

Object-Centric Kinodynamic Planning for Nonprehensile Robot Rearrangement Manipulation

Kejia Ren¹, Gaotian Wang¹, Andrew S. Morgan², Lydia E. Kavraki¹, and Kaiyu Hang¹

Abstract—Nonprehensile actions such as pushing are crucial for addressing multi-object rearrangement problems. Many traditional methods generate *robot-centric* actions, which differ from intuitive human strategies and are typically inefficient. To this end, we adopt an *object-centric* planning paradigm and propose a unified framework for addressing a range of large-scale, physics-intensive nonprehensile rearrangement problems challenged by modeling inaccuracies and real-world uncertainties. By assuming each object can actively move without being driven by robot interactions, our planner first computes desired object motions, which are then realized through 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* planning framework 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. It is important in many real-world applications, such as warehouse sorting [7], recycling lines [8], and home cleaning [9]; and often serves as a key step in downstream manipulation pipelines such as bin picking [10] and packing [11]. 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 [12]. Traditional planning methods allow only a pick-and-place type of prehensile action to move one object at a time [13], [14]. 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 gener-

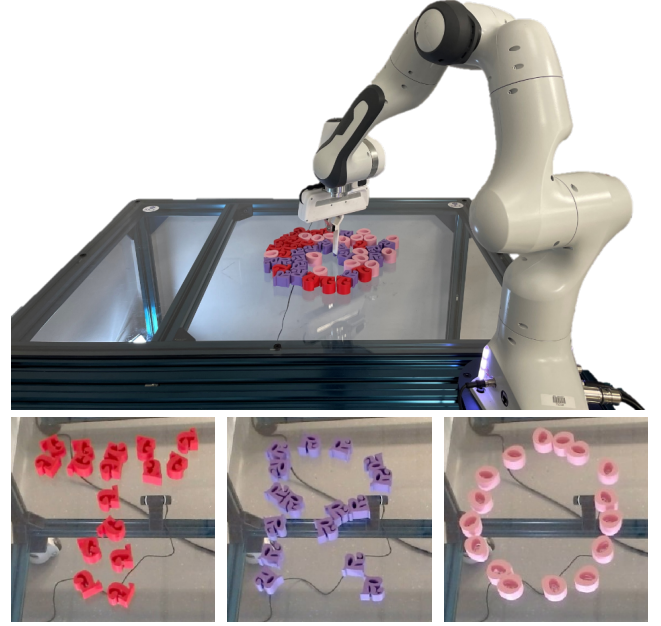


Fig. 1: Through object-centric planning, our framework is able to efficiently rearrange multiple movable objects of different shapes to accomplish various tasks. In the scene, “T”, “R”, and “O” letter-shaped objects are rearranged to form the abbreviation “TRO”.

ate 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 [15]. However, nonprehensile rearrangement planning is challenging due to the sophisticated physics modeling and the high-dimensional problem space it entails. Furthermore, the inevitable modeling inaccuracy of the involved physics makes nonprehensile solutions less robust to real-world uncertainties and causes task failures. To this end, developing an efficient and also reactive manipulation planner for nonprehensile multi-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.

Robot-Centric vs. Object-Centric Planning. Traditional nonprehensile rearrangement planners [16], [17] are mostly *robot-centric*: During planning, robot actions (e.g., joint-space control) are sampled considering only robot-relevant geometric and kinematic constraints. The resulting object motions are predicted after sampling and used to select promising samples for execution. Such *robot-centric* strategies can be inefficient, as planners often spend unnecessary time sampling and evalu-

This work was supported by the U.S. National Science Foundation (NSF) under grants FRR-2133110 and FRR-2240040 (KR, GW, and KH), and grant CCF-2336612 (LEK).

¹Kejia Ren, Gaotian Wang, Lydia E. Kavraki, and Kaiyu Hang are with the Department of Computer Science, Rice University, Houston, TX 77005, USA (email: kr43@rice.edu; gw23@rice.edu; kavraki@rice.edu; kaiyu.hang@rice.edu).

²Andrew S. Morgan is with the RAI Institute, Cambridge, MA 02142, USA (e-mail: andy@rai-inst.com).

	Object-Centric	Nonprehensile	Data-Driven	Closed-Loop	Versatile	Explicit-Goal	Max # Objects	
							Sim	Real
[24], [25]	✓	✗	✗	✗	✗	✓	30 - 50	12
[16], [17]	✗	✓	✗	✗	✗	✗	6 - 7	7
[18]	✓	✓	✗	✗	✗	✗	7	1
[26], [27]	✗	✓	✗	✓	✗	✗	15 - 40	9 - 16
[28]–[31]	✗	✓	✓	✓	✗	✗	11 - 40	8 - 30
[22]	✓	✓	✗	✗	✓	✓	100	32
[32]	✗	✓	✗	✓	✗	✗	200	20
[33]	✗	✓	✗	✗	✗	✗	10	–
[34], [35]	✗	✓	✗	✓	✓	✗	36	20
[23]	✓	✓	✗	✗	✗	✗	16	5
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 incorporates object-centric components (e.g., sampling strategy, action primitives, planning paradigm, etc), 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) *Versatile*: the method has shown the capability to transfer across different rearrangement tasks (e.g., relocating, separating, sorting, etc); 6) *Explicit-Goal*: the method requires an 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.

ating actions that fail to make meaningful interactions with objects. To this end, *object-centric* actions, generated with object-relevant intent [18], have been incorporated. Early work [19], [20] treat the objects (or parts) to be manipulated as actively movable robots, but focused on geometric or prehensile manipulation planning problems, without the ability to integrate physics-involved nonprehensile actions. To better target action generation towards interactions with objects, prior works either define *object-centric* action primitives [18], [21] or bias action sampling near objects [22]. A more intuitive *object-centric* paradigm [23], inspired by how humans reason about rearrangement, decouples the problem into two steps: first, it searches for desired object motions by assuming objects can move on their own (i.e., without considering the robot); then, it computes the robot motions needed to realize those object trajectories. This paradigm is most methodologically aligned with our proposed framework. However, unlike [23], which was open-loop and originally developed for object retrieval in confined spaces, our framework addresses a broader range of large-scale rearrangement tasks involving intensive object-object interactions and real-world uncertainties, by employing closed-loop action generation in real time.

While various nonprehensile rearrangement planners have been investigated, most of them are limited in problem scale or robustness against uncertainties. Some are tailored to specific, small-scale tasks (such as retrieving [36] or pushing a single target object [17]) and tend to become inefficient in more cluttered and complex scenarios. Others rely on explicitly defined goal locations to guide sampling, which restricts their flexibility in more general rearrangement problems [22]. Moreover, many of these methods are open-loop, making them vulnerable to real-world uncertainties and modeling errors, especially in physics-intensive and longer-horizon tasks where errors can easily accumulate. To this end, in this work, we propose an object-centric planning (OCP) framework that

- 1) as a unified framework, efficiently addresses a range of planar nonprehensile rearrangement problems (e.g., with or without explicit goals, single or multiple target objects)

previously handled by separate approaches;

- 2) reacts to perception and physical uncertainties and is robust against modeling inaccuracies, through real-time actions generated in a closed-loop;
- 3) generalizes and scales to various large-scale rearrangement tasks with different task objectives.

Limitations and Generalizability. The scope of our proposed framework is focused on 2D planar (e.g., tabletop) pushing-based nonprehensile rearrangement problems. While incorporating other types of motions (such as sliding or pivoting) could extend its applicability to more general object rearrangement scenarios, those actions are beyond the scope of the current work. Nonetheless, within this category, our framework is highly generalizable: it can be applied to a rearrangement problem as long as the task objective can be described by a heuristic function (as will be introduced in Sec. IV-A), including various settings where one, some, or all objects are considered target objects and whether or not explicit goal poses are specified (see example tasks in Sec. VII-A).

The rest of the paper is structured as follows. We review related literature in Sec. II and provide preliminaries in Sec. III. Under our *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

The table in Fig. 2 qualitatively compares state-of-the-art rearrangement solutions with our proposed planner across several key aspects, as shown in the table columns. 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 related literature from multiple perspectives.

Nonprehensile Manipulation and Planar Pushing: Prehensile manipulation, which relies on robotic grasping, has been extensively studied to facilitate many manipulation tasks [37]–[39]. As complementary skills to prehensile manipulation, nonprehensile manipulation (defined as manipulation without grasping [40]) such as pushing [41], sliding [42]–[44], and pivoting [45], [46], has also been actively explored, to expand the possibilities from single object reconfiguration [47] to large-scale object rearrangement in clutters [48]–[51]. Among them, pushing has gained particular attention for its capability to operate in confined workspaces [11]. Planar pushing of a single object is one of the simplest scenarios of pushing-based manipulation. Existing works have developed analytical models [15], [52]–[57] to address the precise planar pushing problem. However, since they rely on known contact geometry and physical properties, and are typically derived under oversimplified assumptions (e.g., convex shapes), these analytical approaches do not scale well to real-world scenarios. More recently, data-driven methods [58]–[62] 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 substantial 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 extended to 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 its environment, modeling of such physics [63], [64] 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 [65]–[67]. In nonprehensile manipulation problems such as object pushing, however, manipulation outcomes are often difficult to precisely predict, due to inaccuracies in interaction physics modeling and uncertainties in perception and execution. Such real-world uncertainties can easily cause the robot motions generated by kinodynamic planning to fail the real-world manipulation. To address such challenges, some approaches [26], [68], [69] incorporate an uncertainty model into the planning process, to generate more conservative motions and reduce the probabilities of execution failure. Another line of work closes the loop of planning and execution by iterative replanning with receding horizons [34], [35], [70].

Rearrangement-based Manipulation: Object rearrangement is common and important, involving manipulation of small objects in a confined space [71], [72] or heavy objects in a large space [9], [73]. When relying solely on prehensile actions (e.g., pick-and-place) and 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 prehensile approaches use graph-based or tree-based search [24], [25],

[74]–[79] to provide near-optimal solutions; some others use learned models [80]–[85] to handle more complex constraints. However, pick-and-place type of prehensile actions can be infeasible when 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. Using approximated analytical models or physics engines to predict the outcomes of physical interactions, sampling-based planners [16], [17], [33] have been proposed to generate open-loop motion plans. However, due to inaccuracies in physics modeling and real-world uncertainties, the generated plans often fail in execution, especially in long-horizon tasks where errors significantly accumulates. To improve robustness against real-world uncertainties through closed-loop execution, strategies such as online replanning [27], distance-guided greedy local search [22], [32], interleaving progress-controlled planning and execution [34], [35] have been proposed. 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 [86], multi-object sorting [8], [30], [87], object singulation for retrieval [29], [31] and object separation for clutter removal [28], [88]. These 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 [89]. For example, a policy trained for separating objects can be inefficient in relocating an object.

