Supplementary Material for
Learning Reusable Manipulation Strategies

Table of Contents

A Method Detail

A.1 Planning Algorithm

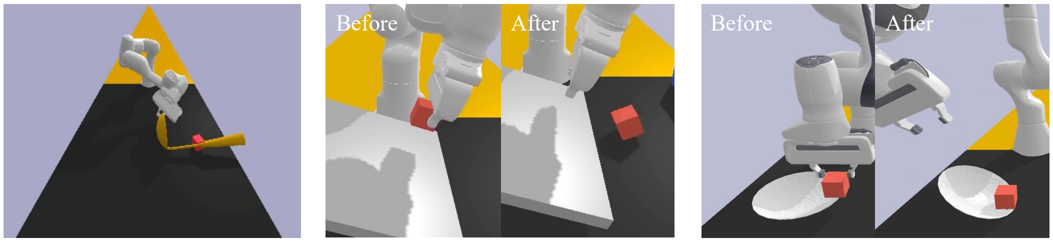
Algorithm 2 shows the bi-level search algorithm we use. In the discrete search level, we enumerate both basis operations as well as mechanism operations. During the continuous parameter search phase, for basis operations instantiated from mechanisms, we use the mechanism-specific sampler rather than the generic sampler for continuous parameters.

Algorithm 2 Bilevel Search Algorithm

procedure BILEVELSEARCH(s_0, g, operator_schemas)
    plan_gen $\leftarrow$ SymbolicSearch(s_0, g, operator_schemas)   ▷ Explore discrete plans
    for all *plan* $\in$ *plan_gen* **do**   ▷ For all candidate sequences of basis operations
        CONTINUOUSSEARCH(s_0, g, *plan*)
procedure CONTINUOUSSEARCH(s_0, g, *plan*)
    grounded_plan $\leftarrow \emptyset$; $s \leftarrow s_0$
    for all *op* $\in plan$ **do**
        for all *arg* $\in$ *op.args* **do**
            arg $\leftarrow$ InvokeSampler(*op.sampler*)   ▷ Generate continuous parameters
        if CheckPrecondition(*op*, s) **then**
            $s \leftarrow \mathcal{T}(s, op)$ ▷ Simulate the operator with sampled parameters.
            grounded_plan $\leftarrow$ *grounded_plan* $\cup \{op\}$
        else break
    if IsGoalAchieved(*grounded_plan*, g) **then**
        return *grounded_plan*   ▷ Return the first plan that achieves the goal
    return *empty*

A.2 Briefly Dynamic Manipulation

The system can handle robot-object and object-object contact without assuming quasi-static motion. For example, when placing objects on surfaces, we consider subsequent pose changes: objects placed on inclined surfaces may slide down, and heavy objects placed on levers can alter the pose of the lever. Formally, we assume a *briefly-dynamic* manipulation setting, where the robot controller is

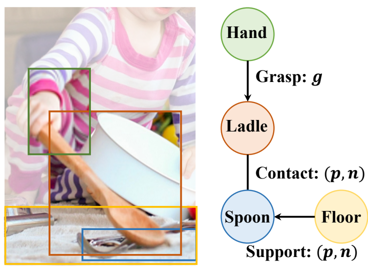


Case 1: non-rigid attachments between objects while moving.

Case 2: object pose change after placement due to physics.

Case 3: support object pose change after new objects being placed.

Figure 7: Illustration of three briefly-dynamic manipulation scenarios in the paper.




```
(:action move-with-contact
 :parameters (?tool - item ?target - item ?support - item
              ?param - contact-move-param ?qt - trajectory)
 :precondition (and
   (holding ?tool)
   (support ?target ?support)
   (valid-contact-move-param      ... ?param)      ;; points, normal
   (valid-contact-move-trajectory ... ?param ?qt)) ;; robot trajectory
 :effect (and
   (assign (robot-config) ...)   ;; update robot position
   (assign (pose ?tool)   ...)   ;; update tool position
   (assign (pose ?target) ...)  ;; update target position
```

Figure 8: The modeling of the robot basis operation *move-with-contact* using a STRIPS-like syntax.

position control-based, and manipulated objects may experience acceleration and velocity until they reach a stable configuration. Figure 7 illustrates a few examples of briefly-dynamic manipulation tasks handled by our sampler and planner.

A.3 Basis Operations

In our manipulation context, each schema represents either a robot action that does not change the contact mode graph (e.g., moving the arm without object collisions) or a primitive action that modifies the contact mode (e.g., grasping an object). Table 1 shows the complete list of basis operators used in this paper.

