1 Dataset Details

We create a synthetic dataset called **ArtImage** to evaluate OMADNet. We choose five categories(laptop, eyeglasses, dishwasher, drawer, scissors) from PartNet-Mobility[9] which is a public dataset that contains CAD models of articulated objects. We re-align the rest pose and direction of the objects in these categories(See Fig. 1). After that, we use Unity[1] to render depth images with random object location, joint state and camera viewpoint for each category. We only adopt viewpoints that are above the object horizontal plane, for articulated objects are usually viewed from above in daily life. This limitation also resolves symmetric problem on categories like scissors and makes different parts distinguishable.

On average, each category has about 50 instances and 200 depth images are generated for each instance. We choose 10% instances as unseen test objects for each category, while the other instances are used for training. On average, we have 10000 training images and 1000 testing images that contains unseen instances for each category. Because all the depth images are synthetic, we can easily get ground-truth annotations such as part segmentation masks, joint states and object poses.

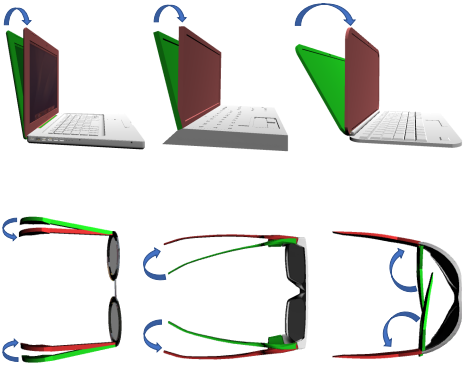

Figure 1: The examples of re-annotation for articulated object models in PartNet-Mobility[9]. We manually align the rest state for each category.

2 Implementation Details

The dimension of shape parameters $\boldsymbol{\beta}$ is 10, which means the shape function of each cateogry has 10 linear shape basis. Joint function $J(\boldsymbol{\beta})$ consists of two-layer MLP with ReLu activation, the hidden dimension of MLP is 64. The total number of keypoints for each category is set to 24 by default(we set it to 32 for drawers, because this category has more part numbers).

2.1 OMAD-PriorNet

The input of OMAD-PriorNet is complete point cloud $\boldsymbol{X}'$ of articulated objects in canonical space, and the output are instance-specific parameters including shape parameters $\boldsymbol{\beta}$, keypoints $\boldsymbol{P}'$ and joint parameters $\boldsymbol{\Phi}'$ in canonical space. For OMAD-PriorNet, we perform surface point sampling on complete object mesh to generate point cloud which has 2048

points per object. The point cloud will be placed in to OMAD space and serve as the input of OMAD-PriorNet. During training, we will perform random scale augmentation. Specifically, we multiply a random scale factor $s \in [0.5, 1.5]$ on x, y, z direction of the input point cloud in OMAD space separately. We use SGD optimizer with a initial learning rate 0.001 with momentum 0.9. We also adopt cosine learning rate decay[6] during training.

2.2 OMADNet

For OMADNet, the channel number of dense feature is 128. We random sample 1024 points from point cloud as input of OMADNet. We use SGD optimizer with a initial learning rate 0.001 with momentum 0.9. We train OMADNet for 100 epochs on ArtImage training dataset. We also adopt cosine learning rate decay[6] during training. For optimization process, we use Adam[4] optimizer with learning rate 0.01. Generally, the optimization takes about 200 iterations to coverage.

2.3 A-NCSH[5]

For the re-training of A-NCSH[5], we modify their official code to adapt to our dataset. We randomly sample 1024 points from point cloud as input. The PointNet++[8] backbone of A-NCSH network is the same as ours in OMADNet. We re-train A-NCSH[5] for 100 epochs on our ArtImage training dataset. We use SGD optimizer with a initial learning rate 0.001 with momentum 0.9. We use step learning rate decay strategy with decay factor 0.7 at the 20-th, 40-th, 60-th, 80-th epoch, which is also adopted in [5]. We follow [5] for all the other hyper-parameters and training settings.

