Seeing in 3D: Towards Generalizable 3D Visual Representations for Robotic Manipulation

Haolun (Harry) Zhang

CMU-RI-TR-23-16

May, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. David Held, *chair*
Prof. Jeffrey Ichnowski
Jianren Wang

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my family.

Abstract

Despite the recent progress in computer vision and deep learning, robot perception remains a tremendous challenge due to the variations of the objects and the scenes in manipulation tasks. Ideally, a robot trying to manipulate a new object should be able to reason about the object's geometric, physical, and topological properties. In this thesis, we aim to investigate different strategies for enabling a robot to reason about objects using 3D visual signals in a generalizable manner.

In the first project, we propose a vision-based system, FlowBot 3D, that learns to predict the potential motions of the parts of a variety of articulated objects to guide downstream motion planning of the system to articulate the objects. To predict the object motions, we train a neural network to output a dense vector field representing the point-wise motion direction of the points in the point cloud under articulation. We then deploy an analytical motion planner based on this vector field to achieve a provably optimal policy. We train a single vision model entirely in simulation across all categories of objects, and we demonstrate the capability of our system to generalize to unseen object instances and novel categories in both simulation and the real world using the trained model for all categories, deploying our policy on a Sawyer robot.

In the second project, we focus on the task-specific pose relationship between relevant parts of interacting objects. We conjecture that this relationship is a generalizable notion of a manipulation task that can transfer to new objects in the same category; examples include the relationship between the pose of a pan relative to an oven or the pose of a mug relative to a mug rack. We call this task-specific pose relationship "cross-pose". We propose a vision-based system, TAX-Pose, that learns to estimate the cross-pose between two objects for a given manipulation task using learned cross-object correspondences. The estimated cross-pose is then used to guide a downstream motion planner to manipulate the objects into the desired pose relationship. We demonstrate our method's capability to generalize to unseen objects in the real world.

We also demonstrate that we are able to combine the two systems together using weighted SVD for more complex manipulation tasks that involve both articulated and free-floating objects. By finetuning pretrained FlowBot 3D and TAX-Pose models, we show that we can generalize to a wider variety of manipulation tasks and even planning.

Acknowledgments

First and foremost, I'm deeply grateful to my advisor, David Held, who has consistently supported me for the past two years. I am fortunate to have Prof. Held's thoughtful guidance throughout the master's study. The weekly meeting is the most enjoyable and rewarding part, where Prof. Held always shares his valuable insights and teaches me to conduct research in a systematic way.

I would also like to thank my mentor, Ben Eisner, who is a role model that I look up to. Whenever I'm stuck with problems, he always inspires me with wonderful suggestions and ideas. I thank Prof, Jeffrey Ichnowski and Jianren Wang for being my committee members and providing helpful feedback.

It is also my great fortune to meet and work with a group of lovely people at CMU and RPAD - Fan Yang, Carl Qi, Yufei Wang, Tianyuan Zhang, Chu Er Pan, Brian Okorn, Jianren Wang, Heng Yu, Xuxin Cheng, Daniel Seita, Wenxuan Zhou and Thomas Weng. Particularly, I would like to thank Zixuan Huang for offering late-night snacks and support when we both stayed late before the paper deadlines. I give my sincere appreciation for all your kind help during my study here.

At last, I also would like to thank my wife - Klara Guan, my parents - Jinhang Zhang and Lihong Hao, and my grandparents - Tongbin Hao, Xiuyun Wang, Zhihua Zhang, and Lianyun Xu, for their continuous love and support along every step of my life.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Robotic manipulation is an interesting problem to solve in order to make robots to fully integrate into our society. Robotic manipulation can be roughly decomposed into two components: perception and planning. Perception is vital in the manipulation pipeline in that the signals obtained from the perception system are used downstream to guide the planning module. However, manipulation tasks remain challenging due to several reasons. Particularly, many end-to-end approaches are known to be confined to the training tasks and are not capable of generalizing to new manipulation tasks and new objects of interests. Moreover, state estimation (such as object pose and parameters estimation) poses yet another challenge due to the methods' brittleness. In this thesis, we study the problem of generalizable 3D objects reasoning in robotic perception systems.

Why do we need to tackle 3D visual perception in manipulation tasks? Our key insight is that once "good" visual model is learned for a task, the output of the visual model should be able to substantially facilitate the downstream planning of the task after careful design. We need to take a more holistic view of perception and planning, designing each module with an awareness of how it will be used within the context of the robotic system.

How do we define generalizability? In this thesis, we focus on being able to generalize to inter-class and intra-class variations across different objects. Specifically, we would like to design perception systems that are able to reason about unseen objects from the same class and from a different class in a consistent manner.

Towards this end, in this thesis, we study the problem of learning 3D generalizable visual perception for two categories of objects: articulated objects and free-floating objects.

- In Chapter 2, we introduce FlowBot 3D, a novel method to perceive and manipulate 3D articulated objects that generalizes to enable a robot to articulate unseen classes of objects. FlowBot 3D learns to predict a dense vector field representing the point-wise motion direction of the points in the point cloud under articulation. An analytical motion planner based on this vector field to achieve a policy that yields maximum articulation will then be deployed to execute the action. Experiments suggest that FlowBot 3D is able to successfully manipulate a wide varirety of real-world articulated objects.
- In Chapter 3, we introduce TAX-Pose (Task-Specific Cross-Pose Estimation) for robotic manipulation. We propose a vision-based system that learns to estimate the cross-pose between two objects for a given manipulation task using learned cross-object correspondences. The estimated cross-pose is then used to guide a downstream motion planner to manipulate the objects into the desired pose relationship (placing a pan into the oven or the mug onto the mug rack). We demonstrate our method's capability to generalize to unseen objects, in some cases after training on only 10 demonstrations in the real world. Results show that our system achieves state-of-the-art performance in both simulated and real-world experiments across a number of tasks.
- In Chapter 4, we propose one possible method to combine the two systems using weighted SVD. The resulting combined model is able to reason about both articulated and free-floating objects and thus will be able to accomplish a wider variety of complex manipulation tasks that involve both categories of objects.

Chapter 2

FlowBot3D: Learning 3D Articulation Flow to Manipulate Articulated Objects

2.1 Introduction

Understanding and being able to manipulate articulated objects such as doors and drawers is a key skill for robots operating in human environments. While humans can rapidly adapt to novel articulated objects, constructing robotic manipulation agents that can generalize in the same way poses significant challenges, since the complex structure of such objects requires three-dimensional reasoning of their parts and functionality. Due to the large number of categories of such objects and intra-class variations of the objects' structure and kinematics, it is difficult to train efficient perception and manipulation systems that can generalize to those variations.

To address these challenges, we propose to separate this problem into one of "affordance learning" and "motion planning." If a robot can predict the potential movements of an objects' parts (a.k.a. "affordances"), it would be relatively easy for the agent to derive a downstream manipulation policy by following the predicted motion direction. Thus, we tackle the problem of manipulating articulated objects by learning to predict the motion of individual parts on articulated objects.

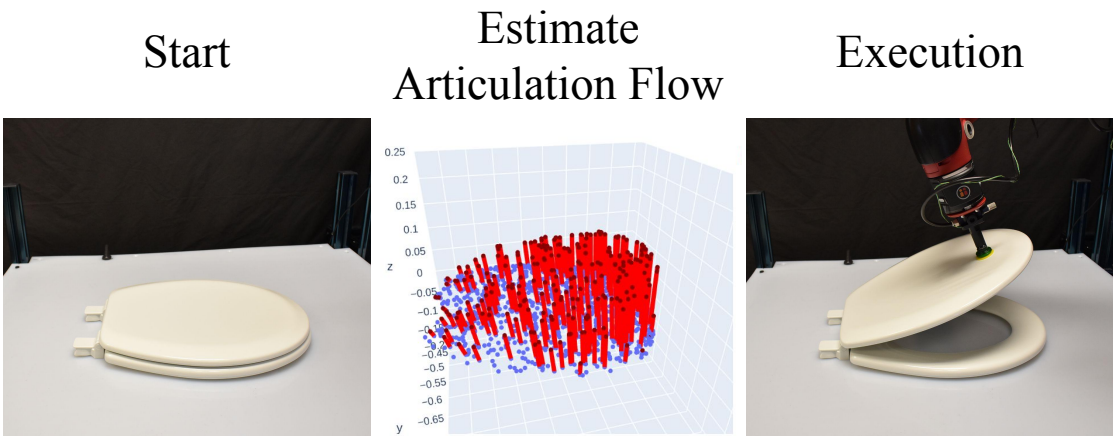


Figure 2.1: FlowBot3D in action. The system first observes the initial configuration of the object of interest, estimates the per-point articulation flow of the point cloud (3DAF), then executes the action based on the selected flow vector. Here, the red vectors represent the direction of flow of each point (object points appear in blue); the magnitude of the vector corresponds to the relative magnitude of the motion that point experiences as the object articulates.

Previous work has proposed to learn the articulation parameters (i.e. rotation axis of revolute joints and translation axis of prismatic joints) in order to guide the manipulation policy [7]. However, such methods often rely on knowing class-specific articulation structures. Without such knowledge, the policies can neither operate nor be applied to novel categories.

To learn a generalizable perception and manipulation pipeline, we need to be robust to the variations of the articulated objects' geometries and kinematic structures. We seek to construct a vision system that can learn to predict how the parts move under kinematic constraints without explicitly knowing the articulation parameters: specifically, the location of the rotational or translational axes for revolute or prismatic parts, respectively.

In this paper [15], we present FlowBot3D, a deep 3D vision-based robotic system that predicts dense per-point motion of an articulated object in 3D space, and leverages this prediction to produce actions that articulate the object. We define such per-point motion as the **3D articulation flow (3DAF)** vectors, since this representation describes how each observed point on the articulated part would "flow" in the 3D space under articulation motion. Such a dense vector field prediction can

then be used to aid downstream manipulation tasks for both grasp point selection as well as predicting the desired robot motion after grasping.

We train a ***single*** 3D perception module to perform this task across many object categories, and show that the trained model generalizes to a wide variety of objects – both in seen categories, and entirely unseen object categories.

The contributions of this paper include:

1. A novel per-point representation of the articulation structure of an object, 3D Articulation Flow (3DAF).
2. A novel 3D vision neural network architecture (which we call ArtFlowNet) that takes as input a static 3D point cloud and predicts the 3D Articulation Flow of the input point cloud under articulation motion.
3. A novel robot manipulation system (FlowBot3D) for using the predicted 3D Articulation Flow to manipulate articulated objects.
4. Simulated experiments to test the performance of our system in articulating a wide range of PartNet-Mobility dataset objects.
5. Real-world experiments deployed on a Sawyer robot to test the generalizablity and feasibility of our system in real-world scenarios.

2.2 Related Work

2.2.1 Articulated Object Manipulation

Manipulation of articulated objects and other objects with non-rigid properties remains an open research area due to the objects' complex geometries and kinematics. Previous work proposed manipulating such objects by hand-designed analytical methods, such as the immobilization of a chain of hinged objects by Cheong et al. [7]. Berenson et al. [3] proposed a planning framework for manipulation under kinematic constraints. Katz et al. [26] proposed a method to learn such manipulation policies in the real-world using a grounded relational representation learned through interaction.

With the development of larger-scale datasets of articulated objects such as the PartNet dataset by Mo et al. [32] and Partnet-Mobility by Xiang et al. [53], several works have proposed learning methods based on large-scale simulation and supervised

visual learning. Mo et al. [33] proposed to learn articulation manipulation policies through large-scale simulation and visual affordance learning. Xu et al. [55] proposed a system that learns articulation affordances as well as an action scoring module, which can be used to articulate objects. Mu et al. [34] provided a variety of baselines for the manipulation tasks of 4 categories of articulated objects in simulation. Several works have focused specifically on visual recognition and estimation of articulation parameters, learning to predict the pose [22, 27, 49, 56, 57] and identify articulation parameters [25, 59] to obtain action trajectories. Moreover, [5, 8, 35] tackle the problem using statistical motion planning.

2.2.2 Optical Flow for Policy Learning

Optical flows [20] are used to estimate per-pixel correspondences between two images for object tracking and motion prediction and estimation. Current state-of-the-art methods for optical flow estimation leverage convolutional neural networks [14, 24, 47]. Amiranashvili et al. [1], Dong et al. [13] use optical flow as an input representation to capture object motion for downstream manipulation tasks. Weng et al. [52] uses flow to learn a policy for fabric manipulation. While the aforementioned optical flows are useful for robotic tasks, we would like to generalize the idea of optical flow beyond pixel space into full three-dimensional space. Instead, we introduce "3D Articulation Flow", which describes per-point correspondence between two point clouds of the same object. Another work that is highly related to ours is Pillai et al. [41], which learns to predict the articulated objects' parts motion using a motion manifold learner. First, while we both predict the parts' motion to derive an implicit policy, we do not rely on the intermediate articulation parameters in order to predict the motion manifold. Second, we do not rely on any demonstration to learn from - our method learns in a completely self-supervised fashion.

2.3 Method - From Theory to Practice

In this section, we examine the physical task of manipulating the articulation of an articulated object. We first present the theoretical grounding behind the intuition of our method, and we slowly relax assumptions and approximations to create a system

that articulates objects in the real world based on point cloud observations.

2.3.1 An Idealized Policy Based On Dynamics and Kinematics

The articulated objects we consider in this work are generally objects that 1) consist of one or more rigid-bodies – or "links" – which are 2) connected to one another by revolute or prismatic joints with exactly 1 degree of freedom each, and 3) have at least one link rigidly attached to an immovable world frame so that the only motion the object experiences is due to articulation. Each joint connects a *parent link* (often the fixed-world link) and a *child link*, which can move freely subject to the articulation constraints. While these conditions may seem restrictive, under normal "everyday" forces many real-world articulated objects (ovens, boxes, drawers, etc.) meet these conditions to a very good approximation.[1]

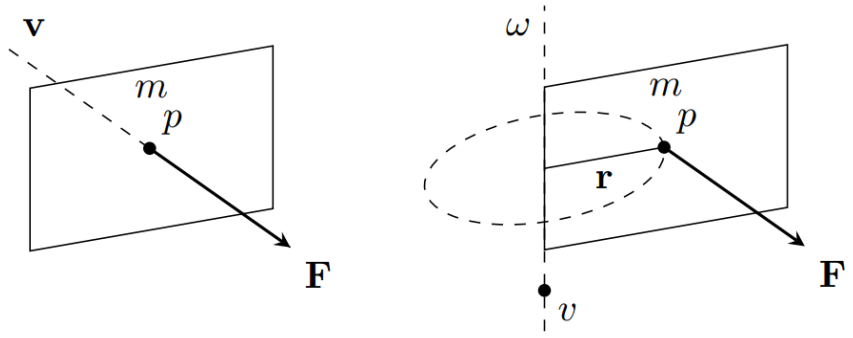


Figure 2.2: Illustrations of prismatic (Left) and revolute (Right) joints.

We now consider an idealized policy to actuate an articulated object. Suppose we are able to attach a gripper to any point $p \in \mathcal{P}$ on the surface $\mathcal{P} \subset \mathbb{R}^3$ of a child link with mass m. At this point, the policy can apply a 3D force $\mathbf{F}$, with constant magnitude $||\mathbf{F}|| = C$ to the object at that point. Our objective is to choose a contact point and force direction $(p^*, \mathbf{F}^*)$ that maximizes the acceleration $\mathbf{a}$ of the articulation's child link. If we limit our analysis to two special classes of articulation,

[1]We therefore exclude objects with socket joints, free-body objects, and deformable objects from our analysis.

revolute joints and prismatic joints, we can very intuitively arrive at the following optimal settings of $(p^*, \mathbf{F}^*)$:

Prismatic: A prismatic joint (such as a drawer) can be described as a single 3D unit vector $\mathbf{v}$ which is parallel to its direction of motion. Since motion of the joint is constrained to $\mathbf{v}$, the object will provide a responding force $\mathbf{F}_n$ to any component of $\mathbf{F}$ not parallel to $\mathbf{v}$. The net force exerted on the joint by the robot is thus $\mathbf{F}_{\text{net}}$, the component of $\mathbf{F}$ in the $\mathbf{v}$ direction:

$$\begin{aligned}\mathbf{F}_{\text{net}} &= \mathbf{F} - \mathbf{F}_n \\ &= \mathbf{F} - (\mathbf{F} - (\mathbf{F} \cdot \mathbf{v})\mathbf{v}) = (\mathbf{F} \cdot \mathbf{v})\mathbf{v} = m\mathbf{a}\end{aligned} \tag{2.1}$$

As one might expect, the force vector $\mathbf{F}^*$ which maximizes the acceleration $\mathbf{a}$ occurs when $||\mathbf{F}^* \cdot \mathbf{v}|| = C$, i.e. when $\mathbf{F}^*$ is parallel to $\mathbf{v}$. Because each point $p \in \mathcal{P}$ moves in parallel, applying the force at any point p on the surface will yield the maximum acceleration. Thus, the optimal policy to articulate a prismatic joint is to select any point on the surface and apply a force parallel to $\mathbf{v}$ at every time step.

Revolute: A revolute joint (such as a door hinge) can be parameterized by a pair $(v, \boldsymbol{\omega})$, where $\boldsymbol{\omega}$ is a unit vector representing the direction of the axis of rotation about which the child link moves, and $v \in \mathbb{R}^3$ is a point in 3D space that the axis of rotation passes through. Each point p on the child link is constrained to move on the 2D circle perpendicular to the axis of rotation with radius $\mathbf{r}$ (where $||\mathbf{r}||$ is the length of the shortest vector from p to the line given by $f(t) = v + t\boldsymbol{\omega}$). Given any point p, we can maximize the acceleration by a similar argument as before, except any force in the direction of $\mathbf{r}$ or $\boldsymbol{\omega}$ will be resisted:

$$\mathbf{F}_{\text{net}} = \mathbf{F} - \mathbf{F}_n = \mathbf{F} - \left(\frac{\mathbf{F} \cdot \mathbf{r}}{||\mathbf{r}||^2}\right)\mathbf{r} - (\mathbf{F} \cdot \boldsymbol{\omega})\boldsymbol{\omega} \tag{2.2}$$

Thus, for any point p the net force (and thus acceleration) is maximized when $\mathbf{F}^*$ is tangent to the circle defined by $\mathbf{r}$. Selecting the point p which produces the maximal linear acceleration when $\mathbf{F}^*$ is applied there is simply the point p on the child link that maximizes $||\mathbf{r}||$, or the point on the object farthest from the axis of rotation $\boldsymbol{\omega}$. Thus, the optimal policy to articulate a revolute joint is to pick the point on the surface farthest from the axis of rotation $\boldsymbol{\omega}$ and apply a force parallel to $\mathbf{r} \times \boldsymbol{\omega}$

at every time step.

2.3.2 Articulation Parameters to 3D Articulation Flow

These parameterizations[2] are an elegant representation of single articulations in isolation. However, when an object contains more than one articulation, or contains points that do not move at all (e.g. the base of a cabinet), in order to create a minimal parametric representation of the object we must describe a kinematic tree (a tree of rigid links, connected by joints described by a set of parameters) and associate each point on the object with a link. This is a hierarchical representation, which is difficult to construct from raw observation without prior knowledge of the hierarchical structure or link membership. A hierarchy-free representation of the kinematic properties of the object could assign each point on the object its own set of parameters; however, this would require a full 6 parameters (v, ω) for each point on the object, and the position v can occur anywhere in $\mathbb{R}^3$ depending on the object's coordinate frame. A more compact, bounded, hierarchy-free representation is the **3D articulation flow (3DAF)** that each point on the object would experience were its part articulated in the positive direction with respect to its articulation parameters. In other words, for each point on each link on the object, define a vector in the direction of motion of that point caused by an infinitesimal displacement $\delta\theta$ of the joint, and normalize it by the largest such displacement on the link. Thus, the 3D articulation flow f_p for point $p \in \mathcal{P}_i$ in link i is:

$$f_p = \begin{cases} \mathbf{v}, & \text{if } i \text{ is a prismatic joint} \\ \frac{\omega \times \mathbf{r}}{||\omega \times \mathbf{r}_{\max}||} & \text{if } i \text{ is a revolute joint} \end{cases} \tag{2.3}$$

where $\mathbf{v}, \omega$, and $\mathbf{r}$ are defined above; note that $\mathbf{v}$ is already a unit vector. We denote the full set of flow vectors for an object as $F = \{f_p\}_{p \in \mathcal{P}}$ where $\mathcal{P} = \cup_i \mathcal{P}_i$.

While this representation is mathematically equivalent to both the hierarchical and point-wise parameter-based representations, 3D articulation flow has several key advantages over parameter-based representations:

[2]An astute reader may recognize that these parameterizations are special cases of twists from Screw Theory. Without loss of accuracy, we choose to omit a rigorous screw-theoretic treatment of articulated objects in favor of an explanation that requires only basic knowledge of physics.

1. It is hierarchy-free, meaning that it can be easily approximated without an explicit model (i.e. kinematic structure); this property will allow our learned method to generalize to novel object categories.
2. Each element in the representation is a scaled orientation vector constrained to lie inside the unit sphere in $\mathbb{R}^3$. This means that the representation is invariant under translation and scaling in the coordinate frame of the underlying object.

Since this representation is defined for any arbitrary point in or on an object, it could be applied to any discrete or continuous geometric representation of said object. However, for the purposes of this work, we apply this representation to 3D point clouds produced from depth images. Thus for a pointcloud $P = \{p_k \in \mathbb{R}^3\}_{k\in[n]}$, we associate each point p_k in P with a flow vector $f_k \in \mathbb{R}^3, \text{s.t.} ||f_i|| \leq 1$.

This formulation of 3D articulation flow is similar in spirit to the intermediate representation proposed by Zeng et al. [59] in their articulation estimation system, FormNet. However, our representation differs in two key ways. First, our representation describes the instantaneous motion of a link, whereas the FormNet formulation predicts the current absolute displacement of a part from a reference position (i.e. a fully-closed door). Second, we demonstrate that our formulation can be used directly by a manipulation policy, whereas the downstream task of FormNet's representation was predicting the articulation parameters of an object.

2.3.3 Predicting 3D Articulation Flow from Vision

We now turn to the question of estimating 3D Articulation Flow from a robot's sensor observations. We consider a single articulated object in isolation; let $s_0 \in \mathcal{S}$ be the starting configuration of the scene with a single articulated object where $\mathcal{S}$ is the configuration space. We assume that the robot has a depth camera and records point cloud observations $O_t \in \mathbb{R}^{3\times N}$, where N is the total number of observable points from the sensor. The task is for the robot to articulate a specified part through its entire range of motion.

For each configuration s_t of the object, there exists a unique ground-truth flow F_t, where the ground-truth flow of each point is given by Equation 2.3. Thus, we would like to find a function $f_\theta(O_t)$ that predicts the 3D articulation flow directly from point cloud observations. We define the objective of minimizing the L2 error of

the predicted flow:

$$\mathcal{L}_{\text{MSE}} = \sum_i ||F_{t,i} - f_\theta(O_t)_i||_2 \tag{2.4}$$

where i indexes over the objects in the training dataset. While f_θ can be any estimator, we choose to use a neural network, which can be trained via a standard supervised learning with this loss function.

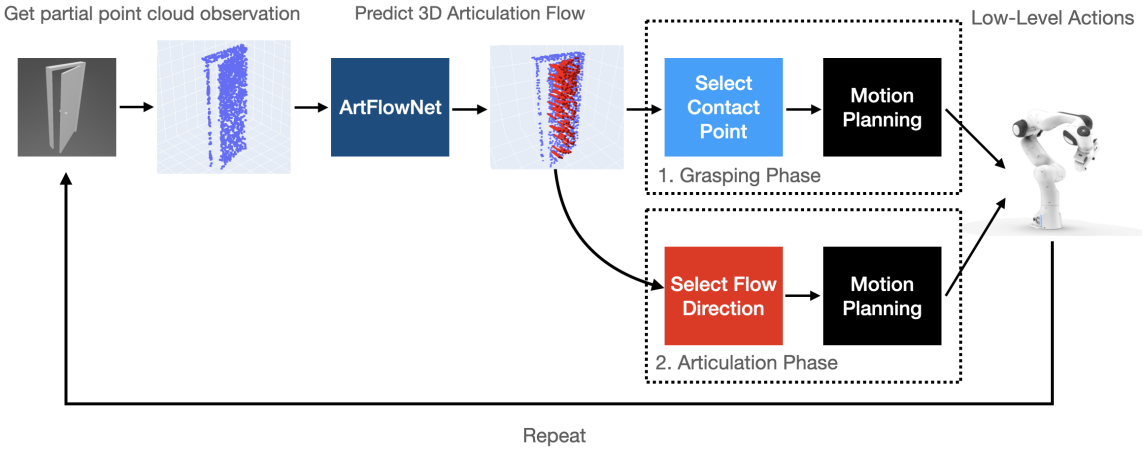


Figure 2.3: FlowBot3D System Overview. Our system in deployment has two phases: the Grasp-Selection phase and the Articulation-Execution Phase. The dark red dots represent the predicted location of each point, and the light red lines represent the flow vectors connecting from the current time step's points to the predicted points. Note that the flow vectors are downsampled for visual clarity. In Grasp-Selection Phase, the agent observes the environment in the format of point cloud data. The point cloud data will then be post-processed and fed into the ArtFlowNet, which predicts per-point 3D flow vectors. The system then chooses the point that has the maximum flow vector magnitude and deploys motion planning to make contact with the chosen point using suction. In Articulation-Execution phase, after making suction contact with the chosen argmax point, the system iteratively observes the pointcloud data and predicts the 3D flow vectors. In this phase, the motion planning module would guide the robot to follow the maximum observable flow vector's direction and articulate the object of interest repeatedly.

2.3.4 A General Policy using 3D Articulation Flow

Our method first takes an observation O_0 and estimates the 3D articulation flow $\hat{F}_0 = f_\theta(O_0)$ for all points in the observation. Given the estimate of the 3D articulation flow $\hat{F}_0$, we now describe a general, closed-loop policy which takes flow as input and actuates an articulated object. The policy is executed in two phases:

Algorithm 1 The FlowBot3D articulation manipulation policy

Require: $\theta \leftarrow$ parameters of a trained flow prediction network
  $(O_0) \leftarrow$ Initial observation
  $\hat{F}_0 \leftarrow f_\theta(O_0, [M_0])$, Predict the initial flow
  $g_0 = \texttt{SelectContact}(O_0, \hat{F}_0)$, Select a contact pose.
  $in\_contact \leftarrow$ False
  while not $in\_contact$ **do**
    Drive an end effector towards g_0
    if `DetectContact()` **then**
      $in\_contact \leftarrow$ True
      `Grasp`(g_0)
  $done \leftarrow$ False
  while not $done$ **do**
    $(O_t) \leftarrow$ Observation
    $\hat{F}_t \leftarrow f_\theta(O_t, [M_t])$, Predict the current flow
    $v_t \leftarrow$ `SelectDirection()`
    Apply a force to the end-effector in the direction of v for small duration t
    $done \leftarrow$ `EpisodeComplete()`

1) Grasp Selection: Based on the estimated 3D articulation flow $\hat{F}_0$, the policy must decide the best place to grasp the object. In this work, we assume access to a suction-type gripper that (in the ideal case) can grasp any point on the object surface. We know that the ideal attachment point is the location on a part where the flow has the highest magnitude in order to achieve the most efficient actuation of the articulated part by maximizing its acceleration, as we showed by maximizing Equations 2.1 and 2.2. We use motion planning to move the end effector to this point, with the end-effector aligned to directly oppose the flow direction. We then grasp the object at this position (using a suction gripper), shown in the left hand side of Fig. 2.3. We assume a rigid contact between the gripper and this contact point going forward.

2) Articulation Execution: At each time step t, we record a new observation O_t and estimate the current flow $\hat{F}_t$. We then select the predicted flow direction $\mathbf{v}_t$ with the greatest magnitude from the visible points from the observation, as shown in the right hand side of Fig. 2.3. To handle objects with multiple articulated parts, we only consider flow vectors close to our point of contact (the contact point itself is likely occluded by the gripper and is thus not visible). While continuing to grasp

the object, we then move the gripper in the direction $\mathbf{v}_t$. This process repeats in a closed loop fashion until the object has been fully-articulated, a max number of steps has been exceeded, or the episode is otherwise terminated. See Algorithm 1 for a full description of the generalized flow articulation algorithm.

