

A Appendix

A.1 RiEMann Trainig Algorighm

Following the notations in Section 4.2, the training algorithm of RiEMann is summarized as follows:

Algorithm 1 RiEMann Training

Input: Demonstrations $\{(\mathbf{P}_i, \mathbf{T}_i)\}_{i=1}^M$, initialized models ϕ, ψ_1, ψ_2 , hyperparameters r_1 and r_2 , epochs n .

- 1: **for** $iter = 0$ to $n - 1$ **do**
- 2: Sample a batch of m demonstrations $\{(\mathbf{P}_i, \mathbf{T}_i)\}_{i=1}^m$, where $\mathbf{T}_i = (\mathbf{R}_i, \mathbf{t}_i)$
- 3: Predict the saliency map $\mathbf{f}_s(x) = \phi(x), x \in \mathbf{P}_i$
- 4: Get x_{t1} by doing weighted sum on \mathbf{P} with the softmax weight from $\mathbf{f}_s(x)$
- 5: Get \mathbf{B}_{ROI} centered on x_{t1} with radius r_1
- 6: Predict $\mathbf{f}_t(x) = \psi_1(x), \mathbf{f}_R(x) = \psi_2(x), \forall x \in \mathbf{B}_{ROI}$
- 7: Get $\hat{\mathbf{t}}$ as the weighted position of $\mathbf{f}_t(x)$ and get $\hat{\mathbf{R}}$ by mean pooing on $\mathbf{f}_R(x)$ on points centered at $\hat{\mathbf{t}}$ with the radius r_2
- 8: Normalize each type-1 vector of $\hat{\mathbf{R}}$
- 9: Update ϕ, ψ_1 , and ψ_2 with $\mathcal{L} = \sum_{i=0}^m [\sum_{j=1}^N (\mathbf{t}_i - \hat{\mathbf{t}}_i)^2 + \sum_{k=1}^{N_B} ((\mathbf{t}_i - \hat{\mathbf{t}}_i)^2 + (\mathbf{R}_i - \hat{\mathbf{R}}_i)^2)]$
- 10: **end for**

Output: Trained models ϕ, ψ_1 , and ψ_2

A.2 Iterative Modified Gram-Schmidt Orthogonalization

We use Iterative Modified Gram-Schmidt Orthogonalization [41] to make the outputted rotation matrix $\hat{\mathbf{R}}$ legal. IMGS works much more stable than the vanilla Gram-Schmidt Orthogonalization. The algorithm is summarized as follows.

Algorithm 2 Iterative Modified Gram-Schmidt

Input: $\hat{\mathbf{R}}$ that contains column vectors v_0, v_1 , and $v_2 \in \mathbb{R}^3$

- 1: **for** $iter = 1$ to 2 **do**
- 2: **for** $i = 0$ to 2 **do**
- 3: $u_i = v_i$
- 4: **for** $j = 0$ to $i - 1$ **do**
- 5: $v_i = v_i - \frac{\langle v_i, u_j \rangle}{\langle u_j, u_j \rangle} u_j$
- 6: **end for**
- 7: $u_i = v_i$
- 8: **end for**
- 9: **end for**

Output: A legal rotation matrix $\hat{\mathbf{R}}$ that contains updated column vectors v_0, v_1 , and $v_2 \in \mathbb{R}^3$

A.3 Proofs of Theories

First, let's review the definition SE(3)-equivariance on our point cloud \mathbf{P} . Given an outputted vector field $\mathbf{f}_{out}(x) = \bigoplus_{i=1}^n \mathbf{f}^i(x), \forall x \in \mathbf{P}$ from SE(3)-transformer [17] where n is the total types of vectors, they are SE(3)-equivariant that means:

$$\mathbf{D}_l(\mathbf{R})\mathbf{f}_l(x) = \mathbf{f}_l(Tx), \forall x \in \mathbf{P}, T = (\mathbf{R}, \mathbf{t}) \in SE(3), l \in n. \quad (5)$$

where $\mathbf{D}_l(R)$ is the Wigner-D matrix. For entities living in the usual 3D physical world, the angular momentum quantum number j of the Wigner-D matrix is 1, thus for vectors with $l = 1$, we have:

$$\mathbf{D}(\mathbf{R}) = e^{-im'\alpha} d_{m',m}^j(\beta) e^{-im\gamma}, \quad (6)$$

where α, β, γ are the Euler angle representation of \mathbf{R} that satisfies $\mathbf{R} = \mathbf{R}_z(\alpha)\mathbf{R}_x(\beta)\mathbf{R}_z(\gamma)$, $m, m' \in \{-1, 0, 1\}$, and $d_{m',m}^j(\beta)$ is the matrix element. Since $\mathbf{D}(\mathbf{R})$ is also a unitary matrix, we can find a set of basis $[v_0, v_1, v_2]$ that makes $\mathbf{D}(\mathbf{R}) = \mathbf{R}$.

