Selective Object Rearrangement in Clutter - Supplementary Material

Bingjie Tang
Department of Computer Sciences
University of Southern California United States
bingjiet@usc.edu

Gaurav S. Sukhatme
Department of Computer Sciences
University of Southern California
United States
gaurav@usc.edu

A Clutter Coefficient

Let $\mathbf{P} = \{(x_1, y_1), ..., (x_n, y_n)\}$ denote n objects' positions. We define the clutter coefficient $c(\mathbf{P})$ based on objects' positions $\mathbf{P}$.

$$c(\mathbf{P}) = -\log\{\frac{1}{n}\sum_{i}^{n}\sqrt{(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2}\}, \hat{x_i} = \mathbf{kNN}(\mathbf{y}), \hat{y_i} = \mathbf{kNN}(\mathbf{x})$$

in which $\mathbf{kNN}(\mathbf{y}), \mathbf{kNN}(\mathbf{x})$ estimates $\hat{x_i}, \hat{y_i}$ through k-nearest neighbors regression on every other object's position. $c(\mathbf{P})$ is calculated as the negative logarithm of the mean error for all variance between the ground-truth and the estimated value. When objects are closer to each other (i.e. the scene is more cluttered), error decreases and $c(\mathbf{P})$ increases. We consider arrangements with $c(\mathbf{P}) \geq 1.0$ as 'cluttered'. In Figure 1, we show several exemplary scenes with different clutter coefficients. In algorithm 1, we give a more detailed description about clutter coefficient calculation.

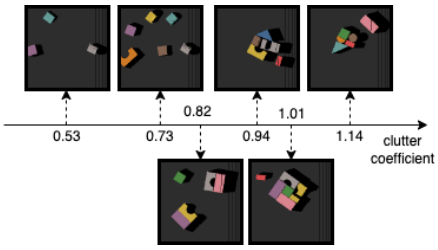


Figure 1: Example scenes with different clutter coefficient value.

Algorithm 1: CLUTTER COEFFICIENT

Input: n objects' positions
$\mathbf{P} = \{(x_1, y_1), ..., (x_n, y_n)\}$.
Output: Clutter coefficient of the scene $c(\mathbf{P})$.

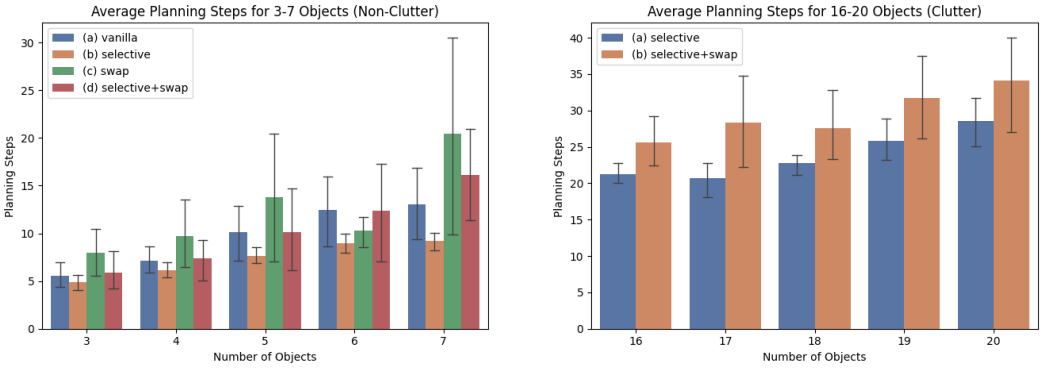
R = KNeighborsRegressor(n_neighbors=$n-1$)
for *every* $p_i \in \mathbf{P}$ **do**
  $\mathbf{P}_i = \mathbf{P} - \{p_i\}$
  Fit the KNN regressor R with $\mathbf{P}_i$
  $\hat{x_i} = R(y_i), \hat{y_i} = R(x_i)$
$c(\mathbf{P}) = -\log\{\frac{1}{n}\sum_{i}^{n}\sqrt{(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2}\}$
return $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

