A Proof of Proposition 4.1

We need the following Lemma regarding the visual state space S and spatial action space A described in Section 3. We use the following notation: $gS = \{gs | s \in S\}$ and $gA = \{ga | a \in A\}$.

Lemma A.1. *Let S be a visual state space and let A by a spatial action space. Then, $\forall g \in \mathrm{SE}(2)$, we have that $S = gS$ and $A = gA$.*

Proof. First, consider the claim that $S = gS$. We will show 1) $S \subseteq gS$ and 2) $gS \subseteq S$. 1) $S \subseteq gS$: This follows from the closure of state under $g \in \mathrm{SE}(2)$. 2) $gS \subseteq S$: Let $s' \in gS$. By the definition of gS, $\exists s \in S$ such that $gs = s'$ and $gs \in gS$. Multiplying both sides by g^{-1}, we have $g^{-1}(gs) \in g^{-1}(gS)$. Using Assumption 3.3, we have $s \in S$. Using the closure of state under g, we have $gs \in S$ or $s' \in S$. A parallel argument can be used to show $A = gA$. □

Proposition 4.1. *Given an MDP $\mathcal{M} = (S, A, T, R, \gamma)$ for which Assumptions 3.1, 3.2, and 3.3 are satisfied, the optimal Q function is invariant to translation and rotation, i.e. $Q^*(s, a) = Q^*(gs, ga)$, for all $g \in \mathrm{SE}(2)$.*

Proof of Proposition 4.1. The Bellman optimality equations for $Q^*(s, a)$ and $Q^*(gs, ga)$ are, respectively:

$$Q^*(s, a) = R(s, a) + \gamma \sup_{a' \in A} \int_{s' \in S} T(s, a, s') Q^*(s', a'), \tag{4}$$

and

$$Q^*(gs, ga) = R(gs, ga) + \gamma \sup_{a' \in A} \int_{s' \in S} T(gs, ga, s') Q^*(s', a'). \tag{5}$$

Using Lemma A.1, we can rewrite Eq. 5 as:

$$Q^*(gs, ga) = R(gs, ga) + \gamma \sup_{\bar{a}' \in gA} \int_{\bar{s}' \in gS} T(gs, ga, \bar{s}') Q^*(\bar{s}', \bar{a}') \tag{6}$$

$$= R(gs, ga) + \gamma \sup_{a' \in A} \int_{s' \in S} T(gs, ga, gs') Q^*(gs', ga'). \tag{7}$$

Using Assumptions 3.1 and 3.2, this can be written:

$$Q^*(gs, ga) = R(s, a) + \gamma \sup_{a' \in A} \int_{s' \in S} T(s, a, s') Q^*(gs', ga'). \tag{8}$$

Now, define a new function $\bar{Q}$ such that $\forall s, a \in S \times A$, $\bar{Q}(s, a) = Q(gs, ga)$ and substitute into Eq. 8, resulting in:

$$\bar{Q}^*(s, a) = R(s, a) + \gamma \sup_{a' \in A} \int_{s' \in S} T(s, a, s') \bar{Q}^*(s', a'). \tag{9}$$

Notice that Eq. 9 and Eq. 4 are the same Bellman equation. Since solutions to the Bellman equation are unique, we have that $\forall s, a \in S \times A$, $Q^*(s, a) = \bar{Q}^*(s, a) = Q^*(gs, ga)$.

□

B Equivariant Kernel Constraint

Consider a standard convolutional layer that takes an $n \times h \times w$ feature map as input and produces an $m \times h \times w$ map as output. It computes $h_i(x) = \sum_{y,j} K_{ij}(y) I_j(x + y)$, where $j \in \{1 \dots n\}$, $i \in \{1 \dots m\}$, $I_j(x)$ is the value of the input at the x pixel and the j channel, $h_i(x)$ is the output at pixel x and channel i, and $K_{ij}(y)$ is the kernel value at y for the j input and i output channels. For a standard convolutional layer, $I_j(x)$, $h_i(x)$, and $K_{ij}(y)$ are all scalars. However, for an equivariant network over C_u, $h_i(x)$ becomes a u-element vector and $K_{ij}(y)$ becomes a $u \times u$ matrix. The u elements of $h_i(x)$ encode the feature values of pixel x at channel i at each orientation in C_u. The kernel constraint is [33]:

$$K_{ij}(g_\theta y) = \rho_{\mathrm{out}}(g_\theta) K_{ij}(y) \rho_{\mathrm{in}}(g_\theta)^{-1}, \tag{10}$$

where $\rho_{\mathrm{in}}(g_\theta)$ and $\rho_{\mathrm{out}}(g_\theta)$ are the permutation matrix of the group element g_θ (note that for the first layer, $K_{ij}(y)$ will be a $1 \times u$ matrix, and $\rho_{\mathrm{in}}(g_\theta)$ will be 1).

(a) (b)

Figure 11: (a) All eight bottle models in the Bottle Arrangement task. (b) All seven object models in the Bin Packing task.

C Experimental Domains

C.1 Block Stacking

In the Block Stacking task (Fig 1a), there are four cubic blocks with a fixed size of $3cm \times 3cm \times 3cm$ randomly placed in the workspace. The goal is stacking all four blocks in a stack. An optimal policy requires six steps to finish this task, and the maximal number of steps per episode is 10.

