NeRP: Neural Rearrangement Planning
for Unknown Objects

Ahmed H. Qureshi[1,2], Arsalan Mousavian[1], Chris Paxton[1], Michael C. Yip[2], and Dieter Fox[1,3]
[1]NVIDIA   [2]University of California San Diego   [3]University of Washington
a1qureshi@ucsd.edu, amousavian@nvidia.com, cpaxton@nvidia.com, yip@ucsd.edu, dieterf@nvidia.com
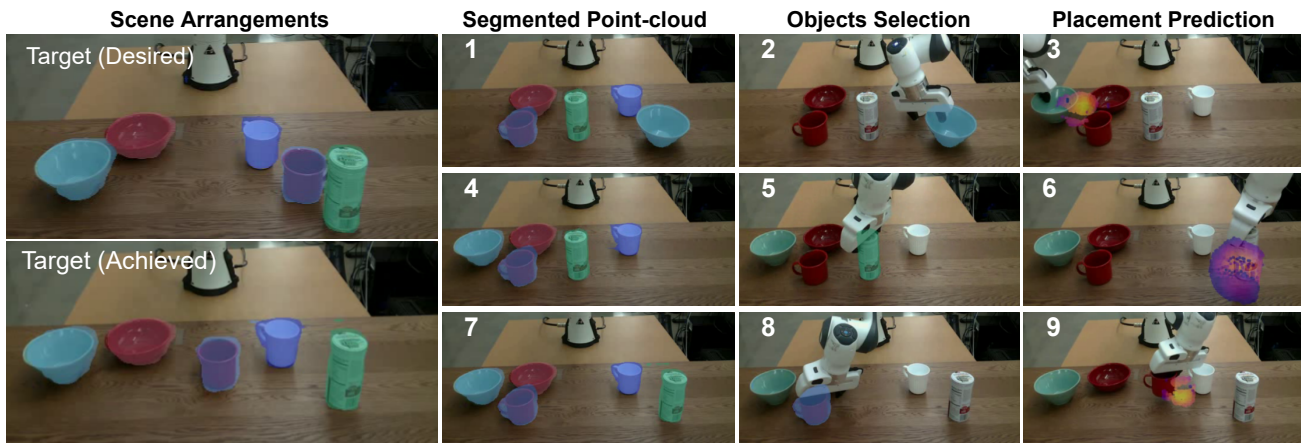
Fig. 1: Neural Rearrangement Planning (NeRP) finds a sequence of pick and place operations to rearrange unknown objects in order to put them in a target arrangement (shown in the top left), choosing both which object to manipulate and where to place it in order to resolve conflicts.

Abstract—**Robots will be expected to manipulate a wide variety of objects in complex and arbitrary ways as they become more widely used in human environments. As such, the rearrangement of objects has been noted to be an important benchmark for AI capabilities in recent years. We propose NeRP (Neural Rearrangement Planning), a deep learning based approach for multi-step neural object rearrangement planning which works with never-before-seen objects, that is trained on simulation data, and generalizes to the real world. We compare NeRP to several naive and model-based baselines, demonstrating that our approach is measurably better and can efficiently arrange unseen objects in fewer steps and with less planning time. Finally, we demonstrate it on several challenging rearrangement problems in the real world[1].**

I. INTRODUCTION

Many real-world robotic tasks boil down to pick-and-place, but with a much wider diversity of objects and scenarios than we typically see in the lab. Robots many also encounter cluttered scenes and blocked goals, cases where typically we might need to perform much longer horizon planning to clear objects and move things out of the way.

Rearrangement of unknown objects has been recently identified as a major challenge problem for embodied AI, especially robotics [1]. While many solutions exist that can solve these problems, particularly throughout the sub-field of task-and-motion planning (TAMP) [7], these often come with a wide range of significant limitations that restrict their utility in the real world. In particular, these approaches often rely on having known object models.

However, recent advances in deep learning have provided ways for us to weaken these assumptions. Work on deep learning for grasping gives us the ability to grasp unknown objects [17, 19], including in cluttered scenes [20, 32]. A growing body of work also looks at using learned representations for robotic task or motion planning [30, 9, 24, 12]. Others have learned samplers for integration into traditional motion planners, e.g. [28, 27]. We can also more accurately segment unknown objects from the world, giving us ways to identify and pick up objects that we have never seen before [34, 35]. Recently, graph neural networks have been used to represent 3D scenes [2].

One question that remains, though, is how we can plan to execute sequences of actions including grasps and placements, in order to perform long-horizon rearrangement tasks in unknown environments. Multiple objects might block one another or prevent things from being moved into new positions. In particular, we need to make decisions about which objects to grasp or move when there are multiple options that are viable at any given time.

In this work, we describe **N**eural **R**earrangement **P**lanning (NeRP), an approach for rearranging unknown objects from perceptual data in the real world, as shown in Fig. 1. NeRP

---

[1]Please refer to our supplementary video: https://youtu.be/CJb1IzH94eo

represents a scene as a graph over segmented objects. Given a current and the goal scene and object segmentation, it finds an alignment between the two sets of unknown objects in the goal and current image using pre-trained ResNet50 [10] features, and uses them to output multi-step rearrangement actions. NeRP is trained using the synthetically generated data in simulation for random object rearrangement.

It uses learned neural networks for choosing the object that needs to be picked and the distribution of possible placements in terms of relative transforms from the current position of the object in the point-cloud space. These operations can be sequenced over time via a sampling-based planning algorithm, which allows us to choose sequences of manipulations in order to perform rearrangement tasks with unknown objects. At execution time, we use model-free grasping [32] to pick and use MPPI controller [4] to place objects and thereby achieve the specified goal. After each placement action, NeRP observes the scene and re-plans the best course of actions to account for inaccuracies in execution. Fig 1 shows execution of object rearrangement on unseen objects with real robot using NeRP.

To our knowledge, this is the first system for end-to-end rearrangement planning for unknown objects. Specifically our contributions are:

- a graph neural network approach for computing which objects to move and estimating where they can be placed to complete a rearrangement task,
- an algorithm for planning and execution that will reactively select where to place these unknown objects in order to complete the rearrangement task.

In addition, we show results both on a simulated object rearrangement task and via real-world robot execution.

