cloud data. In the experimental results presented in the main text, we combine 20 camera views into a single frame point cloud to create dense point cloud. Figure 12 illustrates the process of generating a single-frame point cloud from three views of depth images.

To ensure the point cloud sequence captures sufficient motion information, we independently sample random targets within the motor angle limits for each motor. Additionally, if the robot detects a self-collision, the sequence is restarted with a new set of target motor angles. Randomly sampled data may include similar rotational motions, making it challenging to identify distinct parts. Our method can merge multiple random motion sequences, improving the kinematics inference accuracy.

A.2. Real-world Data

As shown in Figure 13, we conducted real-world experiments using 10 consecutive point cloud scans of a WX200 robot arm, controlling all five motors. The robot was operated through a ROS2 interface, following a randomly sampled motor angle sequence.

Point cloud data was collected using an iPhone camera and the iOS scanning application Scanverse [34]. A bounding box was applied to isolate the robot arm's point cloud from the environment. Despite the real scan data containing misaligned coordinates across time-steps and significant surface noise, our method directly processes the raw point cloud data, achieving accurate segmentation and URDF generation.

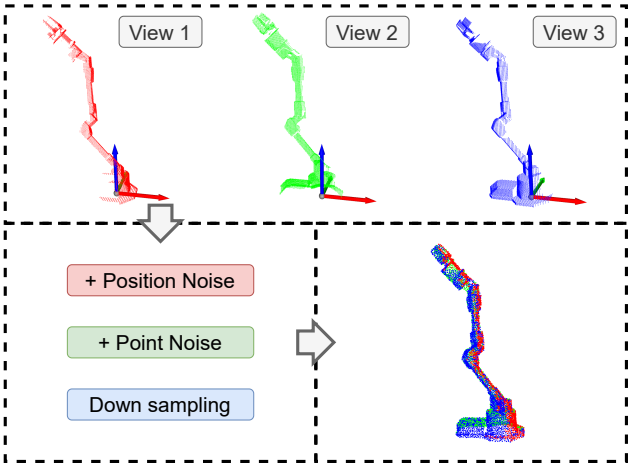

Figure 12. **Synthetic data collection.** One frame of synthetic data is collected by merging multi-view depth maps into a single-point cloud. Global coordinate noise and per-point noise are applied to simulate realistic conditions. This image shows an example of a point cloud created from three depth images.

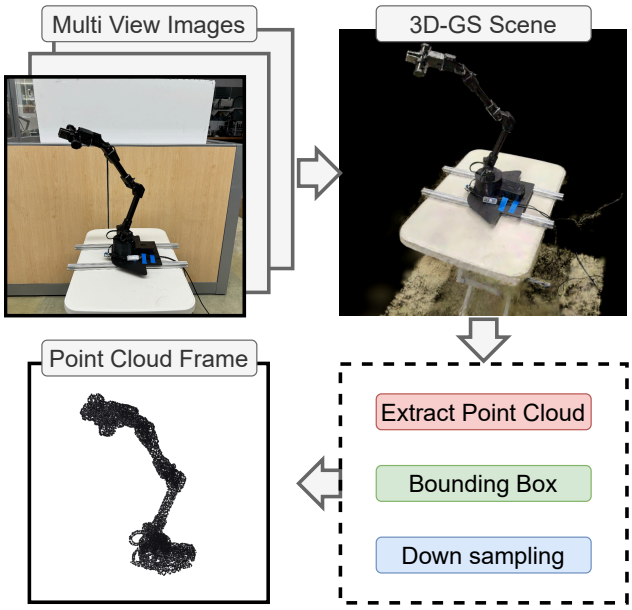

Figure 13. **Real-world data collection.** For each motion step, we capture a video of the robot arm, reconstruct the 3D scene using a 3D Gaussian splatting application [17, 34], and extract the point cloud. A bounding box removes background points, and downsampling standardizes the number of points.

