Decision Making in Joint Push-Grasp Action Space for Large-Scale Object Sorting

Zherong Pan and Kris Hauser†

***Abstract*— We present a planner for large-scale (un)labeled object sorting tasks, which uses two types of manipulation actions: overhead grasping and planar pushing. The grasping action offers completeness guarantee under mild assumptions, and the planar pushing is an acceleration strategy that moves multiple objects at once. We make two main contributions: (1) We propose a bilevel planning algorithm. Our high-level planner makes efficient, near-optimal choices between pushing and grasping actions based on a cost model. Our low-level planner computes one-step greedy pushing or grasping actions. (2) We propose a novel low-level push planner that can find one-step greedy pushing actions in a semi-discrete search space. The structure of the search space allows us to efficiently make decisions. We show that, for sorting up to 200 objects, our planner can find near-optimal actions within 10 seconds of computation on a desktop PC.**

I. Introduction

Countless object sorting machines have been designed over the past century. The robustness of these machines are high enough to be used as a part of a manufacturing process. Early systems use pure mechanical gadgets to force objects into separate buckets according to their shapes [10], [15]. In addition to their robustness, the efficacy of these mechanical systems are rather high, allowing multiple objects to be sorted in parallel. But warehouse automation and service robotics require sorting objects according to visual features, such as the printed address on a package or object color. The vast majority of modern sorting robots [21], [22] solely rely on grasping actions and treat multiple objects in a serial manner. This design choice is largely due to the robustness of grasping to uncertainties in perception and execution. However, serial object grasping does not even reach a fraction of the throughput of purely mechanical gadgets and does not meet the requirements of real world applications, e.g., sorting and delivering billions of objects from amazon fulfillment center per year.

Using planar pushing is a promising direction to achieve more efficient sorting, because many objects can be moved at once [6], [18]. However, planning for pushing is far more complex than for grasping for three reasons. First, the action space of planar pushing is continuous, involving the pusher's initial orientation and the pushing direction, while the action space of overhead grasping is (relatively) discrete. Second, the dynamics of pushing are complex and uncertain, even in the single-object case, since the continuous pressure force distribution between the object and the ground is unknown [7]. Third, although Akella and Mason [1] showed that a single object can be pushed to an arbitrary pose, this has not yet been proven for multi-object pushing. Recent learning-based methods [6], [18], [23] reformulate an object sorting problem as an optimization problem by defining a cost model and using stochastic search to reduce the cost.

This work is partially funded by NSF Grant #2025782. † Zherong Pan and Kris Hauser are with the Department of Computer Science, University of Illinois at Urbana-Champaign. {zherong,kkhauser}@illinois.edu

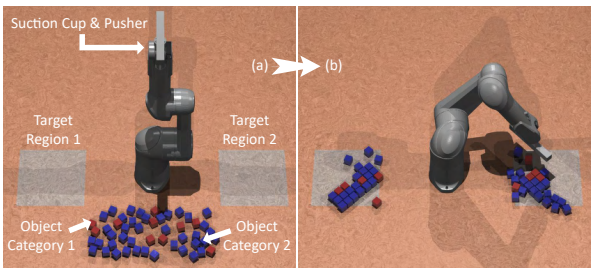

Fig. 1: (a): An illustration of our problem setting. The goal is for the red and blue objects to fall into the 1st and 2nd target region. Our robot is mounted with a suction cup and a pusher (collocated). (b): The objects are sorted with 3 pushing and 1 grasping actions.

Our main contribution is a bilevel motion planner that can efficiently make decisions in joint push-grasp action space. The efficacy of our method is due to two novel techniques. (1) We decompose the responsibility between the high- and low-level planners, such that the low-level planner can efficiently determine one-step greedy grasping or pushing actions and the high-level planner makes binary choices between grasping and pushing actions over multiple steps. Since the high-level planner only considers greedy actions, the branching factor is significantly reduced and searching over multiple steps becomes practical. (2) We make mild assumptions in the low-level push planner, so that finding the optimal pushing action becomes a numerical optimization with piecewise quadratic objective functions. The optimal pushing action can be found via quadratic-piece enumeration, and the brute-force global optimization is avoided.