B Further Experimental Results and Ablation Details

B.1 Planning Steps

Planning steps is defined as the average number of actions the robot takes in each completed episode. It is a measure of the planning efficiency of the learned rearrangement policy. Figure 2 shows the average number of planning steps (executed actions) in completed episodes reported in Table 2 of the main paper. In Figure 2, we observe that when the task setting remains the same, the number of planning steps increases as the number of objects increases. When target selection is involved, the number of planning steps decreases, as the object sequencing mechanism prioritizes removing non-target objects from the table, leaves a more sparse arrangement of objects in the workspace, potentially reducing subsequent task difficulty. The introduction of swap actions, however, significantly increases the number of planning steps in each task completion. The swap action requires the robot to sample 'buffer' locations for objects whose goal position is occupied, place objects at 'buffer' locations, remove the 'placeholder' objects at their goal positions and then reposition

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.



(a) 3-7 objects

(b) 16-20 objects

Figure 2: **Average Planning Steps** Tasks with target selection require fewer planning steps, introducing swap actions in the task setting increases planning steps. The number of planning steps increases as the number of objects in the scene grows.

the objects at their goal locations. This process naturally adds more required actions towards task completion.

B.2 Rotation Error

Even though object position errors are good indicators of reposition accuracy, the orientation of objects being placed affects the visual resemblance of the scene to the goal. We model the orientation of object reposition by rotating the input image in the PLACE policy (Figure 3b in the main paper) to 16 different directions and each image represents an end-effector orientation during placement (subsection 3.2 in the main paper). The cross-entropy loss for the PLACE policy accounts for rotation errors for object reposition, hence our system learns to align object orientation according to the goal image. Rotation error is defined as the planar rotation difference between the goal object orientation and the final achieved object orientation after placement. It measures the accuracy of object rearrangement in terms of object re-orientation. The rotation error varies significantly between different object models, as shown in Figure 3. Since our placement policy is based on visual correspondence, objects with more complex shapes/textures (e.g., Tea box, Jello, and tomato soup can) are reoriented more accurately by our system during placement. However, for object with simpler shapes/textures (e.g., apple, orange), our system struggles to differentiate the rotation for such objects due to the visual similarity across different orientations. For objects with more complex shapes/textures (teabox, jello, tomato soup can, mug, mustard bottle, banana, clamp, strawberry, peach), our system achieves an average of 13.89 degrees of rotation error, comparable with 13.70 degrees reported by IFOR.

The object models are listed in Appendix C, and shown in Figure 6. In Figure 4, we show an example of our system successfully re-orienting an object (jello box) according to the goal image.

B.3 Ablation Studies: Target Selection

Num. Obj.	Match Success (%) ↑		Position Error ($10^{-2}m$) ↓		Classification Acc.(%) ↑	
	ResNet	U-Net	ResNet	U-Net	ResNet	U-Net
1	**93.33** ±2.67	32.67 ±2.33	**1.29** ±0.59	18.59 ±3.06	100.00	100.00
5	**95.66** ±1.67	66.00 ±5.00	**1.53** ±0.20	12.03 ±1.64	98.58 ±0.17	**99.25** ±0.50
10	**93.67** ±5.33	77.00 ±5.00	**2.04** ±1.45	5.64 ±1.43	99.18 ±0.52	**99.26** ±0.29
20	**95.00** ±1.00	82.00 ±8.00	**1.50** ±0.24	3.65 ±1.15	99.28 ±0.09	**99.30** ±0.35

Table 1: **Visual Feature Extraction Model Comparisons.** We select a random subset of objects as targets in each rearrangement task.

