Appendix

Results are better shown visually in videos. Please refer to our website for video results.

Table of Contents

A Full FlowBot++ Manipulation Policy

Given an articulated object, we first observe an initial observation O_0, which is used to classify the object's articulation type. We then predict the initial flow f_0 and projection r_0, where f_0 is used to select a contact pose and grasp object. Then the system infers the articulation parameters based on Eq. 5 or 6 and follows the first H steps. This process repeats in a low-frequency if re-planning is needed 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 policy. The while loop runs in a much lower frequency compared to FlowBot3D, which further bypasses the potential error from heavy occlusions.

B Ablations

We document a variety of Ablation Studies in this section. Specifically, we investigate the effect of using different H values (i.e. replanning frequency), Gram-Schmidt Correction, and mean aggregation using part segmentation masks.

B.1 Controller H Values (Replanning Frequency)

H values represent how many steps of the interpolated trajectory we aim to execute after each prediction. Thus, it also represents the replanning frequency, where s higher H value means a lower replanning frequency, and vice versa. As shown in Fig. 7 and Table 2, when the replanning

Algorithm 1 The FlowBot++ articulation manipulation policy

Require: $\theta \leftarrow$ parameters of a trained flow-projection prediction network, $H \leftarrow$ controller look-ahead horizon, $\psi \leftarrow$ articulation type classifier parameters
$O_0 \leftarrow$ Initial observation
$\texttt{artType} \leftarrow f_\psi(O_0)$, Classify articulation type
$f_0, r_0 \leftarrow f_\theta(O_0)$, Predict the initial flow and projection
$g_0 = \texttt{SelectContact}(O_0, f_0)$, Select a contact pose and grasp object as shown in [6]
$done \leftarrow$ False
while not $done$ **do**
  $O_t \leftarrow$ Observation
  $f_t, r_t \leftarrow f_\theta(O_t)$, Predict the current articulation flow and articulation projection
  $\tau_t \leftarrow \texttt{TrajCalculation}(f_t, r_t)$, Calculate trajectory using Eq. 5 or 6 based on $\texttt{artType}$
  Follow the first H steps in τ_t (MPC)
  $done \leftarrow \texttt{EpisodeComplete}()$

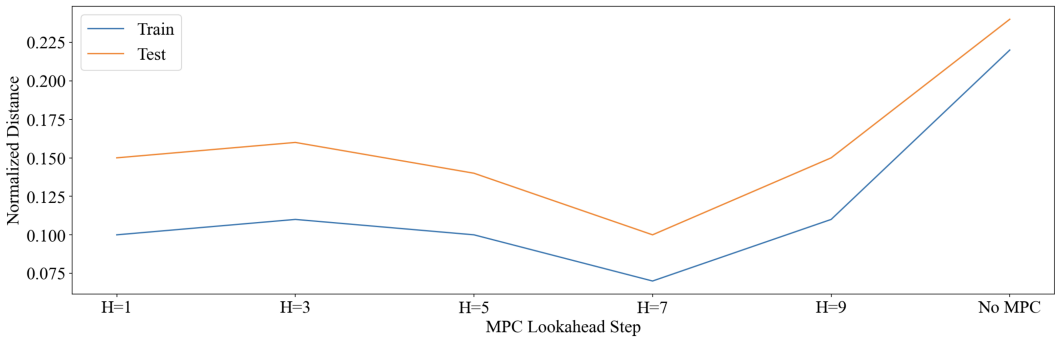


Figure 7: Ablation Studies on Lookahead Horizon. The plot shows the normalized distance performances of different H values on both train and testing objects.

frequency is too low or too high, the performance becomes suboptimal. When $H = 1$, the system effectively reduces to FlowBot 3D, which replans every step. Another interesting comparison in this experiment is that when we do not use this MPC controller (i.e. we do not replan and trust the one-shot open-loop plan), the performance degrades by a lot. This suggest that we do need to replan at a certain frequency to correct ourselves. Experiments suggest that the optimal H value here is $H = 7$ and we use this value in our final system.

B.2 Mean Aggregation with Segmentation Masks

We also ablate on the choice of using a segmentation mask to aggregate the articulation parameters estimates in Table 3. Results suggest the effectiveness of using segmentation masks to aggregate multiple results for a robust estimate. Such effectiveness is better shown on revolute objects as flow directions alone suffice to produce a good motion for prismatic objects.

