

A POLTER and the State-Visitation Entropy

To gain further insights into POLTER, we conduct an experiment following Hazan et al. [2019]. We discretize the state space of the Walker environment and compute the state-visitation entropy during reward-free pretraining. In Table 1, we see that the entropy of the distribution of POLTER regularized algorithms is lower than that of their counterpart. This effect is most pronounced in data-based algorithms, such as ProtoRL and APT, where the performance is also improved the most (see Table 3).

Table 1: State-visitation entropy of the evaluated URL algorithm categories in the Walker domain during pretraining. Averaged over 10 seeds with 50 k states each at pretraining steps 100 k, 500 k, 1 M and 2 M.

POLTER	Data	Knowledge	Competence
✗	0.2772 ± 0.0188	0.2863 ± 0.0036	0.2540 ± 0.0429
✓	0.2545 ± 0.0493	0.2848 ± 0.0054	0.2511 ± 0.0422

Knowledge-based algorithms also benefit from the regularization but have a slightly reduced entropy. Because competence-based algorithms already average over a set of skills found during pretraining, the effect of POLTER is the smallest.

These results imply that POLTER does not lead to a better exploration of the state space. Instead, its performance gains are the result of an improved prior as indicated by the reduced KL-divergence between the policy and the optimal pretraining policy on PointMass (Section 3.2). This experiment showed that a good exploration of the state-space is required but not sufficient to achieve good performance with URL algorithms.

B Deep Deterministic Policy Gradient

DDPG Lillicrap et al. [2016] is an off-policy actor-critic algorithm that optimizes a state-action value function Q_ϕ and uses it to train a policy network μ_θ . The state-action value function is trained by minimizing the loss

$$\mathcal{L}^{\text{critic}}(Q_\phi) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}, d_t) \sim \mathcal{D}} \left[\left(Q_\phi(s_t, a_t) - (r_t + \gamma(1 - d_t)Q_{\phi_{\text{targ}}}(s_{t+1}, \mu_{\theta_{\text{targ}}}(s_{t+1}))) \right)^2 \right] \quad (4)$$

using samples from a replay buffer \mathcal{D} and the policy is trained by maximizing

$$\mathcal{L}^{\text{actor}}(\mu_\theta) = \mathbb{E}_{s_t \sim \mathcal{D}} [Q_\phi(s_t, \mu_\theta(s_t))]. \quad (5)$$

In Equation (3), the URL algorithms loss \mathcal{L}^{URL} corresponds to the actor loss $\mathcal{L}^{\text{actor}}$ of the DDPG agent.

C Environments in the Unsupervised Reinforcement Learning Benchmark

The Unsupervised Reinforcement Learning Benchmark Laskin et al. [2021] contains three domains with the topics of locomotion and manipulation. **Walker**, **Quadruped** and **Jaco**, are used to explore the effects of different URL algorithms. It has a specific training and evaluation protocol which we also follow in this work. The **Walker** domain contains a planar walker constrained to a 2D vertical plane, with an 18-dimensional observation space and a 6-dimensional action space with three actuators for each leg. The associated tasks are *stand*, *walk*, *run* and *flip*. The walker domain provides a challenging start for the agent since it needs to learn balancing and locomotion skills to be able to adapt to the given tasks. The next domain is **Quadruped**, which expands to the 3D space. It has a much larger state space of 56 dimensions and a 12-dimensional action space with three actuators for each leg. The tasks in this environment are *stand*, *walk*, *jump* and *run*. The last environment used is the **Jaco Arm**, which is a robotic arm with 6-DOF and a three-finger gripper. This domain is very different from the other two, as its setting is manipulation and not locomotion. The tasks are *Reach top left*, *Reach top right*, *Reach bottom left* and *Reach bottom right*.

D Hyperparameters and Resources

POLTER Hyperparameters During pretraining we construct the mixture ensemble policy $\tilde{\pi}$ with $k = 7$ members at specific time steps \mathcal{T}_E . For adding each member we choose the ensemble snapshot time steps $\mathcal{T}_E = \{25 \text{ k}, 50 \text{ k}, 100 \text{ k}, 200 \text{ k}, 400 \text{ k}, 800 \text{ k}, 1.6 \text{ M}\}$. The steps were chosen according to initial experiments of applying RND in the Quadruped domain, where there are large changes of the intrinsic reward at the beginning, which become progressively smaller over time. We set the regularization strength $\alpha = 1$ and use the same hyperparameters for each of the three domains unless specified otherwise.

