A Appendix

A.1 L2E Implementation

We use the open-source Stable Baselines3 Raffin et al. (2019) SAC agent for all experiments. To allow for a fair comparison, we use the same hyperparameters for the SAC algorithm in all L2E and HER runs. Both L2E and the HER baseline are trained using rollout episodes of length 250.

For the actor and critic networks, we use 6 hidden layers with sizes decreasing geometrically from 1024 to 64. We train the networks using 0.0003 as the learning rate in batches of size 256. The replay buffer stores 1×10^6 transitions. The L2E and HER agents need ca. 1GB of GPU memory and are trained using an Nvidia RTX-3090 GPU and AMD EPYC 7502 CPU. For full details on the implementation, please also refer to the code at github.com/ischubert/l2e and in the supplementary material.

A.2 Details on Baselines

We compare L2E against several baselines that we describe in the following.

A.2.1 Plan Execution

As basic baseline, we compare against direct execution of the plan using a simple controller. Here, at each step the point p_k in the planned trajectory that is closest to the current position s is selected. To measure the distance, we use $d(\cdot, \cdot)$ introduced in section 3.2. The action is then calculated as the delta between the end effector position at this point and the following point p_{k+1} of the trajectory. We stop the run if either the goal or the time limit of 250 steps is reached.

A.2.2 Plan Execution with Inverse Dynamics Model

We use an inverse dynamics model to transfer the plan-based controller described in section A.2.1 to the simulated environment. This can be seen as a transfer method as introduced by Christiano et al. (2016).

We learn an inverse dynamics model (IM) $\phi : \mathbb{S} \times \mathbb{S} \to \mathbb{A}$ from simulation data. From input pairs of state s and desired next state s', this model learns to predict feasible actions $a = \phi(s, s')$. To collect data, we reset the environment to random initial states, and then perform random actions. In order to bias the data collection towards more informative samples, an action leading the end effector directly towards the box is randomly selected at 10% of all time steps. After 250 time steps, we reset and repeat the process. We then train a neural network to predict actions from pairs of state and next state using mean squared error loss. We use the open-source Pytorch (Paszke et al., 2019) package.

Evaluation is done in the same way as described in section A.2.1, with the only difference that at each step we calculate the action $a = \phi(s, p_{k+1})$ using the inverse dynamics model. Since the planned trajectory does not specify the box's orientation, we always assume the desired orientation to be parallel to the table.

A.2.3 Planned Subgoals and RL

Here we use the planner to create a sequence of subgoals that can be navigated by an RL agent. This approach is inspired by PRM-RL (Faust et al., 2018).

The subgoal is selected as a box position on the plan that is 0.3 ($1/10$ of the table length) away from the current box position. Once the subgoal is reached (tolerance 0.1), the next subgoal is selected.

Thus, long-term planning is provided by the planning module. The RL agent on the other hand is tasked with learning the dynamics of the system to reach relatively short-term goals. This RL agent is conditioned on subgoals, and trained using HER. We use the sampling strategies "Future 1" in the basic pushing environment (see Figure 1a), and "Episode 5" in the obstacle pushing environment (see Figure 1c). These performed best on the respective environment when used without any planning.

A.2.4 HER

We use the open-source Stable Baselines3 Raffin et al. (2019) implementation of HER to learn universal goal-conditioned policies for the goal-conditioned MDP M_G. To allow for a fair comparison, we run several experiments comparing different HER replay strategies. As described in section A.1, we use SAC with the same hyperparameters both for HER and for L2E.

A.3 Plan-conditioned Policies in a Constantly Changing Environment

Since it learns a plan-conditioned policy in contrast to a goal-conditioned policy, the L2E agent can adapt to changed scenarios where some of the initially successful strategies become infeasible. If, e.g., the environment changes, it suffices to only update the planner's crude model of the environment so that it creates plans that are feasible again. These can then directly be fed into the policy without retraining. This is an advantage that is specific to learning plan-conditioned policies over goal-conditioned policies. We demonstrate this using a 2D maze toy example with moving obstacles in the following.

