

378	A Details of Agents	13
379	A.1 DTQN, TSAC	13
380	A.2 ZP-DRQN, ZP-RSAC	13
381	A.3 BA-DTQN, BA-TSAC	13
382	A.4 UA-DTQN, UA-TSAC	14
383	A.5 B-DQN, B-SAC	14
384	A.6 Hyper-parameters	15
385	A.7 Network Structures	16
386	B Details of Domains	18
387	B.1 Sphinx	18
388	B.2 CarFlag-2D	18
389	B.3 Heaven-Hell	19
390	B.4 Robot Domains	19
391	C Representations Training Details	21
392	C.1 Training Data Generation	21
393	C.2 Network Architecture	21
394	C.3 Mutual Information Estimation	21
395	C.3.1 Minimizing $I(z^s; z^o)$	21
396	C.3.2 Maximizing $I(o; z^o)$ and $I(s; z^s)$	23
397	C.4 Hyper-parameters	23
398	D Additional Experiments	24
399	D.1 Using $z^s \oplus z^o$ versus z^s for Task Learning	24
400	D.2 Using Only Auxiliary Task/Intrinsic Rewards	24
401	D.3 Using GRU v.s. GPT	25
402	D.4 Visualization of Intrinsic Rewards	25
403	E Details of Hardware Experiments	25
404	E.1 Obtaining Depth Images	25
405	E.2 Added Perlin Noise for Better Sim-To-Real Transfers	26
406	F Details of SO(2) Rotational Augmentation	26

407 **A Details of Agents**

408 **A.1 DTQN, TSAC**

409 These are variants of DQN and SAC, made memory-based by using a transformer as the sequence model as shown in Fig. 8 and Fig. 9. Similar models have been explored in recent work [21, 22].

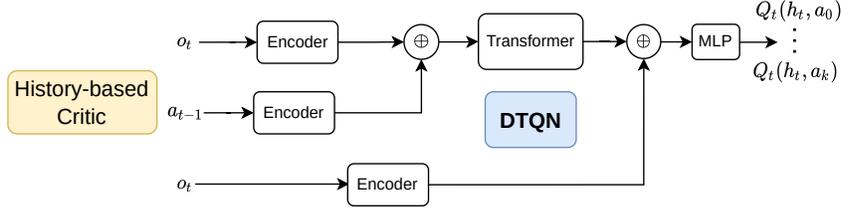


Figure 8: Architecture of DTQN.

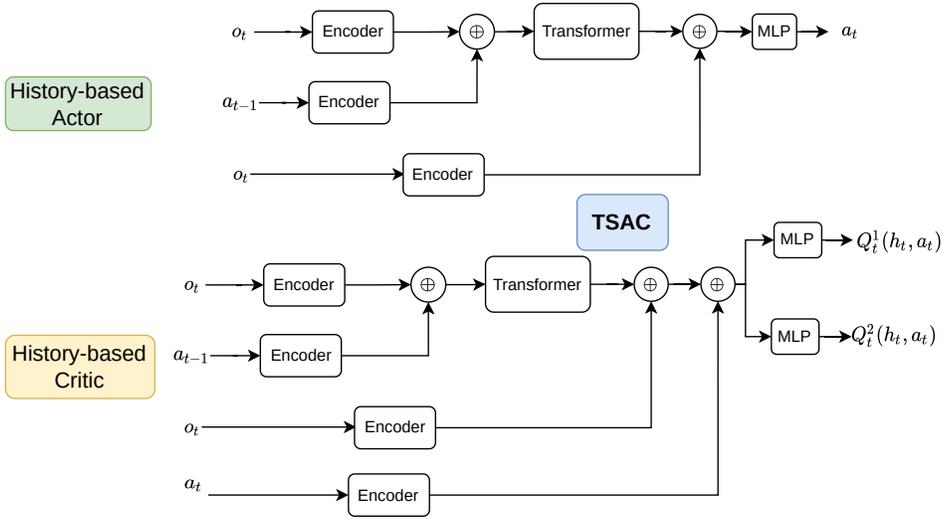


Figure 9: Architecture of TSAC.

410

411 **A.2 ZP-DRQN, ZP-RSAC**

412 These agents [41] are similar to DTQN and TSAC, except they use a recurrent sequence model instead of a transformer. Importantly, using a *recurrent* sequence model (e.g., a GRU [42]) is required (see [41]). Additionally, these agents are regularized with a self-predictive auxiliary task of predicting the next latent state z from a history h . Specifically, given a recurrent encoder $f_\phi : \mathcal{H} \rightarrow \mathcal{Z}$ and a latent dynamics model $g_\theta : \mathcal{Z} \times \mathcal{A} \rightarrow \mathcal{Z}$, the auxiliary task is to minimize:

$$\mathcal{L}_{\text{aux}} = \|g_\theta(f_\phi(h), a) - f_{\bar{\phi}}(h')\|_2^2, \quad (6)$$

417 where $\bar{\phi}$ is the target network of ϕ .

418 **A.3 BA-DTQN, BA-TSAC**

419 In BA-DTQN [6] (see Fig. 10), a state-based critic $Q(s, a)$ and a history-based critic $Q(h, a)$ are learned to leverage the state availability during training but not during execution (i.e., we cannot use $Q(s, a)$ during execution). Unfortunately, as in [6], $Q(s, a)$ is not mathematically well-defined and generally a biased estimate of $Q(h, a)$, which is used to select actions during execution.

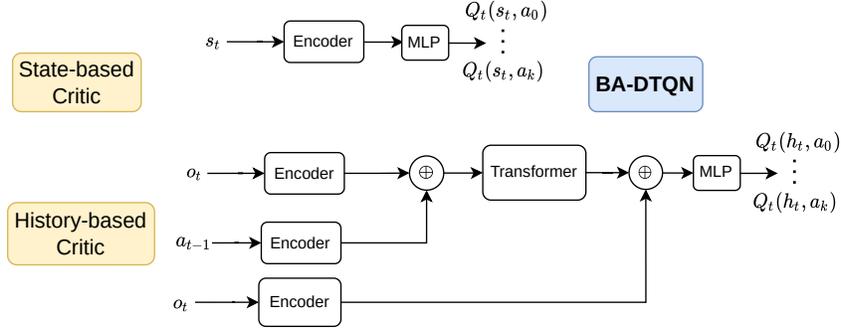


Figure 10: Architecture of BA-DTQN.