2.3.5 FlowBot3D: A Robot Articulation System

With all the pieces of our generalized articulation policy in place, we now describe a real-world robot system – FlowBot3D – which leverages this generalized articulation policy. We define a tabletop workspace that includes a Sawyer BLACK 7-DoF robotic arm mounted to the tabletop with a pneumatic suction gripper as its end-effector, and an Azure Kinect RGB-D camera mounted at a fixed position and pointing at the workspace. See Figure 2.4 for an image of the workspace. We obtain point cloud observations of the scene from the Azure Kinect in the robot's base frame, filtering out non-object points, we use the method proposed in [60] to denoise the data (see supplementary materials for details). For robot control, we use a sampling-based planner, MoveIt! [9], which can move our robot to any non-colliding pose in the scene; we thus use motion planning to move the gripper to a pre-grasp pose. For the grasp and articulation, we directly control the end-effector velocity.

To select the point of contact for the suction gripper, we need to make some modifications from the idealized system described earlier. Unfortunately, a real suction gripper cannot make a proper seal on locations with high curvature (i.e. edges of the object and uneven surface features such as handles). Since the flow vector with the maximum magnitude is often at one of these extreme points, we must choose an alternative grasp point. While contact selection for suction-based grasping is a well-studied problem [2, 30, 31], we find that a simple heuristic performs acceptably; we choose the point with the highest flow magnitude subject to the following constraints:

1. The point itself is not within a certain distance of an edge, where edges are computed using a standard edge-detection algorithm (see supplement for details).
2. The estimated Gaussian curvature of that point does not exceed a certain threshold (see supplement for details).
3. The point is not within a distance of d of any points violating conditions 1 and

2. In practice, we set $d = \texttt{2cm}$ (the radius of the suction tip).

Using this grasp selection method, we are able to execute our general articulation manipulation policy on a real robot. See the supplementary materials for other implementation details.

2.3.6 Training Details

We design a flow prediction network – which we refer to as ArtFlowNet – using the dense prediction configuration of PointNet++ [42] as a backbone, and train it using standard supervised learning with the Adam optimizer. We emphasize that we train a *single* model to predict 3DAF across all categories, using a dataset of synthetically-generated (observation, ground-truth flow) pairs based on the ground-truth kinematic and geometric structure provided by the PartNet-Mobility dataset [53]. During each step of training, we select an object in the dataset, randomize the state S of the object, and compute a new supervised pair (O_S, F_S), which we use to compute the loss and update the model parameters. During training, each object is seen in 100 different randomized configurations. Details of our dataset construction and model architecture can be found in the supplementary materials.

2.4 Results

We conduct a wide range of simulated and real-world experiments to evaluate the FlowBot3D system.

2.4.1 Simulation Results

To evaluate our method in simulation, we implement a suction gripper in the ManiSkill environment [34], which serves as a simulation interface for interacting with the PartNet-Mobility dataset [53]. The PartNet-Mobility dataset contains 46 categories of articulated objects; following UMPNet [55], we consider a subset of PartNet-Mobility containing 21 classes, split into 11 training categories (499 training objects, 128 testing objects) and 10 entirely unseen object categories (238 unseen objects). Several objects in the original dataset contain invalid meshes, which we exclude from

	Novel Instances in Train Categories													Test Categories									
	AVG.													AVG.									
Baselines																							
UMP-DI	0.29	**0.32**	0.33	0.16	0.18	0.37	0.14	0.19	**0.28**	0.72	**0.00**	0.55		0.36	0.32	0.62	0.15	0.00	0.41	0.61	0.17	0.34	0.38	0.58
Normal Direction	0.40	0.52	0.67	0.16	0.19	0.51	0.60	0.13	0.11	0.55	0.61	0.32		0.39	0.69	0.57	0.04	0.00	0.67	0.19	0.66	1.00	0.43	0.26
Screw Parameters	0.40	0.42	0.40	0.42	0.18	0.57	0.45	0.27	0.59	0.51	0.58	0.06		0.18	0.19	0.26	0.08	0.08	**0.23**	**0.06**	0.14	0.21	0.34	0.24
BC	0.74	0.59	0.91	0.63	0.75	0.57	1.00	1.00	0.98	0.62	0.96	0.10		0.87	0.81	0.63	1.00	0.93	0.99	0.74	0.95	0.96	0.83	0.88
DAgger E2E	0.64	0.39	0.85	0.61	0.73	0.50	1.00	0.96	0.90	0.54	0.48	0.10		0.83	0.73	0.62	1.00	0.80	0.95	0.73	0.83	0.98	0.81	0.85
DAgger Oracle	0.51	0.54	0.55	0.20	0.41	0.96	0.64	0.14	0.64	0.47	0.85	0.16		0.56	0.93	0.58	0.61	0.64	0.91	0.27	0.23	0.34	0.27	0.79
Baselines w/ Flow																							
BC + F	0.83	0.59	1.00	0.61	0.91	1.00	0.97	1.00	1.00	0.69	1.00	0.39		0.91	1.00	0.96	1.00	0.89	0.77	0.71	0.95	0.96	1.00	0.89
DAgger E2E + F	0.76	0.59	0.86	0.60	0.76	0.95	1.00	0.86	0.77	0.65	1.00	0.36		0.91	1.00	0.88	1.00	0.76	0.95	0.68	1.00	0.96	1.00	0.88
DAgger Oracle + F	0.50	0.59	0.53	0.25	0.51	0.58	0.86	0.17	0.65	0.56	0.48	0.38		0.60	0.77	0.71	0.62	0.73	0.91	0.28	0.43	0.47	0.31	0.73
Ours																							
FlowBot3D	**0.12**	0.32	0.23	0.11	0.09	**0.02**	**0.00**	**0.09**	0.32	**0.04**	0.13	**0.00**		**0.15**	**0.00**	0.21	**0.00**	**0.00**	0.33	0.22	0.09	0.07	**0.19**	0.35
FlowBot3D w/o Mask	0.17	0.32	0.37	0.10	0.11	0.15	0.00	0.11	0.33	0.05	0.07	0.29		0.19	0.16	0.24	0.05	0.00	0.24	0.24	0.17	0.27	0.19	0.37
FlowBot3D w/o Mask (+VPA)	0.16	0.33	**0.09**	**0.07**	**0.07**	0.16	0.00	0.14	0.49	0.27	0.11	0.00		0.16	0.11	**0.17**	0.23	0.00	0.53	0.10	**0.05**	**0.00**	0.23	**0.20**
Oracle w/ GT 3DAF	0.05	0.10	0.10	0.03	0.11	0.06	0.00	0.12	0.00	0.00	0.02	0.00		0.16	0.00	0.12	0.95	0.00	0.12	0.14	0.02	0.00	0.13	0.12

Table 2.1: Normalized Distance Metric Results (↓): Normalized distances to the target articulation joint angle after a full rollout across different methods. The lower the better.

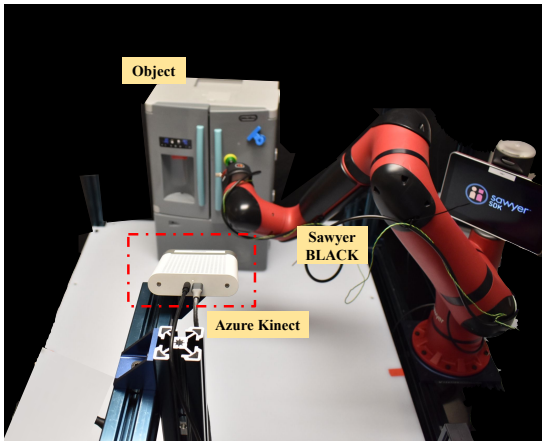


Figure 2.4: Workspace setup for physical experiments. The sensory signal comes from an Azure Kinect depth camera, and the agent is a Sawyer BLACK robot.

Figure 2.5: Fourteen test objects for our real-world experiments. Please refer to Supplementary Material for the exact category of each object.

evaluation. We modify ManiSkill simulation environment to accommodate these object categories. We train our models (ArtFlowNet and baselines) exclusively on the training instances of the training object categories, and evaluate by rolling out the corresponding policies for every object in the ManiSkill environment. Each object starts in the "closed" state (one end of its range of motion), and the goal is to actuate the joint to its "open" state (the other end of its range of motion). For experiments

	Novel Instances in Train Categories													Test Categories									
	AVG.													AVG.									
Baselines																							
UMP-DI	0.52	0.60	0.33	0.65	0.73	0.29	0.67	0.80	**0.50**	0.11	**1.00**	0.00		0.45	0.83	0.03	0.50	1.00	0.31	0.29	0.78	0.33	0.31	0.20
Normal Direction	0.31	0.40	0.00	0.51	0.71	0.00	0.00	0.80	0.50	0.00	0.00	0.50		0.31	0.00	0.00	0.50	1.00	0.00	0.55	0.00	0.00	0.10	0.64
Screw Parameters	0.50	0.50	0.53	0.51	0.80	0.21	0.55	0.60	0.17	0.37	0.43	0.80		0.67	0.17	0.63	0.67	0.92	**0.69**	**0.92**	0.83	0.75	0.50	0.72
BC	0.14	0.40	0.00	0.20	0.18	0.14	0.00	0.00	0.00	0.11	0.00	0.50		0.04	0.17	0.00	0.00	0.04	0.00	0.15	0.00	0.00	0.00	0.00
DAgger E2E	0.14	0.60	0.00	0.26	0.09	0.28	0.00	0.00	0.00	0.00	0.25	0.00		0.04	0.00	0.00	0.00	0.20	0.00	0.17	0.02	0.00	0.00	0.00
DAgger Oracle	0.29	0.40	0.33	0.80	0.27	0.00	0.00	0.80	0.00	0.11	0.00	0.50		0.20	0.00	0.00	0.00	0.36	0.00	0.29	0.49	0.33	0.31	0.20
Baselines w/ Flow																							
BC + F	0.11	0.40	0.00	0.26	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.50		0.03	0.00	0.00	0.00	0.08	0.00	0.15	0.05	0.00	0.00	0.00
DAgger E2E + F	0.14	0.40	0.00	0.00	0.09	0.00	0.00	0.00	0.50	0.00	0.00	0.50		0.04	0.00	0.00	0.00	0.20	0.00	0.15	0.05	0.00	0.00	0.00
DAgger Oracle + F	0.33	0.40	0.33	0.58	0.45	0.00	0.00	0.80	0.00	0.11	0.50	0.50		0.16	0.00	0.00	0.00	0.24	0.00	0.34	0.41	0.33	0.12	0.16
Ours																							
FlowBot3D	**0.77**	0.57	0.56	0.88	0.82	**0.86**	**1.00**	**0.80**	0.50	0.78	0.75	**1.00**		0.69	**1.00**	0.56	**1.00**	**1.00**	0.38	0.43	0.84	0.83	0.43	0.44
FlowBot3D w/o Mask	0.72	**0.67**	0.55	0.85	0.82	0.57	1.00	0.80	0.50	**0.89**	0.75	0.50		0.62	0.83	0.59	0.50	1.00	0.62	0.28	0.76	0.58	0.56	0.50
FlowBot3D w/o Mask + VPA	0.73	0.60	**0.67**	**0.88**	**0.91**	0.43	1.00	0.80	0.50	0.56	0.72	1.00		**0.70**	0.50	**0.63**	0.50	1.00	0.23	0.90	**0.88**	**1.00**	**0.63**	**0.72**
Oracle w/ GT 3DAF	0.92	0.80	1.00	0.97	0.82	0.71	1.00	0.80	1.00	1.00	1.00	1.00		0.82	1.00	0.85	0.00	1.00	0.85	0.86	0.98	1.00	0.81	0.88

Table 2.2: Success Rate Metric Results (↑): Fraction of success trials (normalized distance less than 0.1) of different objects' categories after a full rollout across different methods. The higher the better.

in simulation, we include in the observation O_t a binary part mask indicating which points belong to the child joint of interest. Results are shown in Tables A.2 and A.3[3].

Metrics. During our experiments, we calculate two metrics:

- *Normalized distance*: Following Xu et al. [55], we compute the normalized distance travelled by a specific child link through its range of motion. The metric is computed based on the final configuration after a policy rollout ($\mathbf{j}_{\text{end}}$) and the initial configuration ($\mathbf{j}_{\text{init}}$):

$$\mathcal{E}_{\text{goal}} = \frac{||\mathbf{j}_{\text{end}} - \mathbf{j}_{\text{goal}}||}{||\mathbf{j}_{\text{goal}} - \mathbf{j}_{\text{init}}||}$$

- *Success*: We also define a binary success metric, which is computed by thresholding the final resulting normalized distance at δ: Success $= \mathbb{1}(\mathcal{E}_{\text{goal}} \leq \delta)$. We set $\delta = 0.1$, meaning that we define a success as articulating a part for more than 90%.

Baseline Comparisons: We compare our proposed method with several baseline methods:

[3]Categories from left to right: stapler, trash can, storage furniture, window, toilet, laptop, kettle, switch, fridge, folding chair, microwave, bucket, safe, phone, pot, box, table, dishwasher, oven, washing machine, and door. Clipart pictures are borrowed from UMPNet paper with the authors' permission.

- **UMP-DI**: We implement a variant[4] of UMPNet's Direction Inference network (DistNet) [55], where instead of bootstrapping an action scoring function from interaction, we learn the scoring function by regressing the cosine distance between a query vector and the ideal flow vector for a contact point. At test time, we select the contact point based on ground-truth 3DAF, and after contact has been achieved we use CEM to optimize the scoring function to predict the action direction at every timestep.
- **Normal Direction**: We use off-the-shelf normal estimation to estimate the surface normals of the point cloud using Open3D [61]. To break symmetry, we align the normal direction vectors to the camera. At execution time, we first choose the ground-truth maximum-flow point and then follow the direction of the estimated normal vector of the surface.
- **Screw Parameters**: We predict the screw parameters for the selected joint of the articulated object. We then generate 3DAF from these predicted parameters and use the FlowBot3D policy on top of the generated flow.
- **Behavioral Cloning (BC)**: The agent takes as input a point cloud and outputs the action of the robot. The agent uses the PointNet-Transformer architecture proposed in [34]. The agent is trained end-to-end via L2 regression on trajectories provided by an oracle version of GT 3DAF.
- **BC + F**: Same as BC, but with ground-truth flow at input.
- **DAgger E2E**: We also conduct behavioral cloning experiments with DAgger [44] on the same expert dataset as in the BC baseline. We train it end-to-end (E2E), similar to the BC model above.
- **DAgger E2E + F**: Same as DAgger E2E, but with ground-truth flow as an input.
- **DAgger Oracle**: A two-step policy, where we first use ground-truth flow to select a contact point using the Generalized Articulation Policy heuristic, and train DAgger on expert trajectories generated after the point of contact.
- **DAgger Oracle + F**: Same as DAgger Oracle, but with ground-truth flow at

[4]We could not yet compare directly to UMPNet, as their model and simulation environment had not yet been released at the time between submission and publication.

input.

- **Oracle w/ GT 3DAF**: An oracle version of FlowBot3D that uses ground truth 3DAF vectors instead of the predicted ones for both phases. This serves as an upper bound of FlowBot 3D's performance.[5]

Each method above consists of a single model trained across all PartNet-Mobility training categories. For a more straightforward comparison, we dedicate Table A.2 and A.3 to evaluations in the SAPIEN simulator and we defer the comparison between UMPNet and FlowBot3D to the supplementary material.

Analysis: We can draw two conclusions from our simulated evaluation. First, our formulation of FlowBot3D has a very high success rate across all categories, including test categories, which are completely novel types of objects (but may contain similar parts and articulation structures). This is evidence that the ArtFlowNet network is learning salient geometric features to predict the location and character of articulated points. Based on visual interpretation of actual predicted flows, ArtFlowNet is particularly adept at recognizing doors, lids, drawers, and other large articulated features. One might have thought that 3DAF is essentially estimating normal directions, but this is not the case, as seen in the results of the Normal Direction baseline. Normal Direction estimation suffers from occlusion issues and the normal is not always the correct direction to actuate the object (for example, for the spherical-shaped lid of a teapot). Additionally, our method's accuracy increases when the object is at least partially open, because there is less ambiguity about object structure than when an object is fully "closed". The UMP-DI baseline exhibits similar properties, but the implicit optimization yields noisier direction predictions. Second, none of the Behavior Cloning and DAgger policies, nor their flow-based variants, perform well. The best BC baseline, DAgger Oracle + F, is only able to fully articulate objects 33% of the time.

UMPNet Pybullet Environment: The simulation environment used in the original UMPNet evaluations [55] is a PyBullet-based environment with different physical and collision parameters. However, the source code to run the UMPNet environment was not available for us to run until after this paper was submitted for review; we have since obtained a copy of this environment, and evaluate our method

[5]The description files of the phone meshes contain wrong rotation axis, thus the poor performance of the oracle policy on that category.

on their environment in the supplementary materials.

2.4.2 Real-World Experiments

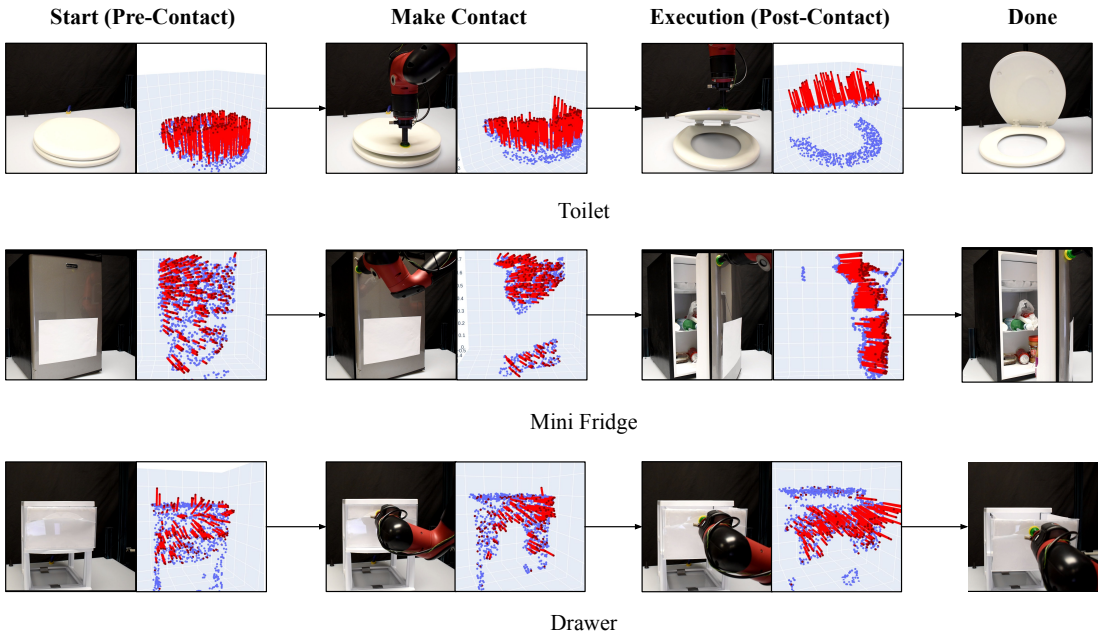


Figure 2.6: Real world examples of FlowBot3D executing an articulation policy based on predicting 3D Articulated Flow. Notice that even with occlusions, such as in the intermediate mini-fridge observation, the network is able to predict reasonable 3D articulation flow vectors for downstream policy.

To evaluate the performance of FlowBot3D when executed in a real robotic environment, we design a set of of real-world experiments in which we attempt to articulate a variety of different household objects using the Sawyer robot in our workspace, as shown in Fig. 2.4. Our experiment protocol is thus: for each object in the dataset, we conducted 5 trials of each method. For each trial, the object is placed in the scene at a random position such that the articulations are visible and the robot can reach every position in the range of motion of each articulation. The policy is then executed for at most 10 steps, terminating earlier if success has been achieved or if the policy predicts an action that cannot be executed safely (this case is infrequent). We conducted one round of evaluation (70 trials in total) for each of the following methods:

- **FlowBot3D**: The version of our generalized articulation policy as presented in Section 2.3. In experiments, we use an ArtFlowNet trained without a part mask in the observation space. In addition, since the camera position in reality is different from that in the ManiSkill environments, we apply a viewpoint augmentation (VPA) at training time, where we render synthetic point clouds from various camera angles in simulation.
- **DAgger Oracle**: The DAgger model trained in simulation to produce closed-loop motion directions, but with the contact selection predicted by the FlowBot3D model.

As in our simulated experiments, we use a single model trained in simulation across multiple object categories without any further finetuning.

											
# Objects	2	1	1	2	1	1	1	1	1	1	2

Table 2.3: Real-world objects used during our experiments.

Objects: We assemble a set of real-world objects that are representative of typical articulations a human may encounter in the real world: doors, drawers, hinges, etc. The objects were selected before experimentation began, and the only criteria for inclusion were 1) that it fit in the workspace, 2) it had a surface that a suction gripper could attach to and actuate, and 3) it wasn't too dark or reflective, so as to be seen by the Azure Kinect's depth camera. Each object falls into one of either the training or test classes we selected from the PartNet-Mobility. We also include several jars with lids, which, while not strictly articulated as 1-DoF joints, can be articulated like a prismatic joint. See Figure 2.5 and Table 2.3 for a summary of the dataset, and the supplementary materials for specifics for each object.

Metrics: During our trials, we compute the following metrics for each policy:

- Overall Success: Was the object articulated more than 90% of its range of motion (defined per-object)?
- Contact Success: Was the contact point chosen on an a joint that can move, and was the suction tip able to successfully form a seal at that point?

- Average Distance: Conditioned on a successful contact, what was the average distance from the end of the object's range of motion after the policy terminated?
- Motion Success: After successful contact, was the object articulated more than 90 % of its range of motion?

Details about how our trials are conducted and measurements computed can be found in the supplementary materials.

Method	Overall Succ.	Contact Succ.	Avg. Dist.	Motion Succ.
FlowBot3D	**45/70 (64.3%)**	64/70 (91.4%)*	**0.22**	**45/64 (70.3%)**
DAgger Oracle	10/70 (14.3%)	**68/70 (97.1%)***	0.73	10/68 (14.7%)

Table 2.4: Trials for FlowBot3D. *Note that both methods in the Contact Success column use the same FlowBot3D contact prediction and execution policy.

Quantitative analysis: We present summary metrics in Table 2.4, and a per-object summary in our supplementary materials. Across all metrics, FlowBot3D performs substantially better than the DAgger baseline. In absolute terms, the policy succeeds a high fraction of the time (64%); the policy selects a suitable contact point on the object 91% of the time, and succeeded 70% of the time after contact was established.

In contrast, the baseline policy succeeded in a very small number of cases, only 14% of the time. While contact rates were comparable to the trials conducted for FlowBot3D (they use the same contact selection method), the motions predicted were almost always unsuccessful.

Qualitative analysis of FlowBot3D: A major goal of our real-world trials was to evaluate how well the Flowbet3D policy transfers from simulation to reality without any retraining. We find that the overall policy performs surprisingly well, and the ArtFlowNet module – trained exclusively on point clouds rendered in simulation – generalizes impressively to real-world objects, producing high-fidelity flow predictions on a range of real objects. This is because there isn't much of a domain shift in the point cloud observations, and the geometric features that signal an articulation are fairly consistent.

Failure modes: We have found that the majority of trial failures were due to two reasons in real world: flow prediction error and contact failure. For flow prediction

errors, after making contact with the object, executing an incorrect 3D articulation flow vector will drive the gripper away from the object, causing the gripper to lose contact with the object. The bulk of flow prediction errors happen either because the robot occludes too much of the scene (which might be rectified by multiple viewpoints, temporal filtering, or a recurrent policy), or because the robot fails to detect the presence of articulations (this occurs on the real oven, for instance). For contact failures, the contact selection heuristic might not filter out all ungraspable points and thus the robot might choose a contact point that is difficult or impossible to make a complete seal on during suction. Overall, we theorize that many of the failures could be mitigated by improving the compliance and control of the gripper, and including a stronger contact quality prediction module.

2.4.3 Simulation Ablations

We conduct two ablations on the design decisions made for ArtFlowNet:

- **Including a part mask** We study the effect of providing a segmentation mask of the articulated part of interest as input to ArtFlowNet; such a mask could theoretically be obtained by a segmentation method or provided by a human to specify the articulation task, as in the SAPIEN challenge [53]. We find while the inclusion of a mask improves predictions in ambiguous cases (i.e. when a door is closed and coplanar with its parent link), removing the mask only decreases performance a small amount (see Tables A.2 and A.3).
- **Applying viewpoint augmentations during training**: We analyze how randomizing the camera viewpoint during the synthetic dataset generation step affects model performance. We find that augmenting the viewpoint has little effect on the performance in a simulated environment (see Tables A.2 and A.3), but improves performance in sim-to-real transfer.

2.5 Conclusion

In this work, we propose a novel visual representation for articulated objects, namely **3D Articulation Flow**, as well as a policy – **FlowBot3D** – which leverages this representation to successfully manipulate articulated objects. We demonstrate the

effectiveness of our method in both simulated and real environments, and observe strong sim-to-real transfer generalization.

While our method shows strong performance on a range of object classes, there is substantial room for improvement. One class of improvements is in system-building and engineering: with a more compliant robotic arm controller, as well as a more sophisticated contact prediction system, we believe we would be able to eliminate a wide class of failure modes. However, the remaining failure modes raise questions we would like to explore in future work. For instance, we would like to explore how our flow representation models might be used in an online adaptation setting, so that incorrect predictions can be corrected. We also would like to explore how our representation might be useful when learning from demonstrations, or in other more complex manipulation settings.

Lastly, FlowBot 3D might be a good 3D visual representation for articulated objects manipulation, is there a good dense, 3D visual representation that we could leverage for **free-floating** objects manipulation? Free-floating objects manipulation is simpler in that the objects are not constrained by parts, but representing their motion in a full 6-DoF $SE(3)$ space does pose new challenges. We explore one potential representation we could use for free-floating objects in Chapter 3.

Chapter 3

TAX-Pose: Task-Specific Cross-Pose Estimation for Robot Manipulation

Many manipulation tasks require a robot to move an object to a location relative to another object. For example, a cooking robot may need to place a lasagna in an oven, place a pot on a stove, place a plate in a microwave, place a mug onto a mug rack, or place a cup onto a shelf. Understanding and placing objects in task-specific locations is a key skill for robots operating in human environments. Further, this skill should generalize to novel objects within the training categories, such as placing new trays into the oven or new mugs onto a mug rack. A common approach in robot learning is to train a policy "end-to-end," mapping from pixel observations to low-level robot actions. However, end-to-end trained policies cannot easily reason about complex pose relationships such as the ones described above, and they have difficulty generalizing to unseen object configurations.

In contrast, we propose a method that learns to reason about the 3D geometric relationship between a pair of objects. For the type of tasks defined above, the robot needs to reason about the relationship between key parts on one object with respect to key parts on another object. For example, to place a mug on a mug rack, the robot must reason about the relationship between the pose of the mug handle and the pose of the mug rack; if the mug rack changes its pose, then the pose of the

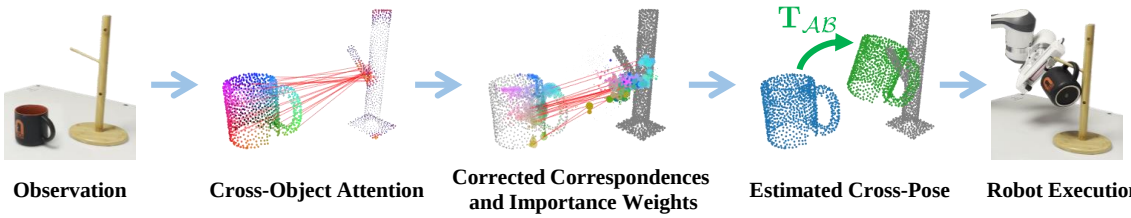


Figure 3.1: To solve a relative placement task, TAX-Pose uses cross-object attention to estimate dense cross-object correspondences and importance weights for each object point. This dense estimate is mapped to a single "cross-pose" which the robot uses to accomplish the given task.

mug must change accordingly in order to still be placed on the rack (see Figure 3.3). We name this task-specific notion of the pose relationship between a pair of objects as "cross-pose" and we formally define it mathematically. Further, we propose a vision system that can efficiently estimate the cross-pose from a small number of demonstrations of a given task, generalizing to novel objects within the training categories. To complete the manipulation task, we use the estimated cross-pose as the target of a motion planning algorithm, which will move the objects into the desired configuration (e.g. placing the mug onto the rack, placing the lasagna into the oven, etc).

