

Multi-Strategy Deployment-Time Learning and Adaptation for Navigation under Uncertainty

Supplementary Material

A CycleGAN Implementation and Training

We use the official PyTorch implementation of CycleGAN provided by Zhu et al. [19] on GitHub¹. As mentioned in Sec. 4.3, we train two models: one for 50 epoch and another for 100 epochs. We use the default parameters for training except that we use a batch size of 8 and no learning rate scheduling is done. We use learning rate of 0.0002 with Adam optimizer. The images are of size 512×128 and we perform default resizing and cropping as a preprocessing step. 1261 images from 10 distinct maps in Maze-Green (where π_{PRETRAIN} is trained) are set aside as target domain images, and we sample 1300 images from the source domain (either Maze-Gray or Maze-Blue) collected during deployment for training the CycleGAN model. Using only 1300 images from deployment-time environments additionally helps to get an unbiased estimate of performance when replaying CycleGAN-adapted policy π_{CYCLEGAN} on older trials from which the images were collected.

B Learning over Subgoals Planning: Subgoal Property Estimator Network Implementation and Training

As discussed in Sec. 3.3, all learning-informed policies rely on the learning over subgoals planning (LSP) abstraction for planning, a model-based approach that relies on a learned model to estimate *subgoal properties*: statistics of unseen space associated with each of the robots temporally-extended high-level actions to explore unseen space. The subgoal property estimator network \mathcal{N}_θ corresponding to π_{PRETRAIN} is trained with data-collected in 500 distinct maps in Maze-Green where the robot navigates using the $\pi_{\text{NONLEARNED}}$ policy. Data labelling procedure is similar to the one described for π_{SCRATCH} (Sec. 4.3) except that at training time the underlying map is known and so can be used to generate ground truth labels for subgoal properties P_S , R_S and R_F corresponding to all subgoals.

The subgoal property estimator network \mathcal{N}_θ is trained via supervised learning using the data collected during an offline training phase. Our neural network architecture and training procedure resemble that of Paudel and Stein [13]. The network takes as input a 512×128 panoramic image centered on a subgoal, relative distance to the subgoal and relative distance to the goal. The image is encoded by passing through 4 convolutional layers and then concatenated with features corresponding to relative distances to subgoal and goal after which the concatenated features are passed through 9 convolutional layers and finally 5 fully connected layers to output 3 subgoal properties P_S , R_S and R_F . We use a learning rate of 0.002 with a decay factor of 0.5 every epoch and train for 8 epochs with Adam optimizer. We use cross-entropy loss for learning logits associated with P_S and L2 loss for learning R_S and R_F . The deployment-time training of subgoal property estimators for π_{SCRATCH} follows a similar architecture and procedure.

C Offline Alt-Policy Replay Details

As discussed in Sec. 3.2, we use offline alt-policy replay to replay the behavior of a policy without deploying it. To replay the behavior of a policy π' , we leverage the record \mathcal{Z}_k collected during trial k under a deployed policy π . At every time step during offline alt-policy replay, the robot leverages the final partial map m_{final} observed in trial k to simulate the laser scan and updates its observed map as the robot moves. The frontiers—boundaries between free and unknown space—revealed in the observed map corresponds to the subgoal-actions that the robot can take to explore the region. To get the robot-view panoramic image corresponding to a subgoal-action, we retrieve

¹<https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>

existing image in record \mathcal{Z}_k that is closest to and in line of sight to the subgoal and recenter it at the subgoal. This image is used to estimate the subgoal properties P_S , R_S and R_F using a neural network corresponding to π' which is then used to compute the next high-level action using Eq. (4). The robot then simulates the low-level motion primitive to move towards the selected subgoal and this process is repeated. At any point during simulated navigation, if the robot attempts to enter a region that is unknown in the final partial map m_{final} via a frontier, we mask that frontier and force the robot to pick a different subgoal-action. The net distance travelled to reach the goal via this procedure is the replay cost of policy π' .

D Cross Validation for Reevaluating Older Trials

As mentioned in Sec. 4.3, we use 5-fold cross validation to get an unbiased estimate of the performance of updated policies. Since our policies trained during deployment from scratch (π_{SCRATCH}) are based on the same data in record \mathcal{Z} that is also used for offline alt-policy replay to reevaluate older trials after updating the policies, such cross-validation approach overcomes the risk of overestimation of performance during replay due to data leak. With 5-fold cross validation, 5 policies are trained, each on the data from four-fifth of older trials, and replayed on the remaining one-fifth of the trials to get the revised performance estimates for all older trials. Finally, a new policy is trained on data from all previous trials and made available for the robot to choose from in the next trial.

E Sample Images from Deployment-Time Visual Domain Adaptation

Image samples transformed from deployment-time environments (Maze-Gray or Maze-Blue) to the training-time environment (Maze-Green) are shown in Fig. 6. The CycleGAN models trained after 40th trial are used to generate the images.

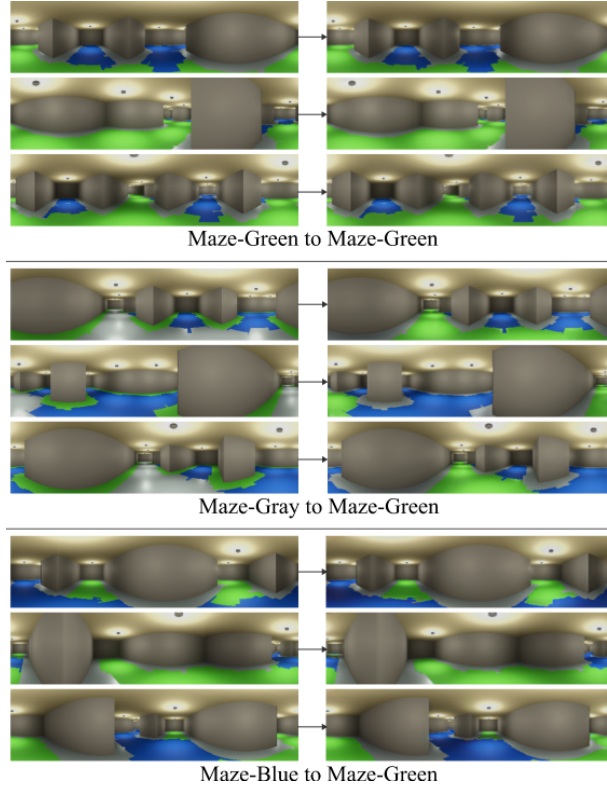


Figure 6: Images transformed from deployment-time environments (input) to look like training-time environments (output) with CycleGAN-based visual domain adaptation.