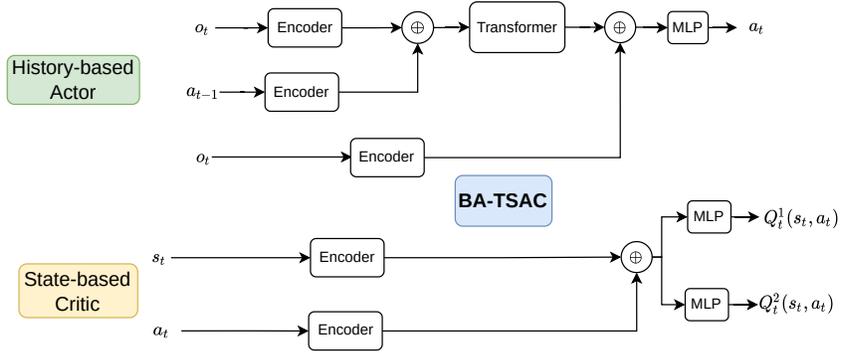


Figure 11: Architecture of BA-TSAC.

423 The difference between BA-TSAC (see Fig. 11) and TSAC is that the critic is trained additionally
 424 using state input during training. Specifically, we learn a state-based critic $Q(s, a)$ instead of the
 425 history-based $Q(h, a)$. Similar to BA-DTQN, BA-TSAC also has bias. For BA-TSAC, during
 426 execution, actions are computed using a history-based actor.

427 A.4 UA-DTQN, UA-TSAC

428 In UA-DTQN [6] (see Fig. 12), a history-state-based critic $Q(h, s, a)$ and a history-based critic
 429 $Q(h, a)$ are learned. Unlike BA-DTQN with $Q(s, a)$, $Q(s, h, a)$ can be well-defined and has been
 430 proven to be an unbiased estimate of $Q(h, a)$. During execution, actions are selected from $Q(h, a)$.

431 Unlike BA-TSAC, UA-TSAC [9] (see Fig. 13) combines *both* state and history features to train the
 432 critic, i.e., we learn a history-state-based critic $Q(s, h, a)$. Similar to UA-DTQN, UA-TSAC does
 433 not introduce learning bias. During execution, actions are computed from a history-based policy.

434 A.5 B-DQN, B-SAC

435 The architectures of these agents are depicted in Fig. 14. These agents are based on Believer [8],
 436 which leveraged the state availability to train an agent in three stages:

437 **Stage 1.** Learning compact state representations with state-labeled transitions, i.e., a batch of sam-
 438 ples (s, o, a, r, s', o') . This stage is similar to our first stage (see Algorithm 1) but without the
 439 information-based regularizations. Instead, the authors proposed to regularize the KL divergence
 440 $\text{KL}[\phi(s) \parallel \mathcal{N}(0, 1)]$ to avoid overlapping features between $\phi(s)$ and $\psi(o)$ by giving penalty when-
 441 ever $\phi(s)$ is used to derive features. This, however, does not avoid the overlapping issue between
 442 learned state and observation features, as shown in our experiment (see Fig. 4).

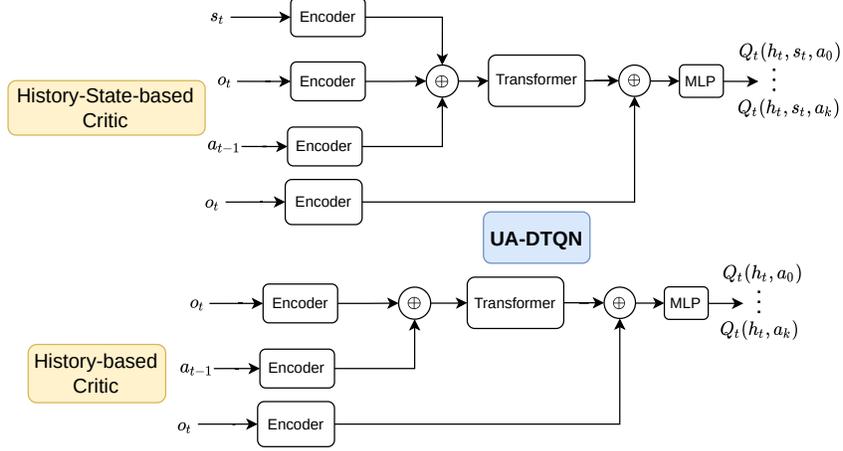


Figure 12: Architecture of UA-DTQN.

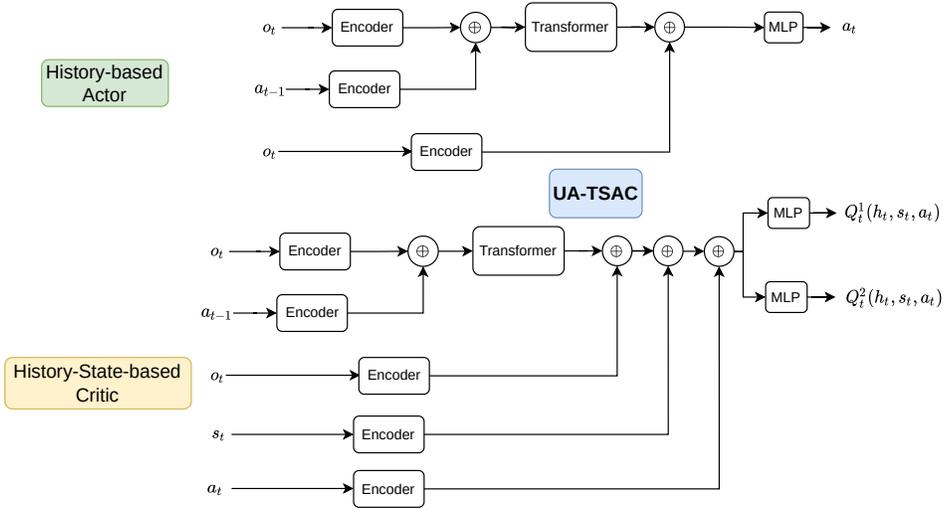


Figure 13: Architecture of UA-TSAC.

443 **Stage 2.** Learn a recurrent history model $p(\phi(s)|h)$ with variational autoencoders [46] (VAE) by
 444 maximizing the joint log-likelihood $p(\phi(s), h)$ averaged over (s, h) samples.

445 **Stage 3.** Use the history module $p(\phi(s)|h)$ for task learning. First, samples are drawn from the
 446 VAE to derive a history summary. Then, this summary is used as the “states” for task learning using
 447 memoryless RL algorithms. The authors optionally fine-tune $p(\phi(s)|h)$ with the on-policy data.

448 As the original paper applied their method for PPO [47], which is on-policy, we had to modify the
 449 method to apply to DQN and SAC, resulting in B-DQN and B-SAC. In Stage 2, to fairly compare
 450 with other baselines, we replace GRU in the history model with the GPT model used in other base-
 451 lines. Moreover, in Stage 3, we fine-tuned the history module for every domain (as used in the
 452 original code). Finally, the sequence model of B-SAC is shared between the actor and the critic,
 453 following the original code.

