Object Rearrangement with Nested Nonprehensile Manipulation Actions

Changkyu Song and Abdeslam Boularias[1]

***Abstract*— This paper considers the problem of rearrangement planning, i.e finding a sequence of manipulation actions that displace multiple objects from an initial configuration to a given goal configuration. Rearrangement is a critical skill for robots so that they can effectively operate in confined spaces that contain clutter. Examples of tasks that require rearrangement include packing objects inside a bin, wherein objects need to lay according to a predefined pattern. In tight bins, collision-free grasps are often unavailable. Nonprehensile actions, such as pushing and sliding, are preferred because they can be performed using minimalistic end-effectors that can easily be inserted in the bin. Rearrangement with nonprehensile actions is a challenging problem as it requires reasoning about object interactions in a combinatorially large configuration space of multiple objects. This work revisits several existing rearrangement planning techniques and introduces a new one that exploits nested nonprehensile actions by pushing several similar objects simultaneously along the same path, which removes the need to rearrange each object individually. Experiments in simulation and using a real Kuka robotic arm show the ability of the proposed approach to solve difficult rearrangement tasks while reducing the length of the end-effector's trajectories.**

I. Introduction

Warehouse automation has witnessed a vast growth in the past few years thanks to the deployment of intelligent robotic solutions. The *Amazon Kiva* mobile robotic fulfillment system is an example of the impressive progress that has been recently achieved in this area. Nevertheless, a large portion of tedious labor performed in warehouse operations is still manual to a large extent. For example, picking and packing products inside bins, such as the small shipping box in Figure 1, is a task that remains highly challenging for robotic systems. In addition to dealing with vision-related challenges, a robotic system needs to plan the trajectories of its arm and the actions to perform by reasoning about the effects of the collisions between the objects in the scene. Moreover, bin packing requires reasoning about the different orders in which the objects should be moved to their marked final poses, which is a combinatorial optimization problem known as object rearrangement planning.

Solving rearrangement problems optimally is computationally prohibitive. Therefore, most proposed algorithms content with finding feasible plans [1]–[3]. The few efficient and optimal algorithms that have been proposed require additional constraints, such as having a clear buffer zone for intermediate placement of objects [4], [5]. Moreover, most of these works use only *pick-and-place* actions and do not take advantage of nonprehensile actions, such as pushing and pulling, that can solve these tasks more efficiently and using only minimalistic end-effectors. While nonprehensile manipulation is a growing topic in robotics [6]–[10], it has not yet been tested on real and dense rearrangement tasks such as the one in Figure 1. In this paper, we present and empirically evaluate an approach for solving this type of tasks. The presented approach utilizes nested pushing actions for moving several objects simultaneously to their targets whenever possible. While global optimality is not a criterion that is maintained or quested by the present work, the proposed approach is tailored for efficient bin packing applications. At a high level, a tree search with backtracking is used for finding subsets of objects to push together. At an intermediate level, trajectories of objects are merged at optimized rendezvous points to benefit from grouped pushing actions. At a low level, the problem of pushing objects to local target points is formulated as a Markov Decision Process and solved offline. The obtained policy is stored in a look-up table and utilized online by the high level planner. Extensive experiments in simulated environments as well as on the real rearrangement tasks shown in Figure 1 clearly demonstrate the effectiveness of this approach.

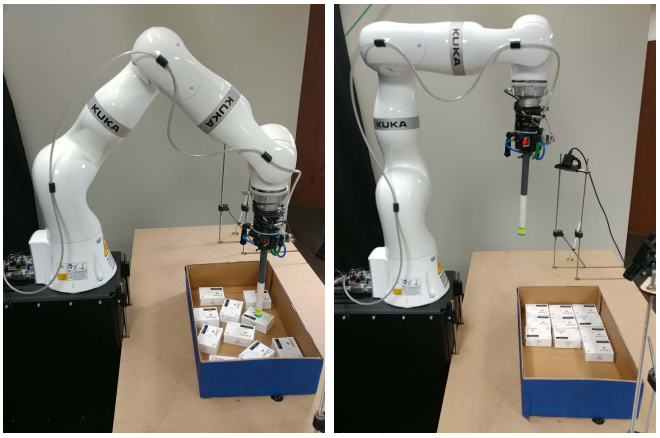


Fig. 1. The product packing problem for cuboid products: initial configuration (left), and achieved goal configuration (right). Experiments are performed using a *Kuka* robotic arm, equipped with a minimalistic end-effector and a depth-sensing camera *SR300*.

II. Related Work

Object rearrangement using non-prehensile actions is related to several areas in robotic manipulation and planning.

Manipulation planning: Object rearrangement problems can be solved with generic sampling-based motion planning

[1]The authors are with the Department of Computer Science of Rutgers University, Piscataway, New Jersey 08854, USA. {cs1080, ab1544}@cs.rutgers.edu

This work was sponsored by NSF IIS-1734492 and IIS-1723869.

algorithms, such as RRT* [11], Kinodynamic RRT [12], [13], and trajectory optimization algorithms such as CHOMP [14]. However, these algorithms do not scale up to spaces that result from combining the configuration spaces of a robot and several manipulated objects. Similar scalability issues are encountered in multi-robot motion planning due to the large number of degrees of freedom [15]. In the context of object grasping, the dimension of the configuration is typically reduced by imposing various motion constraints relative to *transfer* and *transit* actions [16]. It is also often common to explicitly introduce intermediate actions, such as *reconfiguration*, for cost and computation efficiency [17]. Our method is related to the probabilistic path planning method for multiple robots with subdimensional expansion [18], wherein plans in each individual robots configuration space are obtained separately, then those spaces are entangled when robots come into close proximity with one another. In our method, individual spaces of objects are coupled when the objects are physically close to each other.