2.4 Siamese Network

For the Siamese Network used as baseline in the retrieval experiment, we use a two-layer MLP built upon PointNet++[5] backbone to predict a shape vector for each instance, which is similar to the shape branch of our OMADNet. The dimension of shape vector is the same as the dimension of shape parameters $\boldsymbol{\beta}$ used in our OMAD, which is 10 by default. We randomly sample 1024 points from point cloud generated from depth images as input. During training, we randomly construct positive sample pairs and negative sample pairs from the training set of ArtImage. Each positive sample pair consists of two depth images of the same instance with different poses and viewpoints. Each negative sample pair consists of two depth images of two different instances with different poses and viewpoints. We make sure that the number of positive samples and negative samples are balanced during training. We use contrastive loss defined in Equation 1 proposed by [3] to make the predicted shape vector remain consistent in the same instance and increase distances between different instances.

$$L\left(W, Y, \vec{X}_1, \vec{X}_2\right) = (1-Y)\frac{1}{2}\left(D_W\right)^2 + (Y)\frac{1}{2}\left\{\max\left(0, m - D_W\right)\right\}^2 \tag{1}$$

Here D_W denotes the Euclidean distance of the hidden vector of the two samples $\vec{X}_1$ and $\vec{X}_2$, Y is the label which indicates whether this sample pair is negative, and m is a hyper-parameter. We set $m = 1.0$ after grid search. During testing, we use the Euclidean distance between predicted shape vectors as similarity metric.

3 Additional Experiments

3.1 Effects of keypoint numbers

Table 1 shows the effects of keypoint numbers. We can see that the performance becomes worser on most categories when the keypoint number is either very small or very large. Intuitively, we believe that too few keypoints makes the prediction results more sensitive to outliers, which can harm the performance. When using too many keypoints, the keypoint target locations predicted by keypoint branch are less accurate. We believe that it is hard for the neural network to distinguish between different keypoints when the keypoints are too dense, which harms the refined results after optimization.

Table 1: The effects of keypoint numbers.

Category	#Keypoints per Part	Joint State	Joint Parameter	
		error ↓	angle error ↓	distance error ↓
Laptop	4	8.7°	**3.4°**	0.056
	8	4.3°	3.6°	0.034
	12	3.3°	3.7°	**0.033**
	16	**3.0°**	4.0°	0.034
Eyeglasses	4	5.1°, 5.7°	**3.8°, 4.2°**	0.062, 0.052
	8	**4.9°, 5.2°**	4.2°, 4.6°	0.050, 0.054
	12	5.2°, 5.3°	4.8°, 5.2°	**0.046, 0.044**
	16	6.0°, 6.0°	5.0°, 5.5°	0.054, 0.049
Dishwasher	4	10.6°	4.7°	0.119
	8	5.9°	4.8°	0.093
	12	**3.7°**	**4.6°**	0.091
	16	4.0°	4.7°	**0.082**
Scissors	4	5.2°	6.2°	0.071
	8	4.0°	**2.2°**	**0.044**
	12	**3.2°**	2.7°	0.050
	16	3.5°	2.3°	0.059
Drawer	4	0.13, 0.13, 0.10	3.9°, 3.9°, 3.9°	-
	8	**0.11, 0.11, 0.09**	**3.2°, 3.2°, 3.2°**	-
	12	0.11, 0.12, 0.10	4.7°, 4.7°, 4.7°	-
	16	0.11, 0.11, 0.10	4.7°, 4.7°, 4.7°	-

3.2 Effects of shape parameters $\boldsymbol{\beta}$

Table 2 shows the results of using different dimension number(also the number of linear shape basis) of shape parameters $\boldsymbol{\beta}$. Optimization-based estimator is used as default. We can see that the performance becomes worser on most categories when the dimension number of $\boldsymbol{\beta}$ is too small or too large. We believe that very small dimension number of $\boldsymbol{\beta}$ can limit the ability of modeling complex geometric variations, and large dimension number can raise the risk of overfitting. Thus, an appropriate dimension number of $\boldsymbol{\beta}$(i.e. 10) is necessary for good performance.