Baseline Hyperparameters The hyperparameters for our baseline algorithms follow Laskin et al. [2021] and Laskin et al. [2022]. The hyperparameters for the DDPG baseline agent are described in Table 2.

Table 2: Hyperparameters for the DDPG algorithm.

Hyperparameter	Value
Replay buffer capacity	1×10^6
Action repeat	1
Seed frames	4000
n -step returns	3
Batch size	1024
Discount factor γ	0.99
Optimizer	Adam
Learning rate	1×10^{-4}
Agent update frequency	2
Critic target EMA rate	0.01
Feature size	1024
Hidden size	1024
Exploration noise std clip	0.3
Exploration noise std value	0.2
Pretraining frames	2×10^6
Finetuning frames	1×10^5

Compute Resources All experiments were run on our internal compute cluster on NVIDIA RTX 1080 Ti and NVIDIA RTX 2080 Ti GPUs and had 64GB of RAM and 10 CPU cores. In total, we trained over 12 000 models and performed $\approx 3\,500\,200\,000$ environment steps.

E Detailed Results on Unsupervised Reinforcement Learning Benchmark

In this section we provide additional results for our experiments on Unsupervised Reinforcement Learning Benchmark. In the supplementary we provide the raw scores. The statistics comparing URL algorithms with and without POLTER aggregated for finetuning on 12 tasks across 10 seeds can be found in Table 3. In addition we show aggregate statistics of the absolute improvement in expert performance in Figure 8 and the performance profiles [Agarwal et al., 2021] per URL algorithm category in Figure 9. As before, we see a large improvement for data- and knowledge-based algorithms and small or negative for competence-based algorithms. The improvement sometimes varies strongly across seeds and tasks. Also, we show the normalized return after finetuning from different pretraining snapshots for each domain and URL category in Figure 10. Across the Jaco domain, URL algorithms with and without POLTER mostly deteriorate with an increasing number of pretraining steps. Each category shows a different trend on each domain. Interestingly, the competence-based algorithms SMM and DIAYN fail during pretraining in the Jaco domain. In Figure 11 we see the normalized return over finetuning steps. POLTER is mostly on par or speeds up in comparison to the URL algorithm without POLTER. In total, most algorithms do not converge yet after 100 k steps.

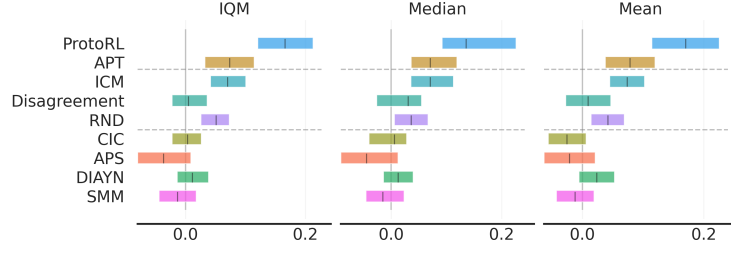


Figure 8: Aggregate statistics of the absolute improvement with POLTER per URL category.

Table 3: Raw aggregate statistics following Agarwal et al. [2021] of evaluated URL algorithms with and without POLTER regularization. The results marked with POLTER* were obtained by tuning the regularization strength to the task domain of locomotion (Walker and Quadruped) and manipulation (Jaco).

Algorithm	IQM \uparrow	Mean \uparrow	Median \uparrow	Optimality Gap \downarrow	POLTER IQM Improvement
ProtoRL	0.56	0.55	0.52	0.45	
ProtoRL+POLTER	0.77	0.71	0.65	0.29	+40%
ProtoRL+POLTER*	0.79	0.76	0.80	0.24	+41%
APT	0.59	0.61	0.56	0.39	
APT+POLTER	0.69	0.68	0.66	0.32	+17%
RND	0.70	0.71	0.67	0.30	
RND+POLTER	0.77	0.75	0.74	0.26	+10%
RND+POLTER*	0.77	0.76	0.71	0.24	+10%
ICM	0.54	0.52	0.59	0.48	
ICM+POLTER	0.63	0.60	0.65	0.40	+17%
Disagreement	0.68	0.69	0.66	0.31	
Disagreement+POLTER	0.69	0.70	0.69	0.31	+1%
CIC	0.78	0.76	0.74	0.24	
CIC+POLTER	0.76	0.74	0.77	0.26	-2%
CIC+POLTER*	0.84	0.81	0.86	0.20	+7%
DIAYN	0.36	0.39	0.42	0.61	
DIAYN+POLTER	0.39	0.42	0.42	0.58	+8%
SMM	0.36	0.42	0.30	0.58	
SMM+POLTER	0.36	0.41	0.30	0.59	$\pm 0\%$
APS	0.56	0.58	0.55	0.42	
APS+POLTER	0.52	0.53	0.54	0.47	-7%
DDPG	0.55	0.54	0.56	0.46	