Object Rearrangement using Prehensile Actions: Most prior works on objects rearrangement focused on *pick-and-place* actions for transporting objects on collision-free paths [1], [19]. One proposed algorithm adapts a manipulation approach for clearing a path to an unreachable object [20], and uses a tree-search with backtracking for finding a feasible order in which objects can be picked and placed [2]. The same algorithm was adapted for solving non-monotone rearrangement tasks, by temporarily moving obstacles to buffer zones. Our approach borrows the general idea of tree-based search from this line of works. A more efficient algorithm that avoids backtracking builds a directed acyclic dependency graph that indicates which object is on the way of which other object, and the objects are then manipulated in the topological order of the dependency graph [3]. Tabletop rearrangement with overhand grasps was also formulated as a TSP problem and an algorithm that minimizes the total distance traveled by the end-effector was proposed [4], [5]. Rearrangement planning has also been used in robotic assembly of discrete architectural structures [21].

Nonprehensile Manipulation: Treating objects always as obstacles that must be avoided at all costs often results in inefficient plans. In practice, collisions can be safe and even helpful in many cases. For example, combined pushing and grasping actions have been shown to succeed where traditional grasp planners fail, and they work well under uncertainty by using the funneling effect of pushing [6]. Environmental contact and compliant manipluation were also leveraged in peg-in-hole planning tasks [7]. Nonprehensile manipluation was also used for inferring 3D shapes and mechanical properties of unknown objects in [22], [23]. As in our rearrangement approach, interactions between objects and end-effectors in [7] are modeled using an MDP near contact manifolds only, while sample-based planners are used in the other regions. DARRT is a sampling-based algorithm for general-purpose motion planning problems with diverse, non-prehensile manipulation actions [8], based on the RRT structure [9]. Our approach also uses RRT and non-prehensile manipulation actions, but it is tailored for efficient rearrangement. The problem of pushing a single target object to a desired goal region was also recently solved through reinforcement learning [10]. In the present work, we are concerned with nested pushing actions, where one or multiple objects are pushed simultaneously.

Object Rearrangement with Nonprehensile Actions: A framework that plans rearrangement of clutter using non-prehensile actions, such as pushing, was introduced by Dogar et al. [24], but it did not consider nested actions. A version of RRT with Kinodynamics also used nonprehensile whole arm rearrangement planning [13], [25]. This approach is related to ours in the sense that whole arm manipulation can be seen as a special type of nested manipulation actions. Monte Carlo tree-search was used to solve rearrangement problems in clutter, with simple non-prehensile actions, the search was guided by policies learned from demonstrations [26]. A similar problem was solved with deep Q-learning [27].

Task and Motion Planning: Rearrangement is a special instance of the general *Task and Motion Planning* (TMP) problem [28]. For example, an iterative TMP algorithm was demonstrated on rearrangement tasks, with the constraints on motion feasibility removed at the task level [29]. Other TMP algorithms rely on factored transition functions [30], impulse exchange constraints at the path optimization level [31], or symbolic representations of the preconditions and effects of the manipulation actions [32]. Efficient heuristics for TMP have also been investigated [33], [34].

Navigation and Manipulation among Movable Objects (NAMO): Rearrangement problems are related to NAMO [35], which is known to be NP-hard even for simple instances [36]. Thus, most efforts have focused on finding time-efficient algorithms that are near-optimal [20], [37], [38]. Minimum constraint removal is another problem related to NAMO, where the planner searches for a transfer path that collides with the minimum number of objects [39], [40]. NAMO techniques were used to improve the computational efficiency of rearrangement planning [3], and can also be used for the approach proposed in the current work.

III. Problem Setup and Notation

Consider a 3D workspace that contains:

- a set of static obstacles $\mathcal{S}$,
- a set of n movable rigid-body objects $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ on a tabletop, where each object $o_i \in \mathcal{O}$ is described by its $3D$ pose (2D position and rotation angle) $p[i] \in \mathcal{P}_i \subseteq SE(2)$,
- and a robotic arm able to rotate, move and push objects. The configuration of the arm is denoted by $q \in \mathcal{Q}$.

Configuration space $\mathcal{C}$ is defined as the Cartesian product of the configuration spaces of the robot and objects: $\mathcal{C} = \mathcal{Q} \times \prod_{i=1}^{n} \mathcal{P}_i$. The valid subset of $\mathcal{C}$ contains all configurations where no object is partly or entirely inside another object, including the robot's arm, and no object is colliding with a static obstacle. Rigid-body collisions between the movable objects, or between the arm and the movable objects, are permitted and used for rearrangement. We denote by $q(p)$

a collision-free configuration of the robotic arm when the end-effector is perpendicular to the tabletop and centered in the 2D position position given by pose p.

