Appendices for *Towards Disturbance-Free Visual Mobile Manipulation*

A. Environment Details

Our experimented environment directly follows the visual mobile manipulation task, ArmPointNav, in the ManipulaTHOR framework [16]. The details of the task and environment can be seen in the ManipulaTHOR paper. Here, we give a high-level overview of this task.

ArmPointNav has a dataset called APND that stores the configurations of each episode. APND consists of 29 kitchen scenes in AI2-THOR [47] that have more than 150 object categories. Among the scenes, 19 of them are used for training, 5 of them for validation, and 5 for testing. There are 12 pickupable categories, and 6 of them (Apple, Bread, Tomato, Lettuce, Pot, Mug) are used for training (*i.e.*, seen objects), and the others (Potato, SoapBottle, Pan, Egg, Spatula, Cup) are used for testing and validation (*i.e.*, novel objects).

Each episode has a configuration that specifies the initial and target positions of the target object and the initial position of the agent. The goal for the agent is to first navigate towards the target object, pick it up with a magnet, and then navigate towards the target location to release the object.

Mathematically, ArmPointNav can be formulated as a POMDP $(\mathcal{S}, \mathcal{O}, \mathcal{A}, R, T, \gamma, P, O)$. Each state $s \in \mathcal{S}$ includes the 3D positions of the robot, the goal, and all the obstacles in the room. Each observation $o \in \mathcal{O}$ includes a depth map (224×224 single-channel image) and a distance coordinate to the goal (3 dimensions). The goal switches from the initial position of the target object to the desired position of the target object once the agent picks it up. The action space ($\mathcal{A}_{\text{large}}$) has 21 actions including: (1) navigation (move ahead, rotate), (2) manipulation (move the arm and gripper), (3) camera rotation, and (4) pick up and done. The details of the action space can be seen in Fig. 8. The reward function R is defined in Eq. 1 and Eq. 2. The time horizon $T = 200$ steps, and the discount factor $\gamma = 0.99$. The transition $P(s_{t+1} \mid s_t, a_t)$ determines how the robot and obstacles move in 3D coordinates, and the emission $O(o_t \mid s_t)$ determines the rendering of egocentric vision to the robot.

The task is a POMDP rather than an MDP because the robot cannot observe the ground-truth positions of the obstacles nearby, which are crucial for optimal control with a Markovian policy. It can only use the historic information of egocentric depth maps to infer them.

B. Experiment Details

We train our agents using the AllenAct framework [93]. All the experiments including baselines and compared self-supervised auxiliary tasks share most training hyperparameters. Each experiment uses 19 processes (each sampling rollouts on one training scene) and trains for 45M frames (for two-stage curriculum: pre-training for 20M frames, and then fine-tuning for 25M frames).

We use the DD-PPO algorithm [96] with default configuration. The model architecture (Fig. 2) uses a modified ResNet18 with group normalization [99] following DD-PPO as the visual encoder, an embedding layer into 32-dim for goal coordinates, an embedding layer into 16-dim for previous actions, then a GRU with a hidden size of 512, and finally linear actor and critic heads.

For the self-supervised auxiliary tasks CPC|A and Inverse Dynamics, we directly follow the implementation of Ye *et al.* [104][6].

For our disturbance prediction auxiliary task, we use a 2-layer MLP of hidden size 128 to predict the disturbance distance signals $\in [0, 1]^{|\mathcal{A}_{\text{large}}|}$, for all the actions $\in \mathcal{A}_{\text{large}}$ (similarly to Deep Q-Network [63]), given the current belief $\in \mathbb{R}^{512}$. The auxiliary task uses the Focal loss [56] with $\gamma = 2.0$ and $\alpha = 0.5$. The overall objective is a weighted sum of the RL loss and the auxiliary task loss, with a fixed weight of 0.1 on the auxiliary task, following Ye *et al.*

C. PPO-Lagrangian Details

PPO-Lagrangian is a common baseline from the safe RL literature [76] which aims to solve constrained MDPs [3]. In our paper, the original objective is the ArmPointNav task with original reward,

$$J(\pi) := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=1}^{T} \gamma^t r_t \right] , \tag{5}$$

and the constraint is on the total disturbance distance, which we want to be non-positive:

$$J_C(\pi) := \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=1}^{T} \gamma^t (d_t^{\text{objects}} - d_{t-1}^{\text{objects}}) \right] \leq 0 . \tag{6}$$

The corresponding Lagrangian is:

$$\min_{\lambda \geq 0} \max_{\pi} J(\pi) - \lambda J_C(\pi) , \tag{7}$$

where λ is the Lagrangian multiplier.

