Object-Centric Kinodynamic Planning for Nonprehensile Robot Rearrangement Manipulation

Kejia Ren[1], Gaotian Wang[1], Andrew S. Morgan[2], Lydia E. Kavraki[1], and Kaiyu Hang[1]

Abstract—Nonprehensile actions such as pushing are crucial for addressing multi-object rearrangement problems. To date, existing nonprehensile solutions are all *robot-centric*, i.e., the manipulation actions are generated with robot-relevant intent and their outcomes are passively evaluated afterwards. Such pipelines are very different from human strategies and are typically inefficient. To this end, this work proposes a novel *object-centric* planning paradigm and develops the first *object-centric* planner for general nonprehensile rearrangement problems. By assuming that each object can actively move without being driven by robot interactions, the *object-centric* planner focuses on planning desired object motions, which are realized via robot actions generated online via a closed-loop pushing strategy. Through extensive experiments and in comparison with state-of-the-art baselines in both simulation and on a physical robot, we show that our *object-centric* paradigm can generate more intuitive and task-effective robot actions with significantly improved efficiency. In addition, we propose a benchmarking protocol to standardize and facilitate future research in nonprehensile rearrangement.

Index Terms—Nonprehensile Manipulation, Multi-object Rearrangement, Object-centric Planning.

I. INTRODUCTION

Rearrangement of multiple objects, which refers to reconfiguring objects into certain desired states, is generally required for various practical manipulation tasks such as singulation for object retrieval [1], [2], obstacle clearance for navigation [3]–[5], multi-object sorting [6], etc. As an essential manipulation skill of robots, rearrangement is enabled by planning the robot's motion with necessary constraints such as collision avoidance and the robot's kinematics, which is proven NP-hard [7]. Traditional planning methods allow only a pick-and-place type of prehensile action to move one object at a time [8], [9]. Although certain optimality can be achieved, prehensile action-based rearrangement can be ineffective or even infeasible since the object geometry (e.g., size or shape) or other environment-relevant constraints (e.g., limited free space) can make the objects not graspable.

Thereafter, nonprehensile actions such as pushing have been investigated. Nonprehensile action-based methods can generate more diverse and effective solutions to object rearrangement, by modeling the interaction physics between the robot and objects and allowing concurrent manipulation of multiple objects [10]. However, nonprehensile rearrangement planning is challenging due to the sophisticated physics modeling and
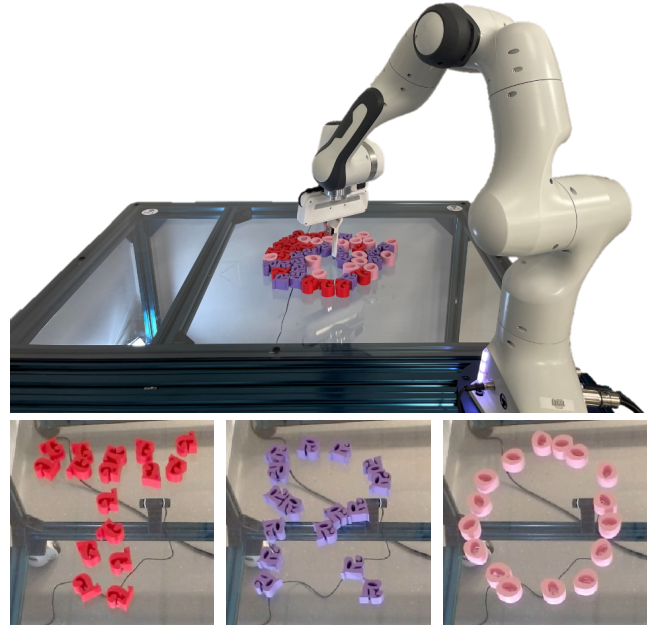
Fig. 1: Through object-centric planning, our framework is able to efficiently rearrange multiple movable objects of different shapes to accomplish various tasks. In the scene, "T", "R", and "O" letter-shaped objects are rearranged to form the abbreviation "TRO".

the high-dimensional problem space it entails. Furthermore, the inevitable modeling inaccuracy of the involved physics makes the nonprehensile solutions less robust to real-world uncertainties and causes task failures. To this end, developing an efficient and reactive manipulation planner that can solve general object rearrangement problems is highly desired. In this work, we develop a manipulation planner for large-scale nonprehensile rearrangement problems where the object clutters are highly packed and concurrent interactions between objects are inevitably common, as exemplified in Fig. 1.

Traditional Robot-Centric Planning. Existing nonprehensile rearrangement planners are all *robot-centric*, meaning that the actions are generated without specific object-relevant intent [11]. Specifically, during the planning process, such robot actions (e.g., in the robot's joint space) are randomly sampled, considering only the robot-relevant geometric and kinematic constraints; no consideration is given to how the state of objects is affected when sampling actions. The outcome of each action is predicted after this action has already been generated via sampling. In the end, from all sampled actions, the actions whose predicted outcomes lead to the task completion will be selected for execution [12], [13].

	Object-Centric	Nonprehensile	Data-Driven	Closed-Loop	Generalizable	Explicit-Goal	Max # Objects	
							Sim	Real
[14], [15]	✗	✗	✗	✗	✗	✓	30 - 50	12
[11]–[13]	✗	✓	✗	✗	✗	✗	6 - 7	7
[16], [17]	✗	✓	✗	✓	✗	✗	15 - 40	9 - 16
[18]–[21]	✗	✓	✓	✓	✗	✗	11 - 40	8 - 30
[22]	✗	✓	✗	✗	✓	✓	100	32
[23]	✗	✓	✗	✓	✗	✗	200	20
[24]	✗	✓	✗	✗	✗	✗	10	–
[25], [26]	✗	✓	✗	✓	✓	✗	36	20
Ours	✓	✓	✗	✓	✓	✗	100	32

Fig. 2: Qualitative comparison of the state-of-the-art rearrangement solutions. Each column represents one characteristic of the solution, from left to right: 1) *Object-Centric:* the proposed method is object-centric, otherwise, robot-centric; 2) *Nonprehensile:* nonprehensile actions are incorporated; 3) *Data-Driven:* the method is data-driven and requires extra time for training; 4) *Closed-Loop:* the method generates closed-loop motion plans that can handle real-world uncertainties; 5) *Generalizable:* the method has shown the capability to transfer across different rearrangement tasks (e.g., relocating, separating, sorting, etc); 6) *Explicit-Goal:* the method requires explicit goal pose or location for each object. 7) *Max # Objects:* the maximum number of movable objects the method can deal with, as shown by the corresponding (simulation and real-world) experiments.

Object-Centric Planning. However, imagining how humans rearrange objects, they never follow this *robot-centric* paradigm: first randomly sampling tens or even hundreds of actions in mind and then selecting the promising ones based on the predicted outcomes. Instead, a more intuitive strategy is to reverse this process: first actively reasoning about how the objects should be moved and then trying to achieve the desired object motions through available manipulation skills. Such a strategy is called *object-centric* since each generated action will have a specific purpose of achieving certain desired object motions (e.g., moving target objects in desired directions or to desired locations). Inspired by this intuition, to the best of the authors' knowledge, we develop the first-ever *object-centric* manipulation planner in this work for nonprehensile rearrangement problems. The proposed *object-centric* planner first actively searches for desired object motions, by assuming that the objects can move on their own without considering the robot; it then tries to achieve the computed object motions through robot actions generated in real-time, e.g., by a closed-loop pushing strategy, to progressively transit the system towards the task goal. By leveraging task-relevant information to efficiently explore and proactively reason about desired action outcomes, our proposed *object-centric* planner can

1) significantly improve the planning efficiency in seeking desired rearrangement actions (e.g., taking only about 4 seconds of planning time to sort 4 classes of 24 objects);
2) effectively guide the generation of *object-centric* robot actions that are more intuitive and effective to complete the task faster;
3) generalize and scale to various large-scale rearrangement tasks with different task objectives while being robust against physical uncertainties.

The rest of the paper is structured as follows. We review related literature in Sec. II and provide preliminaries in Sec. III. Under the novel *object-centric* paradigm, we formulate the nonprehensile rearrangement planning problem in Sec. IV. We present the algorithmic details of our proposed planner in Sec. V, and introduce how real-time robot actions are generated for execution in Sec. VI. By experimental comparisons with state-of-the-art methods in simulation and real

world, we evaluate our planner in Sec. VII. Furthermore, based on our highly efficient rearrangement solution and extensive experimental results, we propose a benchmarking protocol in Sec. VIII to facilitate research in nonprehensile rearrangement. Finally, we conclude in Sec. IX.

II. RELATED WORK

Fig. 2 qualitatively compares state-of-the-art rearrangement solutions with our proposed planner from multiple aspects shown by the columns of the table. Our proposed planner is the only large-scale nonprehensile solution that can generalize to different task setups without necessarily requiring explicit goal definition, while not being data-consuming. Next, we review literatures that relate to our work from multiple perspectives.

Nonprehensile Manipulation and Planar Pushing: Prehensile manipulation which relies on robotic grasping, has been extensively studied to facilitate many manipulation tasks [27]–[29]. As essential skills complementary to prehensile manipulation, nonprehensile manipulation (defined as manipulation without grasping [30]) such as pushing [31], sliding [32]–[34], and pivoting [35], [36], has been studied to bring more possibilities of manipulation ranging from single object reconfiguration [37] to large-scale object rearrangement in clutters [38]–[41]. Among these, pushing actions have been widely used in solving practical tasks [42], due to their capability to work in confined workspaces. Planar pushing of a single object is one of the simplest scenarios of pushing-based manipulation. Existing works have developed analytical models [10], [43]–[48] to address the precise planar pushing problem. However, since they require known contact geometry and physical properties, and are usually derived under over-simplified assumptions such as convex shapes, these analytical approaches are not easily scalable in a real-world setup. More recently, data-driven methods [49]–[53] have been studied for planar pushing under more realistic challenges. However, the learned models in general require vast amounts of data for training and are hard to generalize over different task setups and perception domains without fine-tuning. So far, the research on planar pushing mainly focuses on applications involving a single target object, while the developed pushing

strategies have not been widely incorporated into large-scale problems such as rearrangement to concurrently manipulate multiple movable objects.

Kinodynamic Planning: When system dynamics is involved due to physical interactions between the robot and the environment, the modeling of such physics [54], [55] needs to be incorporated into the planning process to guide the motion of the robot. As such, kinodynamic planning has been proposed to generate actions that comply with the physical constraints of the system [56]–[58]. In nonprehensile manipulation such as object pushing, the manipulation outcome is hard to precisely predict, due to the inaccurate and simplified modeling of the interaction physics or perception and execution uncertainties. Such real-world uncertainties can easily cause the robot motions generated by kinodynamic planning to fail the real-world manipulation. To address such challenges caused by uncertainties, solutions [16], [59], [60] have been proposed by incorporating an uncertainty model into the planning process, which generates conservative motions and reduces the probabilities of execution failure. Another solution is to close the loop of planning and execution by iterative replanning with receding horizons [25], [26], [61].

Rearrangement-based Manipulation: Object rearrangement is common and important, involving manipulation of small objects in a confined space [62], [63] or heavy objects in a large space [64], [65]. By using mainly prehensile actions (e.g., pick-and-place) without considering the physics, the rearrangement problem can be reduced to a geometric problem with a discrete action space, and long-horizon problems can be solved efficiently under such simplifications. Some of the prehensile approaches use graph-based or tree-based search [14], [15], [66]–[71] to provide near optimal solutions; some others use learned models [72]–[77] to address more complex contraints. However, pick-and-place type of prehensile actions can be infeasible when the objects are not graspable by the robot, e.g., in a tightly packed environment with limited free space. By incorporating nonprehensile actions such as pushing, more diverse and efficient solutions can be generated. By predicting the outcome of physical interactions through an approximated analytical model or a physics engine, sampling-based planners [12], [13], [24] have been proposed to generate open-loop motion plans. However, due to the inaccuracy of the physics model and real-world uncertainties, the execution of the generated plans will easily fail, especially for long-horizon tasks where the error significantly accumulates. To address such real-world challenges through closed-loop execution, strategies such as online replanning [17], distance-guided greedy local search [22], [23], interleaving progress-controlled planning and execution [25], [26] need to be enabled. All existing nonprehensile methods are robot-centric while our proposed planner is object-centric. Driven by the recent advance in deep learning, nonprehensile rearrangement policies can be learned from demonstration or experience data to facilitate tasks such as pushing-based object relocation [78], multi-object sorting [20], [79], [80], object singulation for retrieval [19], [21] and object separation for clutter removal [18], [81]. Such data-driven approaches enable real-time action generation directly from raw image inputs. However, they are generally data-consuming and difficult to transfer in different task setups [82]. For example, a policy trained for separating objects can be inefficient in relocating an object.

III. PRELIMINARIES

In general, the nonprehensile object rearrangement problem involves a robot manipulator interacting with N movable objects in a bounded workspace $\mathcal{W} \subset \mathbb{R}^2$ on a 2D plane. We assume the objects are moved by the robot in a quasi-static manner without rolling or flipping. The objective of the problem is to find a sequence of feasible robot actions to manipulate the movable objects, such that they are rearranged to certain desired states while being restricted within $\mathcal{W}$.

A. Kinodynamic Planning for Nonprehensile Rearrangement

The nonprehensile rearrangement problem is conventionally formulated as a kinodynamic planning problem on the composite configuration space of the robot and all objects, for which we introduce the following definitions:

1) Configuration Space of the Robot: Formally, the robot's configuration space is denoted by $\mathcal{Q}^R \subset \mathbb{R}^M$ where $M \in \mathbb{Z}^+$ is the robot's degrees of freedom. $\mathcal{Q}^R_{free} \subset \mathcal{Q}^R$ is the free configuration space of the robot in which the robot does not collide with itself or the static environment, nor exceed its joint limits. Note that contacts between the robot and the movable objects are allowed as they are needed for manipulation. A robot state at time t is denoted by $\boldsymbol{q}_t \in \mathcal{Q}^R$.