There are three different types of valid configurations: *i) Stable*: all the objects rest on the surface and the robotic arm is idle. *ii) Transit*: the robotic arm moves from an initial configuration q_I to a final one q_F while avoiding any collision with the objects, which are described by their joint pose p, we use $\texttt{TRANSIT}(q_I, q_F, p)$ to describe a function that returns a collision-free path to this end. Sets of static obstacles $\mathcal{S}$ and of objects $\mathcal{O}$ are constant during planning and do not need to be provided as arguments to the different functions that we define here. *ii) Push*: the end-effector simultaneously pushes a subset of objects $\mathcal{O}' \subseteq \mathcal{O}$ from their initial joint poses in p_I to final poses in p_F. Collisions with other movable objects are allowed during the transfer, except with the objects that have been successfully moved to their final poses, denoted by set $\mathcal{O}_s$. We use $\texttt{NESTED-PUSH}(q_I, p_I, p_F, \mathcal{O}', L)$ to denote the corresponding path of the end-effector. L is a sorted list that indicates the order in which the objects of $\mathcal{O}'$ should be pushed, i.e the first object in the list pushes the second, while the second pushes the third, and so on. Following the formulation introduced in [2], a legal mode change between two modes is one where the configuration of the robot and objects at the end of the first mode is the same as at the start of the second mode. A rearrangement path $\pi : [0, 1] :\rightarrow \mathcal{C}^H$ is a sequence of states in the joint configuration space $\mathcal{C}$ with legal mode changes. We use $\pi(t).q$ to denote the configuration of the arm at time-step t in path π, and $\pi(t).p[i]$ to denote the pose of object i. Finally, the nonprehensile rearrangement problem is defined as the problem of finding a path π given $x_I = (q_I, p_I)$ and $x_F = (q_F, p_F)$ so that $\pi(0) = x_I$ and $\pi(1) = x_F$. If we define $cost(\pi)$ as the length of the trajectory traversed by the end-effector, then the optimal rearrangement problem is defined as finding the path π with the minimum cost $cost(\pi)$ and that solves the rearrangement task.

IV. Proposed Approach

We first present the general search algorithm for finding a feasible order in which the objects should be pushed. Then, we present the nested pushing approach. Finally, we explain how we obtain the low-level pushing policy.

A. Nested Rearrangement Search (NRS)

The proposed high-level NRS algorithm is given in Algorithm 1 and is an adaptation of an existing rearrangement planning algorithm [2], which itself is an adaptation of a NAMO approach for clearing obstructed paths [20]. The original algorithm of [2] is limited to grasping actions, while the current algorithm considers pushing actions. Moreover, the NRS algorithm also considers nested actions that are applied on sets of objects, as opposed to the mono-object manipulation actions used in [2]. On the other hand, the algorithm of [2] deals also with non-monotone instances, while we focus in this work on monotone instances for simplicity's sake. The subroutines related to pushing and nested operations are presented in the subsequent sections.

Algorithm 1 is a recursive function that takes as input the set $\mathcal{O}_r$ of objects that have not yet been placed in their final poses. In the first call of the function, $\mathcal{O} = \mathcal{O}_r$. Objects that have been successfully placed in their final poses are treated as obstacles during planning, to avoid infinite cycles. Many real-world rearrangement tasks can be solved accordingly. For example, bin packing can be achieved by first placing objects that are close to the edge of the bin. Once the first row is formed, it can be treated as a static obstacle. The bin can be packed by repeating this process, row by row.

Additionally, the algorithm receives the current pose q_I of the robotic arm and the current joint pose p_I of all the objects, which is generally different from the starting pose provided in the first call, because the objects are generally displaced as a result of collisions and pushing actions. The final poses (q_F, p_F) are fixed. The algorithm stops when all objects are correctly placed in their final poses (line 1), in which case a path π that moves the end-effector to its final configuration q_F while avoiding any collision is returned (lines 2-3). Otherwise, the algorithm proceeds into partitioning the remaining objects set $\mathcal{O}_r$ into its powerset of subsets P, with each subset corresponding to a different group of objects that will be potentially pushed simultaneously. Partitions P include sets containing single objects. Sets containing more than one object are manipulated using nested pushes. Given a set P, we create a list of all possible permutations of the objects in P by using subroutine NLC given in Algorithm 2. The set of permutations is denoted by $\mathcal{L}$ (line 5). The algorithm then proceeds into searching for a permutation that provides a feasible nested pushing action that moves all objects in P into their final poses (lines 7-10). Nested pushing actions are performed using subroutine NESTED-PUSH, explained in the next section, which takes as inputs the current state (q_I, p_I) as well as the final joint configuration p_F. An ordered list L indicates the subset of objects in $\mathcal{O}$ that will need to be pushed as well as the order of the nested pushes. NESTED-PUSH returns a path π of the arm-objects configuration (line 8). If the corresponding action is feasible, then $\pi \neq \emptyset$ and the search of permutations stops. The same algorithm is called recursively after removing from $\mathcal{O}_r$ the objects that were successfully placed in the current call (line 12). If it is possible to place the remaining objects, then the algorithm stops after concatenating paths π and π' (line 14), otherwise another subset of objects P is tested, until a feasible order of nested pushing actions of subgroups of objects is found. An empty path $\emptyset$ is returned if no feasible order is found.

B. Nested Pushing

Algorithm 3 returns a path π that moves simultaneously a set of objects to a given goal configuration. The algorithm receives as inputs an initial state (q_I, p_I), a goal configuration of the objects p_F, a list L indicating the objects that will be pushed and the order in which the objects will be nested, in addition to the set of objects that are already successfully

Algorithm 1: Nested Rearrangement Search (NRS)

Input: $\mathcal{O}_r$: set of objects that have not yet been moved to their final poses, (q_I, p_I): initial state, (q_F, p_F): final state;
Output: Path π of the joint arm-objects configuration;

1 **if** $\mathcal{O}_r = \emptyset$ **then**
2 $\pi \leftarrow$TRANSIT(q_I, q_F, p_I);
3 return π;
4 **foreach** $P \in \mathcal{P}(\mathcal{O}_r) \setminus \{\emptyset\}$ **do**
5 $\mathcal{L} =$NLC(P);
6 $\pi \leftarrow \emptyset$;
7 **foreach** $L \in \mathcal{L}$ **do**
8 $\pi \leftarrow$NESTED-PUSH$(q_I, p_I, p_F, \mathcal{O} \setminus \mathcal{O}_r, L)$;
9 **if** $\pi \neq \emptyset$ **then**
10 break;
11 **if** $\pi \neq \emptyset$ **then**
12 $\pi' \leftarrow$NRS$(\mathcal{O}_r \setminus P, \pi(1).q, \pi(1).p, q_F, p_F)$;
13 **if** $\pi' \neq \emptyset$ **then**
14 $\pi \leftarrow \{\pi|\pi'\}$;
15 return π;
16 return $\emptyset$;