C.2 Bottle Arrangement

In the Bottle Arrangement task (Fig 1b), six bottles with random shapes (sampled from 8 different shapes shown in Fig 11a. The bottle shapes are generated from the 3DNet dataset [35]. The sizes of each bottle are around $5cm \times 5cm \times 14cm$) and a tray with a size of $24cm \times 16cm \times 5cm$ are randomly placed in the workspace. The agent needs to arrange all six bottles in the tray. An optimal policy requires 12 steps to finish this task, and the maximal number of steps per episode is 20.

C.3 House Building

In the House Building task (Fig 1c), there are four cubes with a size of $3cm \times 3cm \times 3cm$, a brick with a size of $12cm \times 3cm \times 3cm$, and a triangle block with a bounding box size of around $12cm \times 3cm \times 3cm$. The agent needs to stack those blocks in a specific way to build a house-like block structure as shown in Fig 1c. An optimal policy requires 10 steps to finish this task, and the maximal number of steps per episode is 20.

C.4 Covid Test

In the Covid Test task (Fig 1d), there is a new tube box (purple), a test area (gray), and a used tube box (yellow) placed arbitrarily in the workspace but adjacent to one another. Three swabs with a size of $7cm \times 1cm \times 1cm$ and three tubes with a size of $8cm \times 1.7cm \times 1.7cm$ are initialized in the new tube box. To supervise a COVID test, the robot needs to present a pair of a new swab and a new tube from the new tube box to the test area (see the middle figure in Fig 1d). The simulator simulates the user testing COVID by putting the swab into the tube and randomly place the used tube in the test area. Then the robot needs to re-collect the used tube into the used tube box. Each episode includes three rounds of COVID test. An optimal policy requires 18 steps to finish this task, and the maximal number of steps per episode is 30.

C.5 Box Palletizing

In the Box Palletizing task (Fig 1e) (some object models are derived from Zeng et al. [1]), a pallet with a size of $23.2cm \times 19.2cm \times 3cm$ is randomly placed in the workspace. The agent needs to stack 18 boxes with a size of $7.2cm \times 4.5cm \times 4.5cm$ as shown in Fig 1e. At the beginning of each episode and after the agent correctly places a box on the pallet, a new box will be randomly placed in the empty workspace. An optimal policy requires 36 steps to finish this task, and the maximal number of steps per episode is 40.

C.6 Bin Packing

In the Bin Packing task (Fig 1f), eight objects (the shape of each is randomly sampled from seven different object in Fig 11b. Object models are derived from Zeng et al. [21]) with a maximum size of $8cm \times 4cm \times 4cm$ and a minimum size of $4cm \times 4cm \times 2cm$ and a bin with a size of $17.6cm \times 14.4cm \times 8cm$ are randomly placed in the workspace. The agent needs to pack all eight objects in the bin while minimizing the highest point (h_{max} cm) of all objects in the bin. The Bin Packing task has real value sparse rewards: a reward of $8 - h_{max}$ is given when all objects are placed in the bin. An optimal policy requires 16 steps to finish this task, and the maximal number of steps per episode is 20.

C.7 $\mathrm{SE}(3)$ House Building and Box Palletizing

In the $\mathrm{SE}(3)$ House Building (Fig 9a) and the Box Palletizing (Fig 9b) tasks, a bumpy surface is generated by nine pyramid shapes with a random angle sampled from 0 to 15 degrees. The orientation of the bumpy surface along the z axis is randomly sampled at the beginning of each episode. In the Bumpy House Building task, a flat platform with a size of $13cm \times 13cm$ and a height same as the highest bump is randomly placed in the workspace. The agent needs to build the house on top of the platform. In the Bumpy Box Palletizing task, the pallet is raised by the same height as the highest bump (so that it will be horizontal to the ground). All other parameters mirror the original House Building task and the original Box Palletizing task.

D Network Architecture

All of our network architectures are implemented using PyTorch [36]. We use the e2cnn [14] library to implement the steerable convolutional layers. Appendix D.1 and Appendix D.2 respectively show the network architectures of the equivariant FCN and equivariant ASR using the dynamic filter for partial equivariance. Appendix D.3 shows the architecture of lift expansion partial equivariance. Appendix D.4 shows the architecture of the deictic encoding.

D.1 Equivariant FCN Architecture

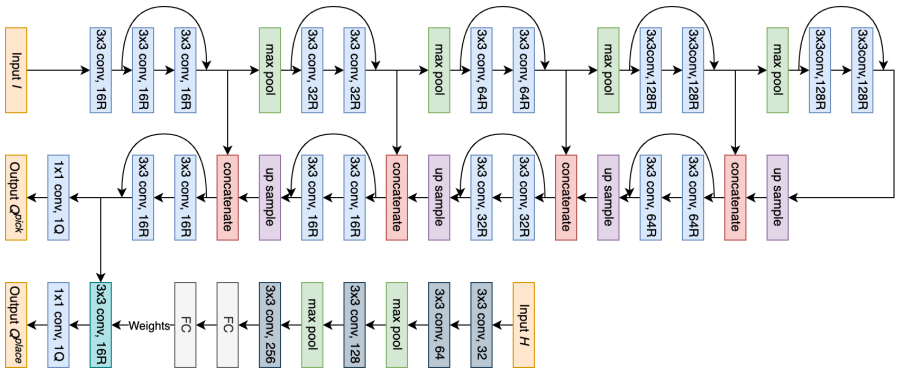
In the Equivariant FCN architecture (Fig 12a), we use we use a 16-stride UNet [37] backbone where all layers are steerable layers. The input is viewed as a trivial representation and is turned into a 16-channel regular representation feature map after the first layer. Every layer afterward in the UNet uses the regular representation, and the output of the UNet is a 16-channel regular representation feature map. This feature map is sent to a quotient representation layer to generate the pick Q value maps for each θ. For the place Q values, the non-equivariant information from H must be Incorporated. H is sent to 4 conventional convolutional layers followed by 2 FC layers. The output is a vector with the same size as the number of the free weights in a 16-channel regular representation steerable layer with a kernel size of 3×3. This output vector is expanded into a steerable convolutional kernel and is convolved with the output of the UNet. The result is sent to a quotient representation layer to generate the place Q value maps for each θ.