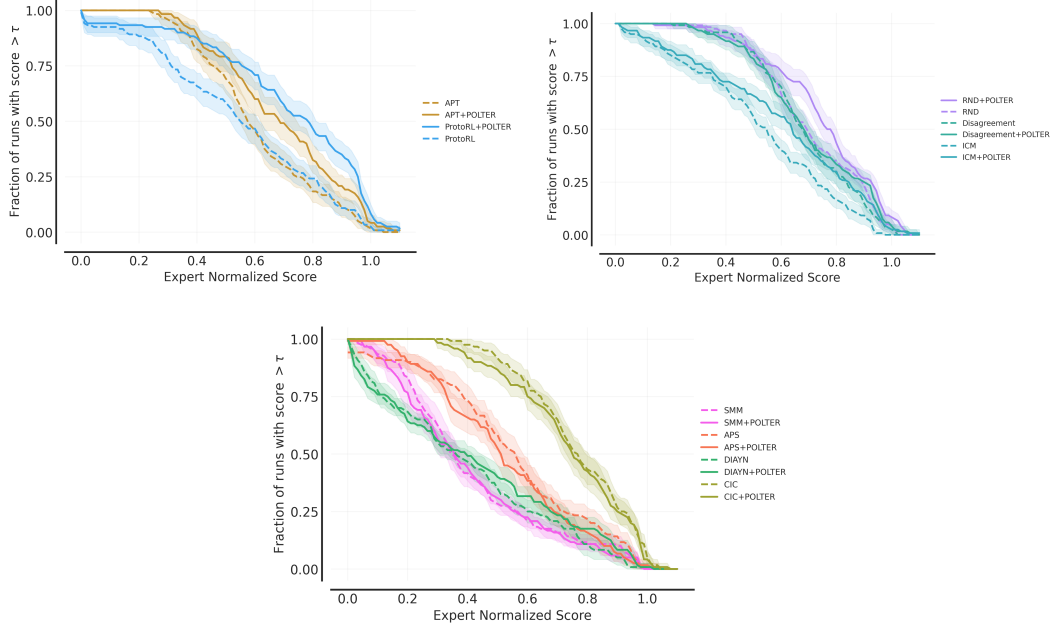


Figure 9: Performance profiles after finetuning of the different algorithms averaged over 10 seeds where the shaded region indicates the standard error. Variants without POLTER are dashed.