2) Configuration Space of All Objects: For a single object, we denote its configuration space by $\mathcal{Q}^i \in SE(2)$ where $i = 1, \cdots, N$. The composite configuration space of all movable objects can be then represented by the Cartesian product $\mathcal{Q}^O = \mathcal{Q}^1 \times \cdots \times \mathcal{Q}^N$. The state of all objects at time t, called an arrangement, is denoted by a tuple $\boldsymbol{s}_t = \left(\boldsymbol{s}_t^1, \cdots, \boldsymbol{s}_t^i, \cdots, \boldsymbol{s}_t^N\right) \in \mathcal{Q}^O$, where $\boldsymbol{s}_t^i = \left(\boldsymbol{p}_t^i, \theta_t^i\right) \in SE(2)$ is the state, i.e., the position $\boldsymbol{p}_t^i = \left(x_t^i, y_t^i\right) \in \mathbb{R}^2$ and the orientation $\theta_t^i \in SO(2)$, of the i-th object.

The problem space of kinodynamic planning is the composite space $\mathcal{Q} = \mathcal{Q}^R \times \mathcal{Q}^O$ of the entire system, consisting of the configurations of the robot and all objects. For a system state to be valid at time t, the robot has to be collision-free $\boldsymbol{q}_t \in \mathcal{Q}^R_{free}$ and the positions of all objects have to stay inside the bounded workspace $\boldsymbol{p}_t^i \in \mathcal{W}, \forall i$. The collisions between objects are allowed to enable concurrent object-object interactions.

3) Action Space: The action space $\mathcal{A}$ consists of all actions the robot is allowed to perform, which can have different representations depending on the action definition used for a specific task. For a M-DoF robot manipulator as an example, we can use the joint velocity space as the action space, i.e., $\mathcal{A} \subset \mathbb{R}^M$, where an action $\boldsymbol{a}_t \in \mathcal{A}$ is an instantaneous joint velocity commanded to move the robot.

4) System Dynamics: Under the quasi-static assumption, the entire system composing the robot and all movable objects is modeled as a discrete-time dynamic system. The system dynamics is represented by a transition function $\Gamma : \mathcal{Q}^R \times \mathcal{Q}^O \times \mathcal{A} \mapsto \mathcal{Q}^R \times \mathcal{Q}^O$ in the following form:

$$(\boldsymbol{q}_{t+1}, \boldsymbol{s}_{t+1}) = \Gamma(\boldsymbol{q}_t, \boldsymbol{s}_t, \boldsymbol{a}_t) \qquad (1)$$
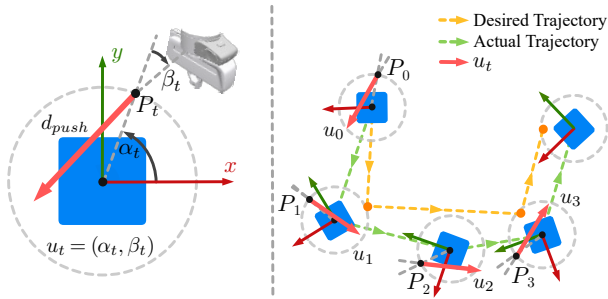
Fig. 3: *Left:* A robot pushing action is represented by $\boldsymbol{u}_t = (\alpha_t, \beta_t)$, where α_t and β_t are two angles specified in the object's body frame. To perform $\boldsymbol{u}_t$, the robot needs to first place its pusher at the position P_t (determined by α_t) and then translate in the direction of β_t by a distance d_{push} to interact with the object (blue cube) through the push (orange arrow); *Right:* By consecutively generating and executing the pushing actions $\boldsymbol{u}_0$ through $\boldsymbol{u}_3$, the robot can push the object to follow a desired reference trajectory (yellow dashed lines), as a sequence of desired poses or positions of the object. With a closed-loop pushing strategy, the object's actual trajectory (green dashed lines) due to execution will not deviate much from the reference.

which infers the outcome robot state $\boldsymbol{q}_{t+1}$ and the outcome arrangement (i.e., the state of all objects) $\boldsymbol{s}_{t+1}$, given the current robot state $\boldsymbol{q}_t$, the current arrangement $\boldsymbol{s}_t$, and action $\boldsymbol{a}_t$ executed by the robot, based on the real-world physics laws.

5) Goal Criterion: To determine whether all objects are successfully rearranged, we need to define a criterion function $g : \mathcal{Q}^O \mapsto \{0, 1\}$. Given an arrangement $\boldsymbol{s}_t \in \mathcal{Q}^O$, $g(\boldsymbol{s}_t) = 1$ indicates all the objects are rearranged to their desired configurations. Some existing works alternatively use an explicitly given goal region $\mathcal{G}_i \subset \mathcal{W}$ for each object i and check whether the object lies inside the goal region or not to determine the task completion. However, for some rearrangement tasks, only relative configurations for some objects are needed, as will be shown by example tasks in Sec. VII. Obtaining an explicit form of $\mathcal{G}_i$ is unnecessary and difficult for such tasks. Therefore, a criterion function g is a more general option for representing the rearrangement goal.

We consider a robot, at its initial configuration $\boldsymbol{q}_0$, to rearrange multiple objects from their initial arrangement $\boldsymbol{s}_0 = (\boldsymbol{s}_0^1, \cdots, \boldsymbol{s}_0^i, \cdots, \boldsymbol{s}_0^N)$. The goal of the nonprehensile rearrangement problem can be formally defined as finding a sequence of T robot actions, denoted as $\tau = \{\boldsymbol{a}_0, \cdots, \boldsymbol{a}_t, \cdots \boldsymbol{a}_{T-1}\}$, such that:

1) The system state transitions under the system dynamics by $(\boldsymbol{q}_{t+1}, \boldsymbol{s}_{t+1}) = \Gamma(\boldsymbol{q}_t, \boldsymbol{s}_t, \boldsymbol{a}_t)$, $t = 0, \cdots, T-1$.
2) For all intermediate time steps $t = 1, \cdots, T$, the system state is always valid: The robot is in its free C-space $\boldsymbol{q}_t \in \mathcal{Q}_{free}^R$, and all objects are within the workspace $\boldsymbol{p}_t^i \in \mathcal{W}, \forall i = 1, \cdots, N$.
3) The final arrangement, after the sequence of robot actions τ has been executed in the desired order, satisfies the goal criterion, i.e., $g(\boldsymbol{s}_T) = 1$.

B. Nonprehensile Object Pushing

In general, consider a single object currently at a state $\boldsymbol{s}_t \in SE(2)$ and given a desired state $\hat{\boldsymbol{s}}_{t+1} \in SE(2)$ for the next time step, the desired motion of the object can be

accordingly calculated by $\Delta \boldsymbol{s}_t = (\boldsymbol{s}_t)^{-1} \cdot \hat{\boldsymbol{s}}_{t+1} \in SE(2)$, as a relative transformation expressed in the object's body frame. With an off-the-shelf pushing strategy, the desired pushing action $\boldsymbol{u}_t$ should be generated based on the desired $\Delta \boldsymbol{s}_t$. We generally denote this process by a function $\boldsymbol{u}_t \leftarrow \text{PUSHSTRATEGY}(\boldsymbol{s}_t, \hat{\boldsymbol{s}}_{t+1})$. As shown in Fig. 3 (left), the robot's pushing action can be parameterized by two angles defined in the object's body frame, $\boldsymbol{u}_t = (\alpha_t, \beta_t)$. The first angle α_t is used to determine the starting position of the robot pusher relative to the object; the second angle β_t is the pushing direction. To perform a single pushing action $\boldsymbol{u}_t$, the robot needs to first place its pusher (or end-effector) to the starting position P_t and then move its pusher in the pushing direction to travel a distance d_{push} parallel to the workspace. Note that $\boldsymbol{u}_t$ differs from the lower-level robot action $\boldsymbol{a}_t$ (e.g., joint velocity) defined in Sec. III-A; A robot controller (e.g., a Cartesian position controller) will convert $\boldsymbol{u}_t$ to $\boldsymbol{a}_t$, which can be directly commended to the robot for execution.

Iteratively calling the pushing strategy and executing the generated actions in a closed loop allows the robot to push the object to follow a certain desired trajectory. As shown in Fig. 3 (right), the object's pose is tracked in real-time by sensors such as cameras. Given a desired trajectory of the object, represented by an ordered sequence of desired waypoints, the robot is able to continuously push the object to follow the trajectory through consecutively switching its pusher to the positions $P_0, \cdots, P_3$ and executing the generated actions $\boldsymbol{u}_0, \cdots, \boldsymbol{u}_3$ in between. In this work, we select to use the UNO Push framework developed in our previous work [83] as the pushing strategy.

IV. PROBLEM STATEMENT

Inspired by human strategy in rearranging objects, we propose a novel planning paradigm by an *object-centric* formulation in Sec. IV-A, to solve large-scale nonprehensile rearrangement problems. Furthermore, to handle real-world uncertainties, we close the planning loop by interleaving it with real execution as detailed in Sec. IV-B.

A. Object-Centric Planning

Traditional sampling-based approaches are mostly *robot-centric*, meaning that the robot actions are sampled only with robot-relevant constraints, and without an explicit purpose of interacting with certain specific objects [12], [13]. The motions of the surrounding objects, due to the execution of the sampled robot actions, are passively predicted by an approximated model of the system dynamics Γ. As such, the generated robot actions are not sufficiently guided by how the objects need to be reconfigured. Therefore, without actively exploiting task-relevant information (e.g., the desired object reconfigurations), the *robot-centric* strategies are not efficient enough.

In this work, we approach the nonprehensile rearrangement problem by an *object-centric* formulation. In short, we first search to find how the objects need to be moved to accomplish the rearrangement task, by planning the desired trajectory for each object without considering the robot. Then, with an available strategy for nonprehensile object pushing (e.g., the

UNO Push framework [83]), the robot actions can be generated and executed in a closed-loop manner, to move each object to follow their desired trajectories which are planned beforehand.

For planning the objects' trajectories, we assume the objects can move by themselves on a flat surface, without the need of being moved by a robot. We virtually attach an actuator to each object so that it can actively translate and rotate in all directions. For the activated object, the trajectory is represented by an ordered sequence of multiple waypoints in $SE(2)$, denoted by $\mathcal{T}^i = \{\hat{\boldsymbol{s}}_1^i, \cdots, \hat{\boldsymbol{s}}_k^i, \cdots, \hat{\boldsymbol{s}}_K^i\}$. Each waypoint $\hat{\boldsymbol{s}}_k^i = \left(\hat{\boldsymbol{p}}_k^i, \hat{\theta}_k^i\right) = \left(\hat{x}_k^i, \hat{y}_k^i, \hat{\theta}_k^i\right) \in SE(2)$ is a desired intermediate configuration of the object along the trajectory. If with perfect control and execution, the object is expected to reach each waypoint one by one from $\hat{\boldsymbol{s}}_1^i$ to $\hat{\boldsymbol{s}}_K^i$.

We assume only one object can be activated each time to actively move. The activated object does not have to be collision-free while following the desired trajectory since collisions between objects are allowed as aforementioned. While the activated object moves to follow its desired trajectory, other objects are passively moved due to object-object interactions. We represent such dynamics involving only mutual interactions between the objects by an object-centric transition function $\Pi : \mathcal{Q}^O \times \{1, \cdots, N\} \times \Xi \mapsto \mathcal{Q}^O$, where Ξ is the space of all possible object trajectories:

$$\boldsymbol{s}_{t+1} = \Pi\left(\boldsymbol{s}_t, i, \mathcal{T}^i\right) \qquad (2)$$

which infers the outcome arrangement $\boldsymbol{s}_{t+1} \in \mathcal{Q}^O$, given the current arrangement $\boldsymbol{s}_t \in \mathcal{Q}^O$, the index i, and the trajectory $\mathcal{T}^i$ of the activated object. We model Π by a physics engine, more details of which will be given in Sec. VII. It is worth noting that, different from the robot-centric transition function defined in Eq. (1), this object-centric transition function does not model any kind of interactions involving the robot. This modeling simplification of Π introduces more inaccuracy compared to real-world physics. However, by interleaving planning and execution (as described in Sec. IV-B), the errors due to such modeling inaccuracy can be adequately mitigated.

B. Interleaved Planning and Execution

For long-horizon and physics-involved manipulation tasks (such as rearranging a large number of objects), the execution will accumulate errors that deviate the real-world system transitions from what is predicted by the planner, due to inaccurate physics modeling and imperfect perception. As a result, the robot will likely fail the task even if a solution has been found. To this end, instead of one-time planning and executing the entire planned motion sequence, we interleave planning and real execution to progressively drive the system towards the task goal. As such, the planning horizon before each execution has to be limited, and a heuristic function to quantitatively evaluate and monitor the planning progress is needed. We denote the task-dependent heuristic function by $h : \mathcal{Q}^O \mapsto \mathbb{R}$, which evaluates a cost given an arrangement $\boldsymbol{s}_t \in \mathcal{Q}^O$. A smaller value of $h(\boldsymbol{s}_t)$ means the system is getting closer to achieving the task goal. As will be shown later in Sec VII, some simple-formed yet effective distance-based functions can be easily used as heuristics for various practical rearrangement problems.