In this paper [37], we present TAX-Pose (TAsk-specific Cross-Pose), a deep 3D vision-based method that learns to predict a task-specific pose relationship between a pair of objects from a set of demonstrations. Our cross-pose estimation system is provably translation equivariant and can generalize from a small number of real-world demonstrations (in some cases as few as 10) to new objects in unseen poses.

The contributions of this paper include:

1. A precise definition of "cross-pose," which defines a task-specific pose relationship between two objects.
2. A novel method that estimates soft-correspondences between two objects, from which the cross-pose between the objects can be estimated (see Figure 3.1); this method is provably translation equivariant and can learn from a small number of real-world demonstrations.
3. A robot system to manipulate objects into the desired cross-pose to achieve a given manipulation task.

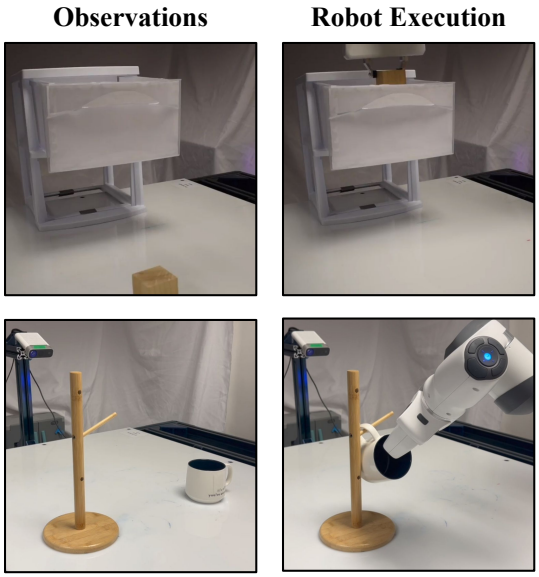


Figure 3.2: We study relative placement tasks, in which one object needs to be placed in a position relative to another object. Here are two of the tasks that we demonstrate our method on: **Top:** *PartNet-Mobility Placement Task* requires one object (e.g. a block) to be placed relative to another object (e.g. a drawer) by a semantic goal position (e.g. inside); **Bottom:** *Mug Hanging Task* requires placing the mug's handle on the mug rack.

We present simulated and real-world experiments to test the performance of our system in achieving a variety of relative placement manipulation tasks.

We demonstrate our method on a semantic placement task, in which the robot must place an object in, on, or around a novel object (Figure 3.2, top). We also demonstrate our method on precise placement tasks, such as hanging a mug on a rack (Figure 3.2, bottom) or placing a bottle or bowl on a shelf; in both cases our method generalizes to new object configurations and new objects within the training categories.

3.1 Problem Statement

3.1.1 Relative placement tasks:

In this paper, we are specifically interested in "relative placement tasks." Given two objects, $\mathcal{A}$ and $\mathcal{B}$, a "relative placement task" is the task of placing object $\mathcal{A}$ at a pose relative to object $\mathcal{B}$. For example, consider the task of placing a lasagna in

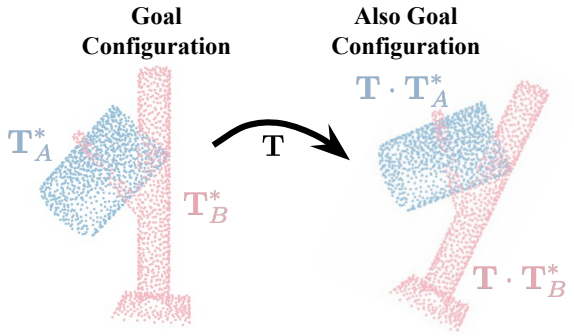


Figure 3.3: If we transform both the action object (mug) and the anchor object (rack) by the same transform, then the relative pose between these objects is unchanged (the mug is still "on" the rack) so the mug is still in the goal configuration.

an oven, placing a mug on a rack, or placing a robot gripper on the rim of a mug. All of these tasks involve placing one object (which we call the "action" object $\mathcal{A}$) at a semantically meaningful location relative to another object (which we call the "anchor" object $\mathcal{B}$)[1].

Specifically, suppose that $\mathbf{T}_{\mathcal{A}}^*$ and $\mathbf{T}_{\mathcal{B}}^*$ are $SE(3)$ poses for objects $\mathcal{A}$ and $\mathcal{B}$ respectively (in a shared world reference frame[2]) for which a desired task is considered complete (lasagna is in the oven; mug is on the rack, etc). Then for a relative placement task, if objects $\mathcal{A}$ and $\mathcal{B}$ are in poses $\mathbf{T} \cdot \mathbf{T}_{\mathcal{A}}^*$ and $\mathbf{T} \cdot \mathbf{T}_{\mathcal{B}}^*$ (respectively) for any transform $\mathbf{T}$, then the task will also be considered to be complete, as seen in Figure 3.3.

In other words, if $\mathbf{T}_{\mathcal{B}}^*$ represents the pose of the rack and $\mathbf{T}_{\mathcal{A}}^*$ represents the pose of the mug on the rack (at task completion); then if we transform the both the mug and rack poses by $\mathbf{T}$, then the mug will still be located on the rack. Formally, this property can be defined with the following Boolean function,

$$\text{RelPlace}(\mathbf{T}_{\mathcal{A}}, \mathbf{T}_{\mathcal{B}}) = \textbf{Success} \text{ iff } \exists \mathbf{T} \in SE(3) \text{ s.t. } \mathbf{T}_{\mathcal{A}} = \mathbf{T} \cdot \mathbf{T}_{\mathcal{A}}^* \text{ and } \mathbf{T}_{\mathcal{B}} = \mathbf{T} \cdot \mathbf{T}_{\mathcal{B}}^*. \tag{3.1}$$

For many real semantic placement tasks, there are actually sets of valid solutions which solve each task (i.e., there are many potential locations to place an object on a

[1]Note that the definition of action and anchor is symmetric; either object can be treated as the action object and the other as the anchor.

[2]All $SE(3)$ transformations in this work are defined in a fixed, arbitrary world frame.

table to achieve a semantic "object-on-table" relationship). However, for this work, we consider precise placement tasks under the simplifying assumption that, for a given pose of object $\mathcal{B}$, there is a single, unambiguous pose of object $\mathcal{A}$ needed to achieve the task.

3.1.2 Definition of Cross-Pose:

Given the above definition of a relative placement task, our goal will be to determine how to move object $\mathcal{A}$ so that it will be in the "goal pose," which, as described above, is defined relative to the pose of object $\mathcal{B}$. To achieve this, one option is to estimate the poses of objects $\mathcal{A}$ and $\mathcal{B}$ separately and then compute the transformation needed to move object $\mathcal{A}$ into the goal pose. However, the pose estimate of each object will have errors, and these errors will accumulate when the poses are combined into the single relative pose needed to reach the goal configuration.

Instead of estimating the pose of each object independently, we aim to learn a function $f(\mathbf{P}_\mathcal{A}, \mathbf{P}_\mathcal{B})$, which takes as input the point clouds $\mathbf{P}_\mathcal{A}$ and $\mathbf{P}_\mathcal{B}$ for both objects $\mathcal{A}$ and $\mathcal{B}$, where $\mathbf{P}_\mathcal{A} \in \mathbb{R}^{3\times N_\mathcal{A}}$ and $\mathbf{P}_\mathcal{B} \in \mathbb{R}^{3\times N_\mathcal{B}}$ are 3D point clouds of sizes $N_\mathcal{A}$ and $N_\mathcal{B}$, respectively. This function outputs an SE(3) rigid transformation, $f(\mathbf{P}_\mathcal{A}, \mathbf{P}_\mathcal{B}) = \mathbf{T}_{\mathcal{AB}}$, where we refer to $\mathbf{T}_{\mathcal{AB}}$ as the "cross-pose" between object $\mathcal{A}$ and object $\mathcal{B}$. For notational convenience, we occasional write f as a function of the poses $\mathbf{T}_\mathcal{A}$, $\mathbf{T}_\mathcal{B}$ of point clouds $\mathbf{P}_\mathcal{A}$ and $\mathbf{P}_\mathcal{B}$ respectively (with respect to a global reference frame) such that $f(\mathbf{T}_\mathcal{A}, \mathbf{T}_\mathcal{B}) := f(\mathbf{P}_\mathcal{A}, \mathbf{P}_\mathcal{B})$. This notational change is to make the transformation math more intuitive; in practice, this function only ever receives point clouds as input.

We will define the cross-pose $\mathbf{T}_{\mathcal{AB}}$ (below) such that, if we transform object $\mathcal{A}$ by $\mathbf{T}_{\mathcal{AB}}$, then object $\mathcal{A}$ will be in the goal pose relative to object $\mathcal{B}$ for the relative placement task. For example, suppose that $\mathbf{T}^*_\mathcal{A}$ and $\mathbf{T}^*_\mathcal{B}$ are poses for objects $\mathcal{A}$ and $\mathcal{B}$, respectively, for which a desired relative placement task is considered complete. In this configuration, the cross-pose of these objects would be $f(\mathbf{T}^*_\mathcal{A}, \mathbf{T}^*_\mathcal{B}) = \mathbf{I}$ where $\mathbf{I}$ is the identity, as object $\mathcal{A}$ does not need to be moved to complete the task. Further, based on the definition of a relative placement task given above, if both objects are transformed by the same transform $\mathbf{T}$, then the objects will still be in the desired

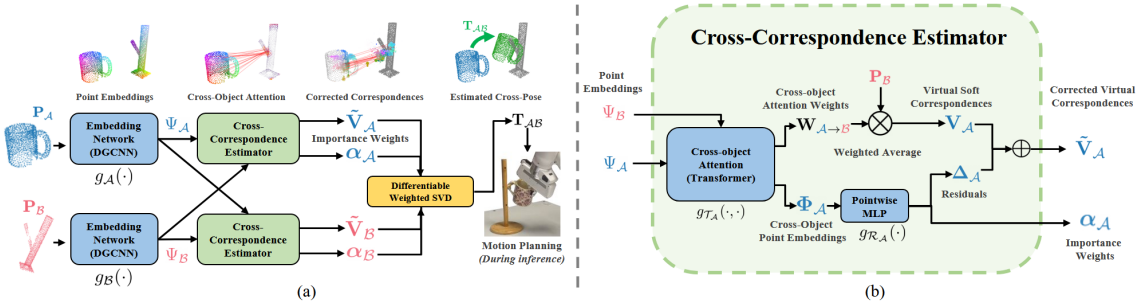


Figure 3.4: TAX-Pose Training Overview: Given a specific task, our method takes as input two point clouds and outputs the cross-pose between them needed to achieve the task. TAX-Pose first learns point clouds features using two DGCNN [40] networks and two Transformers [48]. Then the learned features are each input to a point residual network to predict per-point soft correspondences and weights across the two objects. The desired cross-pose can be inferred analytically from these correspondences using singular value decomposition.

relative pose,

$$f(\mathbf{T} \cdot \mathbf{T}^*_{\mathcal{A}}, \mathbf{T} \cdot \mathbf{T}^*_{\mathcal{B}}) = f(\mathbf{T}^*_{\mathcal{A}}, \mathbf{T}^*_{\mathcal{B}}) = \mathbf{I} \tag{3.2}$$

for any transform $\mathbf{T} \in SE(3)$. Now, let us assume that objects $\mathcal{A}$ and $\mathcal{B}$ are not in the goal configuration and have pose $\mathbf{T}_{\mathcal{A}} = \mathbf{T}_\alpha \cdot \mathbf{T}^*_{\mathcal{A}}$ and $\mathbf{T}_{\mathcal{B}} = \mathbf{T}_\beta \cdot \mathbf{T}^*_{\mathcal{B}}$, respectively, for arbitrary transforms $\mathbf{T}_\alpha$ and $\mathbf{T}_\beta \in SE(3)$. We then define the "cross-pose" of objects $\mathcal{A}$ and $\mathcal{B}$ as:

$$f(\mathbf{T}_{\mathcal{A}}, \mathbf{T}_{\mathcal{B}}) = f(\mathbf{T}_\alpha \cdot \mathbf{T}^*_{\mathcal{A}}, \mathbf{T}_\beta \cdot \mathbf{T}^*_{\mathcal{B}}) = \mathbf{T}_{\mathcal{AB}} := \mathbf{T}_\beta \cdot \mathbf{T}_\alpha^{-1}. \tag{3.3}$$

Note that this definition is equivalent to Equation 3.2 for the special case of $\mathbf{T}_\alpha = \mathbf{T}_\beta$. This definition of cross-pose allows us to move object $\mathcal{A}$ into the goal configuration, relative to object $\mathcal{B}$:

$$\mathbf{T}_{\mathcal{AB}} \cdot \mathbf{T}_{\mathcal{A}} = (\mathbf{T}_\beta \cdot \mathbf{T}_\alpha^{-1}) \cdot (\mathbf{T}_\alpha \cdot \mathbf{T}^*_{\mathcal{A}}) = \mathbf{T}_\beta \cdot \mathbf{T}^*_{\mathcal{A}}, \tag{3.4}$$

satisfying the relative placement condition defined in Equation 3.1 with $\mathbf{T} = \mathbf{T}_\beta$. Alternatively, we could have instead transformed object $\mathcal{B}$ by the inverse of the cross-pose to achieve the task.

3.2 Method

3.2.1 Overview

We frame the task of cross-pose estimation as a soft correspondence-prediction task between a pair of point clouds, followed by an analytical least-squares optimization to find the optimal *cross-pose* for the predicted correspondences. As described in Appendix B.2, this correspondence-based approach allows our method to be translation-equivariant: translating either object ($\mathcal{A}$ or $\mathcal{B}$) will lead to a translated cross-pose prediction. This allows our method to automatically adapt to novel positions of both the anchor and action objects, unlike previous work which assumes a static anchor [45]. Our method for task-specific cross-pose estimation, known as TAX-Pose, consists of the following steps, as shown in Figure 3.4:

1. **Soft Correspondence Prediction**: For a pair of objects $\mathcal{A}, \mathcal{B}$, a neural network learns to predict a per-point embedding to establish a (soft) correspondence between $\mathcal{A}$ and $\mathcal{B}$, which are called "virtual soft correspondences." The corresponding points are constrained to be within the convex hulls of $\mathcal{B}$ and $\mathcal{A}$ respectively.
2. **Adjustment via Correspondence Residuals**: For most estimation tasks, some points in object $\mathcal{A}$ may not be within the convex hull of object $\mathcal{B}$; for instance, when a mug is placed on a mug rack, most points on the mug will be outside of the convex hull of the mug rack. To accommodate these cases, we apply a pointwise residual vector to displace each of the predicted soft correspondences. These "corrected virtual correspondences" allow points in $\mathcal{A}$ to correspond to locations in free space near $\mathcal{B}$.
3. **Find the Optimal Transform**: Because the cross-pose is defined as a rigid transformation of object $\mathcal{A}$, we use a differentiable weighted SVD to find the transformation that minimizes the weighted least squares difference to the corrected virtual correspondences.

Because each step above is differentiable, the whole model can be optimized end-to-end, despite having an interpretable internal structure which we describe below. Our method is heavily inspired by Deep Closest Point (DCP) [50]. The

key difference between our pose alignment model and DCP is that we predict the cross-pose between two *different* objects for a given task instead of registering two point clouds of an identical object. Additionally, TAX-Pose can predict relationships where these clouds may not have any points of contact or overlap.

We now describe our cross-pose estimation algorithm in detail. To recap the problem statement, given objects $\mathcal{A}$ and $\mathcal{B}$ with point cloud observations $\mathbf{P}_\mathcal{A} \in \mathbb{R}^{3\times N_\mathcal{A}}$, $\mathbf{P}_\mathcal{B} \in \mathbb{R}^{3\times N_\mathcal{B}}$ respectively, our objective is to estimate the task-specific cross-pose $\mathbf{T}_{\mathcal{AB}} = f(\mathbf{P}_\mathcal{A}, \mathbf{P}_\mathcal{B}) \in SE(3)$. Note that the cross-pose between object $\mathcal{A}$ and $\mathcal{B}$ is defined with respect to a given task (e.g. putting a lasagna in the oven, putting a mug on the rack, etc).

3.2.2 Cross-Pose Estimation via Soft Correspondence Prediction

Soft Correspondence Prediction

The first step of the method is to compute two sets of correspondences between $\mathcal{A}$ and $\mathcal{B}$, one which maps from points in $\mathcal{A}$ to $\mathcal{B}$, and one which maps from points in $\mathcal{B}$ to $\mathcal{A}$. These need not be a bijection, and can be asymmetric. As we want each step to be differentiable, we follow DCP's conventions and estimate a *soft correspondence*. This assigns a *virtual soft corresponding point* $\mathbf{v}_i^\mathcal{A} \in \mathbf{V}_\mathcal{A}$ to every point $\mathbf{p}_i^\mathcal{A} \in \mathbf{P}_\mathcal{A}$ by computing a convex combination of points in $\mathbf{P}_\mathcal{B}$, and vice versa. Formally:

$$\mathbf{v}_i^\mathcal{A} = \mathbf{P}_\mathcal{B}\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} \quad \text{s.t.} \quad \sum_{j=1}^{N_\mathcal{B}} w_{ij}^{\mathcal{A}\to\mathcal{B}} = 1 \tag{3.5}$$

$$\mathbf{v}_i^\mathcal{B} = \mathbf{P}_\mathcal{A}\mathbf{w}_i^{\mathcal{B}\to\mathcal{A}} \quad \text{s.t.} \quad \sum_{j=1}^{N_\mathcal{A}} w_{ij}^{\mathcal{B}\to\mathcal{A}} = 1 \tag{3.6}$$

with normalized weight vectors $\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} \in \mathbf{W}_{\mathcal{A}\to\mathcal{B}}$ and $\mathbf{w}_i^{\mathcal{B}\to\mathcal{A}} \in \mathbf{W}_{\mathcal{B}\to\mathcal{A}}$. Importantly, these virtual corresponding points are not constrained to the surfaces of $\mathcal{A}$ or $\mathcal{B}$; instead, they are constrained to the convex hulls of $\mathbf{P}_\mathcal{B}$ and $\mathbf{P}_\mathcal{A}$, respectively.

To compute the weights $\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}}$, $\mathbf{w}_i^{\mathcal{B}\to\mathcal{A}}$ in Equations 3.2.2a and 3.2.2b, we first encode each point cloud $\mathbf{P}_\mathcal{A}$ and $\mathbf{P}_\mathcal{B}$ into a latent space using a neural network encoder, DGCNN [40]. This encoder head is comprised of two distinct encoders $g_\mathcal{A}$

and $g_{\mathcal{B}}$, each of which receives point cloud $\mathbf{P}_{\mathcal{A}}$ and $\mathbf{P}_{\mathcal{B}}$, respectively, zero-centers them, and outputs a dense, point-wise embedding for each object (see Figure 3.4): $\mathbf{\Psi}_{\mathcal{A}} = g_{\mathcal{A}}(\bar{\mathbf{P}}_{\mathcal{A}}) \in \mathbb{R}^{N_{\mathcal{A}} \times d}$, $\mathbf{\Psi}_{\mathcal{B}} = g_{\mathcal{B}}(\bar{\mathbf{P}}_{\mathcal{B}}) \in \mathbb{R}^{N_{\mathcal{B}} \times d}$ where $\boldsymbol{\psi}_i^{\mathcal{K}} \in \mathbf{\Psi}_{\mathcal{K}}$ is the d-dimensional embedding of the i-th point in object $\mathcal{K}$, and $\bar{\mathbf{P}}_{\mathcal{K}}$ is the zero-centered point cloud for object $\mathcal{K}$. Because we want the cross-correspondence to incorporate information about both point clouds, we then employ a cross-object attention module between the two dense feature sets to obtain *cross-object point embeddings*, $\mathbf{\Phi}_{\mathcal{A}} \in \mathbb{R}^{N_{\mathcal{A}} \times d}$ and $\mathbf{\Phi}_{\mathcal{B}} \in \mathbb{R}^{N_{\mathcal{B}} \times d}$, defined as:

$$\mathbf{\Phi}_{\mathcal{A}} = \mathbf{\Psi}_{\mathcal{A}} + g_{\mathcal{T}_{\mathcal{A}}}(\mathbf{\Psi}_{\mathcal{A}}, \mathbf{\Psi}_{\mathcal{B}}), \quad \mathbf{\Phi}_{\mathcal{B}} = \mathbf{\Psi}_{\mathcal{B}} + g_{\mathcal{T}_{\mathcal{B}}}(\mathbf{\Psi}_{\mathcal{B}}, \mathbf{\Psi}_{\mathcal{A}}) \tag{3.7}$$

where $g_{\mathcal{T}_{\mathcal{A}}}$, $g_{\mathcal{T}_{\mathcal{B}}}$ are Transformers [48].

Finally, recall that our goal was to compute a set of normalized weight vectors $\mathbf{W}_{\mathcal{A}\to\mathcal{B}}$, $\mathbf{W}_{\mathcal{B}\to\mathcal{A}}$. To compute the virtual corresponding point $\mathbf{v}_i^{\mathcal{A}}$ assigned to any point $\mathbf{p}_i^{\mathcal{A}} \in \mathbf{P}^{\mathcal{A}}$, we can extract the desired normalized weight vector $\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}}$ from intermediate attention features of the cross-object attention module as:

$$\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} = \text{softmax}\left(\frac{\mathbf{K}_{\mathcal{B}}\mathbf{q}_i^{\mathcal{A}}}{\sqrt{d}}\right), \quad \mathbf{w}_i^{\mathcal{B}\to\mathcal{A}} = \text{softmax}\left(\frac{\mathbf{K}_{\mathcal{A}}\mathbf{q}_i^{\mathcal{B}}}{\sqrt{d}}\right) \tag{3.8}$$

where $\mathbf{q}_i^{\mathcal{K}} \in \mathbf{Q}_{\mathcal{K}}$, and $\mathbf{Q}_{\mathcal{K}}, \mathbf{K}_{\mathcal{K}} \in \mathbb{R}^{\mathbf{N}_{\mathcal{K}} \times d}$ are the query and key values (respectively) for object $\mathcal{K}$ associated with cross-object attention Transformer module $g_{\mathcal{T}_{\mathcal{K}}}$ (see Appendix B.3 for details). These weights are then used to compute the virtual soft correspondences $\mathbf{V}_{\mathcal{A}}$, $\mathbf{V}_{\mathcal{B}}$ using Equation 3.2.2.

Adjustment via Correspondence Residuals

As previously stated, the virtual soft correspondences $\mathbf{V}_{\mathcal{A}}, \mathbf{V}_{\mathcal{B}}$ given by Equations 3.2.2a and 3.2.2b are constrained to be within the convex hull of each object. However, many relative placement tasks cannot be solved perfectly with this constraint. For instance, we might want a point on the handle of a teapot to correspond to some point above a stovetop (which lies outside the convex hull of the points on the stovetop). To allow for such off-object correspondences, we further learn a *residual vector*, $\boldsymbol{\delta}_i^{\mathcal{A}} \in \mathbf{\Delta}_{\mathcal{A}}$ for each point i that corrects each virtual corresponding point $\mathbf{v}_i^{\mathcal{A}}$. This allows us to displace each virtual corresponding point to any arbitrary location that might be suitable for the task. To compute these residual vectors, we

use a point-wise neural network $g_{\mathcal{R}_\mathcal{A}}$, $g_{\mathcal{R}_\mathcal{B}}$ to map each point's embedding into a 3D residual vector:

$$\boldsymbol{\delta}_i^{\mathcal{A}} = g_{\mathcal{R}_\mathcal{A}}\left(\boldsymbol{\phi}_i^{\mathcal{A}}\right) \in \mathbb{R}^3, \quad \boldsymbol{\delta}_i^{\mathcal{B}} = g_{\mathcal{R}_\mathcal{B}}\left(\boldsymbol{\phi}_i^{\mathcal{B}}\right) \in \mathbb{R}^3$$

Applying these residual offsets to the virtual points, we get a set of *corrected virtual correspondences*, $\tilde{\mathbf{v}}_i^{\mathcal{A}} \in \tilde{\mathbf{V}}_\mathcal{A}$ and $\tilde{\mathbf{v}}_i^{\mathcal{B}} \in \tilde{\mathbf{V}}_\mathcal{B}$, defined as

$$\tilde{\mathbf{v}}_i^{\mathcal{A}} = \mathbf{v}_i^{\mathcal{A}} + \boldsymbol{\delta}_i^{\mathcal{A}}, \quad \tilde{\mathbf{v}}_i^{\mathcal{B}} = \mathbf{v}_i^{\mathcal{B}} + \boldsymbol{\delta}_i^{\mathcal{B}} \tag{3.9}$$

These corrected virtual correspondences $\tilde{\mathbf{v}}_i^{\mathcal{A}}$ define the estimated goal location relative to object $\mathcal{B}$ for each point $\mathbf{p}_i \in \mathbf{P}_\mathcal{A}$ of object $\mathcal{A}$, and likewise for each point in object $\mathcal{B}$ (see visualization in Appendix B.1.1).

Least-Squares Cross-Pose Optimization with Weighted SVD

Given the sets of dense correspondences, $\left(\mathbf{P}_\mathcal{A}, \tilde{\mathbf{V}}_\mathcal{A}\right)$ and $\left(\mathbf{P}_\mathcal{B}, \tilde{\mathbf{V}}_\mathcal{B}\right)$, we would like to compute a single rigid transformation for object $\mathcal{A}$. To do so, we solve for the transformation $\mathbf{T}_{\mathcal{AB}}$ (the cross-pose) that minimizes the weighted distance between each point and its corrected virtual correspondence. Formally, this leads to the following weighted least squares optimization:

$$\mathcal{J}(\mathbf{T}_{\mathcal{AB}}) = \sum_{i=1}^{N_\mathcal{A}} \alpha_i^{\mathcal{A}} ||\mathbf{T}_{\mathcal{AB}}\ \mathbf{p}_i^{\mathcal{A}} - \tilde{\mathbf{v}}_i^{\mathcal{A}}||_2^2 + \sum_{i=1}^{N_\mathcal{B}} \alpha_i^{\mathcal{B}} ||\mathbf{T}_{\mathcal{AB}}^{-1}\ \mathbf{p}_i^{\mathcal{B}} - \tilde{\mathbf{v}}_i^{\mathcal{B}}||_2^2 \tag{3.10}$$

where the weights $\alpha_i^{\mathcal{A}} \in \boldsymbol{\alpha}_\mathcal{A}$, $\alpha_i^{\mathcal{B}} \in \boldsymbol{\alpha}_\mathcal{B}$ signify the importance of each correspondence and are predicted by a point-wise MLP as shown in Figure 3.4. These weights are learned end-to-end as parameters of our network; they are visualized in Appendix B.1.2, which shows that the network has learned to assign more weight to the parts of the object that are most important for the task, such as the region around the mug handle (on the mug) and the region around the peg (on the rack). Equation 3.10 is the well-known weighted Procrustes problem, for which there exists an analytical solution. To maintain the differentiablity of the system, we use a weighted differentiable SVD operation [39] to compute the cross-pose $\mathbf{T}_{\mathcal{AB}}$ that minimizes this objective (see Appendix B.4 for details). This allows us to train the system end-to-end as described below.