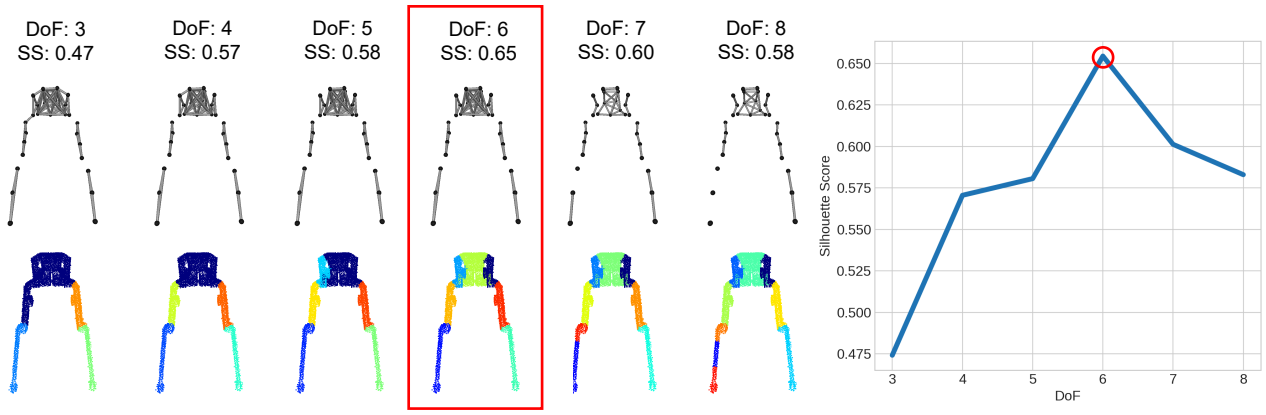

Figure 14. **Silhouette Score Method.** An example of using the Silhouette Score method to identify distinct moving parts and predict the degrees of freedom (DoF) for the Bolt bipedal robot, with a peak score at DoF = 6. This indicates that segmenting the point clusters into seven parts provides the optimal grouping.

B. Method Details

B.1. Silhouette Score Method for Part Segmentation

Figure 14 illustrates an example of using the Silhouette Score [46] method to identify the number of links. With DoF ranging from 3 to 8, the averaged Silhouette Score peaks at DoF = 6, which is the correct prediction for the Bolt robot. Given the number of groups for the segmentation algorithm, the Silhouette Score and Coefficient are calculated as equation 4 and equation 5.

$$\boldsymbol{SS}(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{4}$$

$$\boldsymbol{SC} = \max_k \frac{1}{S} \sum_i \boldsymbol{SS}(i) \tag{5}$$

In Equation 4, $\boldsymbol{SS}(i)$ denotes the Silhouette Score of the i-th node, which, in our case, is a point cluster. The term $a(i)$ represents the average distance of the i-th node to all other nodes within the same group, while $b(i)$ represents the average distance to nodes in the nearest group. The distance is calculated using Equation 3. $\boldsymbol{SC}(i)$ is the Silhouette Coefficient, which depends on the number of groups, k. S represents the total number of nodes. To determine the optimal number of groups for segmentation, we maximize the average Silhouette Score over all groups.

B.2. Topology Inference

As shown in Algorithm 2, topology inference is divided into three main stages to construct the body topology graph $\boldsymbol{\mathcal{G}}$. In the first stage, the algorithm identifies connected components from the segmentation graph G_{seg}, grouping clusters into link components $\{I_c\}$. For each component, it determines its neighboring components using the Minimum Spanning Tree G_{mst}, identifying clusters that are directly connected, and therefore the corresponding groups of direct connection. A dictionary is created for each link, containing a unique identifier (*Id*), a parent link initially set to *None*, and a list of connected link IDs. These dictionaries are stored in the list *Links* for further processing. In the second stage, the kinematic tree structure is derived by iteratively traversing the links. Starting with the root link, which is chosen as the first link sorted by ascending center movements, child links are identified by excluding their parent link from the list of connected links. The algorithm updates the parent-child relationships and adds the child links to the next layer for processing. This process continues layer by layer until all links are processed. In the final stage, the body topology graph $\boldsymbol{\mathcal{G}} = (I, E)$ is constructed, where I is the set of link IDs, and E consists of directed edges representing parent-child relationships.

B.3. Model Architecture

Figure 15 shows the architecture of the registration model. We use a shared neural network to predict incremental updates to the position and rotation of each point cluster. The input dimensions align with the pose representation of the point clusters. A sinusoidal positional encoder enhances the network's ability to capture spatial relationships and patterns[48]. The model includes a fully connected encoder and separate decoders for rotation and position. The network's output is added to the input coordinates to learn incremental updates directly. We use PyTorch [38] to implement the model and optimization is performed using the Adam optimizer[18].