Algorithm 1 Object-Centric Rearrangement Planning

Input: Initial arrangement $\boldsymbol{s}_0$, goal criterion $g(\cdot)$
Output: Task completion ($true$ or $false$)
1: $\boldsymbol{s}^* \leftarrow \boldsymbol{s}_0$
2: **while** TIME.AVAILABLE() **do**
3:      $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D \leftarrow$ OCP($\boldsymbol{s}^*$)   ▷ Sec. V
4:      **for** $d = 1, \cdots, D$ **do**
5:          EXECUTE($\boldsymbol{s}^*, i_d, \mathcal{T}^{i_d}$)   ▷ Sec. VI
6:          $\boldsymbol{s}^* \leftarrow$ OBSERVEOBJECTS()   ▷ via Real-time Sensing
7:      **end for**
8:      **if** $g(\boldsymbol{s}^*) == 1$ **then**   ▷ Task Accomplished
9:          **return** $true$
10:     **end if**
11: **end while**
12: **return** $false$

For each planning cycle, we adopt a greedy objective by minimizing the heuristic cost h while respecting the system physics and constraints. Mathematically, this can be represented as a constrained optimization formalization in Eq. (3).

$$\underset{\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D}{\text{minimize}} \quad h(\boldsymbol{s}_{t+D}) \qquad (3a)$$

$$\text{subject to} \quad 0 \leq D \leq D_{\max}, \qquad (3b)$$

$$\forall d = 1, \cdots, D: \qquad (3c)$$

$$\boldsymbol{s}_{t+d} = \Pi\left(\boldsymbol{s}_{t+d-1}, i_d, \mathcal{T}^{i_d}\right), \qquad (3d)$$

$$\boldsymbol{p}_{t+d}^i \in \mathcal{W}, \quad \forall i \in \{1, \cdots, N\} \qquad (3e)$$

Specifically, given the current arrangement $\boldsymbol{s}_t$, the optimization problem aims to find an ordered sequence of D object-trajectory pairs, i.e., $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$, so that the heuristic cost $h(\boldsymbol{s}_{t+D})$ will be minimized after these D trajectories have been performed. A pair in this sequence, $(i_d, \mathcal{T}^{i_d})$, consists of the index of the activated object $i_d \in \{1, \cdots, N\}$ and a desired trajectory $\mathcal{T}^{i_d}$ planned for this object. The length of the sequence, denoted by an integer variable D, is the horizon of the planning cycle, which is equal to the number of decisions made by the robot to switch to manipulating a different target object. We limit the value of D not to exceed a threshold $D_{\max}$ in Eq. (3b), to ensure the planning horizon is not too long; otherwise, the real execution can accumulate large errors and greatly deviate the system from its planned manipulation results, causing the real-world rearrangement inefficient or even to fail. Eq. (3d) constrains the system to transit under Π to comply with the physics laws, and Eq. (3e) confines each object to be within the robot's workspace.

The pipeline of nonprehensile rearrangement planning with the novel *object-centric* formulation is detailed in Alg. 1. Our framework starts with all objects being at states in $\boldsymbol{s}_0 \in \mathcal{Q}^O$ (i.e., the initial arrangement). Without considering the robot, the framework will first search for an ordered sequence of D object-trajectory pairs $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$ via our object-centric planner (OCP) by minimizing the heuristic cost h, as will be detailed in Sec. V. Then the robot will execute by calling the pushing strategy to make each activated object follow their desired trajectory one by one in the order of $d = 1, \cdots, D$, while strictly complying with the robot's constraints, as will be described in Sec. VI. After each execution, the real-world
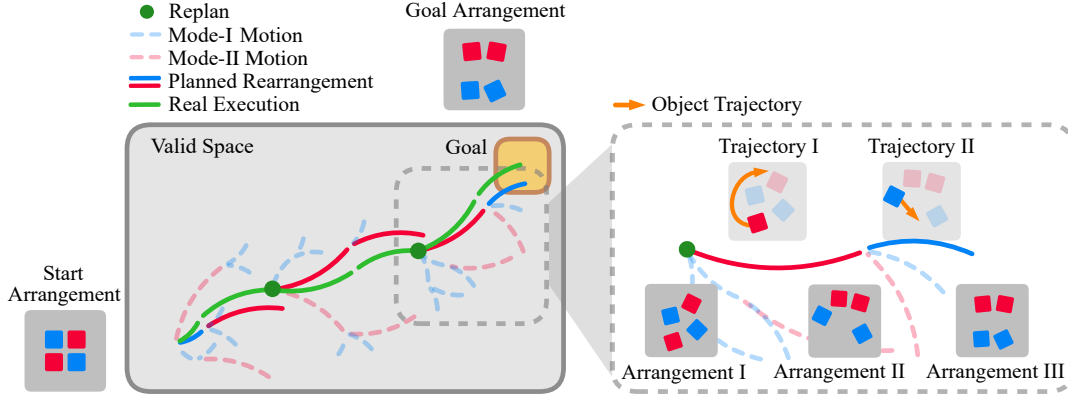
Fig. 4: A schematic plot of OCP. In the left figure, a motion tree is progressively grown from the start arrangement (lower left) towards the goal arrangement (upper right). Each edge of the tree is an explored object motion (i.e., trajectory) generated by one of two exploration modes, *Mode I* (blue line) and *Mode II* (red line). Each edge leads to an outcome arrangement represented by a tree node. Replanning (green dots) by sensing real-world arrangement is needed to eliminate the errors between the planned rearrangement solution (solid blue and red lines) and real execution (green lines) due to real-world uncertainties. The right figure shows the planned rearrangement since the last replanning, which consists of two consecutive object motions. The first (solid red line) is an *Mode II* motion that moves the red cube through a curvy Trajectory I (orange), resulting in an arrangement shown by Arrangement II; the second motion (solid blue line) is under *Mode I*, and leads to the arrangement shown in Arrangement III by moving the blue cube through a straight-line Trajectory II (orange).

arrangement s^* (i.e., the state of all objects) will be observed by sensors for starting the subsequent planning. The robot will repeat this procedure of interleaving planning and execution until it accomplishes the rearrangement task or exceeds the time budget. As such, by sensing the real-world state and adaptively adjusting its actions by replanning, the robot behaves reactively to the discrepancies between the planned motions and real execution and continuously eliminates accumulated errors in the recent execution.

V. OBJECT-CENTRIC SAMPLING-BASED PLANNER

As formulated in Sec. IV, we need to plan the desired object trajectories without considering the robot, by minimizing the heuristic cost as presented in Eq. (3). In practice, finding the optimal solution is intractable and usually not needed. As long as the heuristics cost h is progressively minimized, the rearrangement task can be solved efficiently. We propose to use a sampling-based approach to search the desired object trajectories as a solution to Eq. (3).

Specifically, we develop an object-centric sampling-based planner (OCP) by maintaining a tree-based data structure denoted by Tr. Each node of the tree, denoted by $n \in Tr.\text{NODES}()$, represents an explored arrangement. The arrangement associated with a node can be accessed through $n.s \in \mathcal{Q}^O$. Each edge in the tree directs from one node to its child node, representing an object-trajectory pair $(i_d, \mathcal{T}^{i_d})$ that transits the system (i.e., all objects) to the arrangement associated with the child node. The algorithmic steps of OCP are outlined in Alg. 2, and a schematic plot is shown in Fig. 4.

The tree is initialized with a root node n_{root} corresponding to the current start arrangement $s \in \mathcal{Q}^O$ observed by sensors. We limit the tree size (i.e., the number of tree nodes) by $S_{\max}$. While the tree size has not reached the limit $S_{\max}$, OCP keeps exploring through the following major steps: First, a node n will be randomly sampled to grow the tree for exploration, as introduced in Sec. V-A; Since only one object is activated

---

Algorithm 2 Object-Centric Planner – OCP($\cdot$)

Input: The current arrangement $s^* \in \mathcal{Q}^O$ observed by the sensors
Output: The sequence of the desired object activations and the corresponding trajectories $\{(i_1, \mathcal{T}^{i_1}), \cdots, (i_D, \mathcal{T}^{i_D})\}$
1: $Tr \leftarrow \{n_{root}.s = s^*\}$ ▷ Add Root Node to Tree
2: **while** $T.\text{GETSIZE}() < S_{\max}$ **do**
3:     $n \leftarrow \text{SAMPLENODE}(Tr)$ ▷ Sec. V-A
4:     $i \leftarrow \text{ACTIVATEOBJECT}(n)$ ▷ Sec. V-B
5:     $s_o, \mathcal{T}^i \leftarrow \text{EXPANDTREE}(n, i)$ ▷ Alg. 3
6:     **if** $\forall i, s_o^i \in \mathcal{W}$ and $\mathcal{T}^i \neq null$ **then**
7:         $n_{new} \leftarrow \text{CREATENODE}()$
8:         $Tr.\text{ADDNODE}(n_{new}.s = s_o)$ ▷ Add as a New Node
9:         $Tr.\text{ADDEDGE}((n, n_{new}), (i, \mathcal{T}^i))$
10:     **end if**
11:     **if** $g(s_o) == 1$ **then** ▷ Goal Reached
12:         $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D \leftarrow Tr.\text{BACKTRACE}(n_{new})$
13:         **return** $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$
14:     **end if**
15: **end while**
16: $n^* \leftarrow \arg\min_{n \in Tr.\text{NODES}()} h(n.s)$
17: $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D \leftarrow Tr.\text{BACKTRACE}(n^*)$
18: **return** $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$

---

each time to actively move, one object $i \in \{1, \cdots, N\}$ will be selected through sampling in Sec. V-B; Next, from the sampled node, a trajectory $\mathcal{T}^i$ of the activated object will be simulated with two possible exploration modes in Sec. V-C. As will be detailed in Sec. V-D, one of the exploration modes empowered by a soft-A^* algorithm, is an important design and crucial to the effectiveness of the planner. The outcome arrangement $s_o \in \mathcal{Q}^O$ after simulating the trajectory, if valid (i.e., all objects are within the workspace $\mathcal{W}$), will be added as a new node into the tree. If the arrangement s_o of the newly added node satisfies the goal criterion, the tree expansion will be stopped and we will backtrace this new node to extract the solution; otherwise, this tree expansion procedure will be repeated until the tree size reaches $S_{\max}$. Finally, we will

backtrace from the tree node that has the smallest heuristic cost, denoted by n^*, to extract the planned object trajectories $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^{D}$ as the solution to Eq. (3).

A. Node Sampling for Tree Expansion

For each node n of the tree, we use $n.D$ to denote the depth of the node n in the tree, and $n.N_c$ to denote the number of the child nodes of the node n. Similar to Expansive Space Tree (EST) [84], we associate a weight $w(n)$ for each node in Eq. (4). The probability of sampling a node is equal to its normalized weight, i.e., $P(n) = w(n) / \sum_{n' \in Tr.\text{NODES}()} w(n')$.

$$w(n) = \begin{cases} \frac{1}{n.N_c + 1} & n.D < D_{\max}, \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

The weight of a node is set to zero if its depth reaches $D_{\max}$, to eliminate this node from being sampled for expansion. This mechanism limits the tree depth by $D_{\max}$ to be consistent with the limited planning horizon in Eq. (3). Otherwise, the weight is set to be inverse to $n.N_c$. A large $n.N_c$ indicates that this node has been explored sufficiently with many child nodes, therefore, the weight of this node will be set low so other nodes will have a higher probability of being sampled for exploration. For nodes that have not been explored so far, they will have zero child nodes. We add 1 to $n.N_c$ in the denominator to avoid the issue of dividing by zero. The weights of nodes will be dynamically updated whenever a new node is added into the tree.

B. Heuristics-Guided Object Activation

To expand from the sampled node n in Sec. V-A, the tree will simulate an object trajectory and predict the outcome arrangement s_o based on the object-centric transition function Π. As aforementioned, only one object can be activated at a time, therefore, an effective sampling policy is needed to select which object to activate.

Given the arrangement s associated with the sampled tree node n (i.e., s is used to denote $n.s$ for simplicity), we guide the selection of the activated object by the informative gradients of the heuristic function. Intuitively, if the gradient magnitude of the heuristic function with respect to the state of the i-th object is large, i.e., a large $|\nabla_{s^i} h(s)|$, a high probability of sampling this object will be expected since the local change of this object's state can greatly affect the task progress evaluated by the heuristic cost. In other words, exploiting the motion of objects with high gradient magnitudes is likely to gain fast task progress. To this end, we integrate the gradients of the heuristic function into a weighted mixture of N Gaussian distributions to model the sampling probabilities for object activation. Specifically, each Gaussian component corresponds to an object i and is spatially centered at the position of this object. The gradient for the i-th object, $|\nabla_{s^i} h(s)|$, is used to model the weight of the i-th Gaussian.
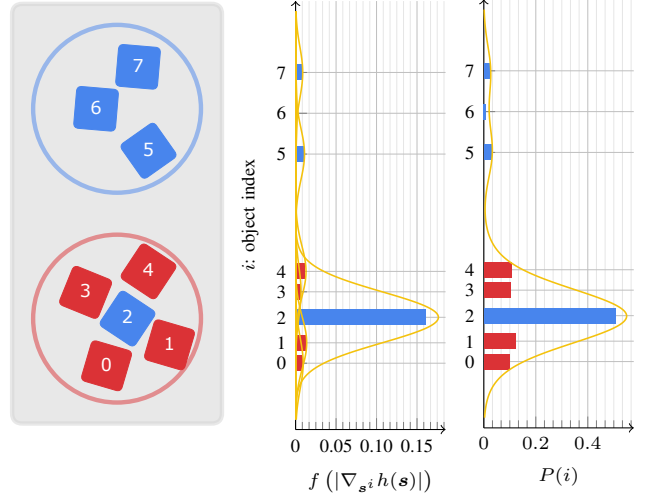

Fig. 5: *Left:* A scenario of the object sorting task. Two classes of cubes (red and blue) need to be relocated inside their corresponding goal regions (circles in the same color as the cubes). At the current state, all cubes except for Cube #2 are already sorted. *Middle:* The value of $f(|\nabla_{s^j} h(s)|)$ for each object, if directly used as the sampling probability, will cause the algorithm trapped by keeping sampling Cube #2 to activate. Since there is no free space around the unsorted Cube #2 for the robot to approach it, the algorithm will be likely stuck at this point. *Right:* Enabled by the weighted mixture of Gaussians, the sampling probabilities of the red cubes (#0, #1, #3, and #4) surrounding Cube #2 are increased so that they can be moved to create some free space for Cube #2, to facilitate the relocation of Cube #2 to its goal region by the subsequent actions.

