Appendix: Learning to Regrasp by Learning to Place

Shuo Cheng[1], **Kaichun Mo**[2], **Lin Shao**[2]

[1]University of California, San Diego — scheng@eng.ucsd.edu
[2]Stanford University — {kaichunm, lins2}@stanford.edu

Overview

In this appendix, we evaluate whether the multi-task learning for contact point predictions is useful or not and how the score threshold affects the performance of stable object placement. Then we report how we generate and annotate the dataset used in this work. We describe the regrasping experiments in the simulation and the real world. We further investigate whether our proposed pipeline is robust to various sources of noises (e.g., object segmentation, depth camera noise, object dynamics). Lastly, we conduct failure cases analysis and report the computational costs.

1 Additional Experiments on Object Stable Placement

1.1 Does Multi-task Learning Help?

Task Combination	Contact Acc.	Contact Prec.	Stable Acc.	Stable Prec.
stable	N/A	N/A	0.888	0.890
stable + contact	0.925	0.977	**0.918**	0.906
stable + contact + offset (ours)	**0.945**	**0.981**	0.912	**0.910**

Table 1: We report the performance of our model trained with different loss variants. Our multi-task joint learning strategy leverages the correlation between tasks to enhance the feature learning, which results in better generalization.

To verify the usefulness of the multi-task joint learning framework, we adopt the same network architecture but with different combinations of task losses. The quantitative results on the test set are shown in Table 1. We use the accuracy and precision metrics to measure the performance of each model. While the model trained with only stable loss reaches a reasonable performance, adding auxiliary losses improves the stable precision from 0.906 to 0.910. Our experiments demonstrate that the contact prediction and contact offset regression are beneficial to the stable pose learning task. This is reasonable because a strong correlation between each subtask eases the joint learning process: 1) for all stable poses, the placed object must have contact points with the supporting environment, and 2) the offset from the support and the object should be consistent with each other.

1.2 Different Score Threshold

We evaluate how different score threshold will affect the accuracy of the proposed stable placements, the results are presented in Fig 1. While observing that filtering out the generated poses using higher thresholds will generally lead to better accuracy of stable placement, we do not consider a too harsh threshold because it will remove all the poses. We choose the threshold to be 0.8 for best practice.

2 Dataset

We construct a dataset containing 50 objects (i.e., spoon, fork, hammer, wrench, etc.) and 30 supporting items (i.e., mug, box, bowl). We then split these objects and supporting items into training and test sets. We generate 249 pairs from the training set of objects and supporting items, and 38 pairs from the test. For each pair of object-supporting item, the initial placement poses are generated by sampling different object poses with respect to the supporting item in PyBullet, and running a

5th Conference on Robot Learning (CoRL 2021), London, UK.

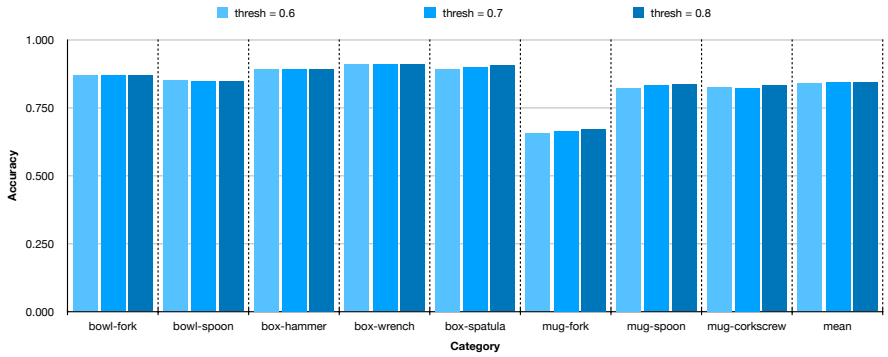

Figure 1: The accuracy of the proposed stable placements under different score thresholds.

forward simulation to see if it remains stable. We collect the placement data for supporting items by randomly placing the supporting items with respect to the initial environment (i.e., the ground), and running the simulation to check whether the supporting items keep static. To make the stable placement robust to variant dynamics and geometry, we randomize the dynamic parameters including friction, mass, and external forces. For each round of simulation, we randomly scale the object mass by [0.9, 1.1], and we randomly select the gravity in the range [$\pm$0.1, $\pm$0.1, -9.81 $\pm$0.1] for approximating the perturbations in the real environment, we also sample the lateral friction, spinning friction, and rolling friction in the range of [0.3, 0.7]. In total, we generate 1,050,884 poses for training items and 15,105 poses for testing items. Some stable object placements from our dataset are visualized in Fig 2.