II. RELATED WORK

Perhaps the most relevant area of work to rearrangement planning is the field of task and motion planning (TAMP). TAMP is a broad area of study that looks at integrating discrete high-level planning (which objects to grab, which actions to execute) with continuous low level planning (which positions in which to place objects) [7]. This area generally relies on model-based methods, using known objects and fully-observable domains, although work has been done to weaken these assumptions, particularly in recent years. For example, recent work has looked at task and motion planning with partial observability [8], enabling re-arrangement even if certain objects are not fully visible, and others have used learning to guide task and motion planning [14].

Rearrangement planning is a common subset of task and motion planning [3, 15, 21, 16, 7]. It has recently been identified as an interesting challenge area for robotics research [1]. It is interesting in part because it involves long-term planning; recent methods often use Monte Carlo Tree Search or similar methods to explore multiple future possibilities, e.g. [15, 16]. Partial observability is a serious issue, as in many cases objects are partly or wholly occluded [21], which necessitates special considerations [8].

One possible route to building on these is via Motion Planning Networks (MPNet) [28, 27, 26], which predict a set of waypoints that you can use for classical planning based on sensor data. Neural Task Graphs [11] propose a way to perform multi-step planning with known objects, but require a demonstration of the correct action sequence. Another approach is via Deep Affordance Foresight [36], which learns predictive affordance models to allow completion of longer horizon tasks.

Visual Robot Task Planning [24] learns an autoencoder style representation for multi-step task planning, but does not generalize to different goals. Similarly, PlaNET works via deep visual predictions, showing what the possible consequences are for near-horizon actions [9]. Universal Planning Networks use a learned latent space for motion planning together with a simple gradient descent planner [30]. Other work like Q-MDP net for planning under partial observability [13], focusing on navigation instead of placement and rearrangement.

Broadly Exploring Local Policy Trees break up task planning into many different learned high level tasks [12], and use an RRT-like approach to navigate between these via a latent space. However, this approach is not applied to the real world in the same way as our approach is, and as it uses a learned latent space it is most likely less general. Simeonov et al. [29] propose a method that might be most similar to ours, which looks at a whole sequence of manipulation skills to do task and motion planing with unseen rigid objects. However, they assume a high level task skeleton is given. To the best of our knowledge, NeRP is the first approach that solves long-horizon rearrangement planning tasks with unknown objects. Additional possible related work can be Transporter Networks [37] which, however, solves significantly simpler tasks requiring one-step predictions and multiple demonstration of each task. Instead, we do long-horizon multi-step predictions with unknown objects from a single target image without any new training.

III. PROBLEM DEFINITION

Let $X = \{x_1, x_2, \cdots, x_n\} \in \mathcal{X}$ denote an unordered set of n points each of dimension d. Let $\mathcal{M} \in \mathbb{R}^n$ be a point-wise selection operator, i.e., $\mathcal{M} \times \mathcal{X} \mapsto \mathcal{X}'$, over a given set, where $\mathcal{X}' \subset \mathcal{X}$, and $\Delta \in \mathbb{R}^d$ be a set action operator such that $\mathcal{X} \times \Delta \mapsto \mathcal{X}$. Therefore, a set-based planner can be defined as a new class of function $\Pi : \mathcal{X} \mapsto \mathcal{M} \times \Delta$ that determines a sequence of set operators in $\mathcal{M} \times \Delta$ space to point-wise transform a given set to another desired set. In this paper, we propose NeRP, a set-based planner that outputs a sequence of actions $\{(M_0, \delta_0), (M_1, \delta_1), \cdots, (M_H, \delta_H)\} \in \mathcal{M} \times \Delta$ for the initial $X(t)$ and target $X(T)$ unordered sets such that $X(t) \times \boldsymbol{M} \times \boldsymbol{\delta} \mapsto X(T)$, where $\boldsymbol{M} \in \mathcal{M}$ and $\boldsymbol{\delta} \in \Delta$. We demonstrate the application of our method to rearrangement planning problems for robotics.

Let $\mathcal{Q}$ and $\mathcal{A}$ be the robot configuration and action spaces, respectively. A low-level agent can be defined as a function $\pi^L : \mathcal{Q} \mapsto \mathcal{A}$ that achieves the given robot configurations $q \in \mathcal{Q}$ by executing a sequence of actions $\boldsymbol{a}_{\{m\}} =$

$\{a_1, a_2, \cdots, a_m\} \in \mathcal{A}$. In our case, we consider the general robot interaction setting, where for a given current $X(t)$ and target $X(T)$ sets, the high-level agent, $\pi^H \in \Pi$, at time t, outputs a set action $(M(t), \delta(t)) \in \mathcal{M} \times \Delta$, leading to an achievable sub-goal set $q_\delta \subset \mathcal{Q}$ for the low-level agent. The low-level agent π^L executes a sequence of actions $\boldsymbol{a} \in \mathcal{A}$ to achieve $(M(t), \delta(t))$ in the environment and return the control to task planner where it gets new set of observation $X(t+1)$ and replans accordingly.

IV. Neural Rearrangement Planning

This section formally presents our approach for Neural Rearrangement Planning (NeRP), which comprises four main components: the object alignment network, the object selection and objects placement prediction networks, and the collision detection network. Fig. 2 shows an overview of this model architecture. We use these networks to perform multi-step planning in order to perform object rearrangement in unknown scenes.

A. Neural Networks

Objects Alignment and Graph Generation: The objects alignment module determines the individual objects' correspondence in the current $X(t)$ and target $X(T)$ scene point-clouds for the graph generation. We use Unknown Objects Instance Segmentation (UCN) [34] to extract specific object's RGB-information from the given scene and it's corresponding point clouds $X^i \subset X$, where i denotes the selected object's index. The RGB information for each individual objects are passed through pre-trained Resnet model [10] to extract their latent features $\boldsymbol{w} \in W$ for computing L2 norm based objects similarity scores $\boldsymbol{s}$, an $N \times N$ matrix with $N \in \mathbb{N}$ denoting the number of instance segmentations in the current and target scenes.

The graph generator takes the current and target scene point clouds with their segmentation labels and the similarity matrix $\boldsymbol{s}$ as an input and generates an undirected graph $G = (V, E) \in \mathcal{G}$. Each vertex $V^i \in V$ of the graph is computed as follows. Objects in the current observation and target observation are assigned to each other by minimizing the assignment cost using Hungarian method. For instance, in Fig 2, object a_t from the current scenario (at time t) is paired with object a_T from the target scenario (at time T) using their feature-based similarity scores $s(a_t, a_T)$. Once object assignment is computed, the center of observed object's point cloud $p_t \in \mathbb{R}^3$ and $p_T \in \mathbb{R}^3$ are concatenated to represent $V^i \in \mathbb{R}^6$, the features for a graph vertex.