454 A.6 Hyper-parameters

455 For DDQN [39], we use an epsilon-greedy exploration strategy with a linear schedule, starting at
 456 $\epsilon = 1.0$ and ending at $\epsilon = \frac{1}{T}$ with T being the episode length. The schedule time is equal to 10% of
 457 the total training timesteps. We use a batch size of 64 episodes.

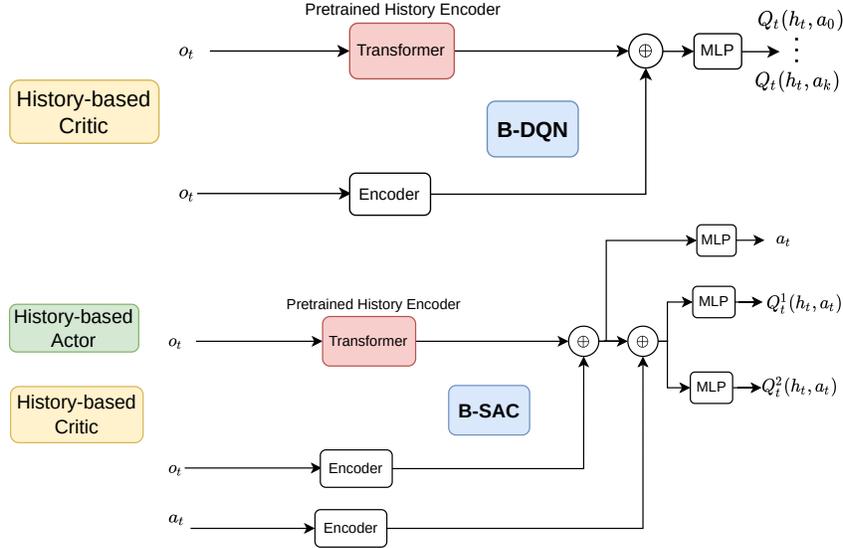


Figure 14: Architectures of B-DQN and B-SAC with a pre-trained history encoder from Believer [8]. We change the history encoder from GRU-based to transformer-based for a fair comparison with other agents. For B-SAC, we use a shared history module, similar to the original code.

458 For continuous actions, we use SAC [40]. We automatically tune the entropy temperature, initializ-
 459 ing at 0.01. The chosen target entropy is equal to the negation of the action dimension. We use the
 460 discount factor $\gamma = 0.99$. We use a batch size of 64 episodes.

461 Other hyper-parameters are in Table 2 with shared parameters and ones specific for each agent.

Table 2: Hyper-parameters used for RL agents. HH: Heaven-Hell, S: Sphinx, CF: CarFlag-2D.

Agent	Hyper-parameter	Value
Shared	Episode Length	50
	Discount Factor	0.99
	Replay Buffer Size	1M: discrete domains, 100k: robot domains
	Target Update Rate	0.005
	Actor Learning Rate	3e-4
	Critic Learning Rate	3e-5: CF and S; 3e-4: other
	Batch size	64
SAC	Initial Entropy Temperature	0.01
	Update Per Step	0.25: discrete domains, 1.0: robot domains
ZP-DRQN	Loss weighting	1.0: discrete domains, 0.1: robot domains
Ours	Loss Weighting	0.5 for all domains
	Reward Weighting	10.0: HH, S, 0.1: robot domains, 0.0: CF
B-DQN, B-SAC	Latent Dimension	32
	X-Dim	16
	Z-Dim	16

462 A.7 Network Structures

463 We use the following acronyms: **FC**(n): a fully connected layer with n outputs; **Conv**(f, s): a
 464 convolutional layer with filter size $f \times f$ and stride s ; **R**: the ReLU activation function; **MaxPool**(w):
 465 a max pooling layer with window size w ; **T**(H, N, HS, D): Transformer with H heads, N layers,
 466 hidden size HS , and the dropout rate D ; **GRU**(N, HS): GRU with N layers and hidden size HS .

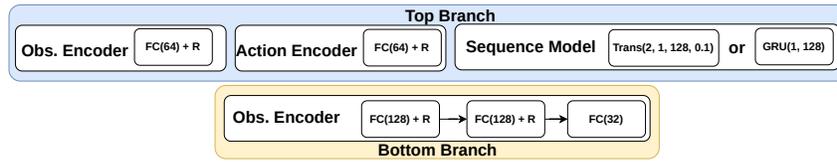


Figure 15: Network structures used in Heaven-Hell.

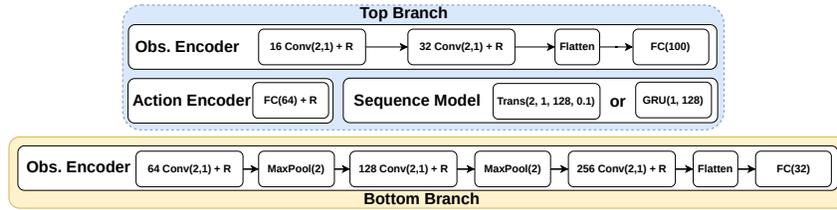


Figure 16: Network structures used in CarFlag-2D.

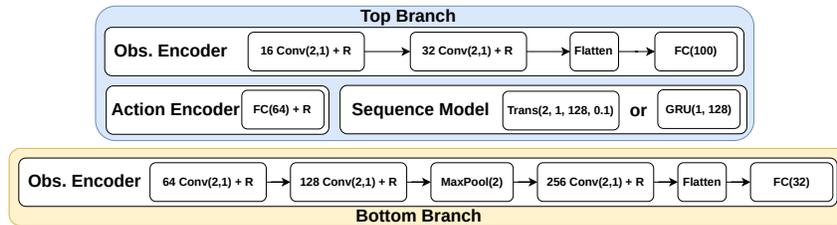


Figure 17: Network structures used in Sphinx.

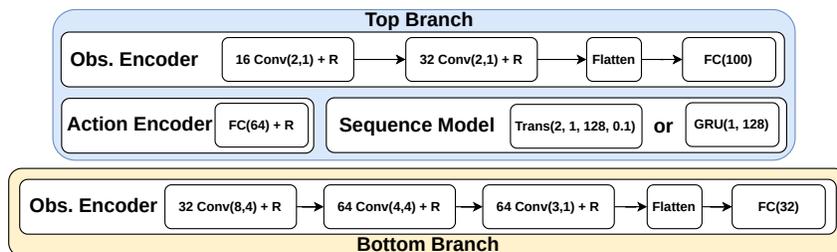


Figure 18: Network structures used in robot domains.