C Baselines

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

- **UMP-DI**: We implement UMPNet's Direction Inference network (DistNet) [12], 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.

	Novel Instances in Train Categories												Test Categories									
	AVG.												AVG.									
FlowBot++ (H=1)	0.10	0.06	0.09	0.09	0.09	0.02	0.01	0.17	0.20	0.17	0.15	0.02	0.15	0.00	0.10	0.12	0.00	0.33	0.12	0.22	0.17	0.18	0.25
FlowBot++ (H=3)	0.11	0.03	0.10	0.06	0.09	0.00	0.24	0.19	0.18	0.21	0.09	0.03	0.16	0.15	0.14	0.14	0.00	0.34	0.04	0.21	0.20	0.19	0.27
FlowBot++ (H=5)	0.10	0.06	0.08	0.06	0.09	0.01	0.23	0.17	0.17	0.12	0.11	0.02	0.14	0.12	0.09	0.15	0.00	0.27	0.01	0.16	0.18	0.18	0.22
FlowBot++ (H=7)	0.07	0.04	0.01	0.04	0.08	0.02	0.19	0.17	0.14	0.07	0.09	0.02	0.10	0.00	0.11	0.09	0.00	0.23	0.02	0.09	0.09	0.20	0.18
FlowBot++ (H=9)	0.11	0.06	0.05	0.05	0.27	0.03	0.17	0.19	0.25	0.06	0.07	0.02	0.15	0.13	0.09	0.22	0.00	0.14	0.22	0.11	0.21	0.17	0.16
FlowBot++ (no MPC)	0.22	0.25	0.24	0.20	0.50	0.04	0.22	0.32	0.37	0.10	0.09	0.07	0.24	0.31	0.14	0.35	0.00	0.21	0.32	0.22	0.18	0.22	0.40

Table 2: Ablation Studies of Lookahead Horizon (H) via Normalized Distance (↓). The lower the better.

	Novel Instances in Train Categories												Test Categories									
	AVG.												AVG.									
FlowBot++	0.07	0.04	0.01	0.04	0.08	0.02	0.19	0.17	0.14	0.07	0.09	0.02	0.10	0.00	0.11	0.09	0.00	0.23	0.02	0.09	0.09	0.20	0.18
FlowBot++ No Seg	0.10	0.06	0.01	0.12	0.08	0.09	0.21	0.18	0.14	0.09	0.09	0.08	0.13	0.15	0.15	0.09	0.00	0.25	0.02	0.12	0.09	0.19	0.25

Table 3: Ablation Studies of Mean Aggregation with Segmentation via Normalized Distance (↓). The lower the better.

- **Normal Direction**: We use off-the-shelf normal estimation to estimate the surface normals of the point cloud using Open3D [33]. 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.
- **DAgger E2E**: We also conduct behavioral cloning experiments with DAgger [31] on the same expert dataset as in the BC baseline. We train it end-to-end (E2E), similar to the BC model above.
- **FlowBot3D**: We also call this AF Only, since it only uses the articulation flow f_p during planning) [6],
- **Without Gram-Schmidt Correction**: Also called AP Only. This is our model but without Gram-Schmidt correction via f_p, hence the name AP Only, since it only uses the inferred articulation parameters during planning without f_p correction).

	Novel Instances in Train Categories												Test Categories									
	AVG.												AVG.									
UMP-DI [12]	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 [1]	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
DAgger E2E [31]	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
FlowBot3D (AF Only) [6]	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
AP Only (Ours)	0.77	0.58	0.59	**0.90**	0.81	0.83	0.80	0.78	0.46	**0.79**	0.72	1.00	0.68	1.00	0.54	0.82	1.00	0.41	0.44	0.79	0.87	0.41	0.45
FlowBot++ (Ours - Combined)	**0.79**	**0.61**	**0.59**	0.89	0.82	**0.96**	0.86	0.78	**0.52**	0.78	0.75	1.00	**0.74**	**1.00**	**0.68**	**1.00**	**1.00**	0.43	0.63	**0.88**	0.81	**0.45**	**0.52**

Table 4: 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.