The probability $P(i)$ for sampling the i-th object to activate is given in Eq. (5a).

$$P(i) = \frac{1}{Z} \sum_{j=1}^{N} f(|\nabla_{s^j} h(s)|) \cdot \varphi(d_{ij}) \quad (5a)$$

$$\propto f(|\nabla_{s^i} h(s)|) + \sum_{j \neq i} f(|\nabla_{s^j} h(s)|) \cdot e^{-\frac{d_{ij}^2}{2\sigma^2}} \quad (5b)$$

where Z is the normalization term to ensure the probabilities for all i sum to 1; $\varphi(\cdot)$ is the density function of a zero-mean Gaussian with variance σ^2; $d_{ij}^2 = \|p^i - p^j\|^2$ is the squared distance between the object i and j; $f : \mathbb{R}^+ \mapsto \mathbb{R}^+$ is a stretching function to exaggerate the magnitude difference between Gaussian components. In practice, we opt for a simple power function to stretch, i.e., $f(x) = x^k \ (k > 1)$.

We use Gaussians to model the distribution of Eq. (5) continuously, in order to prevent the sampling from being trapped by a local minimum arrangement. For example, as showcased in Fig. 5 (left), the blue cube (#2) is still far away from its desired goal region (the blue circle on the top) and has a high gradient magnitude, whereas the surrounding red cubes are already placed at their desired locations and thus have low gradient magnitudes. In the current situation, activating the blue Cube #2 is ineffective, since there is no free space around it to realize a trajectory achievable by the actual robot. In this case, directly using the gradient magnitudes $|\nabla_{s^i} h(s)|$ as the sampling probability is likely to cause the algorithm to get stuck always sampling the ineffective Cube #2. However, smoothened by using Gaussians, the probability of sampling an object depends on not only its own gradient but also the
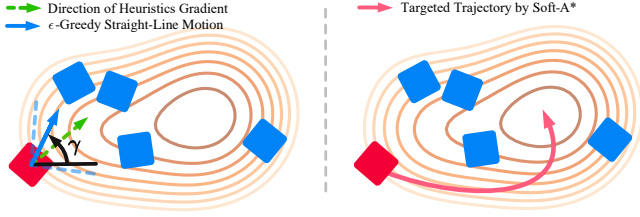
Fig. 6: Two exploration modes of generating the object trajectory for expanding the motion tree. The contour lines represent the heuristic cost concerning only the activated object (red cube). *Left:* With a probability of $1-\epsilon$, *Mode I* samples a straight-line motion (blue solid arrow) with an angle γ to the x-axis of the workspace (black line). The sampling of γ is centered around the direction of the heuristic gradient (green dashed arrow) with a range of $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right)$ (between the blue dashed lines); *Right: Mode II* plans a goal-oriented curvy trajectory (orange curve) for the activated object towards its optimal location (i.e., minimizing the heuristic cost), by using Soft-A^* as will be introduced in Sec. V-D.

gradients of other objects near it, as can be seen in Eq. (5b). In the above example, using Gaussians can make it possible to activate the surrounding red cubes with high probabilities, thus potentially creating some free space around the blue cube for it to be feasibly moved out and toward its desired location.

C. Tree Expansion by Analyzing Object Trajectories

After an activated object i is selected in Sec. V-B, our planner needs to move this object along a certain trajectory to explore the outcome arrangement. A straightforward exploration strategy is to move the object locally in a random direction, i.e., a straight-line trajectory. Such local and random motions of the activated object enable efficient exploration for arrangement by concurrently reconfiguring multiple surrounding objects via object-object interactions. However, since such explorations are local and the interaction outcomes can be random, it is difficult to find motions that are guaranteed to improve the task progress significantly.

Therefore, we also incorporate a more goal-oriented exploration strategy. Instead of exploring the local reconfiguration of multiple surrounding objects, the new strategy focuses on the repositioning of the activated object: Without excessively disturbing other objects, it directly moves the activated object to its desired destination, along an optimal trajectory planned with a certain level of discretization. Ideally, if each object can be directly moved to its desired location one by one, the rearrangement task can be solved progressively without needing much unnecessary exploration. Based on this intuition, the second strategy is designed to enable the possibility of finding effective motions that can directly finish the rearrangement of one object (i.e., the activated object). Fig. 6 illustrates the two exploration strategies, and the details are given below:

1) *Mode I: ϵ-Greedy Straight-Line Motion.* As illustrated in Fig. 6 (left), this mode constructs a straight-line trajectory to guide the motion of the activated object, where the trajectory has only one waypoint $\hat{s}_1^i \in SE(2)$ as the destination. Rather than purely random, we adopt a ϵ-greedy policy to determine the direction of the straight line (i.e., an angle γ) for better exploration efficiency. With a probability ϵ, γ will be

---

Algorithm 3 ExpandTree($\cdot$)

Input: A sampled tree node n, the index of the activated object i
Output: The outcome arrangement $s_o \in \mathcal{Q}^O$, the trajectory $\mathcal{T}^i$ of the activated object generated for exploration
1: $s \leftarrow n.s$ ▷ Arrangement of the Node n
2: $s^i = (x^i, y^i, \theta^i) \leftarrow$ GETOBJECTSTATE(s, i)
3: **if** UNIFORM$(0,1) > p_{A^*}$ or $i \in n.\mathcal{B}$ **then** ▷ *Mode I*
4: $\quad \mathcal{T}^i \leftarrow$ A Straight Line with Angle γ
5: **else** ▷ *Mode II*
6: $\quad \mathcal{T}^i \leftarrow$ SOFT-$A^*(s, i)$ ▷ Sec. V-D
7: $\quad n.\mathcal{B} \leftarrow n.\mathcal{B} \cup \{i\}$
8: **end if**
9: $\hat{s}_1^i \leftarrow$ First Waypoint of $\mathcal{T}^i$
10: $u_0 \leftarrow$ PUSHSTRATEGY$(s^i, \hat{s}_1^i)$ ▷ Sec. III-B
11: **if** not FEASIBLE(u_0) **then**
12: $\quad$ **return** $null$
13: **end if**
14: $s_o \leftarrow \Pi(s, i, \mathcal{T}^i)$ ▷ Simulate via Π
15: **return** $s_o, \mathcal{T}^i$

---

uniformly sampled between $[-\pi, \pi)$; with a probability $1-\epsilon$, γ will be greedily sampled around the gradient direction of the heuristic function, i.e., $\left(\frac{\partial h(s)}{\partial x^i}, \frac{\partial h(s)}{\partial y^i}\right)$, with an allowed angular deviation between $[-\frac{\pi}{4}, \frac{\pi}{4})$. The length of the straight line is uniformly sampled from a preset range $[l_{\min}, l_{\max})$, which are hyperparameters chosen according to the size of the workspace and the scale of the problem. While the activated object translates along with the generated straight line, it also rotates with a constant angular velocity to change its orientation by a randomly sampled $\Delta\theta^i \in [-\pi, \pi)$.

2) *Mode II: Goal-Oriented Trajectory Planned by Soft-A^*.* Under this mode, the trajectory $\mathcal{T}^i$ is deterministically planned to reach the optimal position of the activated object i while being aware of potential collisions with other objects, as illustrated in Fig. 6 (right). The planning of the trajectory $\mathcal{T}^i$ is solved by a softened version of the A^* algorithm, as will be detailed in Sec. V-D. Since *Mode II* is deterministic, for the same tree node n and activated object i, the planned $\mathcal{T}^i$ is always the same. To this end, for each tree node n, we store the indices of the objects that have been explored under *Mode II* in a set $n.\mathcal{B} \in \{1, \cdots, N\}$. If a new *Mode II* exploration is going to be performed on an object i that is already explored under *Mode II* (i.e., $i \in n.\mathcal{B}$), we will skip it to save the computational time cost of a duplicate *Mode II* exploration.

The procedure of generating and simulating a trajectory $\mathcal{T}^i$ for expanding the tree, using one of the two exploration modes, is given in Alg. 3. Which exploration mode to use is randomly determined, and the probability of using *Mode II* is a hyperparameter p_{A^*}. After $\mathcal{T}^i$ is generated, we will then call the pushing strategy to compute the first action u_0 and investigate its feasibility lazily. If u_0 is not feasible (e.g., the starting position $P_0 \in \mathcal{W}$ for the pusher to perform u_0 is occluded by other objects), $\mathcal{T}^i$ cannot be achieved by the robot execution and will be set as $null$. Otherwise, $\mathcal{T}^i$ will be simulated by the object-centric physics Π to predict the outcome arrangement of objects s_o, which will be added as a new node to expand the tree.
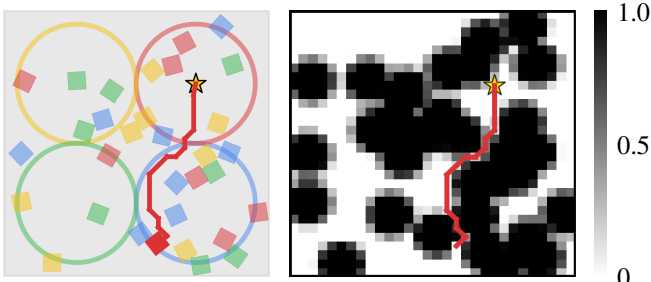
Fig. 7: The grid representation of the workspace (right) for the activated red object (highlighted) in the scene (left). The values of the grids, ranging from 0 to 1, evaluate the collisions between the activated object and other objects. The path (red lines) planned by the Soft-A^* algorithm on the grid map will move the activated object from its current position to the optimal position (the star mark in the left figure) at the center of its goal region (the red circle).

D. Goal-Oriented Trajectory Planned by Soft-A^*

As one of the core designs of our OCP planner, *Mode II* explores the object arrangement in a more goal-oriented manner compared to the random and local *Mode I*. The trajectory $\mathcal{T}^i$ generated by *Mode II* intends to move the activated object i to its optimal position that minimizes the heuristic cost $h(s)$, while trying to avoid collisions with other objects to prevent loss of rearrangement progress. An A^* algorithm can efficiently plan such a collision-free trajectory with the shortest path length. However, a collision-free trajectory may rarely exist, especially in a highly packed setup. To this end, we relax the collision-free requirements of the original A^* algorithm to propose the Soft-A^*, which allows the activated object i to have insignificant collisions with other objects. The Soft-A^* plans the trajectory $\mathcal{T}^i$ by minimizing a cost that takes both the path length and potential collisions into account.

As shown in Fig. 7, we discretize the workspace $\mathcal{W}$ into a finite grid map $\mathcal{I}^i \in \mathbb{R}^{W \times H}$ with a width W and height H. The size of the grid, denoted by Δ, is a hyperparameter used to specify the map's resolution. Each grid of the map, denoted by $g \in \mathcal{I}^i$, has a coordinate $g.\boldsymbol{p} = (g.x, g.y)$ defined by the position of the grid's center point. Each grid also has a value, denoted by $g.v \in [0,1]$, which measures the potential collisions between the activated object i and other objects. We implement the evaluation of grid value by a linear function in terms of the distance between objects, as expressed in Eq. (6a).

$$g.v = \begin{cases} 1 & d_g < C_{\min} \\ 0 & d_g > C_{\max} \\ (C_{\max} - d_g)/(C_{\max} - C_{\min}) & \text{otherwise} \end{cases} \quad (6a)$$

$$d_g = \min_{j \in \{1, \cdots, N\}, j \neq i} \|g.\boldsymbol{p} - \boldsymbol{p}^j\| \quad (6b)$$

where d_g in Eq. (6b) is the distance between a grid g and the nearest object (excluding the activated object i) to this grid. $C_{\min}$ and $C_{\max}$ are two parameters specified based on the size of the objects. When $d_g < C_{\min}$, we consider the collision for the activated object i (when at the position of the grid g) to be unacceptable, and thus set the grid value to 1; when $d_g > C_{\max}$, the activated object is guaranteed not to collide with other objects, and the grid value is set to

0; when d_g decreases from $C_{\max}$ to $C_{\min}$, the grid value will increase linearly from 0 to 1 indicating an increase in potential collisions between the activated object and other objects.

Then we run an A^* algorithm on the grid map $\mathcal{I}^i$ to find a path for the activated object, represented by the sequence of grids $\{g_1, g_2, \cdots, g_*\}$ it traverses. The path starts from the current position and ends up at the optimal grid g_* of the activated object. The optimal grid g_* is the grid that minimizes the heuristic cost $h(\cdot)$ when placing the activated object at this grid. For many simple-formed heuristic functions, analytically finding the optimal g_* is easy; otherwise, we can brute-force search over the entire grid map to find g_*. In practice, we find a grid size Δ similar to object size works well enough. Thus, the number of grids is usually small, making the brute-force search computationally cheap.

An A^* algorithm conventionally needs two functions: 1) a cost function $c_{A^*}(\cdot, \cdot)$ for two adjacent grids, and 2) a heuristic function $h_{A^*}(\cdot)$ that underestimates the cost from a grid to the goal g_*. Our Soft-A^* aims to trade-off between the path length and potential collisions along the path. As such, we propose to use a cost function composed of both terms. Given a grid g and one of its adjacent grid g', the cost is evaluated by Eq. (7):

$$c_{A^*}(g, g') = d(g, g') + \Delta \cdot g'.v$$
$$d(g, g') = \begin{cases} \|g.\boldsymbol{p} - g'.\boldsymbol{p}\| & g'.v < 1 \\ \infty & \text{otherwise} \end{cases} \quad (7)$$

The first term $d(g, g')$ is the distance between the two grids g and g'; the second term $g'.v$ is a grid value that measures collision, which is scaled by the grid size Δ to have a comparable magnitude with the first term. Importantly, when g' has an unacceptable collision (i.e., $g'.v = 1$), the distance $d(g, g')$ is set to infinity to prevent the path from passing through g'. The accumulated $c_{A^*}(\cdot, \cdot)$ along the entire path, called the total cost, will be minimized by the Soft-A^*.

