Appendices

A Experimental Setup Details

Both our real and simulated environments use the following 6-dimensional control scheme:

`[x,y,z,wrist,gripperOpen,moveToNeutral]`

where the `x,y,z` dimensions command changes in the end-effector's position in 3D space, `wrist` commands changes in the angle of the wrist, `gripperOpen` is a continuous value from $[-1, 1]$ that triggers the gripper to close completely when less than -0.5 and open completely when greater than 0.5, and `moveToNeutral` is also a continuous value from $[-1, 1]$ that triggers the robot to move to its starting joint position when greater than 0.5.

A.1 Data Collection Policies

We describe our scripted data collection policies in this section. More details can be found in Algorithms 1-4.

Scripted grasping. Our scripted grasping policy is supplied with the object's (approximate) coordinates. In simulation, this information is readily available, while in the real world we use background subtraction and a calibrated monocular camera to approximately localize the object. Note that this information does not need to be perfect, as we add a significant amount of noise to the scripted policy's action at each timestep. After the object has been localized, the scripted policy takes actions that move the gripper toward the object (i.e action $\leftarrow$ object_position $-$ gripper_position). Once the gripper is within some pre-specified distance of the object, it executes the grasp action (which is a discrete action). Note that this distance threshold is also randomized – sampled from a Gaussian distribution with a mean of 0.04 and a standard deviation of 0.01 (in meters). For the simulated pick and place environment, the scripted policy for grasping obtains a success rate of 50%, while the success rate is 30% for the drawer environment. For the real world drawer environment, the scripted success rate is 30%.

Scripted pick and place. The pick part of the pick and place scripted policy is identical to the grasping policy described above. After the grasp has been executed, the scripted policy uniformly randomly selects a point in the workspace to place the object on, and then takes actions to move the gripper above that point. Once within a pre-specified (and randomized) distance to that point, it executes the gripper open action. The policy is biased to sample more drop points that lie inside the box to ensure we see enough successful pick and place attempts. Once the object has been dropped, the robot returns to its starting configuration (using the moveToNeutral action).

Scripted place. This policy is used in scenes where the robot is already holding the object at the start of the episode. The placing policy is identical to the place component of the pick and place policy described above.

Drawer opening and closing. The scripted drawer opening policy moves the gripper to grasp the drawer handle, then pulls on it to open the drawer. The drawer closing policy is similar, except it pushes on the drawer instead of pulling it. To introduce variability into the data collection process and to ensure that there is irrelevant data in the prior dataset as well, the drawer handle position passed to the scripted policy is incorrect with a probability of 0.25. Further, Gaussian noise is added to the policy actions at every timestep. After the opening/closing is completed, the robot returns to its starting configuration.

Ending scripted trajectories with return to starting configuration We ended the scripted trajectories with a return to the robot's starting configuration. We believe that this return to starting configuration increases the state-distribution overlap of the various datasets collected from scripted policies, making it possible to stitch together relevant trajectories from the prior dataset to extend the skill learned for the downstream task.

Algorithm 1 Scripted Grasping

threshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps $\leftarrow$ 25
for t **in** (0, numTimesteps) **do**
    objPos $\leftarrow$ object position
    eePos $\leftarrow$ end effector position
    objGripperDist $\leftarrow$ distance(objPos, eePos)
    if objGripperDist $>$ threshold **then**
        action $\leftarrow$ objPos $-$ eePos
    else if gripperOpened **then**
        action $\leftarrow$ close gripper
    else if object not raised high enough **then**
        action $\leftarrow$ lift upward
    else
        action $\leftarrow$ 0
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

Algorithm 2 Scripted Pick and Place

threshold, dropDistThreshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps $\leftarrow$ 30
for t **in** (0, numTimesteps) **do**
    eePos $\leftarrow$ end effector position
    dropPos $\leftarrow \begin{cases} \text{point above box} & \text{w/ prob. } 0.75 \\ \text{point outside box} & \text{w/ prob. } 0.25 \end{cases}$
    objectDropDist $\leftarrow$ distance(eePos, dropPos)
    if object not grasped **AND** objectDropDist $>$ dropDistThreshold **then**
        Execute grasp using Algorithm 1
    else if objectDropDist $>$ boxDistThreshold **then**
        action $\leftarrow$ dropPos $-$ eePos
        action $\leftarrow$ lift upward
    else if object not dropped **then**
        action $\leftarrow$ open gripper
    else
        action $\leftarrow$ 0
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