D Metrics

We specify the metrics we used for our simulated experiments. First, shown in Table 1, we use Normalized Distance, which is defined as the normalized distance traveled 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 ($\mathbf{j}_{\text{init}}$):

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

We also conduct experiments using the Success Rate: we define a binary success metric, which is computed by thresholding the final resulting normalized distance at δ:

$$\text{Success} = \mathbb{1}(\mathcal{E}_{\text{goal}} \leq \delta)$$

We set $\delta = 0.1$, meaning that we define a success as articulating a part for more than 90%.

We show the success rate performance of our method and baselines in Table 4. Similar to the results using Normalized Distance, FlowBot++ outperforms previous methods.

E Simulation and Training Details

In simulation, the suction is implemented using a strong force between the robot gripper and the target part. During training step t, we randomly select an object from the dataset, randomize the object's configuration, and compute a new training example which we use to compute the loss using Eq. 7. During training, each object is seen in 100 different randomized configurations.

E.1 Datasets

To evaluate our method in simulation, we implement a suction gripper in the PyBullet environment, which serves as a simulation interface for interacting with the PartNet-Mobility dataset [10]. The PartNet-Mobility dataset contains 46 categories of articulated objects; following UMPNet [12], 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 evaluation. We train our models (FlowProjNet 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 dataset. 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 in simulation, we include in the observation O_t a binary part mask indicating which points belong to the child joint of interest.

E.2 Network Architecture

FlowProjNet, the joint Articulation Flow and Articulation Projection prediction model in FlowBot++, is based on the PointNet++ [34, 30] 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 FlowProjNet architecture is implemented using Pytorch-Geometric, a graph-learning framework based on PyTorch. Since we are doing regression, we use standard L2 loss optimized by an Adam optimizer [35].

The articulation type classifier model in FlowBot++, which is used to predict prismatic vs. revolute objects, is also based on the PointNet++ architecture. The architecture now uses a classification head, which outputs a global binary label representing the articulation label. We use standard Binary Cross Entropy loss optimized by an Adam optimizer [35] and we achieve 97% accuracy on test objects.

E.3 Ground Truth Labels Generation

E.3.1 Ground Truth Articulation Flow

We implement efficient ground truth 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 Articulation Flow.

E.3.2 Ground Truth Articulation Projection

We also implement efficient ground truth Articulation Projection generation. For each object, the system reads the current state of the object of interest in simulation as an URDF file and parses it to obtain the origin v and direction $\boldsymbol{\omega}$ of the axis of articulation. The system then uses Eq. 2 to calculate the Articulation Projection label. Since we have access to part-specific masks in PyBullet, the calculated points' location will be masked out such that only the points on part of interest will be articulated.

E.4 Using Segmentation Masks

As mentioned above, we use part-specific segmentation masks to define tasks. Specifically, we follow the convention in [12, 6, 13], where a segmentation mask is provided to give us the part of interest. Thus, it is possible that an object (e.g. a cabinet) could have multiple doors and drawers at the same time. By the construction of the dataset [12, 6, 13], in each data point (object), a mask is used to define which part on the object needs to be articulated. For the cabinet example, if the mask is provided for a drawer, then the cabinet is classified as a prismatic object. If the mask is provided for a door, then it is classified as a revolute object. We use segmentation masks in the following steps of the FlowBot++ pipeline:

1. **Articulation Flow Ground Truth**: During the generation of articulation flow labels, we use segmentation masks to mask out irrelevant parts on objects so that those parts' articulation flow values will be zeroed out. In this way, FlowProjNet will learn to predict all-zero on irrelevant parts.
2. **Articulation Projection Ground Truth**: Similar to how we use the mask in Articulation Flow Ground Truth generation, we only produces the projection vectors for the relevant masked points.
3. **Articulation Flow and Articulation Projection Prediction**: During the learning step of FlowProjNet, we use segmentation masks as an additional per-point channel into the network, where 1 represents relevant points and 0 represents irrelevant points. In this way, the network output learns to be conditioned on this extra channel such that it does not output values on irrelevant parts.
4. **Articulation Parameters Estimation**: When estimating $\boldsymbol{\omega}$ and v, we first obtain a per-point estimate. To make the estimate more robust, we aggregate all points on the relevant part, which is masked using the segmentation mask, and average them out to get a robust estimate.

E.5 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.