For type-0 vector fields, $\mathbf{D}_0(\mathbf{R})$ is a one-dimensional identical scale factor 1. Let's begin our proofs.

Theorem 1 *Rotation matrices, represented by three type-1 vectors, are SE(3)-equivariant parameterization of rotation actions.*

Proof: For a rotation matrix $\mathbf{R} = [v_0, v_1, v_2] \in \mathbb{R}^9$, where v_0, v_1 , and v_2 are the three column vectors, we use three type-1 vectors to represent these three column vectors. Thus the output vector of the network is as follows:

$$\mathbf{f}_{out}(x) = \bigoplus_{i=1}^3 \mathbf{f}_i^1(x), \forall x \in \mathbf{P}. \quad (7)$$

When the input point cloud is transformed by a SE(3) transformation $\mathbf{T} = (\mathbf{R}, \mathbf{t})$, the rotation matrix representation of the target object also transforms by \mathbf{R} , that is $[v'_0, v'_1, v'_2] = \mathbf{R}[v_0, v_1, v_2]$. According to Equation 5 and 6, the outputted type-1 vector fields are transformed by $\mathbf{D}_1(\mathbf{R}) = \mathbf{R}$, thus we have:

$$[v'_0, v'_1, v'_2] = \mathbf{D}_1(\mathbf{R})[v_0, v_1, v_2] \quad (8)$$

■

Theorem 2 *There is no SE(3)-equivariant vector field representation for Euler angle, quaternion, and axis-angle.*

Proof: We use proof by contradiction to prove theorem 2. Consider the example illustrated in Figure 2.

1) For quaternion, we define a quaternion $q = [\cos(\theta/2), \sin(\theta/2)u_i, \sin(\theta/2)u_j, \sin(\theta/2)u_k]$ where θ is the rotation angle and $[u_i, u_j, u_k]$ is the rotation axis. For the initial pose, we have $q = [1, 0, 0, 0]$. For the end pose, we have $q' = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0, 0]$, thus:

$$q' = q + [\frac{\sqrt{2}}{2} - 1, \frac{\sqrt{2}}{2}, 0, 0]. \quad (9)$$

There are two options for quaternion type- l parameterization: 1) using four type-0 vectors; 2) using one type-1 vector and one type-0 vector. For both cases, the $[\frac{\sqrt{2}}{2} - 1, \frac{\sqrt{2}}{2}, 0, 0]$ operation cannot be represented by a Wigner-D matrix. Thus there is no SE(3)-equivariant vector field representation for quaternion. Axis-angle can be proven in the same way.

2) For Euler angles, we define an Euler angle as $E = (\alpha, \beta, \gamma)$. For the initial pose, we have Euler angles equal to $E = (0, 0, 0)$. For the end pose, we have Euler angles equal to $E' = (-\frac{\pi}{2}, 0, 0)$. Thus we have:

$$E' = E + [-\frac{\pi}{2}, 0, 0]. \quad (10)$$

There are two options for Euler angles' type- l parameterization: 1) using three type-0 vectors; 2) using one type-1 vector. For both cases, the $[-\frac{\pi}{2}, 0, 0]$ operation cannot be represented by a Wigner-D matrix. Thus there is no SE(3)-equivariant vector field representation for Euler angles.

■

A.4 Training Details

A.4.1 Point Cloud Preprocessing

In the real-world experiments, we perform point cloud voxel downsampling before feeding the point cloud into the network with the voxel size equal to 1cm for the *Mug on Rack* task and 2cm for the task *Plane on Shelf*.

After this, we perform color jittering by adding Gaussian noise on each point’s color with a standard variance equal to 0.005. We also perform random color dropping that replaces 30% points’ color to zero, and HSV transformation that randomly transfers the hue, saturation, and brightness of each point by 0.4, 1.5, and 2 times respectively.