467 **B Details of Domains**

468 **B.1 Sphinx**

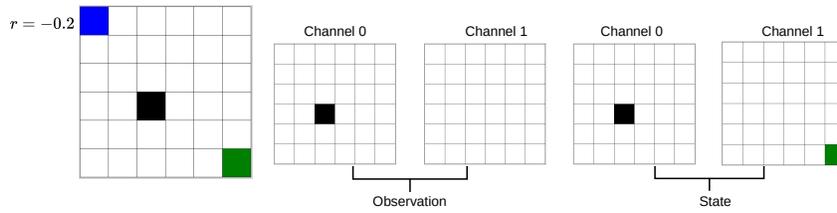


Figure 19: Sphinx domain with two-channeled pixel-based observations and states. Channel 1 of the observation reveals the goal cell (green) only when the agent enters the blue cell. In contrast, the same channel of the state always reveals the goal cell regardless of the agent’s position.

469 In this domain (see Fig. 19), an agent must visit the goal cell, which can be in one of three corners
 470 except the top-left one. The agent must visit the information cell (blue) at the top-left corner to know
 471 the current corner of the goal. However, there is a cost when going to the information cell.

472 **Action.** Move-Right, Move-Left, Move-Up, Move-Down

473 **Observation.** A $6 \times 6 \times 2$ image with the first channel encodes the agent’s position and the second
 474 encodes the goal’s position. The second channel only contains the goal information when the agent
 475 enters the blue cell.

476 **State.** A state has the same structure as an observation, but the second channel always contains the
 477 goal information.

478 **Reward.** +1 when reaching the goal, -0.2 when visiting the information cell, and 0 otherwise.

479 **B.2 CarFlag-2D**

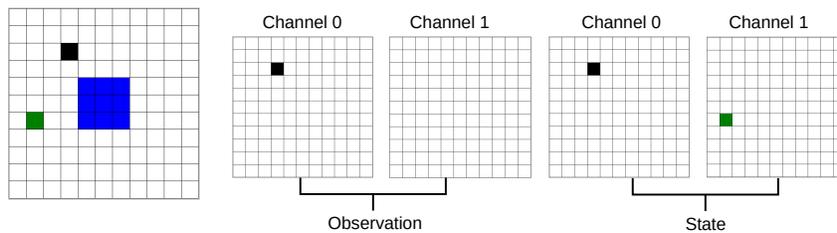


Figure 20: CarFlag-2D domain with two-channeled pixel-based observations and states. Channel 1 of the observation reveals the goal cell only when the agent enters the blue region. In contrast, the same channel of the state always reveals the goal cell.

480 In this domain (see Fig. 20), an agent must visit the goal cell (green) to finish the task. The goal cell,
 481 however, is only present in the observation when the agent visits the information region (blue).

482 **Action.** Move-Right, Move-Left, Move-Up, Move-Down

483 **Observation.** A $11 \times 11 \times 2$ image with the first channel encodes the agent’s position, and the
 484 second encodes the goal’s position. The second channel only contains the goal information when
 485 the agent enters the blue region.

486 **State.** A state has the same structure as an observation, but the second channel always contains the
 487 goal information.

488 **Reward.** +1 when reaching the goal, and 0 otherwise.

489 **B.3 Heaven-Hell**

490 In this domain, an agent must visit heaven (green cell) to finish the task. The goal cell can be either
 491 on the left or on the right side with 50% probability. To observe the side of the goal (left or right),
 the agent must visit the priest, who resides in the bottom right corner.



Figure 21: The Heaven-Hell domain with vector-based observations and states.

492

493 **Action.** Move-Right, Move-Left, Move-Up, Move-Down

494 **Observation.** A vector consists of the agent’s position and the side information. The side informa-
 495 tion can take the value of 0 (no information), 1 (heaven on the right), or -1 (heaven on the left).

496 **State.** Like the observation, but the true side of the goal is always revealed.

497 **Reward.** $+1$ when reaching heaven, -1 when reaching hell, and 0 otherwise.

498 **B.4 Robot Domains**

499 In these domains, the agent must manipulate the only movable object among two objects, which are
 500 exactly the same under the top-down depth image observation.

501 **Action.** An action $a = (\delta_x, \delta_y, \delta_z, \delta_r)$, where $\delta_{xyz} \in [-0.05, 0.05]$ are the displacements of the
 502 gripper in the XYZ axes, and $\delta_r \in [-\pi/8, \pi/8]$ is the angular rotation around the Z axis.

503 **Observation.** All robot domains share the same observation: the top-down depth image taken from
 504 the camera centered at the gripper’s position. Two fingers of the gripper are projected on the image.

505 **State.** The state also has two channels. The first channel is the first top-down depth image of the
 506 observation. While the second channel of the observation is non-informative, the second channel
 507 of the state is an image that masks everything except the movable object (see Fig. 22) in which the
 508 movable objects are colored red for visualizations).

509 **Reward.** In Block-Pulling, the agent receives a reward of 1.0 only when the two blocks are in
 510 contact. In Block-Pushing, the agent receives a reward of 1.0 only when the movable block is
 511 within 5 cm from the center of the goal pad. In Drawer-Opening, the agent receives a reward of
 512 1.0 only when the unlocked drawer is opened more than 5 cm.

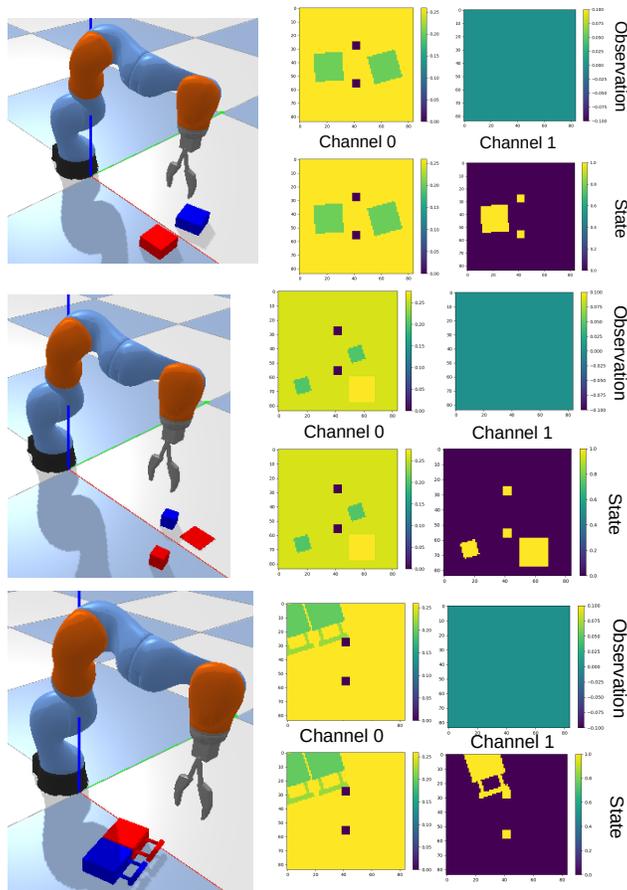


Figure 22: Visualization of an observation and a state in Block-Pulling, Block-Pushing, and Drawer-Opening. The movable object is the red one. The state and the observation have two channels, the first being the top-down depth image. In Block-Pulling, the second channel of the state reveals the movable object and the gripper. In Block-Pushing, the second channel reveals the movable block, the gripper, and the goal pad. In Drawer-Opening, the second channel in the state reveals the unlocked drawer and the gripper.