Compared with learning-based methods [6], [18], [23], we can provide completeness guarantee with the help of the grasp action under mild assumptions that feasibility is not violated by non-prehensile manipulations. (As indicated in [20], non-prehensile manipulations can move objects into unreachable regions of the robot arm, making problem infeasible.) Our method is solely analytical and does not require learning from data. We evaluated our synergetic planner on both labeled and unlabeled tasks of sorting 50-200 objects. The results show that our method can benefit from pushing actions to achieve up to $10\times$ speedup in terms of execution time, as compared with our method using grasping actions alone. Moreover, the computational time to solve for the next action is within 10 s on a desktop PC.

II. Related Work

Multi-object Manipulation allows the robot to move multiple objects simultaneously in order to accomplish a task. Typical tasks involve object sorting [6], [18], clutter removal

[20], [19], object placement [3], and object singulation [23], [8]. We notice two common design choices in these methods. First, all these methods are restricted to 2D workspaces by assuming that the gripper always reaches objects from overhead. Our method also uses this simplification. Second, most of the proposed methods use a single action, either grasping or pushing. An exception is made in [23], where objects are singulated by pushing actions and then grasped, which is similar to our planner. But the pushing action in [23] is used as a grasping auxiliary, and objects are always transferred to target locations using grasping actions, while we allow objects to be transferred by both pushing and grasping. As another difference, our method is analytic whereas all prior works [19], [23] are data-driven.

Grasp Planning is relatively simple in our problem as we assume the use of a suction cup. Most prior works assume more dexterous grippers such as the parallel jaw grippers [11] and multi-fingered grippers [13]. The parallel jaw gripper is available at a low cost and thus assumed in multi-object manipulation planners, e.g. [23], [8], but the gripper feasibility can pose a major problem when objects are densely cluttered. In their most recent work, Mahler [12] used both a parallel jaw gripper and a suction cup mounted on two arms. They argued that the suction cup might fail on certain materials such as hairy deformable objects and objects made of porous media. In applications like warehouse automation, however, this problem can be avoided by packing objects into boxes.

Push Planning is a well-studied problem if there is a single object. Prior work [4] showed that the object motion under pushing can be approximated quasistatically by modeling the limiting surface of contact wrenches. The feasibility of posing a single object using pushing actions has been proved in [1]. Another proof is presented in [24] by showing that planar pushing is differentially flat. However, the object dynamics, motion planning algorithm, and feasibility when pushing multiple objects are still open problems. We propose an aggressively simplified multi-object prediction model. We show that this model allows an efficient computation of the one-step greedy pushing action, while the approximation error is acceptable for accelerating object sorting tasks.

III. Object Sorting Problem

In this section, we formulate large-scale (un)labeled object sorting tasks. We assume that there are N planar objects with center-of-mass at $\mathbf{o}_{1,\cdots,N}$. The planar assumption is used by our low-level grasp planner to enable overhead grasping using suction cups. It is also used by the push planner to analyze object configurations in the 2D projected workspace. These objects are divided into C categories and each object is assigned a category label $l_i \in \{1, \cdots, C\}$. Our problem definition unifies unlabeled object sorting when $C = 1$ and fully labeled object sorting when $C = N$.

We further assume that there are T, pairwise disjoint virtual target regions, where each region is represented as a convex polygon. These regions are virtual and not marked by any physical objects, so that objects will not be blocked when pushed. The convexity of regions is also required by the push planner to predict the result of a potential pushing action. In addition, the regions must be disjoint for the completeness of one-step grasping actions. We denote the closed convex set of the jth target region as $\mathbf{t}_j$. Each $\mathbf{t}_j$ has a capacity for each object category, denoted as $c_{1,\cdots,C}(\mathbf{t}_j)$. The goal of an object sorting task is to move all $\mathbf{o}_i$ such that the following two conditions hold:

$$\sum_{j=1}^{T} \mathbb{I}[\mathbf{o}_i \in \mathbf{t}_j] = 1 \quad \forall i = 1, \cdots, N \tag{1}$$

$$\sum_{l_i=k} \mathbb{I}[\mathbf{o}_i \in \mathbf{t}_j] \le c_k(\mathbf{t}_j) \quad \forall j = 1, \cdots, T \quad k = 1, \cdots, C, \tag{2}$$

where $\mathbb{I}[\bullet]$ is the indicator function. The first equation implies that each object must fall inside one of the target regions. The second equation implies that, in a certain region, the number of objects of a certain category does not exceed the capacity of that region for that category.