The same model architecture is applied to both the Step Model and the Anchor Model, with learning rates of 0.0001 and 0.00005, respectively. The training loss is calculated using the L1 Chamfer distance between the transformed point cloud and the ground truth. An early stopping mechanism halts optimization if the point cloud error does not decrease after a set number of steps, ensuring efficient training.

B.4. Rotation Representation

To efficiently and robustly learn the rotation of clusters To optimize the learning of inter-frame cluster rotations in our registration model, we investigate the efficacy of three rotation representations: Euler Angles, Quaternions, and 6D Rotation representations [54]. We conduct extensive experiments across ten diverse sequences, comparing these representations regarding their training stability and convergence properties. As shown in 16, both Quaternions

Algorithm 2 Topology Inference

Input: Segmentation $\mathcal{G}_{seg}$, MST $\mathcal{G}_{mst}$
Output: Body Topology $\mathcal{G}$
Initialize: $Links \leftarrow$ EmptyList
Initialize: $\mathcal{I}_c$ = connected_components($\mathcal{G}_{seg}$)
where $\mathcal{I}_c$ is a list of cluster indices $\{I_c\}$
// 1. Construct a list of link dictionaries
for $(I_l, \{I_c\})$ **in** enumerate($\mathcal{I}_c$)
  Find connected clusters
  for I_c **in** $\{I_c\}$
    $\{I_{\text{c\_connected}}\} \leftarrow \mathcal{G}_{\text{mst}}$.neighbors($I_c$)
  end for
  Find connected links $\{I_{\text{l\_connected}}\}$ with $\{I_{\text{c\_connected}}\}$
  Build dictionary: Link = {
      Id: I_l;
      parent: None;
      connected_links: $\{I_{\text{l\_connected}}\}$}
  $Links$.append(Link)
end for
Sort $links$ by center movements in ascending order
// 2. Derive the kinematic tree
Initialize: $current\_layer$ = [$links$[0]] // root
repeat
  Initialize $next\_layer \leftarrow \emptyset$, $child\_set \leftarrow \emptyset$
  for Link$_{current}$ **in** $current\_layer$ **do**
    if Link$_{current}$ has parent **then**
      $child \leftarrow$ connected_links excluding parent
    else
      $child \leftarrow$ connected_links
    end if
    for Link$_{child}$ **in** $child$ **do**
      Update Link$_{child}$.parent $\leftarrow$ Link$_{current}$.Id
      Add Link$_{child}$ to $next\_layer$
    end for
    Update $child\_set \leftarrow child\_set \cup child$
  end for
  Update $current\_layer \leftarrow next\_layer$
until $child\_set = \emptyset$
// 3. Build $\mathcal{G}$ from $Links$
$\mathcal{G} = (I, E)$
where $I = \{\text{Link.Id}\}$, $E = \{(\text{Link.}parent, \text{Link.Id})\}$

and 6D Rotation representations demonstrate superior robustness as the angular step size increases from 4° to 10°. The empirical results suggest that these continuous representations maintain consistent performance even under larger rotational variations, while Euler Angles show increased instability at higher angles. This aligns with previous findings regarding the advantages of continuous rotation representations in deep learning frameworks. Based on these results, our implementation supports both Quaternion and 6D Rotation representations, with Quaternion as the default configuration.

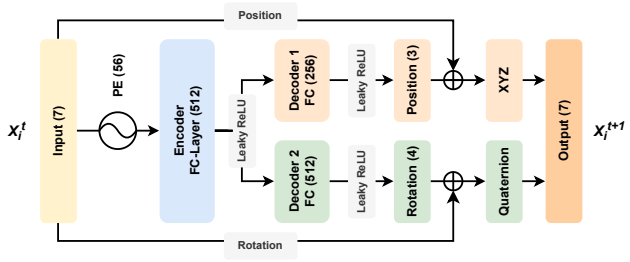

Figure 15. **Registration Model Architecture.** We developed a lightweight neural network for point cluster registration, employing the same model architecture for both the Step Model and Anchor Model.