F Real-World Experiments

F.1 Experiments Details

We experiment with 6 different objects in the real world. Specifically, we choose 5 revolute objects: Oven, Fridge, Toilet, Trashcan, and Microwave, where the predicted trajectories are generated using Eq. 5, and we choose 1 prismatic object: Drawer, where the predicted trajectories are generated via Eq. 6. For each object in this dataset, we conducted 5 trials of each method. For each trial, the object is placed in the scene at a random position and orientation. For each trial, we visualize the point clouds beforehand and hand-label the segmentation masks using bounding boxes. We then pass the segmentation masks as the auxiliary input channel to FlowProjNet and use them to aggregate the final output to improve robustness. We then qualitatively assess the prediction by visualizing the

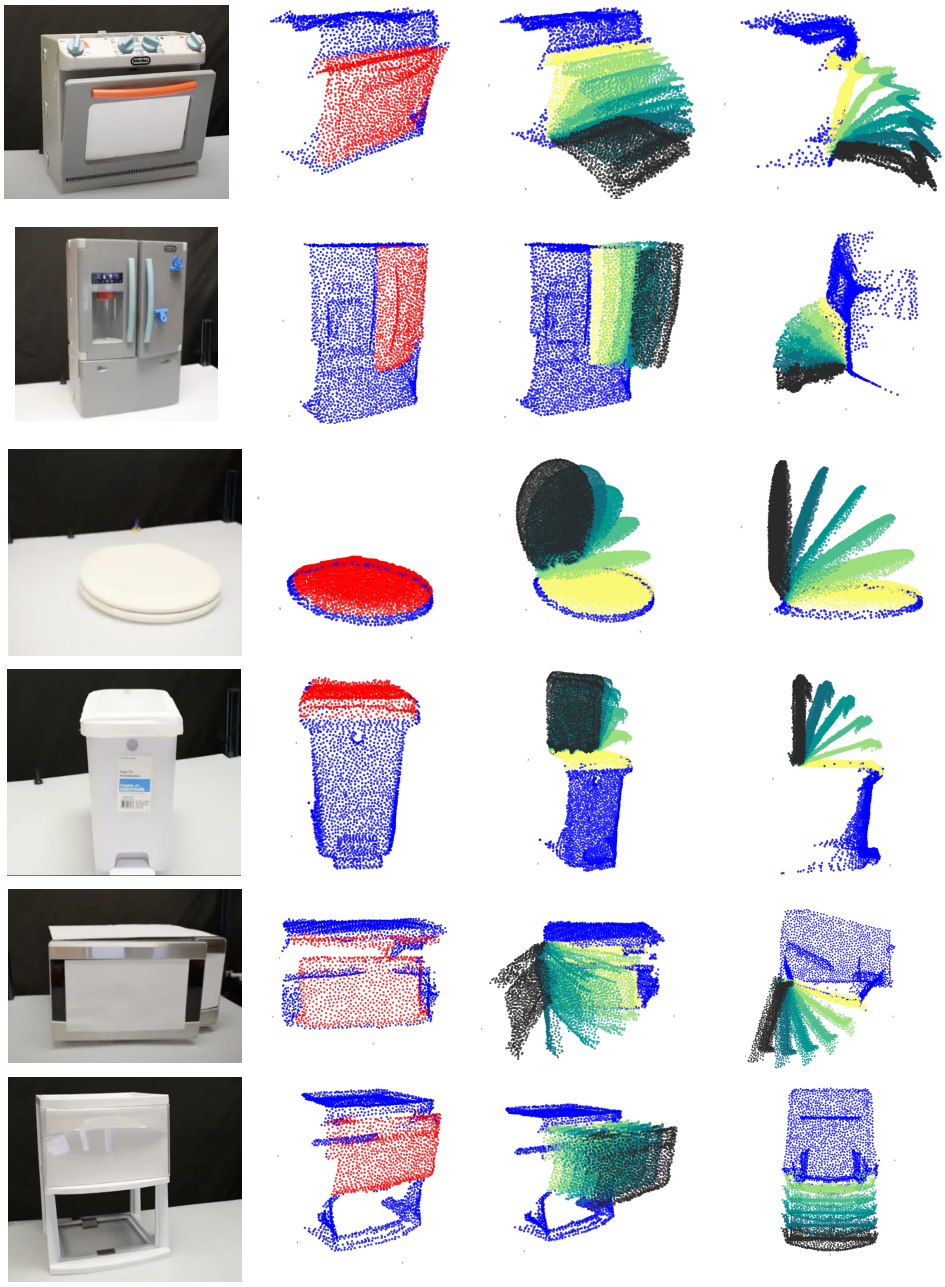


Figure 8: Successful Predictions on 6 Real-World Objects. We show FlowBot++'s prediction quality of 6 different real-world objects. For better readability, we provide an alternative view for each prediction. From top to bottom, the objects are: Oven, Fridge, Toilet, Trashcan, Microwave, and Drawer.

points belonging to the segmented part under the predicted trajectory's transformation. We show some examples of the predictions in Fig. 8.

F.2 Robotic System Implementation

We provide the details of the physical robot system implementation of FlowBot++. The setup remains straightforward and largely similar to previous works [28, 6].

F.2.1 Hardware

In all of our real-world experiments, we deploy our system on a Rethink Sawyer Robot and the sensory data (point cloud) come from an Intel RealSence depth camera. The robot's end effector is an official Saywer Parallel-Jaw Gripper. We set up our workspace in a 1.2 m by 1.00 m space put together using the official Sawyer robot mount and a regular desk. We set up the RealSense camera such that it points toward the center of workspace and has minimal interference with the robot arm-reach trajectory.

F.2.2 Solving for the Robot's Trajectory