Graph Encoder Network: Our graph encoder network is based on a Higher-order Graph Neural Networks [18], known as k-GNNs, that takes the graph $G = (V, E)$ and hierarchically learns the objects' latent embeddings, $\{\boldsymbol{z}^i; \ \forall i \in [0, N]\}$. We use local k-GNNs, more precisely 1-2-3-GNNs with max-pooling aggregation operator, where each graph layer performs message-passing between individual vertices and local sub-graph structures, denoted as $g \subset G$, along the hierarchy from

---

Algorithm 1: NeRP $\big(X(t), X(T)\big)$

1   $K \leftarrow \text{InstanceSegmentation}(X(t), X(T))$
2   **if** at $t = 0$ **then**
3     $\lfloor \ h \leftarrow 2 \times |K|$ **or** $|K| + 1$ //set planning horizon
4   **else if** $h == 0$ **then**
5     $\lfloor$ **report** out-of-planning-budget
6   $\boldsymbol{s} \leftarrow \text{ObjsAlignment}(X(t), X(T), K)$
7   $G = (V, E) \leftarrow \text{GraphGen}(X(t), X(T), K, \boldsymbol{s})$
8   $\overline{X}(t), \overline{G} \leftarrow X(t), G$ //make a backup copy
9   $rollouts \leftarrow []$
10   **for** 1 to n_rollouts **do**
11     $rollout \leftarrow []$
12     $X(t) \leftarrow \overline{X}(t)$
13     $G \leftarrow \overline{G}$
14   **while** $h > 0$ **do**
15       $\boldsymbol{z}_{\{N\}} \leftarrow f_\Theta(G)$ //graph encodings
16       $\rho_{\{N\}} \leftarrow h_\Phi(\boldsymbol{z}_{\{N\}})$
17       $i \sim \text{Mult}(\rho_{\{N\}})$ //graph node sampling
18       $\delta_{\{B\}} \leftarrow \pi_\Omega(\boldsymbol{z}^i_{\{B\}})$ //set action generation
19       $\hat{X}^i_{\{B\}}(t) = X^i_{\{B\}}(t) + \delta_{\{B\}}$
20       $N' = N \backslash i$ //unselected node indices
21       $\hat{X}^{N'}_{\{B\}}(t) = X^{N'}_{\{B\}}(t)$
22       $\boldsymbol{x}_{\{N' \times B\}} \sim \{(\hat{X}^i_{\{B\}}, \boldsymbol{0}) \cup (\hat{X}^j_{\{B\}}, \boldsymbol{1})\} \ \forall j \in N'$
23       $ids \leftarrow u_\xi(\boldsymbol{x}_{\{N' \times B\}}) > \epsilon$
24       $\delta_{\{B'\}} = \delta_{\{B\}}[ids]$ //collision-free actions
25       $v_{\{B'\}} \leftarrow r_\Psi(\boldsymbol{z}^i_{\{B'\}}, \delta_{\{B'\}})$ //set-action scores
26       $j \leftarrow \text{argmax } v_{\{B'\}}$ //best action
27       $V^i(t) \leftarrow V^i(t) + \delta^j$ //update graph vertex
28       $X^i(t) \leftarrow X^i(t) + \delta^j$ //update point-cloud
29       $e \leftarrow \text{ComputeError}(V)$
30       $\lfloor \ rollout \cup (i, \delta_{\{B'\}}, j, e)$
31     $rollouts \cup rollout$
32   $(i, \delta_{\{B'\}}, j) \leftarrow \text{BestRollout}(rollouts)$
33   $h \leftarrow h - 1$
34   $X^i_{best} \leftarrow X^i(t) + \delta_j$ //best placement
35   $\boldsymbol{p}_{best} \leftarrow \text{ComputeMean}(X^i_{best})$
36   $X^i_{all} \leftarrow X^i_{\{B'\}}(t) + \delta_{\{B'\}}$ //all placement points
37   $X^i_{\mathcal{B}} \leftarrow k\text{-Neighbors}(\boldsymbol{p}_{best}, X^i_{all})$
38   $c_{map} \leftarrow \text{ComputeCost}\big(\boldsymbol{p}_{best}, X^i_{\mathcal{B}}\big)$
39   **return** $X^i_{\mathcal{B}}, c_{map}$

$k = 1$ to $k = 3$, i.e.,

$$f_k^t(g) = \sigma\left(f_k^{t-1}(g) \cdot \Theta_1^t + \max_{g \in \mathcal{N}_L(s)} \big(f_k^{t-1}(g) \cdot \Theta_2^t\big)\right) \quad (1)$$

where N_L denotes the local-neighborhood of subgraph g (for more details, refer to [18]), Θ_1^t and Θ_2^t are the learnable weight parameters, and σ is the component-wise non-linear function ReLU. In our problem setting, k-GNNs with max-pooling operators outperformed vanilla GNNs [33], as the former captures the fine to coarse level structure of the given current-target aligned scene graph which is crucial to have
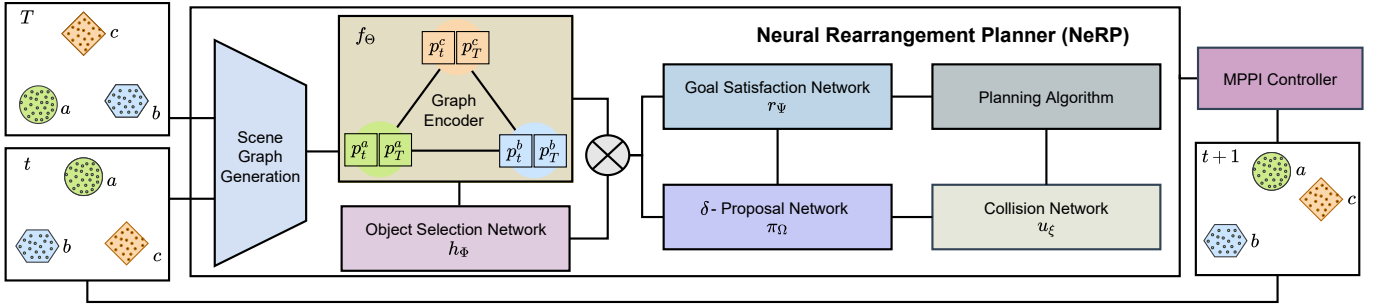
Fig. 2: Model architecture overview. We use UCN [34] to segment out unique objects in the scene, and then compute latent embeddings $\boldsymbol{w}$ for each object alignment in the current and target observation for scene graph generation. Our graph encoder network f_Θ computes the graph embeddings. Then, at each planning step, we use the object selector h_Φ to choose which object to move, and the δ-proposal network π_Ω to generate candidate motions. The goal satisfaction network r_Ψ predicts whether or not individual configurations will satisfy the objective, and the collision detection network rejects particular invalid proposals.

scene-aware node embeddings. Note that our object alignment network captures across the scene correspondence between objects, whereas our graph encoder network captures the overall current and target placement structure (Fig. 2).

