A Illustration of Group and Group Representation in CarFlag-2D

Domain We consider a small version of `CarFlag-2D` (see Figure 9) with a grid size of 3x3, where the agent (red) must navigate to an unknown target cell (green) in a grid world. The agent can always observe its current location but only observe the target cell when it visits the information cell (blue), which is also unknown to the agent.

Observation The observation is a two-channel image size 2x3x3, where the first channel encodes the agent's location and the second encodes the target location. The values of the second channel are non-zero only when the agent is at the information cell (Figure 9).

Actions Movements in four directions (the location does not change if going out of the world).

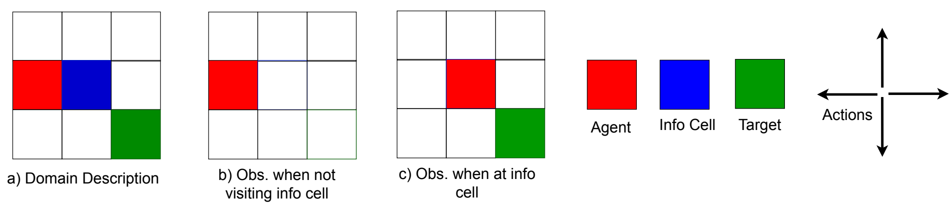


Figure 9: Illustration of the domain, action, and observation.

Domain Symmetry Consider Scenario 1 and Scenario 2 in Figure 10: Scenario 2 is the rotated version of Scenario 1 after a 90° counter-clockwise (CCW) rotation. Therefore, an optimal path (denoted with colored arrows) in Scenario 1 is equally optimal in Scenario 2 if we rotate the path similarly. The same happens if the rotation angle is 180° or 270°. We can capture the rotational symmetry using group $C_4 = \{0°, 90°, 180°, 270°\}$.

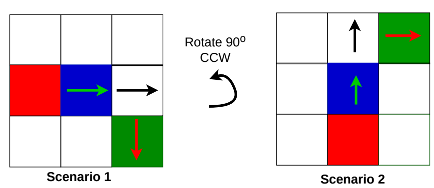


Figure 10: Illustration of domain symmetry. Scenario 2 is the rotated version of Scenario 1 after a 90° counter-clockwise rotation. An optimal path in Scenario 1 can be rotated similarly to become optimal in Scenario 2.

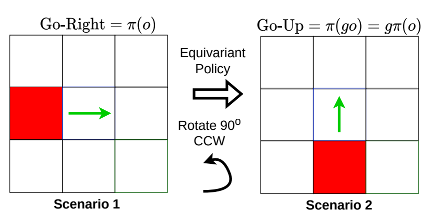


Figure 11: Illustration of the effect of an equivariant policy: the action is automatically rotated when the input observation is rotated.

Equivariant Policy We want our policy to automatically capture the domain symmetry above by making it equivariant. In Figure 11, we illustrate the property of an equivariant policy π with g being a 90° CCW rotation. In Scenario 1, given the first observation o, we assume that π already knows it should go to the right $\texttt{Go-Right} = \pi(o)$ towards the information cell. Now, moving to Scenario 2, when the first observation is the rotated version of o, denoted as go. An equivariant policy automatically calculates the next action in Scenario 2 as:

$$\pi(go) = g\pi(o) = g(\texttt{Go-Right}) = \texttt{Go-Up} \tag{4}$$

Group Representation From Equation (4), to construct an equivariant policy, we need to define how a rotation g acts on an observation at the input (i.e., define go) and on an action at the output (i.e., define $g\pi(o)$). For that purpose, besides defining the group, we need to specify the group representation, i.e., defining an *observation group representation* ρ_o and an *action group representation* ρ_a for π (see below).

Example of Group Acting on Observation and Action The effect of a 90^0 CCW rotation g on the observation via a *trivial* representation $\rho_o = \rho_t$ and the action via a *regular* representation $\rho_a = \rho_r$ is illustrated in Figure 12. A trivial representation ρ_t rotates the observation (like rotating the normal image) while keeping the pixel values unchanged (the value in cell 0 is still $(0_0, 0_1)$). In contrast, a regular representation ρ_r permutes the action distribution output, resulting in a different action, i.e., `Go-Right` $\rightarrow$ `Go-Up`). This automatic change only happens if the policy is equivariant.

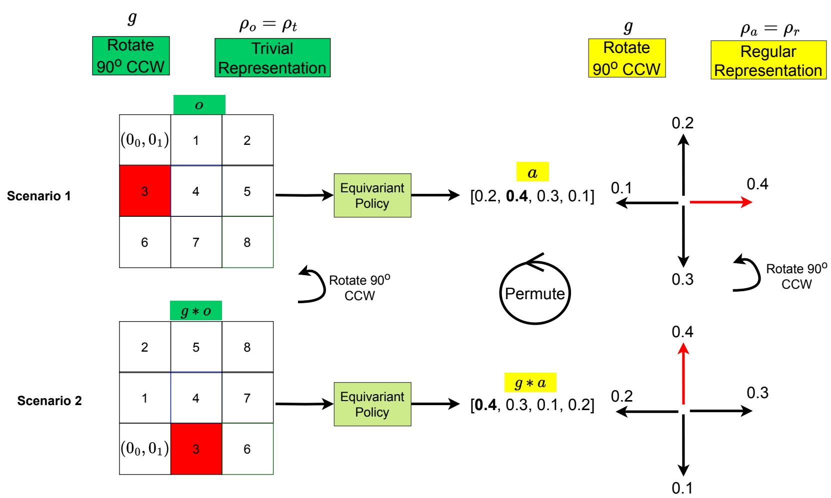


Figure 12: Illustration of the effect of a trivial representation acting on the observation ($\rho_o = \rho_t$) and a regular representation acting on the action ($\rho_a = \rho_r$) with g being a 90^0 CCW rotation. ρ_o rotates the observation and keeps the pixel values unchanged. ρ_a permutes the action distribution output, resulting in a different action (`Go-Right` $\rightarrow$ `Go-Up`).