3.2.3 TAX-Pose Training Pipeline

To train our model, we use a segmented set of demonstration point clouds of a pair of objects in the goal configuration. For each demonstration point cloud, we generate multiple training examples by transforming each object's point cloud, $\mathbf{P}_{\mathcal{A}}$ and $\mathbf{P}_{\mathcal{B}}$ by random SE(3) transformations $\mathbf{T}_{\alpha}$ and $\mathbf{T}_{\beta}$, respectively. The predicted cross-pose, $\mathbf{T}_{\mathcal{AB}}$, is then compared with the ground truth cross-pose, $\mathbf{T}_{\mathcal{AB}}^{GT} := \mathbf{T}_{\beta}\mathbf{T}_{\alpha}^{-1}$, using an average distance loss [19] with dense regularization.

To train the encoders $g_{\mathcal{A}}(\bar{\mathbf{P}}_{\mathcal{A}})$, $g_{\mathcal{B}}(\bar{\mathbf{P}}_{\mathcal{B}})$ as well as the residual networks $g_{\mathcal{R}_{\mathcal{A}}}\left(\boldsymbol{\phi}_i^{\mathcal{A}}\right)$, $g_{\mathcal{R}_{\mathcal{B}}}\left(\boldsymbol{\phi}_i^{\mathcal{B}}\right)$, we use a set of losses defined below. We assume we have access to a set of demonstrations of the task, in which the action and anchor objects are in the target relative pose such that $\mathbf{T}_{\mathcal{AB}} = \mathbf{I}$.

Point Displacement Loss: Instead of directly supervising the rotation and translation (as is done in DCP), we supervise the predicted transformation using its effect on the points. For this loss, we take the point clouds of the objects in the demonstration configuration, and transform each cloud by a random transform, $\hat{\mathbf{P}}_{\mathcal{A}} = \mathbf{T}_{\alpha}\mathbf{P}_{\mathcal{A}}$, and $\hat{\mathbf{P}}_{\mathcal{B}} = \mathbf{T}_{\beta}\mathbf{P}_{\mathcal{B}}$. This would give us a ground truth transform of $\mathbf{T}_{\mathcal{AB}}^{GT} = \mathbf{T}_{\beta}\mathbf{T}_{\alpha}^{-1}$; the inverse of this transform would move object $\mathcal{B}$ to the correct position relative to object $\mathcal{A}$. Using this ground truth transform, we compute the MSE loss between the correctly transformed points and the points transformed using our prediction.

$$\mathcal{L}_{\text{disp}} = \left\|\mathbf{T}_{\mathcal{AB}}\mathbf{P}_{\mathcal{A}} - \mathbf{T}_{\mathcal{AB}}^{GT}\mathbf{P}_{\mathcal{A}}\right\|^2 + \left\|\mathbf{T}_{\mathcal{AB}}^{-1}\mathbf{P}_{\mathcal{B}} - \mathbf{T}_{\mathcal{AB}}^{GT-1}\mathbf{P}_{\mathcal{B}}\right\|^2 \tag{3.11}$$

Direct Correspondence Loss. While the Point Displacement Loss best describes errors seen at inference time, it can lead to correspondences that are inaccurate but whose errors average to the correct pose. To improve these errors we directly supervise the learned correspondences $\tilde{V}_{\mathcal{A}}$ and $\tilde{V}_{\mathcal{B}}$:

$$\mathcal{L}_{\text{corr}} = \left\|\tilde{\mathbf{V}}_{\mathcal{A}} - \mathbf{T}_{\mathcal{AB}}^{GT}\mathbf{P}_{\mathcal{A}}\right\|^2 + \left\|\tilde{\mathbf{V}}_{\mathcal{B}} - \mathbf{T}_{\mathcal{AB}}^{GT-1}\mathbf{P}_{\mathcal{B}}\right\|^2. \tag{3.12}$$

Correspondence Consistency Loss. Furthermore, a consistency loss can be used. This loss penalizes correspondences that deviate from the final predicted transform. A benefit of this loss is that it can help the network learn to respect the rigidity of

the object, while it is still learning to accurately place the object. Note, that this is similar to the Direct Correspondence Loss, but uses the predicted transform as opposed to the ground truth one. As such, this loss requires no ground truth:

$$\mathcal{L}_{\text{cons}} = \left\| \tilde{\mathbf{V}}_{\mathcal{A}} - \mathbf{T}_{\mathcal{AB}} \mathbf{P}_{\mathcal{A}} \right\|^2 + \left\| \tilde{\mathbf{V}}_{\mathcal{B}} - \mathbf{T}_{\mathcal{AB}}^{-1} \mathbf{P}_{\mathcal{B}} \right\|^2. \tag{3.13}$$

Overall Training Procedure. We train with a combined loss $\mathcal{L}_{\text{net}} = \mathcal{L}_{\text{disp}} + \lambda_1 \mathcal{L}_{\text{corr}} + \lambda_2 \mathcal{L}_{\text{cons}}$, where λ_1 and λ_2 are hyperparameters. We use a similar network architecture as DCP [50], which consists of DGCNN [51] and a Transformer [48].

In order to quickly adapt to new tasks, we optionally pre-train the DGCNN embedding networks over a large set of individual objects using the InfoNCE loss [36] with a geometric distance weighting and random transformations, to learn $SE(3)$ invariant embeddings, see Appendix E.2 for details.

3.2.4 Pretraining

We utilize pretraining for the embedding network for the mug hanging task, and describe the details below.

We pretrain embedding network for each object category (mug, rack, gripper), such that the embedding network is $SE(3)$ *invariant* with respect to the point clouds of that specific object category. Specifically, the mug-specific embedding network is pretrained on 200 ShapeNet [6] mug instances, while the rack-specific and gripper-specific embedding network is trained on the same rack and Franka gripper used at test time, respectively. Note that before our pretraining, the network is randomly initialized with the Kaiming initialization scheme [18]; we don't adopt any third-party pretrained models.

For the network to be trained to be $SE(3)$ invariant, we pre-train with InfoNCE loss [36] with a geometric distance weighting and random $SE(3)$ transformations. Specifically, given a point cloud of an object instance, $\mathbf{P}_{\mathcal{A}}$, of a specific object category $\mathcal{A}$, and an embedding network $g_{\mathcal{A}}$, we define the point-wise embedding for $\mathbf{P}_{\mathcal{A}}$ as $\Phi_{\mathcal{A}} = g_{\mathcal{A}}(\mathbf{P}_{\mathcal{A}})$, where $\phi_i^{\mathcal{A}} \in \Phi_{\mathcal{A}}$ is a d-dimensional vector for each point $p_i^{\mathcal{A}} \in \mathbf{P}_{\mathcal{A}}$. Given a random $SE(3)$ transformation, $\mathbf{T}$, we define $\Psi_{\mathcal{A}} = g_{\mathcal{A}}(\mathbf{T}\mathbf{P}_{\mathcal{A}})$, where $\psi_i^{\mathcal{A}} \in \Psi_{\mathcal{A}}$ is the d-dimensional vector for the ith point $p_i^{\mathcal{A}} \in \mathbf{P}_{\mathcal{A}}$.

The weighted contrastive loss used for pretraining, $\mathcal{L}_{wc}$, is defined as

$$\mathcal{L}_{wc} := -\sum_i \log \left[\frac{\exp\left(\phi_i^\top \psi_i\right)}{\sum_j \exp\left(d_{ij}\left(\phi_i^\top \psi_j\right)\right)} \right] \tag{3.14}$$

$$d_{ij} := \begin{cases} \frac{1}{\mu}\tanh\left(\lambda \|p_i^{\mathcal{A}} - p_j^{\mathcal{A}}\|_2\right), & \text{if } i \neq j \\ 1, & \text{otherwise} \end{cases} \tag{3.15}$$

$$\mu := \max\left(\tanh\left(\lambda \|p_i^{\mathcal{A}} - p_j^{\mathcal{A}}\|_2\right)\right) \tag{3.16}$$

For this pretraining, we use $\lambda := 10$.

3.3 Experiments

To evaluate TAX-Pose, we conduct a wide range of simulated and real-world experiments on two classes of relative placement tasks: NDF [45] Tasks and PartNet-Mobility Placement Tasks. All tasks involve placing an "action" object at a specific location relative to an anchor object, in which the relative pose is specified by a set of demonstrations. Our method then generalizes to perform this task on novel objects in unseen configurations. We refer the reader to our project website for additional results and videos.

3.3.1 NDF Tasks

We evaluate our method on all three NDF [45] tasks (*mug* hanging, *bottle* placement, and *bowl* placement); see Appendix B.6.1 for results on *bottle* and *bowl* placement. Results on *mug* hanging are described in more detail below.

Simulation Experiments

For our simulation experiments, we perform the task of hanging a mug on a rack as two sequential cross-pose estimation steps: grasping the mug (estimating the cross-pose between the gripper and the mug) and hanging the mug on the rack (estimating the cross-pose between the mug and the rack). In Pybullet [10], we simulate a Franka Panda above a table with 4 depth cameras placed on the corners of the table. The model is trained on 10 simulated demonstrations of mug hanging.

We evaluate task execution success on unseen mug instances in randomly generated initial configurations.

We measure task success rates of 1) *Grasping*, where success is achieved when the object is grasped stably; 2) *Placing*, where success is achieved when the mug is placed stably on the rack; 3) *Overall*, when the predicted transforms enable both grasp and place success in sequence. We compare our method to Neural Descriptor Field (NDF) [45] and Dense Object Nets (DON) [16].

Dense Object Nets (DON) [16]: Using manually labeled semantic keypoints on the demonstration clouds, DON is used to compute sparse correspondences with the test objects. These correspondences are converted to a pose using SVD. A full description can be found in [45].

Neural Descriptor Field (NDF) [45]: Using the learned descriptor field for the mug, the positions of a constellation of task specific quarry points are optimized to best match the demonstration using gradient descent.

Simulation Results

We evaluate our method in simulation in 100 trials consisting of unseen *mug* instances in random initial and goal configurations for both **Upright** and **Arbitrary** poses. As shown in Table 1, our method significantly outperforms the baselines for simulated mug hanging. We report additional results for simulated *bottle* and *bowl* placement tasks in Table B.4 in Appendix B.6.1.

Ablation Analysis

Effects of Number of Demonstrations. To study how the number of demonstrations observed affects our method's performance, we train our model on {10, 5, 1} demonstrations of upright pose mug hanging. Results are found in Table 2. Our method outperforms the baselines for all number of demonstrations; TAX-Pose can perform well even with as few as 5 demonstrations.

Cross-Pose Estimation Design Choices. We analyze the effects of design choices made in our Cross-Pose estimation algorithm for the upright pose mug hanging task. Specifically, we analyze the effects of 1) computing residual correspondence; 2) the use of *weighted* SVD over non-weighted in computing cross-pose; 3) using a transformer as the cross-object attention, as opposed to simpler model such as a 3-layer MLP. Table 3 shows that each major component of our system is important for task success. See more ablation experiments in Appendix B.6.1.

	Grasp	Place	Overall	Grasp	Place	Overall
	Upright Pose			Arbitrary Pose		
DON	0.91	0.50	0.45	0.35	0.45	0.17
NDF	0.96	0.92	0.88	**0.78**	0.75	0.58
TAX-Pose	**0.99**	**0.97**	**0.96**	0.75	**0.84**	**0.63**

Table 3.1: Mug on rack simulation success rate (↑)

Model	# Demos Used		
	1	5	10
DON	0.32	0.36	0.45
NDF	0.46	0.70	0.88
TAX-Pose	**0.77**	**0.90**	**0.96**

Table 3.2: # Demos vs. Overall success rate (↑)

Ablation	Grasp	Place	Overall
No Res.	0.97	0.96	0.93
Unw. SVD	0.92	0.94	0.88
No Attn.	0.90	0.82	0.76
TAX-Pose	**0.99**	**0.97**	**0.96**

Table 3.3: Mug hanging ablations success rate (↑)

Figure 3.5: Real-world experiments summary. **Left:** In object placement task, we train using simulated demonstrations and test on real-world objects. **Right:** Mug Hanging real-world experiments. We train from just 10 demonstrations from 10 training mugs in the real world and test on 10 unseen test mugs.

Real-World Experiments

We explore the hanging component of the mug on a rack task in a real world environment, which requires estimating the cross-pose between the mug and the rack. We train TAX-Pose using real demonstrations of 10 different mugs hung on a rack (1 demonstration each, for a total of only 10 real-world demonstrations for training). A motion primitive is used to grasp each mug, after which the robot plans a trajectory to apply the predicted cross-pose to the grasped mug. We evaluate the model on the 10 training mugs in novel poses, as well as on 10 unseen mugs (see Figure 3.5). For each of the 20 mugs, we conduct 5 trials, varying the mug's and rack's starting poses in each trial. Success is recorded if a peg penetrates the mug handle at the end of the trial. Our method achieves a success rate of 62% on training mugs in novel poses and 54% on unseen mugs. A visualization of the results can be seen in Figure 3.5 (right) and on the project website. Note that our method is able to perform the mug hanging task while varying the pose of the mug rack (see our project website), whereas the baselines (NDF [45], DON [16]) cannot because they assume a fixed, known rack position (see NDF [45] for baseline details).

3.3.2 PartNet-Mobility Placement Tasks

Task Description

We also define a PartNet-Mobility Placement task as placing a given action object relative to an anchor object based on a semantic goal position. We select a set of household furniture objects from the PartNet-Mobility dataset [53] as the anchor objects, and a set of small rigid objects released with the Ravens simulation environment [58] as the action objects. For each anchor object, we define a set of semantic goal positions (i.e. 'top', 'left', 'right', 'in'), where action objects should be placed relative to each anchor. Each semantic goal position defines a unique task in our cross-pose prediction framework. Given a synthetic point cloud observation of both objects, the task is to predict a cross-pose that places the object at the specific semantic goal. We evaluate both a *goal-conditioned* variant (**TAX-Pose GC**), which is trained across all goals, and a *task-specific* variant (**TAX-Pose**) of our model, which trains a separate model per goal type (see Appendix B.6.2 for details). In both cases we train only 1 model across all action and anchor objects. All models

are trained entirely on simulated data and transfer directly to real-world with no finetuning. Further task details can be found in the Appendix B.7.2.

Baselines

We compare our method to a variety of end-to-end imitation-learning-based methods trained from a motion planner expert in simulation.

- *E2E Behavioral Cloning:* Generate motion-planned trajectories using OMPL that take the action object from start to goal. These serve as "expert" trajectories for Behavioral Cloning (BC), where we train a neural network to output a policy that, at each time step, outputs an incremental 6-DOF transformation that imitates the expert trajectory.
- *E2E DAgger* : Using the same BC dataset as above, we train a policy using DAgger [44].
- *Trajectory Flow*: Using the same BC dataset with DAgger, we train a policy to predict a dense per-point 3D flow vector at each time step instead of a single incremental 6-DOF transformation. Given this dense per-point flow, we can extract a rigid transformation using SVD yielding the next pose [15].
- *Goal Flow*: Instead of training a multi-step policy to reach the goal, train a network to output a single dense prediction which assigns a per-point 3D flow vector that points from each action object point directly to its corresponding goal location. We extract a rigid transformation from these flow vectors using SVD, yielding the goal pose [15].

Note that in the PartNet-Mobility Placement experiments, the pose of the anchor object poses are randomly varied. As such, we omit a comparison to methods that assume a static anchor, such as the Neural Descriptor Field (NDF) [45] and Dense Object Nets (DON) [16] baselines used in the mug hanging task (Section 3.3.1), as both methods assume that the anchor objects are in a fixed, known position.

Results

We report rotation ($\mathcal{E}_{\mathbf{R}}$) and translation ($\mathcal{E}_{\mathbf{t}}$) error between our predicted transform and the ground truth as geodesic rotational distance [17, 23] and $L2$ distance, respectively. In both our simulated experiments (Table 3.4 Top) and our real-world experiments (Table 3.4 Bottom), we find that TAX-Pose outperforms the baseline end-to-end imitation learning methods, with the *goal-conditioned* variant, TAX-Pose

	Average	
	$\mathcal{E}_{\mathbf{R}}$	$\mathcal{E}_{\mathbf{t}}$
E2E BC	42.26	0.73
E2E DAgger	37.96	0.69
Traj. Flow	35.95	0.67
Goal Flow	26.64	0.17
TAX-Pose	6.64	**0.16**
TAX-Pose GC	**4.94**	**0.16**

	Average SR
Goal Flow	0.31
TAX-Pose	**0.92**

Table 3.4: **Top**: Simulation Rotational (°) and Translational (m) Errors (↓). **Bottom**: Real-world goal placement success rate (↑).

GC, performing the best.

In real-world experiments, our method generalizes to novel distributions of starting poses better than the Goal Flow baseline, placing action objects into the goal regions with a 92% success rate. See Figure 3.5 (left) and the website for results; see Appendix B.6.2 for more detailed tables and Appendix B.7.2 for baseline details.

3.4 Conclusions

In this chapter, we show that dense soft correspondence can be used to learn task specific object relationships that generalize to novel object instances. Correspondence residuals allow our method to estimate correspondences to virtual points, outside of the object's convex hull, drastically increasing the number of tasks this method can complete. We further show that this "cross-pose" can be learned for a task, using a small number of demonstrations. Finally, we show that our method far outperforms the baselines on two challenging tasks in both real and simulated experiments. While our method is able to predict relative pose relationships with high precision, it has several limitations:

- **Requires segmentation**: Our method requires an accurate segmentation of two objects in order to predict their relative goal pose.

- **Performance degrades under occlusion**: Our method performs best when complete point clouds are provided, captured via multiple cameras or by repeatedly reorienting the objects.
- **Poorly defined for multimodal relationships**: Because our method extracts a single global estimate of relative pose from a fixed set of correspondences, performance on objects with multiple valid goals is not well-defined. Our method might be augmented with a consensus-based or sampling-based approach to capture the multimodality of the solution space in these cases. We leave this for future work.

Furthermore, we showed that GoalFlow, while being a promising representation of articulated objects, is an inferior visual representation of free-floating objects. This brings up a question worth discussing - is there a way to unify the two representations that can generalize to both articulated and free-floating objects? In the next chapter, we propose one possible solution.

Chapter 4

Weighted Pose: Unified Architecture for Articulated and Free-Floating Objects Manipulation

4.1 Introduction

In Chapter 2 and 3, we introduced two architectures for learning generalizable 3D visual representations for manipulating articulated objects and free-floating objects respectively. While they comprise a wide range of objects manipulation tasks in our daily lives, those two models alone cannot be applied to more complex tasks such as "open an oven, and put a tray inside the oven," which typically involve sequentially manipulating both categories of objects. To this end, we propose a method, WeightedPose, to unify the two architectures in order to accomplish more complex manipulation tasks.

In FlowBot 3D, we assumed that the robot is actuating a standalone articulated object. In TAX-Pose experiments, the anchor object remains still and cannot be directly manipulated. Experimentally, this is because TAX-Pose is not trained to be aware of the kinematic constraints of the objects by construction and 3DAF's accuracy degrades when the goal configuration is far away from the start. Intuitively, we should keep the convention of using TAX-Pose for free-floating objects and FlowBot 3D for

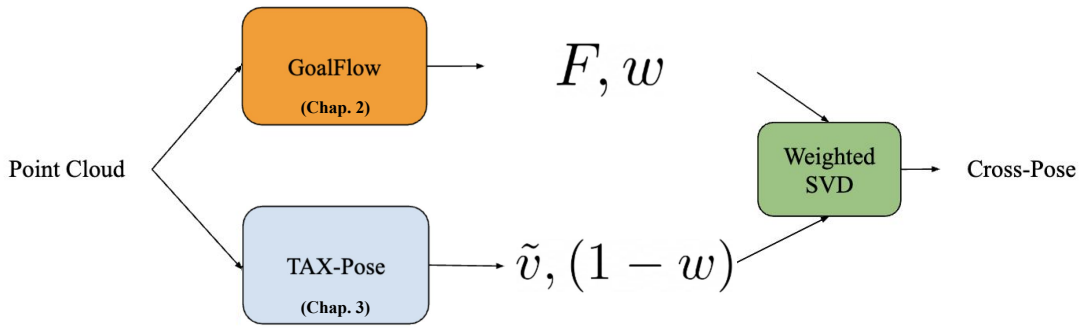


Figure 4.1: Unified Weighted Pose architecture. The model first takes as input a point cloud, and then learns to predict a weight for the point cloud. This weight is used in the downstream SVD module to combine the GoalFlow and TAX-Pose outputs.

articulated objects in WeightedPose.

For more complex manipulation tasks such as "open an oven, and put a tray inside the oven," the robot needs to reason about the relationship between key parts on one object with respect to key parts on another object. Specifically, the robot needs to reason about both the relationship between the oven door and the oven body, as well as the relationship between the lasagna plate and the oven.

To this end, we propose a standalone model, Weighted Pose, that utilizes weighted SVD to reason about both pose relationships between articulated parts and between free-floating objects.

4.2 Method

To make FlowBot 3D compatible with both categories of objects, we first slightly modify the FlowBot 3D training objective. Instead of defining an instantaneous motion vector field as the training data directly, we have the network learn to output a flow field to the open state directly. Since this version of FlowBot 3D learns to output a dense representation to the open (goal) state directly, we call this model Goal Flow.

Formally, given a point cloud $\{\mathbf{p}_i\}$ $\forall i \in \{1, \ldots, N\}$ this Goal Flow model outputs a dense flow field $F \in \mathbb{R}^{N \times 3}$, where each flow vector $\delta_i \in \mathbb{R}^3$ in the F represents a

goal flow vector such that point $p_i + \delta_i$ is in the open goal state. Ideally, this model should be deployed exclusively for articulated objects in that Goal Flow was shown to achieve suboptimal performances in Chapter 3.

To combine the Goal Flow model with TAX-Pose, we also make the Goal Flow network output an auxiliary weight $w \in [0, 1]$, which assigns weight w to Goal Flow and $1 - w$ to TAX-Pose.

For the TAX-Pose component of the Weighted Pose unified architecture, we do not make any significant modifications and ideally the TAX-Pose model should be deployed for free-floating objects exclusively.

To recap the mathematical notations defined in Chapter 3, given objects $\mathcal{A}$ and $\mathcal{B}$ with point cloud observations $\mathbf{P}_\mathcal{A} \in \mathbb{R}^{3\times N_\mathcal{A}}$, $\mathbf{P}_\mathcal{B} \in \mathbb{R}^{3\times N_\mathcal{B}}$ respectively, our objective is to estimate the task-specific cross-pose $\mathbf{T}_{\mathcal{AB}} = f(\mathbf{P}_\mathcal{A}, \mathbf{P}_\mathcal{B}) \in SE(3)$. Note that the cross-pose between object $\mathcal{A}$ and $\mathcal{B}$ is defined with respect to a given task (e.g. putting a lasagna in the oven, putting a mug on the rack, etc).

In TAX-Pose, we have:

$$\mathcal{J}(\mathbf{T}_{\mathcal{AB}}) = \sum_{i=1}^{N_\mathcal{A}} \alpha_i^\mathcal{A} ||\mathbf{T}_{\mathcal{AB}}\ \mathbf{p}_i^\mathcal{A} - \tilde{\mathbf{v}}_i^\mathcal{A}||_2^2 + \sum_{i=1}^{N_\mathcal{B}} \alpha_i^\mathcal{B} ||\mathbf{T}_{\mathcal{AB}}^{-1}\ \mathbf{p}_i^\mathcal{B} - \tilde{\mathbf{v}}_i^\mathcal{B}||_2^2,$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{P}_\mathcal{A}^{*\top} & \tilde{\mathbf{V}}_\mathcal{B}^{*\top} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \tilde{\mathbf{V}}_\mathcal{A}^{*\top} & \mathbf{P}_\mathcal{B}^{*\top} \end{bmatrix}^\top, \quad \mathbf{\Gamma} = \text{diag}\left(\begin{bmatrix} \boldsymbol{\alpha}_\mathcal{A} & \boldsymbol{\alpha}_\mathcal{B} \end{bmatrix} \right)$$

and $\tilde{\mathbf{v}}_i^\mathcal{A}$ define the estimated goal location relative to object $\mathcal{B}$ for each point $\mathbf{p}_i \in \mathbf{P}_\mathcal{A}$ of object $\mathcal{A}$, and likewise for each point in object $\mathcal{B}$. The weights $\alpha_i^\mathcal{A} \in \boldsymbol{\alpha}_\mathcal{A}$, $\alpha_i^\mathcal{B} \in \boldsymbol{\alpha}_\mathcal{B}$ signify the importance of each correspondence and are predicted by a point-wise MLP. See detailed definition in Chapter 3.

To learn which of the two models to use, we want to add another term in the SVD step. Specifically, we want the network to learn which one of the two models, TAX-Pose and Goal-Flow, is more important based on point cloud observations. Thus, we want an SVD step that incorporates the TAX-Pose residual, weighted by

$(1-w)$ and Goal-Flow weighted by w. So the new SVD formulation becomes:

$$\mathcal{J}(\mathbf{T}_{\mathcal{AB}}) = (1-w)\left[\sum_{i=1}^{N_{\mathcal{A}}} \alpha_i^{\mathcal{A}} ||\mathbf{T}_{\mathcal{AB}}\ \mathbf{p}_i^{\mathcal{A}} - \tilde{\mathbf{v}}_i^{\mathcal{A}}||_2^2 + \sum_{i=1}^{N_{\mathcal{B}}} \alpha_i^{\mathcal{B}} ||\mathbf{T}_{\mathcal{AB}}^{-1}\ \mathbf{p}_i^{\mathcal{B}} - \tilde{\mathbf{v}}_i^{\mathcal{B}}||_2^2\right] + w\sum_{i=1}^{N_{\mathcal{A}}} ||\mathbf{T}_{\mathcal{AB}}\ \mathbf{p}_i^{\mathcal{A}} - (\mathbf{p}_i^{\mathcal{A}} + \delta_i^{\mathcal{A}})||_2^2,$$

where $\delta_i^{\mathcal{A}}$ is the $i-$th point's goal flow. To make this approach viable we optimize R and t in $\mathbf{T}_{\mathcal{AB}}$ separately:

$$J = \sum_{i}^{N_{\mathcal{A}}} \left((1-w)\alpha_i \|R\mathbf{p}_i^{\mathcal{A}} + t - \tilde{\mathbf{v}}_i^{\mathcal{A}}\|^2 + w\|R\mathbf{p}_i^{\mathcal{A}} + t - \mathbf{p}_i^{\mathcal{A}} + \delta_i^{\mathcal{A}}\|^2\right) + \sum_{i}^{N_{\mathcal{B}}} (1-w)\alpha_i \|R^{-1}\mathbf{p}_i^{\mathcal{B}} - t - \tilde{\mathbf{v}}_i^{\mathcal{B}}\|^2$$

To optimize this objective function, we first solve for the optimal translation t^*:

$$
\begin{aligned}
\frac{\partial J}{\partial t} &= 0 \\
&= \sum_i^{N_\mathcal{A}} \left(2(1-w)\alpha_i \left(t + R\mathbf{p}_i^\mathcal{A} - \tilde{\mathbf{v}}_i^\mathcal{A}\right) + 2w\left(t + R\mathbf{p}_i^\mathcal{A} - \mathbf{p}_i^\mathcal{A} - \delta_i^\mathcal{A}\right)\right) \\
&+ 2(1-w)\sum_i^{N_\mathcal{B}} \alpha_i \left(R^{-1}\mathbf{p}_i^\mathcal{B} - t - \tilde{\mathbf{v}}_i^\mathcal{B}\right) \\
&= \sum_i^{N_\mathcal{A}} \left((2-2w)\alpha_i + 2w\right) t + \left((2-2w)\alpha_i + 2w\right) R\mathbf{p}_i^\mathcal{A} - 2w\delta_i^\mathcal{A} \\
&- (2-2w)\alpha_i \tilde{\mathbf{v}}_i^\mathcal{A} - 2w\mathbf{p}_i^\mathcal{A} \\
&+ \sum_i^{N_\mathcal{B}} -(2-2w)\alpha_i t + (2-2w)\alpha_i (R^{-1}\mathbf{p}_i^\mathcal{B} - \tilde{\mathbf{v}}_i^\mathcal{B}) \\
&= \left[\sum_i^{N_\mathcal{A}} [(2-2w)\alpha_i + 2w] + \sum_i^{N_\mathcal{B}} -(2-2w)\alpha_i\right] t \\
&+ \sum_i^{N_\mathcal{A}} \left((2-2w)\alpha_i + 2w\right) R\mathbf{p}_i^\mathcal{A} - 2w\delta_i^\mathcal{A} - (2-2w)\alpha_i \tilde{\mathbf{v}}_i^\mathcal{A} - 2w\mathbf{p}_i^\mathcal{A} \\
&+ \sum_i^{N_\mathcal{B}} (2-2w)\alpha_i (R^{-1}\mathbf{p}_i^\mathcal{B} - \tilde{\mathbf{v}}_i^\mathcal{B}) \\
t^* &= -\Bigg(\sum_i^{N_\mathcal{A}} \left((2-2w)\alpha_i + 2w\right) R\mathbf{p}_i^\mathcal{A} - 2w\delta_i^\mathcal{A} - (2-2w)\alpha_i \tilde{\mathbf{v}}_i^\mathcal{A} - 2w\mathbf{p}_i^\mathcal{A} \\
&+ \sum_i^{N_\mathcal{B}} (2-2w)\alpha_i (R^{-1}\mathbf{p}_i^\mathcal{B} - \tilde{\mathbf{v}}_i^\mathcal{B}) \Bigg) \Bigg/ \left[\sum_i^{N_\mathcal{A}} [(2-2w)\alpha_i + 2w] + \sum_i^{N_\mathcal{B}} -(2-2w)\alpha_i\right]
\end{aligned}
$$

Further simplifying, we have:

$$
\begin{aligned}
t^* = & \frac{(1-w)\sum_i^{N_\mathcal{A}} \alpha_i^\mathcal{A}(\tilde{\mathbf{v}}_i^\mathcal{A} - R\mathbf{p}_i^\mathcal{A})}{(1-w)\sum_i^{N_\mathcal{A}} \alpha_i^\mathcal{A} + w\sum_i^{N_\mathcal{A}} 1 + (1-w)\sum_i^{N_\mathcal{B}} \alpha_i^\mathcal{B}} \\
& + \frac{w\sum_i^{N_\mathcal{A}}(\mathbf{p}_i^\mathcal{A} + \delta_i^\mathcal{A}) - R\mathbf{p}_i^\mathcal{A}}{(1-w)\sum_i^{N_\mathcal{A}} \alpha_i^\mathcal{A} + w\sum_i^{N_\mathcal{A}} 1 + (1-w)\sum_i^{N_\mathcal{B}} \alpha_i^\mathcal{B}} \\
& + \frac{(1-w)\sum_i^{N_\mathcal{B}} \alpha_i^\mathcal{B}(\tilde{\mathbf{v}}_i^\mathcal{B} - R^{-1}\mathbf{p}_i^\mathcal{B})}{(1-w)\sum_i^{N_\mathcal{A}} \alpha_i^\mathcal{A} + w\sum_i^{N_\mathcal{A}} 1 + (1-w)\sum_i^{N_\mathcal{B}} \alpha_i^\mathcal{B}}
\end{aligned}
$$

Note here that we colorcode the expression here. Intuitively, the resulting translation is a weighted sum of three translation terms. The red color represents the action object's translation via TAX-Pose, the blue color represents the action object's translation via GoalFlow, and the purple color represents the anchor object's translation via TAX-Pose.