Algorithm 3 Scripted Drawer Opening

threshold $\sim \mathcal{N}(0.04, 0.01)$
error $\sim \mathcal{U}(-0.2, 0.2)$
numTimesteps $\leftarrow$ 30
for t **in** (0, numTimesteps) **do**
    handlePos $\leftarrow$ handle center position
    targetPos $\leftarrow \begin{cases} \text{handlePos} & \text{w/ prob. } 0.75 \\ \text{handlePos+error} & \text{w/ prob. } 0.25 \end{cases}$
    eePos $\leftarrow$ end effector position
    targetGripperDist $\leftarrow$ dist(targetPos, eePos)
    if targetGripperDist $>$ threshold **AND** not drawerOpened **then**
        action $\leftarrow$ targetPos $-$ eePos
    else if not drawerOpened **then**
        action $\leftarrow$ move left to open drawer
    else if gripper not above drawer **then**
        action $\leftarrow$ lift upward
    else
        action $\leftarrow$ `moveToNeutral`
        End scripted trajectory
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

Algorithm 4 Scripted Drawer Closing

threshold $\sim \mathcal{N}(0.04, 0.01)$
numTimesteps $\leftarrow$ 30
for t **in** (0, numTimesteps) **do**
    drawerPos $\leftarrow$ drawer bottom center position
    eePos $\leftarrow$ end effector position
    if not drawerClosed **AND** gripper not next to drawer **then**
        action $\leftarrow$ go next to drawer
    else if not drawerClosed **then**
        action $\leftarrow$ eePos $-$ drawerPos
        // (this pushes the drawer closed)
    else
        action $\leftarrow$ `moveToNeutral`
        End scripted trajectory
    end if
    noise $\sim \mathcal{N}(0, 0.2)$
    action $\leftarrow$ action + noise
    $(s, r, s') \leftarrow$ env.step(action)
end for

A.2 Neural Network Architectures

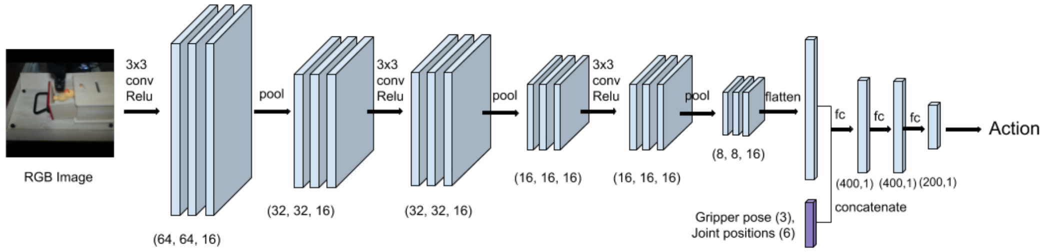

Figure 6: **Neural network architecture for real robot experiments.** We map high dimensional image observations to low level robot commands, such as desired position of the end-effector, and gripper opening/closing.

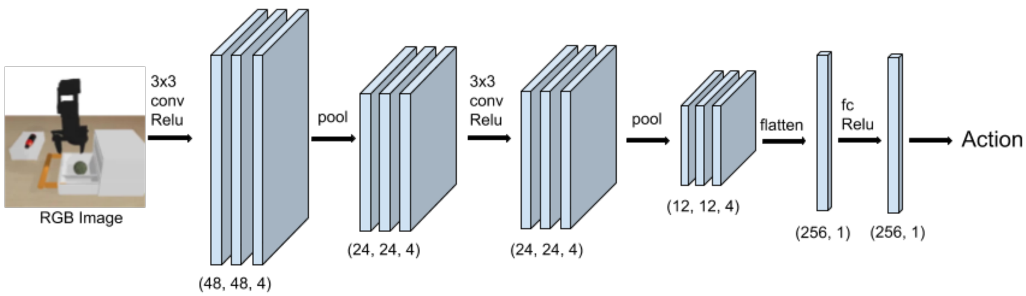

Figure 7: **Neural network architecture for simulated experiments**. Note that we omit the information about the gripper position and finger angle for our simulated experiments, since including this information did not seem to make a difference in our simulated experiments.

Figures 6 and 7 show the neural network architectures used in our real world and simulated experiments, respectively. We experimented with several different architectures (varying the number of convolutional layers from 2 to 4, and varying the number of filters in each layer from 4 to 16), and found these two architectures to perform the best. Note that the real world neural network has substantially more parameters, which is likely due to the increased complexity of real world observations.

