

# Diff-HySAC: Supplementary Materials

Anonymous Author(s)

Affiliation

Address

email

## 1 A Theoretical Results

2 We show the proof of the lower bound objective function. The proof is based on structured varia-  
3 tional inference [1].

4 The evidence distribution over  $\mathcal{O}$  is  $p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t) = \exp(\frac{1}{\alpha} r(\mathbf{s}_t, \mathbf{a}_t))$ , with  $\alpha > 0$ . We want to  
5 approximate the trajectory distribution  $p(\tau)$  with a variational distribution  $q(\tau)$ , as defined

$$p(\tau) = \left[ p(\mathbf{s}_1) \prod_{t=1}^T p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \right] \exp \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right)$$

6 where we assume full trajectory  $\tau$  is  $\{\mathbf{s}_1, \mathbf{a}_1^{0:K}, \dots, \mathbf{s}_T, \mathbf{a}_T^{0:K}\}$ , which takes into account all sampled  
7 actions of the diffusion policy’s denoising process. Notice that given the denoised action  $\mathbf{a}_t^0$  the  
8 evidence distribution, dynamics and rewards are conditionally independent from  $\mathbf{a}^{1:K}$ .

$$\begin{aligned} p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t^{0:K}) &= p(\mathcal{O}_t | \mathbf{s}_t, \mathbf{a}_t^0) \\ p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t^{0:K}) &= p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t^0). \end{aligned}$$

9 From now, we write  $\mathbf{a}_t$  to mean the denoised action  $\mathbf{a}_t^0$ . Assuming that  $q(\mathbf{s}_1) = p(\mathbf{s}_1)$  and  
10  $q(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) = p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$ , and  $q(\tau)$  is defined by

$$q(\tau) = \left[ q(\mathbf{s}_1) \prod_{t=1}^T q(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \right] q(\mathbf{a}_t^{0:K} | \mathbf{s}_t).$$

11 We now compute the lower bound of the maximum reward likelihood as

$$\begin{aligned} \log p(\mathcal{O}_{1:T}) &= \log \int p(\mathcal{O}_{1:T}, \tau) d\tau \\ &= \log \int p(\mathcal{O}_{1:T}, \tau) \frac{q(\tau)}{q(\tau)} d\tau \\ &= \log \mathbb{E}_{\tau \sim q(\tau)} \int \frac{p(\mathcal{O}_{1:T}, \tau)}{q(\tau)} d\tau \\ &\geq \mathbb{E}_{\tau \sim q(\tau)} [\log p(\mathcal{O}_{1:T}, \tau) - \log q(\tau)] \quad (\text{Jensen's inequality}) \\ &= \mathbb{E}_{\tau \sim q(\tau)} [\log p(\mathcal{O}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T}^{0:K}) - \log q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}^{0:K})] \\ &= \mathbb{E}_{\tau \sim q(\tau)} \left[ \sum_{t=1}^T \left( r(\mathbf{s}_t, \mathbf{a}_t) - \alpha \sum_{k=0}^{K-1} \log q(\mathbf{a}_t^{k+1} | \mathbf{a}_t^k, \mathbf{s}_t) \right) \right]. \end{aligned}$$

12 where the variational policy distribution  $q(\mathbf{a}_t^{k+1} | \mathbf{a}_t^k, \mathbf{s}_t)$  is parameterized as  $\pi_\theta(\mathbf{a}_t^{k+1} | \mathbf{a}_t^k, k, \mathbf{s}_t)$ .

## 13 B Algorithm and Training Details

14 **DiffSAC and ConSAC** The pseudo-code of DiffSAC is provided in Algorithm 2. In addition,  
 15 we propose consistency model as RL policy and its maximum entropy policy algorithm, named  
 16 Consistency-SAC (ConSAC). With consistency action inference, the diffusion steps  $K$  can be large,  
 17 but the sub-sequence of time points used by actual inference is a lot smaller. Basically, the sub-  
 18 sequence is a linspace with  $(N - 1)$  sub-intervals,  $\tau_1 = \epsilon$ , and  $\tau_N = K$  where  $N \ll K$  [2]. The  
 19 action inference with learning variance is described in Algorithm 1. The pseudo-code of ConSAC  
 20 has few differences from DiffSAC in Algorithm 2 at: i) the action selection at step 5 that is replaced  
 21 by Algorithm 1, ii) updates at steps 9 and 10 that differentiate through the consistency model policy  
 22 instead of the diffusion policy.