B Proof of Theorem 1

In this section, we introduce the framework of history representation MDP [63] and prove a supporting lemma before arriving at the proof.

B.1 History Representation MDP

A POMDP can be converted into a *history representation MDP* (HR-MDP) [63] whose state is a sufficient statistic of the POMDP history for control, e.g., the well-known *Belief-MDP* [64] construct is a special case of an HR-MDP based on the belief representation. Useful representations such as the belief might require a known POMDP model; however, we adopt a model-free approach with no such knowledge and therefore use the trivial identity representation whereby the history is represented by itself. This effectively converts the POMDP into an equivalent *History-MDP*, which is defined by the tuple $(\mathcal{H}, \mathcal{A}, \bar{T}, \bar{R})$, where:

$$\bar{T}(h, a, h') = \mathbb{E}_{o|h,a}\left[\mathbb{I}\{h' = hao\}\right] \quad \bar{R}(h, a) = \mathbb{E}_{s|h}\left[R(s, a)\right] \,, \tag{5}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function, and

$$\Pr(o \mid h, a) = \mathbb{E}_{s|h}\left[\sum_{s'} T(s, a, s')O(a, s', o)\right] \,, \tag{6}$$

$$\Pr(s' \mid h') \propto \mathbb{E}_{s|h}\left[T(s, a, s')\right] O(a, s', o) \,. \tag{7}$$

B.2 Supporting Lemma

Lemma 1. *The belief function of a group-invariant POMDP (as defined by Definition 1) is group-invariant,*

$$\Pr(gs \mid gh) = \Pr(s \mid h) \,. \tag{8}$$

Proof By Induction.
Base Case. We first prove that the belief after the first observation is invariant. We note here that the observation function for the first timestep takes the form $O(s, o)$, with no preceding action.

$$\Pr(gs_0 \mid go_0) \propto b_0(gs_0)O(gs_0, go_0) = b_0(s_0)O(s_0, o_0) \propto \Pr(s_0 \mid o_0) \,. \tag{9}$$

Since $\Pr(gs_0 \mid go_0)$ and $\Pr(s_0 \mid o_0)$ are both proportional to the same quantity, and they are both normalized to be distributions over states, then they are themselves equal.

Inductive Step. We then prove that if $\Pr(s_t \mid h_t)$ is invariant, then $\Pr(s_{t+1} \mid h_{t+1})$ is also invariant. Per Equation (7),

$$\begin{aligned} \Pr(gs_{t+1} \mid gh_{t+1}) &\propto \Pr(gs_t \mid gh_t)T(gs_t, ga_t, gs_{t+1})O(ga_t, gs_{t+1}, go_{t+1}) \\ &= \Pr(s_t \mid h_t)T(s_t, a_t, s_{t+1})O(a_t, s_{t+1}, o_{t+1}) \propto \Pr(s_{t+1} \mid h_{t+1}) \,. \end{aligned} \tag{10}$$

Since $\Pr(gs_{t+1} \mid gh_{t+1})$ and $\Pr(s_{t+1} \mid h_{t+1})$ are both proportional to the same quantity, and they are both normalized to be distributions over states, then they are themselves equal. By induction, given the base case and the inductive step, the belief function $\Pr(s_t \mid h_t)$ is invariant for any t. □

B.3 Proof

Proof. We begin by constructing the History-MDP associated with a group-invariant POMDP, and showing that it is itself a group-invariant MDP. The transition and reward functions of the History-MDP are shown in Equation (5) and satisfy the group invariance properties.

For this proof, it is simpler to express the history transition function as $\bar{T}(h, a, h') = \Pr(o \mid h, a)$, where o is the observation (if any exists) s.t. $h' = hao$. If no such observation exists, then $\bar{T}(h, a, h') = 0$ is trivially invariant. If it does exist, then it is necessarily the last observation of h',

$$\bar{T}(gh, ga, gh') = \Pr(go \mid gh, ga) = \sum_{s,s'} \Pr(s \mid gh) \Pr(s' \mid s, ga) \Pr(go \mid ga, s')$$

since g permutes the elements of $\mathcal{S}$, we can re-index using $s = g\bar{s}$ and $s' = g\bar{s}'$,

$$
\begin{aligned}
&= \sum_{\bar{s},\bar{s}'} \Pr(g\bar{s} \mid gh) T(g\bar{s}' \mid g\bar{s}, ga) O(go \mid ga, g\bar{s}') \\
&= \sum_{s,s'} \Pr(s \mid h) T(s' \mid s, a) O(o \mid a, s') = \bar{T}(h, a, h') \,. \qquad (11)
\end{aligned}
$$

By using $s = g\bar{s}$, we proceed similarly for history rewards,

$$
\begin{aligned}
\bar{R}(gh, ga) &= \sum_{s} \Pr(s \mid gh) R(s, ga) = \sum_{\bar{s}} \Pr(g\bar{s} \mid gh) R(g\bar{s}, ga) \\
&= \sum_{s} \Pr(s \mid h) R(s, a) = \bar{R}(h, a) \,. \qquad (12)
\end{aligned}
$$

Therefore, $\bar{T}(h, a, h')$ and $\bar{R}(h, a)$ are invariant, and History-MDPs are group-invariant MDPs. By the theory developed in [1], this implies that the optimal Q-value function $Q^*(h, a)$ is invariant and that there exists at least one equivariant deterministic optimal policy $\pi^*(h)$. Moreover,

$$
\begin{aligned}
V^*(gh) &= Q^*(gh, \pi^*(gh)) = Q^*(gh, g\pi^*(h)) \\
&= Q^*(h, \pi^*(h)) = V^*(h) \,, \qquad (13)
\end{aligned}
$$

this ends our proof by showing that $V^*(h)$ is invariant. $\square$

C Environment Details

C.1 Grid-world Domains

C.1.1 `CarFlag-1D`

- Action: Go-Left or Go-Right
- Observation (Discrete): The position of the car, the side of the green flag (-1 or 1 if the car is at the blue flag, and 0 otherwise)
- Reward: step reward: -0.01, reaching the green flag: 1.0, and reaching the red flag: -1.0
- Episode Initialization: The car is randomized such that it is not at the information location (blue flag). The goal (green flag) is always either at the leftmost or rightmost end. The red flag is on the opposite end
- Episode Termination: Reaching either flags or an episode lasts more than 50 timesteps
- World size: The distance between the red and the green flag is 50