Algorithm 2: Nested List Construction (NLC)

Input: $\mathcal{O}_c$: set of objects to chain;
Output: $\mathcal{L}$: set of permutations of objects in $\mathcal{O}_c$;

1 $\mathcal{L} \leftarrow \emptyset$;
2 **foreach** $o \in \mathcal{O}_c$ **do**
3 **foreach** $L \in$NLC$(\mathcal{O}_c \setminus \{o\})$ **do**
4 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(o, L)\}$;
5 return $\mathcal{L}$;

placed, to avoid colliding with them. If the list of objects is empty, then the algorithm returns an empty path (line 3). The returned path π is initialized to an empty list. Counter m counts the number of nested objects that are successfully lined up in front of the end-effector and are ready to be pushed to their final poses. While the algorithm described here is general, we found from our real bin packing experiments that it is practical only for $m \leq 2$. First, the algorithm finds a path π_{main} for pushing the last object alone (line 5). This path is returned by function PUSH that will be described in the next section. The last object is the head of the pushing chain, it is given by $L(N)$ and its aimed final pose is given by $p_F[L[N]]$. π_{main}, like all paths returned by PUSH, avoids collisions with the fixed obstacles $\mathcal{S}$ as well as with objects $\mathcal{O}_s$ that have been successfully placed in the current search branch. But collisions with the remaining objects are allowed. The effects of these collisions are simulated with a physics engine and included in the sequence of states given by the path.

Path π_{main} can be followed simultaneously by a train of

Algorithm 3: Nested Pushing (NESTED-PUSH)

Input: (q_I, p_I): initial state, p_F: final configuration of the objects, $\mathcal{O}_s$: set of objects that are successfully placed in their final configurations, L: sorted list of objects to push;
Output: Path π of the joint arm-objects configuration;

1 $N \leftarrow size(L)$;
2 **if** $N = 0$ **then**
3 return $\emptyset$;
4 $\pi \leftarrow \emptyset$; $m \leftarrow 1$;
5 $\pi_{main} \leftarrow$PUSH$(q_I, p_I, L[N], p_F[L[N]], m, \mathcal{O}_s)$;
6 **if** $\pi_{main} = \emptyset$ **then**
7 return $\emptyset$;
8 **for** $i \leftarrow N-1$; $i > 0$; $i \leftarrow i-1$ **do**
9 $m \leftarrow N - i + 1$;
10 $p_{\text{docking,1}} \leftarrow$DOCKING-POSE$(\pi_{main}, p_I, L, m)$;
11 $p_{\text{docking,2}} \leftarrow$NEAREST-POSE$(\pi_{main}, p_I, L, m, i)$;
12 $p_{\text{rdv}} \leftarrow$RDV-POSE$(\pi_{main}, L, m, p_{\text{docking,2}})$;
13 $\pi_1 \leftarrow$PUSH$(q_I, p_I, L[i], p_{\text{docking,1}}, 1, \mathcal{O}_s)$;
14 $\pi_2 \leftarrow$PUSH$(\pi_1(1).q, \pi_1(1).p, L[N], p_{\text{rdv}}, m, \mathcal{O}_s)$;
15 $\pi_3 \leftarrow$PUSH$(q_I, p_I, L[N], p_{\text{rdv}}, m, \mathcal{O}_s)$;
16 $\pi_4 \leftarrow$PUSH$(\pi_3(1).q, \pi_3(1).p, L[i], p_{\text{docking,2}}, m, \mathcal{O}_s)$;
17 **if** $(\pi_1 = \emptyset \vee \pi_2 = \emptyset) \wedge (\pi_3 = \emptyset \vee \pi_4 = \emptyset)$ **then**
18 return $\emptyset$;
19 **if** $cost(\{\pi_1|\pi_2\}) \leq cost(\{\pi_3|\pi_4\})$ **then**
20 $\pi \leftarrow \{\pi|\pi_1|\pi_2\}$;$(q_I, p_I) \leftarrow (\pi_2(1).q, \pi_2(1).q)$;
21 **else**
22 $\pi \leftarrow \{\pi|\pi_3|\pi_4\}$;$(q_I, p_I) \leftarrow (\pi_4(1).q, \pi_4(1).q)$;
23 $\pi_{all} \leftarrow$PUSH$(\pi(1).q, \pi(1).p, L[N], p_F[L[N]], N, \mathcal{O}_s)$;
24 $\pi \leftarrow \{\pi|\pi_{all}\}$;
25 **for** $i \leftarrow N-1$; $i > 0$; $i \leftarrow i-1$ **do**
26 $\mathcal{O}_s \leftarrow \mathcal{O}_s \cup \{L[i+1]\}$;
27 $\pi_i \leftarrow$PUSH$(\pi(1).q, \pi(1).q, L[i], p_F[L[i]], 1, \mathcal{O}_s)$;
28 **if** $\pi_i = \emptyset$ **then**
29 return $\emptyset$;
30 $\pi \leftarrow \{\pi|\pi_i\}$;
31 return π ;

other objects lined up behind the frontal object if they have a similar or smaller footprint. To this end, the algorithm proceeds into placing the remaining objects in the reverse order of list L (lines 8-22). Note that L can also contain a single object as a special case. In iteration i, the formed chain already contains $m = N - i + 1$ objects. There are various strategies to insert the $(m+1)^{th}$ object to the back of the chain. But for time efficiency, we limit the strategies to two options. The first option is to move the new object directly to the back of the chain. This is done by first finding a *docking point* (line 10), which is a pose denoted by $p_{\text{docking,1}}$ and found by function DOCKING-POSE, explained below. The object is then pushed alone to its docking pose through a path π_1. The second option is to find the nearest point on path