Object Selection Network: Our Object Selection Network, $h_\Phi : \mathcal{G} \mapsto \mathcal{M}$, is an object-centric neural model, with parameters Φ, that takes the individual graph node embeddings $\boldsymbol{z}_{\{N\}}$, representing current-target scene graph, and predicts their selection scores $\rho_{\{N\}}$, where $\rho^i \in [0,1]$, i.e.,

$$\rho_{\{N\}} \leftarrow h(\boldsymbol{z}_{\{N\}}; \Phi) \qquad (2)$$

We convert all selection scores $\rho_{\{N\}}$ into probabilities to parameterize a multinomial distribution. A graph node index i is sampled from this distribution during planning which denotes a particular object-pairs embedding in the scene graph. Note that the index i also maps to the instance segmentation label, thus resulting into a subset selection operator $M_i \in \mathcal{M}$. The selected object pair's graph node embedding $\boldsymbol{z}^i$ is then used to predict the next relative transformation for the subset $M_i \times X(t) \mapsto X^i(t)$. Furthermore, we train this module using the binary-cross entropy loss as:

$$l_{\Phi,\Theta} = -\frac{1}{N}\sum_i y^i \cdot \log(h_\Phi(\boldsymbol{z}^i)) + (1-y^i) \cdot \log(1 - h_\Phi(\boldsymbol{z}^i)) \quad (3)$$

where y^i is the binary label from demonstrations indicating a graph node to be selected.

δ-**Proposal Network:** The δ-Proposal Network $\pi_\Omega : \mathcal{X} \mapsto \Delta$, with parameters Ω, is a stochastic neural model that takes the selected node embeddings $\boldsymbol{z}^i_{\{B\}}$ and outputs a variety of candidate translations $\delta^i_{\{B\}} \in \Delta$ that would move the object to a potential new placement region, i.e.,

$$\delta^i_{\{B\}} \leftarrow \pi(\boldsymbol{z}^i_{\{B\}}; \Omega) \qquad (4)$$

where $\delta^i_{\{B\}}$ contains a variety of δ actions of size B and $\boldsymbol{z}^i_{\{B\}}$ contains B replicas of $\boldsymbol{z}^i$. These $\delta^i_{\{B\}}$ actions are applied to B replicas of selected object's current point-cloud $X^i_{\{B\}}(t)$

which leads to its next placement in the point-cloud space, i.e.,

$$X^i_{\{B\}}(t+1) = \begin{bmatrix} X^i_1(t) \\ \vdots \\ X^i_B(t) \end{bmatrix} + \begin{bmatrix} \delta^i_1 \\ \vdots \\ \delta^i_B \end{bmatrix} \qquad (5)$$

Furthermore, this model obtains its stochasticity from Dropout layers [31] applied to the networks' linear layers with probability $p \in [0,1]$ during execution. For the given Graph $G = (V, E)$ and the true next step delta $\overline{\delta}$ from demonstrations, this module is trained by minimizing the following:

$$l_{\Omega,\Theta}(M, f_\Theta(G), \overline{\delta}) = \left\| \rho_\Omega \big(M \circ f_\Theta(G) \big) - \overline{\delta} \right\|_2 \qquad (6)$$

where M is a mask operator which selects the graph node embedding that corresponds to the given true delta $\overline{\delta}$ label only.

Goal Satisfaction Network: Our goal satisfaction network is a value function that scores the given actions for their ability to accomplish the given target arrangements. It is a neural function $r_\Psi : \mathcal{G} \times \Delta \mapsto [0,1]$, with parameters Ψ, that takes a selected graph node embedding and the given action δ as an input and outputs goal satisfaction scores, i.e.,

$$\hat{y}_r \leftarrow \sigma\big(r_\Psi(M \circ f_\Theta(G), \delta)\big) \qquad (7)$$

where $\hat{y}_r \in [0,1]$, M is a mask operator that selects a graph node embedding corresponding to the given action δ, and σ is a sigmoid function to squash predicted scores to $[0,1]$. The $\hat{y}_r = 1$ indicates that the given action δ can take the selected object to its target location, and $\hat{y}_r = 0$ indicates otherwise. Furthermore, this function is optimized through binary-cross entropy loss as follows:

$$l_{\Psi,\Theta}(M, f_\Theta(G), y_r) = -y_r \cdot \log(\hat{y}_r) - (1-y_r) \cdot \log(1-\hat{y}) \quad (8)$$

where y_r denotes the true label.

Collision Network: Our collision detection network $u_\xi : \mathcal{X} \mapsto [0,1]$, with parameters ξ, uses a number of PointNet++ set-abstraction layers [25] to detect the intersection between any two given sets $X_0, X_1 \subset X$. In our setting, X_0 and

X_1 can be any arbitrarily selected object point-clouds with their feature labeled as $Y_0 = \mathbf{0}$ and $Y_1 = \mathbf{1}$, respectively. Our collision network's input is a subset sampled from a joint-set of given point-clouds and their feature masks, i.e., $\boldsymbol{x} \subset (X_0, Y_0) \cup (X_1, Y_1)$. The feature mask indicates element-wise point-cloud correspondence to the given sets. The network predicts scores $\rho_c \in [0, 1]$ indicating degeree of sets' intersections. We train our collision-checker independently from other NeRP models using the BCE loss with training samples containing both intersecting and disjoint point-cloud sets.
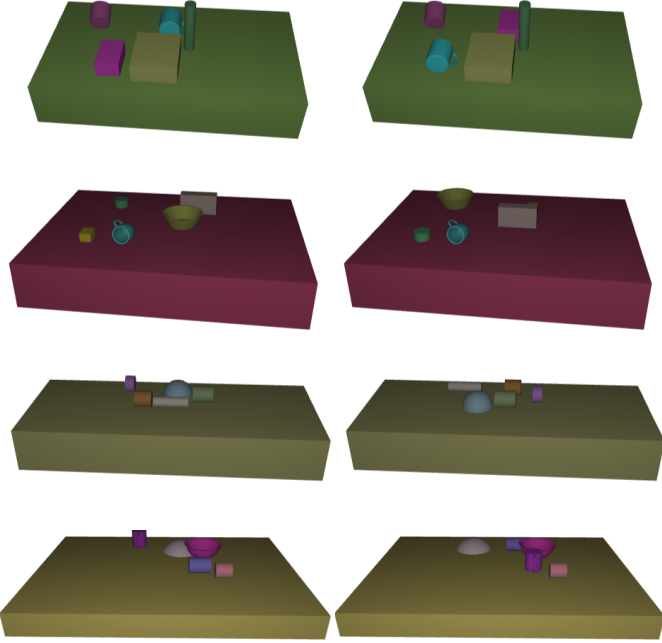

Fig. 3: Examples of generated data. Objects are randomly placed on the table, and we chose different random motions as well.

