# 7 Appendix

## 7.1 Algorithm

**Algorithm 2** CRSfD

**Input:** $Env$ Environment for the new task $M_i$; $\theta^{\pi}$ initial policy parameters; $\theta^{Q}$ initial action-value function parameters; $\theta^{Q'}$ initial target action-value function parameters; N target network update frequency.
**Input:** $B^E$ replay buffer initialized with demonstrations. $B$ replay buffer initialized empty. $K$ number of pre-training gradient updates. $d$ expert buffer sample ratio. $batch$ mini batch size.
**Input:** $\theta^V$ initial value function (potential function), original task discount factor $\gamma_0$.
**Output:** $Q_\theta(s,a)$ action-value function (critic) and $\pi(.|s)$ the policy (actor).
# Estimate value function from demonstration.
**for** step $t$ **in** {0,1,2,...T} **do**
  Sample with $batch$ transitions from $B^E$, calculate their Monte-Carlo return with discount factor $\gamma_0$.
  Estimate $V_\theta(s)$ conservatively by equation **??**
**end for**
# Interact with $Env$.
**for** episode $e$ **in** {0,1,2,...M} **do**
  Initialize state $s_0 \sim Env$
  **for** step $t$ **in** episode length {0,1,2,...T} **do**
    Sample action from $\pi(.|s_t)$
    Get next state and natural sparse reward $s_{t+1}, r_t$
    Shape reward by: $r'_t = r_t + \gamma_i V(s_{t+1}, \theta^V) - V(s_t, \theta^V)$
    Add single step transition $(s_t, a_t, r'_t, s_{t+1})$ to replay buffer $B$.
  **end for**
  **for** update step $l$ **in** {0,1,2,...L} **do**
    Sample with prioritization: $d * batch$ transitions from $B^E$, $(1-d) * batch$ transitions from $B$. Concatenate them into a single batch.
    Perform SAC update for actor and critic:$L_{Actor}(\theta^{\pi}), L_{Critic}(\theta^{Q})$.
    **if** step $l \equiv 0 \pmod N$ **then**
      Update target critic using moving average:$\theta^{Q'} = (1-\tau)\theta^{Q'} + \tau\theta^{Q}$
      Decrease expert buffer sample ratio: $d = d - \delta$ if $d > 0$.
    **end if**
  **end for**
**end for**

## 7.2 Implementation Details

We implemented our CRSfD algorithm and the baseline algorithms in PyTorch and the implementation can be found in the supplementary materials. Simulated environments are based on robosuite framework https://github.com/ARISE-Initiative/robosuite. Our CRSfD algorithm is based on https://github.com/denisyarats/pytorch_sac_ae while baseline algorithms are based on https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail and https://github.com/ku2482/gail-airl-ppo.pytorch.

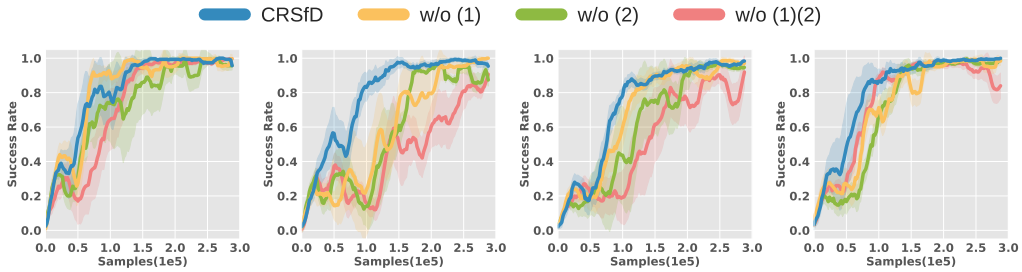
## 7.3 Videos

Videos for simulated environments and real world environments can be found in the supplementary materials.

## 7.4 Ablations

As mentioned in section 4, we make two improvements over the reward shaping method to encourage the agent to explore around the demonstrations conservatively. (1) Regress value function of OOD states to zero. (2) Use a larger discount factor in new tasks.

We ablate these 2 improvements and compare their performance on more environments, as show in Figure 6.