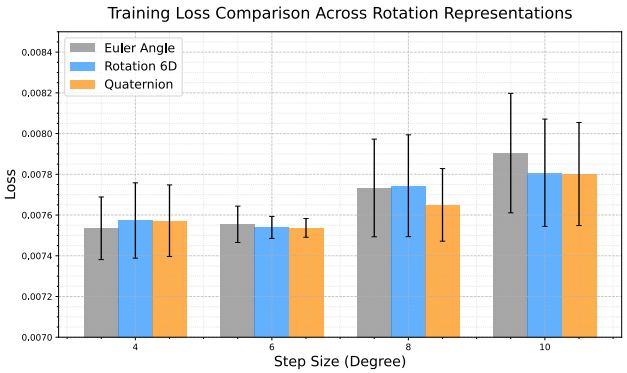

Figure 16. **Training Loss Comparison of Rotation Representations Across Step Sizes.**

C. Additional Results

C.1. Experiment on Number of Input Sequences

The Multi-Sequence Merging Experiment presented in our main text compares the performance of our method using a single sequence of point cloud frames against five sequences of point cloud frames. The results indicate that Our method achieves improved performance in 7 out of 8 robots for both repose evaluation and joint distance evaluation while demonstrating improvements across all robots for joint angle evaluation. Additionally, Figure 17 provides a qualitative comparison. With data from five sequences, our method generates segmentations with higher distinction, creating more edges within the correct group of point clusters, as exemplified in the UR5 robot. Furthermore, it achieves higher accuracy in joint estimation and reposed point cloud generation.

With the starting motor configurations aligned across different sequences, our method merges multiple sequences by registering them to the same set of point clusters and averaging the resulting motion correlation matrices. We perform the repose comparison by repeating the synthetic point cloud collection process (Figure 12) using a new set of random motor configurations applied to both the predicted and ground-truth URDFs.

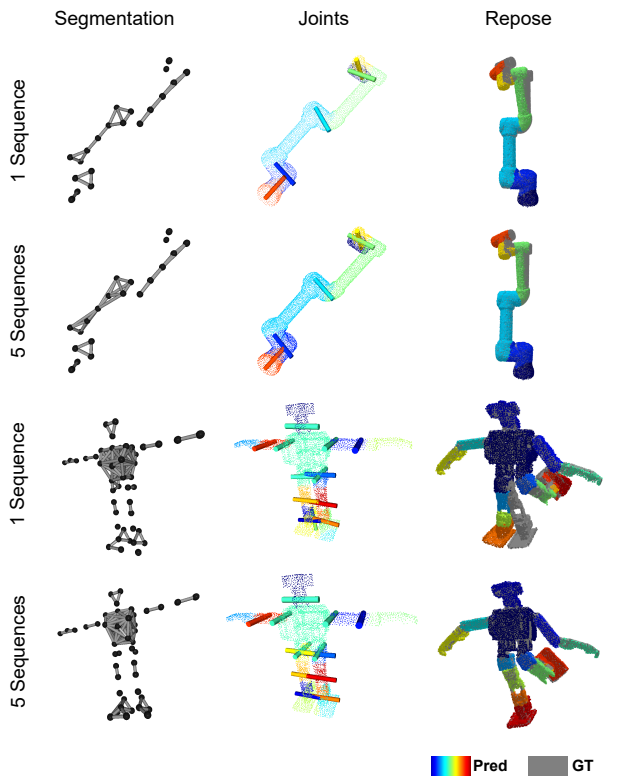

Figure 17. **Qualitative Comparison on Different Number of Input Sequences.**

C.2. Experiment on Number of Clusters

To investigate the sensitivity of our method to the quality and refinement level of initial clusters, we conduct extensive experiments across robots with varying structural complexity. We evaluate each robot configuration across eleven different cluster quantities, comparing against the *Reart* baseline [25]. As shown in 18, we observe that while optimal performance occurs at specific cluster number ranges, our method consistently outperforms the baseline (indicated by blue bars) across a wide range of parameter settings, demonstrating the method's stability across different robot architectures.

Evaluation across three robot configurations (WX200, Solo, PhantomX) with varying structural complexity. Blue bars indicate performance superior to *Reart* [25], shown as the dashed line.

C.3. Experiment on Occlusion and Noise

To evaluate the robustness of our algorithm in real-world conditions, we tested it under varying levels of **noise and occlusion** by adding Gaussian noise and limiting the number of camera views (Fig. 19). The results show that our method outperforms *Reart* [25] in highly noisy and occluded scenarios, in terms of moving parts segmentation.