B. NeRP Training

We train our core models, i.e., graph encoder $f_\Theta(\cdot)$, node selector $h_\Phi(\cdot)$, δ-proposal network $\pi_\Omega(\cdot)$, and goal satisfaction evaluator $r_\Psi(\cdot)$, jointly in an end-to-end manner by optimizing the following objective:

$$\frac{1}{N_\mathcal{M}} \sum_{\tau \sim \mathcal{M}} l_{\Phi,\Theta}(\boldsymbol{z}_{\{N\}}, y_{\{N\}}) + l_{\Omega,\Theta}(\boldsymbol{z}^i, \overline{\delta}^i) + l_{\Psi,\Theta}(\boldsymbol{z}^i, \overline{\delta}^i, y_r),$$
(9)

where $\tau = (G, y_{\{N\}}, \overline{\delta}^i, M, y_r)$ denotes a training sample from demonstration data $\mathcal{M}$, comprising a current-target paired scene graph G with its node embeddings $\boldsymbol{z}_{\{N\}} = f_\Theta(G)$. It also includes the node selection labels as $y_{\{N\}}$, and a differentiable mask M operator to select a desired node embedding $\boldsymbol{z}^i = M \circ \boldsymbol{z}_{\{N\}}$. Furthermore, we also provide a desired next action $\overline{\delta}^i$ and its goal satisfaction value y_r for the selected node. The graph encoder is learned end-to-end with

other core models to capture scene embedding useful for the overall rearrangement planning process whereas the collision network is trained independently.

C. NeRP Planning Algorithm

Algorithm 1 describe our multi-step planning algorithm for efficiently solving tabletop rearrangement tasks. For the given observations, $X(t)$ and $X(T)$, our method begins by computing their instance segmentation K using UCN [34]. A current-target aligned scene graph G is then instantiated using those segmented observations and the objects similarity scores $\boldsymbol{s}$ (Lines 6-7).

Given an observation graph $G = (V, E)$, our multi-step predictive planning approach imagines n action sequences out to horizon h, stores them into a buffer $rollouts$, and executes the first action through a low-level controller of the best-selected planning sequence. In simulated environments, we set horizon length h to $2 \times |K|$, whereas in real-world experiments, we further limit the planning budget to $|K| + 1$. Furthermore, we use a Model Predictive Path Integral (MPPI) controller based on learned collision avoidance models [4] to perform the given placement actions leading to the next observation graph for the planning. After each pick-and-place action is executed, the horizon h is decremented and we replan. If controller fails to execute an action (e.g. it fails to grasp or drops the object early) the plan horizon is incremented again to give another opportunity to the planner for re-planning.

In a planning sequence, the graph encoder, f_Θ, outputs the graph nodes' embeddings $\boldsymbol{z}_{\{N\}}$. The function h_Φ takes graph embeddings and predicts $\rho_{\{N\}}$ which parametrize a Multinomial distribution for sampling an index $i \in [0, N]$. For a selected graph node's embedding $\boldsymbol{z}^i$, multiple replicas are fed to our stochastic δ-proposal network, π_Ω, to determine the various next step actions $\delta_{\{B\}}$ for the placements of the object i (Line 15-18).

Each of the candidate placements of the selected object in $\hat{X}^i_{\{B\}}(t)$ forms a new scenario. To determine the best next-step scene arrangement, we translate the object point-cloud according to the sampled δ and use our collision and goal-satisfaction networks. The collision-network, u_ξ, takes all possible next step objects point-clouds and returns action indices, ids, leading to all collision-free next-step arrangements (Line 22-24). The goal satisfaction network takes these collision-free placement actions, $\delta_{\{B'\}}$, and provides the scores, $v_{\{B'\}}$, to select the best move for simulating the next-step using the graph G and current point-cloud $X(t+1)$ (Line 25-28).

Furthermore, for each planning step, we compute a L2 norms based error e between the updated graph vertices $V(t)$ and $V(T)$, indicating difference between current simulated scene and the final arrangement. All this planning step information, including error e, is stored in the rollout buffer. Once all sequences are unrolled, the first step of the best-unrolled sequence, based on minimum error e, is selected to determine best object i and the placement points, X^i_{all}, and a placement cost map c_{map} for the object i. To select X^i_{all} and compute their cost-map c_{map}, we calculate the centroid, p_{best}, of the

next best placement point-cloud X^i_{best}, and select all points in X^i_{all} that are within a ball of radius of p_{best}.

The cost values are calculated based on the distance between p_{best} and all placement locations $X^i_{\mathcal{B}}$ (Line 38). These placement locations with cost map are given to the MPPI low-level controller for robot execution. The controller generates grasps for object i using [32] and executes the motions using [4]. Once the object is lifted, it chooses the placement with minimum c_{map} for which there is a collision-free, kinematically feasible path.

D. Data Generation

To train the NeRP models, we generate random synthetic scene data rendered with variable camera poses. All scenes contained randomly selected five objects, initially scattered all over the tabletop of changing dimensions (as shown in Fig. 3). Each scenario's target arrangement was determined by randomly swapping the objects' placements in the initial scene, so that each object's goal location is blocked by a random other object. Hence, transitioning from an initial to target scene requires some items to be moved to another empty location, which we call *storage*, to vacate the occupied position. We generate the intermediate placement action using a model-based expert rearrangement planner (as described in Section V). The expert planner generates intermediate scene sequences from which we determine the step-wise relative scene transformations $(M, \overline{\delta}) \in \mathcal{M} \times \Delta$ for training our model-free, set-based rearrangement planner. Following this procedure, we gathered a dataset comprising seven thousand re-arrangement task problems with their intermediate planning sequences.