D.2 Equivariant ASR Architecture

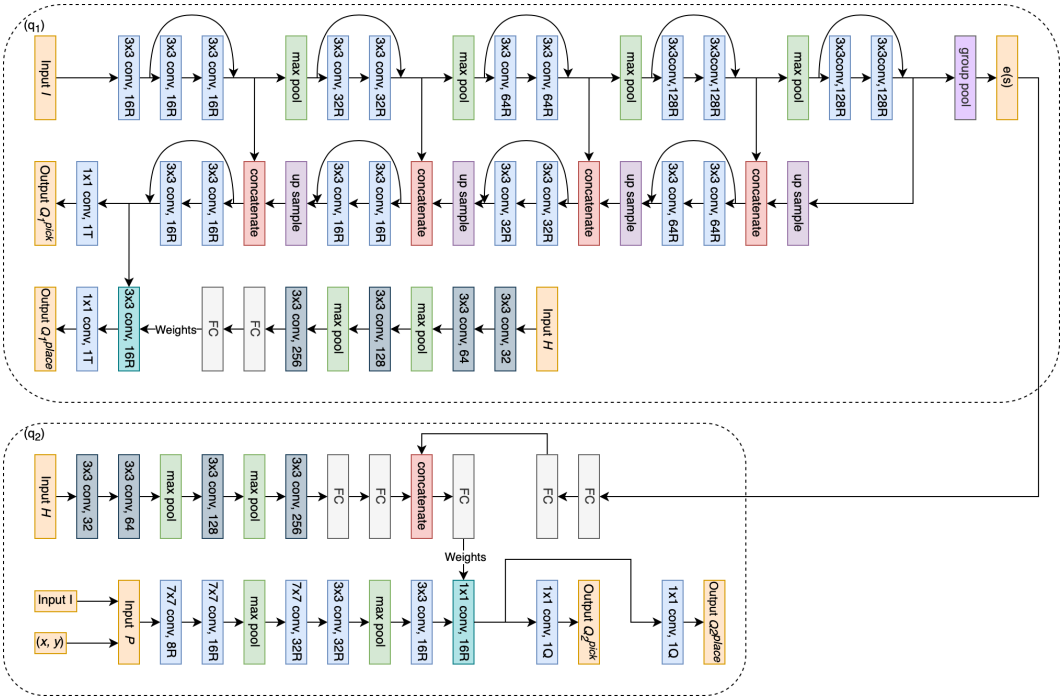
Fig 12b shows the Equivariant ASR network architecture. The q_1 architecture is very similar to the Equivariant FCN network. Its output is a trivial representation instead of a regular representation to generate only one Q map for the x, y positions. The bottleneck feature map is passed through a group pooling layer (a max pooling over the group's dimension) to form $e(s)$, a state encoding that is used by q_2. q_2 uses $e(s)$ and the feature vector from H to generate the weights for a steerable dynamic filter. q_2 processes P using a set of steerable convolution layers in the regular representation, then convolves the feature map with the dynamic filter. The result of the dynamic filter layer is sent to two separate quotient representation layers to generate pick and place values for each θ.

D.3 Lift Expansion Architecture

Fig 13 shows the equivariant FCN using lift expansion for encoding the partial equivariance property. The 128-vector (the output of FC 128 in the top row) is tiled to the same size as the 128-channel



(a) Equivariant FCN Network Architecture



(b) Equivariant ASR Network Architecture

Figure 12: The architecture of the Equivariant FCN Network (a) and the equivariant ASR Network (b). ReLU nonlinearity is omitted in the figure. A convolutional layer with a suffix of R indicates a regular representation layer (e.g., 16R is a 16-channel regular representation layer); a convolution layer with a suffix of Q indicates a quotient representation layer (e.g., 1Q is a 1-channel quotient representation layer); a convolution layer with a suffix of T indicates a trivial representation layer (e.g., 1T is a 1-channel trivial representation layer); a convolutional layer with a suffix of a number indicates a conventional convolutional layer. The convolutional layer colored in cyan is the dynamic filter layer whose weights are from the FC layer pointing to it.

regular representation feature map (the output of the rightmost convolutional layer in the middle row) and concatenated. In ASR, the same Lift Expansion network can be used in q_1.

D.4 Deictic Encoding Architecture

Fig 14 shows the network architecture of the Deictic Encoding network. Its output is a 2-vector, representing the values for pick and place with respect to the action (e.g., top-down rotation θ in q_2) encoded in the input patch P.