A.4.2 Implementation Details

For RiEMann, We use $r_1 = 0.2m$, $r_2 = 0.02m$ for all the tasks in the simulation. We use $r_1 = 0.16m$, $r_2 = 0.02m$ for the real world *Mug on Rack* task, and $r_1 = 0.2m$, $r_2 = 0.02m$ for the *Plane on Shelf* task. Other network hyperparameters of RiEMann are listed in Table 5. We do not use any kind of prior knowledge for the training of RiEMann such as object segmentation, pertaining, or pose augmentations. We also set the robot to first reach some pre-defined pre-grasp pose (e.g., above the mug) and a pre-place pose (e.g., in front of the rack) to eliminate the unnecessary influence of the motion planners.

The full training of RiEMann takes 200 epochs on a single NVIDIA A40 with a batch size of 4 and a learning rate of $1e-4$ for each network module ϕ , ψ_1 , and ψ_2 . However, we find that for *Mug on Rack*, they only need about 50 epochs to converge, which takes about 47 minutes.

Table 5: Network hyperparameters of RiEMann.

	Network Layer	Max Type- l	Head Number	Channels	Message Passing Distance
ϕ	4	4	1	8	0.1m
ψ_1	4	3	1	8	0.07m
ψ_2	4	4	1	8	0.07m

For PerAct [37], the language descriptions of our tasks are: *Put the mug on the rack*, *Place the plane on the shelf*, *Turn the faucet*. We follow the original 3D voxel grid size (100^3) and the patch size (5^3). We use Euler angles as the rotational action representations for PerAct. For fairness, we do not train the gripper action for PerAct. We use 6 self-attention layers for the perceiver Transformer module. The other hyper-parameters are the same with the original paper.

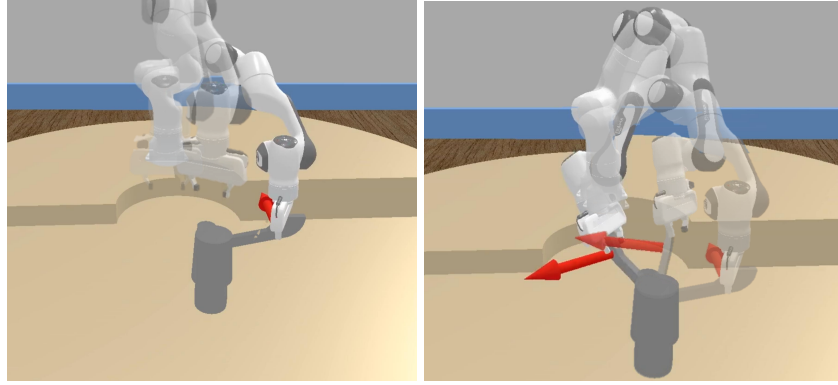
For R-NDF [13], since there is no pre-trained weight for the plane and the faucet, we here pre-train the NDFs using the reconstructed meshes from the point clouds in our demonstrations, and use these model as the NDFs module to run R-NDFs. We observe that R-NDFs fail to accomplish all of the tasks when testing, which shows that R-NDFs cannot perform well without object segmentation, because of the locality requirements of R-NDFs. We also tried to use the original pre-trained weights from the original paper [13] for the task *Mug on Rack*, but we found that the performances were even worse because of the discrepancy of the specific object shapes in the test experiments and the pertaining datasets. Other network hyper-parameters are the same as in the original paper.

For EDF [14] and D-EDF [15], we manually separate the robot end-effector and the grasped object point cloud from the scene rather than setting a series of separate cameras to capture their point cloud. For EDFs, we run the MH for 1000 steps, run the Langevin algorithm for 300 steps, and optimize the samples for 100 steps. We use one query point for picking and three query points for placing. We train EDFs for 200 epochs, the same epochs with RiEMann. For D-EDFs, we train the networks for 1 hour with parallel training of the low-resolution and the high-resolution networks and the energy-based critic network. Other network hyper-parameters are the same as in the original paper.

A.5 Simulation Experiments

A.5.1 Detailed Descriptions of Simulation Tasks

For all simulation tasks, the radius of the table is $0.75m$, and we put a Franka Panda robot arm in the center of the table. We divide the table into a semicircle part and two quarter parts and make them different heights with a height difference of $0.1m$. This is designed to conveniently apply SE(3)



(a) First step: the robot moves to the predicted target pose of the end-effector. (b) Second step: the robot opens the faucet along the predicted direction (red arrow).