A.3.1 Experimental Setup

We consider a 2D world of size 1×1. In a single step, the agent can move to any point within a radius of 0.1. This movement is distorted by random noise with maximum absolute value 0.01. If an obstacle is in the way or the agent reaches the border, nothing happens.

At the end of each episode of length 250, the agent is reset to a random position and a random goal is chosen. Importantly, 3 rectangular obstacles of random size are also created and are set to random positions. Thus, the environment changes at the end of each episode. A simple RRT planner (we use an implementation by Zeng (2019)) then plans a feasible sequence of length 20 through the state space. Three random configurations of the environment are shown in Figure 2a to Figure 2c.

The resulting plan is input to the L2E agent, and is given in full to the plan-conditioned L2E policy. We use the $S^{\text{bias}}_{100,1000}$ strategy here. We compare the L2E agent to using a HER agent with replay strategy "Future 5". The HER policy is only given the goal as input.

A.3.2 Results

The results are shown in Figure 2d. The HER agent learns a goal-conditioned policy, and does not have access to the plan. Thus, it is unable to effectively generalize to a changed environment. In contrast, the L2E agent learns to execute the entire plan. It can generalize to unseen environments as long as it is provided with feasible plans by the RRT planner. As a result, the L2E agent consistently shows success rates around 90%.

A.4 Optimizing HER and L2E replay strategy

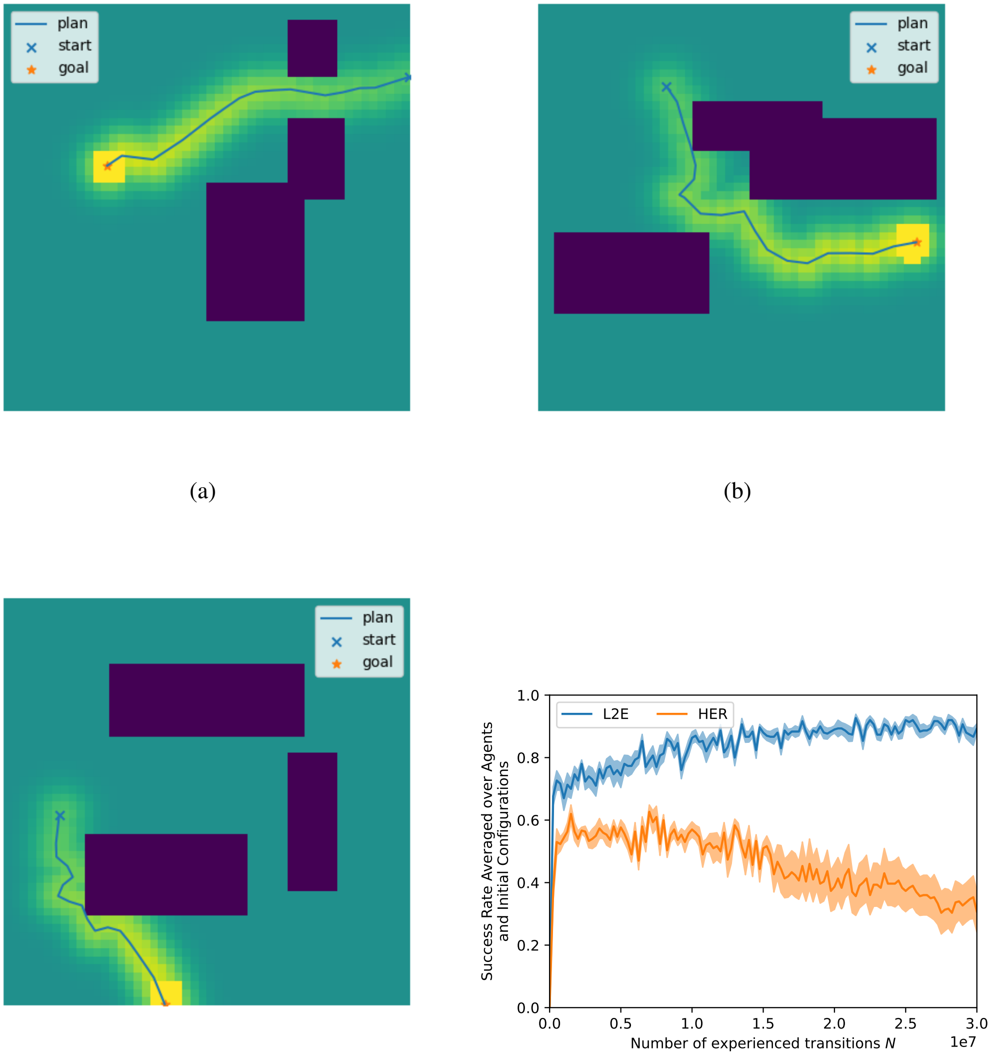
For the basic pushing example (see Figure 1a), we optimize the replay strategies of both HER and L2E to allow for a fair comparison.