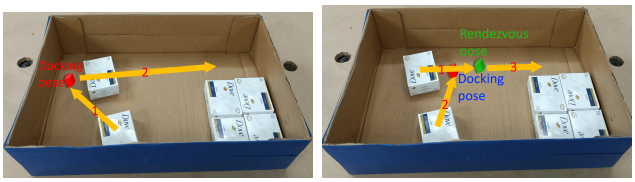


Fig. 2. Strategies for merging paths of two objects

Fig. 3. Nested pushing of objects can rearrange several objects simultaneously and reduce the trajectory of the end-effector.

π_{main} to the current object's pose. The nearest pose, denoted by $p_{\text{docking},2}$ and returned by function NEAREST-POSE, is obtained by using a standard RRT^* search or by a heuristic that ignores other objects and projects the object's position on path π_{main}. From pose $p_{\text{docking},2}$, we compute the corresponding rendezvous pose of the frontal object (line 12). The rendezvous pose indicates where the frontal object should be moved to so that the new object joins the chain when moved to pose $p_{\text{docking},2}$. We then compare the two options, which are (a) first push the new object to the back of the partial chain then push the chain to the final pose of the frontal object, and (b) push the partial chain to that final pose and make sure the new object joins the chain at the rendezvous point. The costs of the two options are computed and the one with the smallest cost is selected (lines 19-22). Note that this choice is not trivial, because the costs also include the transition of the end-effector, which can be initially closer to the start of option (a) or option (b). Option (b) also involves one more transition of the end-effector to bring the new object to the rendezvous point.

At the end of path π, all objects of list L form a chain described in the global state $(\pi(1).q, \pi(1).p)$, with different objects joining at different rendezvous points. The nested chain is pushed so that the frontal object reaches its final pose (line 23). The final steps (lines 25-30) consist in pushing each object individually to its final pose, while growing the set of objects $\mathcal{O}_s$ that are successfully placed (line 26).

We now describe the procedure for computing the docking pose, given by Algorithm 4. The docking pose is simply one that is located near the frontal object of the nested chain. This procedure has two modes: random and path-oriented. In the random mode, the given path is empty, a planar unitary translation vector $\vec{v}$ is randomly sampled. In the path-oriented mode, planar unitary translation vector $-\vec{v}$ is in the opposite direction of the provided path in pose p_1 of the frontal object, which ensures that the docking point is behind the frontal object. Pose p_1 is then translated in the direction of $-\vec{v}$ for a distance of $m \times 2R$ with R being the radius of the largest object in the list L of m nested objects. In our experiments in the present work, we use only cuboid objects that have similar sizes, but this approach can be generalized to objects with different shapes and sizes. The random docking poses are useful for pushing single objects, while path-oriented docking poses are necessary for pushing trains of objects.

Algorithm 4: DOCKING-POSE

Input: π: pushing path, p_I: configuration of objects, L: sorted list of objects to push, m: number of nested objects;
Output: Docking pose p ;

1 $N \leftarrow size(L)$; $i \leftarrow L[N]$;
2 **if** $\pi = \emptyset$ **then**
3 $\quad \vec{v} = (\frac{x}{x^2+y^2}, \frac{y}{x^2+y^2}, 0)$, x and y are random;
4 $\quad p_1 = p_I[i]$;
5 **else**
6 $\quad x_1 = \pi(0); x_2 = \pi(\frac{1}{H})$; $p_1 = x_1.p[i]; p_2 = x_2.p[i]$; ;
7 $\quad \vec{v} \leftarrow p_2 - p_1$, $\vec{v}[3] \leftarrow 0$; $\vec{v} \leftarrow \frac{\vec{v}}{\|\vec{v}\|_2}$;
8 Let R be the radius of the largest object in list L;
9 return $p_1 - 2m\vec{v}R$;

Function RDV-POSE of Algorithm 5 performs the opposite function of DOCKING-POSE, it retrieves the pose where the frontal object should be, given a docking pose, a path π, and a list L of m nested objects to push along path π. Poses returned by RDV-POSE and DOCKING-POSE are translations of the frontal object's pose in $SE(2)$ that preserve its rotation, which is why $\vec{v}[3]$ is set to zero.

Algorithm 5: RDV-POSE

Input: π: pushing path, L: sorted list of objects to push, m: number of nested objects, p_{docking}: docking pose;
Output: Rendezvous pose p ;

1 $N \leftarrow size(L)$; $i \leftarrow L[N]$;
2 $p_1 = p_{\text{docking}}$;
3 Find $t \in [0, 1]$ such that $\pi(t).p[i] = p_1$;
4 $x_2 = \pi(t + \frac{1}{H})$; $p_2 = x_2.p[i]$;
5 $\vec{v} \leftarrow p_2 - p_1$, $\vec{v}[3] \leftarrow 0$; $\vec{v} \leftarrow \frac{\vec{v}}{\|\vec{v}\|_2}$;
6 Let R be the radius of the largest object in list L;
7 return $p_1 + 2m\vec{v}R$;

C. Pushing Policy with Receding Horizons

Algorithm 6 returns a path π for simultaneously pushing a set of m objects nested so that the frontal object o ends up in its final pose p_F. First, the algorithm attempts to sample a docking pose that is in the vicinity of the objects to push and that can be reached by the end-effector in a collision-free path π_{init} (lines 1-5). An RRT^* planner is then called to find the shortest path π for moving the frontal object to its goal (line 8). This path avoids collisions with successfully rearranged objects $\mathcal{O}_s$ and static obstacles. Objects are transported along π through a sequence of highly precise pushing actions to ensure that all the pushed objects remain on π until they arrive at the goal. To achieve

this level of accuracy, π is divided into a sequence of H small horizons. Each small horizon t has an intermediate (waypoint) goal state x_{target} that is given by state $\pi(t+1)$ (line 11).