The choice of the contact point is similar to the procedures described in Eisner et al. [6]. When the contact point's full trajectory is predicted using Eq. 5 or 6 based on the part's articulation type, the robot should just plan motions in order to make its end-effector follow the predicted trajectory. Once a successful contact is made, the robot end-effector is rigidly attached to the action object, and we then use the same predicted trajectory waypoints as the end positions of the robot end effector, and then feed the end-effector positions to MoveIt! to get a full trajectory in joint space using Inverse Kinematics. For *prismatic objects*, this is convenient because the robot gripper does not need to change its orientation throughout the predicted trajectory. For *revolute objects*, we propose a method to efficiently calculate the robot's orientations in tandem with the positions in the planned trajectory. Concretely, the trajectory of Eq. 5 gives us the end-effector's xyz positions in the trajectory; it is rigidly attached to the contact point, so we could treat their xyz positions to be the same in this trajectory. Now, we are interested in obtaining the orientation of the end-effector for each step. Assume the end-effector's orientation (obtained via Forward Kinematics, in the form of rotation matrix in $SO(3)$) is $\mathbf{q}_0$ when making a successful contact with the part of interest. By definition, each step in τ_{revolute} corresponds to a unique rotation matrix $\mathbf{R}(\phi_g/K)$, representing the difference of rotation due to the increase of opening angle in each step. We then calculate the robot end-effector's orientation at each step i:

$$\mathbf{q}_i = \mathbf{R}(\frac{\phi_g}{K})\mathbf{q}_{i-1} \tag{8}$$

by applying the difference of rotation onto the orientation's rotation matrix itself iteratively. Thus, in this case, the robot's end-effector's full $SE(3)$ trajectory is obtained:

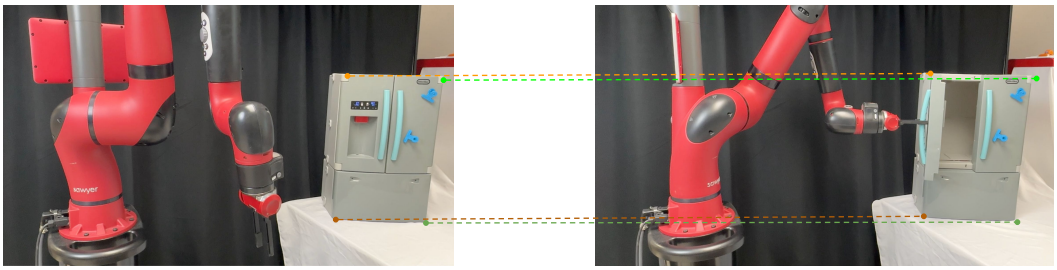
$$\tau_{\text{ee}} = \left\{\tau^i_{\text{revolute}}, \mathbf{q}_i\right\}_{\forall i \in [0,K]} \tag{9}$$