We then construct the matrices as follows:

$$
\mathbf{A} = \begin{bmatrix} \mathbf{P}_\mathcal{A}^\top & \tilde{\mathbf{V}}_\mathcal{B}^\top & \mathbf{P}_\mathcal{A}^\top \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \tilde{\mathbf{V}}_\mathcal{A}^\top & \mathbf{P}_\mathcal{B}^\top & \mathbf{P}_\mathcal{A}^\top + \Delta_\mathcal{A} \end{bmatrix}^\top
$$

$$
\mathbf{\Gamma} = \begin{bmatrix} (1-w)\cdot\boldsymbol{\alpha}_\mathcal{A} & 0 & 0 \\ 0 & (1-w)\cdot\boldsymbol{\alpha}_\mathcal{B} & 0 \\ 0 & 0 & w \end{bmatrix}
$$

where $\Delta_\mathcal{A}$ is the de-meaned goal flow field. Note that everything here is **NOT** de-meaned.

We then solve for the SVD:

$$
\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \text{svd}(\boldsymbol{A}\boldsymbol{\Gamma}\boldsymbol{B}^\top)
$$

and then we solve for rotation matrix R. Plugging it back into the t^* equations we get t.

4.3 Experiments

We use the same PartNet-Mobility dataset as in TAX-Pose to evaluate the system. For semantic tasks, we only select the "In" task. We also evaluate the method on opening articulated objects. Under this formulation, the action object is the articulated part and the anchor object is the static body. We measure rotational error, translational error, and per-point MSE in our experiments. We follow the same train-validation split as in TAX-Pose's PartNet-Mobility dataset.

		GF Pretrained		**TP Pretrained**		**WP OG Loss**		**WP Post-SVD**		**WP Post SVD + T**	
	Metrics	FF	Art	FF	Art	FF	Art	FF	Art	FF	Art
Train	**Rot err**	31.61	**5.16**	**2.61**	55.78	11.69	3.15	13.14	3.28	14.04	5.52
	Trans err	1.21	**0.16**	**0.04**	0.98	0.14	0.08	0.21	0.07	0.21	0.07
	PP MSE	1.04	**0.04**	**0.01**	0.85	0.07	0.05	0.09	0.04	0.11	0.08
Val	**Rot err**	35.5	**9.14**	**9.87**	59.73	11.22	9.01	14.13	9.71	13.06	10.03
	Trans err	1.3	**0.19**	**0.18**	0.99	0.26	0.15	0.22	0.18	0.23	0.18
	PP MSE	1.07	**0.1**	**0.15**	0.82	0.16	0.11	0.17	0.12	0.18	0.11

Table 4.1: Weighted Pose Results: We compare Rotation Error (°), Translational Error (m), and Per-Point MSE for both training and validation objects. Bolded numbers mean the model's performance on the object that it "specializes in."

We compare with the following baselines:

- Goal Flow Pretrained (GF Pretrained): Pretrain Goal Flow on articulated objects only and test on both articulated and free-floating objects.
- TAX-Pose Pretrained (TP Pretrained): Pretrain TAX-Pose on free-floating objects only and test on both articulated and free-floating objects.
- Weighted Pose Original Loss (WP OG Loss): Weighted Pose trained and test on both free-floating and articulated objects using the original TAX-Pose loss.
- Weighted Pose Post-SVD Loss (WP Post-SVD): Weighted Pose trained and test on both free-floating and articulated objects but using the post-SVD TAX-Pose loss.
- Weighted Pose Post-SVD & Transformation Loss (WP Post-SVD): Weighted Pose trained and test on both free-floating and articulated objects but using both the post-SVD TAX-Pose loss and direct transformation loss. Where the transformation loss is the MSE between the predicted $SE(3)$ transformation and the ground-truth $SE(3)$ transformation.

In Table A.3, Goal Flow Pretrained baseline fails on free-floating objects and excels on articulated objects. Similarly for TAX-Pose pretrained, since it was trained on free-floating objects, it does do well on articulated objects, as the high rotational and translational errors indicate.

We next evaluate the variations of WeightedPose, which differ in the training paradigms. Using Weighted Pose Original Loss, which is trained using the original TAX-Pose loss, we are able to achieve better performance on both articulated and free-floating objects. Interestingly, the results we achieve using Weighted Pose Original Loss for articulated objects are better than the results from using Goal Flow pretrained on articulated objects.

While ideally, the w learned in this method should effectively act as a classifier of the input object (1 for articulated objects, 0 for free-floating objects). Intuitively, given a perfect w, the performance of WeightedPose would be upper-bounded by Goal Flow's performance on articulated objects and by TAX-Pose on free-floating objects. However, a weighted combination of the two results due to the imperfect learned w weight could potentially correct the mistake made by each model by summing the results with a weighted sum. This may explain why WeightedPose sometimes performs better than its hypothesized upper bound on some objects.

Lastly, results suggest that the original loss used in TAX-Pose yields the best overall performance. However, it is worth noting that by using post-SVD loss during training, we are able to achieve lower translational error in test time. Interestingly, by introducing a direct $SE(3)$ supervision, the results degrade marginally.

4.4 Conclusions

In conclusion, we have presented a method to combine the two architectures using weighted SVD. While this model is more of a proof-of-concept that attempts to unify the two architectures, it is worth pointing out that using the two models for the two categories of objects is able to help us generate goal poses for various free-floating and articulated objects. Moreover, by finetuning pretrained models from a weighted SVD combination, we are able to outperform the models on their respective training datasets categories. In future work, we wish to generalize the mathematics of the combined architecture beyond tasks that involve articulated and free-floating objects.

We would also like to explore how such a unified architecture can aid motion planning as a geometric suggestor.

Chapter 5

Conclusions

In conclusion, we explored and presented two different approaches to training accurate, dense, generalizable 3D visual representations for manipulation tasks.

In the first part, we propose a novel visual representation for articulated objects, namely 3D Articulation Flow, as well as a policy – FlowBot3D – which leverages this representation to successfully manipulate articulated objects. We demonstrate the effectiveness of our method in both simulated and real environments, and observe strong sim-to-real transfer generalization. We hope our method allows more systems to perceive articulated objects in a more efficient way.

In the second part, we presented a new lens for solving pose estimation for manipulation tasks, instead of seeking to obtain accurate and robust single object absolute poses, we instead of directly train network to learn to predict the desired task-specific cross-pose between a pair of objects by learning from a few task demonstrations. We show that dense soft correspondence can be used to learn task-specific object relationships that generalize to novel object instances. Correspondence residuals allow our method to estimate correspondences to virtual points, outside of the object's convex hull, drastically increasing the number of tasks this method can complete. We further show that this "cross-pose" can be learned for a task, using a small number of demonstrations. Finally, we show that our method far outperforms the baselines on two challenging tasks in both real and simulated experiments.

Lastly, we propose a proof-of-concept model that aims to integrate the two approaches using weighted SVD, allowing for increased flexibility in terms of task

applicability. This unified approach demonstrates the potential for further advances in the field of manipulation tasks using 3D visual representations.

Appendix A

Appendix for Flowbot 3D

A.1 Robot System Details

A.1.1 Hardware

In all of real-world experiments, we deploy our system on a Rethink Sawyer Robot and the sensory data (point cloud) come from an Azure Kinect depth camera. The robot's end effector is an official Saywer Pneumatic Suction Gripper with a suction cup with a diameter of 3 cm. The air supply of the suction gripper is provided by a California Air Tools compressor.

A.1.2 Workspace

We set up our workspace in a 1.08 m by 1.00 m space put together using Vention beams. We set up the Azure Kinect camera such that it points toward the center of workspace and has minimal interference with the robot arm-reach trajectory. Collision geometry are set up using MoveIt's collision box construction tool. We add a number of boxes representing the camera and Vention beams that can potentially be blocking the robot during motion planning.

A.1.3 Hand-Eye Calibration

For Hand-Eye Calibration, we are using the Easy-Hand-Eye ROS package that calculates the transformation from the camera frame to world frame using an ArUco marker fixed on the robot's end effector. The process requires about 30 samples of the robot pose and ArUco marker pose combinations.

A.1.4 Foreground Segmentation

In simulated experiments, we have access to segmentation masks that segment out tabletop and robot from the collected point cloud. In real-world experiments, however, we need to programatically segment out those points ourselves.

Tabletop. We segment out the tabletop plane by simply subtracting the points with z values less than 0.015 m from the collected point cloud after calibration because the table top is placed 1.5 cm below the robot base.

Robot. The robot points are masked out in real-time by rendering the robot 3D model using its URDF file. This is done through a ROS package called Real Time URDF Filter. This filter assumes a perfect calibration of the camera. When the calibration is slightly off, some trailing points from the robot might remain in scene. Thus, we also statistically remove the outliers from the resulting point cloud because the remaining robot points are sparser than the object's points.

A.1.5 Contact Point Heuristic

In simulation, the suction contact is modeled by a kinematic constraint between the gripper point and the contact point. Therefore, in simulation, we have a perfect contact that can almost always successfully grasp the desired part. In real-world experiments, due to the complication of the physics of the suction gripper and the geometry of the target part, we can not always guarantee a successful grasp. Therefore, we add an extra heuristic upon the max-flow selection when selecting which point to grasp. Specifically, we add an interior point selection procedure that calculates the curvature of the points using PCA and we choose the point with curvature value smaller than a threshold. If the max-flow point has a curvature value higher than the threshold, we discard that point and choose the nearest low-curvature point at least

2 cm away from the max-flow point.

A.1.6 Grasp Selection Details

In the Grasp Selection phase of real-world experiments, we predict and estimate the part's 3D articulated flow vectors using FlowNet. We then use the aforementioned contact point heuristic to filter out points that have high curvature values. If the max-flow point is within the remaining points, we keep it and use it as the selected contact point. Otherwise, we choose the nearest low-curvature point at least 2 cm away from the max-flow point. Once we have selected the point, we have also selected the end effector's goal translation. For goal orientation, we align the end effector with the flow vector. The procedure is explained here: assume that the axis connecting the suction gripper tip to the robot hand is called $\mathbf{v}_1$ and the chosen flow vector is $-\mathbf{v}_2$, we aim to find a rotation that aligns $\mathbf{v}_1$ to $\mathbf{v}_2$ (because the robot approach direction is opposite to the flow direction). The difference of the rotation expressed in quaternion is calculated as follows:

$$\begin{aligned}
\phi_{12} &= \cos^{-1}(\mathbf{v}_1 \cdot \mathbf{v}_2) \\
\omega &= \mathbf{v}_1 \times \mathbf{v}_2 = [\omega_x, \omega_y, \omega_z] \\
q_x &= \omega_x \cdot \sin(\omega/2) \\
q_y &= \omega_y \cdot \sin(\omega/2) \\
q_z &= \omega_z \cdot \sin(\omega/2) \\
q_w &= \cos(\omega/2) \\
\mathbf{q} &= [q_x, q_y, q_z, q_w] \\
\mathbf{q}_{\text{diff}} &= \frac{\mathbf{q}}{||\mathbf{q}||}.
\end{aligned}$$

Therefore, when given the robot's starting rotation quaternion $\mathbf{q}_{\text{start}}$, the goal orientation of the robot end-effector $\mathbf{q}_{\text{goal}}$ is given by:

$$\mathbf{q}_{\text{goal}} = \mathbf{q}_{\text{diff}} \cdot \mathbf{q}_{\text{start}}$$

A.1.7 Robot Control Paradigm

In the Grasp Selection Phase of real-world experiments, the robot is controlled using position control by inputting the end-effector pose and solving for the trajectory using an RRTConnect-based [17] IK solver. One caveat about the Grasp Selection Phase in real world is that the robot does not make contact with the select point directly. Instead, the robot first aligns with the chosen flow vector and plans to a point 10 cm in the chosen 3D articulated flow direction away from the max-flow point. Then the robot switches the control mode to velocity control and approaches the proposed point in the aligned (negative selected flow) direction slowly until the force sensor of the robot reports reading greater than a threshold, meaning the robot makes contact with the object. Then in the Articulation-Execution Phase, the velocity controller takes as input the translational velocity represented by the current time step's normalized predicted articulation flow vector multiplied by a constant to decrease the speed and the rotational velocity as the aforementioned $\mathbf{q}_{\text{diff}}$ converted to Euler angles multiplied by another constant to decrease the angular speed.

A.2 Training Details

A.2.1 Network Architecture

ArtFlowNet is based on the PointNet++ [27] architecture. The architecture largely remains similar to the original architecture except for the output head. Instead of using a segmentation output head, we use a regression head. The ArtFlowNet architecture is implemented using Pytorch-Geometric [10], a graph-learning framework based on PyTorch. Since we are doing regression, we use standard L2 loss optimized by an Adam optimizer [16].

A.2.2 Ground Truth 3DAF Generation

We implement efficient ground truth 3D Articulation Flow generation. At each timestep, the system reads the current state of the object of interest in simulation as an URDF file and parses it to obtain a kinematic chain. Then the system uses the kinematic chain to analytically calculate each point's location after a small, given

amount of displacement. In simulation, since we have access to part-specific masks, the calculated points' location will be masked out such that only the part of interest will be articulated. Then we take difference between the calculated new points and the current time step's points to obtain the ground truth 3D Articulation Flow.

A.2.3 Simulator Modifications

We heavily modify the ManiSkill [23] environment, which is a high-level wrapper of the SAPIEN [32] simulator. Specifically, we add in a variety of PartNet-Mobility objects to the environment for more diverse training dataset. We obtain a list of training and testing objects from the authors of UMPNet [33]. We have filtered out some phone objects and door objects due to the collision of meshes in the SAPIEN simulator upon loading, but the dataset remains largely identical to the one used in UMPNet. Furthermore, we implement efficient online ground truth 3D articulation flow calculation in the ManiSkill environment for generating training data online. We also implement camera viewpoint sampling by randomizing the azimuth and elevation for the VPA model training. Instead of using a full robot arm, we only use a floating gripper with 8 DoF (x, y, z for translation, r, p, y for rotation, and speed parameter for each of the two fingers on the gripper) controlled by a velocity controller. The two gripper fingers' speed parameters are not learned in Behavioral Cloning as the two fingers remain closed. To simulate suction, we create a strong force between the gripper fingers and the target object since kinematic constraints are not directly supported in the SAPIEN simulator.

A.2.4 Hyperparameters

We use a batch size of 64 and a learning rate of 1e-4. We use the standard set of hyperparameters from the original PointNet++ paper.

A.3 Simulation Experiments Illustration

Here we briefly illustrate FlowBot3D system in simulation. In simulation, the suction is implemented using a strong force between the robot gripper and the target part.

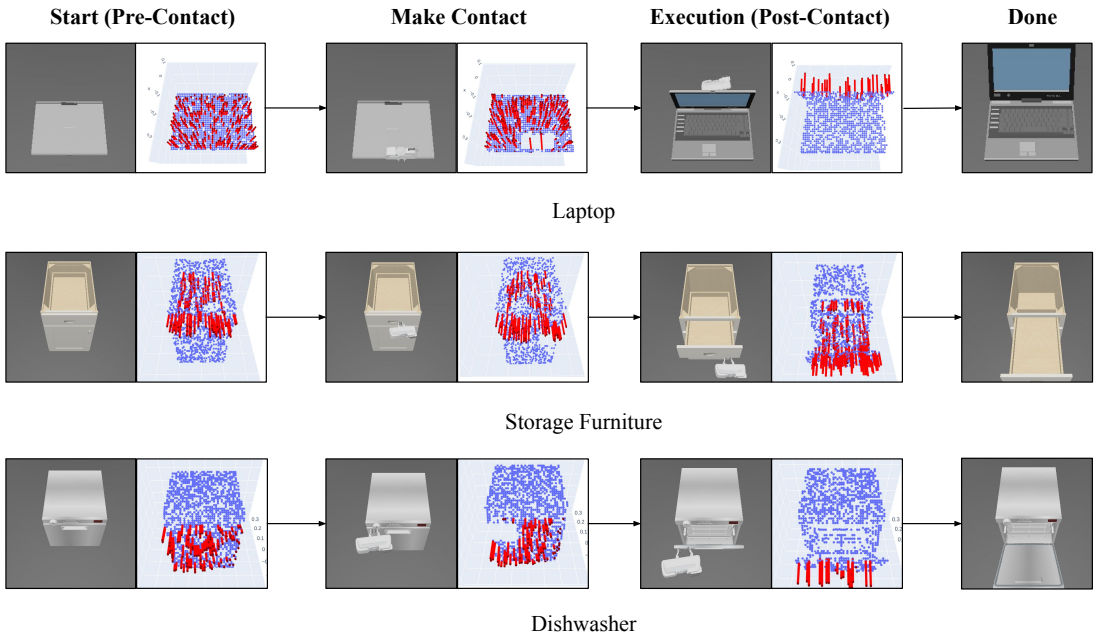


Figure A.1: Simulated rollout examples

A.4 Real-World Dataset

As shown in Section IV-B, we use 14 different objects in real world experiments. The objects labeled 1-14 in Fig. 5 are described here in Table A.1.

In Fig. A.2, we show the 14 objects individually for more clarity.

A.5 Results in the UMPNet Environment

We perform a direct evaluation of FlowBot3D in the UMPNet simulation environment, which was only made public after this manuscript was accepted for publication. There are several major differences between our main simulation environment and the UMPNet evaluation environment:

- The UMPNet environment uses the PyBullet physics simulator, whereas we use the SAPIEN environemnt (backed by PhysX).
- The UMPNet environment disables collisions between the gripper geometry and the rest of the object (except for the part where contact is made). We leave

Label	ID	Category	Type
1	`chest_1`	Box	Revolute
2	`teapot_1`	Kettle	Prismatic
3	`toilet_1`	Toilet	Revolute
4	`fridge_1`	Refrigerator	Revolute
5	`oven_1`	Oven	Revolute
6	`drawer_1`	Storage	Prismatic
7	`safe_1`	Safe	Revolute
8	`microwave_1`	Microwave	Revolute
9	`minifridge_1`	Refrigerator	Revolute
10	`jar_1`	Kitchen Pot	Prismatic
11	`jar_2`	Kitchen Pot	Prismatic
12	`trash_1`	Trash Can	Revolute
13	`laptop_1`	Laptop	Revolute
14	`box_1`	Box	Revolute

Table A.1: Labels and their corresponding objects and the objects' articulation types shown in Fig. 5 of the paper. Note that `jar_1` and `jar_2` are not technically kitchen pots but they do have lids similar to kitchen pots in functionality and have identical ariculation parameters.

full contact enabled.

- The UMPNet environment has a hard contact constraint between the object and the gripper, whereas our contact is softer, acting more like a spring.

	Novel Instances in Train Categories												Test Categories									
	AVG.												AVG.									
Baselines																						
UMPNet	0.18	**0.18**	**0.17**	0.32	0.32	0.05	0.06	**0.12**	**0.24**	0.23	0.18	0.08	**0.15**	0.23	**0.14**	0.04	**0.00**	0.25	0.27	0.09	0.21	**0.13**	**0.19**
Ours																						
FlowBot3D in UMPNet	**0.17**	0.42	0.22	**0.16**	**0.17**	**0.03**	**0.00**	0.20	0.51	**0.07**	**0.00**	**0.08**	0.21	**0.17**	0.29	**0.00**	0.06	**0.21**	**0.10**	**0.06**	**0.16**	0.29	0.73

Table A.2: Normalized Distance Metric Results: Normalized distances evaluated in the official UMPNet environment to the target articulation joint angle after a full rollout across different methods. The lower the better.

We use the UMPNet evaluation script without modification, with the exception that the chosen action is selected based on FlowBot3D instead. In Tables A.2 and A.3, we present the results for the following methods:

- **UMPNet**: We run a pre-trained UMPNet model with the official UMPNet

	Novel Instances in Train Categories												Test Categories									
	AVG.												AVG.									
Baselines																						
UMPNet	0.73	**0.73**	0.71	0.60	0.49	0.89	0.90	0.79	**0.60**	0.64	0.78	0.86	**0.75**	0.55	**0.80**	0.89	**1.00**	0.66	0.64	**0.77**	0.64	**0.75**	**0.76**
Ours																						
FlowBot3D in UMPNet	**0.81**	0.53	**0.74**	**0.81**	**0.82**	**0.96**	**0.99**	**0.79**	0.44	**0.90**	**1.00**	**0.89**	0.70	**0.69**	0.63	**1.00**	0.94	**0.67**	**0.89**	0.75	**0.66**	0.69	0.14

Table A.3: Success Rate Metric Results: Fraction of success trials (normalized distance less than 0.1) of different objects' categories after a full rollout across different methods evaluated in the official UMPNet environment. The higher the better.

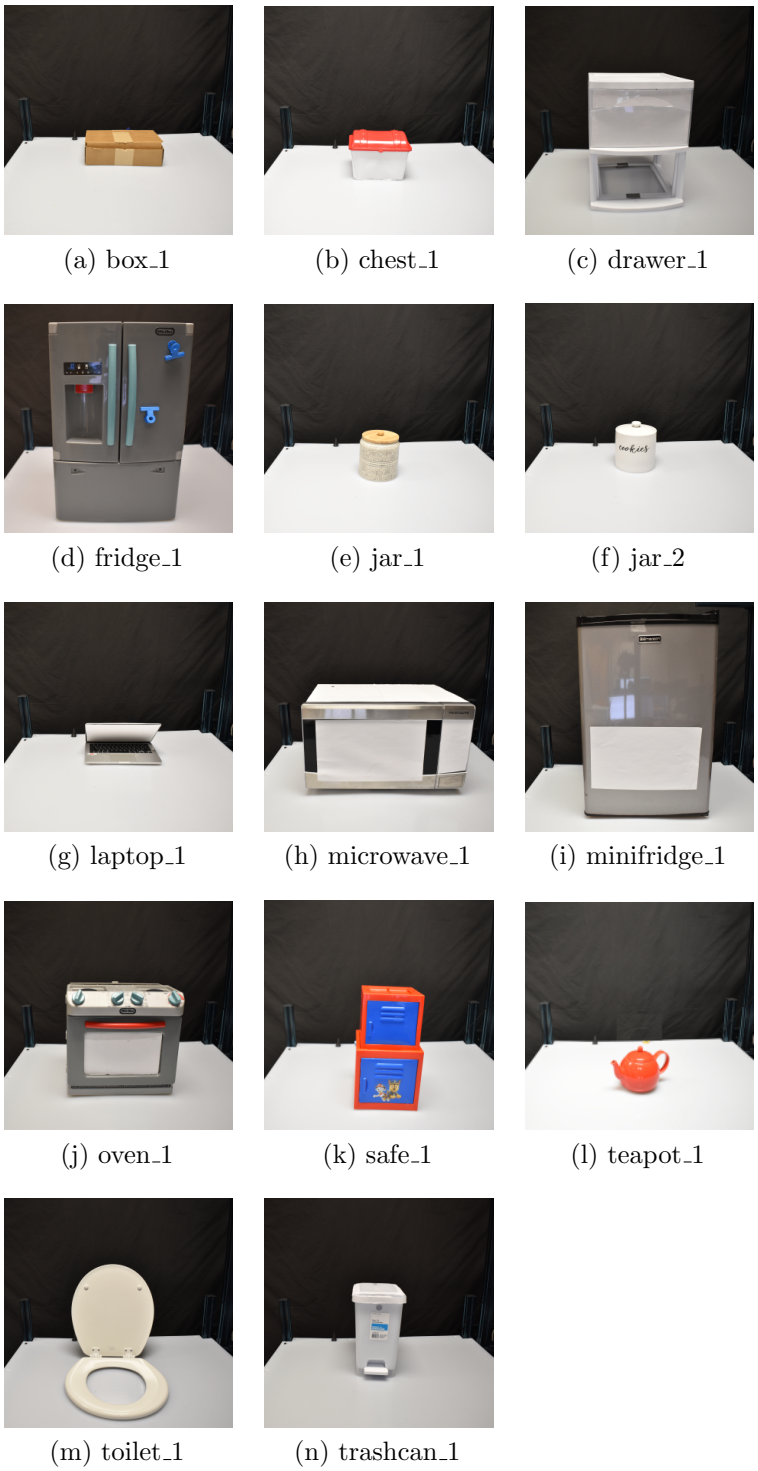
code following the exact same evaluation procedure listed in [33]. The numbers here are consistent with those in the UMPNet paper.

- **FlowBot3D in UMPNet Environment**: FlowBot3D trained and evaluated with the camera parameters and objects' placement randomization from UMPNet's PyBullet environment. Note that in test time, UMPNet takes as input a goal of the articulated object in its fully closed or fully open state, so we use the ground-truth goal to decide if we need to invert the output 3DAF directions (i.e. if the ground-truth goal is a fully closed state, we invert the output direction).