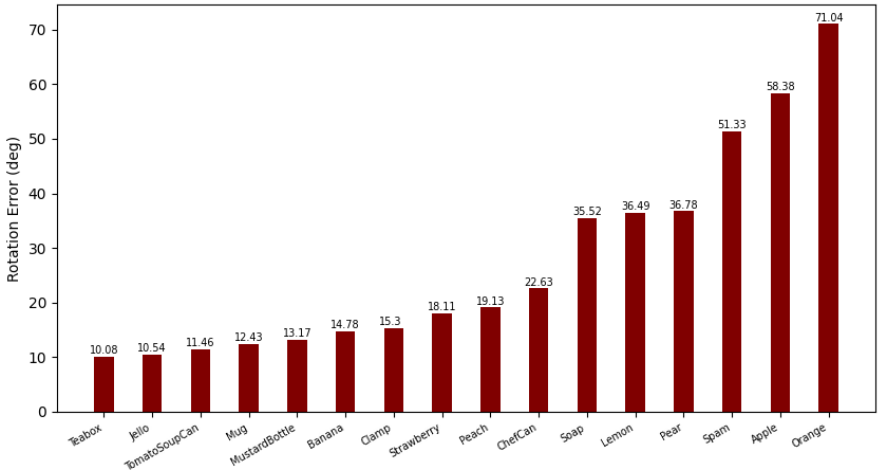
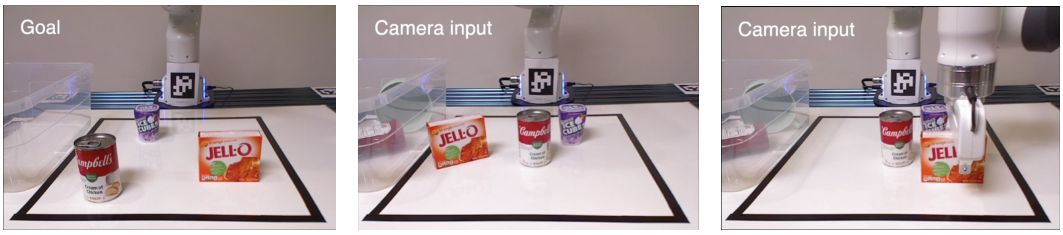


Figure 3: **Rotation Error** for every object model included in the simulated evaluation. Numbers reported are average value over 10 randomly generated episodes.



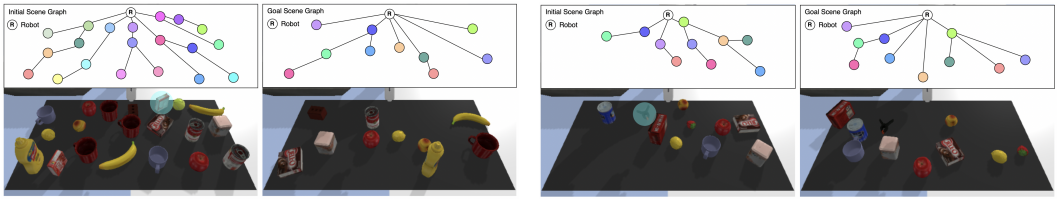
(a) Goal Object Arrangement (b) Before relocating JELLO (c) After relocating JELLO

Figure 4: **Object Re-orientation.** The robot is re-orientating JELLO to match the goal image.

We evaluate the significance of using ResNet to obtain a accurate visual feature cross-correlation and target classification by testing 2 different encoder-decoder structured visual feature extractors, ResNet and U-Net. ResNet is an encoder-decoder model with a 43-layer residual network with 12 residual blocks and 8 stride (3 2 stride convolutions in the encoder and 3 bilinear-upsampling layer in the decoder), followed by image-wide softmax. U-Net is an encoder-decoder model used in the first stage of the scene segmentation model UOIS-Net-3D. In U-Net, each 3×3 convolutional layer is followed by a GroupNorm and ReLU. We pass the raw RGB-D images of the current and goal arrangement through each model to get visual feature map of the same size. For every object in the current scene, we crop its local feature and cross-correlate with the goal visual feature map. If the highest cross-correlation value between an object's local visual feature and the goal feature map is higher than 10^{-4} then we consider it is a target object otherwise we consider it is a non-target object. For each target objects, we calculate the distance (position error) from its goal position to its highest cross-correlation location. If the distance is smaller than 0.05m, we consider the match successful. We report the match success rate, average position prediction error and target classification accuracy over 100 different initial and goal arrangements in Table 1. The choice of visual feature extraction model is crucial to our entire system because it directly effects the accuracy of target object identification, placement location and object correspondence. And Table 1 shows ResNet outperforms U-Net ranging from 1 to 20 objects in terms of match success rate and position error, and has similar target classification accuracy that is close to 100%.