III. PRELIMINARIES

This work aims to address the problem of nonprehensile pushing-based object rearrangement on a 2D plane (e.g., tabletop), where a robot manipulator interacts with N movable objects in a bounded workspace $\mathcal{W} \subset \mathbb{R}^2$. 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 that rearranges the objects into desired states while keeping them within $\mathcal{W}$.

A. Kinodynamic Planning for Nonprehensile Rearrangement

The nonprehensile rearrangement problem is conventionally formulated as a kinodynamic planning problem, where higher-order derivatives (e.g., velocities of the robot and objects) need to be included in the state space to account for system dynamics. However, following the quasi-static assumption and similar to [17], [18], we simplify the problem by reducing the state space to configurations (of the robot and objects) only, excluding their velocities. As such, the simplified kinodynamic planning problem is defined 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. $\mathcal{Q}_{free}^R \subset \mathcal{Q}^R$ 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 movable

objects are allowed as they are necessary for manipulation. The robot state at time t is denoted by $\mathbf{q}_t \in \mathcal{Q}^R$.

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

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. A system state at time t is valid if the robot is collision-free $\mathbf{q}_t \in \mathcal{Q}_{free}^R$ and all object positions remain within the bounded workspace $\mathbf{p}_t^i \in \mathcal{W}, \forall i$. 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 be represented differently depending on the task-specific action definition. For an M -DoF robot manipulator, a common choice is the joint velocity space, i.e., $\mathcal{A} \subset \mathbb{R}^M$, where an action $\mathbf{a}_t \in \mathcal{A}$ 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, 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 Eq. (1):

$$(\mathbf{q}_{t+1}, \mathbf{s}_{t+1}) = \Gamma(\mathbf{q}_t, \mathbf{s}_t, \mathbf{a}_t) \quad (1)$$

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

5) *Goal Criterion*: To assess whether all objects are successfully rearranged, we define a criterion function $g : \mathcal{Q}^O \mapsto \{0, 1\}$, where $g(\mathbf{s}_t) = 1$ indicates that an arrangement $\mathbf{s}_t \in \mathcal{Q}^O$ satisfies the task goal. Some prior works instead specify an explicit goal region $\mathcal{G}_i \subset \mathcal{W}$ for each object i and check whether the object lies within it to determine the task completion [17], [22]. However, for other tasks, such as those requiring only relative configurations of some objects [30], explicitly defining $\mathcal{G}_i$ is unnecessary and difficult (see example tasks in Sec. VII). Hence, a criterion function g offers a more general and flexible representation of rearrangement goals.

We consider a robot starting at its initial configuration $\mathbf{q}_0$, tasked with rearranging multiple objects from an initial arrangement $\mathbf{s}_0 = (\mathbf{s}_0^1, \dots, \mathbf{s}_0^i, \dots, \mathbf{s}_0^N)$. The goal of non-prehensile rearrangement problem can be formally defined as finding a sequence of T robot actions, denoted as $\tau = \{\mathbf{a}_0, \dots, \mathbf{a}_t, \dots, \mathbf{a}_{T-1}\}$, such that:

- 1) The system state transitions under the system dynamics by $(\mathbf{q}_{t+1}, \mathbf{s}_{t+1}) = \Gamma(\mathbf{q}_t, \mathbf{s}_t, \mathbf{a}_t)$, $t = 0, \dots, T-1$.
- 2) For all intermediate time steps $t = 1, \dots, T$, the system state is always valid: The robot is in its free C-space $\mathbf{q}_t \in \mathcal{Q}_{free}^R$, and all objects are within the workspace $\mathbf{p}_t^i \in \mathcal{W}, \forall i = 1, \dots, N$.

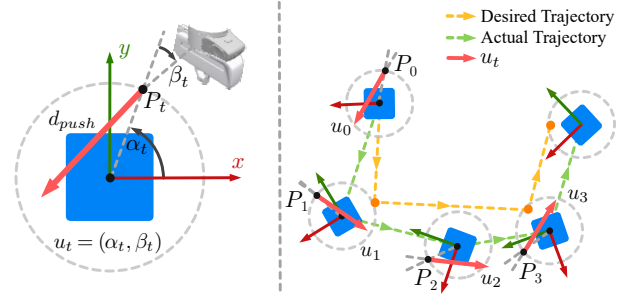


Fig. 3: *Left*: A robot pushing action is represented by $\mathbf{u}_t = (\alpha_t, \beta_t)$, where α_t and β_t are two angles specified in the object's body frame. To perform $\mathbf{u}_t$, the robot needs to first place its pusher at the position P_t (determined by α_t) and then translate in the 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 $\mathbf{u}_0$ through $\mathbf{u}_3$, 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.

- 3) The final arrangement, after the sequence of robot actions τ has been executed in the desired order, satisfies the goal criterion, i.e., $g(\mathbf{s}_T) = 1$.

B. Nonprehensile Object Pushing

In general, consider a single object currently at a state $\mathbf{s}_t \in SE(2)$ and given a desired state $\hat{\mathbf{s}}_{t+1} \in SE(2)$ for the next time step, the desired motion of the object can be accordingly calculated by $\Delta \mathbf{s}_t = (\mathbf{s}_t)^{-1} \cdot \hat{\mathbf{s}}_{t+1} \in SE(2)$, as a relative transformation expressed in the object's body frame. With an off-the-shelf pushing strategy, the desired pushing action $\mathbf{u}_t$ should be generated based on the desired $\Delta \mathbf{s}_t$. We generally denote this process by a function $\mathbf{u}_t \leftarrow \text{PUSHSTRATEGY}(\mathbf{s}_t, \hat{\mathbf{s}}_{t+1})$. As shown in Fig. 3 (left), the robot's pushing action can be parameterized by two angles defined in the object's body frame, $\mathbf{u}_t = (\alpha_t, \beta_t)$. The first angle α_t determines the robot pusher's starting position relative to the object, i.e., a point $P_t \in \mathbb{R}^2$ on a circle centered around the object, where the circle's radius corresponds to the object's size; the second angle β_t specifies the pushing direction. To execute $\mathbf{u}_t$, the robot needs to first place its pusher (or end-effector) to the starting position P_t (computed based on α_t), and then move its pusher in the pushing direction β_t for a fixed distance d_{push} along the workspace plane. The pushing distance d_{push} is a preset hyperparameter; given a fixed pushing frequency, a larger d_{push} is typically used when faster pushing is desired, while a smaller value is preferred when higher pushing precision is required. Note that $\mathbf{u}_t$ differs from the lower-level robot action $\mathbf{a}_t$ (e.g., joint velocities) defined in Sec. III-A. A robot controller (e.g., a Cartesian position controller) will convert $\mathbf{u}_t$ to $\mathbf{a}_t$, which can be directly commanded to the robot for execution.

By iteratively calling the pushing strategy and executing the generated actions in a closed loop, the robot can push the object to follow a 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 along this trajectory, through consecutively adjusting its pusher to the positions $P_0, \dots, P_3$ and executing the corresponding actions $\mathbf{u}_0, \dots, \mathbf{u}_3$ in between. In this work, we use the UNO Push framework from our previous work [90] as the pushing strategy.

IV. PROBLEM STATEMENT

Inspired by human strategies in rearranging objects, we propose to adopt an *object-centric* planning paradigm, as formulated in Sec. IV-A, to address 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*, where robot actions are sampled based solely on robot-relevant constraints, without explicitly targeting interactions with specific objects [16], [17]. The motions of surrounding objects, resulting from these sampled robot actions, are passively predicted by an approximate model of system dynamics Γ . As such, the generated robot actions are not sufficiently guided by how the objects need to be reconfigured; consequently, *robot-centric* strategies are often inefficient, due to their lack of active exploitation of task-relevant information (e.g., the desired object reconfigurations).

In this work, we approach the nonprehensile rearrangement problem using an *object-centric* formulation. In short, we first search to figure out how the objects need to be moved to accomplish the rearrangement task, by planning their desired trajectories independently of the robot. Then, using a nonprehensile object pushing strategy (e.g., the UNO Push framework [90]), the robot generates and executes actions in a closed-loop manner, to move each object along its desired trajectories planned beforehand.

To plan objects' desired trajectories, we assume each object can move autonomously on a flat surface (i.e., without needing to be moved by a robot), as if we equip each object with a virtual actuator to enable its translation and rotation in all directions. The desired trajectory of an object is represented by an ordered sequence of multiple waypoints in $SE(2)$, denoted by $\mathcal{T}^i = \{\hat{\mathbf{s}}_1^i, \dots, \hat{\mathbf{s}}_k^i, \dots, \hat{\mathbf{s}}_K^i\}$. Each waypoint $\hat{\mathbf{s}}_k^i = (\hat{\mathbf{p}}_k^i, \hat{\theta}_k^i) = (\hat{x}_k^i, \hat{y}_k^i, \hat{\theta}_k^i) \in SE(2)$ is a desired intermediate configuration of the object along the trajectory. If under perfect control and execution, the object is expected to reach each waypoint one by one from $\hat{\mathbf{s}}_1^i$ to $\hat{\mathbf{s}}_K^i$.