3.3 Results on Real Images

To evaluate the generalization ability of our method on real images, we collect a real RGB-D dataset on Franka Emika Panda robot arm. This dataset contains 25369 RGB-D images captured by RealSense D435 under different joint states of robot arm and camera viewpoints. We select 18834 images for training and 6535 images for testing. We calibrate the pose of the base part of robot arm for each camera viewpoint, and obtain robot joint states automatically. Since the joint parameters in canonical space can be obtained from the URDF model, the 6D

Table 2: The effects of dimension number of $\boldsymbol{\beta}$.

Category	$\lvert\boldsymbol{\beta}\rvert$	Joint State	Joint Parameter	
		error ↓	angle error ↓	distance error ↓
Laptop	5	3.8°	3.7°	0.037
	10	**3.3°**	3.7°	**0.033**
	15	4.3°	**3.5°**	0.038
Eyeglasses	5	5.0°, 5.8°	4.4°, 4.9°	0.048, 0.048
	10	**4.9°, 5.2°**	**4.2°, 4.6°**	**0.046, 0.044**
	15	5.4°, 5.3°	4.6°, 5.2°	0.059, 0.050
Dishwasher	5	4.9°	5.2°	0.088
	10	**3.7°**	**4.6°**	0.091
	15	6.2°	4.8°	**0.085**
Scissors	5	4.4°	2.9°	**0.045**
	10	**3.2°**	2.7°	0.050
	15	3.6°	**2.2°**	0.046
Drawer	5	0.11, 0.11, 0.10	**3.0°, 3.0°, 3.0°**	-
	10	**0.11, 0.11, 0.09**	3.2°, 3.2°, 3.2°	-
	15	0.12, 0.11, 0.10	3.4°, 3.4°, 3.4°	-

pose of each part of the robot arm in camera space can be calculated with base part pose and joint states. We use Unity[1] to imitate the parameters of the real physical camera and set the 6D pose of the virtual robot URDF model to be the same with the real robot. In this way, part segmentation masks of robot arm on real images can be generated from Unity, which are necessary for methods like A-NCSH[5].

We make some modifications for our method to adapt to instance-level articulated pose estimation task. We use part-level Farthest Point Sampling to generate keypoints which serve as the shape basis of OMAD. The total keypoint number is 84 (12 keypoints for each part). We sample 1024 points from the point cloud generated from the depth image as input. The pose estimation results on our real dataset are shown in Table 3 and Figure 2. We achieve robust and more accurate results on joint distance error compared to A-NCSH[5], and obtain comparable results on joint state error and joint angle error. Generally, the errors of the last few joints are larger due to pose calibration noise of the base part. This experiment indicates that our method has the ability to generalize to real depth images on instance-level task.

Table 3: Performance comparison for instance-level articulated pose estimation on our real image dataset. *OMAD(refine)* represents the refined results with our optimizer. All the six joints are revolute joints.

Method	Joint State	Joint Parameter	
	error(°) ↓	angle error(°) ↓	distance error(m) ↓
A-NCSH[5]	**6.4°**, 6.5°, **8.6°**, 6.6°, **8.8°, 11.6°**	**1.7°, 6.7°**, 6.7°, 7.3°, **6.2°, 8.0°**	**0.008**, 0.036, 0.056, 0.045, 0.083, 0.228
OMAD(refine)	9.9°, **6.2°**, 10.1°, **5.0°**, 14.8°, 16.4°	3.1°, 9.8°, **6.1°, 6.6°**, 6.5°, 17.0°	0.021, **0.010, 0.017, 0.021, 0.019, 0.028**

3.4 Part-based Results

We also report experiment results in Table 4 on part-based metrics such as rotation error and translation error. Our OMAD(refine) achieve some better results compared to A-NCSH[5] on eyeglasses and drawer. A-NCSH performs slightly better on other categories, because they adopt dense point-level optimization with RANSAC, while our method only uses sparse keypoints for optimization. However, our method achieves faster inference speed and better results on joint state error and joint parameter error(See the experiment results in main paper).