E. Model Architectures

This section presents the model-architectures of NeRP. Our networks were implemented using PyTorch [23] and PyTorch Geometric Library [6].

- **Graph Encoder:** It contains two layers of k-GNNs [18] each with hidden feature size of 512 with max operator.
- **Object Selection Network:** The input graph node features of size 512 are given to two fully connected (FC) layers, followed by ReLU, and an output layer of size 512, 256, and 1, respectively. A sigmoid layer squashes the output, i.e., FC(512, 512) → ReLU→FC(512, 256) → ReLU→FC(256, 1) → Sigmoid.
- **δ-proposal Network:** The input node features of size 512 are passed through four FC layers, followed by a ReLU, and an output layer with 512, 384,256,128, and 3 units, respectively. The first three FC layers also contain Dropout with a probability of 0.5, i.e., FC(512, 512) → ReLU→ Dropout (0.5)→FC(512, 384) → ReLU→ Dropout (0.5)→ FC(384, 256) → ReLU→ Dropout (0.5)→FC(256, 128) → ReLU→ FC(128, 3)
- **Goal Satisfaction Network:** This network follows the same structure as the object selection network.
- **Collision Detection Network:** The collision network follows a structure of PointNet++ [25] classifier with

set abstraction (SA) and FC, followed by ReLU, layers. The SA layer's inputs are defined in the following format (sampling ratio, radius, MLP), where MLP denotes Multi-Layer Perceptron. Followed by the SA layers is a GlobalSA layer, i.e., SA(0.5, 0.4, MLP([4, 32, 32, 64])) → SA(0.25, 0.6, MLP([64 + 3, 128, 128])) → GlobalSA(MLP([128 + 3, 256, 512])) → FC(512, 256) → ReLU→FC(256, 128)→ReLU→FC(128, 1)→Sigmoid.

F. Hyperparameters

We use Adagrad optimizer [5] with learning rate= 0.01, learning rate decay= 0, initial accumulator value= 0, weight decay = 0, and eps= $1e - 10$.

V. RESULTS

We performed four sets of experiments. First, we tested our method on unseen synthetic data against various classical baselines. Second, we show generalization of our method on object rearrangement for unseen number of objects. Third, we do ablation study to evaluate effect of each component of NeRP. Fourth, we demonstrate our method's sim-to-real generalization performance on real-world object rearrangement tasks with different sets of unseen objects and unseen rearrangement tasks. We use the following metrics for quantitative comparison of different methods:

- **Success Rate** indicates the percentage of successfully solved unseen arrangement problems; an object arrangement is considered successful if the maximum displacement for each object does not exceed $5mm$.
- **Planning Steps** measures the number of steps required to rearrange the objects from source configuration to target configuration.
- **Final Error** measures the average L2 distance between the desired target arrangement and the actual arrangement achieved by a given planner.

Note that success rate is computed over all the rearrangement scenarios whereas planning steps and final error are computed only on successful rearrangements.

A. Algorithm Comparison

To validate our approach, we first tested on 500 simulated unseen object rearrangement problems that were generated with 5 objects following the procedure in Section IV-D. Table I shows how our method compares to multiple baseline methods. These baselines include:

Model-based expert: The expert planner is a model-based planning approach that uses objects' unique ids, transformations, table mesh, object meshes, and FCL [22] collision-checker for planning. For this model, we set the same step length limit as NeRP. It randomly selects an object to move that is not at its goal position, and then checks to see if its target location is empty or occupied. If the target location is occupied, it will move the occupying object to free space, chosen by randomly sampling a feasible *storage* location on the table. If the target location is free, it will instead move the
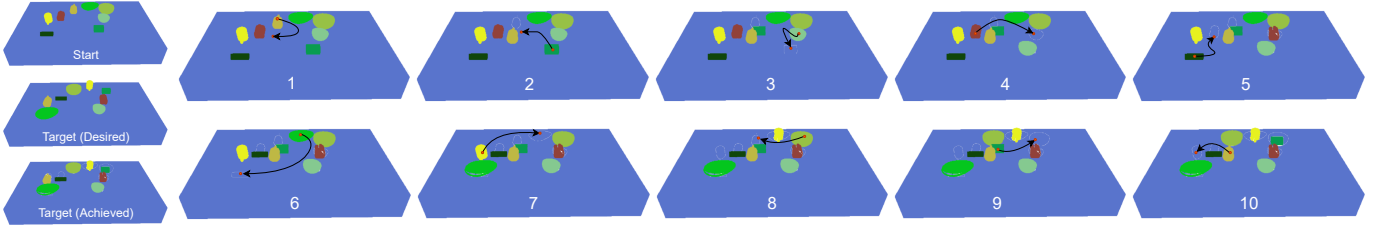
Fig. 4: An example plan rollout showing how NeRP chose to move objects around in order to get between two goal states with very different arrangements of obstacles. In this case, it took 10 steps to get to the goal state.

Algorithms	Performance Metrics		
	Success rate (%) ↑	Planning steps ↓	Final error ↓
Model-based expert	90.67 ± 0.60	8.41 ± 2.61	$\mathbf{0.0 \pm 0.0}$
NeRP (Ours)	$\mathbf{94.56 \pm 0.73}$	$\mathbf{7.01 \pm 2.10}$	0.019 ± 0.013
Classical (heuristic, parallelized)	68.20 ± 0.79	13.60 ± 5.99	0.023 ± 0.017
Classical (random, parallelized)	59.23 ± 1.32	42.80 ± 60.63	0.019 ± 0.011

TABLE I: Comparison between NeRP and several classical baselines. NeRP produces shorter, more accurate plans than baseline methods.

Number of Objects	Performance Metrics		
	Success rate (%) ↑	Planning steps ↓	Final error ↓
3 Objects	98.25 ± 0.57	4.58 ± 0.82	0.039 ± 0.036
4 Objects	97.60 ± 1.20	5.70 ± 1.38	0.027 ± 0.025
6 Objects	98.09 ± 0.40	8.69 ± 2.15	0.013 ± 0.013
7 Objects	90.62 ± 1.03	9.47 ± 2.23	0.011 ± 0.008
8 Objects	87.50 ± 2.50	10.72 ± 2.08	0.010 ± 0.008

TABLE II: Generalization of NeRP to different number of objects. The number of planning steps required increases more or less linearly, though the success rate drops slightly as we add more objects. Note that NeRP is trained on random rearrangements of 5 objects.

object to the goal. This process is repeated until all the objects are at their goal positions.