To reach each intermediate goal x_{target} in a robust manner through a sequence of pushing actions, we formalize this problem as a local Markov Decision Process (MDP). The state space is the cartesian product of the 3D poses of the jointly pushed objects and the end-effector. The 2D position part of the state space is discretized into a regular grid with a limited size, centered around the end-effector. The 1D rotation part is also discretized by dividing the unit circle into 36 arcs. The action space is also discretized into a large number of small rotations and translations of the end-effector. The stochastic transition function T that maps a state s and action a into a distribution over next states s' is learned by first fine-tuning the friction and mass coefficients of the object models inside a physics engine (Bullet), then collecting a large set of training data $\mathcal{D} = \{(s, a, s')\}$ for random starting states s and actions a in the physics engine. Since the true friction and inertial parameters of the objects are not known precisely, next states s' in the data are sampled by adding small random noises to these parameters before simulating the pushing actions. In the real setup, the poses of the objects returned by the vision module are also subject to small errors. Therefore, the 3D poses of the objects in simulation are also perturbed by adding small random noises to the data while learning the transition function of the MDP model. A reward function is also defined by assigning small negative rewards to all states except for the intermediate goal states that are defined as the edges of the grid for the positions, and landmark angles for the orientations. The MDP is then solved by using the value iteration algorithm. Solving this large MDP is time consuming and took around nine hours on a single CPU. However, the MDP is solved only once and offline. The optimal policy resulting from this process is stored in a lookup table and used for all instances of rearrangement planning that involve the same objects.

Algorithm 6: PUSH

Input: (q_I, p_I): initial state, o: frontal object to push, p_F: final configuration of the object, m: number of nested objects, $\mathcal{O}_s$: set of objects that are successfully placed in their final configurations;

Output: Path π of the joint arm-objects configuration;

1 **for** $i \leftarrow 0$; $i < max\_attempts$; $i \leftarrow i+1$ **do**
2 $p \leftarrow$ DOCKING-POSE$(\emptyset, p_I, L, m)$;
3 $\pi_{init} \leftarrow$TRANSIT$(q_I, q(p), p_I)$;
4 **if** $\pi_{init} \neq \emptyset$ **then**
5 break;
6 **if** $i = max\_attempts$ **then**
7 return $\emptyset$
8 $\pi \leftarrow RRT^*(\pi_{init}(1).q, \pi_{init}(1).p, p_F, o, \mathcal{O}, \mathcal{O}_s)$;
9 **for** $t \leftarrow 0$; $t < 1$; $t \leftarrow t + \frac{1}{H}$ **do**
10 $x \leftarrow \pi(t)$;
11 $x_{target} \leftarrow \pi(t+1)$;
12 **for** $i \leftarrow 0$; $i < horizon$; $i \leftarrow i+1$ **do**
13 $\delta q \leftarrow$ MDP-Policy(x, x_{target}, m);
14 $x \leftarrow$ Physics-Propagation$(x, \delta q)$;
15 **if** $\|x - x_{target}\|_2 > \epsilon$ **then**
16 return $\emptyset$;
17 return $\{\pi_{init}|\pi\}$;

V. Evaluation

We report here the results of our experiments for evaluating the proposed method, referred to as NRS (Algorithm 1).

A. Alternative approaches

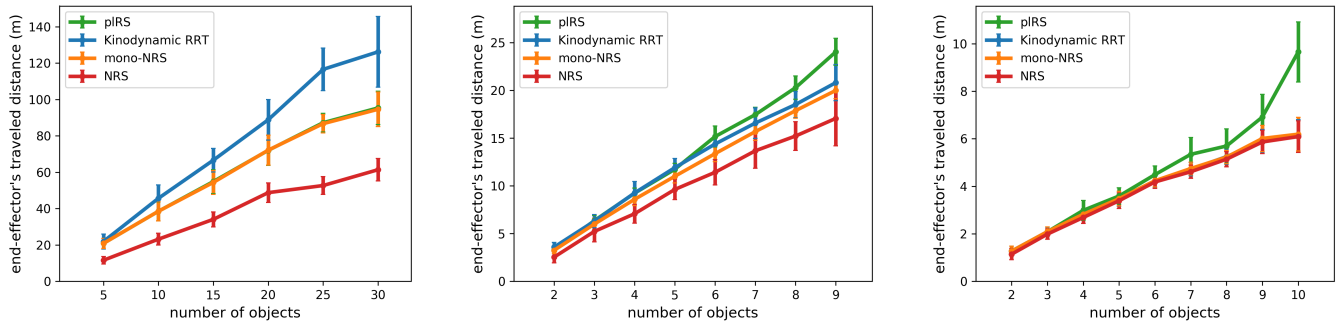
We compare the proposed method to the piecewise linear non-monotone Rearrangement Search (plRS) method [2]. The plRS method was originally proposed for pick-and-place actions, but we extend it here to handle pushing actions as well for a fair comparison. The plRS algorithm finds collision-free paths. If a path is obstructed, then the blocking objects are temporarily moved off the way to the nearest pose in the free space. At a high level, plRS is similar to our method in the sense that it also uses tree search with backtracking to find the order in which the object are moved. To assess the efficiency of the MDP policy for pushing objects, we substitute function PUSH in our algorithm with the Kinodynamic RRT procedure presented in [25] and compare the two methods as well. Kinodynamic RRT was used in [25] for nonprehensile rearrangement.