We then obtain the robot's joint-space trajectory using Inverse-Kinematics (IK):

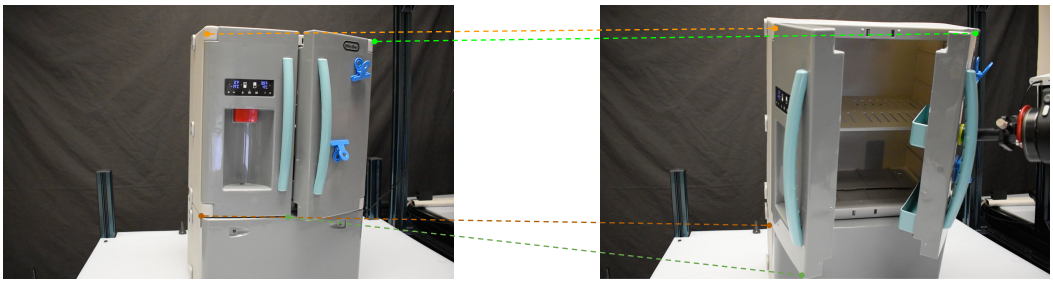
$$\tau_{\text{joint}} = IK(\tau_{\text{ee}}) \tag{10}$$

F.3 Reducing Unwanted Movements

With the ability to control the full 6D pose of the robot end-effector in the trajectory, we are also able to reduce the unwanted movements of the object itself. In FlowBot3D [6], the suction gripper's rotation is controlled using a heuristic based on the flow direction prediction, which is often off. Thus, incorrect rotation could cause the gripper to yank the object too hard in a wrong direction, causing unwanted motion of the articulated object or even detaching the gripper from the object surface. With the full 6D gripper trajectory produced by FlowBot++ as a byproduct, the relative pose between the gripper and the articulated part remains the same as when a contact is made throughout the trajectory. This largely eliminates the unwanted movement problem in [6]. We illustrate this property in Fig. 9. A disadvantage of deploying FlowBot3D in the real world is that each step is prone to error, causing the gripper to move to wrong directions, which would unexpectedly move the object, potentially causing damage. Using the full gripper trajectory derived from Eq. 9, FlowBot++ is more likely to be more compliant with respect to the object's kinematic constraints without using hand-designed heuristics based on Articulation Flow predictions. The position and pose of the body of the articulated object are then able to remain unchanged. In contrast, FlowBot3D has more points of failure due to its closed-loop nature. When a single step's Articulation Flow prediction is off - namely, non-parallel to the ground-truth flow direction, the gripper would move against the object's



FlowBot++ (Ours)



FlowBot3D

Figure 9: FlowBot++ Reducing Unwanted Movements of the Object. **Top:** FlowBot++ opens the left door of the fridge. The position and pose of the body of the fridge remain unchanged. **Bottom:** FlowBot3D opens the right door of the fridge. Due to wrong flow prediction at intermediate steps, the gripper yanks the fridge too hard that it tips over, causing unwanted motions of the fridge and opening the wrong door by accident.

kinematic constraint, moving the other parts of the object unexpectedly. Please note that this is better understood by watching the video comparisons on our website.

F.4 Failure Case

We illustrate a failure case of FlowBot++ deployed in the real world here. The failure is caused by predictions that are off, which results in off-axis rotation. The failure case is shown in Fig. 10. In

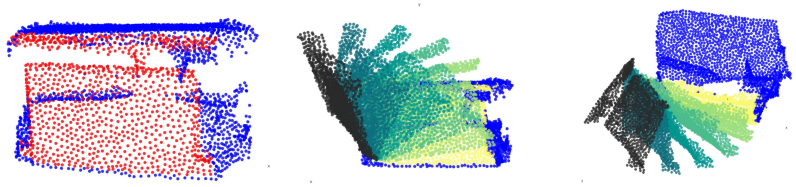


Figure 10: Failed FlowBot++ Prediction on a Microwave. Imperfect articulation parameter prediction caused the rotation to be off-axis.

this prediction, the prediction result in incorrect articulation parameters. From the visualization, the predicted axis is off, causing the rotated part to go "off the hinge." If a real robot were to execute this, the planned motion would either be infeasible or make the robot lose contact with the grasp point.