The heuristic function of our Soft-A^* is simply the straight-line distance from a grid g to the goal grid g_*, i.e., $h_{A^*}(g) = \|g.\boldsymbol{p} - g_*.\boldsymbol{p}\|$. Note that $h_{A^*}(g)$ strictly underestimates the cost from g to g_*, which ensures the optimality of the found path. If the total cost of the found path $\{g_1, g_2, \cdots, g_*\}$ is finite, we will generate a trajectory for the activated object by extracting the coordinates of each grid throughout the path and assuming the orientation of the object is not changed, i.e., $\mathcal{T}^i = \{(g_1.\boldsymbol{p}, \theta^i), (g_2.\boldsymbol{p}, \theta^i), \cdots, (g_*.\boldsymbol{p}, \theta^i)\}$. This $\mathcal{T}^i$ will be returned to expand the motion tree under *Mode II*. Otherwise, when the total cost of the found path is infinite, we will set $\mathcal{T}^i = null$ to abort it.

VI. ROBOT EXECUTION

After the OCP has planned object trajectories $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$ in Sec. V, each trajectory $\mathcal{T}^{i_d}$ needs to be realized by the real robot execution. For each planned trajectory $\mathcal{T}^{i_d} = \{\hat{s}_1^{i_d}, \cdots, \hat{s}_k^{i_d}, \cdots, \hat{s}_K^{i_d}\}$ of an activated object i_d, the robot will specify the activated object as the target object for pushing, and push it to reach each waypoint $\hat{s}_k^{i_d} \in SE(2)$ in the trajectory one by one via a closed-loop pushing strategy (e.g., the UNO Push framework [83]), as outlined in Alg. 4. Specifically, the robot needs to inspect the

first pushing action $\boldsymbol{u}_0$ generated by the pushing strategy. If the starting position $P_0 \in \mathcal{W}$ associated with $\boldsymbol{u}_0$ is occluded by objects, the robot will not be able to insert its pusher to this position for subsequent manipulation of the object i_d. In this case, the robot will stop the current execution and skip to manipulate the next object. Otherwise, the robot will move its pusher to P_0 from above the workspace; and then by reaching each waypoint $\hat{\boldsymbol{s}}_k^i$, $k = 1, \cdots, K$ of the planned trajectory $\mathcal{T}^{i_d}$, the robot can manipulate the object i_d through planar pushing (i.e., the pusher always moves in parallel to the workspace plane) to follow $\mathcal{T}^{i_d}$. For implementing the robot execution, we control the motion of the pusher by commanding Cartesian velocities $\boldsymbol{v} \in se(3)$. The Cartesian $\boldsymbol{v}$ is then projected into the robot's configuration space to generate a robot control $\boldsymbol{a} \in \mathcal{A}$ for commanding the robot. In this work, we use the robot's joint velocity as control, i.e., $\boldsymbol{a} = \dot{\boldsymbol{q}} \in \mathbb{R}^M$, which is generated via null-space projection of the robot Jacobian, as detailed in Eq. (8):

$$\boldsymbol{a} = \dot{\boldsymbol{q}} = J^\dagger \cdot \boldsymbol{v} + \lambda \cdot \left(\mathbb{I} - J^\dagger J\right) \dot{\boldsymbol{q}}_{null} \tag{8}$$

where $\dot{\boldsymbol{q}}_{null} \in \mathbb{R}^M$ is the joint velocity for null-space motion, to improve the motion quality of the robot (e.g., moving away from singularity and joint limits); we compute $\dot{\boldsymbol{q}}_{null}$ by taking the gradient of certain quality measures (e.g., the manipulability and a distance-based cost related to joint limits) [85]; $\dot{\boldsymbol{q}}_{null}$ is projected via $\left(\mathbb{I} - J^\dagger J\right)$ to the Jacobian's null space for not affecting the desired Cartesian behavior of the robot.

Throughout the execution, the robot constraints must be satisfied to ensure that future execution is still feasible. With the same frequency as the low-level controller, the robot will monitor the following three events by real-time sensor readings: 1) self-collision; 2) joint limit violation; and 3) singularity measured by the volume-based manipulability $\sqrt{\det\left(J J^\top\right)}$. If any event is about to occur, we will stop the robot execution, safely move the robot back to the previous configuration, and skip to manipulate the next target object. This strategy ensures the robot complies with its constraints in a lazy manner. Furthermore, to make sure no objects will be pushed outside the workspace $\mathcal{W}$ due to the inaccuracy of the modeled physics and real-world uncertainties, we also monitor the distances between each object and the workspace boundary. When an object gets too close to the workspace boundary, we will use the same pushing strategy to push that object back toward the center of the workspace.

VII. EXPERIMENTS

Extensive experiments were conducted both in simulation and on a physical robot manipulator to evaluate the performance of our proposed object-centric planner OCP by comparative study with selected baselines. We are concerned with two major aspects of the performance: 1) Planning efficiency, which can be reflected by a low average planning time for completing rearrangement tasks; and 2) Effectiveness of the generated actions, which was evaluated by the average number of actions and average execution time (for real-world experiments) needed for task completion. Our planner was

---

Algorithm 4 Execute($\cdot$)

Input: The current arrangement $\boldsymbol{s} \in \mathcal{Q}^O$, the index i and the planned trajectory $\mathcal{T}^i = \{\hat{\boldsymbol{s}}_k^i\}_{k=1}^K$ of the activated object
1:   $\boldsymbol{u}_0 = (\alpha_0, \beta_0) \leftarrow \text{PUSHSTRATEGY}(\boldsymbol{s}^i, \hat{\boldsymbol{s}}_1^i)$ ▷ Sec. III-B
2:   $P_0 \leftarrow \text{STARTINGPOSITION}(\boldsymbol{s}^i, \boldsymbol{u}_0)$
3:   **if** $\text{OCCLUDED}(P_0)$ **then**
4:      **return**
5:   **end if**
6:   $\text{MOVEPUSHERTO}(P_0)$ ▷ Above the Workspace
7:   **for** $k = 1, \cdots, K$ **do**
8:      **while** $\boldsymbol{s}^i$ not reaching $\hat{\boldsymbol{s}}_k^i$ **do**
9:         $\boldsymbol{u} \leftarrow \text{PUSHSTRATEGY}(\boldsymbol{s}^i, \hat{\boldsymbol{s}}_k^i)$ ▷ Sec. III-B
10:       $\text{PUSH}(\boldsymbol{u}, d_{push})$ ▷ Planar Push by a Distance d_{push}
11:       $\boldsymbol{s} \leftarrow \text{OBSERVEOBJECTS}()$ ▷ via Real-time Sensing
12:      **end while**
13: **end for**

---

implemented with the Box2D physics engine [1] to approximate the object-centric transition function Π defined in Sec. IV. All the evaluations were run in Python with a single thread on a 3.4 GHz AMD Ryzen 9 5950X CPU. For simulation evaluation, in the MuJoCo [86] environment, we used a floating gripper to execute the generated actions by the planner, as shown in Fig. 8. The floating gripper is allowed to freely navigate parallel to the workspace, to interact with the objects by an attached fence pusher. The gripper can also teleport to switch to manipulate a different object. For real-world experiments, we evaluated our planner with a 7-DoF Franka Emika Panda robot manipulator.

We selected the following state-of-the-art nonprehensile rearrangement planners as baselines for comparison:

1) MCTS [20]: A Monte Carlo Tree Search (MCTS)-based planner integrated with a learned rollout policy, which is a deep neural network trained offline with image data.
2) ILS [22]: An Iterative Local Search (ILS) algorithm that locally explores and optimizes robot actions that reduce the distance of objects to their goal regions.
3) kdRRF [26]: A forest-based kinodynamic planner that enables concurrent exploration from different subspaces of the problem to find more task-efficient motions. For a fair comparison, we replaced the MuJoCo-based physics model used in the original implementation of kdRRF with the Box2D physics engine.

A. Rearrangement Tasks

We chose the challenging rearrangement tasks used by two state-of-the-art works [20], [22] for evaluation, by considering the variety and large scale of the entailed tasks. For example, some tasks (e.g., singulation) require only local rearrangement of a target object, whereas others need global rearrangement to relatively reconfigure all objects. All evaluated tasks are visualized in Fig. 8. We categorize all the tasks into two kinds, based on how the task goals are differently defined:

1) Tasks without Explicit Goal Definition: Such tasks do not require an explicit definition of a goal pose or goal region for each object. As long as the relative reconfiguration of

---

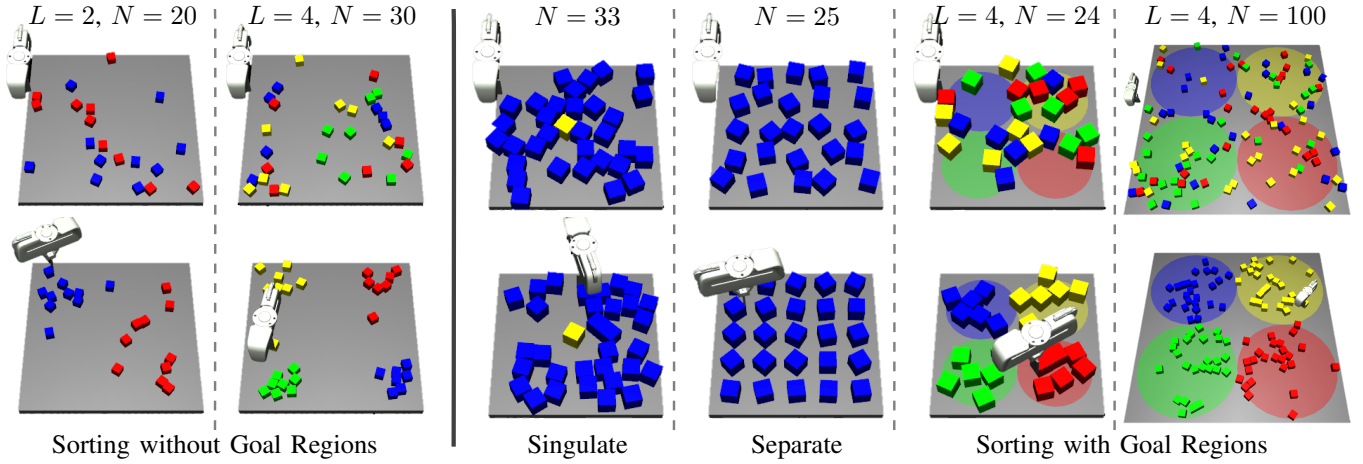[1] Box2D, A 2D Physics Engine for Games: https://box2d.org/

Fig. 8: The evaluated rearrangement tasks solved by our OCP planner in MuJoCo simulation with a floating gripper pusher. The task scenes have different numbers of objects (N) and object classes (L). The top figures show the initial configuration of randomly placed objects, and the bottom figures show the final configurations when the tasks are completed.

the object states meets certain requirements, the task will be considered successful. One example of such tasks is *Object Sorting without Goal Regions* investigated in [20]. As shown in Fig. 8 (left), this task asks the robot to separate objects of L different classes into clusters. No goal pose or goal region is specified for any object. By observing the object arrangement $s \in \mathcal{Q}^O$, a convex hull $\mathbf{CH}_j(s) \subset \mathbb{R}^2$ is constructed for each class $j \in \{1, \cdots, L\}$ of objects, which covers the geometric shapes of all objects belonging to this class j. The task is accomplished with success if the minimum distance between any pair of convex hulls is larger than a threshold ϵ_d. Formally, the goal criterion of this task is satisfied (i.e., $g(s) = 1$) when $\forall i, j \in \{1, \cdots, L\}$:

$$\min_{i \neq j} \text{dist}\left(\mathbf{CH}_i(s), \mathbf{CH}_j(s)\right) > \epsilon_d \qquad (9)$$

To solve this task with our object-centric planner, we equip our planner with a heuristic function $h(\cdot)$ similar to the reward function in [20]. Basically, when the objects in the same class get closer and the objects of different classes get further apart, a lower heuristic cost will be expected.

2) Tasks with Explicit Goal Definition: This kind of task requires an explicit goal region $\mathcal{G}_i \subset \mathcal{W}$ for each movable object $i \in \{1, \cdots, N\}$. The centroid position of an goal region $\mathcal{G}_i$ is denoted by $\boldsymbol{p}^{\mathcal{G}_i} = \left(x^{\mathcal{G}_i}, y^{\mathcal{G}_i}\right)$. In different tasks, the goal regions of some objects can be either distinct or overlapping. This kind of task requires each object to be relocated inside its corresponding goal region. In general, the goal criterion can be defined as:

$$\forall i \in \{1, \cdots, N\} : \boldsymbol{p}^i = \left(x^i, y^i\right) \in \mathcal{G}_i \qquad (10)$$

where $\boldsymbol{p}^i = \left(x^i, y^i\right)$ is the position of the i-th object. the heuristic function used by our object-centric planner can simply be squared distances between objects and their corresponding goal regions:

$$h(\boldsymbol{s}) = \sum_{i \in \{1, \cdots, N\}} \mathbb{1}\left\{\boldsymbol{p}^i \notin \mathcal{G}_i\right\} \frac{\|\boldsymbol{p}^i - \boldsymbol{p}^{\mathcal{G}_i}\|^2}{r_{\mathcal{G}_i}^2} \qquad (11)$$

where $r_{\mathcal{G}_i}$ is the size of the goal region $\mathcal{G}^i$ (e.g., a radius if $\mathcal{G}_i$ is a circle); $\mathbb{1}\left\{\boldsymbol{p}^i \notin \mathcal{G}_i\right\}$ is an indicator function that filters

out the objects that are already inside their goal regions. When our planner uses heuristic gradients to guide the exploration (in Sec. V-B), this indicator function helps our planner focus on exploring the unsolved objects (i.e., objects not at their goals) without wasting time on already solved objects.