513 C Representations Training Details

514 C.1 Training Data Generation

515 Heaven-Hell, CarFlag-2D, Sphinx: In these domains, we use a uniform random agent to generate
 516 training samples, each is a transition (s, o, a, r, s', o') . For the number of samples used in each
 517 domain, please see Table 3.

518 In the robot domains, we use the same number of demonstrations (80 episodes) to learn the represen-
 519 tations during task learning. Furthermore, we augment the training data using random rotations per
 520 transition as used in [48] (also see Appendix F). Finally, we describe the planners used to generate
 521 the demonstrations in these domains.

522 **Planner in Block-Pulling:** The planner randomly selects a block and attempts to pull it to the
 523 other block direction until the task is accomplished. If, for a while, the position of the selected block
 524 remains unchanged, the planner will move the gripper to the other block and repeat the pulling.

525 **Planner in Block-Pushing:** A block is randomly chosen and pushed toward the goal pad. If the
 526 block’s position remains unchanged for a while, the planner will move the gripper to the other block
 527 and resume pushing until the task is finished.

528 **Planner in Drawer-Opening:** The planner selects a drawer randomly and tries to open it. If the
 529 chosen drawer fails to open after a while, the gripper will move to the other drawer and repeat the
 530 opening action.

531 C.2 Network Architecture

532 The specific architecture used to learn representations is shown in Fig. 23.

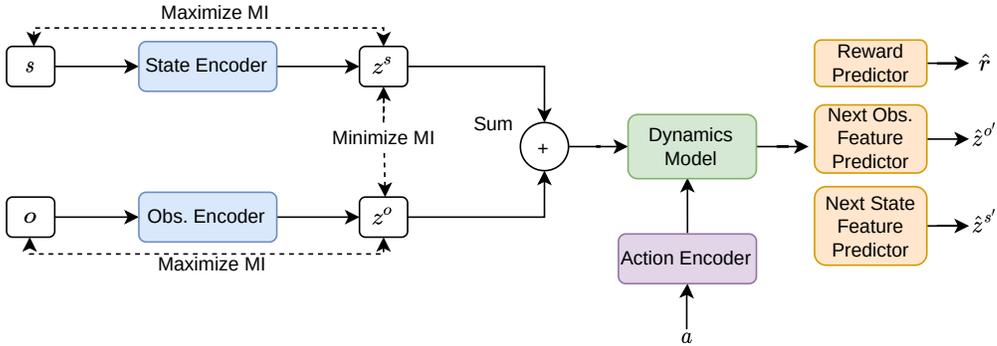


Figure 23: Architecture to learn representations in all domains.

533 Next, we describe the components for each domain from Fig. 24 to Fig. 27. To succinctly describe
 534 the network architecture, we use the following acronyms: **FC**(n): a fully connected layer with
 535 n outputs; **Conv**(f, s): a convolutional layer with filter size $f \times f$ and stride s , **R** is the ReLU
 536 activation, and **MaxPool**(w): a max pooling layer with window size w .

537 C.3 Mutual Information Estimation

538 C.3.1 Minimizing $I(z^s; z^o)$

539 From the upper bound equation Eq. (2), we minimize its variational estimate defined below:

$$\mathcal{L}_{\text{CLUB}} = \frac{1}{B^2} \sum_{i=1}^B \sum_{j=1}^B [\log q(z_i^o | z_i^s) - \log q(z_j^o | z_i^s)] \quad (7)$$

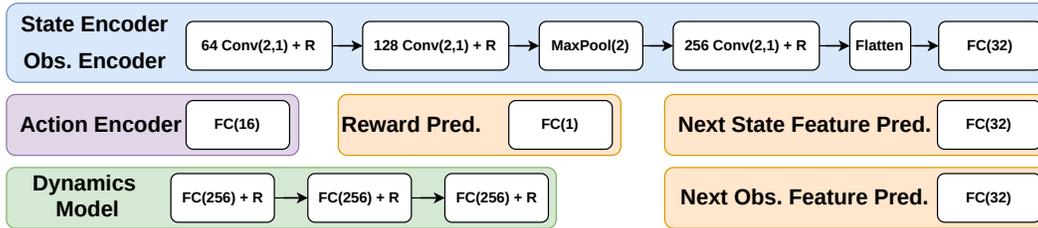


Figure 24: Network architecture in Sphinx.

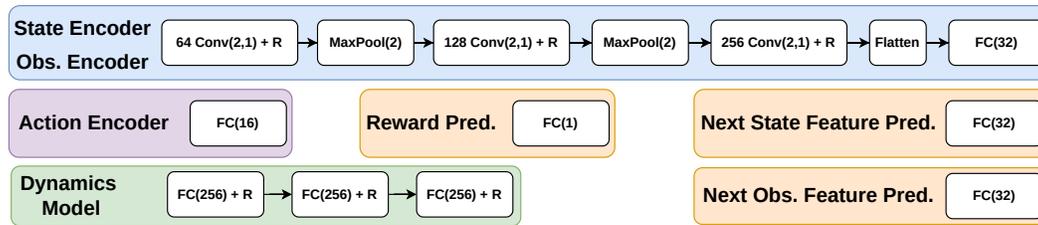


Figure 25: Network architecture in CarFlag-2D.

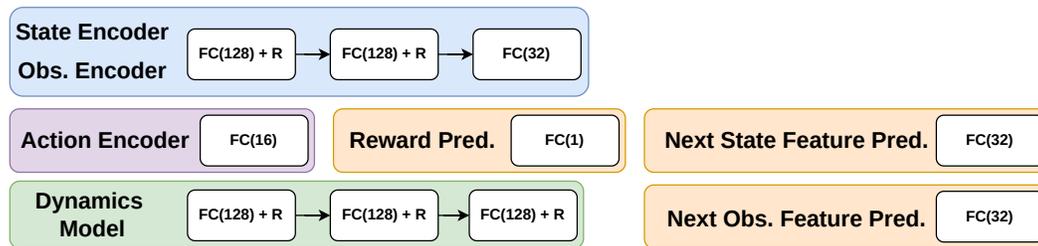


Figure 26: Network architecture in Heaven-Hell.

