A Appendix

A.1 Curriculum Optimization in Low-Fidelity

For optimizing curriculum in the Low-Fidelity (LF) environment, we use the Automated (AC) procedure (`Generate_AC`) proposed in "ACuTE" [1]. The approach is given in Algorithm 1. We start from an empty sequence of source task parameters P_W. The algorithm calls a parameterizing function (`Init_Src`) that assigns random values to the parametric variables for the first source task P^{LF_1} from the range of values $\mathcal{P}^{LF}$ can attain while simultaneously initializing an RL agent (`Init_Agent`). The range of values for $\mathcal{P}^{LF}$ is calculated by taking the range of the parameters in high fidelity $\mathcal{P}^{HF}$ and applying the corresponding mapping function $\mathcal{F}$ on each parameter. Based on P^{LF_1}, the algorithm generates the first task for the agent, M_1^{LF}, using the function (`Generate_Env`). The agent attempts learning this source task (`Learn`) with the initial policy $\pi_{1,w,init}$, until the *stopping criterion* is met. The *stopping criterion* determines if the agent's goal rate (δ) is $\geq \delta_G$ in the last s episodes. Failure to meet the *stopping criterion* implies that the agent has reached maximum permitted episodes (termed *budget* (b)) signifying $\delta < \delta_G$. The value for δ_G is set at 0.85 for our experiments.

The first task of the curriculum is randomly initialized N times and learned until *stopping criterion* is met. The algorithm then finds the W most promising solutions (`Best_Candidates`), based on fewest interactions to meet the *stopping criterion*. To optimize the sequence of the curriculum, we use beam search. Beam search is a greedy search algorithm that uses a breadth-first search approach to formulate a tree. At each level of the tree, the algorithm sorts all the ($N \times W$) successors of the tree (N successor tasks for each task of W) at the current level in increasing order of number of episodes required to learn the task $M_{u,w,n}^{LF}$. Then, it selects (W) number of best tasks at each level, given by fewest interactions to reach *stopping criterion*, and performs the same step until the tree reaches the desired number of levels (U). Now, using the parametric

variables P^{LF_1} for each task in the beam ($w \in W$), the algorithm generates parametric variables P^{LF_2} (`Init_Inter`) for the next task M_2^{LF} in the curriculum. This is done by choosing a goal P_G not encountered by the agent until the current level u in the beam w, and randomly initializing parametric variables $\geq$ the minimum required to accomplish this goal. The agent then attempts learning M_2^{LF} with the final policy of the previous source task in the beam $\pi_{1,w,fin}$ (`Load_Agent`). The task terminates when the agent meets the *stopping criterion*. The algorithm finds the W most promising solutions, given by fewest interactions to reach *stopping criterion* (`Best_Candidates`) and carries out this procedure iteratively, until the final target task M_U^{LF} is learned. The parameters of the curriculum with the lowest number of episodes to reach the *stopping criterion* is selected as the most promising solution (`Best_LF_Params`) P_W for learning the target task. The curriculum generation procedure requires the length of the curriculum U to be $>=$ the number of goals attainable. This

Algorithm 1 `Generate_AC`$(N, W, U, M_U^{LF}, seeds)$

Output: LF Curriculum Parameters: P_W
Placeholder Initialization: Timesteps: $T_1, \ldots, T_U \leftarrow \emptyset$
LF task params for all tasks at each beam level $\xi_1, \ldots, \xi_U \leftarrow \emptyset$
LF task params at each width and level $\xi_{1,W}, \xi_{2,W}, \ldots, \xi_{U,W} \leftarrow \emptyset$
LF curriculum params $P_W \leftarrow \emptyset$
LF task policies for all tasks at each beam level: $\Pi_1, \ldots, \Pi_U \leftarrow \emptyset$
LF task policies for each width and level: $\Pi_{1,W}, \ldots, \Pi_{U,W} \leftarrow \emptyset$
Algorithm:

```
1:  for $u \in U$ do
2:      for $w \in W$ do
3:          for $n \in N$ do
4:              if $u = 1$ then
5:                  $P^{LF_u} \leftarrow$ Init_Src($M_U^{LF}$)
6:                  $\pi_{1,w,init}$ = Init_Agent($seeds$)
7:              else
8:                  $P^{LF_u} \leftarrow$ Init_Inter($\xi_{u-1,W}[w]$,$M_U^{LF}$)
9:                  $\pi_{u,w,init}$ = Load_Agent($\Pi_{u-1,W}[w]$)
10:             end if
11:             $\xi_u\ [w,n] \leftarrow P^{LF_u}$
12:             $M_{u,w,n}^{LF}$ = Generate_Env($P^{LF_u}$)
13:             $(t_{u,w,n}, \pi_{u,w,,n,fin})$ = Learn($M_{u,w,n}^{LF}, \pi_{u,w,n,init}$)
14:             $T_u\ [w,n] \leftarrow t_{u,w,n}$ , $\Pi_u\ [w,n] \leftarrow \pi_{u,w,n,fin}$
15:         end for
16:     end for
17:     $T_{u,W}, \Pi_{u,W}, \xi_{u,W}$= Best_Candidates($T_u$,$\Pi_u$,$W$,$\xi_u$)
18: end for
19: $P_W \leftarrow$ Best_LF_Params($\xi_{1,W}, \xi_{2,W}, \ldots, \xi_{U,W}$)
20: return $P_W$ =0
```

ensures all the goals available (P_G) are encountered by the agent in the curriculum. For our experiments, $N = 15$, $W = 4$ and $U = 3$.