A.4.1 Optimizing HER Replay Strategy

To allow for a fair comparison, we spent a considerable amount of effort to optimize the HER replay strategy. Multiple experiments comparing different choices for the HER replay strategy are shown in Figure 3a. Please also refer to Andrychowicz et al. (2017) for a detailed description of the sampling strategies listed. We find that using one future achieved goal from the same episode ("Fu 1") for replay seems to result in the best long-term performance. The strategies "Ep 1" and "Ep 5" perform comparably well.

A.4.2 Optimizing L2E Replay Strategy

We compare different replay strategies for L2E in Figure 3b. We find that using biased sampling is crucial to achieve high sample efficiency. The uniform strategy S^{uni}_{1000} catches up to the biased agent only after ca. 8 Million transitions (not shown in the figure). However, overly pronounced biasing is



(a) (b)

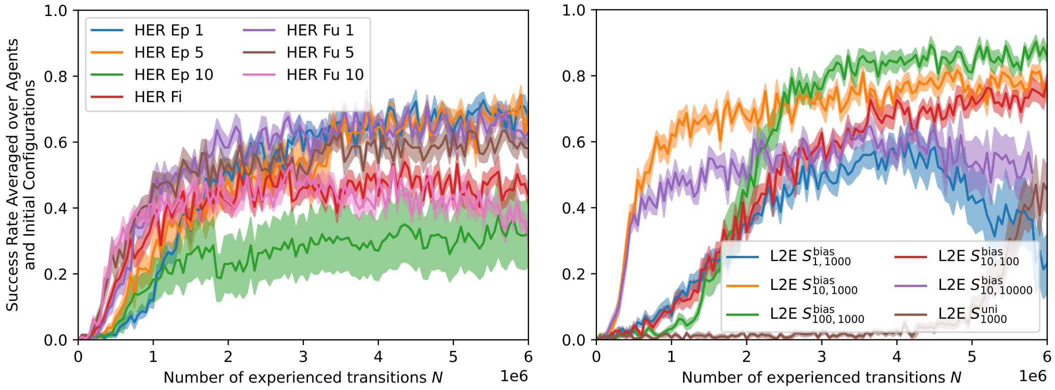
(c) (d)

Figure 2: (a, b, c) The positions of the obstacles in the 2D maze environment change after each episode. Three randomly selected configurations are shown here; obstacles are shown in dark purple. For each configuration, a feasible plan from start to goal is created by an RRT planner (we use an implementation by Zeng (2019)). The yellow heatmap indicates the value of the FV-RS shaping reward created by the L2E agent. (d) Since the L2E agent learns to execute plans, it learns to become consistently successful in this environment by following the plan. In contrast, the HER agent is unable to effectively generalize to a changed environment.

also disadvantageous (see $S^{\text{bias}}_{10,10000}$ in comparison to $S^{\text{bias}}_{10,1000}$). In this case, the L2E agent does not learn to generalize well to the test scenario.

A.5 Experiments on Plan Encoding

For the results reported in section 5.4, we encode plans analytically using box positions as described in section 5.2 We will investigate in the following if (1) an encoding should be used at all and (2) whether it can be learned as well.