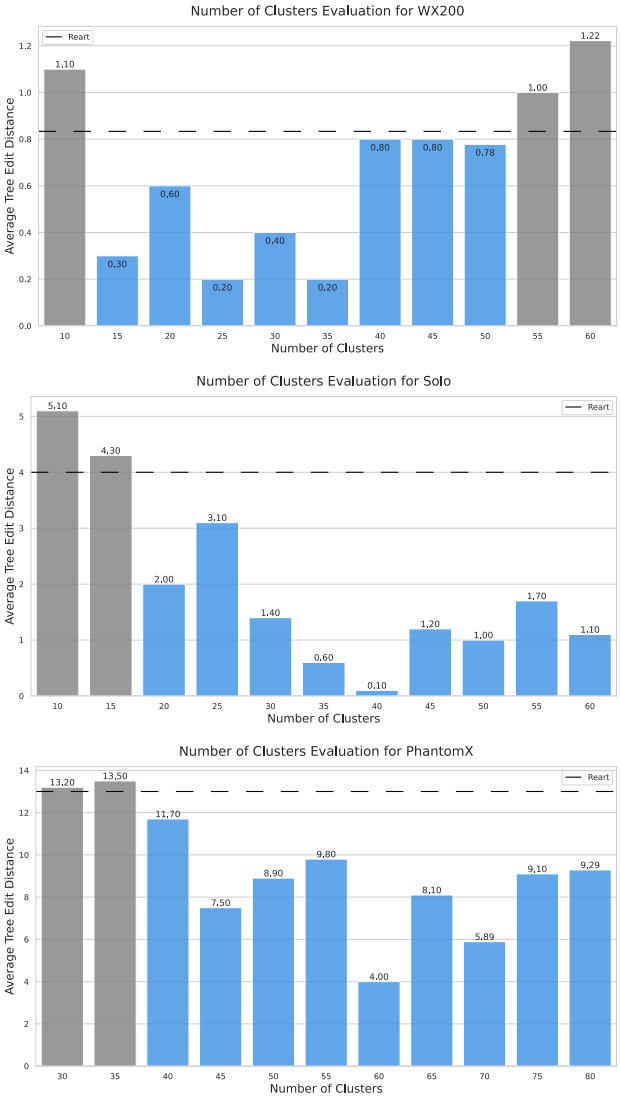

Figure 18. **Impact of Cluster Number on Tree Edit Distance.**

C.4. Experiment on *PartNet-Mobility* Dataset

We evaluate our AutoURDF framework on six common household articulated objects from the *PartNet-Mobility* dataset [32], each featuring one or two degrees of freedom: laptop, trashcan, toilet, dishwasher, faucet, and storage cabinet. As shown in 20, we initialize each object in an open configuration to facilitate distinct part clustering. To enhance the initial segmentation of planar components, we incorporate k-means clustering with normal information from the point cloud. The framework demonstrates robust performance in both segmentation and joint parameter estimation for objects with predominantly planar structures. While the cylindrical geometry of the faucet presents challenges for precise segmentation, the framework still maintains accurate joint axis prediction, highlighting its robustness to partial segmentation errors.