The Lagrangian method alternatively update the policy π and the Lagrangian multiplier λ:

- Given current Lagrangian multiplier λ_k and the learning rate η_π, the policy π_k is updated by

$$\pi_{k+1} \leftarrow \pi_k + \eta_\pi \nabla_\pi (J(\pi_k) - \lambda_k J_C(\pi_k)) . \tag{8}$$

[6] https://github.com/joel99/habitat-pointnav-aux

- Given the current policy π_{k+1} and the learning rate η_λ, the Lagrangian multiplier λ_k is updated by

$$\lambda_{k+1} \leftarrow (\lambda_k + \eta_\lambda J_C(\pi_{k+1}))_+ \,. \tag{9}$$

The initial value of Lagrangian multiplier λ_0 is set by the user, and is shown to be crucial to performance [1].

Applying the Lagrangian method to the PPO algorithm, we obtain the PPO-Lagrangian method. We use the same hyperparameters in (DD-)PPO as that in our method.

D. Additional Results

D.1. Reliable Evaluation Plots

We follow the rliable library[7] to evaluate our method and baselines. Fig. 6 shows the mean and IQM of SR and SRwoD metrics (reported in Table 3), and also their 95% confidence intervals (CIs).

We find that fine-tuning stage (stage II) is much more robust to seeds with narrower CIs, than trained-from-scratch stage (stage I), and also PPO-Lagrangian. This suggests the robustness and reliability of our method.

D.2. Main Results on Testing Scenes with Seen Objects

Table 4 shows the main results on testing scenes with seen objects (recall that Table 3 is on testing scenes with novel objects). Generally speaking, the trend in Table 3 holds in Table 4:

- Auxiliary tasks can improve sample efficiency (Block 1 and 2).
- Training from scratch learns to stop early with poor success rate (Block 2 and 3).
- Two-stage training achieves higher SRwoD without sacrificing SR (Blocks 2, 3, and 5).
- Two-stage training outperforms the safe RL baseline (Block 4 and 5).

Comparing Table 4 to Table 3, we find that the SR in Block 1, 3, and 5 increases by $\approx$ 1-2% for all the auxiliary tasks, and SRwoD in Block 5 increases by $\approx$ 2%. This is understandable because Table 4 are evaluated on seen objects.

D.3. Effect of Disturbance Penalty Coefficient

The disturbance-free objective in Eq. 2 is sensitive to the disturbance penalty coefficient λ_{disturb}. Ideally, the coefficient should be large enough to enforce a hard constraint on disturbance. But a too-large coefficient may hinder the agent from reaching the goal with a very small disturbance distance, thus affecting the success rate. To balance between SR and SRwoD, one has to tune the coefficient on a validation set.

Fig. 7 shows the performance of the fine-tuned models with disturbance prediction auxiliary tasks (Block 5), with different penalty coefficients λ_{disturb}. SRwoD monotonically increases from 79.5% to 82.5% with a larger disturbance penalty coefficient, which shows the effectiveness of our method. We finally choose $\lambda_{\text{disturb}} = 15.0$ because it balances SR and SRwoD best. Surprisingly, disturbance avoidance ($\lambda_{\text{disturb}} = 15.0$) can even help success rate, compared to the model with original reward (*i.e.*, $\lambda_{\text{disturb}} = 0.0$). Note that these experiments are only ablations over the validation set and we only calculate the final performance of our model on the test set when using $\lambda_{\text{disturb}} = 15.0$.

D.4. Is the *Large* Action Space Necessary for Disturbance Avoidance?

As described at the beginning of Sec. 4.2, we found, qualitatively, that the original action space hinders the ability of agents to perform disturbance-free tasks. *E.g.*, the agent may not even be able to see the disturbance it causes, as it cannot look down. Table 5 shows our ablation study on the action space on the *validation* set. Interestingly, under the original objective (Block 1; reward r) in ArmPointNav and without auxiliary tasks, the performance of our baseline drops when switching from the small action space to a large one (Rows 1-2 and 5-6). But with our disturbance prediction task (Row 3 and 7), the performance increases by 2.4%.

However, the large action space helps with achieving a disturbance-free agent. Our best model (Block 5) performs better with the *large* action space when trained with the new disturbance-free objective (Row 4 and 8).