Figure 6: The articulated object manipulation task *Turn Faucet*.

transformations on target objects. We cut the input point cloud into a cube with a side length of 2m centered on the center of the table. Details of different tasks are introduced here:

Mug on Rack: A mug and a rack are placed on the table. The robot has to pick up the mug by the rim and then hang it on the rack by the handle. This is the most representative object rearranging task that is also evaluated in [12, 14, 13, 15]. For the training set \mathbf{T} , we use a mug in blue with a side length of about 17 cm, and a rack with a height of 67cm. The mug must be hung on the highest peg of the rack. For the new instance set \mathbf{NI} , we use a patterned red mug with a height of about 19cm and a base diameter of about 10cm.

Plane on Shelf: A plane model and a box-shape shelf are placed on the table, and the robot has to pick the middle part of the body of the plane and place it on the shelf. For \mathbf{T} , we use a grey plane model with a length of about 20cm. For \mathbf{NI} , we use a blue plane with the same size.

Turn Faucet: As shown in Figure 6, a faucet is placed on the table, and the robot has to turn on the faucet by first moving to the handle of the faucet and then moving along the opening direction. For \mathbf{T} , we use the NO. 5004 faucet model in ManiSkill2 [45]. For \mathbf{NI} , we use the No. 5005 faucet model.

For the *open-faucet* task, we assume that given a target pose $\mathbf{T} = \{\mathbf{R}, \mathbf{t}\} \in SE(3)$ and a target direction $\mathbf{d} \in \mathbb{R}^3$, the robot can accomplish the task by first going to the target pose \mathbf{T} just as done in pick-and-place tasks, and then moving along the target direction \mathbf{d} while keeping the orientation not changed, as illustrated in Figure 6. To encode the extra directional action \mathbf{d} , we add another type-1 vector field on the orientation network ψ_2 , that is: $\mathbf{f}_R = \bigoplus_{i=1}^4 \mathbf{f}_1^i(x), x \in \mathbf{B}_{ROI}$. The final output directional action $\hat{\mathbf{d}}$ is also calculated through mean pooling on points in the radius r_2 . Note the policy needs to continuously predict the output direction during the opening process, which shows that RiEMann can capture the local $SE(3)$ -equivariance of the handle part of the faucet.

In this task, we give demonstrations of not only the target pose \mathbf{T} , but also the opening direction \mathbf{d} at the first frame. Here there exists two kinds of $SE(3)$ -equivariance: the target pose should be equivariant to the pose of the faucet at the first frame, and the opening direction should be equivariant to the handle during the opening process, where the second equivariance requires both local-equivariance and the real-time performance.

A.5.2 Demonstration Collection

We provide the ground truth pose to a point cloud based motion planner MPlib [46] to generate the demonstration trajectory for training. We transform all point cloud input to the end-effector coordinate system. We collect 10 demonstrations for each setting for the evaluation of $SE(3)$ geodesic

distance. We manually exclude those situations that cannot support a successful collision-free motion planning trajectory, as well as in the testing cases.

A.6 Real World Experiments

A.6.1 Environment Setup

We use a Franka Emika Panda robot arm with four RealSense D435i RGB-D cameras for the real-world experiments, as shown in Figure 4. The cameras are calibrated relative to the robot’s base frame. We fuse the point clouds from all four cameras and transform the point cloud into the end-effector frame of the robot for control. We crop the scene to a cube with a side length of 1.5 meters and downsample the point to get 8192 points in the scene.

For real-world tasks, since the point cloud is noisy and usually part-occluded, we do not perform the pose transformation calculation $\hat{\mathbf{T}} = \mathbf{T}_{place} \mathbf{T}_{object}^{-1}$ for the task, i.e., we directly use the predicted target pose as the final action.

A.6.2 Detailed Task Descriptions

Mug on Rack: The task is similar to the version in the simulation. For **T**, we use a pink mug with a side length of about 10cm. We use a rack with a height of 35cm and a base diameter of about 15cm. We split the table into four equal areas, as in the simulated version of this task. In **T**, we only collect demonstrations in a quarter of the desktop area and let the mug rotate along the z-axis for 90 degrees in a top pose. For **NI**, we use a yellow new mug with a similar size to the pink mug. For **NP**, we let the mug on all table regions and rotate in 3 dimensional with any degree.