Overall, the two methods perform similarly on the task. However, while the ArtFlowNet was retrained on point clouds generated in PyBullet, the performance was not significantly tuned on the different task distribution in the UMPNet dataset.

A.6 Full Trials Results

In Table A.4 and A.5, we show the full trials results, which contains the metrics averaged over all 5 trials for each object.



(a) box_1 (b) chest_1 (c) drawer_1

(d) fridge_1 (e) jar_1 (f) jar_2

(g) laptop_1 (h) microwave_1 (i) minifridge_1

(j) oven_1 (k) safe_1 (l) teapot_1

(m) toilet_1 (n) trashcan_1

Figure A.2: Objects in the dataset for real world experiments

Object ID	Object Category	Success/Total	Success %	Contact Success/Total	Distance	Motion-Only Success/Total
chest_1	Box	3/5	60%	4/5	0.22	3/4
teapot_1	Kettle	5/5	100%	5/5	0.00	5/5
toilet_1	Toilet	4/5	80%	5/5	0.02	4/5
fridge_1	Refrigerator	3/5	60%	3/5	0.11	5/5
oven_1	Oven	0/5	0%	5/5	1.00	0/5
drawer_1	Storage	3/5	60%	3/5	0.40	3/3
safe_1	Safe	1/5	20%	2/5	0.73	1/2
microwave_1	Microwave	3/5	60%	5/5	0.11	3/5
minifridge_1	Refrigerator	2/5	40%	5/5	0.155	2/5
jar_1	Kitchen Pot	5/5	100%	5/5	0.00	5/5
jar_2	Kitchen Pot	5/5	100%	5/5	0.00	5/5
trash_1	Trash Can	5/5	100%	5/5	0.02	5/5
laptop_1	Laptop	4/5	100%	5/5	0.07	4/5
box_1	Box	2/5	40%	5/5	0.28	2/5
SUMMARY	**-**	**45/70**	**64.3%**	**64/70**	**0.22**	**45/64**

Table A.4: Real-World Trials for FlowNet

Object ID	Object Category	Success/Total	Success %	Contact Success/Total	Distance	Motion-Only Success/Total
chest_1	Box	1/5	20%	5/5	0.80	1/5
teapot_1	Kettle	2/5	40%	5/5	0.60	2/5
toilet_1	Toilet	0/5	0%	5/5	0.78	0/5
fridge_1	Refrigerator	0/5	0%	5/5	1.00	0/5
oven_1	Oven	0/5	0%	5/5	1.00	0/5
drawer_1	Storage	1/5	20%	5/5	0.72	1/5
safe_1	Safe	1/5	20%	3/5	0.70	1/3
microwave_1	Microwave	0/5	0%	5/5	1.00	0/5
minifridge_1	Refrigerator	0/5	0%	5/5	1.00	0/5
jar_1	Kitchen Pot	3/5	60%	5/5	0.40	3/5
jar_2	Kitchen Pot	1/5	20%	5/5	0.80	1/5
trash_1	Trash Can	0/5	0%	5/5	1.00	0/5
laptop_1	Laptop	1/5	20%	5/5	0.81	1/5
box_1	Box	1/5	20%	5/5	0.80	1/5
SUMMARY	**-**	**10/70**	**14.3%**	**68/70**	**0.73**	**10/68**

Table A.5: Real-World Trials for DAgger Oracle

Appendix B

Appendix for TAX-Pose

B.1 Visual Explanations of TAX-Pose

B.1.1 Illustration of Corrected Virtual Correspondence

The virtual corresponding points, $\mathbf{V}_{\mathcal{A}}$, $\mathbf{V}_{\mathcal{B}}$ given by Equation 3 in the main text, are constrained to be within the convex hull of each object. However, correspondences which are constrained to the convex hull are insufficient to express a large class of desired tasks. For instance, we might want a point on the handle of a teapot to correspond to some point above a stovetop, which lies outside the convex hull of the points on the stovetop. To allow for such placements, for each point-wise embedding $\boldsymbol{\phi}_i$, we further learn a *residual vector*, $\boldsymbol{\delta}_i^{\mathcal{A}} \in \boldsymbol{\Delta}_{\mathcal{A}}$ that corrects each virtual corresponding point, allowing us to displace each virtual corresponding point to any arbitrary location that might be suitable for the task. Concretely, we use a point-wise neural network $g_{\mathcal{R}}$ which maps each embedding into a 3D residual vector:

$$\boldsymbol{\delta}_i^{\mathcal{A}} = g_{\mathcal{R}}\left(\boldsymbol{\phi}_i^{\mathcal{A}}\right) \in \mathbb{R}^3, \;\; \boldsymbol{\delta}_i^{\mathcal{B}} = g_{\mathcal{R}}\left(\boldsymbol{\phi}_i^{\mathcal{B}}\right) \in \mathbb{R}^3$$

Applying these to the virtual points, we get a set of *corrected virtual correspondences*, $\tilde{\mathbf{v}}_i^{\mathcal{A}} \in \tilde{\mathbf{V}}_{\mathcal{A}}$ and $\tilde{\mathbf{v}}_i^{\mathcal{B}} \in \tilde{\mathbf{V}}_{\mathcal{B}}$, defined as

$$\tilde{\mathbf{v}}_i^{\mathcal{A}} = \mathbf{v}_i^{\mathcal{A}} + \boldsymbol{\delta}_i^{\mathcal{A}}, \;\; \tilde{\mathbf{v}}_i^{\mathcal{B}} = \mathbf{v}_i^{\mathcal{B}} + \boldsymbol{\delta}_i^{\mathcal{B}} \tag{B.1}$$

These corrected virtual correspondences $\tilde{\mathbf{v}}_i^{\mathcal{A}}$ define the estimated goal location relative to object $\mathcal{B}$ for each point $\mathbf{p}_i \in \mathbf{P}_{\mathcal{A}}$ in object $\mathcal{A}$, and likewise for each point in object $\mathcal{B}$, as shown in Figure B.1.

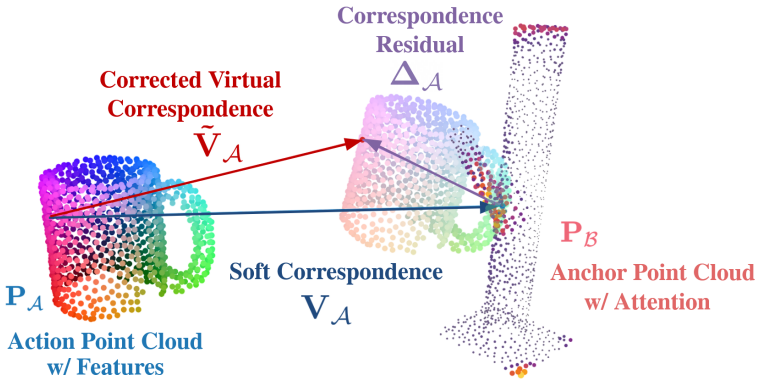


Figure B.1: Computation of Corrected Virtual Correspondence. Given a pair of object point clouds $\mathbf{P}_{\mathcal{A}}, \mathbf{P}_{\mathcal{B}}$, a per-point *soft correspondence* $\mathbf{V}_{\mathcal{A}}$ is first computed. Next, to allow the predicted correspondence to lie beyond object's convex hull, these soft correspondences are adjusted with *correspondence residuals*, $\mathbf{\Delta}_{\mathcal{A}}$, which results in the *corrected virtual correspondence*, $\tilde{\mathbf{V}}_{\mathcal{A}}$. The coloring scheme and the point size on the rack represent the the value of the the attention weights, where the more red and larger the point, the higher the attention weights, the more gray and smaller the point the lower the attention weights.

B.1.2 Learned Importance Weights

A visualization of the learned importance weights, $\alpha_{\mathcal{A}}$ and $\alpha_{\mathcal{B}}$ for the mug and rack are visualized by both color scheme and point size in Figure B.2

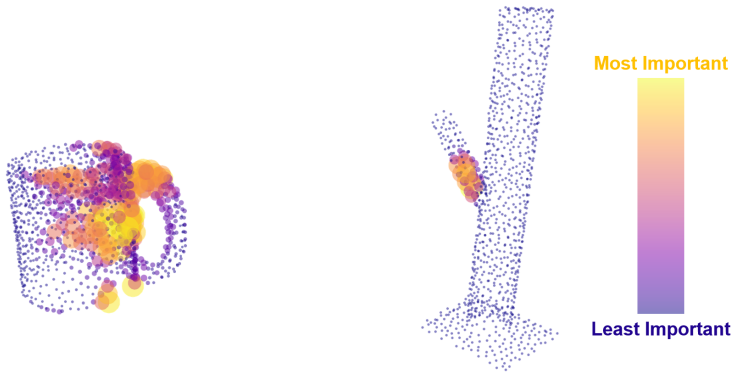


Figure B.2: Learned Importance Weights for Weighted SVD on Mug and Rack. The coloring scheme and the point size on both objects represent the the value of the the learned importance weights, where the more yellow and larger the point, the higher the learned importance weights, the more purple and smaller the point the lower the learned importance weights.

B.2 Proof of TAX-Pose Translational Equivariance

One benefit of our method is that it is translationally equivariant by construction. This mean that if the object point clouds, $\mathbf{P}_{\mathcal{A}}$ and $\mathbf{P}_{\mathcal{B}}$, are translated by random translation $\mathbf{t}_\alpha$ and $\mathbf{t}_\beta$, respectively, i.e. $\mathbf{P}_{\mathcal{A}'} = \mathbf{P}_{\mathcal{A}} + \mathbf{t}_\alpha$ and $\mathbf{P}_{\mathcal{B}'} = \mathbf{P}_{\mathcal{B}} + \mathbf{t}_\beta$, then the resulting corrected virtual correspondences, $\tilde{\mathbf{V}}_{\mathcal{B}}$ and $\tilde{\mathbf{V}}_{\mathcal{A}}$, respectively, are transformed accordingly, i.e. $\tilde{\mathbf{V}}_{\mathcal{B}} + \mathbf{t}_\beta$ and $\tilde{\mathbf{V}}_{\mathcal{A}} + \mathbf{t}_\alpha$, respectively, as we will show below. This results in an estimated cross-pose transformation that is also equivariant to translation by construction. This is achieved because our learned features and correspondence residuals are invariant to translation, and our virtual correspondence points are equivariant to translation.

First, our point features are a function of centered point clouds. That is, given point clouds $\mathbf{P}_{\mathcal{A}}$ and $\mathbf{P}_{\mathcal{B}}$, the mean of each point cloud is computed as

$$\bar{\mathbf{p}}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbf{P}_k.$$

This mean is then subtracted from the clouds,

$$\bar{\mathbf{P}}_k = \mathbf{P}_k - \bar{\mathbf{p}}_k,$$

which centers the cloud at the origin. The features are then computed on the centered point clouds:

$$\mathbf{\Phi}_k = g_k(\bar{\mathbf{P}}_k).$$

Since the point clouds are centered before features are computed, the features $\mathbf{\Phi}_k$ are invariant to an arbitrary translation $\mathbf{P}_{k'} = \mathbf{P}_k + \mathbf{t}_\kappa$.

These translationally invariant features are then used, along with the original point clouds, to compute "corrected virtual points" as a combination of virtual correspondence points, $\mathbf{v}_i^{k'}$ and the correspondence residuals, $\boldsymbol{\delta}_i^{k'}$. As we will see below, the "corrected virtual points" will be translationally equivariant by construction.

The virtual correspondence points, $\mathbf{v}_i^{k'}$, are computed using weights that are a function of only the translationally invariant query and key values from the cross-object attention transformer $g_{\mathcal{T}_\mathcal{K}}$, $\mathbf{Q}_\mathcal{K}$ and $\mathbf{K}_\mathcal{K}$, which are in turn functions of only the translationally invariant features, $\mathbf{\Phi}_k$:

$$\mathbf{w}_i^{\mathcal{A}'\to\mathcal{B}'} = \text{softmax}\left(\frac{\mathbf{K}_{\mathcal{B}'}\mathbf{q}_i^{\mathcal{A}'}}{\sqrt{d}}\right) = \text{softmax}\left(\frac{\mathbf{K}_{\mathcal{B}}\mathbf{q}_i^{\mathcal{A}}}{\sqrt{d}}\right) = \mathbf{w}_i^{\mathcal{A}\to\mathcal{B}}$$

thus the weights are also translationally invariant. These translationally invariant weights are applied to the translated cloud

$$\mathbf{v}_i^{\mathcal{A}'} = \mathbf{P}_{\mathcal{B}'}\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} = (\mathbf{P}_\mathcal{B} + \mathbf{t}_\beta)\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} = \sum_j \mathbf{p}_j^\mathcal{B} \cdot w_{i,j}^{\mathcal{A}\to\mathcal{B}} + \mathbf{t}_\beta \sum_j w_{i,j}^{\mathcal{A}\to\mathcal{B}} = \mathbf{P}_\mathcal{B}\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} + \mathbf{t}_\beta,$$

since $\sum_{j=1}^{N_\mathcal{B}} w_{ij}^{\mathcal{A}\to\mathcal{B}} = 1$. Thus the virtual correspondence points $\mathbf{v}_i^{\mathcal{A}'}$ are equivalently translated. The same logic follows for the virtual correspondence points $\mathbf{v}_i^{\mathcal{B}'}$. This gives us a set of translationally equivaraint virtual correspondence points $\mathbf{v}_i^{\mathcal{A}'}$ and $\mathbf{v}_i^{\mathcal{B}'}$.

The correspondence residuals, $\boldsymbol{\delta}_i^{k'}$, are a direct function of only the translationally

invariant features $\mathbf{\Phi}_k$,

$$\boldsymbol{\delta}_i^{k'} = g_{\mathcal{R}_\mathcal{K}}(\boldsymbol{\phi}_i^{k'}) = g_{\mathcal{R}_\mathcal{K}}(\boldsymbol{\phi}_i^{k}) = \boldsymbol{\delta}_i^{k},$$

therefore they are also translationally invariant.

Since the virtual correspondence points are translationally equivariant, $\mathbf{v}_i^{\mathcal{A}'} = \mathbf{v}_i^{\mathcal{A}} + \mathbf{t}_\beta$ and the correspondence residuals are translationally invariant, $\boldsymbol{\delta}_i^{k'} = \boldsymbol{\delta}_i^{k}$, the final corrected virtual correspondence points, $\tilde{\mathbf{v}}_i^{\mathcal{A}'}$, are translationally equivariant, i.e. $\tilde{\mathbf{v}}_i^{\mathcal{A}'} = \mathbf{v}_i^{\mathcal{A}} + \boldsymbol{\delta}_i^{k} + \mathbf{t}_\beta$. This also holds for $\tilde{\mathbf{v}}_i^{\mathcal{B}'}$, giving us the final translationally equivariant correspondences between the translated object clouds as $\left(\mathbf{P}_\mathcal{A} + \mathbf{t}_\alpha, \tilde{\mathbf{V}}_\mathcal{B} + \mathbf{t}_\beta\right)$ and $\left(\mathbf{P}_\mathcal{B} + \mathbf{t}_\beta, \tilde{\mathbf{V}}_\mathcal{A} + \mathbf{t}_\alpha\right)$, where $\tilde{\mathbf{V}}_\mathcal{B} = \left[\tilde{\mathbf{v}}_1^{\mathcal{A}} \dots \tilde{\mathbf{v}}_{N_\mathcal{A}}^{\mathcal{A}}\right]^\top$.

As a result, the final computed transformation will be automatically adjusted accordingly. Given that we use weighted SVD to compute the optimal transform, $\mathbf{T}_{\mathcal{AB}}$, with rotational component $\mathbf{R}_{\mathcal{AB}}$ and translational component $\mathbf{t}_{\mathcal{AB}}$, the optimal rotation remains unchanged if the point cloud is translated, $\mathbf{R}_{\mathcal{A'B'}} = \mathbf{R}_{\mathcal{AB}}$, since the rotation is computed as a function of the centered point clouds. The optimal translation is defined as

$$\mathbf{t}_{\mathcal{AB}} := \bar{\tilde{\mathbf{v}}}_\mathcal{A} - \mathbf{R}_{\mathcal{AB}} \cdot \bar{\mathbf{p}}_\mathcal{A},$$

where $\bar{\tilde{\mathbf{v}}}_\mathcal{A}$ and $\bar{\mathbf{p}}_\mathcal{A}$ are the means of the corrected virtual correspondence points, $\tilde{\mathbf{V}}_\mathcal{B}$, and the object cloud $\mathbf{P}_\mathcal{A}$, respectively, for object $\mathcal{A}$. Therefore, the optimal translation between the translated point cloud $\mathbf{P}_{\mathcal{A}'}$ and corrected virtual correspondence points $\tilde{\mathbf{V}}^{\mathcal{A}'}$ is

$$\begin{aligned}
\mathbf{t}_{\mathcal{A'B'}} &= \bar{\tilde{\mathbf{v}}}_{\mathcal{A}'} - \mathbf{R}_{\mathcal{AB}} \cdot \bar{\mathbf{p}}_{\mathcal{A}'} \\
&= \bar{\tilde{\mathbf{v}}}_\mathcal{A} + \mathbf{t}_\beta - \mathbf{R}_{\mathcal{AB}} \cdot (\bar{\mathbf{p}}_\mathcal{A} + \mathbf{t}_\alpha) \\
&= \bar{\tilde{\mathbf{v}}}_\mathcal{A} + \mathbf{t}_\beta - \mathbf{R}_{\mathcal{AB}} \cdot \bar{\mathbf{p}}_\mathcal{A} - \mathbf{R}_{\mathcal{AB}} \cdot \mathbf{t}_\alpha \\
&= \mathbf{t}_{\mathcal{AB}} + \mathbf{t}_\beta - \mathbf{R}_{\mathcal{AB}} \cdot \mathbf{t}_\alpha
\end{aligned}$$

To simplify the analysis, if we assume that, for a given example, $\mathbf{R}_{\mathcal{AB}} = \mathbf{I}$, then we get $\mathbf{t}_{\mathcal{A'B'}} = \mathbf{t}_{\mathcal{AB}} + \mathbf{t}_\beta - \mathbf{t}_\alpha$, demonstrating that the computed transformation is translation-equivariant by construction.

B.3 Description of Cross-Object Attention Weight Computation

To map our estimated features $\mathbf{\Psi}_{\mathcal{A}}$ and $\mathbf{\Psi}_{\mathcal{B}}$ obtained from object-specific embedding networks (DGCNN), $g_{\mathcal{A}}$ and $g_{\mathcal{B}}$ respectively, to a set of normalized weight vectors $\mathbf{W}_{\mathcal{A}\to\mathcal{B}}$ and $\mathbf{W}_{\mathcal{B}\to\mathcal{A}}$, we use the cross attention mechanism of our cross-object attention Transformer module [48]. Following Equations 5a and 5b from the paper, we can extract the desired normalized weight vector $\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}}$ for the virtual corresponding point $\mathbf{v}_i^{\mathcal{A}}$ assigned to any point $\mathbf{p}_i^{\mathcal{A}} \in \mathbf{P}^{\mathcal{A}}$ using the intermediate attention embeddings of cross-object attention module as:

$$\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} = \text{softmax}\left(\frac{\mathbf{K}_{\mathcal{B}}\mathbf{q}_i^{\mathcal{A}}}{\sqrt{d}}\right), \quad \mathbf{w}_i^{\mathcal{B}\to\mathcal{A}} = \text{softmax}\left(\frac{\mathbf{K}_{\mathcal{A}}\mathbf{q}_i^{\mathcal{B}}}{\sqrt{d}}\right) \tag{B.2}$$

where $\mathbf{q}_i^{\mathcal{K}} \in \mathbf{Q}_{\mathcal{K}}$, and $\mathbf{Q}_{\mathcal{K}}, \mathbf{K}_{\mathcal{K}} \in \mathbb{R}^{\mathbf{N}_{\mathcal{K}} \times d}$ are the query and key (respectively) for object $\mathcal{K}$ associated with cross-object attention Transformer module $g_{\mathcal{T}_{\mathcal{K}}}$, as shown in Figure B.3. These weights are then used to compute the virtual corresponding points $\mathbf{V}_{\mathcal{A}}$, $\mathbf{V}_{\mathcal{B}}$ using Equations 5a and 5b in the main paper.

B.3.1 Ablation

To explore the importance of this weight computation design choice described in Equation B.2, we conducted an ablation experiment on this design choice against an alternative, arguably simpler method for cross-object attention weight computation that was used in prior work [50]. Since the point embeddings $\boldsymbol{\phi}_i^{\mathcal{A}}$ and $\boldsymbol{\phi}_i^{\mathcal{B}}$ have the same dimension d, we can select the inner product of the space as a similarity metric between two embeddings.

To compute the virtual corresponding point $\mathbf{v}_i^{\mathcal{A}}$ assigned to any point $\mathbf{p}_i^{\mathcal{A}} \in \mathbf{P}^{\mathcal{A}}$, we can extract the desired normalized weight vector $\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}}$ with the softmax function:

$$\mathbf{w}_i^{\mathcal{A}\to\mathcal{B}} = \text{softmax}\left(\mathbf{\Phi}_{\mathcal{B}}^{\top}\boldsymbol{\phi}_i^{\mathcal{A}}\right), \quad \mathbf{w}_i^{\mathcal{B}\to\mathcal{A}} = \text{softmax}\left(\mathbf{\Phi}_{\mathcal{A}}^{\top}\boldsymbol{\phi}_i^{\mathcal{B}}\right) \tag{B.3}$$

This is the approach used in the prior work of Deep Closest Point (DCP) [50]. In the experiments below, we refer to this approach as *point embedding dot-product*.

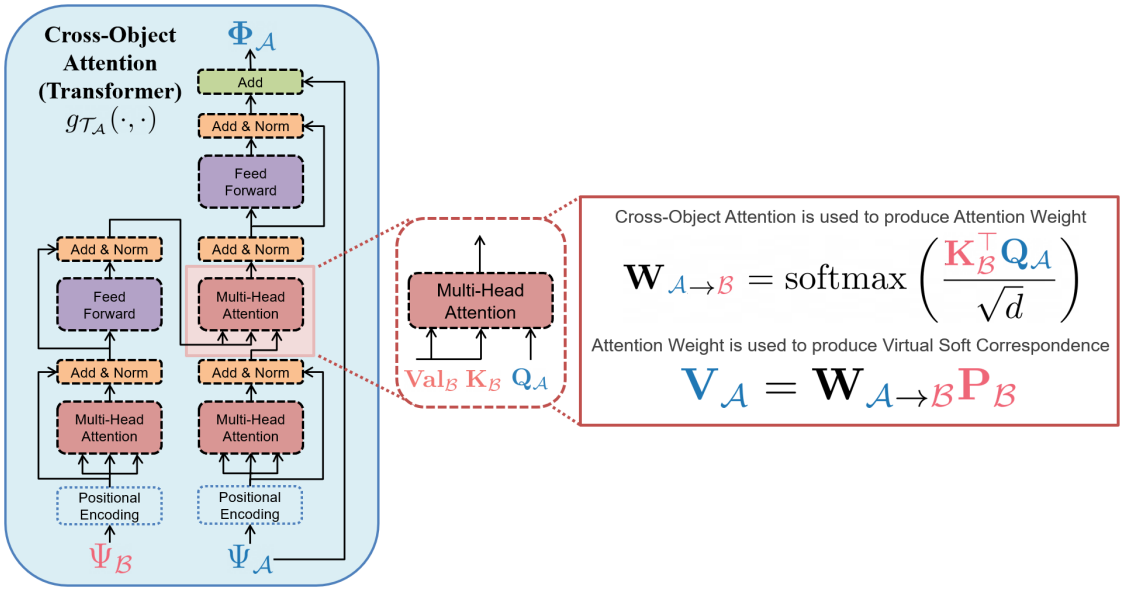


Figure B.3: Cross-Object attention weight computation for virtual soft correspondence $\mathbf{V}_\mathcal{A}$ from object $\mathcal{A}$ to $\mathcal{B}$. $\mathbf{Q}_\mathcal{K}, \mathbf{K}_\mathcal{K}, \mathbf{Val}_\mathcal{K} \in \mathbb{R}^{\mathbf{N}_\mathcal{K} \times d}$ are the query, key and value (respectively) for object $\mathcal{K}$ associated with cross-object attention Transformer module $g_{\mathcal{T}_\mathcal{K}}$. The Transformer block is modified from Figure 2(b) in DCP [50].

We conducted an ablation experiment on the weight computation method used in TAX-Pose (Equation B.2) against the simpler approach from DCP [50] (Equation B.3), on the upright mug hanging task in simulation. The models are trained from 10 demonstrations and tested on 100 trials over the test mug set. As seen in Table B.1, the TAX-Pose approach (Equation B.2) outperforms *point embedding dot-product* (Equation B.3) in all three evaluation categories on *grasp*, *place*, and *overall* in terms of test success rate.

Attention Weight Ablation	**Grasp**	**Place**	**Overall**
Point Embedding Dot-Product (Eqn. B.3)	0.83	0.92	0.92
TAX-Pose (Ours) (Eqn. B.2)	**0.99**	**0.97**	**0.96**

Table B.1: Test success rate (↑) over 100 trials for mug hanging upright task, ablated on attention weight computation methods.

B.4 Description of Weighted SVD

The objective function for computing the optimal rotation and translation given a set of correspondences for object $\mathcal{K}$, $\{\mathbf{p}_i^k \rightarrow \tilde{\mathbf{v}}_i^k\}_i^{N_k}$ and weights $\{\alpha_i^k\}_i^{N_k}$, is as follows:

$$\mathcal{J}(\mathbf{T}_{\mathcal{AB}}) = \sum_{i=1}^{N_{\mathcal{A}}} \alpha_i^{\mathcal{A}} ||\mathbf{T}_{\mathcal{AB}}\ \mathbf{p}_i^{\mathcal{A}} - \tilde{\mathbf{v}}_i^{\mathcal{A}}||_2^2 + \sum_{i=1}^{N_{\mathcal{B}}} \alpha_i^{\mathcal{B}} ||\mathbf{T}_{\mathcal{AB}}^{-1}\ \mathbf{p}_i^{\mathcal{B}} - \tilde{\mathbf{v}}_i^{\mathcal{B}}||_2^2$$

First we center (denoted with $*$) the point clouds and virtual points independently, with respect to the learned weights, and stack them into frame-specific matrices (along with weights) retaining their relative position and correspondence:

$$\mathbf{A} = \begin{bmatrix} \mathbf{P}_{\mathcal{A}}^{*\top} & \tilde{\mathbf{V}}_{\mathcal{B}}^{*\top} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \tilde{\mathbf{V}}_{\mathcal{A}}^{*\top} & \mathbf{P}_{\mathcal{B}}^{*\top} \end{bmatrix}^{\top}, \quad \mathbf{\Gamma} = \text{diag}\left(\begin{bmatrix} \boldsymbol{\alpha}_{\mathcal{A}} & \boldsymbol{\alpha}_{\mathcal{B}} \end{bmatrix} \right)$$

Then the minimizing rotation $\mathbf{R}_{\mathcal{AB}}$ is given by:

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top} = \text{svd}(\mathbf{A}\mathbf{\Gamma}\mathbf{B}^{\top}) \qquad \text{(B.4a)} \qquad\qquad \mathbf{R}_{\mathcal{AB}} = \mathbf{U}\mathbf{\Sigma}_{*}\mathbf{V}^{\top} \qquad \text{(B.4b)}$$

where $\mathbf{\Sigma}_{*} = \text{diag}(\begin{bmatrix} 1, 1, ...\det(\mathbf{U}\mathbf{V}^{\top}) \end{bmatrix}$ and svd is a differentiable SVD operation [39].