The tasks used by [22] can be categorized into this kind:

a) *Singulate:* This task requires a target object $o \in \{1, \cdots, N\}$ to be singulated away from other objects $i \neq o$. Specifically, the target object needs to be relocated inside a small circular goal region $\mathcal{G}^o$ at the center of the workspace, while other objects near one of the four corners of the workspace.

b) *Separate:* The workspace is divided into multiple grids, and each grid contains a tiny circular goal region. Each object needs to be separated into one distinct goal region. The objects are not unique, and the assignment of the objects to their goal regions is dynamically determined.

c) *Sorting with Goal Regions:* There are four classes of objects, visually represented by different colors. The objects of the same class share the same goal region, which is a circle located near one corner of the workspace. The task requires the objects of different classes to be moved to their corresponding goal regions.

In certain tasks (e.g., Separate), some objects are not unique and their corresponding goal regions are interchangeable. In such cases, similar to [22], the assignment of objects to goal regions is dynamically determined. Specifically, suppose the set of non-unique objects is denoted by $\mathcal{D} \subset \{1, \cdots, N\}$, and the set of distinct goal regions for these objects is denoted by $\mathcal{SG} = \{\mathcal{G}^i : i \in \mathcal{D}\}$. Note that $|\mathcal{D}| = |\mathcal{SG}|$. An assignment function $A : \mathcal{D} \mapsto \mathcal{SG}$ bijectively assigns each object in $i \in \mathcal{D}$ to its corresponding goal region $A(i) \in \mathcal{SG}$. This assignment is dynamically determined by minimizing the summed distances between the objects and their assigned goal regions, i.e., $\sum_{i \in \mathcal{D}} \text{dist}\left(\boldsymbol{p}^i, A(i)\right)$.

B. Analysis of Parameter Selection

We first conducted experiments in simulation to analyze how the performance of our OCP planner is affected by
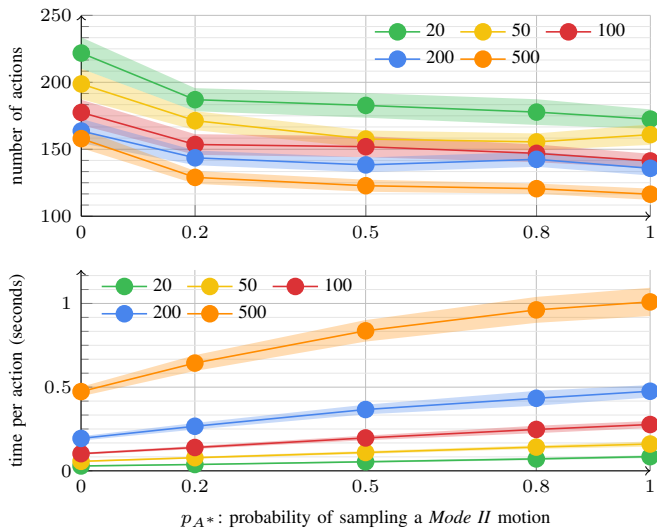
Fig. 9: The statistics of the number of actions (top plot) and the average planning time per action (bottom plot) in terms of different parameter choices of S_{max} (plots in different colors) and p_{A^*} (x-axis). The shaded areas are the quarter standard deviations for the top plot; and the standard deviations for the bottom plot.

different choices of parameters: a) the maximum tree size S_{max}, and b) the probability p_{A^*} of using *Mode II* (i.e., Soft-A^*) for exploration. We selected a representative task, *Sorting with Goal Regions*, to evaluate all parameter choices. The task is to sort 32 objects of 4 classes, where 8 objects belong to each class (similar to the setup shown in Fig. 8). The workspace was set to be a square of size $0.6m$, and the goal regions of the 4 classes were circles located at $(\pm 0.135, \pm 0.135)m$ with the same radius of $0.135m$. 100 trials were conducted for each parameter choice to collect the statistics about relevant metrics. All experiment trials were run with the same initial configuration of the objects.

The results are shown in Fig. 9 with different parameter choices. Specifically, as the tree size increases with $S_{max} = 20, 50, 100, 200$, and 500 (the plots in different colors), the planning time per action increases while the number of actions for task completion decreases. The results are intuitive: as S_{max} increases, the planner needs to explore more object motions to have more tree nodes for generating one robot action. Therefore, the planning time per action increases; at the same time, when S_{max} increases, the planner explores the problem space more extensively with more tree branches, which increases the probability of finding more optimal actions and results in a smaller number of actions needed for task completion. When we varied the probability $p_{A^*} = 0, 0.2, 0.5, 0.8$, and 1.0, the results show that a higher p_{A^*} causes more planning time required for each action. This is because a higher p_{A^*} causes more attempts of a *Mode II* exploration, which requires an extra computational time of A^* search. Furthermore, a higher p_{A^*} reduces the number of actions by having more *Mode II* motions to execute. This is because *Mode II* motions are generally more effective than *Mode I* motions in reducing the heuristic cost and advancing the task.

In general, the results have shown that a larger S_{max} and a higher p_{A^*} generate more effective actions, which may potentially reduce the overall execution time by lowering the number of actions needed for execution. In contrast, a smaller S_{max} and a lower p_{A^*} can facilitate more reactive planning with less planning time required for each action generation, but at the cost of the optimality of generated actions.

C. Comparative Evaluations in Simulation

First, on the *Sorting without Goal Regions* task, we compared our OCP planner against both MCTS and kdRRF. For this task, we did not compare against ILS since ILS requires explicit goal regions for the distance computation, making it incompatible with this task. We used the same setup as in [20]: All movable objects are cubes of size $2.5cm$; the workspace is a square region of size $50cm$; and we evaluated on 6 different scenes, combinations of $L = 2, 3, 4$ (the number of object classes) and $N = 20, 30$ (the total number of objects). The results for MCTS were directly obtained from [20]. The original work of MCTS used a specially designed three-finger robot pusher, which enables the pusher to simultaneously contact multiple objects and concurrently manipulate them by sweeping-like actions. For both kdRRF and our OCP planner, we ran experiments with 100 trials on each scene, where the objects were randomly placed at the beginning of each trial. As reported in [20], the average planning time per action for MCTS is $2.16s$ with a parallelized implementation with 8 threads on an Intel i7-7820X CPU, while our planner only required an average of $0.5s$ on a single thread for generating an object-centric action.

Furthermore, we report the success rate and average number of actions in Fig. 10. Note that the success rates of different methods were evaluated differently, since different methods use different stopping criteria. As elaborated in [20], MCTS fails when 1) no object has been touched by the pusher in subsequent 15 actions, or 2) the relative difference of observed reward is smaller than a threshold. Such stopping conditions are not suitable for kdRRF and our OCP planner, since 1) it is impossible for the pusher to not touch any object for a long time since the sampled actions are targeted around objects, and 2) no notion of reward is used in both planners. Therefore, for kdRRF and our OCP planner, we stop them by setting a planning time budget. If the solution plan is not found within 60 seconds (for scenes with $N = 20$) or 300 seconds (for scenes with $N = 30$) of planning, we regard the trial as a failure. As can be seen from the results in Fig. 10, our object-centric planner generally outperforms the robot-centric kdRRF in terms of both the success rate and the number of actions, since the object-centric paradigm enables more task-relevant motion exploration and thus generates more efficient actions. Furthermore, compared to MCTS, our planner was generally able to achieve a better success rate with fewer required actions while not needing extra time and a large amount of collected data for training. When the scale of the problem becomes larger (e.g., when $L = 4$ and $N = 30$), the performance improvement of our planner becomes less significant, because it is harder for our planner to find more effective actions within limited search horizons than the learned policy used by MCTS.

Next, on the other three tasks, *Singulate*, *Separate*, and *Sorting with Goal Regions* (including two different scenes of sorting $N = 24$ and 100 objects in 4 classes), we compared

Scene # classes	# objects	MCTS [20]	kdRRF [26]	OCP (Ours)
L = 2	N = 20	**100 %** 36.1 ± 1.3	**100 %** 31.1 ± 9.1	**100 %** **28.7 ± 9.1**
	N = 30	96 % 77.5 ± 3.6	**100 %** **58.7 ± 20.4**	98 % 69.0 ± 24.2
L = 3	N = 20	98 % 66.6 ± 2.3	99 % 60.7 ± 16.3	**100 %** **41.4 ± 13.6**
	N = 30	91 % 131.5 ± 5.1	**100 %** 128.0 ± 53.9	**100 %** **99.7 ± 33.6**
L = 4	N = 20	97 % 80.1 ± 2.3	30 % 77.0 ± 12.5	**100 %** **63.9 ± 24.0**
	N = 30	89 % **162.6 ± 4.6**	28 % 253.9 ± 72.6	**98 %** 167.8 ± 51.5

Fig. 10: The success rate and the average number of actions of different methods evaluated on the *Sorting without Goal Regions* task, with different numbers of classes and objects.

our OCP planner against ILS and kdRRF. For each evaluated task and scene, we used the same task setup and time budget as in [22] for a fair comparison: The workspace was restricted to a square region of size $40cm$ (Note: the workspace size was increased to $125cm$ for sorting $N = 100$ objects), and all objects are cubes of size $4cm$; the time budget was set to $20s$ for *Singlute* and *Separate* tasks, and $30s$ and $600s$ for *Sorting with Goal Regions* when $N = 24$ and $N = 100$ respectively. Also, the sizes of goal regions were also kept the same as in [22]. We obtained the results of ILS directly from what is reported in [22], and ran experiments to collect results of kdRRF and our OCP planner with 100 trials for each scene. The success rates and average planning times of different methods are summarized in Fig. 11. Note that the evaluations of ILS in [22] did not consider any uncertainties, i.e., the outcome of the planned motions was perfectly predicted. However, due to the inevitable discrepancies between the object-centric physics model Π (defined in Sec IV) and the actual Γ (defined in Sec. III), our OCP planner was additionally challenged by the modeling uncertainties even though evaluated in the same setup. From the results reported in Fig. 11, kdRRF was not able to finish the *Separate* tasks. This is because the sampled actions by kdRRF are robot-centric, which causes the outcome object motions to be relatively random and makes it almost impossible to precisely relocate the objects inside the small-sized goal regions. More importantly, compared to ILS in all tasks, our OCP planner was able to achieve a similarly high success rate with significantly less planning time, regardless of the setup and the number of objects in the scene. This has shown an apparent improvement in planning efficiency by leveraging the object-centric paradigm of our planner.

D. Comparative Evaluations on a Physical Robot

To more realistically challenge our OCP planner with real-world uncertainties and explore the possibilities of deploying our planner in a real-world setup, we conducted real-world experiments on a physical 7-DoF Franka Emika Panda robot platform, as displayed in Fig. 12. We 3D printed a pusher to replace one finger of the robot gripper. The workspace plane is made of a transparent panel and the objects are tracked via AprilTags [87] by two cameras from beneath the workspace.

First, we evaluated our planner by comparing it with the learning-based MCTS [20]. We used a similar task setup as

(a) Singulate (N = 33)			
Metric	ILS [22]	kdRRF [26]	OCP (Ours)
Success Rate	95 %	71 %	**97 %**
Planning Time (seconds)	> 4.0	8.0 ± 5.4	**2.4 ± 3.2**
Num. Actions	–	128.9 ± 85.5	**26.5 ± 14.4**

(b) Separate (N = 25)			
Metric	ILS [22]	kdRRF [26]	OCP (Ours)
Success Rate	**100 %**	0 %	98 %
Planning Time (seconds)	> 4.6	> 20.0	**4.5 ± 4.2**
Num. Actions	–	> 904	**72.4 ± 47.0**

(c) Sorting with Goal Regions				
Scene	Metric	ILS [22]	kdRRF [26]	OCP (Ours)
N = 24	Success Rate	97 %	99 %	**100 %**
	Time (seconds)	> 18.0	13.5 ± 4.7	**4.3 ± 1.5**
	Num. Actions	–	304.5 ± 105.7	**117.5 ± 26.1**
N = 100	Success Rate	**100 %**	93 %	**100 %**
	Time (seconds)	> 400	126.2 ± 22.4	**100.8 ± 40.5**
	Num. Actions	–	1091.4 ± 185.4	**374.1 ± 94.4**

Fig. 11: Simulation evaluations of different methods on *Singulate*, *Separate*, and *Sorting with Goal Regions* ($N = 24$ and 100) tasks. Due to the absence of reported values in the original work of [22], the planning time of ILS is roughly estimated by processing the presented plot of the cumulative distribution functions.
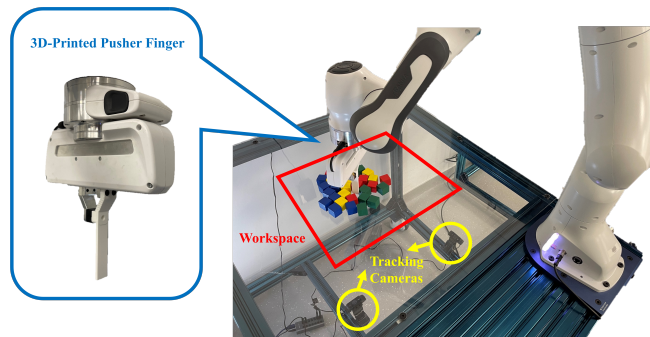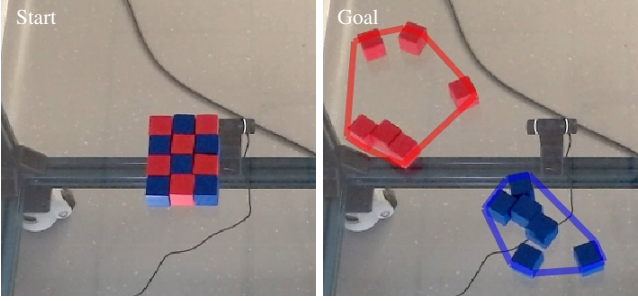

Fig. 12: The experiment setup of the physical robot platform. The objects were tracked by two cameras below the transparent workspace.