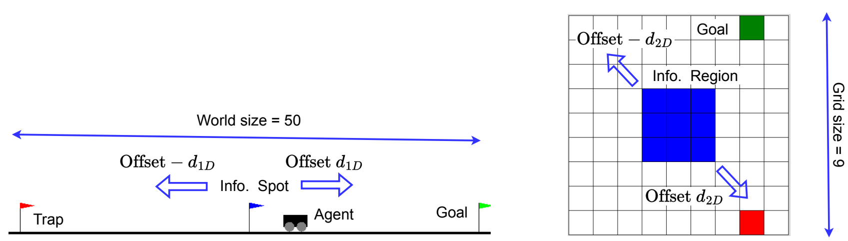


Figure 13: `CarFlag-1D` and `CarFlag-2D` domains. The information regions are not visible to the agent. These domains become asymmetric when the offsets from the information region to the world center, i.e., d_{1D} and d_{2D}, are non-zero.

C.1.2 `CarFlag-2D`

- Action: Right/Left/Up/Down
- Observation: The observation is encoded as an $N \times N \times 2$ image, where N is the grid size, the first channel encodes the position of the car, and the second channel encodes the position of the green cell. The second channel is only informative when the agent is inside the information region (blue)
- Reward: Reaching the green cell: 1.0, otherwise 0.0
- Episode Initialization: The agent and the goal cell are randomized such that the minimum distance between them is at least two steps. Moreover, both the agent and the goal are not initialized inside the information region (blue)
- Episode Termination: Reached the goal or an episode lasts more than 50 timesteps

C.2 Robot Manipulation Domains

For these domains, an episode is terminated when it lasts over 50 timesteps or the task is achieved. Because all robot domains share the same observation and action, we only describe them below.

Action. An action $a = (\delta_w, \delta_x, \delta_y, \delta_z, \delta_r)$, where $\delta_w \in [0, 1]$ is the absolute openness of the gripper (0: fully open, 1: fully closed), $\delta_{x,y,z} \in [-0.05, 0.05]$ are the displacements of the gripper in the X, Y, and Z axis, and $\delta_r \in [-\pi/8, \pi/8]$ is the angular rotation around the Z axis (see Figure 14a)

Observation. An observation is a top-down depth image taken from a camera located at the end-effector. Specifically, an observation $o = (I, k)$, where $I \in \mathbb{R}^{84 \times 84}$ is the depth image and $k \in \{1, 0\}$ indicates the current holding status of the gripper. I and k are combined to create a unified depth observation $o \in \mathbb{R}^{2 \times 84 \times 84}$. Moreover, two fingers of the gripper are also projected on I (black squares in Figure 14b)

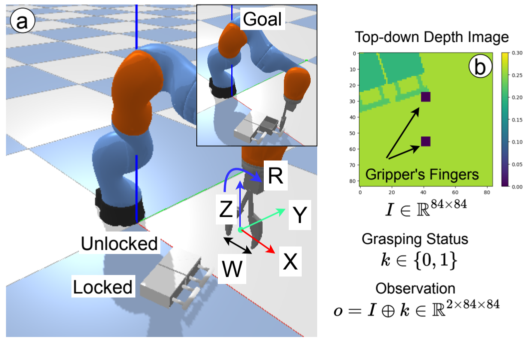
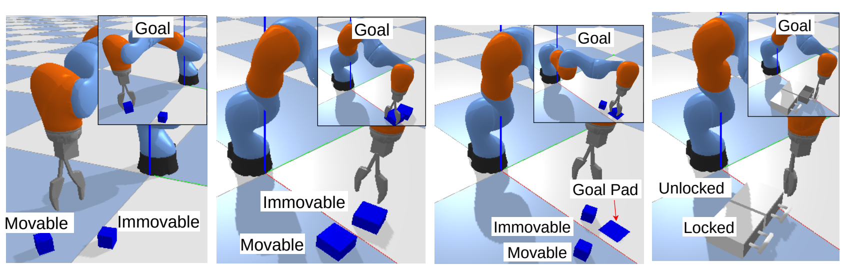


Figure 14: Visual description of `Drawer-Opening`.

Partial Observability. These domains characterize the natural partial observability when certain physical properties of objects, e.g., whether a drawer in Figure 14a is unlocked or not, are often unobservable using pixel observations alone

C.2.1 `Block-Picking`

- Reward: A reward of 1.0 only when the movable block is picked and brought higher than 8cm
- Episode Initialization: The poses of the two blocks are randomized. The arm is initialized at a fixed pose
- Expert Generation: An expert (a planner with access to all object poses) randomly chooses one block to pick. If the expert picks the movable block, it will bring the block up to achieve the task. Otherwise, the expert keeps trying for several timesteps before switching to pick the movable block to achieve the task



`Block-Picking` `Block-Pulling` `Block-Pushing` `Drawer-Opening`

Figure 15: Robot manipulation domains.