(a)

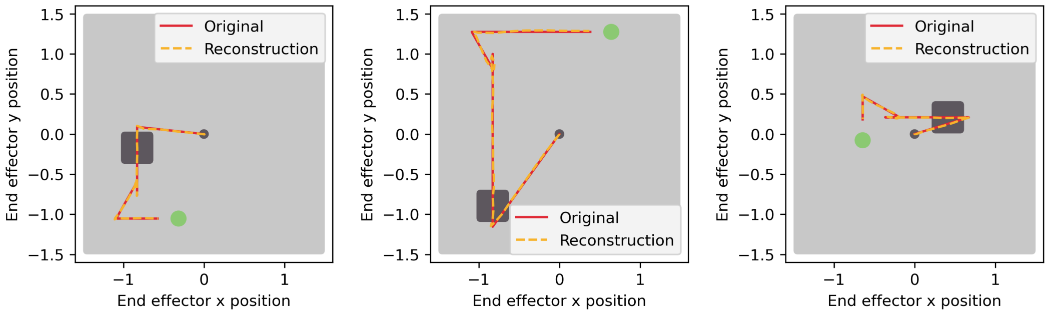
(b)

Figure 3: For the basic pushing example (see Figure 1a), we optimize the replay strategies of both HER and L2E to allow for a fair comparison. (a) For HER, we find that the "Episode 1", "Episode 5", and "Future 1" replay strategies result in comparable long-term performance, but "Future 1" seems to allow for slightly faster learning in the beginning. (b) For L2E, we find that using biased replay significantly improves sample efficiency. The L2E agents with uniform replay strategy S^{uni}_{1000} only catches up to the biased agents at $N \approx 8 \times 10^6$ (not shown). Out of the strategies tested, $S^{\text{bias}}_{100,1000}$ performed best.

We contrast three different ways to encode the plan for the basic pushing environment (see Figure 1a):

1. Analytical encoding into a 4D latent space (entries correspond to intermediate box positions)
2. Learned encoding using a variational autoencoder (VAE) trained with mean squared error (MSE) reconstruction loss, again using a 4D latent space.
3. No encoding (plans are given in full to the policy).

For option 2, we use an encoder made up of 6 fully connected layers with sizes decreasing geometrically from 1024 to 32. For the decoder, we use the same architecture in opposite direction. Figure 4 shows plans reconstructed by the trained VAE, together with the ground truth.



(a)

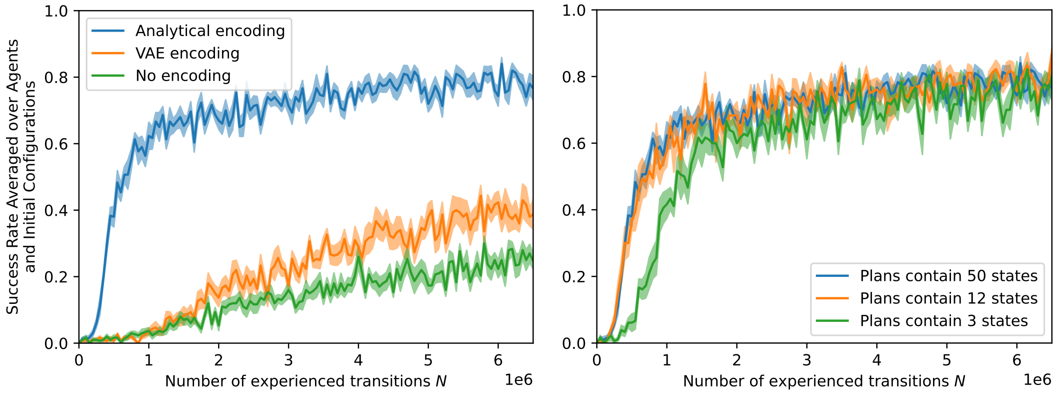
(b)

(c)

Figure 4: Plans reconstructed by the trained VAE, together with the ground truth. The table is indicated in light gray, the initial configuration is indicated in dark gray, and the box's goal position is shown in green. The plans are sequences of length 50, containing 6D vectors of the planned positions of end effector and box. Shown here are only planned x- and y-positions of the end effector.