We also compare to a variant of our method, referred to as mono-NRS, which is identical to NRS except that the number of pushed objects in a given path is limited to one. This is equivalent to replacing the powerset in line 4 of Algorithm 1 with a set of mono-object sets.

B. Experiment Setup

We used the methods discussed above to solve four rearrangement types of tasks, three in simulation and one with a real robot. The simulation tasks are *open-space*, *tabletop*, and *inside-box*. The *open-space* setup is a 2m×2m workspace without obstacles. The *tabletop* setup is a more realistic environment constrained by the robot's configuration space and its field of view as shown in Figure 4. The *inside-box* setup is another realistic environment with a tight free space. For each task, 20 different scenes are generated by sampling random poses of various numbers of objects. The reported results are averages of these independent runs. In the *open-space* task, the targeted final poses are arbitrary. In *tabletop*, objects are pushed from the left side and packed on the right side. In *inside-box*, the targeted final poses are assigned to objects based on their initial poses, each object is assigned to the final pose that is nearest to its initial pose.

The real robot experiments are performed on the setup shown in Figure 1, using 9.5cm×6.5cm×3.5cm cuboid objects inside a 49cm×33cm cardboard box. We randomly generated the initial collision-free poses of the objects in



(a) *open-space* (b) *tabletop* (c) *inside-box*

Fig. 5. Traveled distance of the end-effector as a function of the number of objects in the tasks.

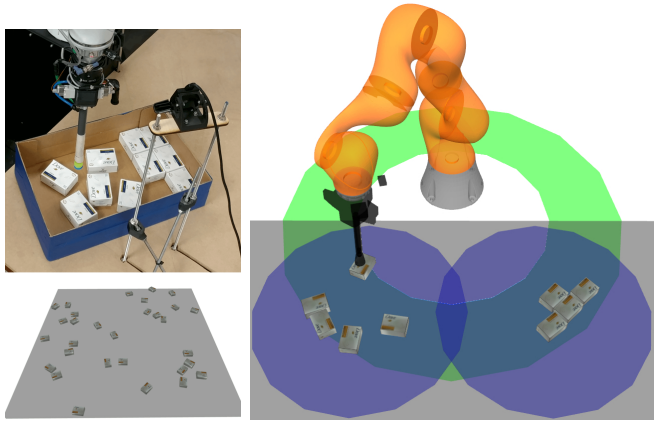


Fig. 4. Experiments are performed on three tasks (left-down) *open-space*, (right) *tabletop*, and (left-up) *inside-box*. The *inside-box* task is performed both in simulation and with a real robot. In (right), green represents the area where the end-effector can operate freely, and blue is the field of view of the robot with two cameras.

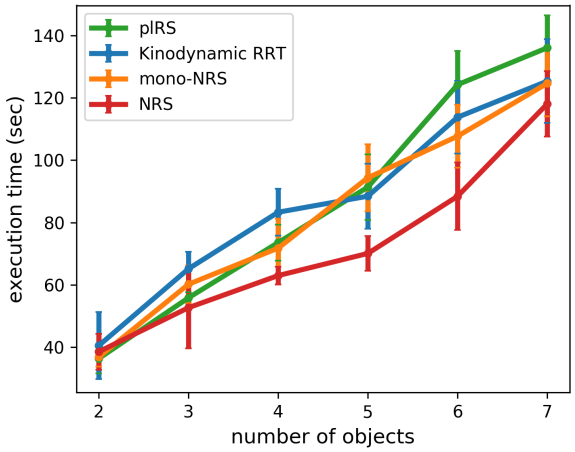


Fig. 6. Execution time as a function of the number of objects in the real robot tasks.

each round, and used the same initial poses for all methods. Poses of objects are then estimated by segmenting the point cloud of the scene and performing PCA on the clusters. The final arrangement poses are those shown in Figure 1 (right). The experiments are performed on bins with two to seven objects. Each experiment for each method is repeated four times. The total number of tested real scenes is 96 $(6\times4\times4)$.

C. Results

	Position	Rotation
plRS [2]	85.2%(±10.9)	82.4%(±12.2)
Kinodynamic RRT [25]	85.2%(±10.9)	76.9%(±14.3)
mono-NRS	85.2%(±03)	81.5%(±06)
NRS	88.0%(±05.6)	86.1%(±08.2)

TABLE I

SUCCESS RATES IN THE REAL ROBOT EXPERIMENT.

The time limit for all methods is set to 10 minutes per task. Within this limit, all methods achieved a 100% success rate in simulation. A successful object placement is defined as one where the final reached pose is within $1cm$ and 10° of the final desired pose.

Figure 5 shows the total traveled distance of the end-effector as a function of the number objects used in the simulated rearrangement task. The results demonstrate that the nested-push approach effectively reduces the end-effector's movement by moving multiple objects together. It also seems like the advantage of this method becomes more pronounced as more objects are considered. In fact, finding objects to push together becomes easier in the presence of multiple objects. On the other hand, when too many objects are cramped inside a small space, it becomes more difficult to successfully push more than one object at a time, which is the reason why the advantage of NRS is clearer in task *open-space*. Because collisions are allowed in the mono-NRS method, objects are sometimes accidentally pushed to their targeted final poses, which contributes to reducing the cost of that method in *inside-box*. Accidental placements are less likely to occur in large open spaces, such as *open-space*

The threshold for success in the real robotic experiments, reported in Table I, is defined as $2cm$ for translation errors and 15° for rotation errors. All methods achieved a reasonable success rate, with no significant difference between the methods. Execution time, however, is significantly smaller when NRS is used, as shown in Figure 6, which confirms the simulation results.