Plan on Shelf: The task is similar to the version in the simulation. For **T**, we use a blue plane model, and collect demonstrations in the same manner as above. For **NI**, we use a green plane with a similar size. The shelf is a set of discrete racks that can support the plane if it is placed in the correct pose.

A.6.3 Demonstration Collection

We use the teaching mode of the robot arm to give demonstrations, as illustrated in Figure 1 and in supplementary videos. We transform all the point clouds into the end-effector coordinate system.

A.7 More Illustrations

A.7.1 Failure Case Illustrations

We show some failure case point clouds of RiEMann in both tasks in Figure 7. We can see that the lower section of the object is not captured by the cameras, which leads to incomplete geometries.

A.7.2 Results Illustrations

We here illustrate the results and the features of the *plane-on-shelf* task in Figure 8.

A.8 Ablation Studies

A.8.1 Ablation of Hyperparameters

We test the trained model on the **NP** case of the task *Mug on Rack* in simulation. Results are shown in Figure 9. We can see that with more points in the scene input point cloud, the performance of the model is better, and the same is the number of demonstrations. Our model can achieve competitive results with less than 10 demonstrations. For the hidden layer type- l experiment, we can see that with a higher max type- l , the model can work better. However, in practice, higher type- l will extremely increase the computational cost and GPU memory usage. In the *Mug On Rack* task, a network with a maximum number l equals 5 only supports batch size = 1 during training on an NVIDIA A40. Lastly, the last group of Figure 9 shows that the saliency map network can not only reduce the training burden of the policy network but also improve the final pose estimation results.

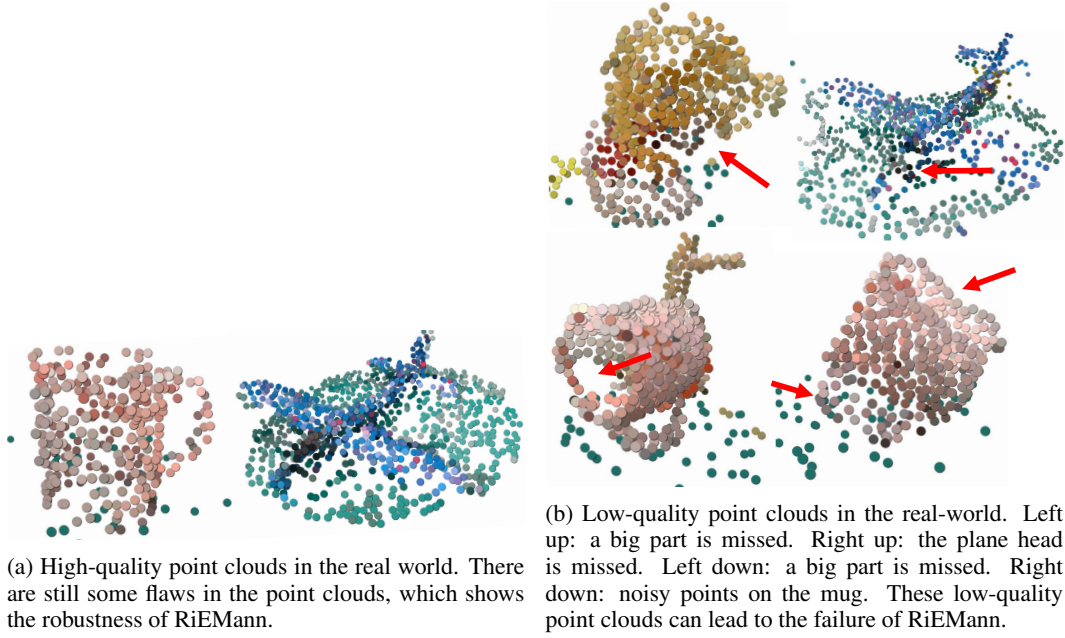


Figure 7: Visualization of the low-quality data in the real-world experiments that cause the failure of experiments.