A. 3D Workspace

We conduct experiments in a 3D workspace as shown in Figure 1, but our planner performs all the computations in the projected 2D workspace. The 2D-to-3D gap is closed by using a reachability analysis of the gripper. We precompute the inverse kinematics for each uniformly sampled planar position, and approximate the inverse kinematics in between samples using bilinear interpolation. The neighboring samples are connected such that a planar trajectory can be globally resolved using the algorithm proposed in [5], which is important for realizing a pushing action. Our planner will use this reachability map in three ways:

- Function $\mathbf{reach}(\mathbf{x})$ checks whether $\mathbf{x}$ can be reached.
- Function $\mathbf{traj}(\mathbf{x}, \mathbf{y})$ finds a trajectory from $\mathbf{x}$ to $\mathbf{y}$.
- Function $\mathbf{range}(\mathbf{x}, \mathbf{d})$ returns a pair of distances (a, b) that defines the maximal resolvable push from $\mathbf{x}$ along $\mathbf{d}$, i.e. a push from $\mathbf{x} + a\mathbf{d}$ to $\mathbf{x} + b\mathbf{d}$ is the longest, globally resolvable push (a, b can be negative).

B. Overview

Finding reasonable pushing or grasping actions is challenging due to a continuous decision space and a long planning horizon. Although the decision space for grasping actions is discrete, push planner must search over a continuous space of the pusher's position, orientation, and moving distance. If the continuous space is exhaustively discretized, then the branching factor can be prohibitively high.

As illustrated in Figure 2, we combine two ideas to design a practical bilevel planning algorithm. First, we introduce two cost functions: object cost, or cost for short, and robot transit cost. The object cost measures the closeness between an arbitrary configuration and a final, sorted configuration. The transit cost is the traversal distance by the end-effector to accomplish an action. Our low-level planner minimizes the object cost, while the high-level planner also considers the transit cost. Second, we significantly reduce the branching factor by only considering one-step greedy actions. In other words, our high-level planner only chooses the type of

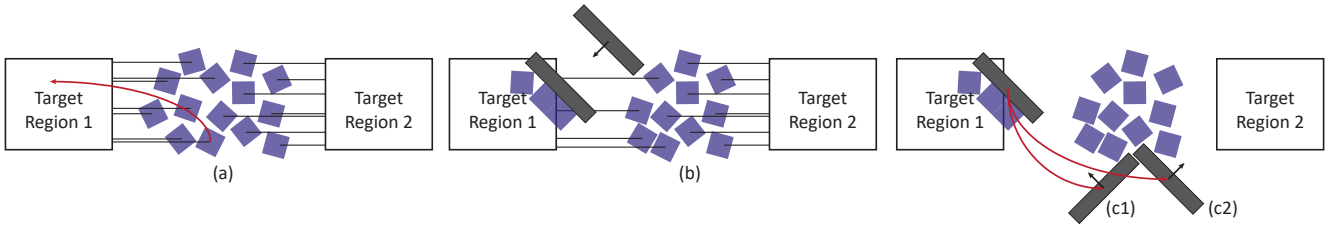

Fig. 2: We illustrate a problem with $C = 1, N = 14$. (a): Given the object-to-region assignment, the greedy grasping action is illustrated as the red arrow. (b): A greedy pushing action would lead to larger reduction in the cost and thus chosen as the first action. The second action could be (c1) or (c2) and our high-level planner would choose (c1) over (c2), since (c1) induces a lower transit cost (red arrow).

actions (grasping or pushing), while we use two low-level planners to ensure that the chosen action leads to the highest reduction in object cost among all actions of the same type.

Intuitively, our cost function J sums over the distances between objects and target regions. This cost is zero if and only if sorting is successful. Among all possible object-to-target-region pairings, we choose the one with lowest cost value, which amounts to the following optimization:

$$J(\mathbf{o}_i) = \min_{b_{ij}\in\{0,1\}} \sum_{i=1}^{N}\sum_{j=1}^{T} b_{ij}\mathbf{dist}(\mathbf{o}_i,\mathbf{t}_j) \quad \mathbf{s.t.} \sum_{j=1}^{T} b_{ij} = 1 \quad \sum_{l_i=k} b_{ij} \le c_k(\mathbf{t}_j), \tag{3}$$

where $\mathbf{dist}$ is the Euclidean distance between a point and a convex polygon. Computing J amounts to solving an optimal assignment problem, for which the Hungarian algorithm can be used at a computational cost of $\mathcal{O}(N^3)$, by introducing dummy variables to absorb the capacity constraints. Obviously, $J(\mathbf{o}_i) = 0$ if and only if the two conditions in Equation 1,2 hold, but the function J allows us to monitor the progress and compare different planning algorithms.

