Supplementary material for adversarial actor-critic algorithm for task and motion planning problems using planning experience

Hyperparameter settings

We describe hyperparameters and learning rates used for different algorithms. For each algorithm, we have a hyperparameter λ. For PPO and GAIL, this refers to the clipping ratio limit, for DDPG, this is the soft-update parameter, and for ADMON this is the adversarial objective control parameter.

Algorithm	lr_α	lr_θ	λ
PPO	1e-3	1e-4	0.3
ADMON	1e-3	1e-4	2
GAIL	1e-3	1e-4	0.2
DDPG	1e-3	1e-4	0.01

Table 1: Hyperparameter setting for the conveyor belt domain

Algorithm	lr_α	lr_θ	λ
PPO	1e-4	1e-4	0.3
ADMON	1e-4	1e-4	2
GAIL	1e-4	1e-4	0.3
DDPG	1e-4	1e-4	0.001

Table 2: Hyperparameter setting for the object fetch domain

Architecture descriptions

The architecture for the critic is exactly as described in Figure 2b of the main paper - we describe the number of hidden layers and neurons, and activation functions here.

Conveyor belt domain

For this domain, we only learn a place operator policy. We have 982 key configurations. For the architecture in each of the red boxes of Figure 2b, we use five hidden layers with each of 64 neurons. The outputs at the end of these five hidden layers are the 982 green neurons, each representing the importance of each key configuration for predicting the value of the given place base pose. We then max-pool these outputs with a pool size of two. Lastly, we connect these max-pooled outputs with two hidden layers each with 64 neurons. For the actor, we have the exactly the same architecture, except that what gets fed into the red box is only the collision information $\phi^{(i)}$. For each layer, we use rectified linear activation function, except at the last layer, where we use linear activation function.

Object fetching domain

For this domain, we learn both pick and place operator policies. We have 1028 key configurations. For the place operator, we use exactly the same architecture as the conveyor domain, except the input vector consists of the current robot configuration in addition to the robot place base pose.

For the pick operator critic, we do additional layers to convert the relative base pose to a global base pose. We take the current object pose and the relative pick base pose, and add them together using a *Add* layer to get the global robot pick base pose. We then take this pick base pose and the current robot base pose, which is also expressed in the global coordinate frame, and feed it to the red box in Figure 2b along with the key configuration vector. The architecture in the red box is the same as the place operator. The green-neuron output is max-pooled as done in the place operator as well, and gets fed into a dense layer with 64 hidden units. On a separate network, we take the object shape and the grasp and connect it into a single dense layer with 64 hidden units. The outputs from the green neurons and grasp network are then concatenated, and gets connected to a single dense layer with 64 hidden units. This gets connected to an output layer. For the actor, we have exactly the same network, except that the input does not have grasp and relative pick base pose; these are outputs of the actor.

Description of ϕ_{fetch} for the fetching domain

As described in the main article, given a new problem instance, we begin by sampling a pick base configuration and a grasp for the target object for the target object, and calling the motion planner to obtain a "fetching path" for the target object that will make the robot to move to the selected pick configuration, where collisions with the movable obstacles are allowed. To encode this fetching path, we re-use the key configurations to make another binary input vector $\phi^{(i)}_{fetch}$ in addition to $\phi^{(i)}$, with $\phi^{(i)}_{fetch} = 1$ if key configuration i is within a fixed distance threshold of the fetching path

and 0 otherwise. This gets passed along with the collision information to the actor and critic neural networks.