FLEX: A Framework for Learning Robot-Agnostic Force-based Skills Involving Sustained Contact Object Manipulation

Shijie Fang[1†], Wenchang Gao[1†], Shivam Goel[1†], Christopher Thierauf[1], Matthias Scheutz[1], Jivko Sinapov[1]

***Abstract*— Learning to manipulate objects efficiently, particularly those involving sustained contact (e.g., pushing, sliding) and articulated parts (e.g., drawers, doors), presents significant challenges. Traditional methods, such as robot-centric reinforcement learning (RL), imitation learning, and hybrid techniques, require massive training and often struggle to generalize across different objects and robot platforms. We propose a novel framework for learning object-centric manipulation policies in *force space*, decoupling the robot from the object. By directly applying forces to selected regions of the object, our method simplifies the action space, reduces unnecessary exploration, and decreases simulation overhead. This approach, trained in simulation on a small set of representative objects, captures object dynamics—such as joint configurations—allowing policies to generalize effectively to new, unseen objects. Decoupling these policies from robot-specific dynamics enables direct transfer to different robotic platforms (e.g., Kinova, Panda, UR5) without retraining. Our evaluations demonstrate that the method significantly outperforms baselines, achieving over an order of magnitude improvement in training efficiency compared to other state-of-the-art methods. Additionally, operating in force space enhances policy transferability across diverse robot platforms and object types. We further showcase the applicability of our method in a real-world robotic setting. Link: `https://tufts-ai-robotics-group.github.io/FLEX/`**

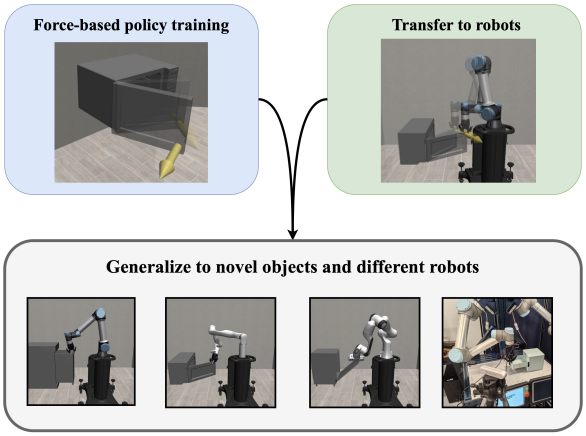

Fig. 1: Our method (**FLEX**: **F**orce-based **L**earning for **EX**tended Manipulation) learns force-based skills for manipulating articulated objects across various robots by decoupling policy learning from robot dynamics. The blue box (top-left) shows the core mechanism of force-based skill learning, while the green box (top-right) illustrates the transfer to different robots. The white box (bottom) highlights manipulation tasks: L-R, the UR5e opens a cabinet, Kinova opens a microwave door, Panda opens a dishwasher rack, and a real UR5 opens a drawer. Detailed architecture in Figure 3.

I. Introduction

Learning to manipulate objects, particularly in tasks involving continuous interaction, such as pushing, sliding, or handling articulated objects like drawers and doors, is a significant challenge in robotics. These tasks, often referred to as sustained contact manipulation [1], require the continuous application of force throughout the interaction. Efficiently learning such policies is especially difficult when aiming to generalize across different objects and robotic platforms.

Learning-based approaches like reinforcement learning (RL) and imitation learning (IL) typically focus on robot-centric control strategies tailored to specific tasks or configurations. However, these methods require extensive training data and often struggle to generalize to unseen objects or transfer across different robotic systems. Additionally, traditional control techniques such as model predictive control (MPC) rely on precise object and robot dynamics modeling, limiting their applicability in real-world environments.

Recent efforts have sought to simplify RL by leveraging task structures to improve efficiency [2] and by decoupling object-specific dynamics from robot control to enhance generalization [3]. While they show potential, achieving both efficient learning and robust generalization across diverse objects and robots remains challenging, particularly for tasks involving sustained contact and articulated object dynamics.

This work proposes an approach (Figure 1) to address these challenges by learning object-centric RL policies in *force space*. Rather than relying on robot-specific control strategies, our method governs object behavior through forces applied directly to the object. By training policies in simulation, we capture key object dynamics, such as movement constrained by joint configurations, which enables generalization across objects of the same type. This allows the learned policies to adapt effectively to novel objects by focusing on their motion-based properties and joint constraints.

Additionally, our method reduces reliance on pre-trained perception models for identifying joint types and configurations. By decoupling policies from low-level robot dynamics, it enables direct transfer across platforms without retraining, while object-centric force interactions improve data efficiency, policy performance, and generalization across diverse objects and robots.

Our method offers three key advantages for sustained contact manipulation of articulated objects: (1) improved data and time efficiency, achieved by representing actions in

†These authors contributed equally to this work, and their names are listed in alphabetical order by last name.

[1]Tufts University School of Engineering, Computer Science. Medford, Massachusetts, United States of America {fistname.lastname}@tufts.edu

force space and eliminating unnecessary exploration, which accelerates training by removing robot-specific kinematics from the simulation; (2) generalizability across objects with similar dynamics by capturing essential object features; and (3) direct transferability across different robotic platforms, as demonstrated through evaluations on the Kinova, UR5, and Panda robots in simulation and UR5 arm in real-world.