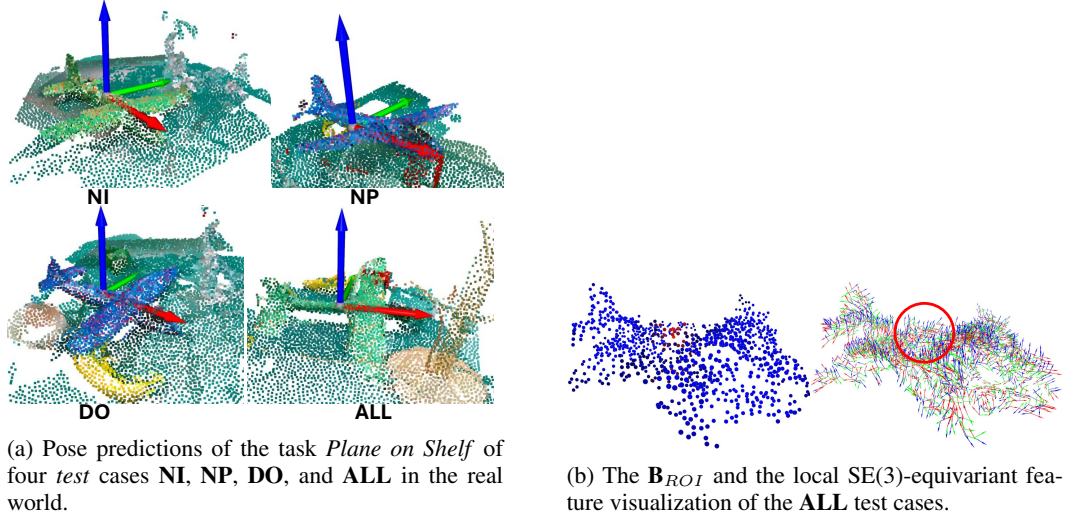


Figure 8: Test pose predictions and feature visualization of real-world evaluations of the *plane on shelf* task.

586 A.8.2 Ablation of Radius r_1 and r_2

587 Here we add additional experiments to answer this question. We train the *mug-on-rack* task in the
 588 simulation with different r_1 and r_2 . Results are in Table 6.

589 Note in our paper, we choose $r_1 = 0.16m$ and $r_2 = 0.02m$. In the *mug-on-rack* task, the height of
 590 the mug is about $0.22m$ and the width of the mug is about $0.20m$. Note $r_2 = 0.01m$ means that
 591 there is only one point for ψ_2 to perform mean pooling, because we perform point cloud voxelization
 592 for the point cloud with a voxel size of $0.01m$.

593 From the results, we can see that:

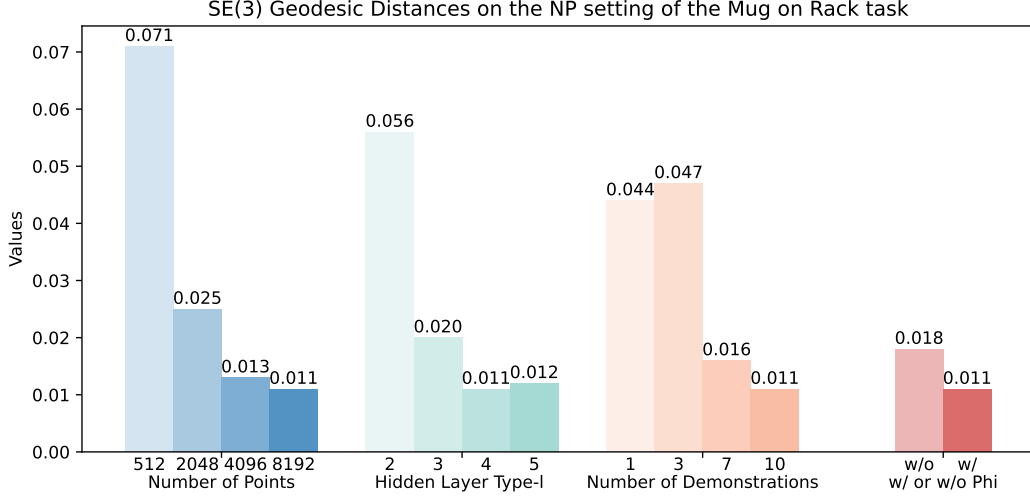


Figure 9: Ablation studies of different hyperparameters of RiEMann. Each value is the average result of 20 random seeds.