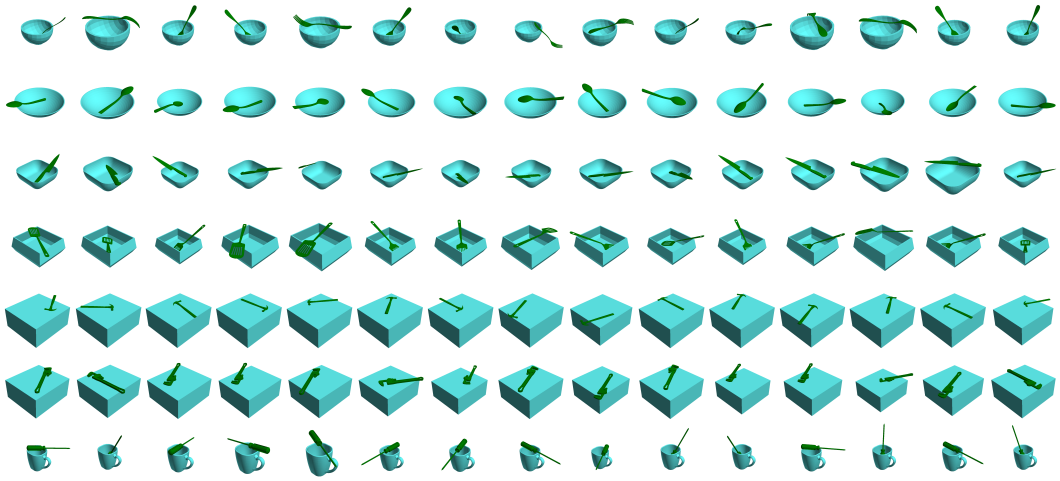

Figure 2: Visualization of random sampled stable placements in our dataset.

3 Regrasping Simulation Experiments

Leveraging the stable pose prediction model and object grasping model, we evaluate the regrasping performance of our proposed system. We randomly select multiple objects from our dataset when creating each initial manipulation scene. These objects are placed stably, except that the robot might grasp one object. In each scene, there is one regrasp query, which is a target grasp pose of one object, for our system to accomplish.

In total, we have 30 scenes as the test set. Our system generates the regrasp graph by predicting the stable poses of each object and checking whether there are edges connecting two nodes. The regrasp task is considered successful if our system finally finds the target regrasp poses. We report the average success rate among the test scenes. We adopt a baseline approach by replacing our object

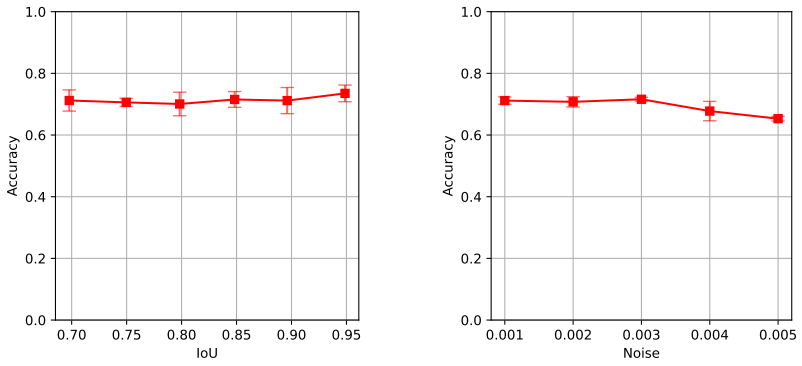

Figure 3: The accuracy of the proposed stable placements under different segmentation IoUs and noise.

placement prediction model with the baseline method (*Random*) and keeping the same rest settings. Our proposed pipeline achieves 83.3% success rate while the baseline does not have any successful case.