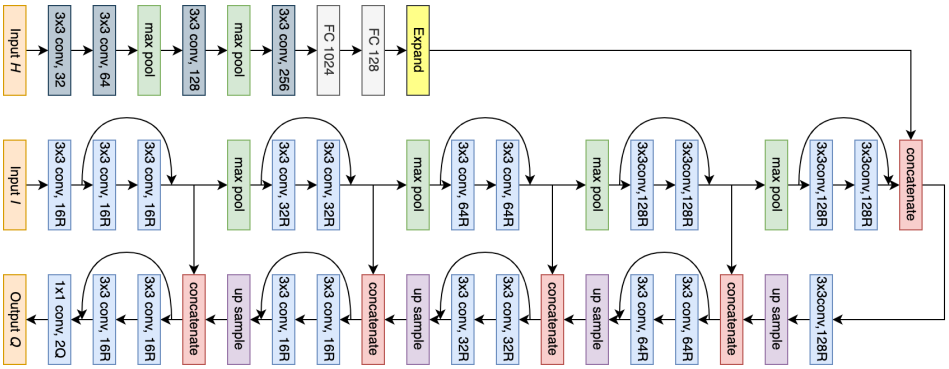


Figure 13: The Equivariant FCN with Lift Expansion

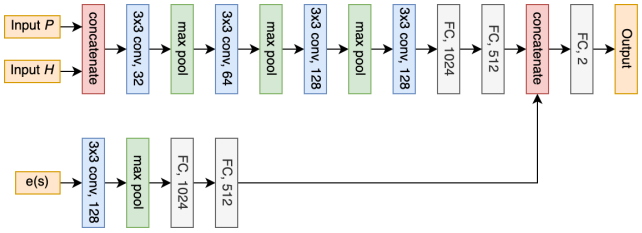


Figure 14: Deictic Encoding Network Architecture

E Baseline Details

E.1 FCN Baselines

Fig 15 shows the baseline FCN architecture. For the Conventional FCN baseline, the number of output channels $n = 2 \times |\Theta| = 12$ (i.e., one pick and one place channel for each rotation). The RAD [7] baseline uses the same baseline architecture, but during the training, each transition in the minibatch is applied with a rotational augmentation randomly sampled from C_{12}. The DrQ [8] baseline uses the same baseline architecture, but the Q targets are calculated by averaging over K augmented versions of the sampled transitions; the Q estimates are calculated by averaging over M augmented versions of the sampled transitions. Random rotation sampled from C_{12} is used for the augmentation, and we use $K = M = 2$ as in [8]. Note that in RAD and DrQ, since we are learning an equivariant Q network instead of an invariant Q network, we apply the rotational augmentation on both the state and action, rather than only augmenting the state as in the prior works. The Rot FCN baseline uses the same network backbone, but the number of output channels $n = 2$ (for pick and place, respectively). Rotations are encoded by rotating the input and output accordingly for each θ in the action space [21]. The Transporter baseline uses three FCNs (one for picking and two for placing) with the same FCN backbone shown in Fig 15. For placing, there are two networks with the same architecture for features (with an input of I) and filters (with an input of H), and the outputs of both are 3-channel feature maps. The correlation between them forms the 1-channel output. Rotations are encoded by rotating the input H for each θ in the action space. The pick network is the same as the Rot FCN baseline.

E.2 ASR Baselines

Fig 12b shows the network architecture for the Conventional ASR baseline. The RAD [7] baseline uses the same baseline architecture, but during the training, each transition in the minibatch is applied with a rotational augmentation randomly sampled from C_{32}. The DrQ [8] baseline uses the same baseline architecture, but the Q targets are calculated by averaging over K augmented versions of the sampled transitions; the Q estimates are calculated by averaging over M augmented versions of the sampled transitions. Random rotation sampled from C_{32} is used for the augmentation, and we use $K = M = 2$ as in [8]. The Transporter network baseline uses the same architecture as in Appendix E.1.

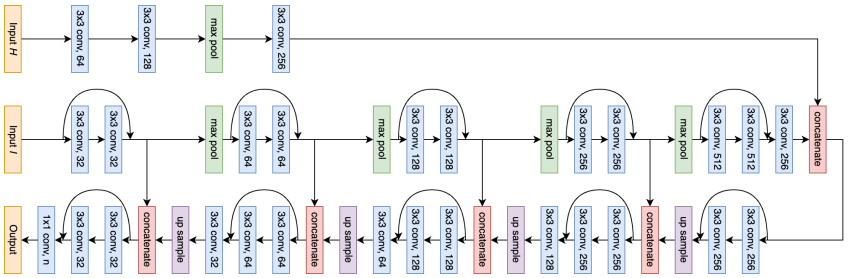


Figure 15: The baseline FCN architecture

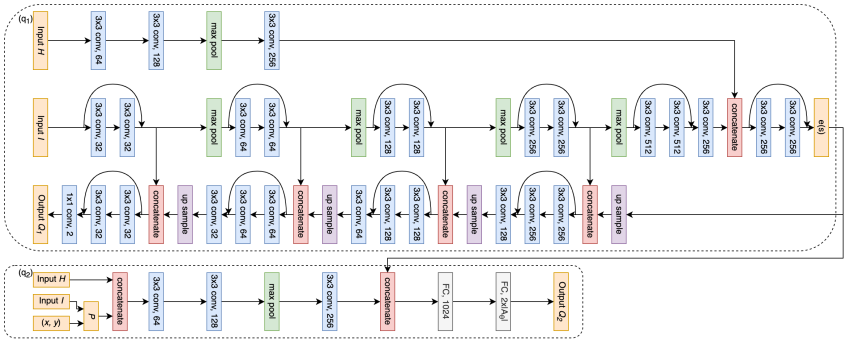


Figure 16: The baseline ASR architecture

F Training Details

F.1 SDQfD

SDQfD (Strict Deep Q Learning from Demonstrations [29]) is a variation of DQfD [38] that is better suited for large action spaces. It penalizes all actions that have a Q value larger than the expert action's Q value minus a non-expert margin. Let A^{s,a^e} be the set of actions to penalize, A^{s,a^e} is defined as:

$$A^{s,a^e} = \left\{a \in A \middle| Q(s,a) > Q(s,a^e) - l(a^e,a)\right\} \tag{11}$$

where $l(a^e,a) = l$ if $a^e \neq a$ and 0 otherwise. The margin loss term is defined as:

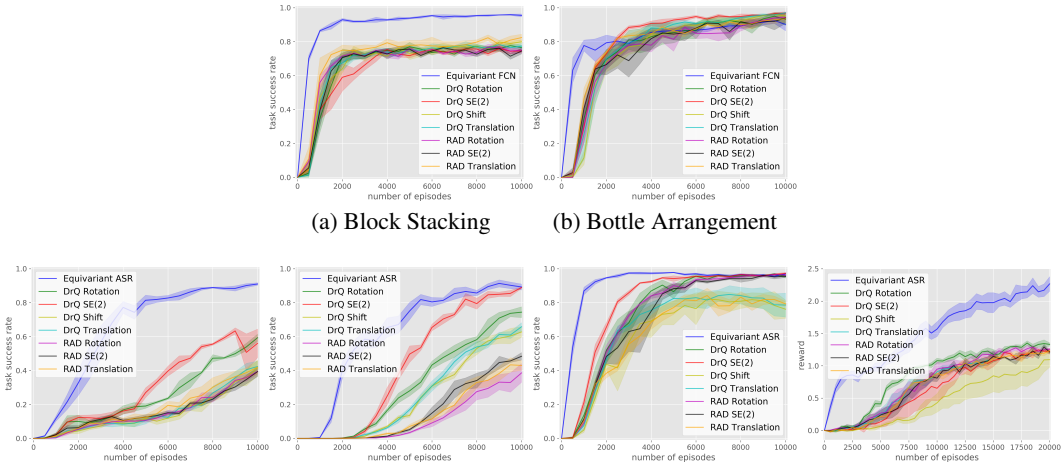
$$\mathcal{L}_{SLM} = \frac{1}{|A^{s,a^e}|} \sum_{a \in A^{s,a^e}} \left[Q(s,a) + l(a^e,a) - Q(s,a^e)\right] \tag{12}$$

$\mathcal{L}_{SLM}$ is combined with the TD loss $\mathcal{L}_{TD}$: $\mathcal{L} = \mathcal{L}_{TD} + w\mathcal{L}_{SLM}$ where w is the weight for the margin loss. Note that $\mathcal{L}_{SLM}$ is only applied for expert transitions, while on-policy transitions only apply the TD loss term.

F.2 Parameters

We implement our experimental environments using the PyBullet simulator [32]. The workspace has a size of $0.4m \times 0.4m$. In Section 4.2, I covers the workspace with a size of 90×90 pixels, and is padded with 0 to 128×128 pixels (this padding is required for the Rot FCN baseline because it needs to rotate the image to encode θ. To ensure a fair comparison, we apply the same padding to all methods). In Section 4.3, I covers the workspace with a size of 128×128 pixels. The in-hand image H is a 24×24 image crop centered and aligned with the previous pick in $\mathrm{SE}(2)$ experiments. In $\mathrm{SE}(3)$ experiments, H is a three-channel orthographic projection image (with a size of $3 \times 40 \times 40$) of a point cloud centered and aligned with the previous pick. The image patch P has a size of 24×24 in $\mathrm{SE}(2)$ experiments and a size of 40×40 in $\mathrm{SE}(3)$ experiments. In $\mathrm{SE}(2)$ experiments, z is selected by reading the height value of the area around the selected xy position.

We train our models using PyTorch [36] with the Adam optimizer [39] with a learning rate of 10^{-4} and weight decay of 10^{-5}. We use Huber loss [40] for the TD loss and cross entropy loss for the



(a) Block Stacking (b) Bottle Arrangement

(c) House Building (d) Covid Test (e) Box Palletizing (f) Bin Packing

Figure 17: Comparison against RAD and DrQ with more data augmentation operators in equivariant FCN (a-b) and equivariant ASR (c-f). Results averaged over four runs. Shading denotes standard error.

Environment	Block Stacking	Bottle Arrangement
Equivariant FCN	**0.881**	**0.781**
Transporter	0.804	0.663

Table 3: Comparison between equivariant FCN and Transporter network. Results averaged over four runs.

behavior cloning loss. The discount factor γ is 0.95. The batch size is 16 for SDQfD agents and 8 for behavior cloning agents. In SDQfD, we use the prioritized replay buffer [41] with prioritized replay exponent $\alpha = 0.6$ and prioritized importance sampling exponent $\beta_0 = 0.4$ as in Schaul et al. [41]. The expert transitions are given a priority bonus of $\epsilon_d = 1$ as in Hester et al. [38]. The buffer has a size of 100,000 transitions. The weight w for the margin loss term is 0.1, and the margin $l = 0.1$.