In the rest of the paper, we first introduce the low-level grasp planner (Section IV) and the push (Section V) planner. We then introduce a single high-level planner (Section VI) that chooses greedy actions over multiple steps in a receding-horizon manner as outlined in Algorithm 1. The completeness guarantees are provided in our extended report [16] .

IV. Low-Level Grasp Planner

We present a grasp planner that finds a single grasp action to minimize the cost function. This planner provides a fail-safe feasibility guarantee for any sorting problem under the mild assumption that all objects are reachable. We show such grasp actions can be found via a small-scale mixed-integer linear program (MILP), which can be solved within a couple hundreds of milliseconds. We first define the radius of an object. If we compute a bounding circle for the ith object centered at $\mathbf{o}_i$ with radius r_i, then we define $R = \max_{i=1,\cdots,N} r_i$. We sample a set of potential positions $\mathbf{p}_{mn}$ to put the grasped object, and we assume uniform sampling with a spacing equals to $\sqrt{2}R$, i.e. $\mathbf{p}_{mn} \triangleq \left(\sqrt{2}Rm, \sqrt{2}Rn\right)$. To find the one-step greedy grasping action that reduces the cost as much as possible, we introduce binary variables b_{ij} as in the cost model, where $b_{ij} = 1$ implies that $\mathbf{o}_i$ is not the object to be grasped and it is assigned to $\mathbf{t}_j$. We further introduce another set of binary variables p_{mnj} for each $\mathbf{p}_{mn}$ and $j = 1, \ldots, T$, where $p_{mnj} = 1$ implies that an object will be grasped and put to the sampled location $\mathbf{p}_{mn}$ and this grasped object will be assigned to $\mathbf{t}_j$. After solving for b_{ij}, p_{mnj}, we can identify the object $\mathbf{o}_i$ to be grasped if $\sum_{j=1}^{T} b_{ij} = 0$ and we will move it to $\mathbf{p}_{mnj}$ if $p_{mnj} = 1$. Finally, we compute the gripper trajectory by calling $\mathbf{traj}(\mathbf{o}_i, \mathbf{p}_{mn})$. We solve for b_{ij}, p_{mnj} using the following MILP:

$$\begin{aligned} &\underset{b_{ij},p_{mnj}\in\{0,1\}}{\mathbf{argmin}} \quad \mathcal{O} \\ &\mathbf{s.t.}\ J_{post} \le J(\mathbf{o}_i) \\ &\sum_{i=1}^{N}\sum_{j=1}^{T} b_{ij} = N-1 \quad \sum_{mn}\sum_{j=1}^{T} p_{mnj} = 1 \\ &\sum_{j=1}^{T} b_{ij} \le 1 \quad \sum_{l_i=k} b_{ij} + \mathbb{I}\left[\sum_{j=1}^{T} b_{ij} = 0\right] \sum_{mn} p_{mnj} \le c_k(\mathbf{t}_j) \\ &J_{post} \triangleq \sum_{i=1}^{N}\sum_{j=1}^{T} b_{ij}\mathbf{dist}(\mathbf{o}_i,\mathbf{t}_j) + \sum_{mn}\sum_{j=1}^{T} p_{mnj}\mathbf{dist}(\mathbf{p}_{mn},\mathbf{t}_j), \end{aligned} \tag{4}$$

where J_{post} is the post-grasping cost. We have used three types of constraints. First, we ensure that the cost is monotonically reduced. Second, we ensure that only one object will be grasped and the object will be put to only one sampled location. Finally, we have the assignment constraints (each object can only be assigned to one target region) and capacity constraints (the multiplication with indicator function can be relaxed as mixed integer linear constraints). The objective function $\mathcal{O}$ can take multiple forms. If we want to reduce the total cost as much as possible, then $\mathcal{O} = J_{post}$, and we denote the resulting grasping action as $\mathcal{G}(\mathbf{o}_i, \mathbf{p}_{mn})$. In this case, the function grasp(STATE) in Alg. 1 returns $\{\mathcal{G}(\mathbf{o}_i, \mathbf{p}_{mn})\}$ and contributes 1 to the branching factor. If we want to reduce the cost related to a single target region, e.g. $\mathbf{t}_j$, then we can define:

$$\mathcal{O} = \sum_{i=1}^{N} b_{ij}\mathbf{dist}(\mathbf{o}_i,\mathbf{t}_j) + \sum_{mn} p_{mnj}\mathbf{dist}(\mathbf{p}_{mn},\mathbf{t}_j),$$

and denote the resulting grasping action as $\mathcal{G}_j(\mathbf{o}_i, \mathbf{p}_{mn})$. In this case, grasp(STATE) returns $\{\mathcal{G}_1, \cdots, \mathcal{G}_T\}$ and contributes T to the branching factor. Finally, we show in our extended report [16] that J_{cost} can be monotonically reduced to zero under mild assumptions, which is a completeness guarantee.

V. Low-Level Push Planner

Although grasp actions are easy to find, they only move objects sequentially. In this section, we present our push planners that potentially accelerate sorting by moving multiple objects at once. A push planner is challenging to design as we are making decisions in a continuous action space that involves the pusher's location, pushing direction, and pushing distance. Indeed, even predicting the single object

Algorithm 1 High-Level Planner

Input: Initial state STATE $\leftarrow \{\mathbf{o}_i\}$, max horizon H
Initialize stack STACK $\leftarrow \{$STATE$\}$
Best action ACTION* $\leftarrow$ None, $J^*_{rate} \leftarrow \infty$
while STACK not empty **do**
  STATE $\leftarrow$ pop(STACK)
  {ACTION} $\leftarrow$ grasp(STATE) $\cup$ push(STATE)
  for ACTION $\in$ {ACTION} **do**
    STATE$^+$ = simulate(STATE, ACTION)
    if horizon(STATE$^+$) $< H$ **then**
      STACK $\leftarrow$ STACK $\cup$ {STATE$^+$}
    else if J_{rate}(STATE$^+$) $< J^*_{rate}$ **then**
      ACTION* $\leftarrow$ backTrace(STATE$^+$)
      $J^*_{rate} \leftarrow J_{rate}$(STATE$^+$)
Return ACTION*

motion during pushing is non-trivial [7]. To analyze multi-object motions, our method is based on the following two assumptions similar to [19]: 1) The pusher is rectangle and pushing direction is orthogonal; 2) Objects will only translate along pushing direction. We illustrate some key notions in Figure 3(a). We assume that the pusher can only move in one of D directions. For each pushing direction $\mathbf{d}$, we define the affected region (gray) as the region formed by sweeping the pusher along $\mathbf{d}$. Any object that falls entirely inside this region (blue) will be considered affected. There are boundary cases when objects fall partially in this region (red). We assume that objects of boundary cases will not be affected by the pushing action. Our push planner consists of two steps. First, we show that there are only discrete number of possible pusher locations that can be enumerated. For each pusher's location, we then compute the optimal pushing distance d.

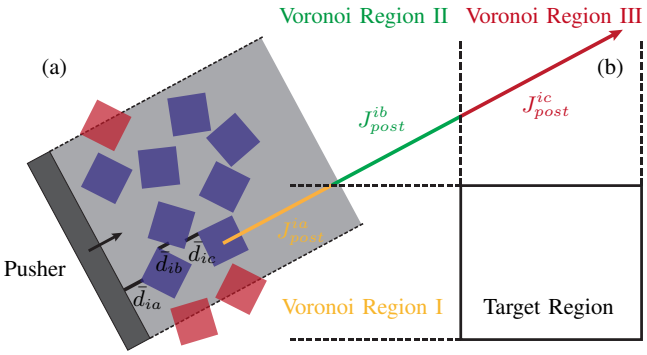

Fig. 3: (a): We illustrated the pusher (dark gray), pushing direction (arrow), the affected region (light gray), and the boundary case (red objects). (b): We illustrate the cost function $J(\mathbf{o}_i)$ of each object being pushed. $J(\mathbf{o}_i)$ is piecewise quadratic, where the quadratic pieces are dictated by the Voronoi regions of the target area (J^{ia}_{post},J^{ib}_{post},J^{ic}_{post}). The compression distance is the sum of three short segments (black line segments $\bar{d}_{ia,ib,ic}$.)