Classical (heuristic, parallelized): This is a model-free version of the "model-based expert" planner. It uses instance segmentation and our object alignment model to match objects between the current and target scenes, and follows a following greedy heuristics. Once objects pair are computed, it randomly selects an object, checks if its target location is occupied or empty, and moves the occupying object to storage by sampling multiple placements in parallel and rejecting colliding placement positions using our collision-net u_ξ instead of using FCL. Once the target location is free, it moves the selected object to its target, and proceeds by randomly selecting another object and repeating the process.

Classical (random, parallelized): The random baseline plans along multiple sequences, and executes the first action of the best selected sequence, following the algorithm given by Alg. 1. The best sequence is selected based on minimum error, e, from the target arrangement. However, unlike, NeRP, this baseline randomly chooses an object and performs a sequence execution like a classical (heuristic, parallelized) approach mentioned above.

Comparison to classical methods: Table I shows how NeRP out-performs all baseline methods, including the model-based expert that uses all environment state information. This is for several reasons. First, the random placements can often be in-collision, especially in cases with a small tabletop area. Second, model-free baselines diverge in cases with noisy feature-based objects' pairing, which is inevitable with learning-based feature matching approaches. Third, in the multi-step, multi-sequence classical planning cases, executing the first action and replanning the sequences again instead of greedily utilizing each action step takes a longer time to converge.

Generalization to different numbers of objects: Table II

Algorithms	Performance Metrics		
	Success rate (%) ↑	Planning steps ↓	Final error ↓
NeRP	94.56 ± 0.73	7.01 ± 2.10	0.019 ± 0.013
w/o Dropout	36.87 ± 0.24	8.23 ± 3.07	0.019 ± 0.012
w/o OS h_Φ	30.12 ± 1.10	11.82 ± 3.30	0.025 ± 0.021
w/o GS r_Ψ	48.21 ± 0.68	14.84 ± 1.14	0.016 ± 0.010

TABLE III: Analysis of the effects of ablation of various components of the network. Removing stochasticity, the object selection network or the goal selection network has a significant negative effect on performance.

shows NeRP performance over 500 scenarios with a number of objects ranging from two to eight, whereas NeRP training dataset contained scenes with only five objects. Note that we show final errors for only the successful task completions; due to this, the reported errors go down together with the success rates as the number of objects in the scene increases. Fig. 4 shows the NeRP execution in a scene arrangement task with eight objects. From these results, we can see that NeRP's object-centric planning is robust to scene clutter and can efficiently sample multiple *storage* locations along the planning sequence for achieving a complex target arrangement setup.

B. Ablation Studies

We performed a set of experiments ablating various components of our method. In particular, we look at the importance of the stochastic Dropout added to our δ-proposal network π_Ω, the object selection network h_Φ, and the goal satisfaction network r_Ψ.

The results are shown in Table III. We can see that all of these components are quite important. Without Dropout
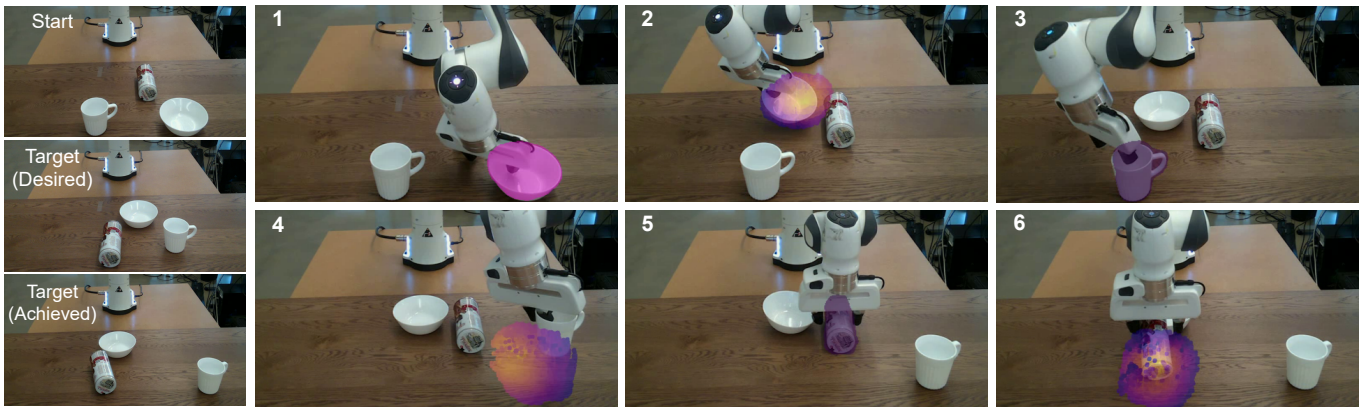
Fig. 5: Example of a planning sequence. The robot repeatedly selects which object to move and either moves it to the appropriate goal position or to a *storage* position in order to enable future execution.
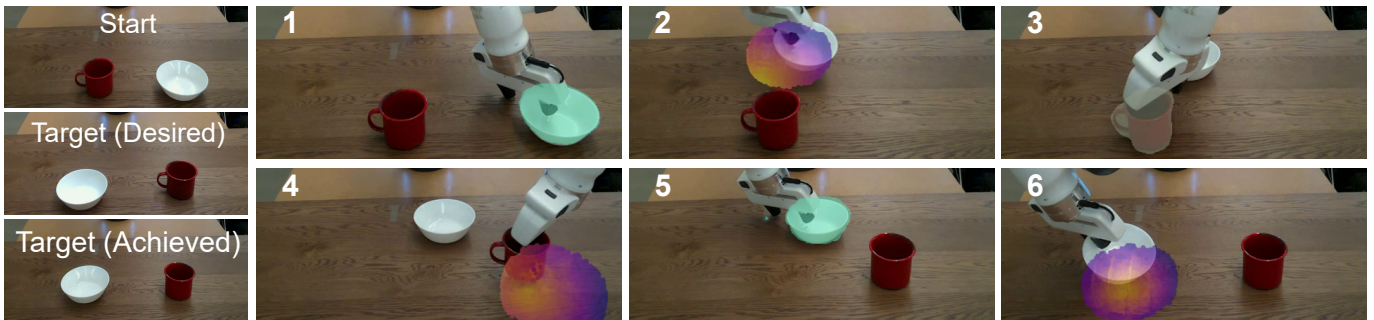

Fig. 6: Swapping an unseen mug and bowl using NeRP: For the given $X(\text{Start})$ and $X(\text{End})$ arrangements, NeRP generates an encoded scene graph using which the object selection network (h_Φ) selects an object in the given scenarios (e.g., 1, 3 & 5). The δ-proposal network (π_Ω) predicts $\delta \in \Delta$ for the selected object, leading to its next placement region with a cost map c_{map} (e.g., 2, 4 & 6). The c_{map} is originated based on the goal satisfaction network's (r_Ψ) scores, v, to indicate the robot with the best placement locations during execution.