23 **Diff-HySAC and Con-HySAC** The pseudo-code for the Diff-HySAC and Con-HySAC algo-  
 24 rithms are depicted in Algorithm 3. Diff-HySAC is implemented with learning variance [3] and  
 25 adaptive temperature hyperparameter [4]. Con-HySAC uses consistency action inference in Algo-  
 26 rithm 1.

### 27 B.1 Algorithms

---

#### Algorithm 1 Consistency Action Inference with Learning Variance

---

```

1: Given: state  $s$ ,  $f_\theta$ , a sub-sequence of time points  $\{\tau_n\}_{n \in [N]}$ , diffusion steps  $K$ .
2: Initialize mean  $\mathbf{a}_N = \mathbf{0}$ , variance  $\Sigma_N = K^2 \mathbf{I}$ 
3: for  $n = N$  to 1 do
4:   Sample  $\hat{\mathbf{a}}$  from  $\mathcal{N}(\mathbf{a}_n, \Sigma_n)$ 
5:   Compute  $c_{\text{skip}} = 0.25 / ((\tau_n - \epsilon)^2 + 0.25)$  and  $c_{\text{out}} = 0.5 * (\tau_n - \epsilon) / \sqrt{(\tau_n^2 + 0.25)}$ 
6:   Compute  $\mathbf{a}_n, \Sigma_n = f_\theta(s, \hat{\mathbf{a}}, \tau_n)$ 
7:   Compute output  $\mathbf{a}_n = c_{\text{skip}} * \hat{\mathbf{a}} + c_{\text{out}} * \mathbf{a}_n$ 
8: end for
9: return  $\mathbf{a}_1$ 

```

---



---

#### Algorithm 2 DiffSAC

---

```

1: Initialize policy  $\pi_\theta$  and critic networks  $Q_{\phi_1}$  and  $Q_{\phi_2}$ 
2: Initialize the target networks:  $Q_{\phi'_1}$  and  $Q_{\phi'_2}$ 
3: Initialize replay buffer:  $\mathcal{D} = \emptyset$ 
4: while not converge do
5:   Sample action  $\mathbf{a}_t$  from diffusion policy as in Eq. 3, e.g. DDPM sampling
6:   Execute action  $\mathbf{a}_t^0$ , observe  $r_t$  and  $s_{t+1}$ 
7:   Add sample  $\{s_t, \mathbf{a}_t^0, r_t, \mathbf{a}_{t+1}^0\}$  to replay buffer  $\mathcal{D}$ 
8:   Sample a minibatch  $\{s, \mathbf{a}^0, r_t, s'\}$  from  $\mathcal{D}$ 
9:   Update the critic  $L(\phi)$  as in Section 3.3
10:  Update the actor with gradient defined by Eq. 6
11:  Adjust temperature  $\alpha$ 
12:  Update the target networks, i.e. using standard delayed updates like in SAC.
13: end while
14: return final policy  $\pi$ .

```

---

### 28 B.2 Training Details

29 HySAC (Ours): HySAC is implemented as a modification on top of HACMan ([https://](https://github.com/HACMan-2023/HACMan)  
 30 [github.com/HACMan-2023/HACMan](https://github.com/HACMan-2023/HACMan)) [5]. The reinforcement learning implementation is based  
 31 on Stable-Baselines3 (<https://github.com/DLR-RM/stable-baselines3>). The encoder back-  
 32 bones for both the actor and the critic are PointNet++ segmentation-style from PyG ([https://](https://github.com/pytorch/pytorch)