II. Related Works

Robot learning for sustained contact manipulation: Sustained contact manipulation, involving tasks like pulling drawers or opening microwaves, poses challenges due to high-dimensional action spaces and extended task duration [4]. Early RL and IL approaches [5], [6] were hidered by time-consuming real-world data collection. Simulated environments [7]–[9] have improved efficiency but still struggle with stable interactions throughout extended manipulation tasks and managing complex state-action dynamics [10].

Recent works simplify RL by reducing action spaces through decoupling high-level policies from low-level motion planning [11], [12], primitive actions [12], [13], or robot control [3]. Nasiriany et al. [12] use predefined primitive actions, while Lu et al. [3] decouple object dynamics from robot control. Other strategies, like structured RL combined with task and motion planning [14] or model-based approaches [15], aim to reduce the state space but struggle to prevent suboptimal exploration into irrelevant states.

Learning in the force domain offers a promising way to reduce action space dimensionality and prevent unnecessary exploration. Combining force-based RL with symbolic planning has been explored [16], but reliance on accurate digital twins limits generalization to new objects. Similarly, using video demonstrations to apply force on objects [17] improves exploration, but guiding the force applier requires significant exploration, increasing training time. Additionally, dependence on large, high-quality demonstration datasets and extensive visual training exacerbates the sim-to-real gap.

Object-centric representation for manipulation focus on objects rather than specific robots, enabling generalization across tasks and robotic platforms [4]. Methods leveraging 3D perception, such as object-centric actions [18], trajectories [19], and affordances [20], enhance the transferability of skills to new objects. For example, point clouds can infer joint configurations [21], while articulation flow prediction guides action direction [22]. UMPNet [23] generates closed-loop SE(3) action trajectories from RGB-D inputs using Arrow-of-Time inference, enabling generalization across unseen articulation structures without explicit joint modeling.

While these approaches have simplified robot learning through action space reduction using object-centric representations, generalizing across objects and platforms remains challenging — particularly due to their reliance on pose-space control and robot-specific execution, which limit transferability and hinder stable force application in sustained contact interactions. We build on these efforts by focusing on force-space learning of manipulation skills, targeting object properties like joint configurations. Decoupling robot-specific dynamics and operating solely in object space avoids suboptimal exploration and reduces training time. This accelerates learning, enhances generalization, and enables easy policy transfer across robotic platforms without retraining.

III. Preliminaries

A. Articulated Object Parametrization

An articulated object consists of multiple rigid bodies, referred to as **links**, connected by **joints** that permit relative motion of object parts. In this work, we focus on objects with joints that allow one degree of freedom, either rotational (revolute) or translational (prismatic). Each joint connects parent and child links, enabling the object to change configuration [22]. Many real-world articulated objects (e.g., drawers, doors) follow this structure, with their dynamics determined by the joint type and configuration.

Prismatic joint: A prismatic joint allows for linear motion along a fixed axis. The position of a point $\boldsymbol{p}$ along the prismatic joint is defined as:

$$\boldsymbol{p} = \boldsymbol{p_0} + k \cdot \boldsymbol{h_p}, \quad k \in \mathbb{R}, \quad \boldsymbol{p}, \boldsymbol{p_0} \in \mathbb{R}^3 \tag{1}$$

where $\boldsymbol{p_0}$ is the initial position, $\boldsymbol{h_p} \in \mathbb{R}^3$ is the unit vector representing the direction of the joint axis, and k is the scalar displacement along the axis (Figure 2a).

Revolute joint: A revolute joint allows for rotational movement around a fixed joint axis. The position of a point $\boldsymbol{p}$ on a revolute joint satisfies the constraint:

$$\|(\boldsymbol{p} - \boldsymbol{p_r}) - [(\boldsymbol{p} - \boldsymbol{p_r}) \cdot \boldsymbol{h_r}]\, \boldsymbol{h_r}\| = r \tag{2}$$

where $\boldsymbol{p_r} \in \mathbb{R}^3$ is a point on the axis of rotation (joint origin), $\boldsymbol{h_r} \in \mathbb{R}^3$ is the unit vector along the rotation axis, and r is the radius of the circular motion. This equation constrains $\boldsymbol{p}$ to move in a circular path around the axis (Figure 2b).

B. Reinforcement Learning for Force-Based Manipulation

We formulate the manipulation of articulated objects as an RL problem, aiming to learn force-based policies that generalize across objects with similar joint configurations. This problem is modeled using two Markov Decision Processes (MDPs): one for prismatic joints, $M_p = \langle S_p, A, T_p, R, \gamma \rangle$, and one for revolute joints, $M_r = \langle S_r, A, T_r, R, \gamma \rangle$. In this setup, the state spaces S_p and S_r represent the object's joint-specific information, such as joint axis direction and configuration of the object. The shared action space A consists of 3D force vectors applied to the surface of the object. The transition functions T_p, and T_r govern the object's motion

(a) Prismatic **(b)** Revolute

Fig. 2: Illustration of the two joint configurations and states

dynamics based on the applied forces, reflecting the behavior of prismatic and revolute joints, respectively. The reward function R encourages efficient manipulation by maximizing object displacement with minimal applied force, with γ being the discount factor. Through this formulation, we aim to learn two policies: π_p for prismatic joints and π_r for revolute joints, trained to maximize the expected cumulative reward: $G_t = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_t, a_t, s_{t+1})\right]$, where $s_t \in S_p$ or S_r and $a_t \in A$. These policies will generalize across articulated objects with the same joint configurations, allowing the robot to manipulate a wide range of objects by applying the correct force-based strategy for the given joint type.