B.4 Ablation Studies: Graph-based Object Sequencing

To verify the importance of applying graph-based object sequencing to minimize the number of actions (i.e. planning steps) needed to complete a rearrangement, we test 2 different scene graph



(a) 20 objects, 10 targets (b) 10 objects, 10 targets

Figure 5: **Object Sequencing Example.**

generation methods and observe their impact on average planning steps. We also consider the situation when no sequencing mechanism is used (**no scene graph**) and the robot picks the next object only based on PUSH and GRASP Q-value estimates.

Before generating the scene graph, UOIS-Net-3D takes the raw RGB-D images from the camera and generate object segmentation. We generate the scene graph in two ways:

Scene Graph	10	20
N/A	35.13±3.55	45.22±4.70
Position	19.94±4.93	29.29±3.52
Accessibility	**15.61**±3.84	**25.45**±3.88

Table 2: **Scene Graph Comparisons.** Average planning steps vs. # of objects in the initial scene. All scenarios have 10 target objects.

(1) **Position**: let $v_i = (x_i, y_i), v_i \in \mathcal{V}, i = 1, ..., N$ represents a detected object where x_i, y_i denote the object's location on the tabletop. We sort $\mathcal{V}$ in descending order of x-coordinates and y-coordinates, and let $\mathcal{V}^x$ and $\mathcal{V}^y$ denotes the sorted list respectively. We add edges to $\mathcal{E}^x$ between every pair of node (v_i, v_{i+1}) in $\mathcal{V}^x$ and add edges to $\mathcal{E}^y$ between every pair of node (v_j, v_{j+1}) in $\mathcal{V}^y$. The final scene graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}^x \cup \mathcal{E}^y)$, which captures the basic geographical relationships among objects.

(2) **Accessibility**: detailed description is in the main paper, subsection 3.3 algorithm 1.

We then perform object sequencing by algorithm 2 in the main paper, given the current scene graph $\mathcal{G}_t$ and the goal scene graph $\mathcal{G}_g$. The results are shown in Table 2. Through the accessibility graph, the robot is able to use significantly less planning steps to transform the initial arrangement to the goal configuration. Compared to no scene graph cases, because SimGNN selects the object that causes the most difference between current and goal scene graphs, using scene graph that captures the spatial relationship among objects can reduce the planning steps towards task completion. When there are non-target objects present, SimGNN chooses to remove them from the workspace (Figure 5a), which decreases the clutter coefficient of the scene and benefits future rearrangement. When there are only target objects present, the accessibility graph captures the shortest traversal path from the root vertex (robot end-effector) to any other vertex (i.e. object). When combined with SimGNN, it makes choices based on the trade-off between the closer object closer to the end-effector which has higher reachability (and hence likely to yield a higher rearrangement success rate), and the object that causes more dissimilarity between the current graph and the goal graph (Figure 5b). We also observe that removing 10 non-target objects from the initial arrangement cost approximately 10 actions which aligns with the target classification accuracy shown in Table 1.

B.5 Ablation Studies: Swap Only

We conduct 6 sets of simulation experiments (each with 3 random seeds and 100 episodes), in each episode, we generate random synthetic scene (with random selected object models, random object positions and orientations), each scene contains a number of objects ranging from three to eight. The goal arrangement is given by randomly swapping the objects' placements in the initial scene, so that each object's goal location is blocked by a random other object. The goal specification is given by a single image. This experimental setting is consistent with previous work NeRP [24]. Results are shown in Table 3, statistics on NeRP is reported in their original paper (codebase and data not publicly released). Our system handles both translation and rotation while NeRP is restricted to translation in object repositioning, which is demonstrated qualitatively in [25]. Despite this handicap, our system produces swap only results that are comparable to NeRP.