4 Regrasping Real World Experiments

We mount a two-fingered customized gripper onto a ur5 shown in the video on the project webpage https://sites.google.com/view/regrasp. A realsense camera is mounted at the robot's wrist and its relative transformation to the end-effector is calibrated. We gather raw point clouds from the realsense camera leveraging its calibrated intrinsic camera parameters. In the current setting, we use the black color cloth to get object segmentation. Other segmentation approaches could also be adopted. These point clouds are fed into our model to predict stable poses. We run position control to move and place the object at its predicted pose. The video currently plays at 4x speed.

5 Generalization and Robustness

Our pipeline assumes the object and support can be segmented from the background using off-the-shelf methods [1, 2, 3]. Considering the noise induced by the perceptual algorithm and real sensor, we also add gaussian noise on point clouds during training to make the models more robust to outliers. For evaluation of the tolerance on segmentation errors, we first test our pipeline under different IoUs using simulation data (the ground-truth segmentation mask is randomly expanded or cropped), the performance is shown in Fig 3. To quantify the robustness of our models to sensor noise, we add Gaussian noise with different standard deviation to the input point clouds. The results suggest that our trained models can resist segmentation noise and sensor noise in a reasonable range.

To mitigate the uncertainties related to the object dynamics, we add various perturbations when annotating the stable poses during the data generation stage. In the real-world execution, we would prefer the stable object placement object that a small neighborhood of the specific stable pose remains stable according to the stable pose classifier. This specific stable pose is helpful during the real-world execution when the noise in robot actuation may lead to alignment errors between the object and the surrounding environments.

6 Failure Cases and More Result Analysis

6.1 Stable Pose Prediction Module

Our model takes raw point clouds of the object and surrounding environment along with random variables as inputs, and outputs a list of stable poses. However, these poses can not be 100% accuracy, the failure cases are shown in Fig. 4. We analyze that some predicted poses might be sensitive to tiny perturbation, or missing parts of the depth scans, these predicted poses might result in slight

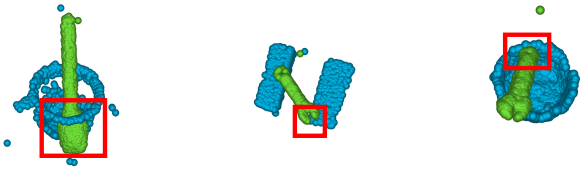

Figure 4: Failure cases visualization.

collisions between the object and the surrounding environments. These failure cases could be mitigated if force/tactile sensors are leveraged to produce closed-loop manipulation policies.

6.2 Entire Regrasping Pipeline

The regrapsing task is quite challenging. A typical regrasping process includes the object placement stage and the object regrasping stage.

During the object placement stage, when the robot is placing the object (for example, a spoon) into the supporting item (for example, a bowl), the final object pose might be stable but different from the predicted desired pose. The 6D pose of the supporting item might also change when receiving external forces in the process. These changes in the object and supporting item's 6D poses might ruin the following regrasping stage. For example, the spoon might fall into the bottom of the bowl, which makes the spoon's regrasping process impossible.

In the object regrasping stage, the robot might fail to find a feasible collision-free approach plan. Given that the robot finds a collision-free approach, the grasping process might still fail due to sensor noise, friction. In the regrasping stage, although the object is already at a stable pose, it might still be sensitive to external perturbations resulting in failure grasps. The most failure cases are caused by the false stable object pose predictions.

7 Computational Costs of the ReGrasping Pipeline

Our system creates and maintains a regrasp graph to tackle the regrasping problem. Each node represents a list of all objects' stable poses in the scene, and each edge indicates that there is one regrasping operation to change from a list of stable poses to another list of stable poses.

In daily tasks such as regrasping spoons and forks, the regrasping process usually happens between two or three objects. In our cases, there are two or three objects in the scene. It takes one or two second to generate 128 predicted stable poses per object and 0.5 to 1 second to predict each object's grasping parameters. It takes 5 to 10 seconds to construct the nodes in our cases and around 30 seconds to generate the edges. Since the regrasp graph in our experiments is relatively small, it takes less than one second to search the graph.

References

[1] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[2] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[3] A. Tremeau and N. Borel. A region growing and merging algorithm to color segmentation. *Pattern recognition*, 30(7):1191–1203, 1997.