C. Problem Formulation

The challenge we address is solving sustained contact manipulation tasks for articulated objects characterized by prismatic or revolute joints. This requires learning force-based policies in simulation that generalize across various robotic platforms by adapting to specific joint dynamics. Key aspects include (1) learning efficient force-based manipulation strategies for both joint types and (2) executing these learned policies across different robots without retraining.

The problem can be broken down into three components: (1) **Policy learning**: The objective is to learn two manipulation policies: one for prismatic joints, $\pi_p : S_p \rightarrow A$, and one for revolute joints, $\pi_r : S_r \rightarrow A$. These policies must handle different contact points on objects and generalize across various objects with the same joint type. The learned policies should also be directly transferable to different robotic platforms without retraining.
(2) **Joint identification during policy execution**: The robot must infer the joint type (prismatic or revolute) of the object it is interacting with. Observing a trajectory of N end-effector positions $\xi = \{\boldsymbol{p_i^{eef}} \in \mathbb{R}^3\}_{i=1}^{N}$, the joint type $J \in \{\text{prismatic, revolute}\}$ is inferred by maximizing the likelihood of the trajectory by Maximum Likelihood Estimation (MLE) problem:

$$\hat{J} = \underset{J \in \{\text{prismatic, revolute}\}}{\operatorname{argmax}} \; p(\xi \mid J) \tag{3}$$

where $p(\xi \mid J)$ is the likelihood of trajectory ξ conditioned on joint type J and its parameters (e.g., joint axis, joint point, etc.). The robot selects the joint type with the highest likelihood and applies the corresponding pre-learned policy.
(3) **Efficient execution across robots**: The learned object-centric policies generalize across robotic platforms, allowing manipulation without the need for retraining. This ensures that policies learned in an object-centric simulation are directly transferable to different robots.

IV. Learning Methodology

The architecture of FLEX consists of three main components: (1) Force-space skill learning, (2) Joint parameter estimation, and (3) Robot execution, as shown in Figure 3.

A. Force-based skill learning

We train object-centric manipulation policies in a simulation environment without a robot. As described in Section III-B, learning is modeled using two MDPs: one for prismatic joints, $M_p = \langle S_p, A, R, T_p, \gamma \rangle$, and one for revolute joints, $M_r = \langle S_r, A, R, T_r, \gamma \rangle$. The state spaces S_p and S_r differ depending on the joint type. Formally:

$$s_t^p = \begin{bmatrix} \boldsymbol{h_p} \\ \boldsymbol{\Delta p_t} \end{bmatrix}, \quad s_t^r = \begin{bmatrix} \boldsymbol{h_r} \\ \boldsymbol{v_t} \end{bmatrix}$$
$$\boldsymbol{\Delta p_t} = \boldsymbol{p_t} - \boldsymbol{p_0}$$
$$\boldsymbol{v_t} = (\boldsymbol{p_t} - \boldsymbol{p_r}) - [(\boldsymbol{p_t} - \boldsymbol{p_r}) \cdot \boldsymbol{h_r}]\boldsymbol{h_r}$$

For prismatic objects, the state $s_t^p \in S_p$ consists of the unit vector $\boldsymbol{h_p} \in \mathbb{R}^3$ representing the joint axis direction, and the displacement $\boldsymbol{\Delta p_t}$, which is the difference between the current force application point $\boldsymbol{p_t} \in \mathbb{R}^3$ and initial position $\boldsymbol{p_0} \in \mathbb{R}^3$ at time step t (for illustration refer to Figure 2a).

For revolute objects, the state $s^r \in S_r$ includes the unit vector $\boldsymbol{h_r} \in \mathbb{R}^3$ representing the joint axis direction, and $\boldsymbol{v_t}$, the projection of the current force point $\boldsymbol{p_t} \in \mathbb{R}^3$ onto the joint axis. The joint position is represented by $\boldsymbol{p_r} \in \mathbb{R}^3$ (2b).

The action space shared across the two MDPs is defined as $A = \{\boldsymbol{F} \in \mathbb{R}^3 \mid \|\boldsymbol{F}\| \leq \eta\}$, which is the set of force vectors with a maximum magnitude of η. Every time step t the policy selects an action $a_t \in A$, which serves as the force $\boldsymbol{F}_t$ applied at the point of contact, and η is the maximum allowable force that can be applied to the object by the robot.