---

**Algorithm 3** Diff-HySAC

---

```
1: Initialize policy  $\pi_\theta$  and critic networks  $Q_{\phi_1}$  and  $Q_{\phi_2}$ 
2: Initialize the target networks:  $Q_{\phi'_1}$  and  $Q_{\phi'_2}$ 
3: Initialize replay buffer:  $\mathcal{D} = \emptyset$ 
4: while not converge do
5:   Forward the encoder to compute features  $f = f(s_t)$ 
6:   Sample action map  $\mathbf{a}_t^m$  from diffusion policy as in Eq. 3, e.g. DDPM sampling
7:   Compute Q-value map  $Q = Q_\phi(f, \mathbf{a}_t^m)$ 
8:   Select contact point  $x_i$  using location policy Eq. 4
9:   Select corresponding action's motion parameter  $\mathbf{a}_{t,i}^m$ 
10:  Execute action  $(x_i, \mathbf{a}_{t,i}^{m,0})$ , observe  $r_t$  and  $\mathbf{s}_{t+1}$ 
11:  Add sample  $\{s_t, (x_i, \mathbf{a}_{t,i}^{m,0}), r_t, \mathbf{s}_{t+1}\}$  to replay buffer  $\mathcal{D}$ 
12:  Sample a minibatch  $\{s, (x_i, \mathbf{a}_{t,i}^0), r_t, \mathbf{s}'\}$  from  $\mathcal{D}$ 
13:  Update the critic as in Eq. 2 with target  $y_t$  defined in Section 4.3.
14:  Update the actor with loss  $J_\pi(\theta)$  as defined in Section 4.3
15:  Adjust temperature  $\alpha$ 
16:  Update the target networks, i.e. using standard delayed updates like in SAC.
17: end while
18: return final policy  $\pi_\theta$ .
```

---

33 [//pytorch-geometric.readthedocs.io](https://pytorch-geometric.readthedocs.io)). The network size and learning rate are identical for  
34 both the actor and the critic.

35 HybridDiff-TD3 (Ours): The diffusion model implementation is based on Diffusion Poli-  
36 cies for Offline RL method, Diffusion-Q Learning ([https://github.com/Zhendong-Wang/](https://github.com/Zhendong-Wang/Diffusion-Policies-for-Offline-RL)  
37 [Diffusion-Policies-for-Offline-RL](https://github.com/Zhendong-Wang/Diffusion-Policies-for-Offline-RL)).

38 HybridCon-TD3 (Ours): The consistency model implementation is based on Consistency Models  
39 for RL ([https://github.com/quantumiracle/Consistency\\_Model\\_For\\_Reinforcement\\_](https://github.com/quantumiracle/Consistency_Model_For_Reinforcement_Learning)  
40 [Learning](https://github.com/quantumiracle/Consistency_Model_For_Reinforcement_Learning)).

41 Diff-HySAC (Ours) and Con-HySAC (Ours): The implementations are based on HySAC and their  
42 corresponding HybridDiff-TD3 and HybridCon-TD3 models.

43 HACMan (Baseline): We use the original implementation from the authors ([https://github.](https://github.com/HACMan-2023/HACMan)  
44 [com/HACMan-2023/HACMan](https://github.com/HACMan-2023/HACMan)).

45 The hyperparameters these algorithms are included in Table 1. We manually fine-tune the location  
46 policy temperature  $\beta$  and select the best setting for each algorithm.

Table 1: Hyperparameters.

Hyperparameters	Values
Initial timesteps	2000
Batch size	64
Discount factor ( $\gamma$ )	0.99
Critic update frequency per env step	2
Actor update frequency per env step	0.5
Target update frequency per env step	0.5
Learning rate	0.0001
Noise scheduler	Cosine
Diffusion steps	5
MLP size	[128, 128, 128]
Location policy temperature ( $\beta$ )	0.1(TD3) 0.01(SAC)

## 47 C Simulation and Real Robot Settings

### 48 C.1 Simulation Setting

49 We base our implementation on HACMan. In particular, the observation space comprises a point  
50 cloud of the entire scene  $\mathcal{X}$ , consisting of background points  $\mathcal{X}^b$  and object points  $\mathcal{X}^{obj}$ . Af-  
51 ter each action, the gripper is moved to a reset position, ensuring it is not visible in the point  
52 cloud. Three cameras surrounding the bin capture depth data, which is transformed into point lo-  
53 cations in the robot’s base frame and then merged. The object points are downsampled to a voxel  
54 size of 0.005 m, while the background points are downsampled to 0.02 m. Normals of the ob-  
55 ject points are estimated using Open3D ([http://www.open3d.org/docs/0.7.0/python\\_api/](http://www.open3d.org/docs/0.7.0/python_api/open3d.geometry.estimate_normals.html)  
56 [open3d.geometry.estimate\\_normals.html](http://www.open3d.org/docs/0.7.0/python_api/open3d.geometry.estimate_normals.html)), and these estimated normals are utilized during  
57 execution.