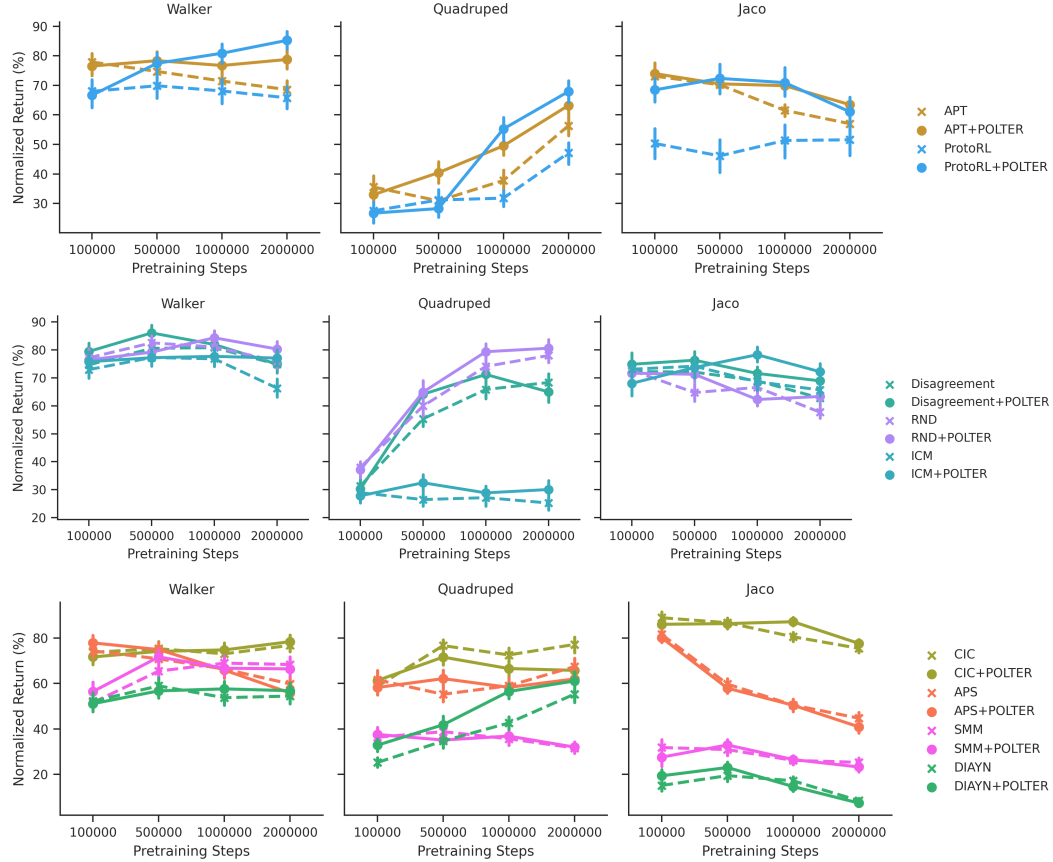


Figure 10: Finetuning from different pretraining snapshots of data-, knowledge- and competence-based algorithms. The error bars indicate the standard error of the mean.