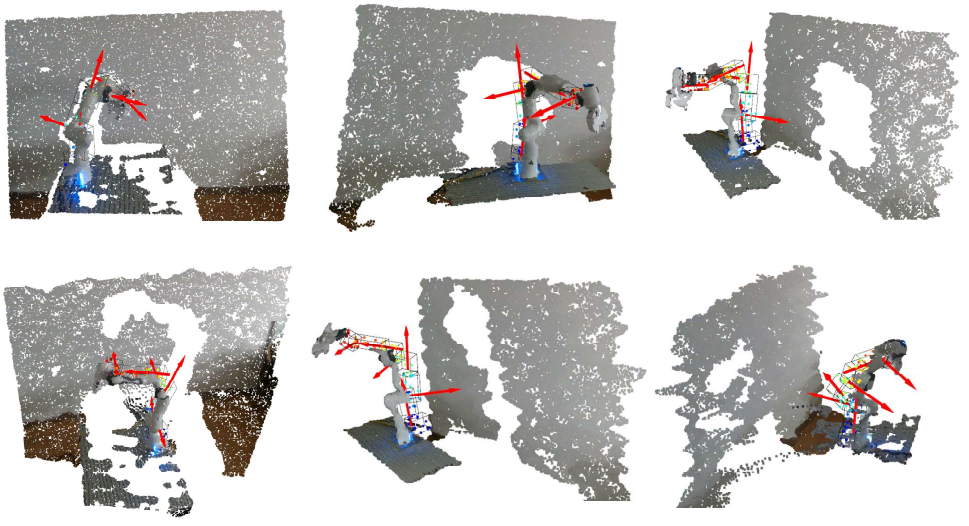

Figure 2: Visualization results of OMADNet on real depth images for instance-level articulated pose estimation task. Keypoint indexes are represented by colors. Joint parameters are represented by red arrows.

Table 4: Performance comparison on ArtImage dataset with part-based metrics. *OMAD(initial)* represents the initial joint state and joint parameters predicted by OMADNet, and *OMAD(refine)* represents the refined results with our optimizer.

Category	Method	Per-part Metrics	
		rotation error(°) ↓	translation error(m) ↓
Laptop	A-NCSH[5]	**5.3**, **3.4**	**0.054**, **0.043**
	OMAD(initial)	24.6, 25.4	0.218, 0.289
	OMAD(refine)	5.4, 4.3	0.062, 0.061
Eyeglasses	A-NCSH[5]	**3.7**, 22.3, 23.2	**0.049**, 0.313, 0.324
	OMAD(initial)	23.7, 32.0, 31.6	0.211, 0.387, 0.385
	OMAD(refine)	4.9, **7.5**, **7.5**	0.062, **0.103**, **0.104**
Dishwasher	A-NCSH[5]	**4.0**, **4.8**	**0.059**, **0.123**
	OMAD(initial)	19.3, 26.8	0.220, 0.339
	OMAD(refine)	6.0, 6.2	0.104, 0.142
Scissors	A-NCSH[5]	**2.0**, **2.9**	**0.035**, **0.025**
	OMAD(initial)	24.8, 25.0	0.291, 0.297
	OMAD(refine)	3.9, 3.4	0.048, 0.039
Drawer	A-NCSH[5]	**2.8**, **3.5**, **3.9**, **2.9**	**0.045**, 0.155, 0.157, **0.075**
	OMAD(initial)	19.1, 19.1, 19.1, 19.1	0.268, 0.342, 0.339, 0.339
	OMAD(refine)	4.4, 4.4, 4.4, 4.4	0.111, **0.144**, **0.143**, 0.115