The reward function is shared between the two MDPs. It is defined as a dense reward $R(s_t, a_t, s_{t+1})$ that encourages efficient manipulation by maximizing object displacement with minimal applied force, as well as a large positive reward when the agent successfully reaches the goal state. Formally,

$$R(s_t, a_t, s_{t+1}) = \begin{cases} \|\Delta x\| \cos(\theta_{a_t, \Delta x}) + \mathbb{I}(s_{t+1} = s_{\text{goal}}) R_{\text{goal}} & \text{if } \Delta q \geq 0, \\ -\|\Delta x\| \cos(\theta_{a_t, \Delta x}) & \text{else.} \end{cases}$$

where: $\theta_{a_t, \Delta x}$ is the angle between the applied force a_t and the object's movement direction Δx, Δq represents the change in the object's joint configuration (either position for prismatic joints or rotation angle for revolute joints), $\mathbb{I}(s_{t+1} = s_{\text{goal}})$ is an indicator function that equals 1 if the agent reaches the goal state s_{goal}, and 0 otherwise, and R_{goal} is the large positive reward given when the agent successfully reaches the goal state. The transition functions T_p and T_r model the object's motion dynamics for prismatic and revolute joints, respectively.

The policies π_r (revolute) and π_p (prismatic) are learned using TD3 [24], with each policy network predicting force direction, $\pi_{\text{direction}}(s_t)$, and magnitude, $\pi_{\text{scale}}(s_t) \in [0, 1]$. The applied force at time t is $a_t = \pi_{\text{direction}}(s_t) \cdot \pi_{\text{scale}}(s_t) \cdot \eta$. This separation improves control precision and learning efficiency in force-space manipulation.

The learned policies are contact-agnostic, with contact points randomly sampled from a predefined region in each episode (see Section V-A for more details). By applying forces at various points, the robot learns a general manipulation strategy, ensuring robustness and adaptability regardless of the exact contact point.

At the end of training, the robot has two learned policies: π_r for revolute joints and π_p for prismatic joints. The next step is identifying the objects's joint type so the robot can select and apply the correct manipulation. We detail this process of joint estimation in the following subsection.

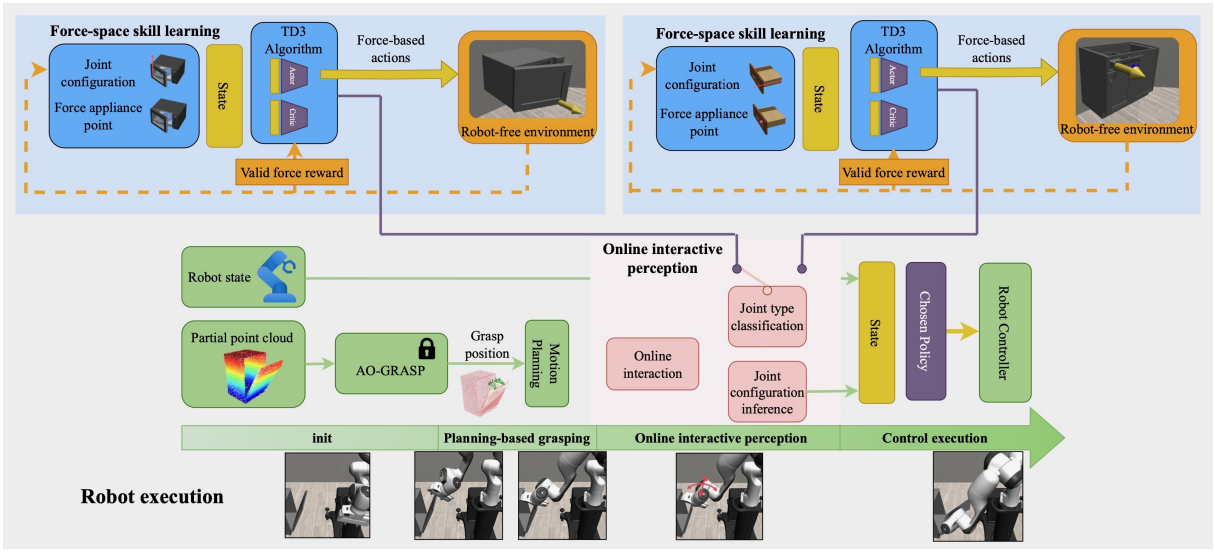

Fig. 3: The figure presents the architecture of **FLEX** (**F**orce-based **L**earning for **EX**tended Manipulation) for sustained contact manipulation of articulated objects. The blue boxes represent *Force-based skill learning* in simulation, the red boxes depict *Interactive joint parameter estimation*, and the green boxes show the *Robot execution* phase, where the learned policies are applied to perform tasks.

B. Interactive Joint Parameter Estimation

We now describe how the robot leverages these skills for object manipulation. The robot platform has an RGB-D camera that generates partial point clouds of the object. The pre-trained AO-GRASP model [25] identifies suitable grasp points, and motion planners guide the robot to reach and maintain a successful grasp.

The robot is not provided with any prior information about the object's joint configuration. To infer the joint type $J \in \{\text{prismatic, revolute}\}$, it securely holds the object and applies small forces in a specific sequence of predetermined directions—forward, backward, left, and right. This controlled interaction generates a short trajectory $\xi = \{\boldsymbol{p_i^{eef}} \in \mathbb{R}^3\}_{i=1}^{N}$, where each $\boldsymbol{p_i^{eef}}$ is the end-effector position at time step i, while ensuring that the robot maintains stable contact with the object. Based on this trajectory, the joint type and its parameters are estimated, guiding the selection of the appropriate manipulation policy.