(a) Hole "1" (b) Hole "2" (c) Hole "3" (d) Hole "4"

Figure 6: Ablation studies of the conservativeness techniques. (1) means regressing value function to zero for OOD states. (2) means setting larger discount factors.

### 7.5 Proof for theorem

**Theorem 1** *For task $M_0$ with transition $T_0$ and new task $M_k$ with transition $T_k$, define total variation divergence $D_{TV}(s,a) = \Sigma_{s'}|T_0(s'|s,a) - T_k(s'|s,a)| = \delta$. If we have $\delta < (\gamma_k - \gamma_0)\mathbb{E}_{T_2(s'|s,a)}[V^D_{M_0}(s')]/\gamma_0 \max_{s'} V^D_{M_0}(s')$, then following the expert policy in new task will result in immediate reward greater then 0:*

$$\mathbb{E}_{a\sim\pi(.|s)} r'(s,a) \geq (\gamma_k - \gamma_0)\mathbb{E}_{T_k(s'|s)}[V^D_{M_0}(s')] - \gamma_0 \delta \max_{s'} V^D_{M_0}(s') > 0 \tag{5}$$

**Proof:** For simplify, denote demonstration state value function in original task $V^D_{M_0} = V_1(s)$. Start from the reward shaping equations, and extend $V_1(s)$ for one more time step:

$$\begin{aligned} r'(s,a,s') =& r(s,a,s') + \gamma_k V_1(s') - V_1(s) \\ r'(s,a) =& r(s,a) + \gamma_k \mathbb{E}_{T_k(s'|s,a)}[V_1(s')] - V_1(s) \\ =& (\gamma_k - \gamma_0)\mathbb{E}_{T_k(s'|s,a)}[V_1(s')] + \big(r(s,a) + \gamma_0 \mathbb{E}_{T_k(s'|s,a)}[V_1(s')] - V_1(s)\big) \\ \geq& (\gamma_k - \gamma_0)\mathbb{E}_{T_k(s'|s,a)}[V_1(s')] + (Q^{\pi_1}(s,a) - V_1(s)) - \gamma_0 \delta \max_{s'} V_1(s') \end{aligned} \tag{6}$$

Take expectation on demonstration policies:

$$\mathbb{E}_{a\sim\pi(.|s)} r'(s,a) \geq \mathbb{E}_{a\sim\pi(.|s)}\left[(\gamma_k - \gamma_0)\mathbb{E}_{T_k(s'|s,a)}[V_1(s')]\right] - \gamma_0 \delta \max_{s'} V_1(s') \tag{7}$$

For a sparse reward environment, we have $r(s,a) = 0$ almost everywhere:

$$\begin{aligned} \mathbb{E}_{a\sim\pi(.|s)} r'(s,a) \geq& \mathbb{E}_{a\sim\pi(.|s)}\left[(\gamma_k - \gamma_0)\mathbb{E}_{T_k(s'|s,a)}[V_1(s')]\right] - \gamma_0 \delta \max_{s'} V_1(s') \\ =& (\gamma_k - \gamma_0)\mathbb{E}_{T_k(s'|s)}[V_1(s')] - \gamma_0 \delta \max_{s'} V_1(s') \end{aligned} \tag{8}$$

### 7.6 Increasingly Larger Task Mismatch

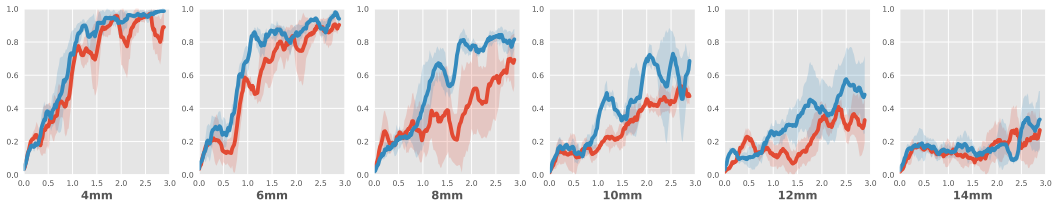



Figure 7: Increasingly larger task mismatch. Experiments are done on hole shape 0 with increasing random hole position.

We can observe that as task difference increases, our method first gradually outperforms baseline methods. When task mismatch are too large, our method gradually loss some performance and has similar performance with baselines.