Moreover, by qualitatively examining the actions taken by well-trained agents (Block 5), we find that, in the large action space setting, agents almost always take the $\texttt{LookDown} \in \mathcal{A}_{\text{large}} \setminus \mathcal{A}_{\text{small}}$ as their first action allowing them a better view of the impact of their actions (see App. D.5). Thus, quantitatively and qualitatively, we find that the added actions indeed help avoid disturbance.

D.5. Agent Action Distribution

To better understand how the agent behaves across time, we plot the heatmap of the action distribution of our best agent (the last row of Table 3) in Fig. 8. We show the action distribution for each time-step averaged across episodes. This visualization gives us several insights into the agent's behavior. Firstly, `LookDown` is almost always taken at the first time-step to enable the agent to have a better perspective, this justifies the necessity of using the large action space for the disturbance-free objective. Sec-

[7] https://github.com/google-research/rliable

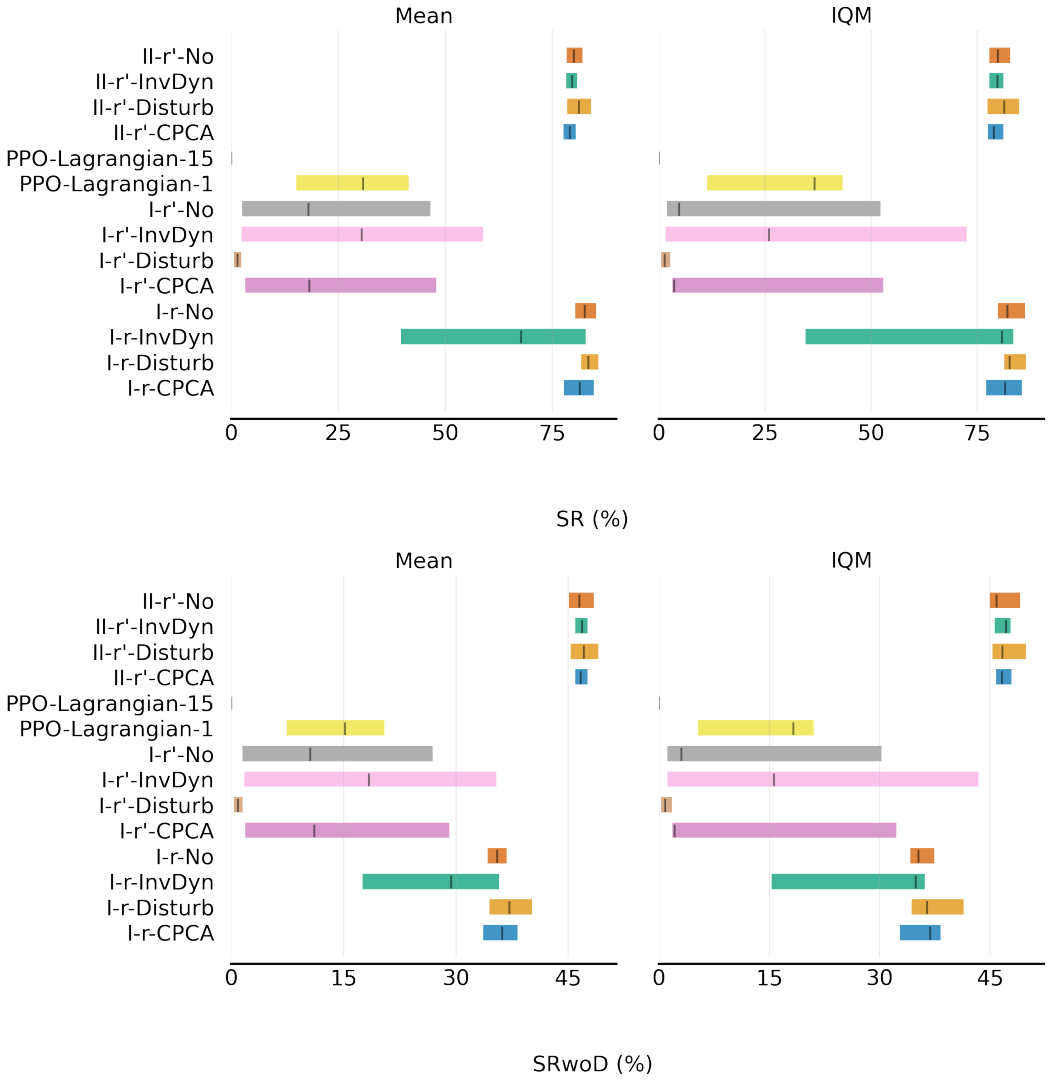

Figure 6. **Mean and IQM with 95% (stratified bootstrap) confidence intervals** on **SR** (top figure) and **SRwoD** (bottom figure), following the rliable library [2]. We denote each method by its stage (I, II), reward (r,r'), and auxiliary task. 1 and 15 in PPO-Langragian stand for the initial value of the multiplier. All methods are trained for 45M frames.