We assume only one object is activated at a time to actively move. The activated object does not have to be collision-free while following its desired trajectory since collisions between objects are allowed as aforementioned. As such, other objects may be passively moved due to object-object interactions. We represent such dynamics involving only mutual interactions between objects by an object-centric transition function Π :

$\mathcal{Q}^O \times \{1, \dots, N\} \times \Xi \mapsto \mathcal{Q}^O$, where Ξ is the space of all possible object trajectories:

$$\mathbf{s}_{t+1} = \Pi(\mathbf{s}_t, i, \mathcal{T}^i) \quad (2)$$

which infers the outcome arrangement $\mathbf{s}_{t+1} \in \mathcal{Q}^O$, given the current arrangement $\mathbf{s}_t \in \mathcal{Q}^O$, the index i of the activated object, and its trajectory $\mathcal{T}^i$. 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 Γ defined in Eq. (1), this object-centric transition function Π does not model any interactions involving the robot. This modeling simplification in 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-intensive 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 address this, rather than executing the entire plan at once, we interleave planning and real execution to progressively guide 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 $\mathbf{s}_t \in \mathcal{Q}^O$. A smaller value of $h(\mathbf{s}_t)$ indicates that the system state gets closer to achieving the task goal. As will be shown later in Sec VII, some simple yet effective distance-based functions can be easily used as heuristics for various practical rearrangement problems.

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

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

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

$$\forall d = 1, \dots, D : \quad (3c)$$

$$\mathbf{s}_{t+d} = \Pi(\mathbf{s}_{t+d-1}, i_d, \mathcal{T}^{i_d}), \quad (3d)$$

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

Specifically, given the current arrangement $\mathbf{s}_t$, the optimization problem aims to find an ordered sequence of D object-trajectory pairs, i.e., $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$, such that the heuristic cost $h(\mathbf{s}_{t+D})$ will be minimized after these D trajectories have been performed. Each pair in this sequence, $(i_d, \mathcal{T}^{i_d})$, consists of the index $i_d \in \{1, \dots, N\}$ of the activated object 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

Algorithm 1 Object-Centric Rearrangement Planning

Input: Initial arrangement s_0 , goal criterion $g(\cdot)$
Output: Task completion (*true* or *false*)

```

1:  $s^* \leftarrow s_0$ 
2: while TIME.AVAILABLE() and  $g(s^*) == 0$  do
    $\triangleright$  Task not Accomplished
3:    $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D \leftarrow \text{OCP}(s^*)$   $\triangleright$  Sec. V
4:   for  $d = 1, \dots, D$  do
5:     EXECUTE( $s^*, i_d, \mathcal{T}^{i_d}$ )  $\triangleright$  Sec. VI
6:      $s^* \leftarrow \text{OBSERVEOBJECTS}()$   $\triangleright$  via Real-time Sensing
7:   end for
8: end while
9: return true if  $g(s^*) == 1$  else false

```

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, leading to inefficient or even failed real-world rearrangement. 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 for nonprehensile rearrangement planning under our *object-centric* formulation is outlined in Alg. 1. Our framework starts with all objects being at states in $s_0 \in \mathcal{Q}^O$ (i.e., the initial arrangement). Without considering the robot, our object-centric planner (OCP) will first search for an ordered sequence of D object-trajectory pairs $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$ 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 $d = 1, \dots, D$, while strictly respecting the robot's physical and kinematic constraints, as will be described in Sec. VI. After each execution, the real-world 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 replanning and adjusting its actions, the robot reacts to 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 $n \in$

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 $\{(i_1, \mathcal{T}^{i_1}), \dots, (i_D, \mathcal{T}^{i_D})\}$

```

1:  $Tr \leftarrow \{n_{root}.s = s^*\}$   $\triangleright$  Add Root Node to Tree
2: while  $T.GETSIZE() < S_{\max}$  do
3:    $n \leftarrow \text{SAMPLENODE}(Tr)$   $\triangleright$  Sec. V-A
4:    $i \leftarrow \text{ACTIVATEOBJECT}(n)$   $\triangleright$  Sec. V-B
5:    $s_o, \mathcal{T}^i \leftarrow \text{EXPANDTREE}(n, i)$   $\triangleright$  Alg. 3
6:   if  $\forall i, s_o \in \mathcal{W}$  and  $\mathcal{T}^i \neq \text{null}$  then
7:      $n_{new} \leftarrow \text{CREATENODE}()$ 
8:      $Tr.ADDNODE(n_{new}.s = s_o)$   $\triangleright$  Add as a New Node
9:      $Tr.ADDEDGE((n, n_{new}), (i, \mathcal{T}^i))$ 
10:  end if
11:  if  $g(s_o) == 1$  then  $\triangleright$  Goal Reached
12:     $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D \leftarrow Tr.BACKTRACE(n_{new})$ 
13:    return  $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$ 
14:  end if
15: end while
16:  $n^* \leftarrow \arg \min_{n \in Tr.NODES()} h(n.s)$ 
17:  $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D \leftarrow Tr.BACKTRACE(n^*)$ 
18: return  $\{(i_d, \mathcal{T}^{i_d})\}_{d=1}^D$ 

```

$Tr.NODES()$, 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 each time to actively move, one object $i \in \{1, \dots, N\}$ 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).

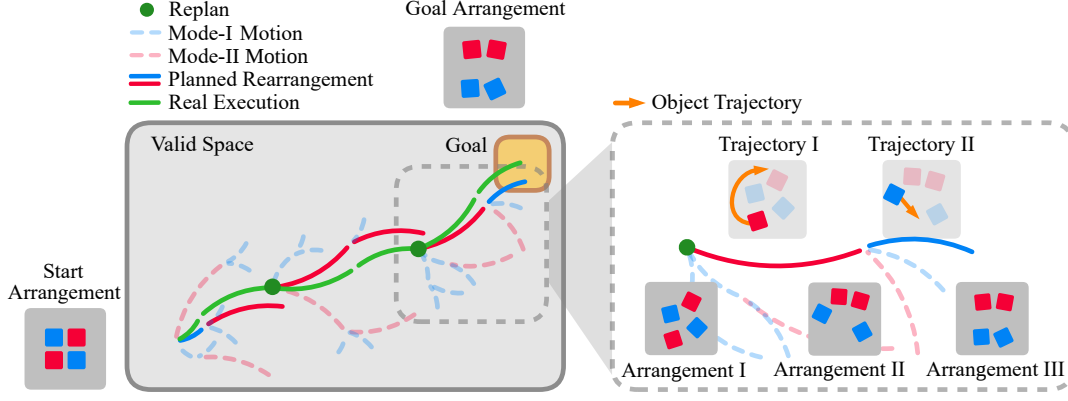


Fig. 4: A schematic plot of OCP. In the left figure, a motion tree is progressively grown from the start arrangement (lower left) towards the goal arrangement (upper right). Each edge of the tree is an explored object motion (i.e., trajectory) generated by one of two exploration modes, *Mode I* (blue 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).

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) [91], 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., $P(n) = w(n) / \sum_{n' \in Tr.Nodes()} w(n')$.

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

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 II. 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. In practice, the gradients of the heuristic function can be either analytically derived or numerically approximated via finite differences applied to each dimension of s ; In our implementation, we use PyTorch's automatic differentiation tool [92] to compute these gradients efficiently. 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. 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 ; $d_{ij}^2 = \|\mathbf{p}^i - \mathbf{p}^j\|^2$ 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

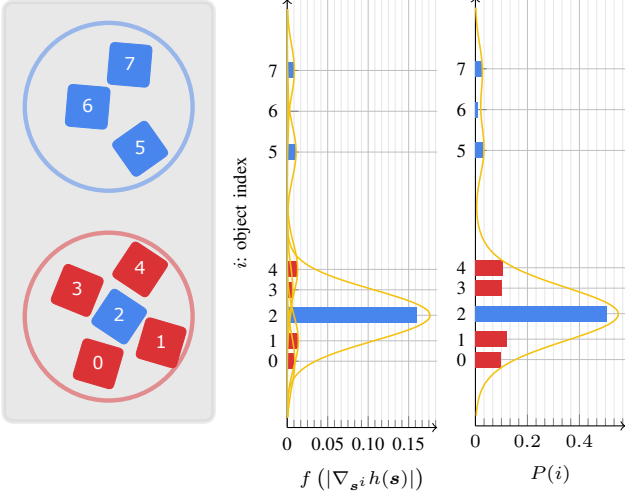


Fig. 5: *Left*: A scenario of the object sorting task. Two classes of cubes (red and blue) need to be relocated inside their corresponding goal regions (circles in the same color as the cubes). At the current state, all cubes except for Cube #2 are already sorted. *Middle*: The value of $f(|\nabla_{s,i} h(s)|)$ 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.

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 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

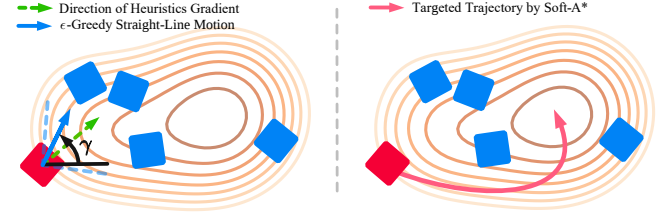


Fig. 6: Two exploration modes of generating the object trajectory for expanding the motion tree. The contour lines represent the heuristic cost concerning only the activated object (red cube). *Left*: With a probability of $1 - \epsilon$, *Mode I* samples a straight-line motion (blue solid arrow) with 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 $[-\frac{\pi}{4}, \frac{\pi}{4}]$ (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.