The precondition and the effect of an action schema describe the contact relationships between objects before and after the execution. Fig. 8 showcases a concrete example. The action schema involves three objects: the object being held *?tool*, the target object that is in contact with the tool object *?target*, and the object that is currently supporting the target object *?support*. Additionally, there are two continuous parameters: *?param* specifies the contact between *?target* and *?tool*, including the contact surface and contact normal; *?qt* specifies the robot arm trajectory as a sequence of joint-space waypoints. This action updates the robot joint angles, and the poses of *?tool* and *?target*. Given the discrete and continuous parameters, we use a joint-space position controller to execute the actions and use the execution results to update the state variables.

We will present the implementation details for the samplers associated with each operator in the next section (Appendix A.4). At a high level, these samplers are designed to be very generic: for grasping, it randomly samples two parallel surfaces on objects; for object-object contact, it randomly samples two surfaces on the object and then transforms the object held by the robot so that two surface normals point to each other.

A.4 Samplers for Generic Operations

Recall that there are three types of continuous variables to be sampled for the basis operators described in Table 1: grasping poses relative to an object (represented as $SE(3)$ poses of the end-effector relative to the object), placement poses (represented as $SE(3)$ poses in the support object frame), contacts between two objects (represented as the $SE(3)$ pose of object 1 in the frame of object 2), and robot arm trajectories (represented as a sequence of arm trajectories). Here, we supplement the list of samplers we use to generate these continuous parameters. They are designed to be generic, relying solely on geometry and not specific object semantics (e.g., soup ladle grasping).

Grasp (G). The grasp sampler, $G(\mathcal{O}, T_o)$, accepts the object's shape and current pose, $\mathcal{O}$ and T_o respectively, and identifies two "parallel" surfaces on the object mesh, represented as (p_1, n_1) and (p_2, n_2), where p_1 and p_2 are two points and n_1 and n_2 are surface normals. The definition of being parallel is that: $(p_1 - p_2) \cdot n_1 = 1$ and $n_1 \cdot n_2 = -1$. It then computes a corresponding robot end-effector pose T_{ee} such that T_ee centered at the midpoint between p_1 and p_2, and T_{ee} is perpendicular to n_1. It then checks the distance between two surfaces so that the parallel gripper can hold the object at T_{ee}. Finally, it checks the reachability of T_{ee} using an inverse-kinematics solver.

Placement (P). The placement position sampler, $P(\mathcal{O}_1, \mathcal{O}_2, T_{o2})$, considers the shapes of both the holding object, $\mathcal{O}_1$, and the target support object, $\mathcal{O}_2$, and the pose of $\mathcal{O}_2$. It randomly samples two surfaces, represented as (p_1, n_1) and (p_2, n_2), one on each object such that $n_2 \cdot (0, 0, 1)^T > 0.9$ (i.e., n_2 is close to the $+z$ direction). Next, it solves for a transform T on $\mathcal{O}_1$ such that $Tp_1 = T_{o2}p_2$ and $Tn_1 = -T_{o2}n_2$ (essentially place p_1 on $\mathcal{O}_1$ at p_2 and pointing towards n_2).

Object Contact (C). For both robot-object and object-object contact, the object contact sampler, $C(\mathcal{O}_1, \mathcal{O}_2, T_{o2}, \mathcal{O}_s, T_s)$ takes five arguments, including the current holding object $\mathcal{O}_1$ (or the robot gripper itself when not holding anything), the object to contact $\mathcal{O}_2$ and its current pose T_{o2}, and the object that supports $\mathcal{O}_2$: $\mathcal{O}_s$ and its pose T_s. It first randomly samples two surfaces, represented as (p_1, n_1) and (p_2, n_2) on $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. Since we do not consider pushing $\mathcal{O}_2$ "towards" the supporting object $\mathcal{O}_s$, we additionally require that n_2 is perpendicular to n_s, which is the direction of the support force from $\mathcal{O}_s$ to $\mathcal{O}_2$. Next, it solves for a transform T on $\mathcal{O}_1$ such that $Tp_1 = T_{o2}p_2$ and $Tn_1 = -T_{o2}n_2$ (essentially place p_1 on $\mathcal{O}_1$ at p_2 and pointing towards n_2 to excert force).

Trajectory (T). For grasping and placement trajectories, the trajectory sampler, $T(T_{init}, T_{target})$, considers the initial and target end-effector pose of the robot gripper. It first uses an inverse kinematic solver to solve for two robot configurations at the designated end-effector pose q_{init} and q_{target}. Next, we compute a collision-free trajectory (except for collisions with the object being held and the object to contact) using a Bidirectional Rapidly-exploring Random Tree (BiRRT) algorithm.