The set of goals for the Crafter-TurtleBot environment are:
$P_G^{Nav} = \{Navigation, Breaking, Crafting\}$.

And, the set of goals for the Panda-Pick-And-Place environment are:
$P_G^{Man} = \{Reach, Pick, PickAndPlace\}$.

B RL Implementation

The RL algorithm for both the Low-Fidelity environments was Policy Gradient [2] network with ϵ-greedy action selection for learning the optimal policy. The episode terminates when the agent successfully reaches the goal state (crafting a stone axe in the Crafter-TurtleBot environment and Placing objects at target locations in the Panda-Pick-And-Place environment) or exceeds the total number of timesteps permitted, which is 100 and 6×10^2 in the LF and HF environments for Crafter-TurtleBot and 50 in the LF and HF environments for Panda-Pick-And-Place. All experiments are averaged over 10 trials

For the Panda-Pick-And-Place HF environment, the RL agent was learned using Deep Deterministic Policy Gradient [3] with Hindsight Experience Replay [4]. After learning a policy for a task in the curriculum, the agent clears its replay buffer before proceeding onto the next task.

B.1 Policy gradient implementation details

The experiments were conducted using a 64-bit Linux Machine, having Intel(R) Core(TM) i9-9940X CPU @ 3.30GHz processor and 126GB RAM memory. The maximum duration for running the experiments was set at 24 hours. Table 1 lists the hyperparameters for policy gradient.

Parameter	**Value**
discount factor γ	0.995
learning rate α	1×10^{-3}
optimizer	RMSProp
Gradient moving average decay factor ρ	0.99
exploration rate ϵ	0.1
action distribution	categorical with 5 bins

Table 1: Parameters used for training the Policy Gradient on the High-Fidelity task

B.2 Deep Deterministic Policy Gradient implementation details

Table 2 lists the hyperparameters for DDPG. DDPG+HER was used for the Panda-Pick-And-Place high-fidelity task.

Parameter	**Value**
Actor Learning Rate	1×10^{-3}
Critic Learning Rate	1×10^{-3}
Actor and Critic optimizer	Adam
Polyak average factor ρ	0.05
Actor and Critic Network	$[256, 256, 256]$
Batch Size	256
Discount Factor	0.98
action distribution	4 Continuous actions within $[-1, 1]$

Table 2: Parameters used for training the Deep Deterministic Policy Gradient on the Panda-Pick-And-Place High-Fidelity task

B.3 Imperfect Mappings

In certain partially observable environments, it might not be possible to obtain an accurate mapping between the two environments. To demonstrate the efficacy of our approach in imperfect mappings, we evaluate the experiments by incorporating noise in the mapping function. The noisy mapping function involves a multivariate noise over the range of the parametric variables. While obtaining the noisy parameters, we do not incorporate any noise in the parameter for the goal condition, as we can safely assume that the goals have been mapped accurately. The noisy mappings are given by:

$$P_{noisy} = P_{exact} + \mathcal{N}(0, \Sigma)$$

where $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & \dots \\ 0 & \sigma_2 & 0 & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \sigma_n \end{bmatrix}_{n \times n}$ is the covariance matrix, which is symmetric and positive semi-definite, and $\sigma_i = (max(P_i) - min(P_i))/6$, covers the entire range of the parametric variable in six standard deviations. We verify whether the noisy parameters meet the minimum requirements of reaching the goal for the sub-task. If the requirements are not met, we generate a new set of noisy parameters until the requirements are met. We compare the performance of noisy mappings with learning curves for curriculum transfer generated through exact mappings, and with learning from scratch. Even with noisy mappings, the automated curriculum transfer outperforms learning from scratch, and performs comparable to the automated curriculum transfer with exact mappings.

References

[1] Shukla, Y., Thierauf, C., Hosseini, R., Tatiya, G., and Sinapov, J. (2022) ACuTE: Automatic Curriculum Transfer from Simple to Complex Environments. To appear in proceedings of the 2022 ACM Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

[2] Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8, 3-4 1992.

[3] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Wierstra, D. (2016, January). Continuous control with deep reinforcement learning. In ICLR (Poster).

[4] Andrychowicz, Marcin, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight experience replay." Advances in neural information processing systems 30 (2017).