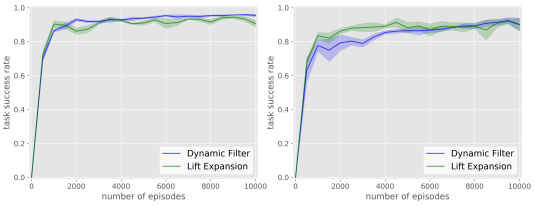
G Ablation Studies

G.1 RAD and DrQ with more data augmentation operators

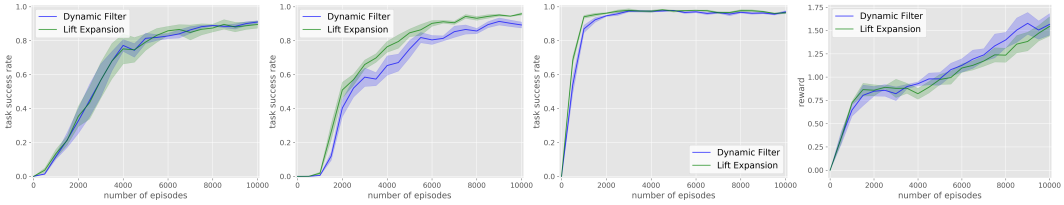
In Section 4.2 and Section 4.3, we compare the equivariant architectures with RAD [7] and DrQ [8] with rotational data augmentation. In this experiment, we run the comparison with more data augmentation operators: 1) Rotation: random rotation in C_{12} and C_{32}, same as in Section 4.2 and Section 4.3. 2) Translation: random translation. 3) SE(2): the combination of 1) and 2). 4) Shift: random shift of $\pm$ 4 pixels as in [8]. Note that only 1) is a fair comparison because our equivariant models do not inject extra translational knowledge into the network. Even though, the equivariant networks outperforms all data augmentation methods in five out of the six environments.

G.2 Dynamic Filter vs Lift Expansion

In this experiment, we compare the Dynamic Filter and Lift Expansion methods for encoding partial equivariance property. We evaluated both the equivariant FCN architecture and the equivariant ASR architecture (note that we only test this variation in q_1. q_2 uses the Dynamic Filter regardless of the architecture of q_1). The results are shown in Fig 18. Both methods generally perform equally well.

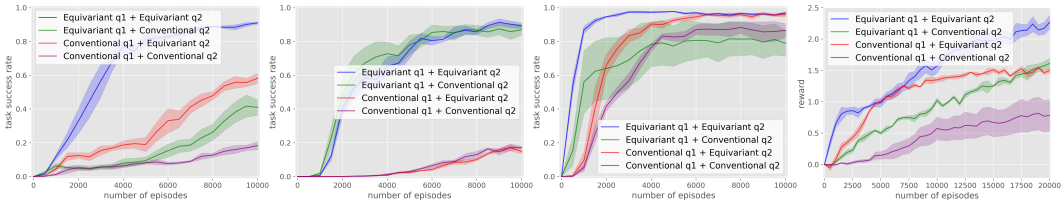


(a) Block Stacking (b) Bottle Arrangement



(c) House Building (d) Covid Test (e) Box Palletizing (f) Bin Packing

Figure 18: Comparison between Dynamic Filter and Lift Expansion in equivariant FCN (a-b) and equivariant ASR (c-f). Results averaged over four runs. Shading denotes standard error.



(a) House Building (b) Covid Test (c) Box Palletizing (d) Bin Packing

Figure 19: Comparison of four variations of equivariant/conventional and q_1/q_2 combinations. Results averaged over four runs. Shading denotes standard error.

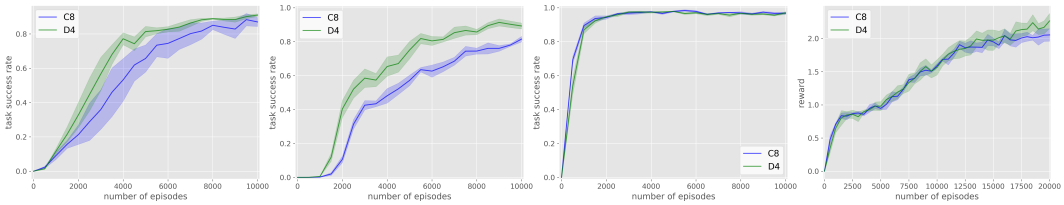
G.3 Equivariant Network in Behavior Cloning

In this experiment, we evaluate the performance of our equivariant network in a behavior cloning setting compared with the Transporter network [1]. Both methods use the same cross entropy loss function and the same data augmentation strategy. The experimental parameters mirror Section 4.2. The results are shown in Table 3. The equivariant network outperforms the Transporter network in both environments.