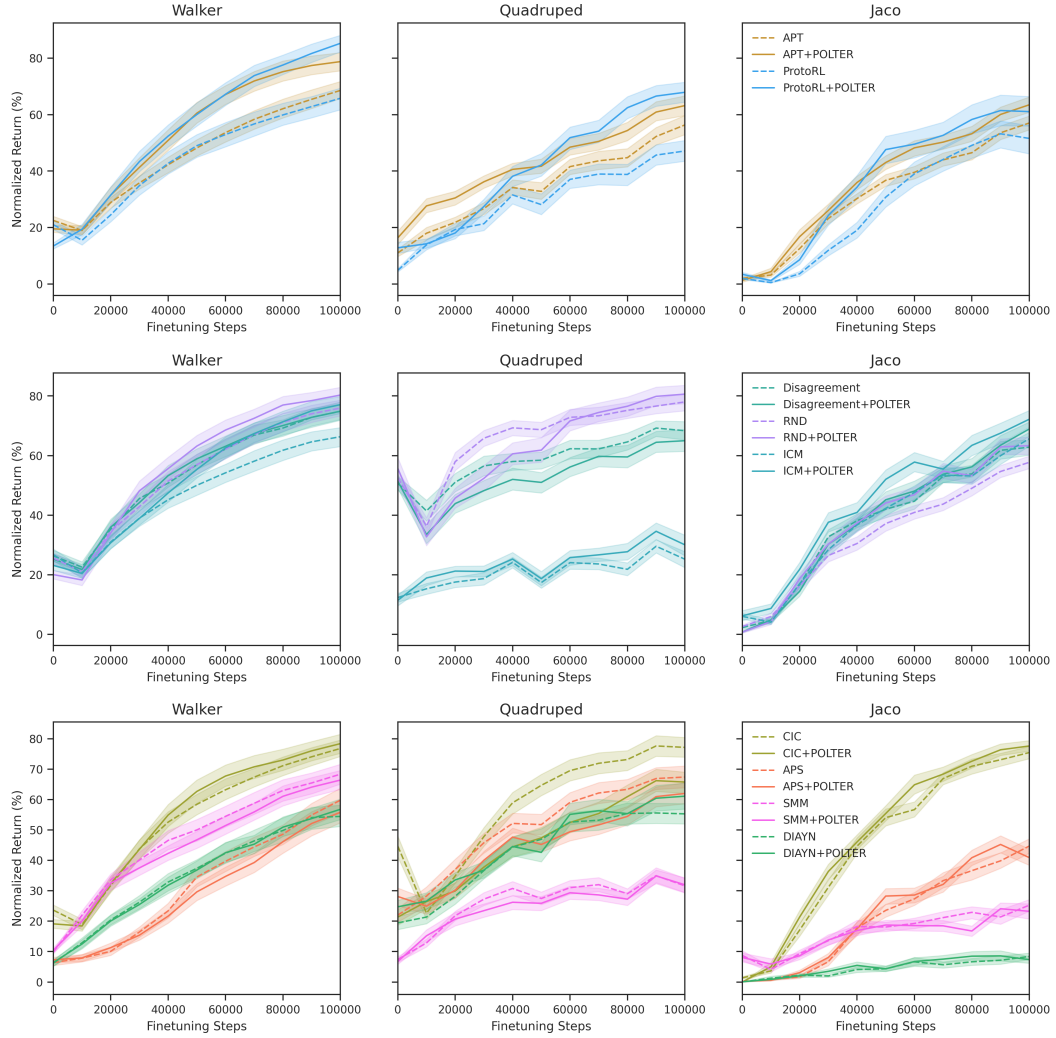


Figure 11: Finetuning curves of data-, knowledge- and competence-based algorithms after pretraining for 2 M steps. The shaded area indicates the standard error.

560 F Additional Analysis of PointMass

561 Figure 12a and Figure 12b complement the figures Figure 2a and Figure 2b in the main text. They
 562 provide further evidence that the POLTER regularized policy is closer to the optimal prior through-
 563 out the whole pretraining process and demonstrate the improvement in sample-efficiency during
 564 finetuning.

565 In Figure 13a, we can see the state distribution changing during pretraining with and without POLTER.
 566 With POLTER, the state space coverage is less and the trajectories seem more ordered. RND without
 567 POLTER also seems to visit the edges often at the end. When using the POLTER regularization, we
 568 can see that each pretraining checkpoint is visiting different states, as indicated by the visibility of
 569 the previous checkpoint’s state visitations. When not using POLTER we can see that the visitations
 570 overlap each other. Figures 13b and 13c show the discretized position and speed the agent explores
 571 over the course of pretraining. Especially the discretized speed (Figure 13c) demonstrates the tradeoff
 572 between dampening the exploration with POLTER and finding better prior policies, because with
 573 POLTER less states are frequented and the states are less extreme.