ondly, we can clearly see that there are two phases during an episode. The first phase is to move to pick up the target object (roughly from time-step 0 to 45), where the agent first moves ahead (`MoveAheadContinuous`) and rotates (`RotateLeft/RightContinuous`) to navigate, then the agent moves the arm and gripper downwards (`MoveArmHeightM`, `MoveArmYM`), and finally the agent picks up (`PickUpMidLevel`) the target object. The second phase focuses on taking the object to the target location (roughly from time-step 40 to 100). Similar to the first phase, the agent first moves ahead and rotates to navigate, and then moves the arm and gripper downwards. But the agent also demonstrates delicate arm behavior during the second phase, such as moving the gripper in XY plane (`MoveArmX/Y*`) and rotating the arm wrist (`RotateArmWrist*`), to reduce disturbance to the other objects when placing the target object.

E. Code and Videos

Our code is available at `https://github.com/allenai/disturb-free`. The videos of our method and compared methods can be accessed at `https://sites.google.com/view/disturb-free`.

Stage	Reward	Initial	Frames	Aux Task	SR (%)	SRwoD (%)
I	r	scratch	20M	None (Original)	66.3	32.1
I	r	scratch	20M	None (New)	74.7	32.2
I	r	scratch	20M	CPC|A [25, 104]	76.5	31.6
I	r	scratch	20M	Inv. Dyn. [68, 104]	78.3	34.3
I	r	scratch	20M	Disturb (Ours)	79.6	34.4
I	r	scratch	45M	None (New)	**84.1**	35.8
I	r	scratch	45M	CPC|A	82.4	36.7
I	r	scratch	45M	Inv. Dyn.	69.3	28.7
I	r	scratch	45M	Disturb	84.0	36.7
I	r'	scratch	45M	None (New)	18.4	10.1
I	r'	scratch	45M	CPC|A	17.9	10.4
I	r'	scratch	45M	Inv. Dyn.	31.2	18.1
I	r'	scratch	45M	Disturb	1.4	0.6
PPO-Lagrangian [76] ($\lambda_0 = 1.0$)			45M	None (New)	33.3	15.6
PPO-Lagrangian ($\lambda_0 = 15.0$)			45M	None (New)	0.0	0.0
II	r'	finetune	20M+25M	None (New)	80.8	49.0
II	r'	finetune	20M+25M	CPC|A	79.6	47.5
II	r'	finetune	20M+25M	Inv. Dyn.	80.9	48.8
II	r'	finetune	20M+25M	Disturb	81.7	**49.4**

Table 4. **Main results on testing scenes with seen objects using the large action space $\mathcal{A}_{\text{large}}$.** Each method is labeled by its stage in our curriculum (Fig. 3.4), the reward it received (r for original reward; r' for new reward), the weight initialization (from scratch or fine-tuned), number of training frames, and what auxiliary task it used. For none auxiliary task, "original" refers to the original baseline, and "new" refers to our improved variant. Results are averaged over 5 random seeds.

Row	Action Space	Reward	Initial	Aux Task	SR (%)	SRwoD (%)
1	$\mathcal{A}_{\text{small}}$	r	scratch	None (Original)	55.8	12.3
2	$\mathcal{A}_{\text{small}}$	r	scratch	None (New)	73.7	18.1
3	$\mathcal{A}_{\text{small}}$	r	scratch	Disturb	70.0	16.3
4	$\mathcal{A}_{\text{small}}$	r'	finetune	Disturb	**74.2**	**26.2**
5	$\mathcal{A}_{\text{large}}$	r	scratch	None (Original)	56.4	11.9
6	$\mathcal{A}_{\text{large}}$	r	scratch	None (New)	66.8	16.9
7	$\mathcal{A}_{\text{large}}$	r	scratch	Disturb	72.9	17.0
8	$\mathcal{A}_{\text{large}}$	r'	finetune	Disturb	**78.5**	**29.7**

Table 5. ***Large* action space can increase the performance on disturbance avoidance.** $\mathcal{A}_{\text{small}}$ stands for the original action space, while $\mathcal{A}_{\text{large}}$ represents the augmented action space this paper adopted. The counterparts (Row 1 & 5, 2 & 6, 3 & 7, 4 & 8) are trained with same setting except for the action space. Results are on validation set.