the physical experiments in [20]. Specifically, the experiments were conducted on the *Sorting without Goal Regions* task, using the same 1-inch wooden cubes as the objects to manipulate. The task scenes used for evaluation were also the same as [20]: 1) 2 classes × 6 objects, and 2) 3 classes × 5 objects. Since the hardware robot we used is different from [20] and the robot had different reachability relative to the workspace, we could not use the same workspace as [20]. However, we ensured the area of our workspace (a rectangle of size $33cm \times 30cm$) is approximately the same with [20]. In addition, for a more extensive comparison, we also implemented another baseline kdRRF [26] on the real robot. We ran kdRRF and our OCP planner for 10 trials in each task scene, with the same initial object configuration as [20], as shown in Fig. 13. For both scenes, the time budgets were set to 10 minutes for robot execution, i.e., a task failure will be reported if the robot does not finish the task within 10 minutes. The results of relevant metrics are reported in Fig. 13, where the results for MCTS are directly obtained from the work [20] (only the success rate and number of actions are reported). In the results, our OCP planner always succeeded within 5 minutes and outperformed both baselines regarding the number of actions needed for task completion. Furthermore, compared to kdRRF, our OCP planner also
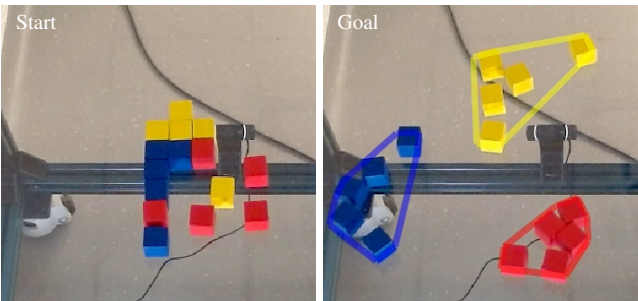
Scene I ($L = 2$, $N = 12$)

Metric	MCTS [20]	kdRRF [26]	OCP (Ours)
Success Rate	16 / 20	**10 / 10**	**10 / 10**
Planning Time (seconds)	–	16.5 ± 5.3	**9.5 ± 1.9**
Plan. Time / Action (seconds)	–	0.69	**0.50**
Execution Time (minutes)	–	6.2 ± 1.9	**4.0 ± 0.5**
Num. Actions	37.0 ± 3.5	23.8 ± 8.1	**19.1 ± 2.9**

(a) 2 classes × 6 objects ($N = 12$)

Scene II ($L = 3$, $N = 15$)

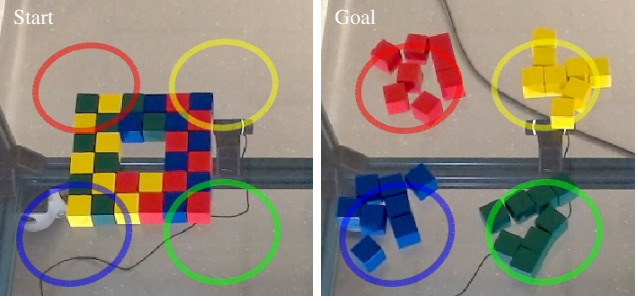| Metric	MCTS [20]	kdRRF [26]	OCP (Ours)
Success Rate	15 / 20	5 / 10	**10 / 10**
Planning Time (seconds)	–	27.3 ± 9.5	**9.3 ± 2.7**
Plan. Time / Action (seconds)	–	0.97	**0.46**
Execution Time (minutes)	–	6.9 ± 1.7	**3.5 ± 0.6**
Num. Actions	27.7 ± 2.1	28.2 ± 8.8	**20.5 ± 4.3**

(b) 3 classes × 5 objects ($N = 15$)

Fig. 13: Real-world evaluations of different methods on the *Sorting without Goal Regions* task under two different settings, where the results of MCTS are directly obtained from [20].
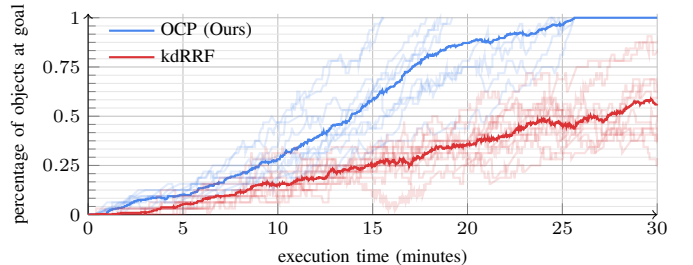
required significantly less planning time while achieving a higher success rate. This shows that, in the real-world setup, our planner is still consistently able to plan efficiently and generate more effective actions.

Then, we compared our planner against the ILS and kdRRF baseline on the same task as in [22], which is *Sorting with Goal Regions* for $N = 32$ cubes (i.e., 4 classes × 8 cubes). We used the same setting as [22], including the same size of cubes and circular goal regions, and the same time budget (i.e., 30 minutes for execution). Due to the different experiment space and hardware, the workspace of our setup has a different length and width (ours is $394mm \times 330mm$). However, for a fair comparison, we ensured that the area of our workspace was the same as in [22] to challenge our planner with the same packing factor. We ran kdRRF and our OCP planner for 10 trials, with the same adversarially designed initial object configuration as used by [22]. The results of relevant metrics are reported in Fig. 14. Since the actions generated by the open-loop ILS can easily diverge the execution, the results of ILS reported in [22] are from running an ϵ-greedy policy alternative. Except for the success rate, no detailed statistics

Scene III ($L = 4$, $N = 32$)

Metric	ILS [22]	kdRRF [26]	OCP (Ours)
Success Rate	5 / 5	0 / 10	**10 / 10**
Planning Time (seconds)	–	> 181.8	83.2 ± 15.1
Plan. Time / Action (seconds)	–	0.84	0.72
Execution Time (minutes)	< 30	> 30	20.7 ± 4.0
Num. Actions	> 160	> 215.8	**115.4 ± 20.1**

Fig. 14: Real-world evaluations of different methods on the *Sorting with Goal Regions* task for 4 classes × 8 objects, where the results of ILS are directly obtained from [22]. The bottom figure plots the percentage of objects at their goals throughout the execution of the kdRRF baseline (red) and our object-centric planner OCP (blue). The transparent plots show each execution of the 10 trials, and the solid plots are the average across all 10 trials.

about other metrics are reported in the work of [22], so we roughly counted the execution time and the number of actions from their supplementary video to the paper. Even by a rough comparison, it is clear that our OCP planner enabled the robot to complete the task in less time by generating more effective actions. In Fig. 14 (bottom), for every trial of kdRRF and our OCP planner, we plotted the percentage of objects inside their goal regions throughout the execution. While kdRRF still made steady progress as the execution proceeded, it never succeeded within the 30-minute budget. In contrast, our planner always successfully sorted all objects within roughly 25 minutes of execution.

E. Real-world Qualitative Demonstration

All the aforementioned quantitative evaluations were performed on cube-shaped objects for fair comparisons with existing baselines. However, not limited to cubes, our object-centric planner OCP can generalize to manipulate arbitrary object shapes under real-world settings. As shown in Fig. 15 (top), our planner can rearrange non-convex objects of shapes of "T", "R", and "O" to draw the letters formed by their final configurations. Moreover, in Fig. 15 (bottom), we show that our OCP planner can also rearrange objects of different sizes into clusters corresponding to the colors of objects.

VIII. BENCHMARK

In general, we perceive the nonprehensile rearrangement as a combination of global manipulation of object clusters and
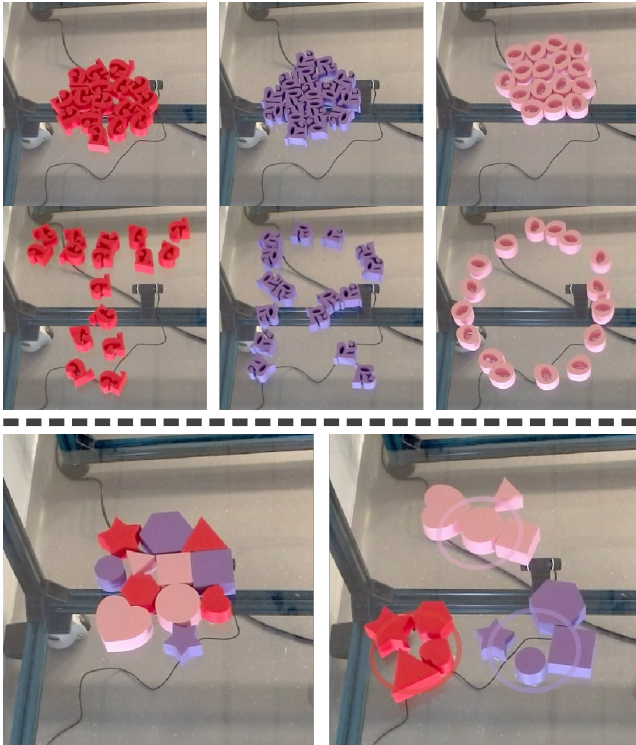
Fig. 15: *Top:* Letter-shaped objects of "T", "R", and "O" are rearranged to form their corresponding letter specified by the goal regions of all objects. *Bottom:* Objects of different shapes (heart, star, square, and triangle) and different sizes (ranging from $4cm$ to $7cm$) are sorted into three clusters with different object colors.

local relocation of single objects. We propose a standardized real-world benchmarking protocol to facilitate future research in nonprehensile manipulation, with rearrangement tasks requiring both intra-class global rearrangement and inter-class relocation. We also include evaluated metrics of our OCP planner as a benchmark baseline. In our benchmark, we use the 1-inch wooden cubes (Item #65 in the YCB dataset [88]) as the objects to rearrange. The following metrics are used to evaluate relevant methods:

1) *Success Rate:* the percentage of successful trials within the given time budget for execution.
2) *Planning Time:* the average planning time of running the planner for successful trials, excluding the execution time of the robot.
3) *Execution Time:* the average time of the entire robot execution for successful trials.
4) *Number of Actions:* the average number of robot actions needed for a successful task completion.
5) *Planning Time per Action:* the average runtime of the planner for generating a robot action.

There are in total five scenes proposed in the benchmark, with different packing factors (i.e., the area of the cubes over the area of the workspace) to reflect the difficulty of each scene. Each scene has a virtual out-of-bounds region of $2cm$ thickness. The first three scenes, Scene I (Fig. 13a), II (Fig. 13b), and III (Fig. 14), are just the scenes used for the real-world evaluation in Sec. VII-D. In addition, we customized another two real-world benchmark scenarios,

Scene IV and V, as shown in Fig. 16: The objects of different classes need to be separated into individual clusters, and at the same time, the position of each object in the same class has to lie in one of the distinct non-overlapping goal regions. Note that the objects in the same class are not unique, and their corresponding goal regions are interchangeable and dynamically assigned by minimizing the summed distance between objects and their goals. For both Scene IV and V, the workspace is restricted to a $30cm \times 30cm$ square; the goal regions for all objects are circles with a radius of $2cm$. We adversarially designed the initial object configurations for both scenarios to make them sufficiently challenging. We ran our OCP planner 10 times by setting execution time budgets of 10 minutes (Scene IV) and 15 minutes (Scene V). The statistics of relevant metrics are reported in Fig. 16. All five scenes of the benchmark are summarized below; for Scene I, II, and III, the initial and final object configurations, and the associated metrics evaluated using our planner can be found in the referred figures.

1) Scene I (packing factor: 0.10): Sorting 2 classes × 6 objects without explicit goal definition, as shown in Fig. 13a.
2) Scene II (packing factor: 0.13): Sorting 3 classes × 5 objects without explicit goal definition, as shown in Fig. 13b.
3) Scene III (packing factor: 0.20): Sorting 4 classes × 8 objects with explicit goal definition, as shown in Fig. 14.
4) Scene IV (packing factor: 0.095): Rearranging 2 classes × 5 objects ($N = 10$). The centers of goal regions are at $(9, 9)cm$, $(3, 9)cm$, $(-3, 9)cm$, $(9, 3)cm$, and $(9, -3)cm$ for the first class (blue); and are at $(-9, -9)cm$, $(-9, -3)cm$, $(-9, 3)cm$, $(-3, -9)cm$, $(3, -9)cm$ for the second class (red), as shown in Fig. 16 (top).
5) Scene V (packing factor: 0.15): Rearranging 4 classes × 4 objects ($N = 16$). The centers of goal regions are evenly distributed on a circle of radius $10cm$, and the goal regions of the same class are adjacent, as shown in Fig. 16 (bottom).

IX. CONCLUSION

In this work, we proposed a novel *object-centric* planning paradigm for nonprehensile object rearrangement. In contrast to the traditional robot-centric planning which first samples robot actions and then selects actions to execute by evaluating their predicted outcomes, our novel planning paradigm first reasons about the desired outcomes (i.e., the desired object motions) of robot actions and then realizes the desired object motions via closed-loop pushing actions generated online. Using the object-centric paradigm, we proposed a sampling-based rearrangement planner OCP which can more efficiently plan and generate task-effective actions. We equip our proposed OCP with two different exploration modes that can be switched alternatively, which enables a goal-oriented and extensive search over the problem space to more efficiently find the desired object arrangement achievable by the robot execution.