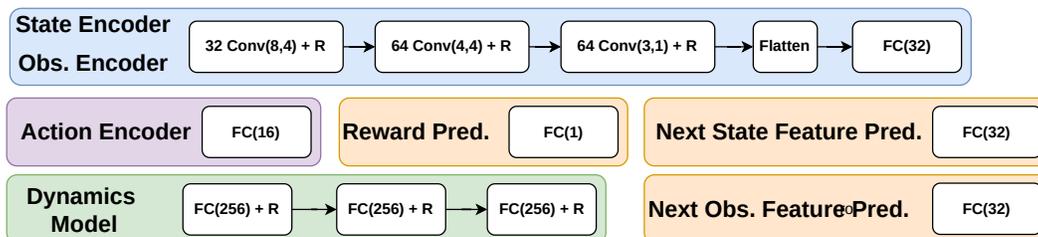


Figure 27: Network architecture in robot domains.

540 The variational distribution $q(z^o|z^s)$ is updated to minimize $\mathbb{D}_{\text{KL}} [q(z^o|z^s) || p(z^o|z^s)]$. We assume
 541 $q(z^o|z^s)$ follows a Gaussian distribution and use the following network architectures:

542 **Mean network:** $\text{FC}(32) \rightarrow \mathbf{R} \rightarrow \text{FC}(32)$

543 **Log variance network:** $\text{FC}(32) \rightarrow \mathbf{R} \rightarrow \text{FC}(32) \rightarrow \text{Tanh}$

544 We use the batch size $B = 500$ and use a learning rate of 0.001 for all tasks, except Heaven-Hell,
 545 in which a learning rate of 0.0003 is used. We update q whenever we update $\phi(s)$ and $\psi(o)$.

546 C.3.2 Maximizing $I(o; z^o)$ and $I(s; z^s)$

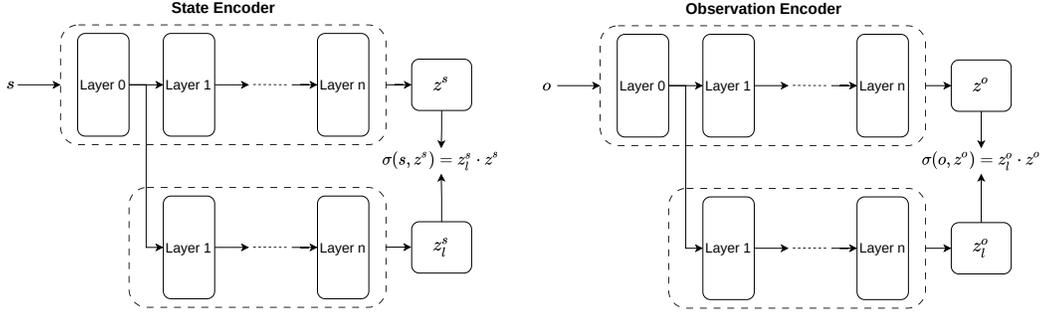


Figure 28: Architecture to calculate $\sigma(s, E(s))$ and $\sigma(o, E(o))$ using the dot product operation.

547 From the lower bound equation Eq. (3), we minimize the following loss:

$$\mathcal{L}_{\text{DIM}} = \frac{1}{B^2} \sum_{i=1}^B \sum_{j=1}^B [\text{sp}(-\sigma(x_i, E(x_i))) + \text{sp}(\sigma(x_j, E(x_i)))] \quad (8)$$

548 As shown in Fig. 28, the discriminator σ uses the same architecture of the state encoder ϕ (when
 549 calculating the state feature z_l^s of s) and the observation encoder ψ (when calculating the observation
 550 feature z_l^o of o). We dot product to compute $\sigma(s, E(s)) = z_l^s \cdot z^s$ and $\sigma(o, E(o)) = z_l^o \cdot z^o$.

551 C.4 Hyper-parameters

552 We provide the hyper-parameters used for training representations in Table 3.

Table 3: Hyper-parameters used in learning representation. HH: Heaven-Hell, S: Sphinx, CF: CarFlag-2D, BP: Block-Pulling, BPs: Block-Pushing, and DO: Drawer-Opening.

Domain	HH	CF	S	BP	BPs	DO
# of samples	21785	45406	13682	1226	1240	1234
# of episodes	500	1000	500	80	80	80
# of augmentations per sample	-	-	-	4	12	6
# of training epochs	1000	1000	1000	1000	1000	1000
Batch size B	500	500	500	500	500	500
Learning rate	0.003	0.001	0.001	0.001	0.001	0.001
Reward loss coeff. λ_r	10.0	1.0	10.0	10.0	100.0	100.0
State loss coeff. λ_s	1.0	1.0	0.5	0.1	1.0	1.0
Observation loss coeff. λ_o	0.5	5.0	0.03	1.0	1.0	1.0
$\downarrow I(z^s; z^o)$ loss coeff. λ_{CLUB}	1.0	10.0	0.3	10.0	0.001	1.0
$\uparrow I(s; z^s)$ loss coeff. λ_{DIM}	0.0	0.0	0.0	0.1	0.01	0.001
$\uparrow I(o; z^o)$ loss coeff. λ_{DIM}	1.0	1.0	0.5	1.0	1.0	1.0

553 **D Additional Experiments**

554 **D.1 Using $z^s \oplus z^o$ versus z^s for Task Learning**

555 Continuing the experiment from Section 5.2.1, we report the performance using z^s and $z^s \oplus z^o$ for
 556 task learning in all domains.

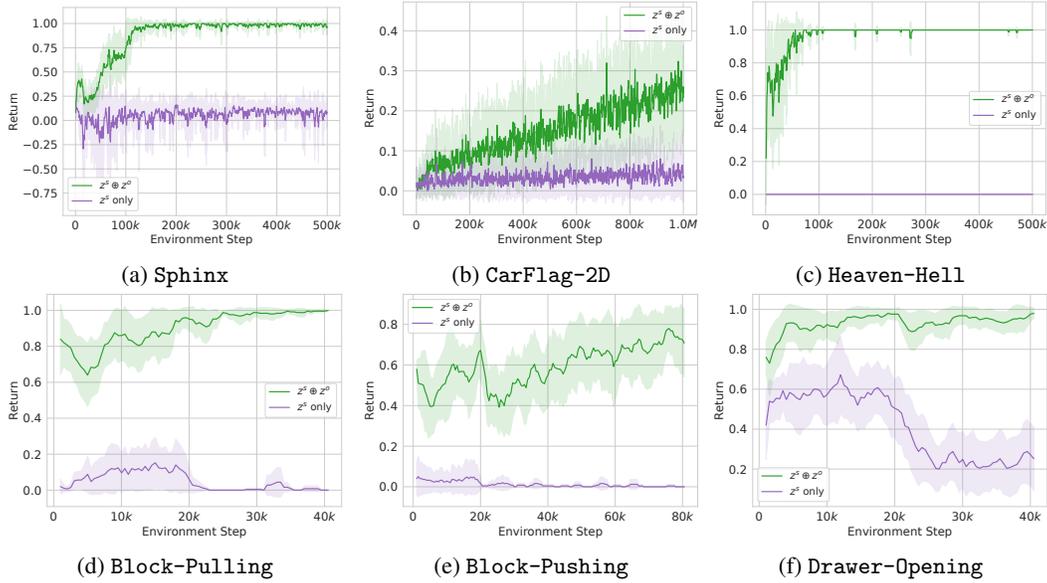


Figure 29: Task learning performance when using z^s and $z^s \oplus z^o$ as the “state”.