C.2.2 `Block-Pulling`

- Reward: A reward of 1.0 only when two blocks are in contact
- Expert Generation: An expert randomly chooses one block to pull towards the other block. If the block is pullable, then it will be pulled towards the other block to achieve the task. Otherwise, the expert keeps trying for a while before pulling the other block

C.2.3 `Block-Pushing`

- Reward: A reward of 1.0 only when the pushable block is within 5cm from the center of the goal pad. The agent additionally receives a penalty of 0.1 per timestep if it changes the height of the movable block by 5mm to prevent picking the block instead of pushing it
- Episode Initialization: The poses of the two blocks and the goal pad are randomly initialized

- Expert Generation: An expert randomly chose one block to push towards the goal pad. If the block is pushable, it will then continue pushing until reaching the goal pad. Otherwise, the expert keeps trying for several timesteps before doing the same thing with the other (pushable) block

C.2.4 `Drawer-Opening`

- Reward: A reward of 1.0 only when the unlocked drawer is opened more than 5cm
- Episode Initialization: Two drawers are randomly placed next to each other with the same heading angle
- Expert Generation: An expert randomly chooses one drawer to open. If it chooses the unlocked drawer, it will then open the drawer to achieve the task. Otherwise, the expert keeps opening the unlocked drawer several timesteps before opening the other drawer

D Implementation Details

D.1 Network Structure of Equivariant Recurrent A2C (Equi-RA2C)

Figure 16 shows the specific architecture of Equi-RA2C used in `CarFlag` domains. Because the actions can be inferred from the observations in these domains, we do not include the feature extractor for the previous actions. We also omit the skip-connections. The input representation is some representation of the observation ρ_o, depending on the domains (see below).

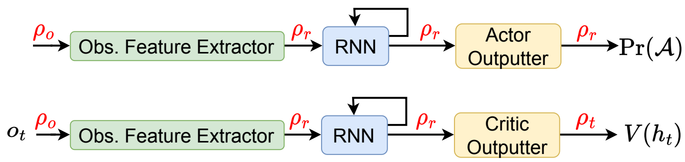


Figure 16: The architecture of Equi-RA2C used in `CarFlag-1D` and `CarFlag-2D`.

Figure 17 shows the details of Equi-RA2C used in `CarFlag-1D` for the `flip2dOnR2` group in the `escnn` [1] [45, 65] library. Notice that the input x_t for the LSTM cell using the *irreducible* representation of the `flip2dOnR2` group denoted as ρ_{irr}. For `CarFlag-1D`, using this representation in the input would negate the signs of every component in x_t, i.e., flipping the positions of the car, the sides of the green flag, and the previous actions in the history. Because the observation in this domain is feature-based, we remove the observation feature extractor and directly feed the observation to the equivariant LSTM.

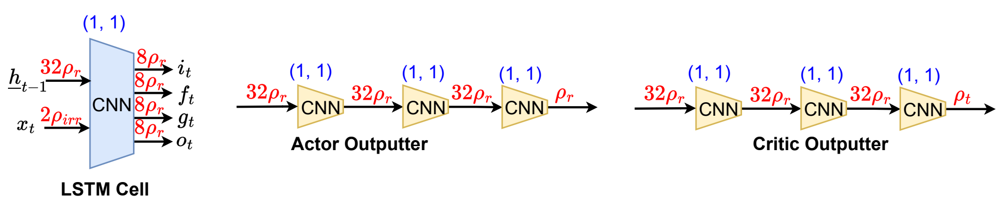


Figure 17: Details of Equi-RA2C used in `CarFlag-1D` for the `flip2dOnR2` group. Numbers inside brackets (blue - on top) denote the value of kernel sizes and strides used for the CNN modules on the bottom. The numbers next to the representations, e.g., $32\rho_r$, denote the number of feature fields.

Figure 18 shows the details of Equi-RA2C used in `CarFlag-2D` for the C_4 group.

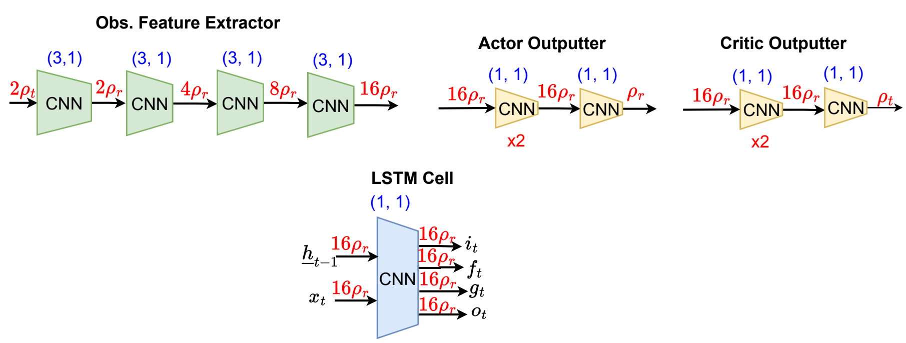


Figure 18: The details of Equi-RA2C used in `CarFlag-2D` for the C_4 group.

D.2 Network Structure of Equivariant Recurrent SAC (Equi-RSAC) with C_4 Group

Figure 19 shows the details of Equi-RSAC used in the robot manipulation domains with the C_4 group. The input representation is *mixed* for the action feature extractor because the action in-

[1] https://github.com/QUVA-Lab/escnn

put has components that transform differently under a rotation. Specifically, given an action $a = (\delta_w, \delta_x, \delta_y, \delta_z, \delta_r)$, the trivial representation ρ_t is chosen for the $\delta_w, \delta_z, \delta_r$ components (which should be unchanged under the rotation). In contrast, the standard representation ρ_s is chosen for the lateral components (δ_x, δ_y), which should rotate. For the same reason, for the actor outputter, ρ_μ is mixed, i.e., the trivial representations ρ_t are used for the w, z, r components, and ρ_s is used for the x, y components.