The optimal translation can be computed as:

$$\mathbf{t}_{\mathcal{A}} = \bar{\bar{\mathbf{v}}}_{\mathcal{B}} - \mathbf{R}_{\mathcal{AB}}\bar{\mathbf{p}}_{\mathcal{A}} \qquad \mathbf{t}_{\mathcal{B}} = \bar{\mathbf{p}}_{\mathcal{B}} - \mathbf{R}_{\mathcal{AB}}\bar{\bar{\mathbf{v}}}_{\mathcal{A}} \qquad \mathbf{t} = \frac{N_{\mathcal{A}}}{N}\mathbf{t}_{\mathcal{A}} + \frac{N_{\mathcal{B}}}{N}\mathbf{t}_{\mathcal{B}} \tag{B.5a}$$

with $N = N_{\mathcal{A}} + N_{\mathcal{B}}$. In the special translation-only case, the optimal translation and be computed by setting $\mathbf{R}_{\mathcal{AB}}$ to identity in above equations. The final transform can be assembled:

$$\mathbf{T}_{\mathcal{AB}} = \begin{bmatrix} \mathbf{R}_{\mathcal{AB}} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{B.6}$$

B.5 Training Details

B.5.1 Supervision

To train the encoders $g_{\mathcal{A}}(\bar{\mathbf{P}}_{\mathcal{A}})$, $g_{\mathcal{B}}(\bar{\mathbf{P}}_{\mathcal{B}})$ as well as the residual networks $g_{\mathcal{R}_{\mathcal{A}}}\left(\boldsymbol{\phi}_i^{\mathcal{A}}\right)$, $g_{\mathcal{R}_{\mathcal{B}}}\left(\boldsymbol{\phi}_i^{\mathcal{B}}\right)$, we use a set of losses defined below. We assume we have access to a set of demonstrations of the task, in which the action and anchor objects are in the target relative pose such that $\mathbf{T}_{\mathcal{AB}} = \mathbf{I}$.

Point Displacement Loss [28, 54]: Instead of directly supervising the rotation and translation (as is done in DCP), we supervise the predicted transformation using its effect on the points. For this loss, we take the point clouds of the objects in the demonstration configuration, and transform each cloud by a random transform, $\hat{\mathbf{P}}_{\mathcal{A}} = \mathbf{T}_{\alpha}\mathbf{P}_{\mathcal{A}}$, and $\hat{\mathbf{P}}_{\mathcal{B}} = \mathbf{T}_{\beta}\mathbf{P}_{\mathcal{B}}$. This would give us a ground truth transform of $\mathbf{T}_{\mathcal{AB}}^{GT} = \mathbf{T}_{\beta}\mathbf{T}_{\alpha}^{-1}$; the inverse of this transform would move object $\mathcal{B}$ to the correct position relative to object $\mathcal{A}$. Using this ground truth transform, we compute the MSE loss between the correctly transformed points and the points transformed using our prediction.

$$\mathcal{L}_{\text{disp}} = \left\|\mathbf{T}_{\mathcal{AB}}\mathbf{P}_{\mathcal{A}} - \mathbf{T}_{\mathcal{AB}}^{GT}\mathbf{P}_{\mathcal{A}}\right\|^2 + \left\|\mathbf{T}_{\mathcal{AB}}^{-1}\mathbf{P}_{\mathcal{B}} - \mathbf{T}_{\mathcal{AB}}^{GT-1}\mathbf{P}_{\mathcal{B}}\right\|^2 \tag{B.7}$$

Direct Correspondence Loss. While the Point Displacement Loss best describes errors seen at inference time, it can lead to correspondences that are inaccurate but whose errors average to the correct pose. To improve these errors we directly

supervise the learned correspondences $\tilde{V}_{\mathcal{A}}$ and $\tilde{V}_{\mathcal{B}}$:

$$\mathcal{L}_{\text{corr}} = \left\| \tilde{\mathbf{V}}_{\mathcal{A}} - \mathbf{T}_{\mathcal{AB}}^{GT} \mathbf{P}_{\mathcal{A}} \right\|^2 + \left\| \tilde{\mathbf{V}}_{\mathcal{B}} - \mathbf{T}_{\mathcal{AB}}^{GT-1} \mathbf{P}_{\mathcal{B}} \right\|^2. \tag{B.8}$$

Correspondence Consistency Loss. Furthermore, a consistency loss can be used. This loss penalizes correspondences that deviate from the final predicted transform. A benefit of this loss is that it can help the network learn to respect the rigidity of the object, while it is still learning to accurately place the object. Note, that this is similar to the Direct Correspondence Loss, but uses the predicted transform as opposed to the ground truth one. As such, this loss requires no ground truth:

$$\mathcal{L}_{\text{cons}} = \left\| \tilde{\mathbf{V}}_{\mathcal{A}} - \mathbf{T}_{\mathcal{AB}} \mathbf{P}_{\mathcal{A}} \right\|^2 + \left\| \tilde{\mathbf{V}}_{\mathcal{B}} - \mathbf{T}_{\mathcal{AB}}^{-1} \mathbf{P}_{\mathcal{B}} \right\|^2. \tag{B.9}$$

Overall Training Procedure. We train with a combined loss $\mathcal{L}_{\text{net}} = \mathcal{L}_{\text{disp}} + \lambda_1 \mathcal{L}_{\text{corr}} + \lambda_2 \mathcal{L}_{\text{cons}}$, where λ_1 and λ_2 are hyperparameters. We use a similar network architecture as DCP [50], which consists of DGCNN [51] and a Transformer [48].

In order to quickly adapt to new tasks, we optionally pre-train the DGCNN embedding networks over a large set of individual objects using the InfoNCE loss [36] with a geometric distance weighting and random transformations, to learn $SE(3)$ invariant embeddings, see Appendix E.2 for details.

B.5.2 Pretraining

We utilize pretraining for the embedding network for the mug hanging task, and describe the details below.

We pretrain embedding network for each object category (mug, rack, gripper), such that the embedding network is $SE(3)$ *invariant* with respect to the point clouds of that specific object category. Specifically, the mug-specific embedding network is pretrained on 200 ShapeNet [6] mug instances, while the rack-specific and gripper-specific embedding network is trained on the same rack and Franka gripper used at test time, respectively. Note that before our pretraining, the network is randomly initialized with the Kaiming initialization scheme [18]; we don't adopt any third-party pretrained models.

For the network to be trained to be $SE(3)$ invariant, we pre-train with InfoNCE

loss [36] with a geometric distance weighting and random $SE(3)$ transformations. Specifically, given a point cloud of an object instance, $\mathbf{P}_{\mathcal{A}}$, of a specific object category $\mathcal{A}$, and an embedding network $g_{\mathcal{A}}$, we define the point-wise embedding for $\mathbf{P}_{\mathcal{A}}$ as $\Phi_{\mathcal{A}} = g_{\mathcal{A}}(\mathbf{P}_{\mathcal{A}})$, where $\phi_i^{\mathcal{A}} \in \Phi_{\mathcal{A}}$ is a d-dimensional vector for each point $p_i^{\mathcal{A}} \in \mathbf{P}_{\mathcal{A}}$. Given a random $SE(3)$ transformation, $\mathbf{T}$, we define $\Psi_{\mathcal{A}} = g_{\mathcal{A}}(\mathbf{T}\mathbf{P}_{\mathcal{A}})$, where $\psi_i^{\mathcal{A}} \in \Psi_{\mathcal{A}}$ is the d-dimensional vector for the ith point $p_i^{\mathcal{A}} \in \mathbf{P}_{\mathcal{A}}$.

The weighted contrastive loss used for pretraining, $\mathcal{L}_{wc}$, is defined as

$$\mathcal{L}_{wc} := -\sum_i \log \left[\frac{\exp\left(\phi_i^\top \psi_i\right)}{\sum_j \exp\left(d_{ij}\left(\phi_i^\top \psi_j\right)\right)} \right] \tag{B.10}$$

$$d_{ij} := \begin{cases} \frac{1}{\mu}\tanh\left(\lambda \|p_i^{\mathcal{A}} - p_j^{\mathcal{A}}\|_2\right), & \text{if } i \neq j \\ 1, & \text{otherwise} \end{cases} \tag{B.11}$$

$$\mu := \max\left(\tanh\left(\lambda \|p_i^{\mathcal{A}} - p_j^{\mathcal{A}}\|_2\right)\right) \tag{B.12}$$

For this pretraining, we use $\lambda := 10$.

B.5.3 Architectural Variants

Goal-Conditioned TAX-Pose: To enable a single TAX-Pose model to scale to multiple related placement sub-tasks for a pair of action and anchor objects, we design a **goal-conditioned** variant (**TAX-Pose GC**), which receives a one-hot encoding of the desired semantic goal position (e.g. 'top', 'left', ...) for the task. This contextual encoding is incorporated into each DGCNN module in the same way as proposed in the original DGCNN paper. This encoding can be used to provide an embedding of the specific placement relationship that is desired in a scene (e.g. selecting a "top" vs. "left" placement position) and thus enable goal conditioned placement.

Vector Neurons: We briefly experimented with Vector Neurons [12] and found that this led to worse performance on this task.

B.6 Additional Results

B.6.1 NDF Placement Tasks

Further Ablations on Mug Hanging Task

In order to examine the effects of different design choices in the training pipeline, we conduct ablation experiments with final task-success (*grasp*, *place*, *overall*) as evaluation metrics for Mug Hanging task with upright pose initialization for the following components of our method, see Table B.2 for full ablation results along six ablated dimensions as detailed below. We also performed an ablation experiment on alternative cross-object attention weight computation, as explained in Appendix B.3 and results can be found in Table B.1. For consistency, all ablated models are trained for 15K steps.

1. **Loss.** In the full pipeline reported, we use a weighted sum of the three types of losses described in **Section 4.2** of the paper. Specifically, the loss used $\mathcal{L}_{\text{net}}$ is given by

$$\mathcal{L}_{\text{net}} = \mathcal{L}_{\text{disp}} + \lambda_1 \mathcal{L}_{\text{cons}} + \lambda_2 \mathcal{L}_{\text{corr}} \tag{B.13}$$

   where we chose $\lambda_1 = 0.1$, $\lambda_2 = 1$ after hyperparameter search.

   We ablate usage of all three types of losses, by reporting the final task performance in simulation for all experiments, specifically, we report task success on the following $\mathcal{L}_{\text{net}}$ variants.

   (a) Remove the point displacement loss term, $\mathcal{L}_{\text{disp}}$, after which we are left with

$$\mathcal{L}'_{\text{net}} = (0.1)\mathcal{L}_{\text{cons}} + \mathcal{L}_{\text{corr}}$$

   (b) Remove the direct correspondence loss term, $\mathcal{L}_{\text{corr}}$, after which we are left with

$$\mathcal{L}'_{\text{net}} = \mathcal{L}_{\text{disp}} + (0.1)\mathcal{L}_{\text{cons}}$$

   (c) Remove the correspondence consistency loss term, $\mathcal{L}_{\text{cons}}$, after which we are left with

$$\mathcal{L}'_{\text{net}} = \mathcal{L}_{\text{disp}} + \mathcal{L}_{\text{corr}}$$

(d) From testing loss variants above, we found that the point displacement loss is a vital contributing factor for task success, where removing this loss term results in no overall task success, as shown in Table B.2. However, in practice, we have found that adding the correspondence consistency loss and direct correspondence loss generally help to lower the rotational error of predicted placement pose compared to the ground truth of collected demos. To further investigate the effects of the combination of these two loss terms, we used a scaled weighted combination of $\mathcal{L}_{\text{cons}}$ and $\mathcal{L}_{\text{corr}}$, such that the former weight of the displacement loss term is transferred to consistency loss term, with the new $\lambda_1 = 1.1$, and with $\lambda_2 = 1$ stays unchanged. Note that this is different from variant (a) above, as now the consistency loss given a comparable weight with dense correspondence loss term, which intuitively makes sense as the consistency loss is a function of the predicted transform $\mathbf{T}_{\mathcal{AB}}$ to be used, while the dense correspondence loss is instead a function of the ground truth transform, $\mathbf{T}^{GT}_{\mathcal{AB}}$, which provides a less direct supervision on the predicted transforms. Thus we are left with

$$\mathcal{L}'_{\text{net}} = (1.1)\mathcal{L}_{\text{cons}} + \mathcal{L}_{\text{corr}}$$

2. **Usage of Correspondence Residuals.** After predicting a per-point soft correspondence between objects $\mathcal{A}$ and $\mathcal{B}$, we adjust the location of the predicted corresponding points by further predicting a point-wise correspondence residual vector to displace each of the predicted corresponding point. This allows the predicted corresponding point to get mapped to free space outside of the convex hulls of points in object $\mathcal{A}$ and $\mathcal{B}$. This is a desirable adjustment for mug hanging task, as the desirable cross-pose usually require points on the mug handle to be placed somewhere near but not in contact with the mug rack, which can be outside of the convex hull of rack points. We ablate correspondence residuals by directly using the soft correspondence prediction to find the cross-pose transform through weighted SVD, without any correspondence adjustment via correspondence residual.
3. **Weighted SVD vs Non-weighted SVD.** We leverage weighted SVD as described in **Section 4.1** of the paper as we leverage predicted per-point weight

to signify the importance of specific correspondence. We ablate the use of weighted SVD, and we use an un-weighted SVD, where instead of using the predicted weights, each correspondence is assign equal weights of $\frac{1}{N}$, where N is the number of points in the point cloud $\mathbf{P}$ used.

4. **Pretraining.** In our full pipeline, we pretrain the point cloud embedding network such that the embedding network is $SE(3)$ invariant. Specifically, the mug-specific embedding network is pretrained on 200 ShapeNet mug objects, while the rack-specific and gripper specific embedding network is trained on the same rack and Franka gripper used at test time, respectively. We conduct ablation experiments where
   (a) We omit the pretraining phase of embedding network
   (b) We do not finetune the embedding network during downstream training with task-specific demonstrations.

   Note that in practice, we find that pretraining helps speed up the downstream training by about a factor of 3, while models with or without pretraining both reach a similar final performance in terms of task success after both models converge.
5. **Usage of Transformer as Cross-object Attention Module.** In the full pipeline, we use transformer as the cross-object attention module, and we ablate this design choice by replacing the transformer architecture with a simple 3-layer MLP with ReLU activation and hidden dimension of 256, and found that this leads to worse place and grasp success.
6. **Dimension of Embedding.** In the full pipeline, the embedding is chosen to be of dimension 512. We conduct experiment on much lower dimension of 16, and found that with dimension =16, the place success is much lower, dropped from 0.97 to 0.59.

Ablation Experiment	Grasp	Place	Overall
No $\mathcal{L}_{\text{disp}}$	0.01	0	0
No $\mathcal{L}_{\text{corr}}$	0.89	0.91	0.84
No $\mathcal{L}_{\text{cons}}$	**0.99**	0.95	0.94
Scaled Combination: $1.1\mathcal{L}_{\text{cons}} + \mathcal{L}_{\text{corr}}$	0.10	0.01	0.01
No Adjustment via Correspondence Residuals	0.97	0.96	0.93
Unweighted SVD	0.92	0.94	0.88
No Finetuning for Embedding Network	0.98	0.93	0.91
No Pretraining for Embedding Network	**0.99**	0.72	0.71
3-Layer MLP In Place of Transformer	0.90	0.82	0.76
Embedding Network Feature Dim = 16	0.98	0.59	0.57
TAX-Pose (Ours)	**0.99**	**0.97**	**0.96**

Table B.2: Mug Hanging Ablations Results

Effects of Pretraining on Mug Hanging Task

We explore the effects of pretraining on the final task performance, as well as training convergence speed. We have found that pretraining the point cloud embedding network as described in B.5.2, is a helpful but not necessary component in our training pipeline. Specifically, we find that while utilizing pretraining reduces training time, allowing the model to reach similar task performance and train rotation/translation error with much fewer training steps, this component is not necessary if training time is not of concern. In fact, as see in Table B.3, we find that for mug hanging tasks, by training the models from scratch without our pretraining, the models are able to reach similar level of task performance of 0.99 grasp, 0.92 for place and 0.92 for overall success rate. Furthermore, it is able to achieve similar level of train rotation error of 4.91° and translation error of $0.01m$, compared to the models with pretraining. However, without pre-trainig, the model needs to be trained for about 2 times longer (26K steps compared to 15K steps) to reach the similar level of performance. Thus we adopt our object-level pretraining in our overall pipeline to allow lower training time.

Another benefit of pretraining is that the pretraining for each object category

is done in a task-agnostic way, so the network can be more quickly adapted to new tasks after the pretraining is performed. For example, we use the same pre-trained mug embeddings for both the gripper-mug cross-pose estimation for grasping as well as the mug-rack cross-pose estimation for mug hanging.

Ablation Experiment	Grasp	Place	Overall	Train Rotation Error (°)	Train Translation Error (m)
No Pre-Training for Embedding Network (trained for 26K steps)	**0.99**	0.92	0.92	4.91	**0.01**
No Pre-training for Embedding Network (trained for 15K steps)	**0.99**	0.72	0.71	15.39	**0.01**
TAX-Pose (Ours) (trained for 15K steps)	**0.99**	**0.97**	**0.96**	**4.33**	**0.01**

Table B.3: Ablation Experiments on the Effects of Pre-Training. We report the task success rate for upright mug hanging task over 100 trials each, as well as the grasping model's training rotational error (°) and translation error (m).

Additional Simulation Experiments on Bowl and Bottle Placement Task

Object	Algorithm	Grasp	Place	Overall	Grasp	Place	Overall
		Upright Pose			Arbitrary Pose		
Mug	DON	0.91	0.50	0.45	0.35	0.45	0.17
	NDF	0.96	0.92	0.88	**0.78**	0.75	0.58
	TAX-Pose (Ours)	**0.99**	**0.97**	**0.96**	0.75	**0.84**	**0.63**
Bowl	DON	0.50	0.35	0.11	0.08	0.20	0
	NDF	0.91	**1**	0.91	**0.79**	**0.97**	0.78
	TAX-Pose (Ours)	**0.99**	0.92	**0.92**	0.74	0.85	**0.85**
Bottle	DON	0.79	0.24	0.24	0.05	0.02	0.01
	NDF	**0.87**	**1**	**0.87**	**0.78**	**0.99**	**0.77**
	TAX-Pose (Ours)	0.55	0.99	0.55	0.61	0.55	0.52

Table B.4: Unseen Object Instance Manipulation Task Success Rates (↑) in Simulation on *Mug*, *Bowl* and *Bottle* for Upright and Arbitrary Initial Pose. Each result is the success rate over 100 trials.

Additional results on *Grasp*, *Place* and *Overall* success rate in simulation for **Bowl** and **Bottle** are shown in Table B.4. For bottle and bowl experiment, we follow the same experimentation setup as in [45], where the successful *grasp* is considered if a stable grasp of the object is obtained, and a successful *place* is considered when the bottle or bowl is stably placed upright on the elevated flat slab over the table without

falling on the table. Reported task success results in are for both *Upright Pose* and *Arbitrary Pose* run over 100 trials each.

Unlike mugs, bowls and bottles exhibit rotational object symmetry, which we have found cause the trained model to perform poorly in the *Grasp* task. To mitigate this, we applied symmetry breaking techniques for the **Bowl** and **Bottle** placement tasks by algorithmically creating symmetry labels, $l_i^{\mathcal{K}} \in [-1, 1]$ for object $\mathcal{K}$, as continuous real numbers between -1 and 1 inclusive for each point $\mathbf{p}_i^{\mathcal{K}}$ during training and testing. The symmetry label for i-th point is concatenated with the associated point-wise embedding, $\psi_i^{\mathcal{K}}$, resulting in an augmented point-wise embedding, $\hat{\psi}_i^{\mathcal{K}} := \begin{bmatrix} \psi_i^{\mathcal{K}} & l_i^{\mathcal{K}} \end{bmatrix}^\top \in \mathbb{R}^{d+1}$, which is then passed into the Cross-Correspondence Estimators. The input layer of these estimators are modified to account for the corresponding new input dimension.

The symmetry labels for each object are generated using a easily computed bisecting plane. Given segmented point cloud demonstration of the gripper and the bottle/bowl in goal configuration, we apply Principle Component Analysis (PCA) to the individual object point cloud of the gripper and bottle/bowl.

The symmetry labels for the gripper are defined using the PCA vector with largest principal component positioned at the gripper point cloud centroid, $\hat{\mathbf{s}}_{gripper}$. This vector defines the plane that bisects the gripper along its axis of actuation.

For each point $\mathbf{p}_i^{gripper}$ in the gripper point cloud, compute the unit vector between it and gripper centroid $\mu_{gripper}$,

$$\hat{\mathbf{v}}_i^{gripper} = \frac{\mathbf{p}_i^{gripper} - \mu_{gripper}}{\|\mathbf{p}_i^{gripper} - \mu_{gripper}\|}, \tag{B.14}$$

and retrieve the symmetry labels as the dot-product between the unit vectors,

$$l_i^{gripper} = \left\langle \mathbf{s}_{gripper}, \hat{\mathbf{v}}_i^{gripper} \right\rangle. \tag{B.15}$$

To compute the symmetry labels for for the bottle/bowl point clouds at training time, we retrieve the rotational symmetry axis, $\hat{\mathbf{v}}_{rot}$ of the bottle/bowl (pointing upward towards the bottle/bowl opening) using PCA. This vector is the largest principal component for the bottles and the smallest component for the bowls. We define a bisecting plane using both this symmetry axis, as well as the normalized

vector pointing from the centroid of bottle/bowl to the centroid of the gripper, $\hat{\mathbf{v}}_{gripper}$. For the bottle points, the normal of the bisecting plane is found using the normalize cross product of these two vectors,

$$\mathbf{s}_{bottle} = \frac{\hat{\mathbf{v}}_{rot} \times \hat{\mathbf{v}}_{gripper}}{\|\hat{\mathbf{v}}_{rot} \times \hat{\mathbf{v}}_{gripper}\|}, \tag{B.16}$$

which separates the bottle into left and right sides with respect to the gripper. For the bowl points, we orthoganalize the gripper vector, $\hat{\mathbf{v}}_{gripper}$ to symmetry vector, $\hat{\mathbf{v}}_{rot}$,

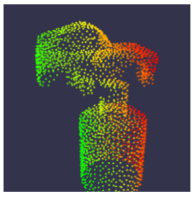
$$\mathbf{s}_{bowl} = \frac{\hat{\mathbf{v}}_{gripper} - \langle \hat{\mathbf{v}}_{gripper}, \hat{\mathbf{v}}_{rot} \rangle \hat{\mathbf{v}}_{rot}}{\|\hat{\mathbf{v}}_{gripper} - \langle \hat{\mathbf{v}}_{gripper}, \hat{\mathbf{v}}_{rot} \rangle \hat{\mathbf{v}}_{rot}\|}. \tag{B.17}$$

This results in a bisecting plane that seperates the bowl into a near and far half with respect to the gripper. Similar the the gripper symmetry labels, the symmetry labels for the bottle/bowl are computed using normalized vector between each bottle/bowl point and the bottle/bowl centroid, $\hat{\mathbf{v}}_i^{\{bottle,bowl\}}$,

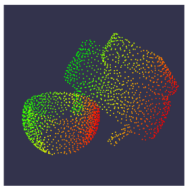
$$l_i^{\{bottle,bowl\}} = \left\langle \mathbf{s}_{\{bottle,bowl\}}, \hat{\mathbf{v}}_i^{\{bottle,bowl\}} \right\rangle. \tag{B.18}$$

At inference, instead of using the gripper location to compute $\mathbf{v}_{gripper}$ for the bottle and bowl labels, we use a random vector perpendicular to $\mathbf{v}_{rot}$. This allows us to use semantically meaningful symmetry labels at training time, and then arbitrarily break the symmetry at test time.

See Figure B.4 for visualization of symmetry labels obtained from aforementioned procedures, where the color spectrum of red represents symmetry label of 1, and green represents -1.



(a) Symmetry labels for bottle and gripper



(b) Symmetry labels for bowl and gripper

Figure B.4: Symmetry breaking

Failure Cases

Some failure cases for TAX-Pose occur when the predicted gripper misses the rim of the mug by a xy-plane translation error, thus resulting in failure of grasp, as seen in Figure B.5. A common failure mode for the mug placement subtask is characterized by an erroneous transform prediction that results in the mug's handle completely missing the rack hanger, thus resulting in placement failure, as seen in Figure B.5.

(a) Failure of *grasp* prediction. Predicted TAX-Pose for the gripper misses the rim of mug.

(b) Failure of *place* prediction. Predicted TAX-Pose for mug results in the mug handle misses the rack hanger completely.

Figure B.5: An illustration of unsuccessful TAX-Pose predictions for mug hanging. In both subfigures, red points represent the anchor object, blue points represent action object's starting pose, and green points represent action object's predicted pose.

B.6.2 PartNet-Mobility Tasks

Expanded Results Tables

In the main text, we presented aggregated results of the performance of each method by averaging the quantitative metrics for each sub-task for each object ("In", "On", "Left", and "Right" in simulation and "In", "On" and "Left" in real-world), and then averaged across object classes to arrive at a single metric per method. Here, we present the per-class breakdown of performance. See Table B.5 for simulated results, and Table B.6 for real-world results.

		AVG.																	
		$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$
Baselines	E2E BC	42.26	0.73	37.82	0.82	37.15	0.65	44.84	0.68	30.69	1.06	40.38	0.69	45.09	0.76	45.00	0.79	45.65	0.64
	E2E DAgger	37.96	0.69	34.15	0.76	36.61	0.66	40.91	0.65	24.87	0.97	35.95	0.70	40.34	0.74	32.86	0.79	39.45	0.53
Ablations	Traj. Flow	35.95	0.67	31.24	0.82	39.21	0.72	34.35	0.66	28.48	0.75	37.14	0.59	29.49	0.70	39.60	0.76	39.69	0.48
	Goal Flow	26.64	0.17	25.88	**0.15**	25.05	0.15	30.62	0.15	27.61	0.10	28.01	**0.18**	20.96	**0.24**	29.02	0.23	22.13	0.20
Ours	**TAX-Pose**	6.64	**0.16**	6.85	0.16	2.05	**0.10**	3.87	0.12	4.04	0.08	12.71	0.31	**6.87**	0.37	5.89	0.13	14.93	0.18
	TAX-Pose GC	**4.94**	**0.16**	**6.18**	0.16	**1.75**	**0.10**	**2.94**	**0.10**	**3.02**	**0.06**	**10.15**	0.27	6.93	0.35	**3.76**	**0.11**	**4.76**	**0.11**

Table B.5: Goal Inference Rotational and Translational Error Results (↓). Rotational errors ($\mathcal{E}_\mathbf{R}$) are in degrees (°) and translational errors ($\mathcal{E}_\mathbf{t}$) are in meters (m). The lower the better.

	In			On			Left		
Goal Flow	0.00	0.10	0.30	0.05	N/A	0.20	0.50	0.65	0.60
TAX-Pose	**1.00**	**1.00**	**0.85**	**1.00**	N/A	**1.00**	**0.85**	**0.90**	**0.70**

Table B.6: Combined per-task results for real-world goal placement success rate.

We further provide results per-sub-task in simulation. For each category of anchor objects, sub-tasks may or may not all be well-defined. For example, the doors of safes might occlude the action object completely in a demonstration for "Left" and "Right" tasks due to the handedness of the door; and a table's height might be too tall for the camera to see the action object placed during the "Top" task. To avoid these ill-defined cases, we omit object-category / sub-task pairings which cannot be consistently defined from training and evaluation. We show visualizations of each defined task for each object category in Figure B.6. Results for each sub-task can be found in Tables B.7[1], B.8, B.9, and B.10 respectively.

[1]Categories from left to right: microwave, dishwasher, oven, fridge, table, washing machine, safe, drawer.