4 OMAD

4.1 Deformation Function

In the main paper, deformation function is defined in Eq. 2 and Eq. 3:

$$\bar{\boldsymbol{p}}_i = G_k(\boldsymbol{\Phi}', \boldsymbol{\Theta})\bar{\boldsymbol{p}}'_i \tag{2}$$

$$G_k(\boldsymbol{\Phi}', \boldsymbol{\Theta}) = \prod_{j \in A(k)} F_j(\boldsymbol{\Phi}'_j, \boldsymbol{\Theta}_j) \tag{3}$$

Here $F_j(\mathbf{\Phi}'_j, \mathbf{\Theta}_j)$ is the relative 4×4 transformation matrix caused by the reference joint of part j. We will explain the definition of $F_j(\mathbf{\Phi}'_j, \mathbf{\Theta}_j)$ for three types of joints separately.

1. **Free Joint** $F_j(\mathbf{\Phi}'_j, \mathbf{\Theta}_j)$ is defined in Eq. 4:

$$F_j(\mathbf{\Phi}'_j, \mathbf{\Theta}_j) = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{4}$$

   where $\mathbf{R} \in \mathrm{SO}^3, \mathbf{t} \in \mathbb{R}^3$. Here $\{\mathbf{R}, \mathbf{t}\}$ are derived from joint state $\mathbf{\Theta}_j$.

2. **Revolute Joint** $F_j(\mathbf{\Phi}'_j, \mathbf{\Theta}_j)$ is the relative 4×4 transformation matrix that can make a vector $\boldsymbol{n}$ rotate around a joint axis $\mathbf{\Phi}'_j$ for $\theta \in \mathbf{\Theta}_j$ degrees.

3. **Prismatic Joint** $F_j(\mathbf{\Phi}'_j, \mathbf{\Theta}_j)$ is the relative 4×4 transformation matrix that can make a vector $\boldsymbol{n}$ translate distance $\theta \in \mathbf{\Theta}_j$ along joint direction $\boldsymbol{u} \in \mathbf{\Theta}_j$.

5 OMAD-PriorNet

Based on the method proposed by Fernandez *et al.*[2], we propose OMAD-PriorNet which can learn OMAD prior in an unsupervised manner. The input of OMAD-PriorNet is complete point cloud $\boldsymbol{X}'$ of articulated objects in canonical space, and the output are instance-specific parameters including shape parameters $\boldsymbol{\beta}$, keypoints $\boldsymbol{P}'$ and joint parameters $\mathbf{\Phi}'$ in canonical space. The parameters $\{\boldsymbol{B}, \mathbf{\Gamma}\}$ of shape function $S(\boldsymbol{\beta}; \boldsymbol{B})$ and joint function $J(\boldsymbol{\beta}; \Gamma)$ will be treated as shared network parameters of OMAD-PriorNet for each category, and they will be updated during training process. After the training, these parameters will be fixed as category-specific **OMAD prior**.

5.1 Structure Details

Our OMAD-PriorNet consists of two branches, namely *node branch* and *shape branch*. For categories with symmetry, we make some modifications to increase the quality of the learned keypoints.

1. **Node branch**. This branch will output potential unorderd keypoint candicates(also called nodes) which can be represented by $\mathcal{N}' = \{\mathbf{n}'_1, \mathbf{n}'_2 \ldots, \mathbf{n}'_M\}$, where M is the number of nodes. The number of nodes is equal to the number of order keypoints. Similar to [2], we also adopt Farthest Point Sampling to generate initial nodes on object point cloud, and use KNN and MLP to generate offsets relative to these initial nodes. Different from [2], we assign part labels for these nodes, and the Farthest Point Sampling process is performed at part-level.

2. **Shape branch.** This branch will output instance-specific shape parameters $\boldsymbol{\beta}$, and generate ordered keypoints $\boldsymbol{P}'$ and joint parameters $\mathbf{\Phi}'$ in canonical space based on shape function $\boldsymbol{P}' = S(\boldsymbol{\beta}; \boldsymbol{B})$ and joint function $\mathbf{\Phi}' = J(\boldsymbol{\beta}; \mathbf{\Gamma})$. This branch uses a PointNet[7]-like network to extract global feature, and use MLP to directly regress shape parameters $\boldsymbol{\beta}$. Different from [2], the shape basis $\boldsymbol{B}$ in shape function is part-level, which means the final generated ordered keypoints $\boldsymbol{P}'$ are also part-level.