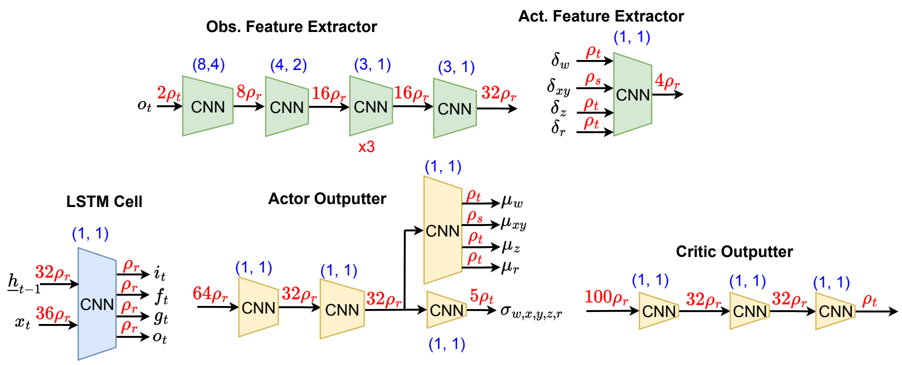


Figure 19: Details of Equi-RSAC with the robot manipulation domains and the C_4 group.

D.3 Implementation Using The ESCNN Library

Given the definition of each equivariant component above, we can easily implement it with `escnn`. For instance, the following PyTorch [66] code defines the observation feature extractor in Figure 18a with ReLU as a non-linearity component:

```
import escnn.nn as enn

# Define group C4
s = escnn.gspaces.rot2dOnR2(4)

# Define in/out representations
repr_i  = enn.FieldType(s,  2*[s.trivial_repr])
repr_m0 = enn.FieldType(s,  2*[s.regular_repr])
repr_m1 = enn.FieldType(s,  4*[s.regular_repr])
repr_m2 = enn.FieldType(s,  8*[s.regular_repr])
repr_o  = enn.FieldType(s, 16*[s.regular_repr])

obs_feature_extractor = enn.SequentialModule(
          enn.R2Conv(repr_i, repr_m0, 3, 1),
          enn.ReLU(repr_m0),
          enn.R2Conv(repr_m0, repr_m1, 3, 1),
          enn.ReLU(repr_m1),
          enn.R2Conv(repr_m1, repr_m2, 3, 1),
          enn.ReLU(repr_m2),
          enn.R2Conv(repr_m2, repr_o, 3, 1),
          enn.ReLU(repr_o),
)
```

Implementing the mixed representation is also straightforward by summing different field types. In order to create the actor and the critic, we simply chain components by using the `SequentialModule` as in native PyTorch.

D.4 Training Details

We implement using PyTorch. The batch size for all agents is 32 (episodes). The replay buffer has a capacity of 100,000 transitions. We use the Adam optimizer [67] with a learning rate of 3e-4 for actors and critics and 1e-3 for optimizing α for SAC-based agents. The target entropy $\bar{H}$

for SAC-based agents is -dim($\mathcal{A}$) followed the common practice, and α is initialized at 0.1. After prepopulating the replay buffer with 80 expert episodes, the buffer is filled with 20 episodes with random actions. We use the same 1:1 environment/gradient step ratio for all agents.

D.5 Implementing Equivariant LSTM

We implement the equivariant LSTM [47] based on a public code of ConvLSTM [48] at `https://github.com/Hzzone/Precipitation-Nowcasting` as the authors did not release the official code.

E Baseline Details

RA2C [51] We modified the code at https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail. We used 16 environments in parallel and used recurrent policies. Other hyper-parameters are kept at default.

DPFRL [14] We used the authors' code at https://github.com/Yusufma03/DPFRL. We used 30 particles, MGF particle aggregation type, and the hidden dimension is 128.

RAD [54] We collected depth images of size 90x90 to perform random cropping to reduce the size to 84x84. We perform the same type of random cropping for every depth image within an episode.

DrQ [55] We used random shift of ± 4 pixels as suggested by the original work. The same type of shifting is used for every depth image within a sequence. We also followed the authors' suggestions when using the numbers of augmentations for calculating the Q-targets and the Q-values are $K = 2$ and $M = 2$, respectively.

SLAC [56] We used a Pytorch implementation at https://github.com/toshikwa/slac.pytorch, which has been benchmarked against the performance reported in the original paper. We pre-train the latent variable model for 2k steps before iterating between data collection, model update, and evaluation. We also pre-fill the replay buffer with the same number of expert and random episodes before training and use four extra augmented episodes for each episode during training to ensure a fair comparison. The sequence length is extended from 8 (originally) to 50 (maximum episode length). We varied the sequence length for better performance, but the performance did not improve much. For any episode shorter than 50 steps, we zero-pad dummy transitions *in front*.

DreamerV2 [52] We used the official code at https://github.com/danijar/dreamerv2. For `CarFlag` domains, we mainly keep the default hyper-parameters (suggested by the authors). In `CarFlag-2D`, the observation image is extended to have the size of 64×64×3 by zero-padding around the original image and is added with a dummy channel (all zero).

DreamerV3 [53] We used the official code at https://github.com/danijar/dreamerv3 and performed similar steps like in the case of DreamerV2. We used the *small* world models with about 18M trainable parameters (predefined in the repo's configuration file) for our `CarFlag` domains.

F Visualization of SLAC Reconstructed Images

Figure 20 shows the comparison between the depth images produced by the trained latent model of SLAC [56] (top row) and the ground-truth ones (bottom row) in `Block-Pulling` after 40k training steps. It can be seen that small squares representing the gripper have been reconstructed quite well, but the model fails to reconstruct the two blocks representing the gripper's position in the scene.

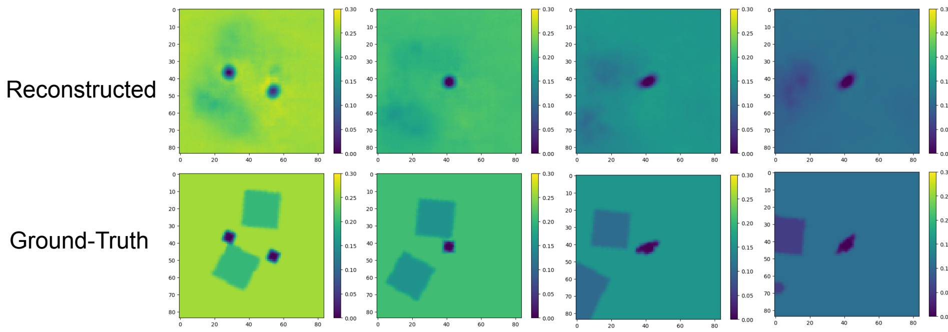


Figure 20: Images reconstructed by the latent model of SLAC [56] in `Block-Pulling`: reconstructed (top row), ground-truth (bottom row).