References

[1] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, and K. Bekris, "Rearranging similar objects with a manipulator using pebble graphs," in *2014 IEEE-RAS International Conference on Humanoid Robots*, Nov 2014, pp. 1081–1087.

[2] A. Krontiris and K. Bekris, "Dealing with difficult instances of object rearrangement," in *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[3] A. Krontiris and K. E. Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3924–3931.

[4] H. Shuai, N. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, "High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods," in *Robotics: Science and Systems (RSS)*, 2017.

[5] S. Han, N. Stiffler, K. Bekris, and J. Yu, "Efficient, high-quality stack rearrangement," *IEEE Robotics and Automation Letters*, vol. 107, no. 3, pp. 1608–1615, 2018.

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

[7] C. Guan, W. Vega-Brown, and N. Roy, "Efficient planning for near-optimal compliant manipulation leveraging environmental contact," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, 2018, pp. 215–222. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8462696

[8] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Manipulation with multiple action types," in *International Symposium on Experimental Robotics*, June 2012. [Online]. Available: http://lis.csail.mit.edu/pubs/barry-iser12.pdf

[9] S. Jentzsch, A. Gaschler, O. Khatib, and A. Knoll, "Mopl: A multi-modal path planner for generic manipulation tasks," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 6208–6214.

[10] L. Pinto, A. Mandalika, B. Hou, and S. Srinivasa, "Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions," *CoRR*, vol. abs/1810.10654, 2018. [Online]. Available: http://arxiv.org/abs/1810.10654

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.*, vol. 30, no. 7, pp. 846–894, June 2011. [Online]. Available: http://dx.doi.org/10.1177/0278364911406761

[12] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *ICRA*. IEEE, 2013, pp. 5054–5061.

[13] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *2016 IEEE International Conference on Robotics and Automation, ICRA, Stockholm, Sweden, May 16-21, 2016*, 2016, pp. 3940–3947.

[14] M. Zucker, N. D. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *I. J. Robotics Res.*, vol. 32, pp. 1164–1193, 2013.

[15] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning," *I. J. Robotics Res.*, vol. 35, pp. 501–513, 2014.

[16] J. Mirabel and F. Lamiraux, "Manipulation planning: Addressing the crossed foliation issue," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4032–4037.

[17] J. E. King, M. Klingensmith, C. M. Dellin, M. R. Dogar, P. Velagapudi, N. S. Pollard, and S. S. Srinivasa, "Pregrasp manipulation as trajectory optimization," in *Robotics: Science and Systems*, 2013.

[18] G. Wagner, M. Kang, and H. Choset, "Probabilistic path planning for multiple robots with subdimensional expansion," *2012 IEEE International Conference on Robotics and Automation*, pp. 2886–2892, 2012.

[19] R. Shome, W. N. Tang, C. Song, C. Mitash, C. 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)*, Nomination for Best Paper Award in Automation. Montreal, Canada: Nomination for Best Paper Award in Automation, 05/2019 2019. [Online]. Available: http://robotpacking.org/

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

[21] Y. Huang, C. R. Garrett, and C. T. Mueller, "Automated motion planning for robotic assembly of discrete architectural structures," *CoRR*, vol. abs/1810.00998, 2018.

[22] C. Song and A. Boularias, "Inferring 3d shapes of unknown rigid objects in clutter through inverse physics reasoning," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 2, 2019.

[23] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast model identification via physics engines for improved policy search," 2018.

[24] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, no. 3, pp. 217–236, Oct 2012.

[25] 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 2015, Seattle, WA, USA, 26-30 May, 2015*, 2015, pp. 2508–2515. [Online]. Available: https://doi.org/10.1109/ICRA.2015.7139535

[26] J. E. King, V. Ranganeni, and S. S. Srinivasa, "Unobservable monte carlo planning for nonprehensile rearrangement tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 4681–4688.

[27] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with nonprehensile manipulation using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, 2018, pp. 270–277. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8462863

[28] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 639–646, 2014.

[29] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems*, 2016.

[30] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "Sample-based methods for factored task and motion planning," in *Robotics: Science and Systems (RSS)*, 2017.

[31] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. of Robotics: Science and Systems (R:SS 2018)*, 2018.

[32] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez, "Constructing symbolic representations for high-level planning," in *AAAI*, 2014.

[33] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *WAFR*, 2014.

[34] C. R. Garrett, T. Lozano-Prez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.

[35] M. Stilman, K. Nishiwaki, S. Kagami, and J. J. Kuffner, "Planning and executing navigation among movable obstacles," *Advanced Robotics*, vol. 21, no. 14, pp. 1617–1634, 2007.

[36] E. D. Demaine, M. L. Demaine, M. Hoffmann, and J. O'Rourke, "Pushing blocks is hard," *Computational Geometry*, vol. 26, no. 1, pp. 21 – 36, 2003, the Thirteenth Canadian Conference on Computational Geometry - CCCG'01. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772102001700

[37] K. Hauser, "The minimum constraint removal problem with three robotics applications," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[38] M. Levihn, J. Scholz, and M. Stilman, "Hierarchical decision theoretic planning for navigation among movable obstacles," in *WAFR*, 2012.

[39] K. K. Hauser, "The minimum constraint removal problem with three robotics applications," in *WAFR*, 2012.

[40] K. Hauser, "Minimum constraint displacement motion planning," in *Robotics: Science and Systems*, Berlin, Germany, June 2013.