[31], the architecture cannot find *storage* positions to place an object if the goal is blocked, making it challenging to swap two objects – a deterministic network could potentially learn this behavior, but it would be a much more difficult and less generalizable solution, and perhaps harder to learn from random imitation data. Without r_Ψ, we see much longer plans as the model makes numerous small adjustments without terminating, as evidenced by the lower final error. Without object selection, we select random objects that are not at their final positions, also leading to more inefficiency.

C. Real Robot Experiments

Finally, we performed a set of real-world experiments using a Franka Panda robot arm with an externally mounted Intel Realsense L515 camera. We set up the two types of testing scenarios i) swapping objects to achieve a given target arrangement and ii) sorting objects into different categories similar to the given target observation. Fig. 1 shows the robot sorting bowls and mugs, Fig. 5 shows the swapping of two objects and Fig. 6 demonstrates the swapping tasks based on NeRP's predicted actions. However, the major limiting factors in real-robot tasks were noisy scene segmentation and feature

extractions, which often led to rearrangement failure due to incorrect object correspondences, as also highlighted in the supplementary video.

We see in all these different planning setups that despite being trained on synthetic data with only five object categories, NeRP generalizes to novel problem settings with high performance. In sim-to-real transfer, it generates actions for the low-level controller to move the object, while in other scenarios, the objects were directly teleported to the best placement location. See the supplementary materials for additional examples of NeRP real-world execution.

VI. CONCLUSIONS

We presented Neural Rearrangement Planning (NeRP), a deep neural network-based rearrangement approach for unknown objects. NeRP can rearrange unseen objects without models, and works in an end-to-end fashion, given segmented point-clouds from an RGB-D camera. We evaluate NeRP on challenging problems and demonstrate its sim-to-real generalizations. NeRP relies on scene segmentation and correspondence matching techniques to generate a scene graph between the current and target observations. The scene graph is used to

generate a sequence of intermediate object selection and their placement actions for reaching the given target arrangement. However, in our sim-to-real transfer experiments, we observed, as also shown in our supplementary video, that existing scene segmentation and feature-based object correspondence techniques often fail in the real-setup. This results in an incorrect scene graph and, therefore NeRP behavior.

Hence, in our future works, we plan to augment NeRP with robust segmentation and feature matching algorithms to enhance its real-world settings' performance. Another of our future objectives is to extend NeRP for model-free planning in SE-3 space to compute both relative translation and rotations for transforming point sets into complex shapes and arrangements.

REFERENCES

[1] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020.

[2] Daniel M Bear, Chaofei Fan, Damian Mrowca, Yunzhu Li, Seth Alter, Aran Nayebi, Jeremy Schwartz, Li Fei-Fei, Jiajun Wu, Joshua B Tenenbaum, et al. Learning physical graph representations from visual scenes. *arXiv preprint arXiv:2006.12373*, 2020.

[3] Akansel Cosgun, Tucker Hermans, Victor Emeli, and Mike Stilman. Push planning for object placement on cluttered table surfaces. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 4627–4632. IEEE, 2011.

[4] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. *International Conference on Robotics and Automation*, 2021.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 (61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

[6] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[7] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *arXiv preprint arXiv:2010.01083*, 2020.

[8] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5678–5684. IEEE, 2020.

[9] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition*, 2015.

[11] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8565–8574, 2019.

[12] Brian Ichter, Pierre Sermanet, and Corey Lynch. Broadly-exploring, local-policy trees for long-horizon task planning. *arXiv preprint arXiv:2010.06491*, 2020.

[13] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*, 2017.

[14] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 38(7):793–812, 2019.

[15] Jennifer E King, Vinitha Ranganeni, and Siddhartha S Srinivasa. Unobservable monte carlo planning for non-prehensile rearrangement tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4681–4688. IEEE, 2017.

[16] Yann Labbé, Sergey Zagoruyko, Igor Kalevatykh, Ivan Laptev, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Monte-carlo tree search for efficient visually guided rearrangement planning. *IEEE Robotics and Automation Letters*, 5(2):3715–3722, 2020.

[17] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-net 3.0: computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning. *arXiv preprint arXiv:1709.06670*, 2017.

[18] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[19] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910, 2019.

[20] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. 6-dof grasping for target-driven object manipulation in clutter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6232–6238. IEEE, 2020.

[21] Changjoo Nam, Jinhwi Lee, Younggil Cho, Jeongho Lee, Dong Hwan Kim, and ChangHwan Kim. Plan-

ning for target retrieval using a robotic manipulator in cluttered and occluded environments. *arXiv preprint arXiv:1907.03956*, 2019.

[22] J. Pan, S. Chitta, and D. Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012. doi: 10.1109/ICRA.2012.6225337.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[24] Chris Paxton, Yotam Barnoy, Kapil Katyal, Raman Arora, and Gregory D Hager. Visual robot task planning. In *2019 international conference on robotics and automation (ICRA)*, pages 8832–8838. IEEE, 2019.

[25] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[26] Ahmed H Qureshi, Jiangeng Dong, Asfiya Baig, and Michael C Yip. Constrained motion planning networks x. *arXiv preprint arXiv:2010.08707*, 2020.

[27] Ahmed H Qureshi, Jiangeng Dong, Austin Choe, and Michael C Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.

[28] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 2020.

[29] Anthony Simeonov, Yilun Du, Beomjoon Kim, Francois R Hogan, Joshua Tenenbaum, Pulkit Agrawal, and Alberto Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. *arXiv preprint arXiv:2011.08177*, 2020.

[30] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4732–4741. PMLR, 2018.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[32] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[34] Yu Xiang, Christopher Xie, Arsalan Mousavian, and Dieter Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. *arXiv preprint arXiv:2007.15157*, 2020.

[35] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on robot learning*, pages 1369–1378. PMLR, 2020.

[36] Danfei Xu, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Deep affordance foresight: Planning through what can be done in the future. *arXiv preprint arXiv:2011.08424*, 2020.

[37] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020.