G Visualization of Data Augmentations

We show visualizations of different ways for augmenting the observations withing a training sequence in `Drawer-Opening`: random rotation (Figure 21), random crop (Figure 22), and random shift (Figure 23). Note that the same operation (rotation/crop/shift) is applied similarly to every observation in an episode. For each training episode, we perform this augmentation four times to generate four auxiliary episodes.

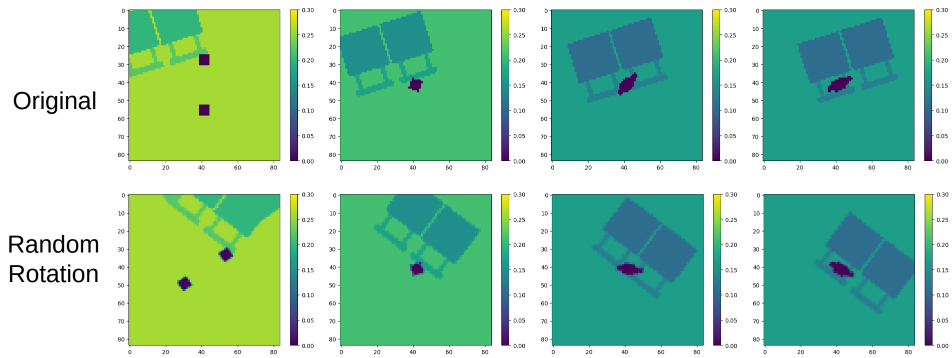


Figure 21: Visualization of randomly rotated augmentations in `Drawer-Opening`: original observations (top row), randomly rotated observations (bottom row).

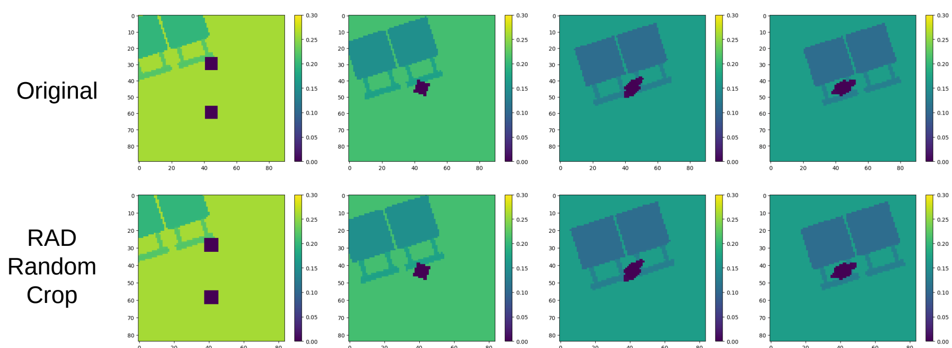


Figure 22: Visualization of randomly cropped augmentation for RAD [54] in `Drawer-Opening`: original observations (top row), randomly cropped observations (bottom row).

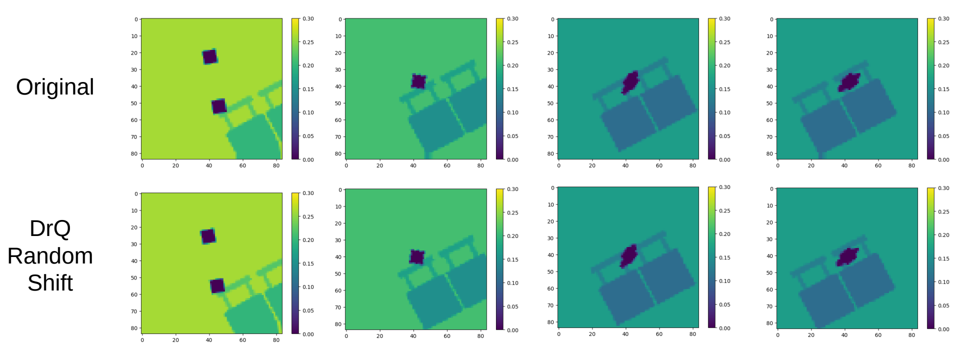


Figure 23: Visualization of randomly shifted augmentation for DrQ [55] in `Drawer-Opening`: original observations (top row), randomly shifted observations (bottom row).

H Ablation Studies

H.1 Equivariant Actor or Critic Only

In Figure 24, we additionally show the learning performance when only either actor or critic is equivariant in `Block-Pushing` and `Drawer-Opening`. From the figure, having an equivariant critic (purple) is more beneficial than having an equivariant actor (blue). However, having both being equivariant (green) yields the best performance.