As introduced in Section III-C, the problem of joint type inference is formalized as an MLE problem (see equation 3).

For a prismatic joint, the joint axis $\hat{\boldsymbol{h}}_{\boldsymbol{p}}$ and origin $\hat{\boldsymbol{p}}_{\boldsymbol{o}}$ are:

$$\hat{\boldsymbol{h}}_{\boldsymbol{p}}, \hat{\boldsymbol{p}}_{\boldsymbol{o}} = \underset{\boldsymbol{h_p}, \boldsymbol{p_o}}{\operatorname{argmax}}\; p\left(\xi \mid \boldsymbol{h_p}, \boldsymbol{p_o}, \text{prismatic}\right)$$

where $\boldsymbol{h_p}$ is the prismatic joint axis direction, and $\boldsymbol{p_o}$ is the origin of the prismatic joint. The likelihood $p(\xi \mid \text{prismatic})$ models the trajectory as linear motion along the joint axis.

Similarly, for revolute joints, the joint axis $\hat{\boldsymbol{h}}_{\boldsymbol{r}}$, joint point $\hat{\boldsymbol{p}}_{\boldsymbol{r}}$, and radius $\hat{r}$ are estimated as:

$$\hat{\boldsymbol{h}}_{\boldsymbol{r}}, \hat{\boldsymbol{p}}_{\boldsymbol{r}}, \hat{r} = \underset{\boldsymbol{h_r}, \boldsymbol{p_r}, r \leq \rho}{\operatorname{argmax}}\; p\left(\xi \mid \boldsymbol{h_r}, \boldsymbol{p_r}, r, \text{revolute}\right)$$

where $\boldsymbol{h_r}$ is the joint axis direction, $\boldsymbol{p_r}$ is the joint point, and r is the radius of the circular motion. The constraint $r \leq \rho$ ensures physical plausibility, where ρ is the maximum allowable radius.

To estimate these parameters, Principal Component Analysis (PCA) is applied to the end-effector trajectory ξ to yield initial estimates of the joint direction and orientation. These estimates are refined by solving least squares optimization.

For prismatic joints, the objective is to minimize the projection errors of the trajectory points onto the joint axis:

$$\min_{\boldsymbol{p_o}, \boldsymbol{h_p}} \sum_i \left\| \boldsymbol{p_i^{eef}} - \boldsymbol{p_o} - \left[\left(\boldsymbol{p_i^{eef}} - \boldsymbol{p_o} \right) \cdot \boldsymbol{h_p} \right] \boldsymbol{h_p} \right\|_2^2$$

This ensures that the trajectory points lie along the straight path, characteristic of prismatic motion.

For revolute joints, the goal is to minimize the error between the trajectory points and a circle of radius r around the joint point, as defined by:

$$\min_{\boldsymbol{p_r}, r \leq \rho} \sum_i \left(\left\| \boldsymbol{p_i^{eef}} - \boldsymbol{p_r} \right\|_2^2 - r^2 \right)$$

This captures the revolute motion, where the object's movement follows a circular trajectory. After determining the joint parameters, the robot uses the observed trajectory to compute the likelihood for both joint types. The robot then compares the likelihoods of the prismatic and revolute models, selecting the joint type J that maximizes the likelihood:

$$\hat{J} = \underset{J \in \{\text{prismatic, revolute}\}}{\operatorname{argmax}} \; p(\xi \mid J, \hat{\boldsymbol{\theta}}_J),$$

where $\hat{\boldsymbol{\theta}}_J$ represents the estimated parameters for joint type J, such as the axis $\boldsymbol{h}$ and origin $\boldsymbol{p_o}$ for prismatic joints, or the axis $\boldsymbol{h_r}$, point $\boldsymbol{p_r}$, and radius r for revolute joints.

C. Robot Execution

The final step applies the learned force-based manipulation policies to robotic platforms. The execution pipeline (green in Figure 3) consists of three components: perception and grasping, state construction, and control execution.

Perception and grasping: The robot uses its onboard RGB-D camera to generate partial point clouds of the object. Using the pre-trained AO-GRASP model [25], it identifies optimal grasp points, and a motion planner helps securely grasp the object, ensuring stable interaction before manipulation.

State construction: After grasping the object, the robot uses the interactive joint parameter estimation (Section IV-B) to infer the joint type and configuration $[\hat{\boldsymbol{h}}_{\boldsymbol{J}}, \hat{\boldsymbol{p}}_{\boldsymbol{J}}]$. This knowledge, along with the robot's end-effector position $\boldsymbol{p}_{\boldsymbol{i}}^{\boldsymbol{eef}}$, is used to construct the state for the manipulation policy. This object-centric state construction is crucial, as it enables *hard transfer* by aligning the policy learned in simulation with the object's real-world configuration, allowing direct application of the learned policies without further retraining.

Control execution: Once the state is constructed, the robot selects the policy π_J, which provides the forces to be applied based on the object's joint dynamics. The forces predicted by the policy are scaled and mapped to generate the desired end-effector (EEF) pose changes. These pose changes are then converted into joint torques via an Operational Space Controller (OSC) [26], ensuring smooth adaptation to variations in object weight and dynamics. The robot applies the specified forces until the manipulation task (e.g., opening a drawer or microwave door) is successfully completed.