For the basic pushing environment, Figure 5a shows the results of the 3 options introduced above.

These results show that using an encoding is beneficial, even if it has to be learned. At least for the present example, using an analytical encoding was however more effective.



(a) (b)

Figure 5: Analysis of (a) plan encoding methods and (b) the influence of plan density for the L2E $S^{\text{bias}}_{10,1000}$ agent in the basic pushing environment (see Figure 1a). (a) We find that using any encoding significantly outperforms using no encoding at all. Furthermore, using an analytical encoding, at least in this case, outperforms using a learned encoding of the same dimensionality. (b) We find that there is no significant difference between using 50 states and using 12 states. When using 3 states, the less informative reward signal leads to a slightly flatter learning curve in the beginning. Analytical encoding is used for all runs shown in (b).

A.6 Experiments on Plan Density

Throughout this work, plans are represented as a trajectory of length 50 for the basic pushing environment and 100 for the obstacle pushing environment, consisting of 6D elements representing end effector position and box position.

Less dense plans result in a less informative reward shaping signal for the L2E agent. To understand how much L2E relies on high-quality dense plans, we compare training the L2E agent using planned state sequences of different density. This comparison is performed in the basic pushing environment (see Figure 1a). For the same distribution of plans Figure 5b shows the results for using 50, 12, or 3 states as a representation.

We find that in this example, there is no significant difference between using 50 states and using 12 states. When using 3 states, the less informative reward signal leads to a slightly flatter learning curve in the beginning, but the agent seems to recover from this later on.

In this environment, L2E seems to be largely robust to lower plan densities.

A.7 Basic Pushing Environment: Longer Runs for the best performing L2E Agent

Figure 6 shows results for running the L2E ${S^{\text{bias}}}_{100,1000}$ longer than shown in Figure 1b. The agent improves further. After 30 million time steps, the mean success rate averaged over 10 independently trained agents and 30 randomized test rollouts is around 90%.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our claims made in the abstract and introduction are demonstrated in the experiments section.
   (b) Did you describe the limitations of your work? [Yes] See section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We see no immediate societal impact of our work.

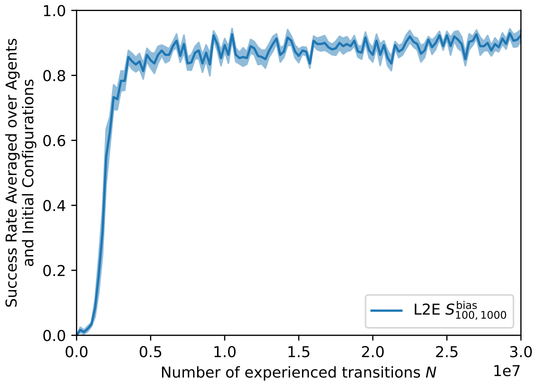


Figure 6: Results for running the L2E $S^{\text{bias}}{}_{100,1000}$ longer than shown in Figure 1b. The agent improves further. After 30 million time steps, the mean success rate averaged over 10 independently trained agents and 30 randomized test rollouts is around 90%.

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We state the requirements for the background methods to be applicable in section 3, for L2E to be applicable in section 4, and further discuss them in section 6.
   (b) Did you include complete proofs of all theoretical results? [N/A] We do not state formal theorems in this work.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include the entire open-source code base to fully reproduce all figures in this paper in the supplementary material, and at github.com/ischubert/l2e. This includes README files to explain how it can be used. Although this should be sufficient to easily reproduce all results in this paper, we will also make the data files containing the results shown in this paper available upon request.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section A.1 and section 5.4.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All figures show mean values as well confidence intervals (standard deviation of the mean).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section A.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use open-source libraries that are cited where used.
   (b) Did you mention the license of the assets? [Yes] We mention in the text that the assets are open source. More details on the exact license can be found in the references.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See question 3(a).
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We do not use external data assets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not use external data assets.
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We do not use crowdsourcing or conducted research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We do not use crowdsourcing or conducted research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We do not use crowdsourcing or conducted research with human subjects.