58 Each point’s features include its XYZ coordinates, the goal flow, and a segmentation mask (ma-  
59 nipulated object vs background). The goal flow for object points is determined based on the goal  
60 pose, whereas background points have a goal flow of zero. Segmentation labels are obtained from  
61 Robosuite during simulation. The goal flow indicates the displacement from each object point in the  
62 current point cloud to its corresponding point in the goal point cloud. This 3D vector is concatenated  
63 with other features of the input point cloud.

64 Action Representation: As detailed in HACMan, the object-centric action space is composed of two  
65 parts: a contact location  $a^{loc}$  on an object and motion parameters  $a^m$ , which define the gripper’s  
66 delta movements after contact. When executing an action, the end-effector first moves to a free-  
67 space location near  $a^{loc}$ , then interacts with the object according to the motion parameters  $a^m$ .

68 Goal flow: same as described in HACMan, the goal flow is calculated by subtracting each corre-  
69 sponding current point from their transformed goal point cloud.

### 70 Benchmarking HACMan tasks

71 *All Training Objects:* The task, as described in HACMan [5], involves a dataset of 32 objects with  
72 initial and goal poses randomly selected from a list of 100 stable goal poses for each object. These  
73 poses are sample from an SE(3) object pose above the bin, and wait until it becomes stable and then  
74 recorded. An episode is considered successful if the average distance between the corresponding  
75 points of the object and the goal is less than 3 cm.

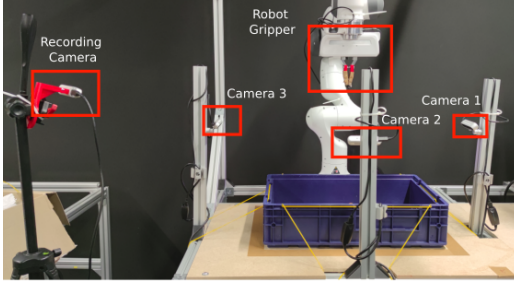
76 *Cube:* This task, also described in HACMan [5], focuses on a single cube object. The initial and  
77 goal poses of the cube are randomly sampled from the stable goal poses.

### 78 C.2 Real Robot Setting

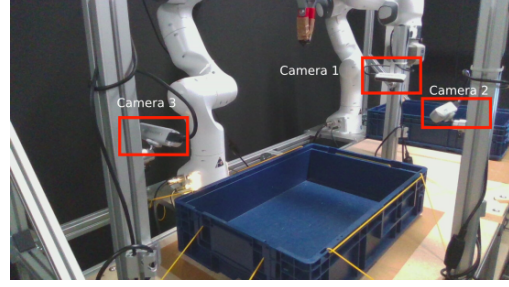
79 The real robot setup is shown in Fig 1a. The setup contains three Realsense cameras on different  
80 perspectives, a Franka Robot arm with parallel gripper (in closed state). In addition, we set up a  
81 recording camera for experiment video, the view from recording camera can be view at Fig 1b. The  
82 point clouds recorded from three cameras are registered to get a full point cloud of the scene using  
83 Open3D library [6]. We segment the object point cloud from the full point cloud using the known  
84 location and dimensions of the bin.

85 For the goals in the real world evaluation, we record 10 goal point clouds for each object by manually  
86 random setting the objects into different poses. During each time step, we use pointcloud registration  
87 algorithm to estimate the goal transformation to calculate the goal flow. Specifically, we use the  
88 global registration implementation from Open3D and then use Iterative Closest Point (ICP). The  
89 evaluation process is done automatically, the reward and the episode termination condition (less  
90 than 3cm) are both calculated automatically.