The learned policy does not dynamically adjust force magnitude for unforeseen factors like friction or spring tension. Addressing this limitation is left for future work, particularly for handling novel objects with varying dynamics.

V. Experiments

We evaluate FLEX on three key aspects: learning efficiency in simulation, policy transferability across objects, and its task execution performance across different robots in simulation and in the real world.

A. Experimental Setup

The experiments were conducted in Robosuite [8], a simulation platform built on Mujoco, focusing on four manipulation tasks: opening cabinets, trashcans, microwaves, and dishwashers. Cabinets and dishwashers have prismatic joints, while microwaves and trashcans involve revolute joints. All objects are sourced from the PartNet-Mobility dataset [27].

We trained two force-based policies: one for prismatic joints (using a drawer) and another for revolute joints (using both a microwave and a dishwasher[1]). For revolute joints, either the microwave or dishwasher is randomly chosen at the start of each episode. During training, the agent is provided with ground-truth joint configurations and randomized force application points from accessible regions, such as handles and front panels, derived from the objects' URDF files.

Training efficiency is measured through wall time and timesteps. The simulation runs at 20 Hz, with each timestep corresponding to 0.05 seconds of simulated time. A convergence criterion is defined based on the alignment between the applied force and the object's movement direction: $\chi = \frac{1}{N}\sum_{t=0}^{N}\frac{a_t \cdot \Delta x}{\|a_t\|\|\Delta x\|} = \frac{1}{N}\sum_{t=0}^{N}\cos\theta_{a_t,\Delta x}$, where χ is the average percentage of force applied in the correct direction throughout the episode. Once χ averages at least 0.75 over the last 100 episodes, the policy is considered to have converged. This criterion yields a 100% success rate for both prismatic and revolute joint scenarios during training[2].

For force-based reinforcement learning, each training episode has 200 timesteps, while robot-centric reinforcement learning baselines extend to 500 steps per episode. Our method reduces the required timesteps and accelerates simulation by excluding the robot from the environment, thereby eliminating computational costs associated with collisions, physics, and robot control. The training process is conducted across ten independent trials, with the mean and standard deviation of the training times in Table I (columns 1& 2).

B. Evaluation and Metrics

We evaluate FLEX's ability to generalize to unseen objects and complete the manipulation task autonomously. The evaluation is conducted using 13 previously unseen objects from the PartNet-Mobility [28] dataset, including four microwaves, three dishwashers, three trashcans, and three cabinets. For each object, the robot is provided with a partial point cloud and must autonomously (1) infer the joint type, (2) grasp the object using AO-GRASP [25] and motion planner, (3) load the corresponding pre-trained policy, and (4) execute the manipulation skill. Success is defined as opening the object to at least 80% of its joint limit while maintaining a stable grasp[3]. To prevent "lucky success," any incorrect joint classification leads to immediate task failure.

To streamline the evaluation process, grasp proposals from the AO-GRASP model are generated once at the beginning of each trial and remain consistent throughout the 20 manipulation attempts for each object. The evaluation protocol is repeated for all 13 unseen objects, with 20 trials conducted per object. The success rate is computed for each class of objects, and we report the mean and standard deviation for 10 independent trials, as shown in Table I (column 3).

We also assess the transferability of the learned policies across different robots (Panda, UR5e, and Kinova Gen3) within the Robosuite environment, recording success rates for each manipulation task. To demonstrate the real-world applicability of our method, we conduct a proof-of-concept experiment using a UR5 robotic arm to open a drawer.[4]

C. Baselines

End-to-end RL: We implemented an end-to-end RL baseline using the TD3 algorithm from Stable-Baselines3 [29] with a hand-crafted, dense, staged reward function inspired by [3].

CIPS: We selected CIPS [14], a hybrid RL and planning method that reduces task horizon by planning up to the point of object contact. To speed up training, the planning phase

[1]Two objects were used due to the complexity of this joint type.

[2]All experiments, including baselines, were run on Intel® Core™ i9-13900 (5 GHz) and RTX 4090 GPU, with 24 processes for multi-process.

[3]Relaxed to 70% for baselines to avoid zero success rates.

[4]Video demonstration is available in the supplementary materials.