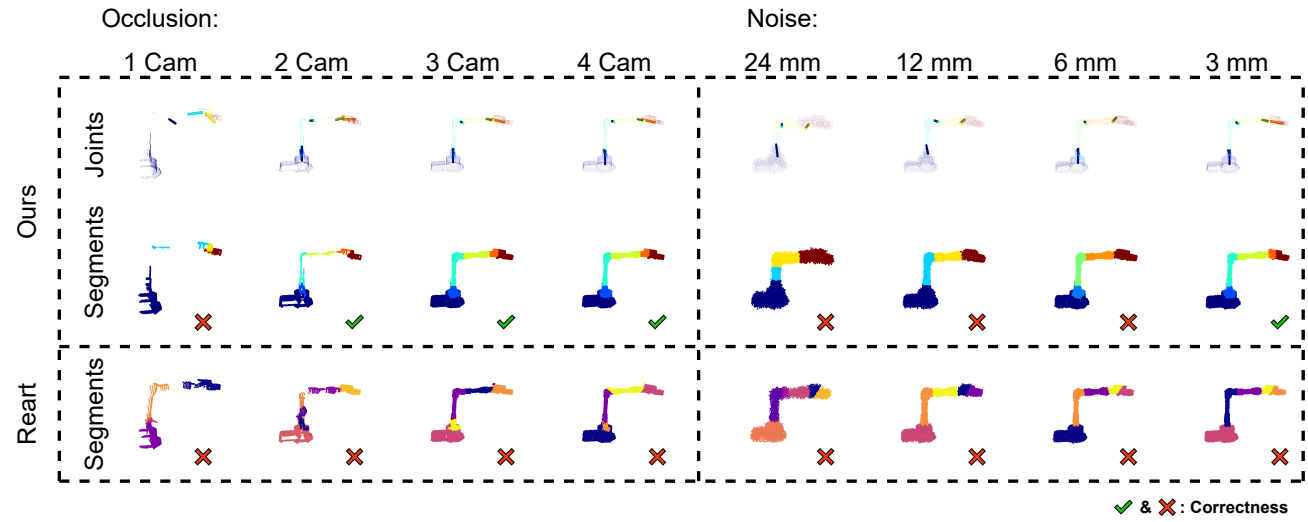

Figure 19. **Impact of completeness and noise of input point cloud.**

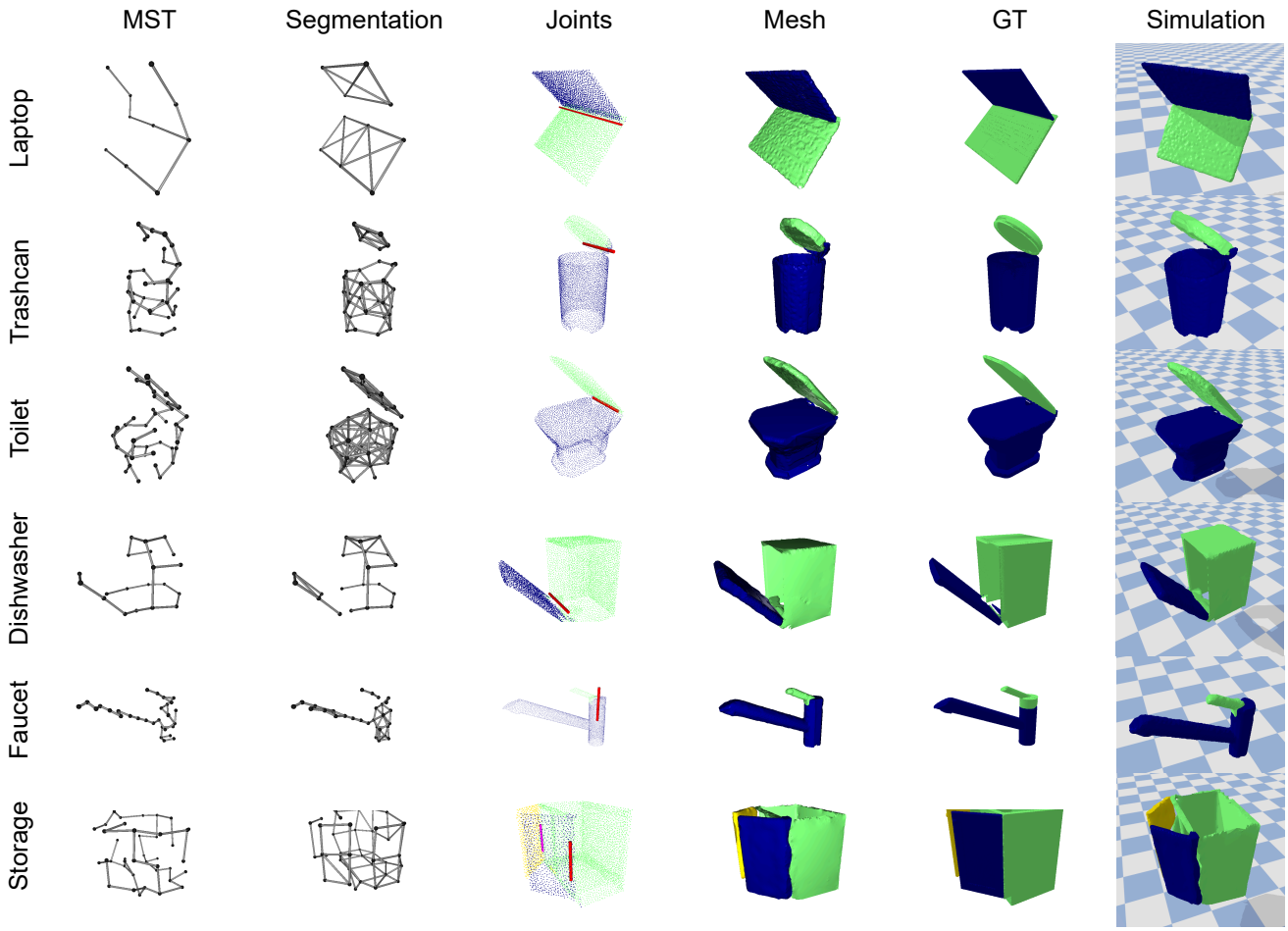