Table 6: Success rates and SE(3)-geodesic distances of different r_1 and r_2 of the *mug-on-rack* task in simulation. Each value is evaluated under 20 random seeds. In our original paper, we choose $r_1 = 0.16m$ and $r_2 = 0.02m$.

r_1/m		0.05			0.10			0.16			0.22			0.28		
r_2/m		0.01	0.02	0.04	0.01	0.02	0.04	0.01	0.02	0.04	0.01	0.02	0.04	0.01	0.02	0.04
SR	T	0.00	0.00	0.00	0.50	0.85	0.80	0.90	1.00	1.00	0.90	1.00	1.00	0.95	1.00	1.00
	NI	0.00	0.00	0.00	0.10	0.80	0.75	0.80	0.90	0.95	0.90	0.95	0.90	0.90	0.90	0.95
	NP	0.00	0.00	0.00	0.45	0.75	0.80	0.85	0.95	1.00	0.90	0.95	0.95	0.95	0.95	1.00
	DO	0.00	0.00	0.00	0.35	0.80	0.70	0.70	1.00	1.00	0.85	0.95	1.00	0.80	1.00	1.00
	ALL	0.00	0.00	0.00	0.00	0.50	0.55	0.50	0.85	0.85	0.70	0.85	0.90	0.70	0.85	0.80
\mathcal{D}_{geo}	T	1.366	1.478	1.175	0.688	0.178	0.186	0.278	0.053	0.056	0.067	0.064	0.062	0.059	0.060	0.055
	NI	1.159	1.759	1.311	0.982	0.220	0.201	0.321	0.066	0.061	0.068	0.061	0.062	0.066	0.063	0.062
	NP	1.668	1.076	1.079	0.804	0.123	0.202	0.339	0.069	0.066	0.063	0.066	0.055	0.060	0.064	0.058
	DO	1.460	1.298	1.290	0.866	0.175	0.199	0.297	0.058	0.056	0.082	0.062	0.059	0.079	0.064	0.063
	ALL	1.982	1.333	1.077	1.390	0.797	0.639	0.427	0.071	0.070	0.079	0.077	0.068	0.088	0.079	0.072






- r_1 should be large enough ($\geq 0.16m$) to capture all points of the target objects to make sure the position and orientation network can understand the full object geometry. If r_1 is less than the radius of the mug, the success rate will drop quickly, and the SE(3) geodesic error will increase quickly. If r_1 is too small ($r_1 = 0.05m$), the success rate becomes zero. The reason is that the training becomes very unstable because every time the position and the orientation network may choose different parts of the point cloud for training since the saliency map network is not that strong to give very precise position predictions.
- r_2 should capture more than one point ($> 0.01m$) to avoid noisy and accidental errors caused by too few points.
- Although $r_2 = 0.04m$ also works well, it brings more burden for training since the orientation network ψ_2 outputs the highest type- l vectors in our whole pipeline, and using fewer points for training is better for reducing the memory usage for ψ_2 .
- For $r_2 = 0.01m$, the results are significantly worse than other situations in **DO** and **ALL**. This shows that noisy points will influence the results if we only use one point for prediction.
- The worst SE(3)-geodesic error of RiEMann ($r_1 = 0.05m$) is smaller than the R-NDF baseline [13] in Table 2. This is because the saliency map network will restrict the position error to be in a certain range.

A.8.3 The Geometry Generalization Ability of RiEMann

The generalization ability of RiEMann on the new geometry comes from the neural network structure and its inductive bias. We think it is the locality property of the message-passing mechanism in the equivariant backbone that brings this geometry-level generalization. We perform the *new instance* experiments because, in related works [14, 15], they found that equivariant networks have a certain degree of geometric generalization ability only with 5 to 10 demonstrations, and we confirmed this point in our experiments. Yes, compared to NDFs [12, 13], our geometry generalization ability is worse because their encoder is trained on a large-scale in-category dataset.

To better demonstrate how strong the generalization ability of RiEMann is on new object geometries, here we add an additional experiment. We gradually deform the mesh of the mug trained in the *mug-on-rack* task, put them into the **NP** setting, and record the corresponding SE(3)-geodesic error of the predictions from the original trained model. We keep the longest side of the mug the same. The results are in Table 7.

Table 7: SE(3)-geodesic distance evaluations of the trained model on the training mug. Each value is evaluated under 20 random SE(3) poses.

Picture Description					
	original	flat	tall	fat	new
\mathcal{D}_{geo}	0.055	0.187	0.127	0.094	0.395

We can see that if the general shape of the mug remains (especially for the rim of the mug), the \mathcal{D}_{geo} remains at a low level, which shows that RiEMann has a certain power of geometry generalization ability. However, if the geometry of the mug is deformed too much, the result becomes worse.