G.4 Equivariant ASR Ablations

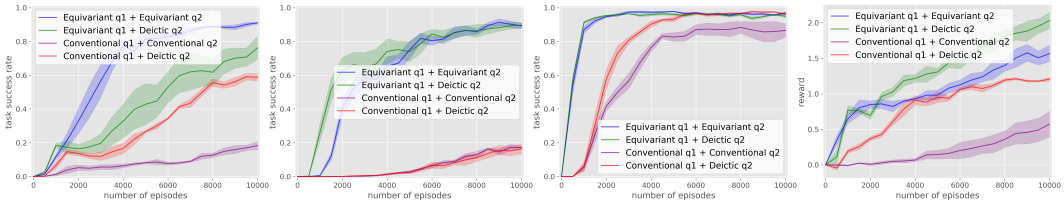
G.4.1 Only Using Equivariant Network in q_1 or q_2

In this ablation study, we evaluate the effect of the equivariant network by only applying it in q_1 or q_2. There are four variations: 1) Equivariant q_1 + Equivariant q_2: both q_1 and q_2 use the equivariant network; 2) Equivariant q_1 + Conventional q_2: q_1 uses the equivariant network, q_2 uses the conventional convolutional network; 3) Conventional q_1 + Equivariant q_2: q_1 uses the conventional convolutional network, q_2 uses the equivariant network; 4) Conventional q_1 + Conventional q_2: both q_1 and q_2 use the conventional convolutional network. The results are shown in Fig 19, where Using the equivariant network in both q_1 and q_2 (blue) always shows the best performance. Note that only applying the equivariant network in q_2 (red) demonstrates a greater improvement compared with only applying the equivariant network in q_1 (green) in three out of four environments. This is because q_2 is responsible for providing the TD target for both q_1 and q_2 [29], which raises its importance in the whole system.



(a) House Building (b) Covid Test (c) Box Palletizing (d) Bin Packing

Figure 20: Comparison of two different symmetry groups for q_1. Results averaged over four runs. Shading denotes standard error.



(a) House Building (b) Covid Test (c) Box Palletizing (d) Bin Packing

Figure 21: Comparison of the deictic encoding and baselines. Results averaged over four runs. Shading denotes standard error.

G.4.2 Symmetry Group in q_1

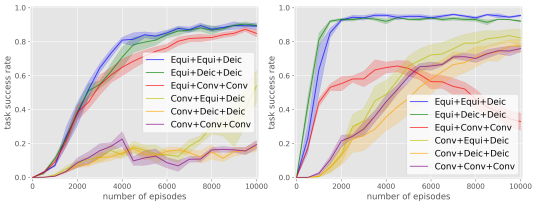
In this experiment, we evaluate two different symmetry groups that q_1 can be defined upon: the Cyclic group C_8 that encodes eight rotations every 45 degrees, and the Dihedral group D_4 that encodes four rotations every 90 degrees and reflection. Both groups have an order of 8, i.e., the network will be equally heavy. As is shown in Fig 20, D_4 has a minor advantage over C_8.

G.4.3 Deictic Encoding in $\mathrm{SE}(2)$

This experiment compares the deictic encoding equipped with the equivariant ASR and the conventional ASR in $\mathrm{SE}(2)$. The comparison is conducted in the following four variations: 1) Equivariant q_1 + Equivariant q_2: both q_1 and q_2 use the equivariant network; 2) Equivariant q_1 + Deictic q_2: q_1 uses the equivariant network, q_2 uses the deictic encoding; 3) Conventional q_1 + Conventional q_2: both q_1 and q_2 use the conventional convolutional network; 4) Conventional q_1 + Deictic q_2: q_1 uses the equivariant network, q_2 uses the deictic encoding. The results are shown in Fig 21. When q_1 is using the equivariant network, using the deictic encoding in q_2 (green) outperforms using equivariant network in q_2 (blue) in Bin Packing, while the equivariant q_2 outperforms in House Building. In Covid Test and Box Palletizing, they tends to have similar performance. When q_1 uses conventional CNN, using deictic encoding in q_2 (red) generally provides a significant performance, compared with using conventional CNN in q_2 (purple). In Covid Test, the use of the deictic encoding does not make a big difference. We suspect that this is because in Covid Test the bottleneck of the whole system is q_1.

G.4.4 Deictic Encoding in $\mathrm{SE}(3)$

This experiment studies the different network choices for q_1 (equivariant network, conventional network), q_2 (equivariant network, deictic encoding, conventional network), and $q_3 - q_5$ (deictic encoding, conventional network). We evaluate two proposed approaches: 1) Equi+Equi+Equi uses the equivariant network in q_1 and q_2 and deictic encoding in q_3 through q_5 (the three components in the name mean the architecture of q_1, q_2, and q_3-q_5); 2) Equi+Deic+Deic uses equivariant network in q_1, and deictic encoding in q_2 through q_5. We compare the proposal with the following baselines: 1) Equi+Conv+Conv uses equivariant network in q_1, and conventional convolutional network in q_2 through q_5; 2) Conv+Equi+Deic uses conventional convolutional network in q_1, equivariant network in q_2, and deictic encoding in q_3 through q_5; 3) Conv+Deic+Deic uses conventional convolutional



(a) Bumpy House Building (b) Bumpy Box Palletizing

Figure 22: Comparison of different network choices in $SE(3)$. Results averaged over four runs. Shading denotes standard error.

method	Conventional FCN	RAD FCN	DrQ FCN	Rot FCN	Equivariant FCN	Equivariant ASR
time(s)	0.08	0.09	0.22	0.42	0.72	0.45

Table 4: The average time for each training step in a rotation space of C_{12}/C_2

method	Conventional ASR	RAD ASR	DrQ ASR	Equivariant ASR
time(s)	0.09	0.14	0.45	0.49

Table 5: The average time for each training step in a rotation space of C_{32}/C_2

network in q_1, and deictic encoding in q_2 through q_5; 4) Conv+Conv+Conv uses conventional convolutional network in q_1 through q_5. The results are shown in Fig 22, where our two proposed approaches outperform the baseline architectures in both environments. Note that swapping the conventional convolutional network with the equivariant network or the deictic encoding generally improves the performance, except that Equi+Conv+Conv in Bumpy Box Palletizing underperforms Conv+Conv+Conv. We suspect that this is because the target of q_1 given by the conventional convolutional networks is less stable.