3. **Symmetry.** In practive, we find many categories of articulated objects are symmetric. To better utilize such instance-wise symmetry, we adopt similar techniques in [2] which make all the shape basis $\boldsymbol{B}$ are symmetric about a certain plane. Specifically, we use such symmetry on categories including laptop, dishwasher and drawer.

5.2 Loss Function

We will introduce all the loss functions for the unsupervised training of OMAD-PriorNet.

1. **Chamfer loss.** We introduce a part-level chamfer loss L_{chf}.

$$L_{\text{chf}} = \frac{1}{K}\sum_{k=1}^{K}\Big(\sum_{\mathbf{n}'_i \in \mathcal{N}'_{(k)}} \min_{\mathbf{p}'_j \in \mathbf{P}'_{(k)}} \left\|\mathbf{n}'_i - \mathbf{p}'_j\right\|_2 + \sum_{\mathbf{p}'_j \in \mathbf{P}'_{(k)}} \min_{\mathbf{n}'_i \in \mathcal{N}'_{(k)}} \left\|\mathbf{n}'_i - \mathbf{p}'_j\right\|_2\Big) \tag{5}$$

where $\mathcal{N}'_{(k)}$ represents the node set that belong to part k, which are predicted by the node branch of OMAD-PriorNet. $\boldsymbol{P}'_{(k)}$ represents the ordered keypoint set that belong to part k, which are predicted by the shape branch of OMAD-PriorNet. This chamfer loss will make the distance between unordered nodes and order keypoints as small as possible, thus making the predicted keypoints have semantic consistency in category.

2. **Coverage loss.** The chamfer loss mentioned above can not make sure that the predicted keypoints can capture object shape geometry. Thus we need to introduce a part-level coverage loss L_{cov}.

$$L_{\text{cov}} = \frac{1}{K}\sum_{k=1}^{K} \text{SmoothL1}(bound(\mathcal{N}'_{(k)}), bound(\mathbf{X}'_{(k)})) \tag{6}$$

where $bound(\mathcal{N}'_{(k)})$ represents the corner points of 3D bounding box that covers all the unordered nodes $\mathcal{N}_{(k)'}$ for k-th part, and $bound(\mathbf{X}'_{(k)})$ represents the corner points of 3D bounding box that covers all the point cloud $\boldsymbol{X}'_{(k)}$ for k-th part. This coverage loss can make the predicted unordered nodes coverage each part of the articulated objects.

3. **Surface loss.** We want the predicted keypoints to be close to the object surface in order to capture object surface geometry. We introduce a part-level surface loss L_{surf}.

$$L_{\text{surf}} = \frac{1}{K}\sum_{k=1}^{K}\sum_{\mathbf{n}'_i \in \mathcal{N}'_{(k)}} \min_{\boldsymbol{x}'_j \in \mathbf{X}'} \left\|\mathbf{n}'_i - \boldsymbol{x}'_j\right\|_2 \tag{7}$$

where $\mathcal{N}'_{(k)}$ represents the unordered nodes that belong to k-th part, and $\boldsymbol{X}'$ represents complete object point cloud in canonical space.