A.3 Hyperparameters for Reinforcement Learning

We used the conservative Q-learning (CQL) [20] algorithm for chaining behaviors. We now present the hyperparameters used by our method below:

- **Discount factor**: 0.99 (identical to SAC, CQL),
- **Learning rates**: Q-function: 3e-4, Policy: 3e-5 (identical to CQL),
- **Batch size**: 256 (identical to SAC, CQL),
- **Target network update rate**: 0.005 (identical to SAC, CQL),
- **Ratio of policy to Q-function updates**: 1:1 (identical to SAC, CQL),
- **Number of Q-functions**: 2 Q-functions, $\min(Q_1, Q_2)$ used for Q-function backup and policy update (identical to SAC, CQL),
- **Automatic entropy tuning**: True, with target entropy set to $-\log|\mathcal{A}|$ (identical to SAC),
- **CQL version**: CQL($\mathcal{H}$) (note that this doesn't contain an additional $-\alpha \log \pi(\mathbf{a}|\mathbf{s})$ term in the Q-function backup),
- α **in CQL**: 5.0 (we used the non-Lagrange version of CQL($\mathcal{H}$)),

- **Number of negative samples used for estimating logsumexp:** 1 (instead of the default of 10 used in CQL; reduces training wall-clock time substantially when learning from image observations)
- **Initial BC warmstart period**: 40k gradient steps for drawer task, 10k for pick and place,
- **Evaluation maximum trajectory length**: 80 timesteps for simulated drawer environment, 30 timesteps for simulated pick and place. For real world drawer environment, this value is equal to 35 timesteps.

B Learning Curves

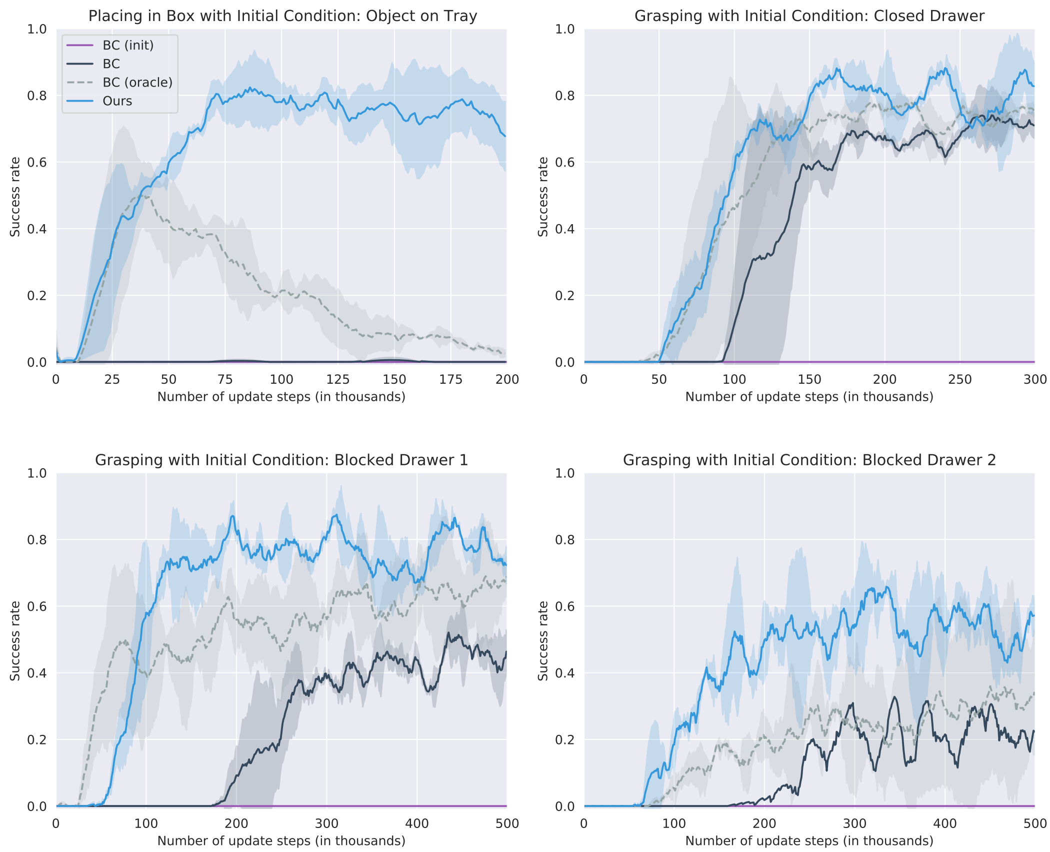

Figure 8: **Learning curves for simulated experiments by method and initial condition**. Here we compare the success rate curves of our method to the three behavioral cloning baselines in the four settings of Table 1 where prior data is essential for solving the task: the place in box task with the object starting in the tray (upper left), as well as the grasp from drawer task with a closed drawer (upper right), blocked drawer 1 (lower left), and blocked drawer 2 (lower right).

Here are detailed learning curves for the experiments we summarized in Table 1. Note that the x-axis here denotes number of update steps made to the policy and Q-function, and not the amount of data available to the method. Since we operate in an offline reinforcement learning setting, all data is available to the methods at the start of training. We see that our method is able to achieve a high performance across all initial conditions for both the tasks. We substantially outperform comparisons to prior approaches that are based on pretraining using behavior cloning, including an oracle version (shown above in a grey dashed line) that only trains on manually selected successful trajectories.