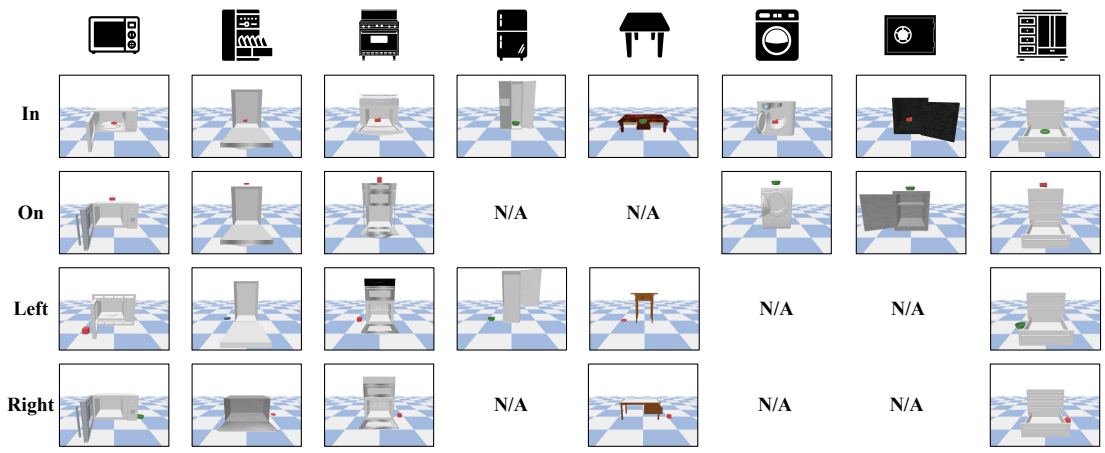


Figure B.6: A visualization of all categories of anchor objects and associated semantic tasks, with action objects in ground-truth TAX-Poses used in simulation training.

		AVG.															
		$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$
Baselines	E2E BC	42.37	0.69	40.49	0.80	50.79	0.59	48.02	0.61	30.69	1.09	36.59	0.81	48.48	0.42	41.42	0.84	42.49	0.37
	E2E DAgger	36.06	0.67	38.57	0.68	43.99	0.63	42.34	0.57	24.87	0.96	30.87	0.90	42.96	0.46	29.79	0.83	35.08	0.33
Ablations	Traj. Flow	34.48	0.65	35.39	0.85	43.42	0.63	35.51	0.60	28.26	0.80	27.67	0.68	25.91	0.44	43.59	0.82	36.05	0.36
	Goal Flow	27.49	**0.21**	25.41	**0.08**	31.07	0.13	27.05	0.27	27.80	0.11	29.02	**0.38**	19.22	**0.36**	31.56	0.18	**28.81**	**0.19**
Ours	**TAX-Pose**	**11.74**	0.23	**5.81**	0.11	**1.82**	**0.08**	**5.92**	**0.11**	**3.67**	**0.07**	**19.54**	0.41	**7.96**	0.63	**5.96**	**0.12**	43.27	0.33

Table B.7: Goal Inference Rotational and Translational Error Results (↓) for the "**In**" Goal. Rotational errors ($\mathcal{E}_\mathbf{R}$) are in degrees (°) and translational errors ($\mathcal{E}_\mathbf{t}$) are in meters (m). The lower the better.

		AVG.															
		$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$	$\mathcal{E}_R$	$\mathcal{E}_t$
Baselines	E2E BC	42.69	0.74	41.94	0.74	36.70	0.52	38.23	0.73	41.69	1.10	48.57	0.75	48.98	0.63		
	E2E DAgger	37.68	0.70	39.24	0.69	31.63	0.54	41.06	0.68	37.72	1.03	35.94	0.75	40.47	0.51		
Ablations	Traj. Flow	35.13	0.76	34.78	0.70	39.14	0.59	31.10	0.69	33.07	0.97	35.61	0.71	37.09	0.87		
	Goal Flow	22.10	0.20	27.82	0.26	20.43	**0.09**	34.66	0.10	22.71	0.12	26.48	0.27	0.48	0.32		
Ours	**TAX-Pose**	**4.45**	**0.12**	**4.21**	**0.12**	**2.29**	0.10	**2.73**	**0.09**	**5.77**	**0.10**	**5.81**	**0.13**	**5.89**	**0.19**		

Table B.8: Goal Inference Rotational and Translational Error Results (↓) for the "**On**" Goal. Rotational errors ($\mathcal{E}_\mathbf{R}$) are in degrees (°) and translational errors ($\mathcal{E}_\mathbf{t}$) are in meters (m). The lower the better.

		AVG.													
		$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$
Baselines	E2E BC	44.87	0.74	30.95	0.89	36.86	0.72	56.86	0.52	34.35	1.03	31.69	0.77	46.86	0.78
	E2E DAgger	41.32	0.68	31.40	0.84	38.49	0.73	47.64	0.51	36.47	0.99	27.72	0.73	39.83	0.51
Ablations	Traj. Flow	38.85	0.58	31.87	1.07	39.48	0.44	39.48	0.44	28.71	0.69	41.06	0.73	40.70	0.31
	Goal Flow	29.64	**0.10**	28.51	**0.10**	26.33	**0.08**	32.96	**0.07**	27.42	0.10	22.04	**0.09**	27.42	0.15
Ours	**TAX-Pose**	**6.02**	0.17	**12.73**	0.28	**1.59**	0.11	**2.91**	0.12	**4.41**	**0.08**	**12.12**	0.34	**6.38**	**0.12**

Table B.9: Goal Inference Rotational and Translational Error Results (↓) for the "**Left**" Goal. Rotational errors ($\mathcal{E}_\mathbf{R}$) are in degrees (°) and translational errors ($\mathcal{E}_\mathbf{t}$) are in meters (m). The lower the better.

		AVG.											
		$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$	$\mathcal{E}_\mathbf{R}$	$\mathcal{E}_\mathbf{t}$
Baselines	E2E BC	39.11	0.76	37.89	0.86	24.26	0.77	36.27	0.88	52.86	0.48	44.26	0.78
	E2E DAgger	36.80	0.73	27.40	0.84	32.31	0.74	32.61	0.82	49.27	0.46	42.40	0.78
Ablations	Traj. Flow	35.33	0.71	22.93	0.66	34.78	1.22	31.29	0.92	42.71	0.37	44.93	0.36
	Goal Flow	27.34	0.16	21.79	0.15	22.37	0.28	27.79	0.15	32.96	**0.07**	31.79	0.15
Ours	**TAX-Pose**	**4.33**	**0.13**	**4.64**	**0.14**	**2.48**	**0.11**	**3.91**	**0.15**	**6.47**	0.17	**4.17**	**0.08**

Table B.10: Goal Inference Rotational and Translational Error Results (↓) for the "**Right**" Goal. Rotational errors ($\mathcal{E}_\mathbf{R}$) are in degrees (°) and translational errors ($\mathcal{E}_\mathbf{t}$) are in meters (m). The lower the better.

Goal-Conditioned Variant

We train our goal-conditioned variant **TAX-Pose GC** (Appendix B.5.3) to predict the correct cross-pose across sub-tasks, incorporating a one-hot encoding of each sub-task (i.e. 'top', 'in', 'left', 'right') so the model can infer the desired semantic goal location. Importantly, as with the task-specific model (**TAX-Pose**), the **TAX-Pose GC** model is trained across **all PartNet-Mobility object categories**. We report the performance of the variants in Table B.5.

Failure Cases

Some failure cases for TAX-Pose occur when the predicted cross-pose does not respect the physical constraints in the scene. For example, as seen in Fig. B.7. TAX-Pose would fail when the prediction violates the physical constraints of the objects, which in this case the action object collides with the anchor object. In the real world, this would yield the robot unable to plan a correct path.

(a) Failure of "In" prediction. Predicted TAX-Pose violates the physical constraints by penetrating the oven base too much.

(b) Failure of "Left" prediction. Predicted TAX-Pose violates the physical constraints by being in collision with the leg of the drawer.

Figure B.7: An illustration of unsuccessful real-world TAX-Pose predictions. In both subfigures, red points represent the anchor object, blue points represent action object's starting pose, and green points represent action object's predicted pose.

B.7 Task Details

B.7.1 NDF Task Details

In this section, we describe the Mug Hanging task of the NDF Tasks and experiments in detail. The Mug Hanging task is consisted of two sub tasks: *grasp* and *place*. A *grasp* success is achieved when the mug is grasped stably by the gripper, while a *place* success is achieved when the mug is hung stably on the hanger of the rack. Overall mug hanging success is achieved when the predicted transforms enable both grasp and place success for the same trial. See Figure B.8 for a detailed breakdown of the mug hanging task in stages.

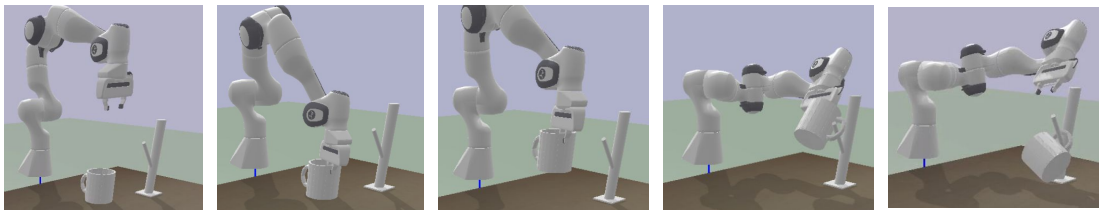


Figure B.8: Visualization of Mug Hanging Task (Upright Pose). Mug hanging task is consisted of two stages, given a mug that is randomly initialized on the table, the model first predicts a $SE(3)$ transform from gripper end effector to the mug rim $\mathbf{T}_{g \to m}$, then grasp it by the rim. Next, the model predicts another $SE(3)$ transform from the mug to the rack $\mathbf{T}_{m \to r}$ such that the mug handle gets hanged on the the mug rack.

Baseline Description

In simulation, we compare our method to the results described in [45].

- **Dense Object Nets (DON)** [16]: Using manually labeled semantic keypoints on the demonstration clouds, DON is used to compute sparse correspondences with the test objects. These correspondences are converted to a pose using SVD. A full description of usage of DON for the mug hanging task can be found in [45].
- **Neural Descriptor Field (NDF)** [45]: Using the learned descriptor field for the mug, the positions of a constellation of task specific query points are optimized to best match the demonstration using gradient descent.

Training Data

To be directly comparable with the baselines we compared to, we use the exact same sets of demonstration data used to train the network in NDF [45], where the data are generated via teleportation in PyBullet, collected on 10 mug instances with random pose initialization.

Training and Inference

Using the pretrained embedding network for mug and gripper, we train a grasping model for the grasping task to predict a transformation $\mathbf{T}_{g \to m}$ in gripper's frame from gripper to mug to complete the *grasp* stage of the task. Similarly, using the pretrained

embedding network for rack and mug, we train a placement model for the placing task to predict a transformation $\mathbf{T}_{m \to r}$ in mug's frame from mug to rack to complete the *place* stage of the task. Both models are trained with the same combined loss $\mathcal{L}_{net}$ as described in the main paper. During inference, we simply use grasping model to predict the $\mathbf{T}_{g \to m}$ at test time, and placement model to predict $\mathbf{T}_{m \to r}$ at test time.

Motion Planning

After the model predicts a transformation $\mathbf{T}_{g \to m}$ and $\mathbf{T}_{m \to r}$, using the known gripper's world frame pose, we calculate the desired gripper end effector pose at grasping and placement, and pass the end effector to IKFast to get the desired joint positions of Franka at grasping and placement. Next we pass the desired joint positions at gripper's initial pose, and desired grasping joint positions to OpenRAVE motion planning library to solve for trajectory from gripper's initial pose to grasp pose, and then grasp pose to placement pose for the gripper's end effector.

Real-World Experiments

We pre-train the DGCNN embedding network with rotation-equivariant loss on ShapeNet mugs' simulated point clouds in simulation. Using the pre-trained embedding, we then train the full TAX-Pose model with the 10 collected real-world point clouds.

B.7.2 PartNet-Mobility Object Placement Task Details

In this section, we describe the PartNet-Mobility Object Placement experiments in detail. We select a set of household furniture objects from the PartNet-Mobility dataset as the anchor objects, and a set of small rigid objects released with the Ravens simulation environment as the action objects. For each anchor object, we define a set of semantic goal positions (i.e. 'top', 'left', 'right', 'in'), where action objects should be placed relative to each anchor. Each semantic goal position defines a unique task in our cross-pose prediction framework.

Dataset Preparation

Simulation Setup. We leverage the PartNet-Mobility dataset [53] to find common household objects as the anchor object for TAX-Pose prediction. The selected subset of the dataset contains 8 categories of objects. We split the objects into 54 seen and 14 unseen instances. During training, for a specific task of each of the seen objects, we generate an action-anchor objects pair by randomly sampling transformations from $SE(3)$ as cross-poses. The action object is chosen from the Ravens simulator's rigid body objects dataset [59]. We define a subset of four tasks ("In", "On", "Left" and "Right") for each selected anchor object. Thus, there exists a ground-truth cross-pose (defined by human manually) associated with each defined specific task. We then use the ground-truth TAX-Poses to supervise each task's TAX-Pose prediction model. For each observation action-anchor objects pair, we sample 100 times using the aforementioned procedure for the training and testing datasets.

Real-World Setup. In real-world, we select a set of anchor objects: Drawer, Fridge, and Oven and a set of action objects: Block and Bowl. We test 3 ("In", "On", and "Left") TAX-Pose models in real-world without retraining or finetuning. The point here is to show the method capability of generalizing to unseen real-world objects.

Metrics

Simulation Metrics. In simulation, with access to the object's ground-truth pose, we are able to quantitatively calculate translational and rotation error of the TAX-Pose prediction models. Thus, we report the following metrics on a held-out set of anchor objects in simulation.

Translational Error

The L2 distance between the inferred cross-pose translation ($\mathbf{t}_{\mathcal{AB}}^{\text{pred}}$) and the ground-truth pose translation ($\mathbf{t}_{\mathcal{AB}}^{\text{GT}}$).

$$\mathcal{E}_{\mathbf{t}} = ||\mathbf{t}_{\mathcal{AB}}^{\text{pred}} - \mathbf{t}_{\mathcal{AB}}^{\text{GT}}||_2$$

Rotational Error

The geodesic $SO(3)$ distance [17, 23] between the predicted cross-pose rotation

($\mathbf{R}_{\mathcal{AB}}^{\text{pred}}$) and the ground-truth rotation ($\mathbf{R}_{\mathcal{AB}}^{\text{GT}}$).

$$\mathcal{E}_{\mathbf{R}} = \frac{1}{2} \arccos\left(\frac{\text{tr}(\mathbf{R}_{\mathcal{AB}}^{\text{pred}\top}\mathbf{R}_{\mathcal{AB}}^{\text{GT}}) - 1}{2}\right)$$

Real-World Metrics. In real-world, due to the difficulty of defining ground-truth TAX-Pose, we instead manually, qualitatively define goal "regions" for each of the anchor-action pairs. The goal-region should have the following properties:

- The predicted TAX-Pose of the action object should appear visually correct. For example, if the specified task is "In", then the action object should be indeed contained within the anchor object after being transformed by predicted TAX-Pose.
- The predicted TAX-Pose of the action object should not violate physical constraints of the workspace and of the relation between the action and anchor objects. Specifically, the action object should not interfere with/collide with the anchor object after being transformed by the predicted TAX-Pose. See Figure B.7 for an illustration of TAX-Pose predictions that fail to meet this criterion.

Motion Planning

In both simulated and real-world experiments, we use off-the-shelf motion-planning tools to find a path between the starting pose and goal pose of the action object.

Simulation. To actuate the action object from its starting pose $\mathbf{T}_0$ to its goal pose transformed by the predicted TAX-Pose $\hat{\mathbf{T}}_{\mathcal{AB}}\mathbf{T}_0$, we plan a path free of collision. Learning-based methods such as [11] deal with collision checking with point clouds by training a collision classifier. A more data-efficient method is by leveraging computer graphics techniques, transforming the point clouds into marching cubes [29], which can then be used to efficiently reconstruct meshes. Once the triangular meshes are reconstructed, we can deploy off-the-shelf collision checking methods such as FCL [38] to detect collisions in the planned path. Thus, in our case, we use position control to plan a trajectory of the action object $\mathcal{A}$ to move it from its starting pose to the predicted goal pose. We use OMPL [46] as the motion planning tool and the constraint function passed into the motion planner is from the output of FCL after converting the point clouds to meshes via marching cubes.

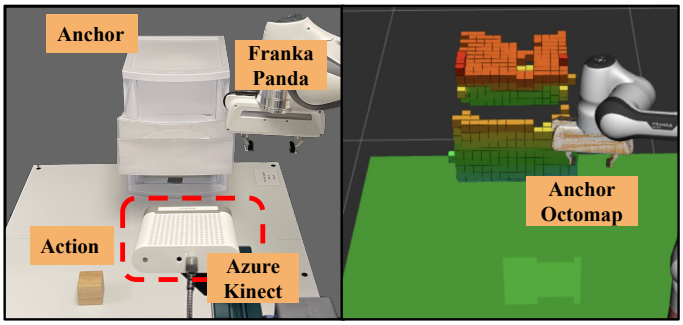


Figure B.9: Real-world experiments illustration. **Left:** work-space setup for physical experiments. **Center:** Octomap visualization of the perceived anchor object.

Real World. In real-world experiments, we need to resolve several practical issues to make TAX-Pose prediction model viable. First, we do not have access to a mask that labels action and anchor objects. Thus, we manually define a mask by using a threshold value of y-coordinate to automatically detect discontinuity in y-coordinates, representing the gap of spacing between action and anchor objects upon placement. Next, grasping action objects is a non-trivial task. Since, we are only using 2 action objects (a cube and a bowl), we manually define a grasping primitive for each action object. This is done by hand-picking an offset from the centroid of the action object before grasping, and an approach direction after the robot reaches the pre-grasp pose to make contacts with the object of interest. The offsets are chosen via kinesthetic teaching on the robot when the action object is under identity rotation (canonical pose). Finally, we need to make an estimation of the action's starting pose for motion planning. This is done by first statistically cleaning the point cloud [15] of the action object, and then calculating the centroid of the action object point cloud as the starting position. For starting rotation, we make sure the range of the rotation is not too large for the pre-defined grasping primitive to handle. Another implementation choice here is to use ICP [4] calculate a transformation between the current point cloud to a pre-scanned point cloud in canonical (identity) pose. We use the estimated starting pose to guide the pre-defined grasp primitive. Once a successful grasp is made, the robot end-effector is rigidly attached to the action object, and we can then use the same predicted TAX-Pose to calculate the end pose of the robot end effector, and thus feed the two poses into MoveIt! to get a full trajectory in joint space. Note here that the collision function in

motion planning is comprised of two parts: workspace and anchor object. That is, we first reconstruct the workspace using boxes to avoid collision with the table top and camera mount, and we then reconstruct the anchor object in RViz using Octomap [21] using the cleaned anchor object point cloud. In this way, the robot is able to avoid collision with the anchor object as well. See Figure B.9 for the workspace.

Baselines Description

In simulation, we compare our method to a variety of baseline methods.

- **E2E Behavioral Cloning**: Generate motion-planned trajectories using OMPL that take the action object from start to goal. These serve as "expert" trajectories for Behavioral Cloning (BC). Our policy is represented as a PointNet++ [43] network that, at each time step, takes as input the point cloud observation of the action and anchor objects and outputs an incremental 6-DOF transformation that imitates the expert trajectory. The 6-DoF transformation is expressed using Euclidean xyz translation and rotation quaternion. The final achieved pose of the action object at the terminal state is used for computing the evaluation metrics.
- **E2E DAgger**: Using the same BC dataset and the same PointNet++ [43] architecture as above, we train a policy that outputs the same transformation representation as in BC using DAgger [44]. The final achieved pose of the action object at the terminal state is used for computing the evaluation metrics.
- **Trajectory Flow**: Using the same BC dataset with DAgger, we train a dense policy using PointNet++ [43] to predict a dense per-point 3D flow vector at each time step instead of a single 6-DOF transformation. Given this dense per-point flow, we add the per-point flow to each point of the current time-step's point cloud, and we are able to extract a rigid transformation between the current point cloud and the point cloud transformed by adding per-point flow vectors using SVD, yielding the next pose. The final achieved pose of the action object at the terminal state is used for computing the evaluation metrics.
- **Goal Flow**: Instead of training a multi-step policy to reach the goal, we train a PointNet++ [43] network to output a single dense flow prediction which assigns a per-point 3D flow vector that points from each action object point

from its starting pose directly to its corresponding goal location. Given this dense per-point flow, we add the per-point flow to each point of the start point cloud, and we are able to extract a rigid transformation between the start point cloud and the point cloud transformed by adding per-point flow vectors using SVD, yielding goal pose. We pass the start and goal pose into a motion planner (OMPL) and execute the planned trajectory. The final achieved pose of the action object at the terminal state is used for computing the evaluation metrics.

Bibliography

[1] Artemij Amiranashvili, Alexey Dosovitskiy, Vladlen Koltun, and Thomas Brox. Motion perception in reinforcement learning with dynamic objects. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 156–168. PMLR, 2018. 2.2.2

[2] Yahav Avigal, Vishal Satish, Zachary Tam, Huang Huang, Harry Zhang, Michael Danielczuk, Jeffrey Ichnowski, and Ken Goldberg. Avplug: Approach vector planning for unicontact grasping amid clutter. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1140–1147. IEEE, 2021. 2.3.5

[3] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011. 2.2.1

[4] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992. B.7.2

[5] Felix Burget, Armin Hornung, and Maren Bennewitz. Whole-body motion planning for manipulation of articulated objects. In *2013 IEEE International Conference on Robotics and Automation*, pages 1656–1662, May 2013. 2.2.1

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 3.2.4, B.5.2

[7] Jae-Sook Cheong, A Frank Van Der Stappen, Ken Goldberg, Mark H Overmars, and Elon Rimon. Immobilizing Hinged Polygons. *Int. J. Comput. Geom. Appl.*, 17(01):45–69, February 2007. 2.1, 2.2.1

[8] Sachin Chitta, Benjamin Cohen, and Maxim Likhachev. Planning for autonomous door opening with a mobile manipulator. In *2010 IEEE International Conference*

on Robotics and Automation, pages 1799–1806, May 2010. 2.2.1

[9] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014. 2.3.5

[10] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016–2020. 3.3.1

[11] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017. IEEE, 2021. B.7.2

[12] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021. B.5.3

[13] Siyuan Dong, Devesh K Jha, Diego Romeres, Sangwoon Kim, Daniel Nikovski, and Alberto Rodriguez. Tactile-rl for insertion: Generalization to objects of unknown geometry. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6437–6443. IEEE, 2021. 2.2.2

[14] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks, 2015. 2.2.2

[15] Ben Eisner*, Harry Zhang*, and David Held. Flowbot3d: Learning 3d articulation flow to manipulate articulated objects. In *Robotics: Science and Systems (RSS)*, 2022. 2.1, 3.3.2, B.7.2

[16] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning*, pages 373–385. PMLR, 2018. 3.3.1, 3.3.1, 3.3.2, B.7.1

[17] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International journal of computer vision*, 103(3):267–305, 2013. 3.3.2, B.7.2

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 3.2.4, B.5.2

[19] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian*

conference on computer vision, pages 548–562. Springer, 2012. 3.2.3

[20] Berthold K P Horn and Brian G Schunck. Determining optical flow. *Artif. Intell.*, 17(1):185–203, August 1981. 2.2.2

[21] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013. B.7.2

[22] Ruizhen Hu, Wenchao Li, Oliver Van Kaick, Ariel Shamir, Hao Zhang, and Hui Huang. Learning to predict part mobility from a single static snapshot. *ACM Trans. Graph.*, 36(6):1–13, November 2017. 2.2.1

[23] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009. 3.3.2, B.7.2

[24] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017. 2.2.2

[25] Ajinkya Jain, Rudolf Lioutikov, Caleb Chuck, and Scott Niekum. ScrewNet: Category-Independent articulation model estimation from depth images using screw theory. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13670–13677, May 2021. 2.2.1

[26] Dov Katz, Yuri Pyuro, and Oliver Brock. Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Robotics: Science and Systems IV*. Robotics: Science and Systems Foundation, June 2008. 2.2.1

[27] Xiaolong Li, He Wang, Li Yi, Leonidas J Guibas, A Lynn Abbott, and Shuran Song. Category-Level articulated object pose estimation, 2020. 2.2.1

[28] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698, 2018. B.5.1

[29] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. B.7.2

[30] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning. In *2018 IEEE International Conference on robotics and automation (ICRA)*, pages 5620–5627. IEEE, 2018. 2.3.5

[31] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose,

Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26), 2019. 2.3.5

[32] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019. 2.2.1

[33] Kaichun Mo, Leonidas J Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6813–6823, 2021. 2.2.1

[34] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Learning-from-Demonstrations benchmark for generalizable manipulation skills. *arXiv e-prints*, pages arXiv–2107, 2021. 2.2.1, 2.4.1

[35] Venkatraman Narayanan and Maxim Likhachev. Task-oriented planning for manipulating articulated mechanisms under model uncertainty. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3095–3101, May 2015. 2.2.1

[36] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 3.2.3, 3.2.4, B.5.1, B.5.2

[37] Chuer Pan, Brian Okorn, Harry Zhang, Ben Eisner, and David Held. Tax-pose: Task-specific cross-pose estimation for robot manipulation. In *Conference on Robot Learning*, pages 1783–1792. PMLR, 2023. 3

[38] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012. B.7.2

[39] Théodore Papadopoulo and Manolis IA Lourakis. Estimating the jacobian of the singular value decomposition: Theory and applications. In *European Conference on Computer Vision*, pages 554–570. Springer, 2000. 3.2.2, B.4

[40] Anh Viet Phan, Minh Le Nguyen, Yen Lam Hoang Nguyen, and Lam Thu Bui. Dgcnn: A convolutional neural network over large-scale labeled graphs. *Neural Networks*, 108:533–543, 2018. (document), 3.4, 3.2.2

[41] Sudeep Pillai, Matthew R Walter, and Seth Teller. Learning articulated motions from visual demonstration. *arXiv preprint arXiv:1502.01659*, 2015. 2.2.2

[42] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep

learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 2.3.6

[43] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. B.7.2

[44] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 2.4.1, 3.3.2, B.7.2

[45] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400. IEEE, 2022. 3.2.1, 3.3, 3.3.1, 3.3.1, 3.3.2, B.6.1, B.7.1, B.7.1

[46] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. B.7.2

[47] Zachary Teed and Jia Deng. RAFT: Recurrent All-Pairs field transforms for optical flow. In *Computer Vision – ECCV 2020*, pages 402–419. Springer International Publishing, 2020. 2.2.2

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. (document), 3.4, 3.2.2, 3.2.3, B.3, B.5.1

[49] Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. Shape2motion: Joint analysis of motion parts and attributes from 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8876–8884, 2019. 2.2.1

[50] Yue Wang and Justin M Solomon. Deep closest point: Learning representations for point cloud registration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3523–3532, 2019. (document), 3.2.1, 3.2.3, B.3.1, B.3.1, B.3, B.5.1

[51] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 3.2.3, B.5.1

[52] Thomas Weng, Sujay Man Bajracharya, Yufei Wang, Khush Agrawal, and David Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In

Conference on Robot Learning, pages 192–202. PMLR, 2022. 2.2.2

[53] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, and Others. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11097–11107, 2020. 2.2.1, 2.3.6, 2.4.1, 2.4.3, 3.3.2, B.7.2

[54] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems (RSS)*, 2018. B.5.1

[55] Zhenjia Xu, He Zhanpeng, and Shuran Song. Umpnet: Universal manipulation policy network for articulated objects. *IEEE Robotics and Automation Letters*, 2022. 2.2.1, 2.4.1

[56] Zihao Yan, Ruizhen Hu, Xingguang Yan, Luanmin Chen, Oliver Van Kaick, Hao Zhang, and Hui Huang. Rpm-net: recurrent prediction of motion and parts from point cloud. *arXiv preprint arXiv:2006.14865*, 2020. 2.2.1

[57] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. Deep part induction from articulated object pairs. *arXiv preprint arXiv:1809.07417*, 2018. 2.2.1

[58] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021. 3.3.2

[59] Vicky Zeng, Timothy E Lee, Jacky Liang, and Oliver Kroemer. Visual identification of articulated object parts. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2443–2450. IEEE, 2020. 2.2.1, 2.3.2, B.7.2

[60] Harry Zhang, Jeffrey Ichnowski, Yahav Avigal, Joseph Gonzales, Ion Stoica, and Ken Goldberg. Dex-Net AR: Distributed deep grasp planning using a commodity cellphone and augmented reality app. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 552–558, May 2020. 2.3.5

[61] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 2.4.1