557 **D.2 Using Only Auxiliary Task/Intrinsic Rewards**

Here, we show the learning performance when using intrinsic rewards and/or the auxiliary task.

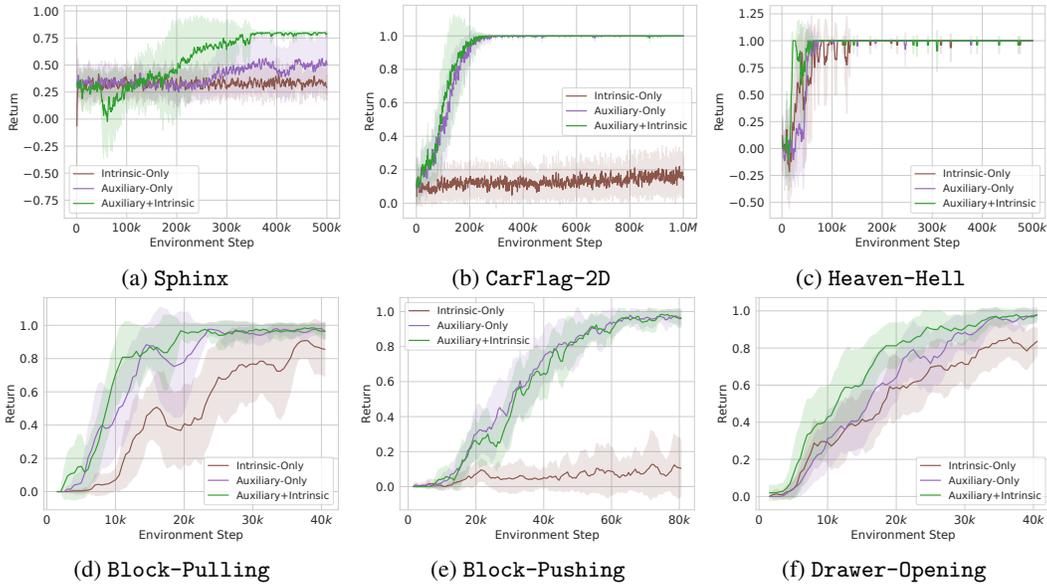


Figure 30: Comparing using intrinsic rewards or the auxiliary task versus using both.

559 **D.3 Using GRU v.s. GPT**

560 Here, we report the performance in all domains when using a GRU versus GPT as the sequence
 561 model in our proposed agent.

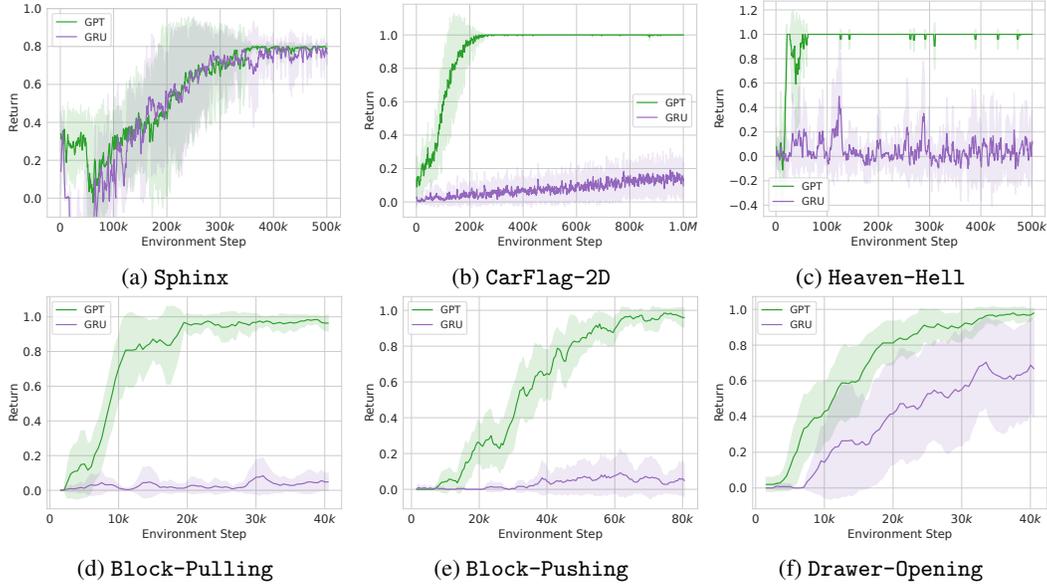


Figure 31: Task learning performance when using a GRU v.s. GPT.

561

562 **D.4 Visualization of Intrinsic Rewards**

563 We visualize the intrinsic rewards of trained agents in three grid-world domains in Fig. 32. The
 564 intrinsic rewards peak when the agents perform the information-gathering actions.

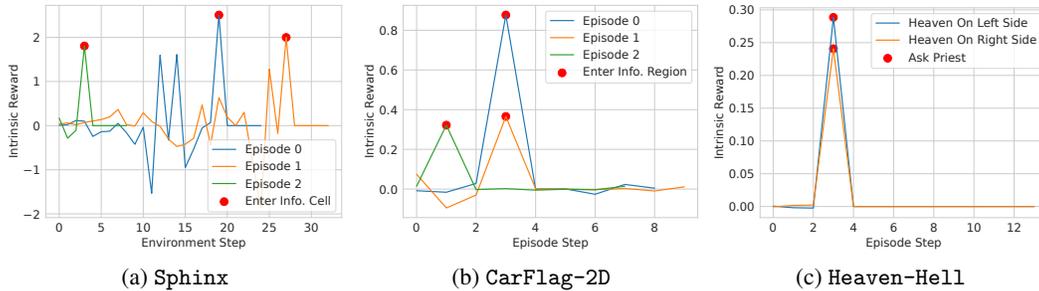


Figure 32: Intrinsic rewards within an episode of trained agents in three grid-world domains. Red circles denote when the intrinsic rewards peak, e.g., when they perform informative actions.

565 **E Details of Hardware Experiments**

566 **E.1 Obtaining Depth Images**

567 We fuse the point clouds from two RealSense D455 cameras (Cam 1 and Cam 2) and one Azure
 568 Kinect camera (Cam 3) to create an integrated point cloud (see Fig. 33). We then orthographically
 569 project the point cloud at the gripper’s position to create a depth image observation. Examples of
 570 observations in the three robot domains can be seen in Fig. 34.