For move-with-contact trajectories, the trajectory sampler, $T(T_{\text{init}}, p_1, n_1, p_2, n_2)$, accepts the initial configuration of the robot, g_{init}, and the contact surfaces on the two objects sampled using the object contact sampler C: (p_1, n_1) and (p_2, n_2). It proceeds to randomly sample a "push" distance, d, along the contact normal direction, n_1, from a uniform distribution in the range $[0.05, 0.25]$ meters. Subsequently, the sampler generates the arm trajectory by invoking the BiRRT algorithm to follow a set of waypoints corresponding to a linear Cartesian-space motion along n_1 by distance d.

B Experiment Detail

B.1 Mechanism Learning Setup

Our evaluation encompasses six distinct mechanisms, grouped into two categories: the first four tasks assess “tool-use.”

(Edge) pushing objects to the edge of a table for pickup. There are four object models used in this mechanism: plate, calculator, caliper, and document.

(Hook) using tools to reach for distant objects. There are five objects that can be used as the “hook:” wooden L-shape stick, soup ladle, hammer, spoon, and caliper.

(Lever) flipping objects using heavy objects as levers. There are four “heavy” objects that can be used to flip the plate: box, spoon, dipper, and walnut.

(Poking) using tools to poke objects out of a tunnel. There are three object models that can be used as the “poking” tool: wooden stick, spatula, and spoon.

The remaining two tasks fall under the “reasoning about stability” category.

(Center-of-Mass) achieving stable object placement on another object. There are three object models to be placed on the small block: plate, calculator, and document.

(Slope-and-Blocker) using objects as blockers to prevent objects from falling off inclined surfaces. There are three object models that can be used as the blocker: wooden stick, wooden L-shape stick, and spoon.

For each environment, we first manually defined a canonical pose for each object such that the mechanism is feasible. Next, for each training and testing instance, we randomly apply small translations (a uniform distribution within $\pm$ 5 centimeters) and small rotations (uniform within $\pm$ 15 degrees) to the canonical pose of each movable object.

B.2 Sampler Learning

Taking a closer look at the importance of sampler learning, Fig. 9 illustrates a breakdown of the number of samples required for the “hook use” mechanism using our planning algorithm, with the generic sampler and with the learned sampler. Fig. 3b shows the inferred macro definition for this mechanism, and here we count the number of samples produced by each individual sampler. In this case, most of the samplers are produced to generate candidate grasping poses of the tool and possible contacts between the tool and the target (i.e., how to reach the tool).

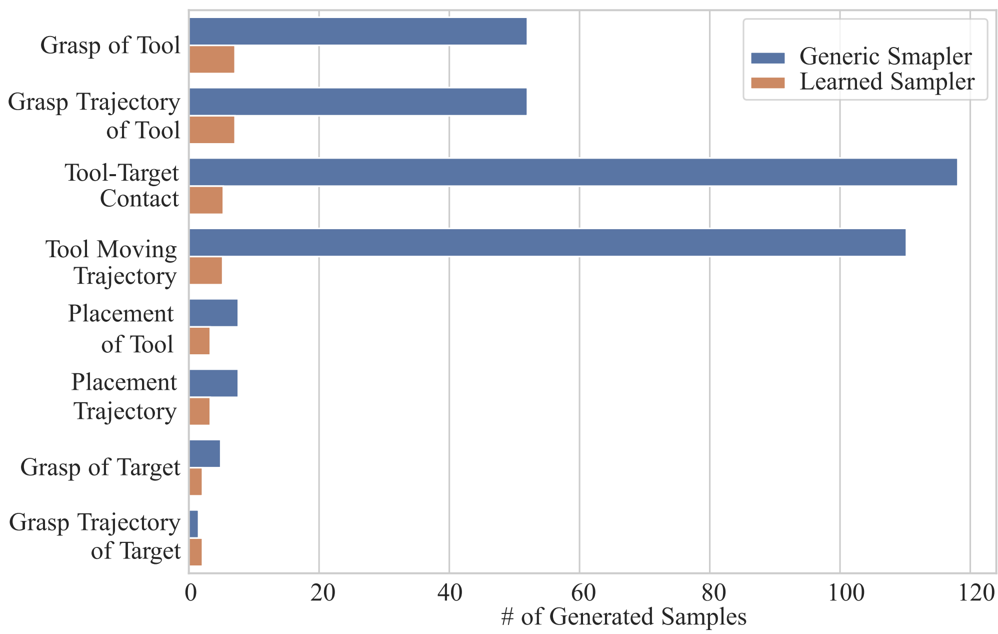


Figure 9: Breakdown of samples produced by different samplers for the hook-using task.