A. Locating the Pusher

For a pushing direction $\mathbf{d}$, its orthogonal direction is denoted as $\mathbf{d}^\perp$. A pusher's location is expressed as $\alpha\mathbf{d}+\beta\mathbf{d}^\perp$. We compute the two coefficients α, β by sorting objects' locations along $\mathbf{d}$ and $\mathbf{d}^\perp$. Since we assume that an object $\mathbf{o}_i$ is a convex polygon, we can define its vertices as $\mathbf{v}_i^1, \cdots, \mathbf{v}_i^{V(\mathbf{o}_i)}$, where $V(\mathbf{o}_i)$ is the number of vertices in $\mathbf{o}_i$. We then define the four supports of $\mathbf{o}_i$ along $\mathbf{d}$ and $\mathbf{d}^\perp$ as:

$$\alpha_{min(i)} = \min_{k=1,\cdots,V(\mathbf{o}_i)} \langle \mathbf{d}, \mathbf{v}_i^k \rangle \quad \alpha_{max(i)} = \max_{k=1,\cdots,V(\mathbf{o}_i)} \langle \mathbf{d}, \mathbf{v}_i^k \rangle$$
$$\beta_{min(i)} = \min_{k=1,\cdots,V(\mathbf{o}_i)} \langle \mathbf{d}^\perp, \mathbf{v}_i^k \rangle \quad \beta_{max(i)} = \max_{k=1,\cdots,V(\mathbf{o}_i)} \langle \mathbf{d}^\perp, \mathbf{v}_i^k \rangle.$$

Similarly, the pusher is rectangular so it spans two ranges: $[\alpha_{min(p)}, \alpha_{max(p)}]$ along $\mathbf{d}$ and $[\beta_{min(p)}, \beta_{max(p)}]$ along $\mathbf{d}^\perp$. We then record all β values satisfying:

$$\beta + \beta_{min(p)} = \beta_{min(i)} \vee \beta + \beta_{min(p)} = \beta_{max(i)} \vee$$
$$\beta + \beta_{max(p)} = \beta_{min(i)} \vee \beta + \beta_{max(p)} = \beta_{max(i)},$$

for some i and we sort these key β values in ascending order denoted as $\beta_1 \le \beta_2 \le \cdots \le \beta_{4N}$, where there are at most $4N$ cases. When the pusher moves between β_n and β_{n+1} along $\mathbf{d}^\perp$, the set of affected objects is invariant and denoted as:

$$\mathcal{A}^\perp_n = \{\mathbf{o}_i | \beta_n + \beta_{min(p)} \le \beta_{min(i)} \le \beta_{max(i)} \le \beta_{n+1} + \beta_{max(p)}\}.$$

We repeat this procedure along $\mathbf{d}$ to define the $4N$ key values $\alpha_1 \le \alpha_2 \le \cdots \le \alpha_{4N}$, and the set of affected objects:

$$\mathcal{A}_m = \{\mathbf{o}_i | \alpha_m + \alpha_{max(p)} \le \alpha_{min(i)}\}.$$

Note that the definition of $\mathcal{A}_m$ is different from $\mathcal{A}^\perp_n$ in that we only consider objects in front of the pusher, as illustrated in Figure 3 (a). Finally, we define an additional set of objects overlapping the pusher as $\mathcal{I}_{nm}$, which is computed via collision checking. In summary, a possible pusher location $\alpha\mathbf{d} + \beta\mathbf{d}^\perp$ must satisfy the following conditions:

$$\alpha \in [\alpha_m, \alpha_{m+1}] \quad \beta \in [\beta_n, \beta_{n+1}]$$
$$\mathcal{A}_m \cap \mathcal{A}^\perp_n \neq \varnothing \quad \mathcal{I}_{nm} = \varnothing,$$

where there are at most $16N^2$ choices. To further reduce the computational cost, we only retain one nominal pusher position within each valid α, β range.

Algorithm 2 Computing the compression distance $\bar{d}_i$

$\bar{d}_i \leftarrow \alpha_{min(i)} - \alpha - \alpha_{max(p)}$
for Each vertices $\mathbf{v}_i^k$, $k = 1, \cdots, V(\mathbf{o}_i)$ **do**
  Shoot a ray from $\mathbf{v}_i^k$ along $-\mathbf{d}$, record first intersection.
  if Ray intersects object $\mathbf{o}_j$ after traveling $\bar{d}_i^k$ **then**
    if $\mathbf{o}_j \in \mathcal{A}_m \cap \mathcal{A}_n$ **then**
      $\bar{d}_i \leftarrow \mathbf{min}(\bar{d}_i, \bar{d}_i^k + \bar{d}_j)$ ▷ Recursion
Return $\bar{d}_i$