91 For the real robot experiments, we use the trained policy for ”All Object” dataset with the corre-  
92 sponding ”6D” and ”non-planar” tasks. We perform zero-shot sim2real transfer evaluation without  
93 finetuning or additional data collection steps.



(a) Real Robot Setup - Main View



(b) Real Robot Setup - Recording Camera View

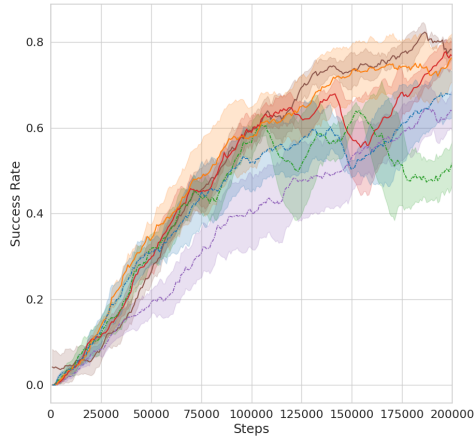
## 94 **D Additional Results**

### 95 **D.1 Simulation Results**

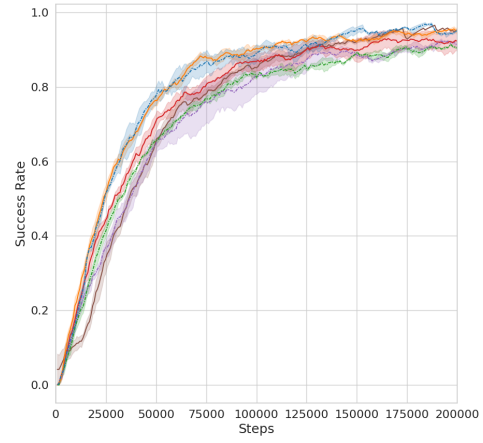
96 In this section, we include the evaluation results for all methods with translation and 6D tasks for  
 97 all Objects and Cube. Fig. 2 include the evaluation success rate for all methods, over 200k training  
 98 steps. As discussed in Section 5, the performance of Diff-HySAC and Con-HySAC is comparable  
 99 to or better than baselines HACMan and HySAC on simple tasks (on Train objects and Cube).

### 100 **D.2 Ablation**

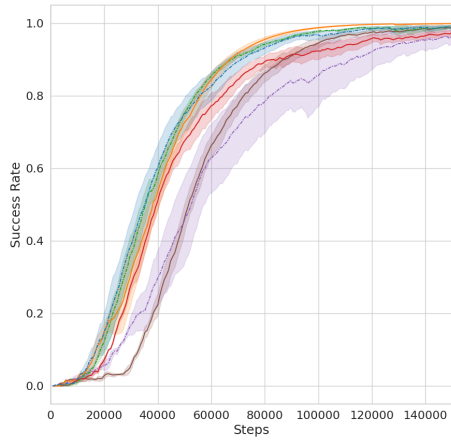
101 In this section, we conduct ablations where we trained with different parameters on a hammer ob-  
 102 ject from the train dataset. Fig. 3 report results that evaluate different diffusion steps setting for  
 103 HybirdDiff-TD3, Con-HySAC, and HybridCon-TD3. The results showed that a diffusion step of 5  
 104 give the best performance (Note that the diffusion steps of consistency models is  $N$  in Algorithm 1).