(a) `Block-Picking`

(b) `Block-Pulling`

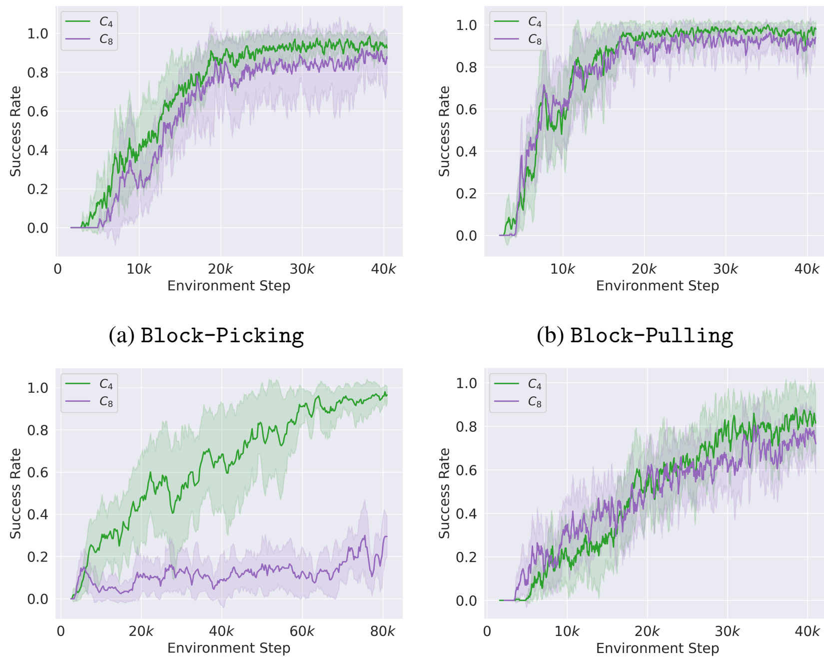
(c) `Block-Pushing`

(d) `Drawer-Opening`

Figure 24: Comparing the effect of only using equivariant actor or critic.

H.2 Different Symmetry Groups

Figure 25 shows the performance when the C_4 and C_8 symmetry groups in the robot manipulation domains. Using C_4 is much better than using C_8 in `Block-Pushing`, but the two groups perform similarly in the remaining domains. Furthermore, it is possible to use other group symmetries which extend C_n with reflection, such as the dihedral groups D_4 or D_8.



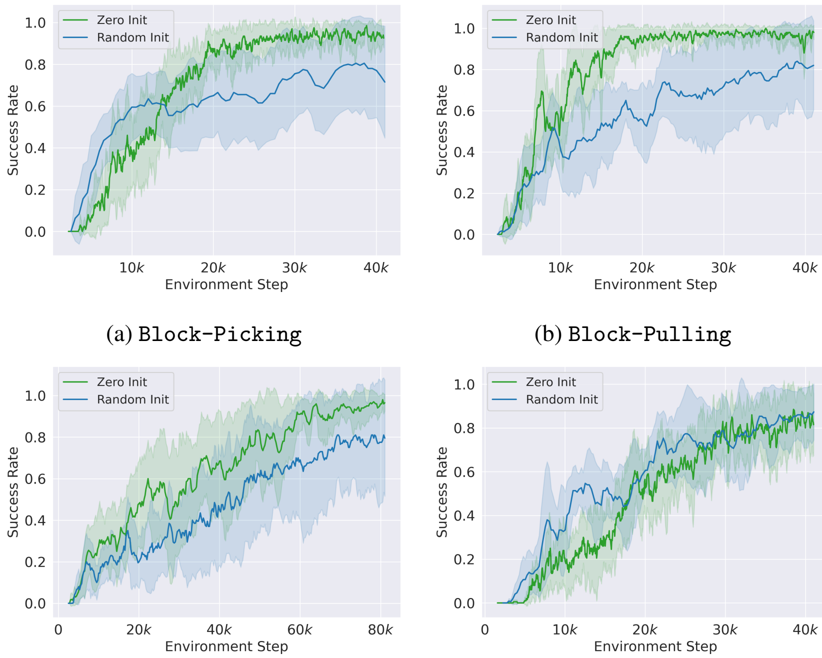
(a) `Block-Picking` (b) `Block-Pulling`

(c) `Block-Pushing` (d) `Drawer-Opening`

Figure 25: Comparing the effect of using symmetry groups C_4 and C_8.

H.3 Randomly Initialized Cell and Hidden States of Equivariant LSTM

Figure 26 shows the performance when the equivariant LSTM is initialized with random instead of zero cell and hidden states. Random initialization results in a worse performance because the equivariance of the actor and the critic is broken. However, our method is generally robust to this change when the performance is still better than the baselines.



(a) `Block-Picking` (b) `Block-Pulling`

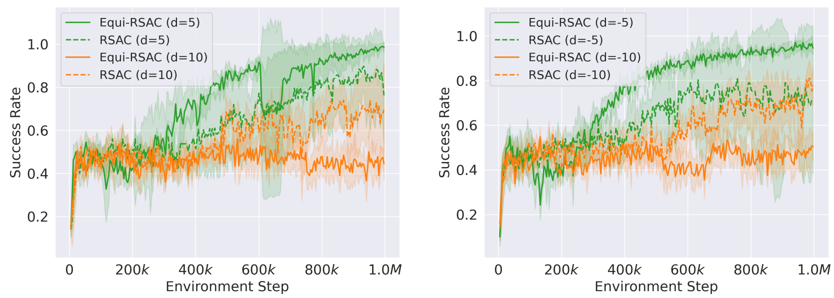
(c) `Block-Pushing` (d) `Drawer-Opening`

Figure 26: Comparing the performance when initializing the cell and hidden states of the equivariant LSTM with zero and random values. Random initialization results in a worse performance because the actor and the critic's equivariance is broken.

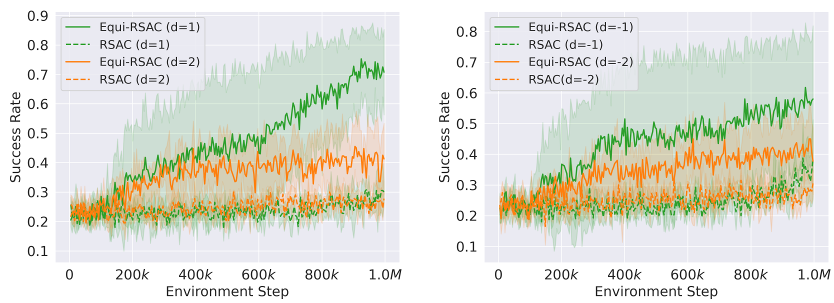
I Additional Experimental Results

I.1 Performance in Asymmetric `CarFlag` Domains

Figure 27 shows the evaluation success rates in asymmetric variants of `CarFlag` domains with different offsets.