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 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., $(\frac{\partial h(s)}{\partial x^i}, \frac{\partial h(s)}{\partial y^i})$, 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

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:  $s^i = (x^i, y^i, \theta^i) \leftarrow \text{GETOBJECTSTATE}(s, i)$ 
3: if  $\text{UNIFORM}(0, 1) > p_{A^*}$  or  $i \in n.B$  then ▷ Mode I
4:    $\mathcal{T}^i \leftarrow \text{A Straight Line with Angle } \gamma$ 
5: else ▷ Mode II
6:    $\mathcal{T}^i \leftarrow \text{SOFT-A}^*(s, i)$  ▷ Sec. V-D
7:    $n.B \leftarrow n.B \cup \{i\}$ 
8: end if
9:  $\hat{s}_1^i \leftarrow \text{First Waypoint of } \mathcal{T}^i$ 
10:  $u_0 \leftarrow \text{PUSHSTRATEGY}(s^i, \hat{s}_1^i)$  ▷ Sec. III-B
11: if not  $\text{FEASIBLE}(u_0)$  then
12:   return null
13: end if
14:  $s_o \leftarrow \Pi(s, i, \mathcal{T}^i)$  ▷ Simulate via II
15: return  $s_o, \mathcal{T}^i$ 

```

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 $n.B \in \{1, \dots, N\}$. If a new *Mode II* exploration is going to be performed on an object i that is already explored under *Mode II* (i.e., $i \in n.B$), 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 (i.e., 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.

Note that the feasibility check for pushing actions considers only whether the starting position of a push is occluded by other objects or not. We check only the first u_0 as it requires the robot to reach the position P_0 from above the workspace (as will be described in Sec. VI), where undesired collisions with the top of objects may occur. Subsequent pushing actions are assumed feasible, as they involve only in-plane motions of the robot. Other feasibilities with respect to robot kinematics (e.g., joint limits) are not verified at this stage, since the robot configuration is not yet determined during object-centric planning. While this may lead to discrepancies during actual execution (e.g., the robot may be unable to execute a later action due to kinematic constraints and must skip it or replan, as detailed in Sec. VI), in practice we find that the planner remains effective and reliably completes long-horizon tasks.

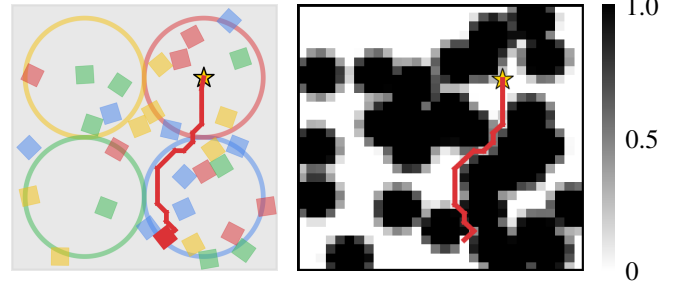


Fig. 7: The grid representation of the workspace (right) for the activated red object (highlighted) in the scene (left). The values of grid cells, ranging from 0 to 1, evaluate the collisions between the activated object and other objects. The path (red lines) planned by the Soft- A^* algorithm on the 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. While our Soft- A^* is fundamentally based on an A^* algorithm with a modified cost function, it differs from traditional A^* and its variants, which seek collision-free paths among static obstacles. In contrast, our Soft- A^* is motivated by the nature of nonprehensile multi-object rearrangement tasks, where object-object interactions are often essential and unavoidable.

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 a grid cell, denoted by Δ , is a hyperparameter used to specify the map's resolution. Each cell of the grid, denoted by $g \in \mathcal{I}^i$, has a coordinate $g.p = (g.x, g.y)$ defined by the position of the cell's center point. Each cell 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 cell value by a linear function in terms of the distance between objects, as expressed in Eq. (6a).

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

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

where d_g in Eq. (6b) is the distance between a cell g and the nearest object (excluding the activated object i) to this cell. $C_{\min}$ 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 cell g) to be unacceptable, and thus set the cell value to 1; when $d_g > C_{\max}$, the activated object is guaranteed not to collide with other objects, and the cell value is set to 0; when d_g decreases from $C_{\max}$ to $C_{\min}$, the cell 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 cells $\{g_1, g_2, \dots, g_*\}$ it traverses. The path starts from the current position and ends up at the optimal cell g_* of the activated object. The optimal cell g_* is the cell that minimizes the heuristic cost $h(\cdot)$ when placing the activated object at this cell. 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 resolution Δ similar to object size works well enough. Thus, the size of the grid 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 cells, and 2) a heuristic function $h_{A^*}(\cdot)$ that underestimates the cost from a cell 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 cell g and one of its adjacent cell g' , the cost is evaluated by Eq. (7):

$$c_{A^*}(g, g') = d(g, g') + \Delta \cdot g'.v$$

$$d(g, g') = \begin{cases} \|g \cdot \mathbf{p} - g' \cdot \mathbf{p}\| & g'.v < 1 \\ \infty & \text{otherwise} \end{cases} \quad (7)$$

The first term $d(g, g')$ is the distance between the two cells g and g' ; the second term $g'.v$ is a cell value that measures collision, which is scaled by the cell 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 cell g to the goal cell g_* , i.e., $h_{A^*}(g) = \|g \cdot \mathbf{p} - g_* \cdot \mathbf{p}\|$. 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 $\{g_1, g_2, \dots, g_*\}$ is finite, we will generate a trajectory for the activated object by extracting the coordinates of each cell throughout the path and assuming the orientation of the object is not changed, i.e., $\mathcal{T}^i = \{(g_1 \cdot \mathbf{p}, \theta^i), (g_2 \cdot \mathbf{p}, \theta^i), \dots, (g_* \cdot \mathbf{p}, \theta^i)\}$. 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 $\mathcal{T}^i = \text{null}$ to abort it.

Algorithm 4 Execute($\cdot$)

Input: The current arrangement $\mathbf{s} \in \mathcal{Q}^O$, the index i and the planned trajectory $\mathcal{T}^i = \{\hat{\mathbf{s}}_k^i\}_{k=1}^K$ of the activated object

- 1: $\mathbf{u}_0 = (\alpha_0, \beta_0) \leftarrow \text{PUSHSTRATEGY}(\mathbf{s}^i, \hat{\mathbf{s}}_1^i)$ ▷ Sec. III-B
- 2: $P_0 \leftarrow \text{STARTINGPOSITION}(\mathbf{s}^i, \mathbf{u}_0)$
- 3: **if** $\text{OCCLUDED}(P_0)$ **then**
- 4: **return**
- 5: **end if**
- 6: $\text{MOVEPUSHERTO}(P_0)$ ▷ Above the Workspace
- 7: **for** $k = 1, \dots, K$ **do**
- 8: **while** $\mathbf{s}^i$ not reaching $\hat{\mathbf{s}}_k^i$ **do**
- 9: $\mathbf{u} \leftarrow \text{PUSHSTRATEGY}(\mathbf{s}^i, \hat{\mathbf{s}}_k^i)$ ▷ Sec. III-B
- 10: $\text{PUSH}(\mathbf{u}, d_{\text{push}})$ ▷ Planar Push by a Distance d_{push}
- 11: $\mathbf{s} \leftarrow \text{OBSERVEOBJECTS}()$ ▷ via Real-time Sensing
- 12: **end while**
- 13: **end for**

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 $\mathcal{T}^{i_d} = \{\hat{\mathbf{s}}_1^{i_d}, \dots, \hat{\mathbf{s}}_k^{i_d}, \dots, \hat{\mathbf{s}}_K^{i_d}\}$ 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 $\hat{\mathbf{s}}_k^{i_d} \in SE(2)$ in the trajectory one by one via a closed-loop pushing strategy (e.g., the UNO Push framework [90]), as outlined in Alg. 4. Specifically, the robot needs to inspect the first pushing action $\mathbf{u}_0$ generated by the pushing strategy. If the starting position $P_0 \in \mathcal{W}$ associated with $\mathbf{u}_0$ 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 $\hat{\mathbf{s}}_k^i$, $k = 1, \dots, K$ 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 $\mathbf{v} \in se(3)$. The Cartesian $\mathbf{v}$ is then projected into the robot's configuration space to generate a robot control $\mathbf{a} \in \mathcal{A}$ for commanding the robot. In this work, we use the robot's joint velocity as control, i.e., $\mathbf{a} = \dot{\mathbf{q}} \in \mathbb{R}^M$, which is generated via null-space projection of the robot Jacobian, as detailed in Eq. (8):

$$\mathbf{a} = \dot{\mathbf{q}} = \mathbf{J}^\dagger \cdot \mathbf{v} + \lambda \cdot (\mathbb{I} - \mathbf{J}^\dagger \mathbf{J}) \dot{\mathbf{q}}_{\text{null}} \quad (8)$$

where $\dot{\mathbf{q}}_{\text{null}} \in \mathbb{R}^M$ 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 $\dot{\mathbf{q}}_{\text{null}}$ by taking the gradient of certain quality measures (e.g., the manipulability and a distance-based cost related to joint limits) [93]; $\dot{\mathbf{q}}_{\text{null}}$ is projected via $(\mathbb{I} - \mathbf{J}^\dagger \mathbf{J})$ 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

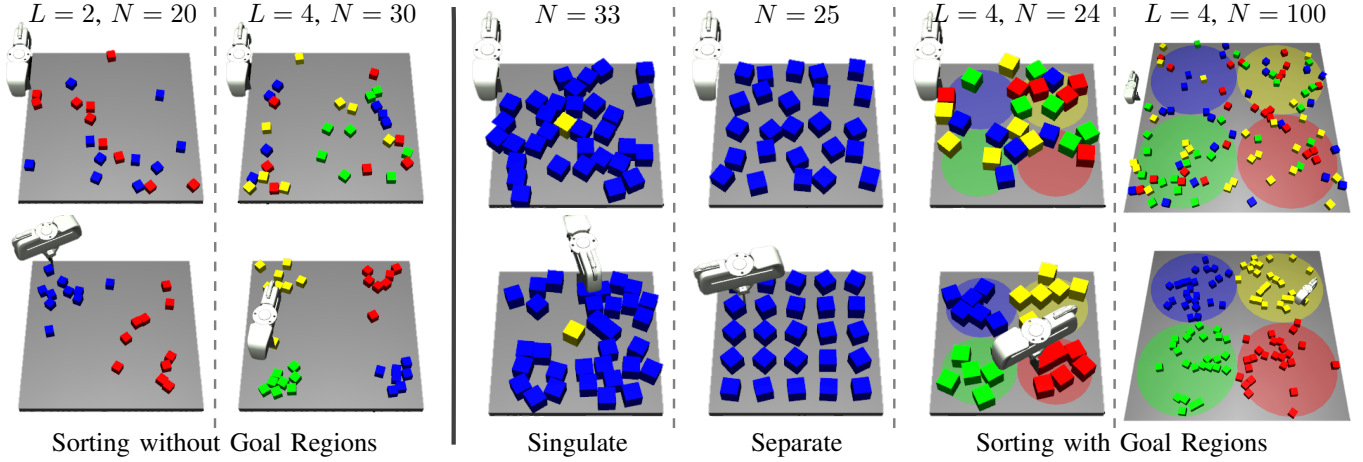


Fig. 8: The evaluated rearrangement tasks solved by our OCP planner in MuJoCo simulation with a floating gripper pusher. The task scenes have different numbers of objects (N) and object classes (L). The top figures show the initial configuration of randomly placed objects, and the bottom figures show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

the 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 $\sqrt{\det(JJ^T)}$. 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 in both simulation and on a physical robot manipulator to evaluate the performance of our proposed object-centric planner (OCP) against selected baselines. Our evaluation focuses on two key aspects: 1) Planning efficiency, measured by the average planning time for task completion (lower is better); and 2) Action effectiveness, assessed by the average number of actions and average execution time (for real-world experiments) to complete a task. Our planner was implemented with the Box2D physics engine¹ to approximate the object-centric transition function Π defined in Sec. IV. Unless otherwise specified, all evaluations were run in Python with a single thread on a 3.4 GHz AMD Ryzen 9 5950X CPU. For simulation evaluation, we used a floating gripper in the MuJoCo [94] environment to execute actions generated by the planner, as shown in Fig. 8. The floating gripper is allowed to freely navigate parallel to the workspace, to interact with objects using an attached fence pusher. The gripper can also teleport to switch to manipulate a different object. For real-world experiments, we evaluated our planner on a 7-DoF Franka Emika Panda robot manipulator.