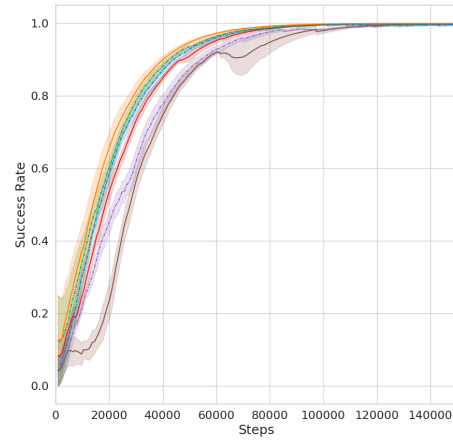
(a) 6D Task All Objects



(b) Translation Task All Objects



(c) 6D Task Cube



(d) Translation Cube

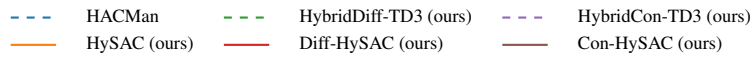
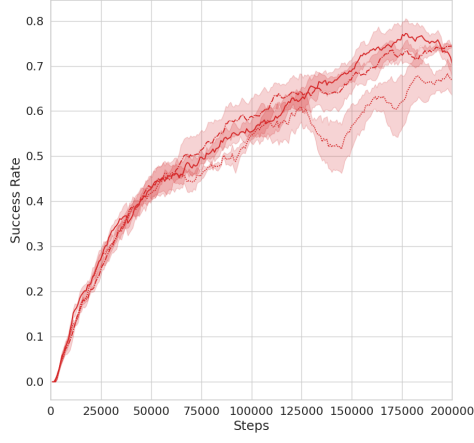
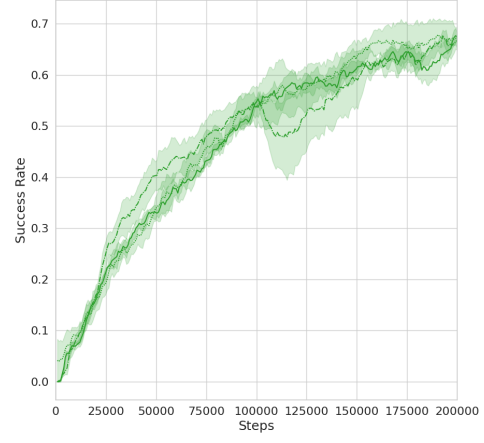


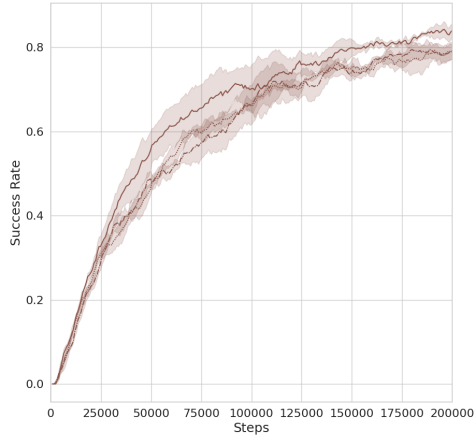
Figure 2: Evaluation of success rate on 6D and Translation taskf for All Objects (200k training steps) and Cube (150k training steps).



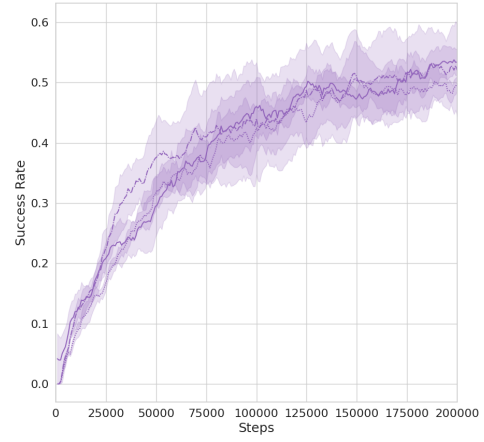
(a) Diff-HySAC



(b) HybridDiff-TD3



(c) Con-HySAC



(d) HybridCon-TD3

..... Diffusion Steps - 2    ——— Diffusion Steps - 5    - - - Diffusion Steps - 10

Figure 3: Ablation on different diffusion steps of the Cosine scheduler.

## References

- [1] S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.
- [2] Y. Song and P. Dhariwal. Improved techniques for training consistency models. In *The Twelfth International Conference on Learning Representations*, 2024.
- [3] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.
- [4] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.
- [5] W. Zhou, B. Jiang, F. Yang, C. Paxton, and D. Held. HACMan: Learning hybrid actor-critic maps for 6d non-prehensile manipulation. In J. Tan, M. Toussaint, and K. Darvish, editors, *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229 of *Proceedings of Machine Learning Research*, pages 241–265. PMLR, 2023. URL <https://proceedings.mlr.press/v229/zhou23a.html>.
- [6] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.