Algorithm	Training (timesteps)	Training (walltime)	Success Rate
	Mean±SD	Mean±SD	Mean±SD
Cabinet [Prismatic]			
FLEX	0.4 ± 0.26 M	0.36 ± 0.2 h	0.91 ± 0.08
End-to-end RL	≥ 10 M	≥ 24 h	≤ 0.01
CIPS	3 M	1.46 ± 0.02 h	0.34 ± 0.31
GAMMA+Planning	N/A	N/A	0.49 ± 0.29
Trashcan [Prismatic]			
FLEX	0.4 ± 0.26 M	0.36 ± 0.2 h	0.88 ± 0.10
End-to-end RL	≥ 10 M	≥ 24 h	≤ 0.01
CIPS	3 M	1.46 ± 0.02 h	0.19 ± 0.25
GAMMA+Planning	N/A	N/A	0.22 ± 0.18
Microwave [Revolute]			
FLEX	0.9 ± 0.3 M	0.63 ± 0.22 h	0.67 ± 0.14
End-to-end RL	≥ 10 M	≥ 24 h	≤ 0.01
CIPS	5 M	2.84 ± 0.2 h	0.33 ± 0.26
GAMMA+Planning	N/A	N/A	0.49 ± 0.18
Dishwasher [Revolute]			
FLEX	0.9 ± 0.3 M	0.63 ± 0.22 h	0.64 ± 0.11
End-to-end RL	≥ 10 M	≥ 24 h	≤ 0.01
CIPS	5 M	2.84 ± 0.2 h	0.55 ± 0.36
GAMMA+Planning	N/A	N/A	0.47 ± 0.38

TABLE I: Results on novel object instances evaluated using the Panda robot. Our method (FLEX) is highlighted in grey. Training time in hours, and timesteps in millions.

was skipped by pre-recording 4 instances per object, with the robot already grasping the object at the start of each episode.

GAMMA: To benchmark against a SOTA visual perception-based manipulation method, we selected GAMMA [21], a pre-trained model that infers joint configurations and grasp poses from partial point clouds. As GAMMA is a pre-trained model that uses planning, training time was not applicable.

All RL-based baselines were trained on the same objects and ground-truth observations as FLEX. During rollout, we provide ground-truth joint configuration and type for these baselines. We assume the ground-truth knowledge for joint type is needed to complete the task, and we select the correct result from multiple proposals given by GAMMA. To maintain consistency, we use AO-GRASP for grasp proposals in both CIPS and GAMMA.

VI. Results & Discussion

The results of our experiments are summarized in Tables I and II. It can be seen that FLEX consistently outperforms all baselines, including CIPS [2] and GAMMA [21], with success rates ranging from 64% to 91% across various objects. In some cases, FLEX achieves up to four times higher success rates, demonstrating its superior performance in task execution. Additionally, FLEX shows significantly lower standard deviations in success rates, emphasizing the stability and reliability of the learned policies.

FLEX significantly reduces training time, converging faster with fewer timesteps and less wall time, as shown in Table I. This efficiency is twofold: by decoupling the robot from the training environment, FLEX avoids simulating robot dynamics, collisions, and physics, accelerating training and reducing computational costs. Additionally, learning in force space reduces the action space and prevents inefficient exploration (e.g., detaching from objects), allowing FLEX to converge to a successful policy with fewer timesteps and improved sample efficiency compared to the baselines.

Object	Panda	UR5e	Kinova3
Cabinet	0.91 ± 0.08	0.82 ± 0.13	0.77 ± 0.12
Trashcan	0.88 ± 0.10	0.75 ± 0.15	0.79 ± 0.10
Microwave	0.67 ± 0.14	0.53 ± 0.08	0.55 ± 0.12
Dishwasher	0.64 ± 0.11	0.62 ± 0.11	0.59 ± 0.13

TABLE II: Success rates for different objects across different robotic platforms.

In contrast, the end-to-end RL baseline struggles to perform the task, even after 10 million timesteps of training, achieving negligible success rates despite using a staged reward function [3]. This highlights the difficulty of the task, particularly in balancing exploration and learning in high-dimensional action spaces – in robot-centric environments.

CIPS [14] performs moderately well, but inefficient exploration limits its stability. Once the robot enters undesirable states (e.g., detaching from the object), it tends to repeat these behaviors, resulting in poor performance in some trials.

GAMMA performs decently on most object types but struggles with the *trashcan* task, where it fails to infer the correct joint type. This leads to errors in joint configuration inference and subsequent control, highlighting its difficulty in generalizing to unseen objects outside its training set.

A key advantage of FLEX is its transferability across robots. By decoupling manipulation skills from robot-specific dynamics, FLEX can transfer to various robots (Panda, UR5e, Kinova Gen3) without retraining. As shown in Table II, FLEX performs consistently across robots with minimal degradation, whereas CIPS and GAMMA require significant tuning and adaptation, limiting their scalability.

VII. Conclusion & Future Work

We introduced FLEX, a framework for force-based learning of manipulation skills that decouples robot-specific dynamics from policy learning. FLEX enhances training efficiency, reduces action space complexity, and generalizes well across various articulated objects and robotic platforms. Our evaluations demonstrate FLEX's high success rates and consistent performance across multiple robot platforms, including transfer to real-world settings, as evidenced by a UR5 robot manipulating a drawer.

Future work will address the current lack of dynamic force adjustments in response to varying object properties, such as weight, friction, or joint damping. Another promising direction will involve FLEX autonomously generating task-specific environments in real-time, allowing it to adapt and learn policies for novel goals. This would further enhance its effectiveness in open-world scenarios. Finally, integrating learning from demonstrations could improve data efficiency, making the framework more adaptable and robust.

ACKNOWLEDGMENTS

We thank the reviewers for their valuable feedback and suggestions. We also thank Brennan Miller-Klugman for his valuable help in setting up the robots. This work was supported in part by ONR grant N00014-24-1-2024.

REFERENCES

[1] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems*, vol. 156, p. 104224, 2022.