4. **Regularization loss.** In order to make the predicted shape parameters $\boldsymbol{\beta}$ have better generalization ability, we introduce a regularization loss L_{reg}.

$$L_{\text{reg}} = \|\boldsymbol{\beta}\|_2^2 \tag{8}$$

5. **Separation loss.** In order to prevent the collapsing of keypoints, we introduce a part-level separation loss L_{sep}.

$$L_{\text{sep}} = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{|\mathcal{N}'_{(k)}|^2}\sum_{\boldsymbol{n}'_i\in\mathcal{N}'_{(k)}}\sum_{\boldsymbol{n}'_j\in\mathcal{N}'_{(k)},j\neq i}\max\left(0,\delta^2-\left\|\boldsymbol{n}'_i-\boldsymbol{n}'_j\right\|_2^2\right)\right) \tag{9}$$

where $\mathcal{N}'_{(k)}$ represents unordered nodes that belong to k-th part, $|\mathcal{N}'_{(k)}|$ represents the number of nodes that belong to k-th part and hyper-parameter σ is the distance threshold. When the distance between any two nodes is less than δ, the result will be punished. This separation loss will prevent multiple nodes collaps to the same location but not keep these nodes too far away from each other.

6. **Joint loss.** We introduce joint loss defiend in Eq. 10, Eq. 11 and Eq. 12 to supervise all the joint parameters $\boldsymbol{\Phi}' = [\boldsymbol{\phi}'_1,\ldots,\boldsymbol{\phi}'_K]$, $\boldsymbol{\phi}'_k = (\boldsymbol{u}'_k,\boldsymbol{q}'_k)$ in canonical space. Here $\boldsymbol{\phi}'_k$ represents the joint parameters of k-th joint, $\boldsymbol{u}'_k$ represents the axis direction of k-th joint and $\boldsymbol{q}'_k$ represents pivot point location of k-th joint. These joint parameters are predicted by joint function $\boldsymbol{\Phi}' = J(\boldsymbol{\beta};\boldsymbol{\Gamma})$ in the shape branch of OMAD-PriorNet.

$$L_{\text{joint}} = \lambda_{\text{joint}}L_{\text{joint\_dir}} + L_{\text{joint\_loc}} \tag{10}$$

$$L_{\text{joint\_dir}} = \frac{1}{K}\sum_{k=1}^{K}\frac{\boldsymbol{u}'_k\cdot\boldsymbol{u}'^*_k}{\left\|\boldsymbol{u}'_k\right\|\cdot\left\|\boldsymbol{u}'^*_k\right\|} \tag{11}$$

$$L_{\text{joint\_loc}} = \frac{1}{\sum_{k=1}^{K}\mathbb{1}_k^{(\text{r})}}\sum_{k=1}^{K}\mathbb{1}_k^{(\text{r})}\left|\frac{\boldsymbol{u}'_k\times\boldsymbol{u}'^*_k}{\left\|\boldsymbol{u}'_k\times\boldsymbol{u}'^*_k\right\|}(\boldsymbol{q}'_k-\boldsymbol{q}'^*_k)\right| \tag{12}$$

In Eq. 10, $L_{\text{joint\_dir}}$ represents joint direction loss, $L_{\text{joint\_loc}}$ represents joint location loss and hyper-parameter λ_{joint} is set to 0.5 by default. In Eq. 11, $L_{\text{joint\_dir}}$ is used to measure the angle error between predicted joint axis direction $\boldsymbol{u}'_k$ and ground-truth joint axis direction $\boldsymbol{u}'^*_k$. In Eq. 12, $L_{\text{joint\_loc}}$ is used to measure the distance between predicted joint axis and ground-truth joint axis, and $\mathbb{1}_k^{(\text{r})}$ indicates whether the k-th joint is a revolute joint(prismatic joint does not need joint location).

References

[1] Unity game engine. http://www.unity.com/.

[2] Clara Fernandez-Labrador, Ajad Chhatkuli, Danda Paudel, Jose Guerrero, Cédric Demonceaux, and Luc Gool. Unsupervised learning of category-specific symmetric 3d keypoints from point sets. In *16TH EUROPEAN CONFERENCE ON COMPUTER VISION, ECCV 2020*. Springer, 2020.

[3] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Xiaolong Li, He Wang, Li Yi, Leonidas J Guibas, A Lynn Abbott, and Shuran Song. Category-level articulated object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3706–3715, 2020.

[6] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[7] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[8] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[9] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020.