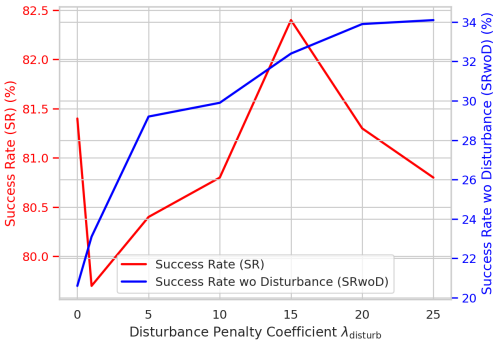

Figure 7. **The effect of disturbance penalty coefficient λ_{disturb} on validation scenes.** The curves show the final results of fine-tuning with disturbance prediction task after 25M steps with different λ_{disturb}.

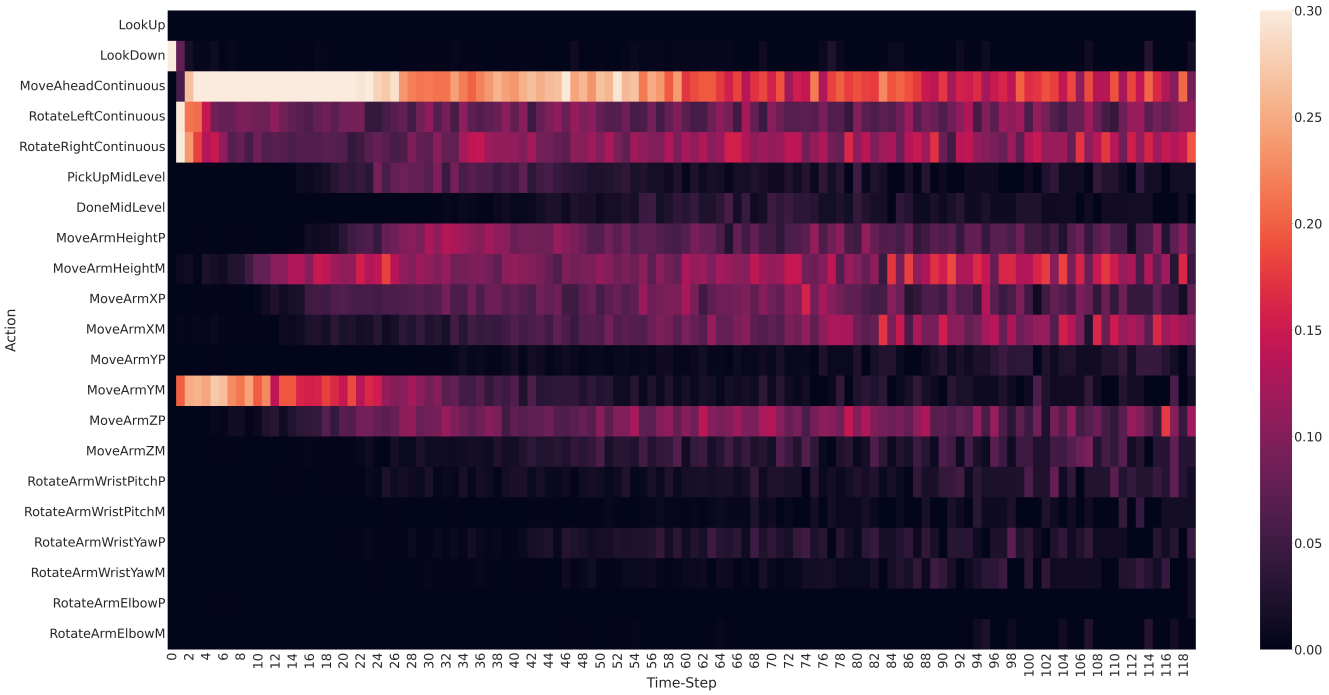

Figure 8. **Heatmap of the action distribution of our best agent over time.** The y-axis lists all possible actions in the large action space ($\mathcal{A}_{\text{large}}$). The x-axis shows the time-step from 0 to 120. The sum of all the cells in each column (time-step) is 1.0. We clip the cell value to 0.3 for better visualization.