Scene IV ($L = 2$, $N = 10$)

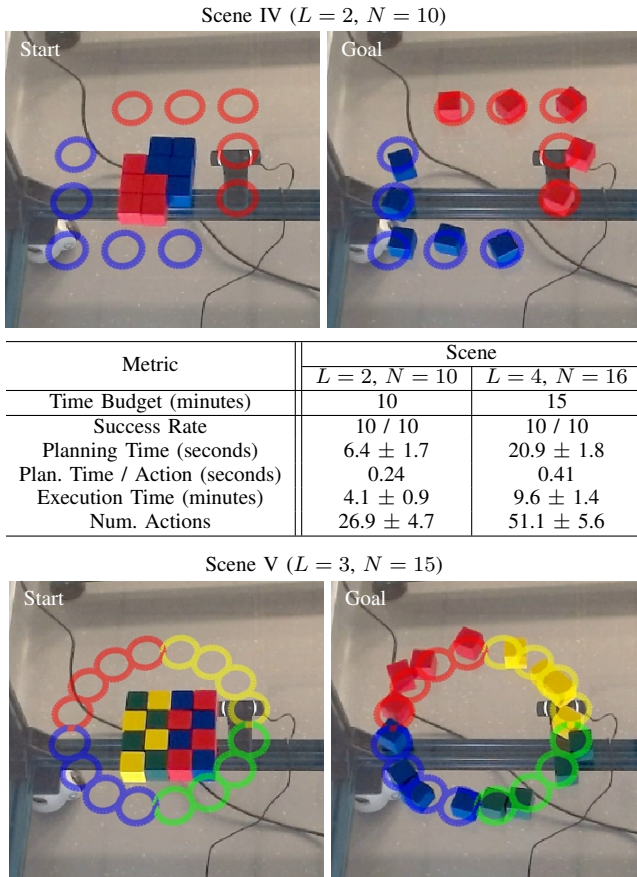| Metric	Scene	
	$L = 2$, $N = 10$	$L = 4$, $N = 16$
Time Budget (minutes)	10	15
Success Rate	10 / 10	10 / 10
Planning Time (seconds)	6.4 ± 1.7	20.9 ± 1.8
Plan. Time / Action (seconds)	0.24	0.41
Execution Time (minutes)	4.1 ± 0.9	9.6 ± 1.4
Num. Actions	26.9 ± 4.7	51.1 ± 5.6

Scene V ($L = 3$, $N = 15$)

Fig. 16: Initial and final configurations of the customized Scene IV (top) and Scene V (bottom) in our proposed benchmark. The relevant metrics evaluated with our proposed object-centric planner OCP on both scenes are given in the table (middle).

With extensive simulation and real-world experiments by comparing against selected state-of-the-art baselines on various rearrangement tasks, we show that our object-centric planner can improve the planning efficiency by reducing the required runtime, and generate more effective robot actions to reduce the execution time for task completion. In addition, we propose a real-world benchmarking protocol and provide relevant metrics evaluated using our proposed planner to facilitate future research in nonprehensile rearrangement.

In future work, we plan to incorporate prehensile or other nonprehensile primitives other than pushing to generate more diverse and complete rearrangement solutions, while reducing the involved uncertainties during planning. We also consider generalizing the current framework to more challenging task setups, such as dual-arm manipulation and rearranging objects with more dynamic motions (e.g., rolling balls).

REFERENCES

[1] M. R. Dogar and S. S. Srinivasa, "A framework for push-grasping in clutter." in *Robotics: Science and Systems*, vol. 2, 2011.

[2] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 183–189.

[3] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *The International Journal of Robotics Research*, vol. 2, no. 04, pp. 479–503, 2005.

[4] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "An effective framework for path planning amidst movable obstacles," in *Algorithmic Foundation of Robotics VII: Selected Contributions of the Seventh International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2008, pp. 87–102.

[5] J. Van Den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path planning among movable obstacles: a probabilistically complete approach," in *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2010, pp. 599–614.

[6] M. Gupta and G. S. Sukhatme, "Using manipulation primitives for brick sorting in clutter," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 3883–3889.

[7] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of the fourth annual symposium on Computational geometry*, 1988, pp. 279–288.

[8] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[9] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 3327–3332.

[10] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile dynamic manipulation: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1711–1718, 2018.

[11] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3940–3947.

[12] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3075–3082.

[13] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2508–2515.

[14] R. Wang, K. Gao, D. Nakhimovich, J. Yu, and K. E. Bekris, "Uniform object rearrangement: From complete monotone primitives to efficient non-monotone informed search," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6621–6627.

[15] K. Gao, S. W. Feng, B. Huang, and J. Yu, "Minimizing running buffers for tabletop object rearrangement: Complexity, fast algorithms, and applications," *The International Journal of Robotics Research*, vol. 42, no. 10, pp. 755–776, 2023.

[16] Muhayyuddin, M. Moll, L. Kavraki, J. Rosell *et al.*, "Randomized physics-based motion planning for grasping in cluttered and uncertain environments," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 712–719, 2017.

[17] W. C. Agboh and M. R. Dogar, "Real-time online re-planning for grasping under clutter and uncertainty," in *IEEE International Conference on Humanoid Robots (HUMANOIDS)*. IEEE, 2018, pp. 1–8.

[18] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[19] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics Research: The 18th International Symposium ISRR*. Springer, 2020, pp. 405–419.

[20] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9433–9440.

[21] B. Huang, T. Guo, A. Boularias, and J. Yu, "Interleaving monte carlo tree search and self-supervised learning for object retrieval in clutter," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 625–632.

[22] E. Huang, Z. Jia, and M. T. Mason, "Large-scale multi-object rearrangement," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 211–218.

[23] Z. Pan and K. Hauser, "Decision making in joint push-grasp action space for large-scale object sorting," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6199–6205.

[24] E. R. Vieira, D. Nakhimovich, K. Gao, R. Wang, J. Yu, and K. E. Bekris, "Persistent homology for effective non-prehensile manipulation,"

in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1918–1924.

[25] K. Ren, L. E. Kavraki, and K. Hang, "Rearrangement-based manipulation via kinodynamic planning and dynamic planning horizons," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 1145–1152.

[26] K. Ren, P. Chanrungmaneekul, L. E. Kavraki, and K. Hang, "Kinodynamic rapidly-exploring random forest for rearrangement-based nonprehensile manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 8127–8133.

[27] K. Gao, D. Lau, B. Huang, K. E. Bekris, and J. Yu, "Fast high-quality tabletop rearrangement in bounded workspace," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1961–1967.

[28] R. Wang, Y. Miao, and K. E. Bekris, "Efficient and high-quality prehensile rearrangement in cluttered and confined spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1968–1975.

[29] B. Wen, W. Lian, K. Bekris, and S. Schaal, "Catgrasp: Learning category-level task-relevant grasping in clutter from simulation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 6401–6408.

[30] M. T. Mason, "Progress in nonprehensile manipulation," *The International Journal of Robotics Research*, vol. 18, no. 11, pp. 1129–1141, 1999.

[31] N. Dengler, D. Großklaus, and M. Bennewitz, "Learning goal-oriented non-prehensile pushing in cluttered scenes," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 1116–1122.

[32] K. Hang, A. S. Morgan, and A. M. Dollar, "Pre-grasp sliding manipulation of thin objects using soft, compliant, or underactuated hands," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 662–669, 2019.

[33] C. Song and A. Boularias, "A probabilistic model for planar sliding of objects with unknown material properties: Identification and robust planning," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5311–5318.

[34] X. Yi and N. Fazeli, "Precise object sliding with top contact via asymmetric dual limit surfaces," 2023.

[35] Y. Hou, Z. Jia, A. M. Johnson, and M. T. Mason, "Robust planar dynamic pivoting by regulating inertial and grip forces," in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020, pp. 464–479.

[36] X. Zhang, S. Jain, B. Huang, M. Tomizuka, and D. Romeres, "Learning generalizable pivoting skills," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5865–5871.

[37] N. Doshi, O. Taylor, and A. Rodriguez, "Manipulation of unknown objects via contact configuration regulation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2693–2699.

[38] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6723–6729.

[39] F. Paus, T. Huang, and T. Asfour, "Predicting pushing action effects on spatial object relations by learning internal prediction models," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10584–10590.

[40] M. S. Saleem and M. Likhachev, "Planning with selective physics-based simulation for manipulation among movable objects," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6752–6758.

[41] J. Lee, C. Nam, J. Park, and C. Kim, "Tree search-based task and motion planning with prehensile and non-prehensile manipulation for obstacle rearrangement in clutter," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8516–8522.

[42] R. Shome, W. N. Tang, C. Song, C. Mitash, H. Kourtev, J. Yu, A. Boularias, and K. E. Bekris, "Towards robust product packing with a minimalistic end-effector," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9007–9013.

[43] J. Zhou, M. T. Mason, R. Paolini, and D. Bagnell, "A convex polynomial model for planar sliding mechanics: theory, application, and experimental validation," *The International Journal of Robotics Research*, vol. 37, no. 2-3, pp. 249–265, 2018.

[44] J. Zhou, Y. Hou, and M. T. Mason, "Pushing revisited: Differential flatness, trajectory planning, and stabilization," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1477–1489, 2019.

[45] N. Chavan-Dafle, R. Holladay, and A. Rodriguez, "Planar in-hand manipulation via motion cones," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 163–182, 2020.

[46] M. Halm and M. Posa, "A quasi-static model and simulation approach for pushing, grasping, and jamming," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer, 2020, pp. 491–507.

[47] F. Bertoncelli, F. Ruggiero, and L. Sabattini, "Linear time-varying mpc for nonprehensile object manipulation with a nonholonomic mobile robot," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11032–11038.

[48] F. R. Hogan and A. Rodriguez, "Reactive planar non-prehensile manipulation with hybrid model predictive control," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 755–773, 2020.

[49] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 30–37.

[50] M. Bauza, F. R. Hogan, and A. Rodriguez, "A data-efficient approach to precise and controlled pushing," in *Conference on Robot Learning*. PMLR, 2018, pp. 336–345.

[51] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3066–3073.

[52] M. Bauza, F. Alet, Y.-C. Lin, T. Lozano-Pérez, L. P. Kaelbling, P. Isola, and A. Rodriguez, "Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgb-d video," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4265–4272.

[53] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects from sensory data," *The International Journal of Robotics Research*, vol. 41, no. 8, pp. 778–797, 2022.

[54] S. Zickler and M. M. Veloso, "Efficient physics-based planning: sampling search via non-deterministic tactics and skills." in *AAMAS (1)*, 2009, pp. 27–33.

[55] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt, "Two-level rrt planning for robotic push manipulation," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 678–685.

[56] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[57] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2007, pp. 704–710.

[58] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 2427–2433.

[59] M. C. Koval, J. E. King, N. S. Pollard, and S. S. Srinivasa, "Robust trajectory selection for rearrangement planning as a multi-armed bandit problem," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 2678–2685.

[60] A. M. Johnson, J. E. King, and S. Srinivasa, "Convergent planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1044–1051, 2016.

[61] W. Bejjani, R. Papallas, M. Leonetti, and M. R. Dogar, "Planning with a receding horizon for manipulation in clutter using a learned value function," in *IEEE International Conference on Humanoid Robots (HUMANOIDS)*. IEEE, 2018, pp. 1–9.

[62] R. Wang, K. Gao, J. Yu, and K. Bekris, "Lazy rearrangement planning in confined spaces," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 385–393.

[63] K. Wada, S. James, and A. J. Davison, "Reorientbot: Learning object reorientation for specific-posed placement," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8252–8258.

[64] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets *et al.*, "Habitat 2.0: Training home assistants to rearrange their habitat," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 251–266, 2021.

[65] C. Gan, S. Zhou, J. Schwartz, S. Alter, A. Bhandwaldar, D. Gutfreund, D. L. Yamins, J. J. DiCarlo, J. McDermott, A. Torralba *et al.*, "The threedworld transport challenge: A visually guided task-and-motion planning benchmark towards physically realistic embodied ai," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8847–8854.

[66] S. D. Han, N. M. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, "Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1775–1795, 2018.

[67] Y. Labbé, S. Zagoruyko, I. Kalevatykh, I. Laptev, J. Carpentier, M. Aubry, and J. Sivic, "Monte-carlo tree search for efficient visually guided rearrangement planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3715–3722, 2020.

[68] C. Nam, S. H. Cheong, J. Lee, D. H. Kim, and C. Kim, "Fast and resilient manipulation planning for object retrieval in cluttered and confined environments," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1539–1552, 2021.

[69] R. Shome, K. Solovey, J. Yu, K. Bekris, and D. Halperin, "Fast, high-quality two-arm rearrangement in synchronous, monotone tabletop setups," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 888–901, 2021.

[70] H. Tian, C. Song, C. Wang, X. Zhang, and J. Pan, "Sampling-based planning for retrieving near-cylindrical objects in cluttered scenes using hierarchical graphs," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 165–182, 2022.

[71] D. Halperin, M. van Kreveld, G. Miglioli-Levy, and M. Sharir, "Space-aware reconfiguration," *Discrete & Computational Geometry*, vol. 69, no. 4, pp. 1157–1194, 2023.

[72] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2021, pp. 726–747.

[73] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox, "Nerp: Neural rearrangement planning for unknown objects," *Robotics: Science and Systems*, 2021.

[74] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6010–6017.

[75] J. Liang, B. Wen, K. Bekris, and A. Boularias, "Learning sensorimotor primitives of sequential manipulation tasks from visual demonstrations," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8591–8597.

[76] X. Zhang, Y. Zhu, Y. Ding, Y. Zhu, P. Stone, and S. Zhang, "Visually grounded task and motion planning for mobile manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1925–1931.

[77] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 2086–2092.

[78] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork, "End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer," *Robotics and Autonomous Systems*, vol. 119, pp. 119–134, 2019.

[79] S. D. Han, B. Huang, S. Ding, C. Song, S. W. Feng, M. Xu, H. Lin, Q. Zou, A. Boularias, and J. Yu, "Toward fully automated metal recycling using computer vision and non-prehensile manipulation," in *IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 891–898.

[80] B. Tang and G. S. Sukhatme, "Selective object rearrangement in clutter," in *Conference on Robot Learning*. PMLR, 2023, pp. 1001–1010.

[81] B. Huang, S. D. Han, A. Boularias, and J. Yu, "Dipn: Deep interaction prediction network with application to clutter removal," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4694–4701.

[82] L. P. Kaelbling, "The foundation of efficient robot learning," *Science*, vol. 369, no. 6506, pp. 915–916, 2020.

[83] G. Wang, K. Ren, and K. Hang, "Uno push: Unified nonprehensile object pushing via non-parametric estimation and model predictive control," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024.

[84] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 1997, pp. 2719–2726.

[85] J. Haviland and P. Corke, "Manipulator differential kinematics: Part 2: Acceleration and advanced applications," *IEEE Robotics and Automation Magazine*, 2023.

[86] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[87] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 3400–3407.

[88] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.