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

A. Planar Pushing-based Rearrangement Tasks

We selected challenging rearrangement tasks from two state-of-the-art works [22], [30] for evaluation, by considering the variety and large scale of entailed tasks. For example, some tasks (e.g., singulation) require only local rearrangement of a target object, whereas others (e.g., sorting) involve global rearrangement and relative reconfiguration of all objects. All evaluated tasks are visualized in Fig. 8. We generally categorize all tasks into two types, based on how the task goals are defined differently:

1) *Tasks without Explicit Goal Definition*: Such tasks do not require an explicitly defined goal pose or region for each object. Instead, as long as the relative reconfiguration of object states meets certain requirements, the task is considered successful. One example of such tasks is *Object Sorting without Goal Regions* investigated in [30]. As shown in Fig. 8 (left), the robot is tasked to separate objects of L different classes into clusters. No specific goal pose or region is assigned to any object. By observing the object arrangement $s \in \mathcal{Q}^O$, a convex hull $\text{CH}_j(s) \subset \mathbb{R}^2$ is computed for each class $j \in \{1, \dots, L\}$ to enclose objects of that class (i.e., covering geometric shapes of all objects belonging to class j). The task is considered successful if the minimum distance between any pair of convex hulls is larger than a threshold ϵ_d . Formally, the goal criterion is satisfied (i.e., $g(s) = 1$) when $\forall i, j \in \{1, \dots, L\}$:

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

To solve this task, we equip our object-centric planner with a heuristic function $h(\cdot)$ similar to the reward function in [30]. A lower heuristic cost will be expected when intra-class objects cluster more tightly and inter-class objects are more separated.

2) *Tasks with Explicit Goal Definition*: This type of task requires an explicit goal region $\mathcal{G}_i \subset \mathcal{W}$ for each movable object $i \in \{1, \dots, N\}$, with its centroid position denoted by $p^{\mathcal{G}_i} = (x^{\mathcal{G}_i}, y^{\mathcal{G}_i})$. In different tasks, the goal regions of some objects can be either distinct or overlapping. The task

is considered successful when each object is relocated within its designated goal region, formally defined as:

$$\forall i \in \{1, \dots, N\} : \mathbf{p}^i = (x^i, y^i) \in \mathcal{G}_i \quad (10)$$

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

$$h(\mathbf{s}) = \sum_{i \in \{1, \dots, N\}} \mathbb{1}\{\mathbf{p}^i \notin \mathcal{G}_i\} \frac{\|\mathbf{p}^i - \mathbf{p}^{\mathcal{G}_i}\|^2}{r_{\mathcal{G}_i}^2} \quad (11)$$

where $r_{\mathcal{G}_i}$ denotes the size of the goal region $\mathcal{G}^i$ (e.g., a radius if $\mathcal{G}_i$ is a circle); $\mathbb{1}\{\mathbf{p}^i \notin \mathcal{G}_i\}$ is an indicator function that excludes objects already within their goal regions. When using heuristic gradients to guide exploration (in Sec. V-B), this indicator function helps our planner focus on exploring unsolved objects (i.e., objects not at their goals) without wasting time on already solved objects.

The tasks from [22] can all be categorized into this type:

- a) *Singulate*: This task requires a target object $o \in \{1, \dots, N\}$ to be isolated from the 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 are moved near one of the four corners.
- b) *Separate*: The workspace is divided into multiple grid cells, each containing a tiny circular goal region. Each object must be placed inside a distinct goal region. Since objects are not unique, the assignment of 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, let $\mathcal{D} \subset \{1, \dots, N\}$ denote the set of non-unique objects, and let $\mathcal{SG} = \{\mathcal{G}^i : i \in \mathcal{D}\}$ denote the set of distinct goal regions for these objects. Note that $|\mathcal{D}| = |\mathcal{SG}|$. An assignment function $A : \mathcal{D} \mapsto \mathcal{SG}$ bijectively assigns each object $i \in \mathcal{D}$ to its corresponding goal region $A(i) \in \mathcal{SG}$. This assignment is dynamically determined by minimizing the summed distances between objects and their assigned goal region, i.e., $\sum_{i \in \mathcal{D}} \text{dist}(\mathbf{p}^i, A(i))$.

B. Selected Baselines for Evaluation

As extensively discussed in Sec. I and Sec. II, many relevant approaches are proposed to address planar nonprehensile rearrangement, including sampling-based [16], [17] and end-to-end learning-based methods [29], as well as robot-centric [34] and object-centric approaches [21], [23]. However, many of these methods are designed for specific scenarios

(e.g., confined spaces) or smaller-scale problems (e.g., pregrasp manipulation for object retrieval [36]), making them less adaptable to larger-scale tasks with diverse objectives as described in Sec. VII-A. Among them, we select three relevant baselines that align with our task setups and can be reasonably re-implemented for evaluation. For fair comparison, we re-implemented each baseline and adapted its implementation to meet our task requirements, as detailed below:

- **Hybrid-RRT (H-RRT)** [18]: A sampling-based approach based on a kinodynamic RRT, where actions are hybridly sampled between robot-centric actions and object-centric pushing primitives. For convenience, we refer to this method as Hybrid-RRT (abbreviated as H-RRT) throughout the paper. To improve sampling efficiency for large-scale tasks, we introduce a goal bias to RRT with a probability of 0.2 for sampling a goal state for tree expansion, if goal configurations are explicitly given by the task. In addition, we made the following key adaptations: 1) The original planner is fully open-loop and prone to failure in long-horizon tasks due to modeling inaccuracies. To address this, we adapted the implementation to perform limited-horizon planning, execute the best candidate node, sense the real-world state, and then replan iteratively. 2) The original planner assumes a single target object. To extend it to handle multiple target objects, when sampling an object-centric action, we randomly sample one object that is not at its goal and treat it as the current target. If no goal is specified, we sample a random object to push. 3) For fair comparison, we use the same Box2D physics engine for dynamics propagation.

- **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. The original implementation also uses Box2D. We made the following key adaptations: 1) Since the algorithm is open-loop, we modify it to execute each locally searched trajectory, followed by sensing and replanning to close the loop. 2) The original ILS assumes goal regions are given and uses them to guide object and (greedy) action sampling. For tasks without explicit goal regions, we instead randomly sample an object, and action sampling becomes fully random. 3) For tasks without explicit goal regions, the original distance-based cost is no longer applicable; in such cases, we replace it with the same heuristic cost used in our planner.

- **kdRRF** [35]: 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.

In addition to the re-implemented baselines, we also include a comparison with the method from [30] when evaluating on the same tasks used in their study. Their approach uses a Monte Carlo Tree Search (MCTS)-based planner guided by a learned rollout policy (i.e., a deep neural network trained offline using image data). We found that this method requires substantial dataset collection and retraining to adapt to new rearrangement tasks or setups. For this reason, we did not re-implement it for new tasks and instead report their original

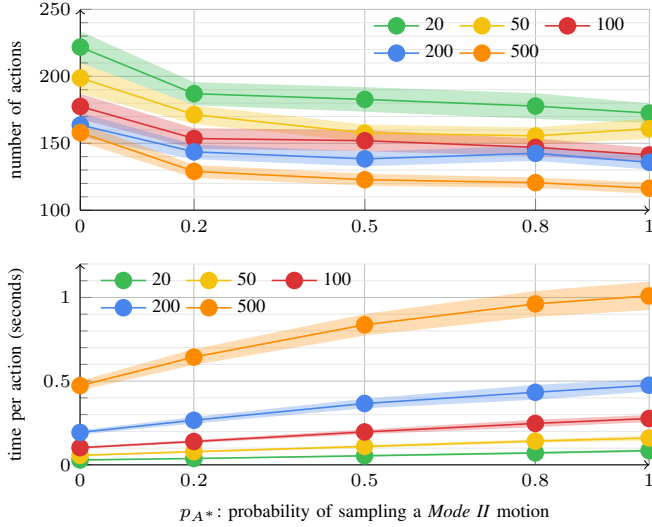


Fig. 9: The statistics of the number of actions (top plot) and the average planning time per action (bottom plot) in terms of different parameter choices of $S_{\max}$ (plots in different colors) and p_{A^*} (x-axis). The shaded areas are the quarter standard deviations for the top plot; and the standard deviations for the bottom plot.

results as a reference. As will be shown by the experiments in Sec. VII-D and VII-E, even compared to their original parallelized implementation and fine-tuned policy with data, our planner achieves comparable success on similar tasks, without requiring any data collection or retraining.

C. Analysis of Parameter Selection and Ablation Study