Figure 20. **Qualitative Results on *PartNet-Mobility* [32] Dataset.**

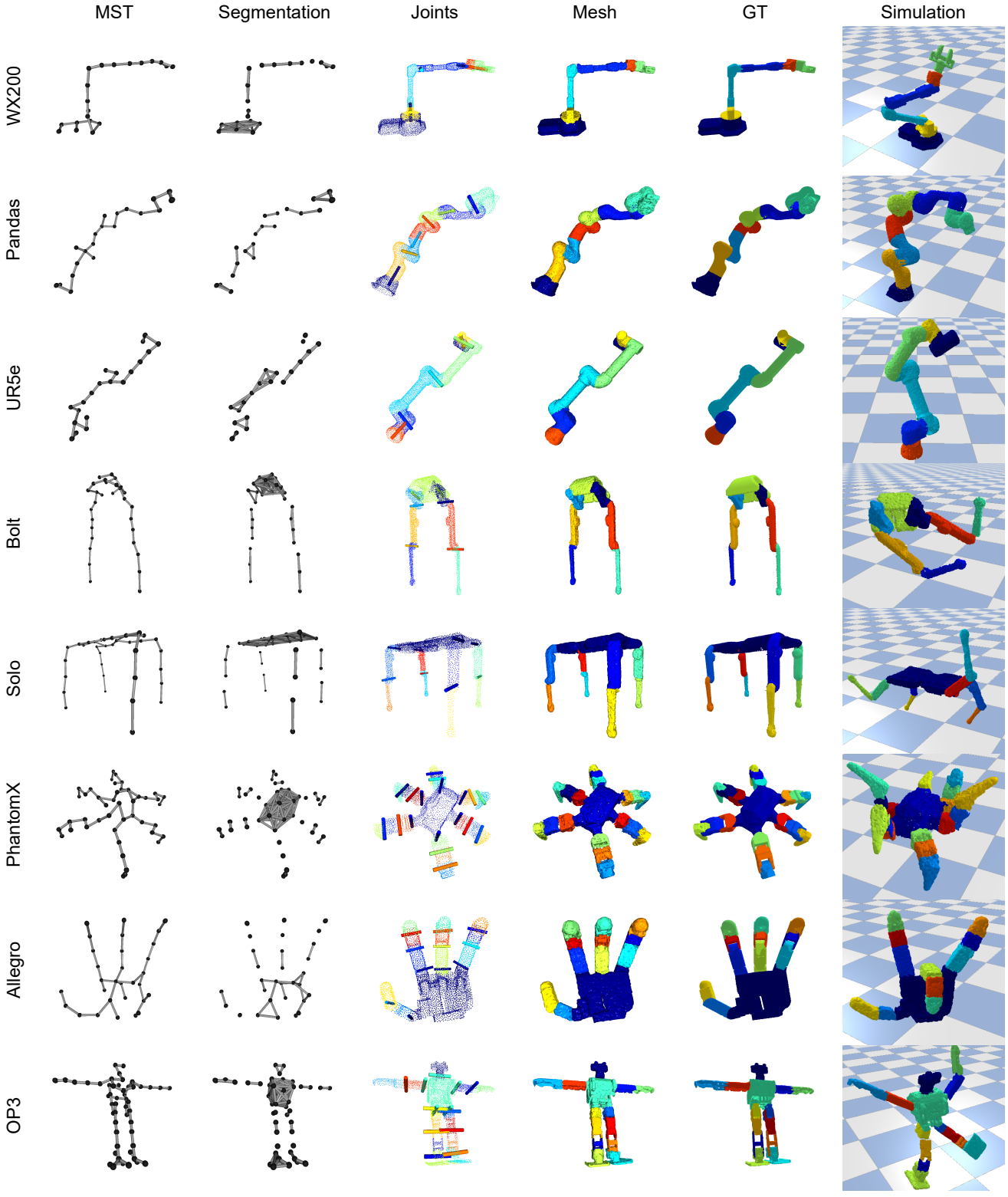

Figure 21. **Quantative Results on AutoURDF Dataset.**