Num. Obj.	Task Completion (%) ↑		Planning Steps ↓		Position Error ($10^{-2}m$) ↓	
	NeRP	Ours	NeRP	Ours	NeRP	Ours
3	98.25 ±0.57	94.67 ±2.67	4.58 ±0.82	7.63 ±1.15	0.039 ±0.036	1.31 ±0.43
4	97.60 ±1.20	94.33 ±1.33	5.70 ±1.38	10.10 ±1.63	0.027 ±0.025	1.02 ±0.29
5	94.56 ±0.73	90.67 ±1.33	7.01 ±2.10	14.72 ±1.34	0.019 ±0.013	1.17 ±0.41
6	98.09 ±0.40	85.33 ±3.33	8.69 ±2.15	10.34 ±1.54	0.013 ±0.013	1.50 ±0.31
7	90.62 ±1.03	84.67 ±2.33	9.47 ±2.23	20.02 ±2.17	0.011 ±0.008	1.18 ±0.33
8	87.50 ±2.50	82.67 ±2.67	10.72 ±2.08	22.46 ±2.85	0.010 ±0.008	1.74 ±0.46

Table 3: **Swap Only Scenarios.** with a number of objects ranging from three to eight.

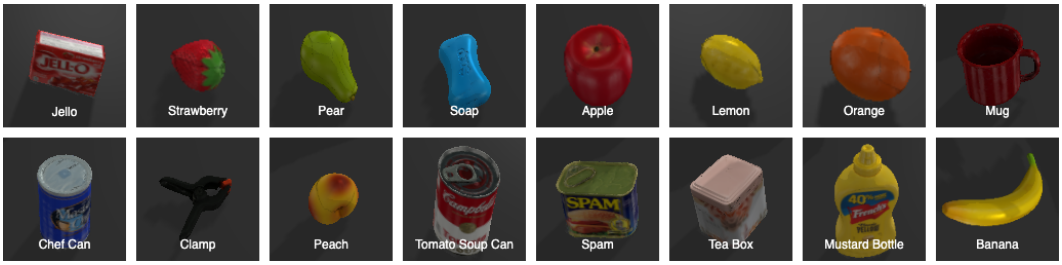


Figure 6: **Object Models** used for simulated evaluation.

B.6 Ablation Studies: Clutter Only

We tested 43 non-selective 10-object scenarios where all objects in the initial cluttered scene needed to be rearranged in the final scene (no binning) in order to demonstrate our system's capability of dealing with initially cluttered arrangements. In these 43 scenarios, all initial object arrangements are with clutter coefficients $> 1.0(1.03 \pm 0.02)$, which are considered as clutter according to the definition of clutter coefficient in Appendix A. Our system succeeds in 33 trials with 27.73 ± 4.27 planning steps and position error of 1.22 ± 0.17 cm, which shows even though removing irrelevant objects from the workspace does decrease the clutter coefficient of the scene, our system does not rely on removing objects to solve the clutter problem.

C Data

To show our method can generalize to novel object instances that are unseen during simulated training, we use a set of 10 everyday objects (jello, strawberry, pear, soap, apple, lemon, orange, mug, chef can, clamp) during training, and we add another set of 6 everyday objects (peach, tomato soup can, spam, tea box, mustard bottle, banana) during testing. Object models used in simulation are imported from `https://github.com/eleramp/pybullet-object-models` and `https://github.com/ChenEating716/pybullet-URDF-models`, object models used are shown in Figure 6. In real robot demonstrations, we used the objects shown in Figure 7. While the majority of objects used in the real robot demonstrations were never seen in simulation, some of them were (e.g., the simulation has models of spam, jello and the soup can).

Figure 7: **Objects** used for real robot demonstration.