We first conducted simulation experiments to analyze how the performance of our OCP planner is affected by different choices of key parameters: a) the maximum tree size $S_{\max}$, and b) the probability p_{A^*} of selecting *Mode II* (i.e., Soft- A^*) for exploration. Other parameters, such as the pushing distance d_{push} , the length range of a sampled *Mode I* motion $[l_{\min}, l_{\max})$, and the grid resolution Δ of Soft- A^* can be selected more intuitively relative to the object and workspace sizes and do not require extensive fine-tuning for functionality. We chose the representative task, *Sorting with Goal Regions*, to evaluate all parameter settings. In this task, 32 objects belonging to 4 classes (8 per class) must be sorted, following a similar setup to Fig. 8. The workspace was a $0.6m \times 0.6m$ square, with circular goal regions (for the 4 classes) centered at $(\pm 0.135, \pm 0.135)m$, each with a radius of $0.135m$. For each parameter setting, we ran 100 trials with the same initial object configuration to collect performance statistics.

The results, shown in Fig. 9, illustrate the effects of different parameter choices. Specifically, as the tree size increases with $S_{\max} = 20, 50, 100, 200$, and 500 (plotted in different colors), the planning time per action increases, while the number of actions required to complete the task decreases. This trend is intuitive: A larger $S_{\max}$ means the planner explores more object motions (i.e., more tree nodes) before selecting an action, increasing computational effort per action; at the same time, when $S_{\max}$ increases, the planner explores the problem space more extensively with more tree branches,

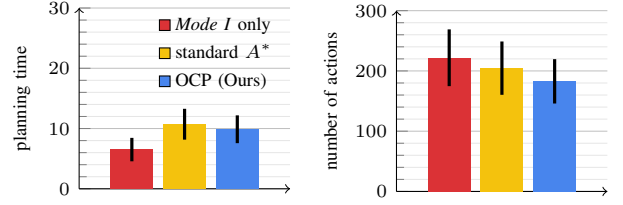


Fig. 10: The statistics of planning time (left) and number of actions (right) for different variants of our planner in the ablation study: only with *Mode I* motions by setting $p_{A^*} = 0$ (red); our planner with Soft- A^* replaced by a standard A^* (yellow); our planner (blue).

which increases the probability of finding more optimal actions and results in a smaller number of actions needed for task completion. When varying the probability $p_{A^*} = 0, 0.2, 0.5, 0.8$, and 1.0 , we observed that a higher p_{A^*} led to a longer planning time required for each action. This is because a higher p_{A^*} triggers more attempts of a *Mode II* exploration, which requires an extra computational time of A^* search. However, higher p_{A^*} also results in fewer needed actions. This is because *Mode II* motions are generally more effective than *Mode I* motions in reducing the heuristic cost and progressing towards the task goal.

Overall, 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 shorter planning time per action, but at the cost of the optimality of generated actions.

Furthermore, with a tree size limit $S_{\max} = 50$, we conducted an ablation study on the same sorting task to evaluate the importance of both exploration modes in our planner. Specifically, we compared the performance of our full OCP planner against three variants: 1) *Mode II* (Soft- A^*) only; 2) *Mode I* only (by setting $p_{A^*} = 0$); 3) using standard A^* (i.e., only allowing strictly collision-free paths) in *Mode II*. We observed that the first variant (i.e., *Mode II* only) was highly ineffective in cluttered scenarios (i.e., zero success rate), as the algorithm often got stuck in states where no solutions to the deterministic Soft- A^* are feasible for any object. In such cases, it failed to make progress and lacked a mechanism to escape such dead ends, highlighting the necessity of *Mode I* exploration to recover. For the other two variants, we reported the performance in Fig. 10. As can be seen from the results, compared to using *Mode I* alone, our planner incurred slightly higher planning time due to the computational cost of Soft- A^* in *Mode II*, but produced more effective behavior with fewer actions; compared to using standard A^* , our planner generated more optimal actions with lower planning time, highlighting the benefits of incorporating our Soft- A^* . Overall, the results show the importance of combining both exploration modes for effective and robust planning.

D. Comparative Evaluations in Simulation

First, on the *Sorting without Goal Regions* task, we compared our OCP planner against all three re-implemented base-

Scene # objects # classes		Metric	MCTS [30]*	H-RRT [18]	ILS [22]	kdRRF [35]	OCP (Ours)
$N = 20$	$L = 2$	Success Rate	100 %	95 %	96 %	99 %	100 %
		Planning Time (seconds)	–	15.5 ± 5.2	5.9 ± 2.5	16.1 ± 4.7	8.5 ± 3.1
		Num. Actions	36.1 ± 1.3	132.3 ± 44.2	195.1 ± 55.0	30.8 ± 8.8	28.7 ± 9.1
	$L = 3$	Success Rate	98 %	47 %	83 %	87 %	100 %
		Planning Time (seconds)	–	32.7 ± 7.6	16.8 ± 6.8	31.3 ± 7.1	11.9 ± 3.7
		Num. Actions	66.6 ± 2.3	279.2 ± 65.0	364.9 ± 98.7	56.7 ± 12.8	41.4 ± 13.6
	$L = 4$	Success Rate	97 %	4 %	66 %	30 %	100 %
		Planning Time (seconds)	–	43.7 ± 3.9	38.9 ± 12.6	45.5 ± 7.3	20.4 ± 8.3
		Num. Actions	80.1 ± 2.3	376.8 ± 127.8	623.6 ± 151.6	77.0 ± 12.5	63.9 ± 24.0
	$L = 2$	Success Rate	96 %	98 %	88 %	98 %	98 %
		Planning Time (seconds)	–	47.3 ± 13.7	19.7 ± 13.3	41.8 ± 11.7	33.9 ± 14.7
		Num. Actions	77.5 ± 3.6	279.2 ± 79.9	376.7 ± 96.4	56.8 ± 15.2	69.0 ± 24.2
$N = 30$	$L = 3$	Success Rate	91 %	11 %	62 %	83 %	97 %
		Planning Time (seconds)	–	96.6 ± 19.4	63.2 ± 26.7	81.1 ± 19.0	52.8 ± 21.8
		Num. Actions	131.5 ± 5.1	573.9 ± 118.4	797.9 ± 189.6	107.9 ± 24.8	98.8 ± 31.1
	$L = 4$	Success Rate	89 %	0 %	2 %	10 %	87 %
		Planning Time (seconds)	–	> 180	89.6 ± 17.6	136.8 ± 34.7	91.2 ± 28.6
		Num. Actions	162.6 ± 4.6	> 1055	1208.5 ± 64.5	174.3 ± 44.4	155.0 ± 36.5

* The results for MCTS are reported as originally published in [30] without re-implementation.

Fig. 11: The success rate, planning time, and the average number of actions for different methods evaluated on the *Sorting without Goal Regions* task, with different numbers of objects (N) and classes (L) in simulation.

lines in Sec. VII-B. We used the same setup as in [30]: 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). For all evaluated planners including ours, we ran experiments with 100 trials on each scene, where the objects were randomly placed at the beginning of each trial. We set different planning time budgets based on the difficulty of each scene: for scenes with $N = 20$, the time limits are 30, 45, and 60 seconds for $L = 2, 3$, and 4, respectively; for scenes with $N = 30$, the limits are 90, 120, and 180 seconds. A trial is considered a failure if no solution is found within the allotted time of planning. We report the success rate, planning time, and average number of actions in Fig. 11. As can be seen from the results, benefiting from our heuristics-guided and more task-relevant motion exploration, our object-centric planner consistently outperforms all re-implemented baselines across most metrics, particularly as task complexity increases. As the number of classes and objects grows, the success rates of all baselines drop significantly. For instance, without explicit goal-guided action sampling, ILS failed in nearly all trials at $L = 4$, $N = 4$, and H-RRT struggled in every scene with $L > 2$, while our planner remained consistently successful.

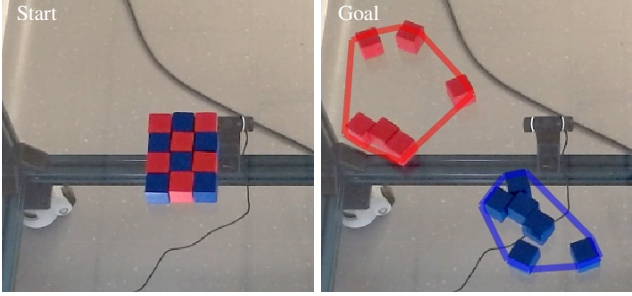
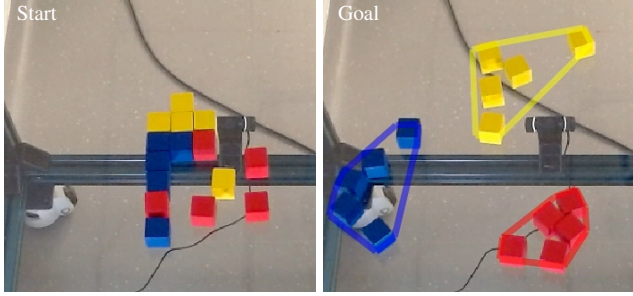
Note that in Fig. 11, although we did not re-implement MCTS, we still reported its performance (specifically, success rate and number of actions) using the original results from [30] as a reference. The original MCTS implementation employed a specially designed three-finger robot pusher, which enables the pusher to simultaneously contact multiple objects and concurrently manipulate them by sweeping-like actions. However, even with a learned policy trained on a large dataset, MCTS generally requires more actions than our planner to complete the task, whereas our planner does not need any data for training. In addition, despite using a parallelized implementation with 8 threads on an Intel i7-7820X CPU, MCTS had an average planning time of $2.16s$ per action, while

(a) Singulate ($N = 33$)				
Metric	H-RRT [18]	ILS [22]	kdRRF [35]	OCP (Ours)
S. R.	10 %	98 %	71 %	97 %
P. T.	10.3 ± 5.8	5.9 ± 4.1	8.0 ± 5.4	2.4 ± 3.2
N. A.	109.1 ± 61.7	109.6 ± 72.5	128.9 ± 85.5	26.5 ± 14.4
(b) Separate ($N = 25$)				
Metric	H-RRT [18]	ILS [22]	kdRRF [35]	OCP (Ours)
S. R.	0 %	93 %	0 %	98 %
P. T.	> 20.0	13.8 ± 2.3	> 20.0	4.5 ± 4.2
N. A.	> 268	200.6 ± 27.5	> 330	72.4 ± 47.0
(c) Sorting with Goal Regions ($L = 4, N = 24$)				
Metric	H-RRT [18]	ILS [22]	kdRRF [35]	OCP (Ours)
S. R.	45 %	99 %	99 %	100 %
P. T.	21.7 ± 6.0	13.4 ± 4.6	13.5 ± 4.7	4.3 ± 1.5
N. A.	304.2 ± 82.6	402.9 ± 134.7	304.5 ± 105.7	117.5 ± 26.1
(d) Sorting with Goal Regions ($L = 4, N = 100$)				
Metric	H-RRT [18]	ILS [22]	kdRRF [35]	OCP (Ours)
S. R.	96 %	100 %	93 %	100 %
P. T.	126.6 ± 23.1	142.0 ± 25.3	126.2 ± 22.4	100.8 ± 40.5
N. A.	464.3 ± 83.8	1017.2 ± 174.2	1091.4 ± 185.4	374.1 ± 94.4

Fig. 12: Simulation evaluations of different methods on *Singulate*, *Separate*, and *Sorting with Goal Regions* ($N = 24$ and 100) tasks. The metrics “S. R.”, “P. T.”, and “N. A.” stand for Success Rate, Planning Time (seconds), and Number of Actions, respectively.

our planner only required an average of $0.5s$ per action on a single thread.

Next, on the other three tasks, *Singulate*, *Separate*, and *Sorting with Goal Regions* (including two scenes with $N = 24$ and 100 objects of 4 classes), we compared our OCP planner against all three re-implemented baselines. For each evaluated task and scene, we followed the same setup and time budget as in [22] for a fair comparison: The workspace was a square region of size $40cm$ (expanded to $125cm$ for the $N = 100$ sorting task), and all objects were cubes of size $4cm$; the time budget was set to $20s$ for *Singulate* and *Separate* tasks, and $30s$ and $600s$ for *Sorting with Goal Regions* with $N = 24$ and $N = 100$ respectively. Also, the sizes of goal regions were also kept the same as in [22]. We conducted 100 trials