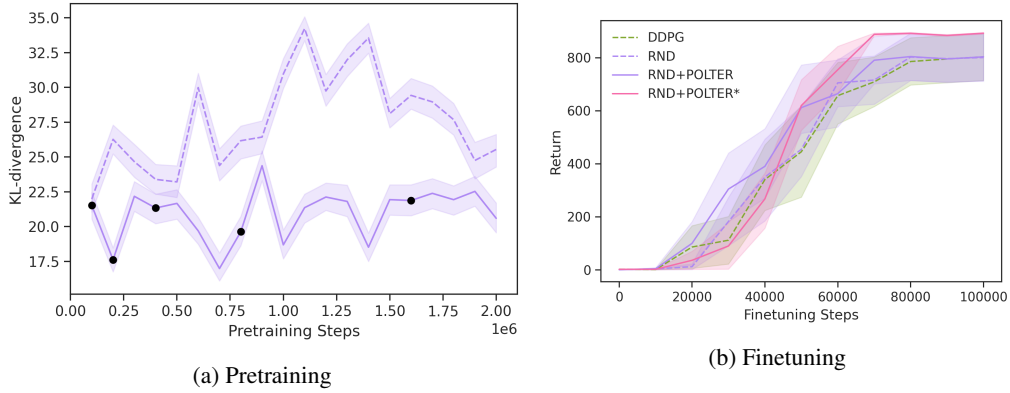
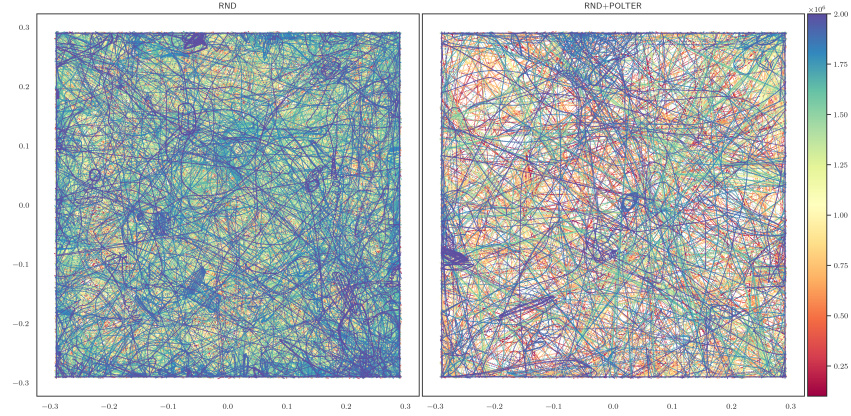
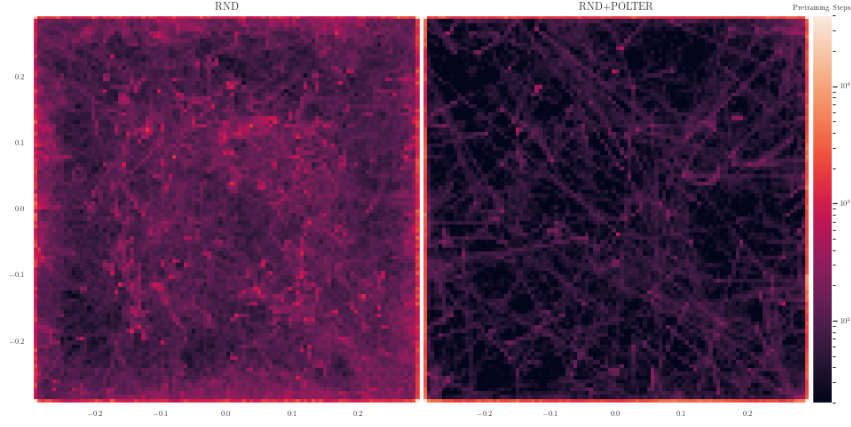


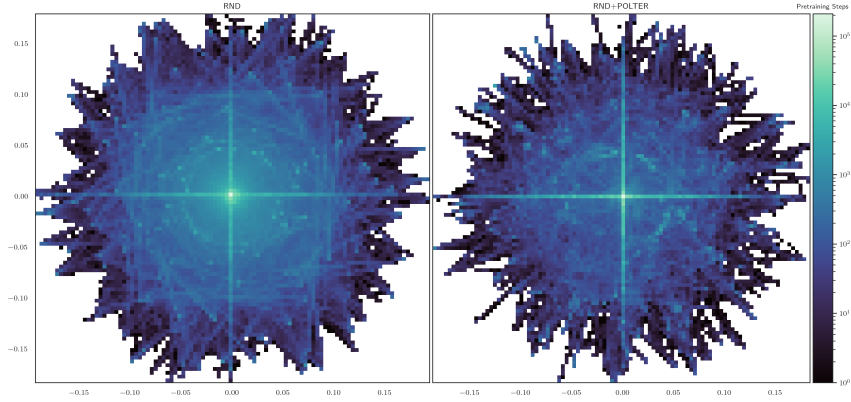
Figure 12: **(a)** Average KL-divergence of RND (dashed) and RND+POLTER (solid) between the pre-training policy $\pi(s_0)$ and the optimal pretraining policy $\pi_T^*(s_0)$ in the PointMass domain during reward-free pretraining for 2 M steps. Each policy is evaluated every 100 k steps on 20 initial states over 10 seeds. The black dots indicate the steps a snapshot is added to the ensemble. **(b)** Return during finetuning after 200 k pretraining steps, where the target is placed at a fixed random position for each of the 10 seeds. We also provide a DDPG baseline without pretraining and RND+POLTER* using the optimal policy instead of the ensemble. The shaded area indicates the standard error.



(a) State distribution $\rho(s)$ of several checkpoints.



(b) Discretized state histogram of the positions. If you look closely, you can see that the edges of the 2D plane are very often visited. This is due to policies always applying the same force and thus reaching the edge of the 2D plane.



(c) Discretized state histogram of the speeds. Note that the prominent horizontal and vertical lines result from moving at the edges and being stuck at the corners of the 2D plane.

Figure 13: State distribution and histogram of RND (always left column) and RND+POLTER (right column) during pretraining on the PointMass environment.