H Runtime Analysis

Table 4 and Table 5 shows the average runtime in the setting of the experiments in Section 4.2 and Section 4.3, respectively. The runtime is calculated by averaging over 500 training steps on a single Nvidia RTX 2080 Ti GPU. Both the Equivariant FCN and Equivariant ASR requires a longer time for each training step. However, Equivariant ASR is faster and similar to the best performing data augmentation method DrQ.

I Robot Experiment

To ensure better sim to real transfer, we train our model used in the real world with a Perlin noise [42] (with a maximum magnitude of 7mm) applied to the observations.

Bottle Arrangement: In the Bottle Arrangement task, we use the bottles and tray shown in Fig 23 for testing.

House Building: In the House Building task, we train the model with object size randomization within $\pm 8.3\%$. A Gaussian filter is applied after the Perlin noise during training to make the observation noisier. The model is trained for 20k episodes instead of 10k as in the simulation experiment.

Box Palletizing: In the Box Palletizing task, we add an object size randomization within $\pm 3.75\%$ and increase the size of H and P from 24×24 to 40×40.

Fig 24 shows an example episode of the robot finishing the Bottle Arrangement task. Fig 25 shows an example episode of the robot finishing the Box Palletizing task.

Figure 23: The bottles used in the Bottle Arrangement robot experiment.

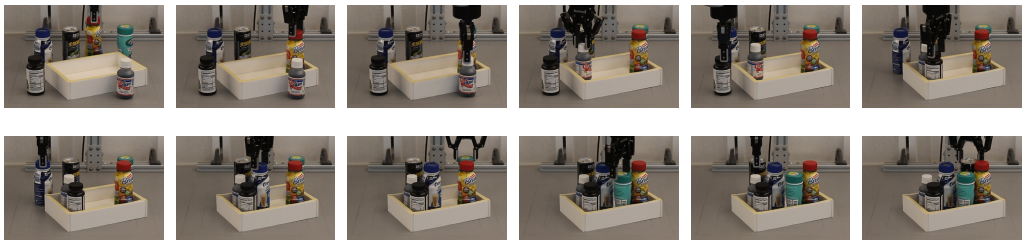


Figure 24: An example episode of the Bottle Arrangement in the real world.

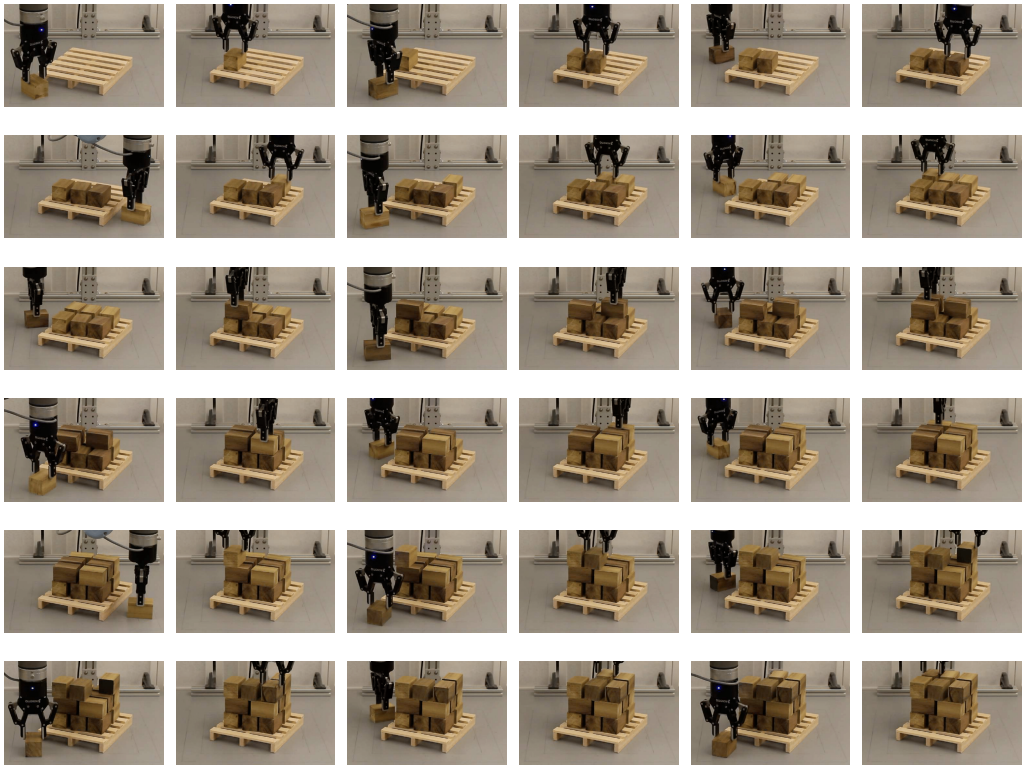


Figure 25: An example episode of the Box Palletizing in the real world.