Appendix

This appendix is organized into several sections to provide comprehensive details about our experimental setup, methodology, and additional results. We cover key aspects such as simulation environments, learning processes, execution details, baseline settings, real-world experiments, and object details. Each section delves into assumptions, settings, hyperparameters, and supplementary results that could not be included in the main paper.

A. Simulation Details

Our experiments are performed in three simulation environments:

- **TrainingEnv:** This environment excludes the robot and focuses on force-based interactions with objects. Random objects from the training set are selected and placed at $[-1, 0]$ with randomized rotations around the Z-axis. Force application points are randomly sampled, and actions are applied using the simulator.
- **BaselineTrainingEnv:** Includes both a Panda robot and objects. The objects are placed on a partial circle around the robot to ensure reachability. Object orientations are randomized for evaluation consistency.
- **EvalEnv:** Both FLEX and baseline methods are evaluated in this environment with consistent settings.

B. Learning Process

1) Network Design: The actor network for TD3 has two heads:

$$\text{Action Direction: 3 dimensions,}$$
$$\text{Action Scale: 1 dimension.}$$

Both the action dimensions and action scale are limited to (0, 1] using bounded activation functions. The action is then computed as:

$$a_t = \pi_{\text{direction}}(s_t) \cdot \pi_{\text{scale}}(s_t) \cdot \eta$$

where η is the maximum allowable force.

2) Curriculum Learning for Revolute Objects: Training for revolute objects uses the following curricula with increasing rotation randomization:

$$(-0.25, 0.25),$$
$$(-\pi/2, 0),$$
$$(-\pi/2, \pi/2),$$
$$(-\pi, 0),$$
$$(-\pi, \pi).$$

The next curriculum is used when the agent applies 80% of force in the correct direction over 100 episodes.

3) Network Structure: The actor network is a two-headed MLP network. One input layer and two hidden layers encode the state. The scale head and direction head then map the encoded features to normalized three-dimensional action direction and one-dimensional action scale in order to compute actions.

Parameter	**Value**
Actor Hidden layers	2
Actor Hidden layer dimensions	(400,300)
$\pi_{\text{direction}}$ Activation function	Sigmoid
π_{scale} Activation function	Tahn
Critic Hidden Layers	2
Critic Hidden layer dimensions	(400,300)
Critic Activation Function	ReLU

TABLE I: Agent Network Struture

Parameter	**Value**
Discount Factor (γ)	0.99
Batch Size	100
Learning Rate	0.001
Polyak Update (τ)	0.995
Noise Clip	1
Policy Delay	2
Max Timesteps per Episode	200
Rollouts	5
State Dimension	6
Action Dimension	3
Max Action	5
η	0.02

TABLE II: Reinforcement Learning Hyperparameters

4) RL Hyperparameters: The training was performed with 24 parallel environments on an Intel i9-12900K processor with an Nvidia RTX 3080 GPU.

C. Execution Details

1) Execution Assumptions and Object Placement: To avoid collisions, revolute objects are placed 0.7m away with a randomized rotation angle between $[-\pi/2, 0]$, while prismatic objects are placed 0.9m away with an angle between $[-3\pi/4, \pi/4]$.

D. Baselines

1) End-to-End RL: The end-to-end RL agent is trained using the StableBaselines3 TD3 agent, with objects placed at a fixed location. The reward function is structured to encourage the robot to approach, grasp, and manipulate the object in stages.

2) CIPS Training: To improve efficiency, CIPS training begins with the robot *already* grasping the object. Four pre-recorded initial states are uniformly sampled from the $1/4$ circle around the robot. This method accelerates training by reducing unnecessary exploration.

3) GAMMA: As stated in the paper, we assume knowledge of joint type for GAMMA evaluation. We designed a simple planning method that complies with the ideal joint dynamics. Specifically, for revolute objects,

$$\boldsymbol{a}_t^{GAMMA} = -\hat{\boldsymbol{h}_r} \times \hat{\boldsymbol{v}}_t$$

For prismatic objects,

$$\boldsymbol{a}_t^{GAMMA} = \hat{\boldsymbol{h}_p}$$

Where $\hat{\boldsymbol{h}_r}$ and $\hat{\boldsymbol{h}_p}$ are the selected prediction results of joint parameters given by GAMMA.

We observed that GAMMA tends to overestimate the number of joints on an object. As a result, we only choose the first inference result that complies our knowledge of the joint type from the inference result. If there is no correct inferred joint type, meaning that GAMMA fails to detect the type of object, we randomly choose one inference result.

Parameter	**Value**
Policy Learning Rate	Default (SB3)
Discount Factor (γ)	Default (SB3)
Replay Buffer Size	Default (SB3)

TABLE III: Baseline Hyperparameters for CIPS

E. Real-World Setup

1) Hardware and Simplifications: The real-world experiments used a UR-5 arm and an Intel Realsense D455 RGBD camera. To simplify, we used an inverse kinematics (IK) controller and a toy drawer. We assumed knowledge of the drawer pose and skipped interactive perception. The hyperparameters stay the same with simulation experiment settings.

F. Additional Experiments and Results

1) 80% Opening Criterion: In our experiments, success is defined as opening the object to at least 80% of its joint limit.

2) Additional Experiments: Several additional experiments were conducted on objects with different joint configurations, but due to space limitations, they are not included in the main paper.

G. Object Details and Dataset

1) Object Scaling and Preprocessing: Objects were selected from the Partnet-Mobility dataset and scaled for simulation. Prismatic objects were scaled to 50% and revolute objects to 30% of their original size.

2) Collision Handling: Due to self-collision issues in the dataset, we disabled collisions for the main body of objects during training. Since the objects were partially opened before manipulation, this did not affect the results.