[2] B. Abbatematteo, E. Rosen, S. Thompson, T. Akbulut, S. Rammohan, and G. Konidaris, "Composable interaction primitives: A structured policy class for efficiently learning sustained-contact manipulation skills," in *Proceedings of the 2024 IEEE Conference on Robotics and Automation*. Proceedings of the 2024 IEEE Conference on Robotics and Automation, 2024.

[3] K. Lu, B. Yang, B. Wang, and A. Markham, "Decoupling skill learning from robotic control for generalizable object manipulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 864–870.

[4] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of machine learning research*, vol. 22, no. 30, pp. 1–82, 2021.

[5] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[6] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on robot learning*. PMLR, 2018, pp. 651–673.

[7] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[8] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," *arXiv preprint arXiv:2009.12293*, 2020.

[9] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, L. Yi, A. X. Chang, L. J. Guibas, and H. Su, "Sapien: A simulated part-based interactive environment," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[10] Í. Elguea-Aguinaco, A. Serrano-Muñoz, D. Chrysostomou, I. Inziarte-Hidalgo, S. Bøgh, and N. Arana-Arexolaleiba, "A review on reinforcement learning for contact-rich robotic manipulation tasks," *Robotics and Computer-Integrated Manufacturing*, vol. 81, p. 102517, 2023.

[11] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4583–4590.

[12] S. Nasiriany, H. Liu, and Y. Zhu, "Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7477–7484.

[13] K. Pertsch, Y. Lee, and J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Conference on robot learning*. PMLR, 2021, pp. 188–204.

[14] E. Rosen, B. M. Abbatematteo, S. Thompson, T. Akbulut, and G. Konidaris, "On the role of structure in manipulation skill learning," in *CoRL 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*, 2022.

[15] M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, and D. Fox, "Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning," in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 7505–7512.

[16] C. Thierauf and M. Scheutz, "Fixing symbolic plans with reinforcement learning in object-based action spaces," *learning*, vol. 21, p. 22, 2024.

[17] P. Li, T. Liu, Y. Li, M. Han, H. Geng, S. Wang, Y. Zhu, S.-C. Zhu, and S. Huang, "Ag2manip: Learning novel manipulation skills with agent-agnostic visual and action representations," *arXiv preprint arXiv:2404.17521*, 2024.

[18] K. Mo, L. J. Guibas, M. Mukadam, A. Gupta, and S. Tulsiani, "Where2act: From pixels to actions for articulated 3d objects," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6813–6823.

[19] R. Wu, Y. Zhao, K. Mo, Z. Guo, Y. Wang, T. Wu, Q. Fan, X. Chen, L. Guibas, and H. Dong, "Vat-mart: Learning visual action trajectory proposals for manipulating 3d articulated objects," *arXiv preprint arXiv:2106.14440*, 2021.

[20] Y. Geng, B. An, H. Geng, Y. Chen, Y. Yang, and H. Dong, "Rlafford: End-to-end affordance learning for robotic manipulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5880–5886.

[21] Q. Yu, J. Wang, W. Liu, C. Hao, L. Liu, L. Shao, W. Wang, and C. Lu, "Gamma: Generalizable articulation modeling and manipulation for articulated objects," 2024. [Online]. Available: https://arxiv.org/abs/2309.16264

[22] B. Eisner, H. Zhang, and D. Held, "Flowbot3d: Learning 3d articulation flow to manipulate articulated objects," *arXiv preprint arXiv:2205.04382*, 2022.

[23] Z. Xu, Z. He, and S. Song, "Universal manipulation policy network for articulated objects," *IEEE robotics and automation letters*, vol. 7, no. 2, pp. 2447–2454, 2022.

[24] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[25] C. P. Morlans, C. Chen, Y. Weng, M. Yi, Y. Huang, N. Heppert, L. Zhou, L. Guibas, and J. Bohg, "Ao-grasp: Articulated object grasp generation," *arXiv preprint arXiv:2310.15928*, 2023.

[26] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[27] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang *et al.*, "Sapien: A simulated part-based interactive environment," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 097–11 107.

[28] P. Xie, R. Chen, S. Chen, Y. Qin, F. Xiang, T. Sun, J. Xu, G. Wang, and H. Su, "Part-guided 3d rl for sim2real articulated object manipulation," *IEEE Robotics and Automation Letters*, 2023.

[29] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

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 III: Agent Network Struture

Parameter	Value
Discount Factor (γ)	0.99
Batch Size	100
Learning Rate	0.001
Policy 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
max force magnitude (η)	0.02

TABLE IV: 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 designed to guide the robot through sequential stages: approaching, grasping, and manipulating the object. . Despite the use of a dense reward function, the poor (almost nil) performance highlights the complexity of the task and demonstrates why *conventional* RL-based control is an inefficient approach.

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.

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

TABLE V: Baseline Hyperparameters for CIPS

We observed that GAMMA tends to overestimate the number of joints on complex objects with multiple moving parts, leading to incorrect joint inferences. To mitigate this, we only select the first inference result that complies with our ground-truth knowledge of the joint type. If no correct inference exists, we randomly choose from the remaining detected joints.

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 stayed 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.