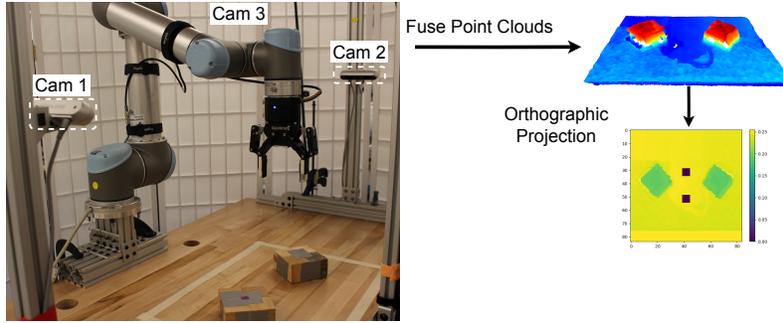


Figure 33: We fuse the point clouds from three cameras (to avoid occlusions) and performed an orthographic projection at the gripper’s position to create a depth image observation.

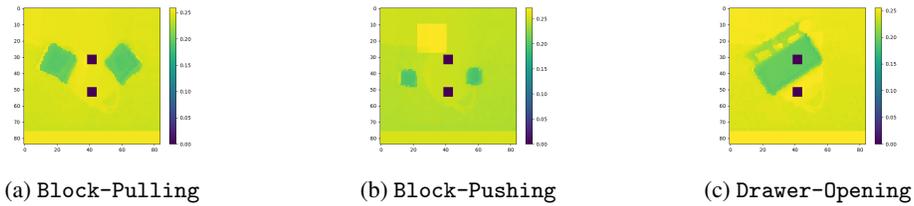


Figure 34: Examples of observations in real robot experiments.

571 **E.2 Added Perlin Noise for Better Sim-To-Real Transfers**

572 Following [38], we found it useful for better sim-to-real transfers by adding the Perlin [45] noise to
 573 the depth images during training for more robust policies by being closer to real-world depth images.
 For all robot domains, we applied the noise with a magnitude of 7mm (see Fig. 35).

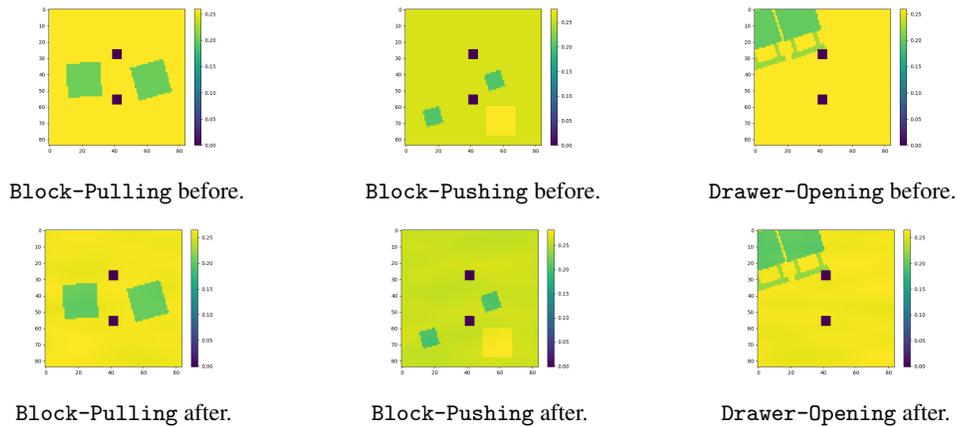


Figure 35: Depth images before and after adding Perlin [45] noise for better sim-to-real transfers.

574

575 **F Details of SO(2) Rotational Augmentation**

576 We perform SO(2) rotational augmentation by choosing a random angle and rotating the depth im-
 577 ages around its center. We perform this augmentation in two cases:

578 **When learning the representations to utilize the data better.** For each transition (s, o, a, r, s', o') ,
 579 we sample a random angle and rotate s, o, s', o' at the same angle. Each transition has its own
 580 random angle, see Fig. 36 for examples.

581 **When performing task learning robot domains.** Given an episode, we first sample a random angle
 582 and apply the rotation with this angle for *every* s, o, s', o' within the episode. Because we are trying
 583 to learn a history-based policy, this is to ensure the augmented history is valid (see Fig. 37).

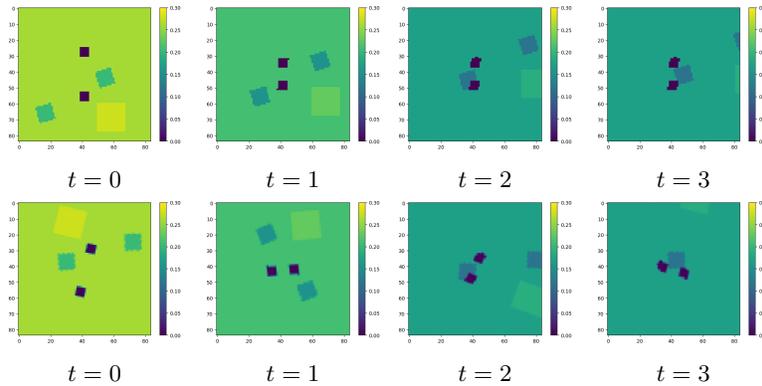


Figure 36: Examples of rotation data augmentation applied for transitions in an episode in Block-Pushing to augment the data for learning the representation: a *different* random rotation is applied independently for s, o, s', o' in each timestep in an episode.

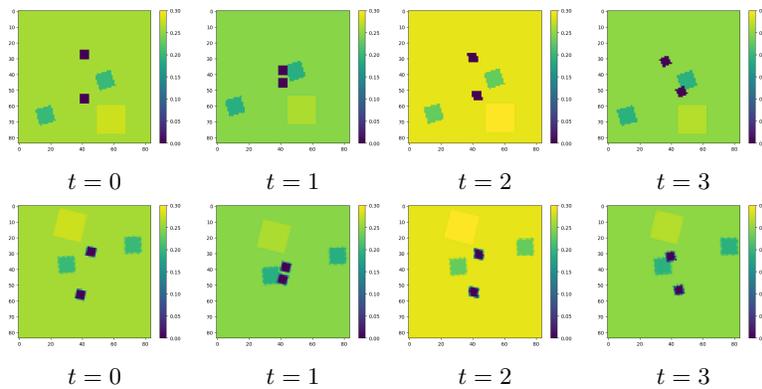


Figure 37: Examples of rotation data augmentation applied for an episode in Block-Pushing: the *same* random rotation is consistently applied to every s, o, s', o' within an episode.