(a) `Asym-CarFlag-1D` w/ positive offsets (b) `Asym-CarFlag-1D` w/ negative offsets

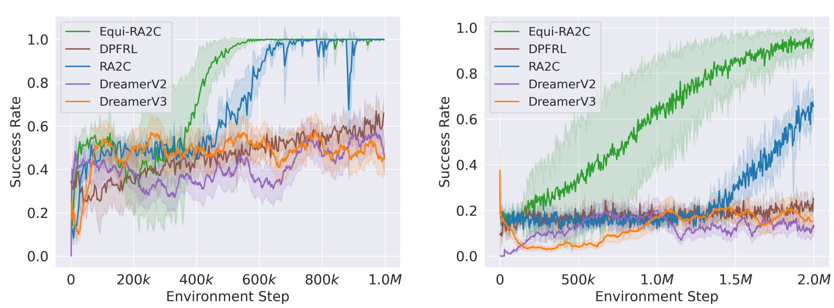


(c) `Asym-CarFlag-2D` w/ positive offsets (d) `Asym-CarFlag-2D` w/ negative offsets

Figure 27: Learning performance with asymmetric version of `CarFlag` domains.

I.2 Performance in Variants of `CarFlag` Domains

Figure 28 show the evaluation success rates in different variants of `CarFlag` domains with a different world size and grid size. Our equivariant agent still outperforms other baselines.

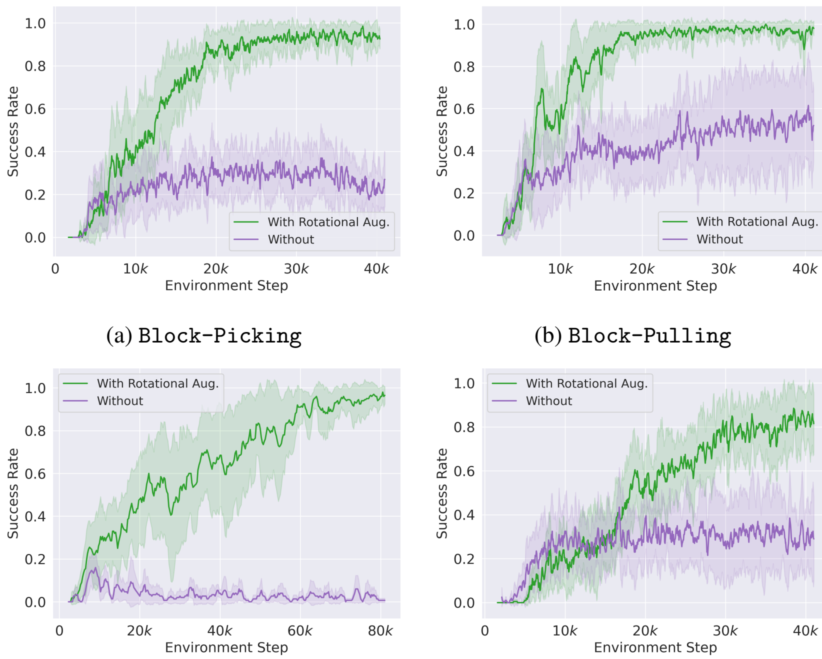


(a) `CarFlag-1D` with world size 40 (b) `CarFlag-2D` with grid size 11

Figure 28: Learning performance in `CarFlag` domains with different sizes.

I.3 Effect of Rotational Augmentation

Figure 29 shows that including rotational augmented episodes significantly improves the learning performance of equivariant agents. These rotational augmented episodes possibly help equivariant agents distinguish different discrete rotations within a group, thus boosting performance.



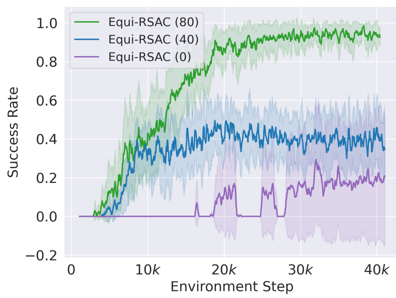
(a) `Block-Picking` (b) `Block-Pulling`

(c) `Block-Pushing` (d) `Drawer-Opening`

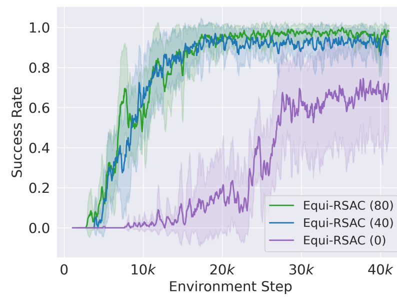
Figure 29: Comparing the performance of our equivariant agents when using/not using rotational augmentation episodes.

I.4 Effect of Number of Demonstration Episodes

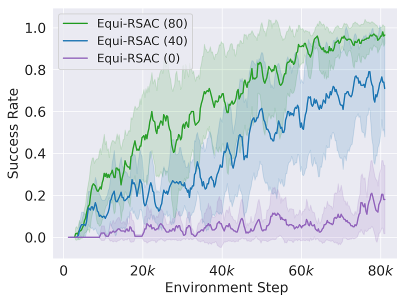
Figure 30 shows that the performance improves when using more demonstrations in all domains, as expected.



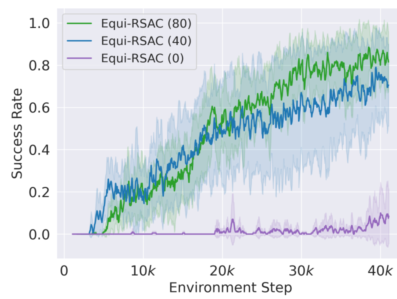
(a) `Block-Picking`



(b) `Block-Pulling`



(c) `Block-Pushing`



(d) `Drawer-Opening`

Figure 30: Using different numbers of demonstration episodes.