Scene I ($L = 2, N = 12$)		Scene II ($L = 3, N = 15$)				
						
Scene	Metric	MCTS [30]*	H-RRT [18]	ILS [22]	kdRRF [35]	OCP (Ours)
Scene I ($L = 2, N = 12$)	Success Rate	16 / 20	6 / 10	9 / 10	10 / 10	10 / 10
	Execution Time (minutes)	–	7.6 ± 1.0	6.3 ± 1.9	6.2 ± 1.9	4.0 ± 0.5
	Num. Actions	37.0 ± 3.5	86.5 ± 31.3	62.3 ± 20.4	23.8 ± 8.1	19.1 ± 2.9
	Planning Time (seconds)	–	19.4 ± 7.8	6.4 ± 4.0	16.5 ± 5.3	9.5 ± 1.9
	Plan. Time / Action (seconds)	–	0.22	0.15	0.69	0.50
Scene II ($L = 3, N = 15$)	Success Rate	15 / 20	0 / 10	4 / 10	5 / 10	10 / 10
	Execution Time (minutes)	–	> 10.0	8.1 ± 2.2	6.9 ± 1.7	3.5 ± 0.6
	Num. Actions	27.7 ± 2.1	> 96	82.5 ± 22.4	28.2 ± 8.8	20.5 ± 4.3
	Planning Time (seconds)	–	> 30.1	20.4 ± 19.2	27.3 ± 9.5	9.3 ± 2.7
	Plan. Time / Action (seconds)	–	0.31	0.21	0.97	0.46

* The results for MCTS are reported as originally published in [30] without re-implementation.

Fig. 13: Real-world evaluations of different methods on the *Sorting without Goal Regions* task under two different settings.

for each evaluated planner on each task scene, and summarized the relevant metrics in Fig. 12. From the results reported in Fig. 12, H-RRT and kdRRF baselines performed poorly on the *Separate* task. This is because, unlike our method which generates real-time actions, the baselines, although equipped with replanning, did not close the loop tightly enough, making them ineffective at precisely pushing objects into small-sized goal regions when discrepancies between modeling and actual execution persist. More importantly, across all tasks, our OCP planner achieved comparable success rate to ILS but with significantly lower planning time and fewer actions, regardless of the task setup or the number of objects. This highlights a clear improvement in planning efficiency enabled by the object-centric action generation strategy of our planner.

E. Comparative Evaluations on a Physical Robot

To more realistically challenge our OCP planner under real-world uncertainties and explore its deployment potential, we conducted real-world experiments on a physical 7-DoF Franka Emika Panda robot platform, as displayed in Fig. 14. We 3D printed a pusher to replace one finger of the robot gripper. The workspace plane was made of a transparent panel and the objects were tracked via AprilTags [95] by two cameras positioned beneath the workspace.

First, we evaluated the performance of our planner on tasks without explicit goals. We used a similar task setup as the physical experiments in [30]. Specifically, the experiments were conducted on the *Sorting without Goal Regions* task, using the same 1-inch wooden cubes as objects to manipulate. The task scenes used for evaluation were also the same as [30]: 1) 2 classes $\times$ 6 objects, and 2) 3 classes $\times$ 5 objects. Since the hardware robot we used is different from [30] which had different reachability relative to the workspace, we could not use the exact same workspace as [30]. However, we ensured

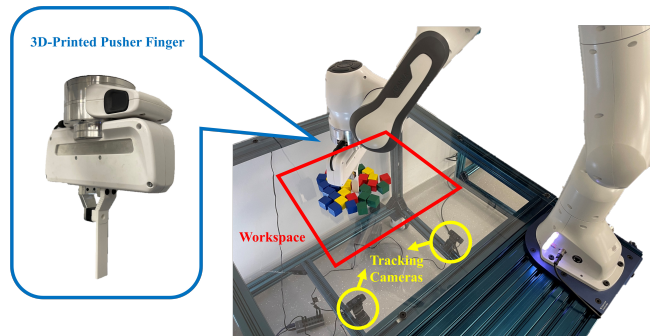


Fig. 14: The experiment setup of the physical robot platform. The objects were tracked by two cameras below the transparent workspace.

that our workspace (a rectangle of size $33\text{cm} \times 30\text{cm}$) had approximately the same area as the one used in [30]. In addition, for a more extensive comparison, we re-implemented all the three baselines [18], [22], [34] on the real robot. Each re-implemented baseline, along with our OCP planner, was evaluated over 10 trials per task scene, using the same initial object configuration as in [30], as shown in Fig. 13. For both scenes, the time budget for robot execution was set to 10 minutes, i.e., a task was considered a failure if the robot did not finish the task within 10 minutes. The results for relevant metrics are reported in the table of Fig. 13, where the results for MCTS are directly taken from [30] (only the success rate and number of actions are reported). In the results, our OCP planner always succeeded within 5 minutes while other baselines could exceed the 10-minute time budget, particularly in Scene II which involved more objects and classes. Additionally, our OCP planner outperformed all baselines regarding the number of actions needed for task completion. This shows that, in real-world settings, our planner is still consistently able to plan efficiently and generate more effective actions. In terms

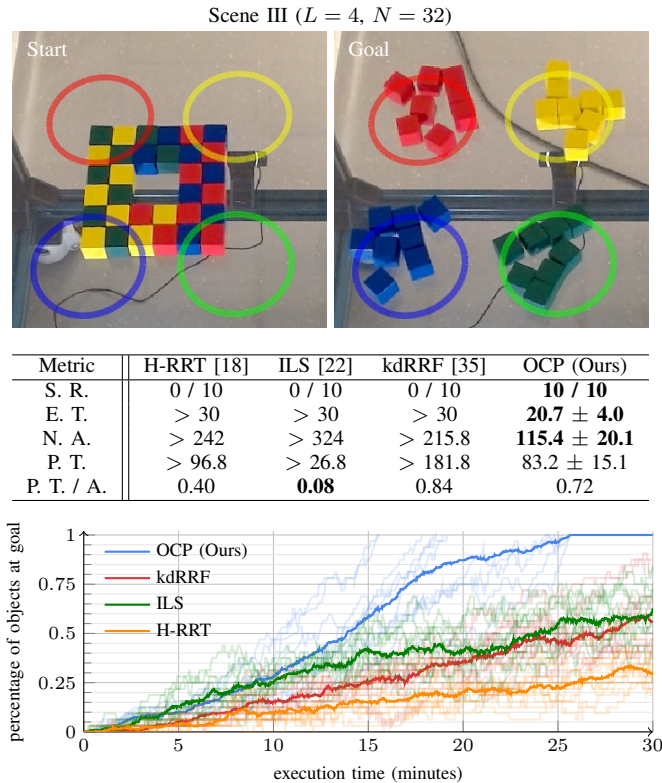


Fig. 15: Real-world evaluations of different methods on the *Sorting with Goal Regions* task for 4 classes $\times$ 8 objects, where the metrics “S. R.”, “E. T.”, “N. A.”, “P. T.”, and “P. T. / A.” stand for Success Rate, Execution Time (minutes), Number of Actions, Planning Time (seconds), and Planning Time per Action (seconds), respectively. The bottom figure plots the percentage of objects at their goals throughout the execution of the kdRRF (red), ILS (green), and H-RRT (orange) baselines 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.

of planning time, ILS appeared particularly fast at generating a single action, as it requires only a small number of rollouts in each planning cycle without extensive search or optimization. However, in real-time scenarios, we did not observe substantial differences since all evaluated planners are sufficiently reactive with low per-action planning time. Execution time and number of actions are usually more representative metrics for assessing practical performance.

Then, we compared our planner against all re-implemented baselines on a task with explicit goal regions, *Sorting with Goal Regions* as used in [22]. The task involves sorting $N = 32$ cubes (i.e., 4 classes $\times$ 8 cubes). We followed the same experimental setup as [22], including cube size, circular goal region size, and a 30-minute time budget for execution. Although our workspace dimension ($394mm \times 330mm$) differed from [22] due to different experiment space and hardware, 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 10 trials for all evaluated planners, with the same adversarially designed initial object configuration from [22]. Results are summarized in Fig. 15. From the results, it is clear that our OCP planner consistently enabled the robot with faster task



Fig. 16: *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.

completion than all baselines, by generating more effective actions in real time. In Fig. 15 (bottom), for every trial of each evaluated planner, we plotted the percentage of objects inside their goal regions throughout the execution. While all three baselines made steady progress as the execution proceeded, it never succeeded within the 30-minute budget. In contrast, our planner always successfully sorted all objects across all trials within roughly 25 minutes of execution.

F. Real-world Qualitative Demonstration

All the aforementioned quantitative evaluations were conducted using cube-shaped objects to ensure fair comparisons with existing baselines. However, our object-centric planner OCP is not limited to cubes and can generalize to manipulate arbitrary object shapes under real-world settings. As shown in Fig. 16 (top), our planner successfully rearranges non-convex objects shaped like “T”, “R”, and “O” to draw the corresponding letters formed by their final configurations. Moreover, in Fig. 16 (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 nonprehensile rearrangement as a combination of global manipulation of object clusters and local relocation of individual objects. To facilitate future research in this area, we propose a standardized real-world benchmarking

protocol to featuring rearrangement tasks that require both intra-class global rearrangement and inter-class relocation. We also include evaluated performance metrics of our OCP planner as a reference baseline for the benchmark. All tasks in the benchmark use 1-inch wooden cubes (Item #65 in the YCB dataset [96]) as the objects to be rearranged. The following metrics are used to evaluate relevant methods:

- 1) *Success Rate*: Percentage of successful trials completed within the given execution time budget.
- 2) *Execution Time*: Average time of the entire robot execution for successful trials.
- 3) *Number of Actions*: Average number of robot actions needed to complete the task successfully.
- 4) *Planning Time*: Average total planning time for successful trials, excluding robot execution time.
- 5) *Planning Time per Action*: Average runtime the planner takes to generate a valid robot action.

The benchmark includes five scenes, each designed with a different packing factor (i.e., the ratio of the total object area to the workspace area) to reflect varying levels of difficulty. Each scene also has a virtual out-of-bounds region with a thickness of 2cm . The first three scenes, Scene I (Fig. 13 left), II (Fig. 13 right), and III (Fig. 15), corresponds to those used in the real-world evaluations presented in Sec. VII-E. In addition, we introduce another two custom real-world benchmark scenarios, Scene IV and V, as shown in Fig. 17: In both Scene IV and Scene V, 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; their goal regions are interchangeable and dynamically assigned by minimizing the summed distance between objects and their assigned goals. Each scene uses a $30\text{cm} \times 30\text{cm}$ square workspace, and all goal regions are circular with a radius of 2cm . We adversarially designed the initial object configurations for both scenes to make them sufficiently challenging. We ran our OCP planner 10 times on each scene using execution time budgets of 10 minutes for Scene IV and 15 minutes for Scene V. The statistics of relevant metrics are reported in Fig. 17. All five benchmark scenes are summarized below; for Scene I, II, and III, the initial and final object configurations, along with associated metrics evaluated using our planner can be found in the referred figures.

- 1) Scene I (packing factor: 0.10): Sorting 2 classes $\times$ 6 objects without explicit goal definitions, as shown in Fig. 13 (left).
- 2) Scene II (packing factor: 0.13): Sorting 3 classes $\times$ 5 objects without explicit goal definitions, as shown in Fig. 13 (right).
- 3) Scene III (packing factor: 0.20): Sorting 4 classes $\times$ 8 objects with explicit goal definitions, as shown in Fig. 15.
- 4) Scene IV (packing factor: 0.095): Rearranging 2 classes $\times$ 5 objects ($N = 10$). The centers of goal regions are at $(9, 9)\text{cm}$, $(3, 9)\text{cm}$, $(-3, 9)\text{cm}$, $(9, 3)\text{cm}$, and $(9, -3)\text{cm}$ for the first class (blue); and are at $(-9, -9)\text{cm}$, $(-9, -3)\text{cm}$, $(-9, 3)\text{cm}$, $(-3, -9)\text{cm}$, $(3, -9)\text{cm}$ for the second class (red), as shown in Fig. 17 (top).

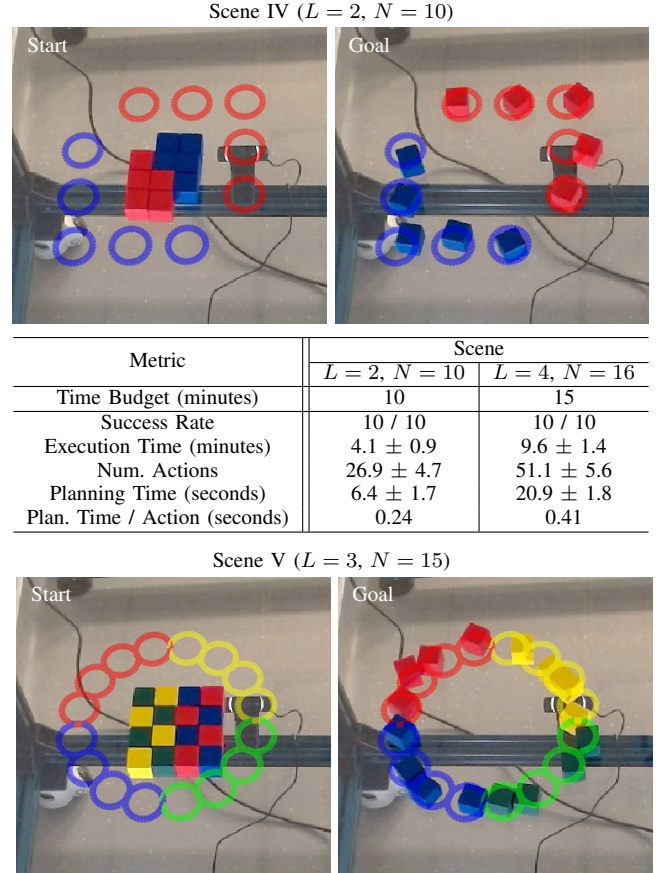


Fig. 17: Initial and final configurations of the customized Scene IV (top) and Scene V (bottom) in our proposed benchmark. The relevant metrics evaluated with our proposed object-centric planner OCP on both scenes are given in the table (middle).

- 5) Scene V (packing factor: 0.15): Rearranging 4 classes $\times$ 4 objects ($N = 16$). The centers of goal regions are evenly spaced along a circle of radius 10cm , with adjacent goal regions belonging to the same class, as shown in Fig. 17 (bottom).

IX. CONCLUSION

In this work, we proposed a unified framework for non-prehensile rearrangement of multiple movable objects, by adopting an *object-centric* planning paradigm: it 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. The proposed framework is built on sampling-based kinodynamic planning and is equipped with two complementary exploration modes that alternate during planning. This design enables both goal-oriented and broad exploration over the problem space, allowing the robot to more efficiently find desired object arrangements achievable by the robot execution. Furthermore, by interleaving planning and execution, the planner can react in real time to perception and modeling uncertainties, improving robustness in real-world scenarios. As a result, our approach can address a range of large-scale, long-horizon, and physics-intensive rearrangement tasks.

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] S. Talia, A. Thareja, C. Mavrogiannis, M. Schmittle, and S. S. Srinivasa, "Pushr: A multirobot system for nonprehensile rearrangement," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 5380–5387.
- [8] 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.
- [9] 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.
- [10] M. Nieuwenhuisen, D. Droeschel, D. Holz, J. Stückler, A. Berner, J. Li, R. Klein, and S. Behnke, "Mobile bin picking with an anthropomorphic service robot," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 2327–2334.
- [11] 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.
- [12] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of the fourth annual symposium on Computational geometry*, 1988, pp. 279–288.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] S. Sundaram, I. Remmler, and N. M. Amato, "Disassembly sequencing using a motion planning approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2. IEEE, 2001, pp. 1475–1480.
- [20] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, and K. Bekris, "Rearranging similar objects with a manipulator using pebble graphs," in *IEEE International Conference on Humanoid Robots (HUMANOIDS)*. IEEE, 2014, pp. 1081–1087.
- [21] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Springer, 2013, pp. 531–545.
- [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] D. M. Saxena and M. Likhachev, "Planning for complex non-prehensile manipulation among movable objects by interleaving multi-agent pathfinding and physics-based simulation," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 8141–8147.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.

- [35] K. Ren, P. Chanrungrameekul, 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.
- [36] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa, “Physics-based grasp planning through clutter,” *Robotics: Science and Systems*, p. 57, 2012.
- [37] 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.
- [38] 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.
- [39] 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.
- [40] M. T. Mason, “Progress in nonprehensile manipulation,” *The International Journal of Robotics Research*, vol. 18, no. 11, pp. 1129–1141, 1999.
- [41] N. Dengler, D. Großklau, 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.
- [42] 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.
- [43] 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.
- [44] X. Yi and N. Fazeli, “Precise object sliding with top contact via asymmetric dual limit surfaces,” 2023.
- [45] 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.
- [46] 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.
- [47] 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.
- [48] 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.
- [49] 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. 10 584–10 590.
- [50] 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.
- [51] 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.
- [52] 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.
- [53] 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.
- [54] 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.
- [55] 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.
- [56] F. Bertonecelli, 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. 11 032–11 038.
- [57] 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.
- [58] 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.
- [59] 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.
- [60] 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.
- [61] 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.
- [62] 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.
- [63] 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.
- [64] 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.
- [65] 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.
- [66] 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.
- [67] 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.
- [68] 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.
- [69] A. M. Johnson, J. E. King, and S. Srinivasa, “Convergent planning,” *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1044–1051, 2016.
- [70] 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.
- [71] 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.
- [72] 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.
- [73] C. Gan, S. Zhou, J. Schwartz, S. Alter, A. Bhandwadar, 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.
- [74] S. D. Han, N. M. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, “Complexity results and fast methods for optimal tabletop rearrangement with overhead grasps,” *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1775–1795, 2018.
- [75] 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.
- [76] 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.
- [77] 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.
- [78] 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.
 - [79] 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.
 - [80] 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.
 - [81] 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.
 - [82] 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.
 - [83] 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.
 - [84] 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.
 - [85] 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.
 - [86] 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.
 - [87] B. Tang and G. S. Sukhatme, “Selective object rearrangement in clutter,” in *Conference on Robot Learning*. PMLR, 2023, pp. 1001–1010.
 - [88] 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.
 - [89] L. P. Kaelbling, “The foundation of efficient robot learning,” *Science*, vol. 369, no. 6506, pp. 915–916, 2020.
 - [90] 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.
 - [91] 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.
 - [92] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.
 - [93] J. Haviland and P. Corke, “Manipulator differential kinematics: Part 2: Acceleration and advanced applications,” *IEEE Robotics and Automation Magazine*, 2023.
 - [94] 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.
 - [95] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 3400–3407.
 - [96] 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.