B. Finding the Optimal Pushing Distance

For a given pair of $\alpha \in [\alpha_m, \alpha_{m+1}]$ and $\beta \in [\beta_n, \beta_{n+1}]$, we plan the optimal pusher distance that reduces $J(\mathbf{o}_i)$ the most. We first solve Equation 3 to find $b_{ij} = 1$, i.e. an affected object $\mathbf{o}_i$ is assigned to the target region $\mathbf{t}_j$. If we move the pusher by distance d, then we need to compute the following post-pushing cost function:

$$J^i_{post}(d) \triangleq \mathbf{dist}(\mathbf{o}_i(d), \mathbf{t}_j).$$

If $J^i_{post}(d)$ can be expressed analytically, then we can find the optimal pushing distance by solving:

$$d^* = \operatorname*{\mathbf{argmin}}_d \mathcal{O}$$
$$\mathbf{s.t.} \sum_{\mathbf{o}_i \in \mathcal{A}_m \cap \mathcal{A}^\perp_n} J^i_{post}(d) \le J(\mathbf{o}_i) \qquad (5)$$
$$d \in \mathbf{range}(\alpha\mathbf{d} + \beta\mathbf{d}^\perp, \mathbf{d}),$$

where we have added a constraint to ensure monotonic cost reduction. Similar to the case with grasping actions, the objective function $\mathcal{O}$ can take multiple forms. If we want to reduce the overall cost function, we can set $\mathcal{O} = \sum_{\mathbf{o}_i \in \mathcal{A}_m \cap \mathcal{A}_n^\perp} J^i_{post}(d)$ and denote the resulting pushing action as $\mathcal{P}(\mathbf{d}, \alpha, \beta, d^*)$. In this case, push(STATE) in Algorithm 1 returns $\{\mathcal{P}\}$ and contributes to the 1 branching factor. If we want to reduce the cost related to a single target region, e.g. $\mathbf{t}_j$, we can set $\mathcal{O} = \sum_{\mathbf{o}_i \in \mathcal{A}_m \cap \mathcal{A}_n^\perp \wedge b_{ij}=1} J^i_{post}(d)$ and denote the resulting pushing action as $\mathcal{P}_j(\mathbf{d}, \alpha, \beta, d^*)$. In this case push(STATE) in Algorithm 1 returns $\{\mathcal{P}_1, \cdots, \mathcal{P}_T\}$ and contributes T to the branching factor.

Algorithm 3 Push planner

Solution $< \alpha^*, \beta^*, d^* > \leftarrow$ None, best $\mathcal{O}^+ \leftarrow \infty$
Compute all possible ranges $\{[\alpha_m, \alpha_{m+1}]\}$ and $\{[\beta_n, \beta_{n+1}]\}$
for Each $[\alpha_m, \alpha_{m+1}] \in \{[\alpha_m, \alpha_{m+1}]\}$ **do**
  for Each $[\beta_n, \beta_{n+1}] \in \{[\beta_n, \beta_{n+1}]\}$ **do**
    $\mathcal{O}(d) = 0$ ▷ Build objective
    for $i = 1, \cdots, N$ **do**
      ▷ Only consider objects in affected region
      ▷ Only consider objects in front of pusher
      if $i \in$ affected($[\alpha_m, \alpha_{m+1}]$,$[\beta_n, \beta_{n+1}]$) **then**
        Compression distance $\bar{d}_i$ (Algorithm 2)
        $\mathcal{O}(d) \leftarrow \mathcal{O}(d) + J^i_{post}(d)$
    Solve Equation 5 for $\mathcal{O}^+, d^+$ ▷ Minimize objective
    if $\mathcal{O}^+ < \mathcal{O}^*$ **then**
      $< \alpha^*, \beta^*, d^* > \leftarrow < \frac{\alpha_m + \alpha_{m+1}}{2}, \frac{\beta_m, \beta_{m+1}}{2}, d^+ >$
      $\mathcal{O}^* \leftarrow \mathcal{O}^+$
Return $< \alpha^*, \beta^*, d^* >$

To solve for the global minima of the above 1D optimizations analytically, we show that each $J^i_{post}(d)$ is piecewise quadratic and so is their summation. As a result, the 1D optimization can be solved by enumerating and finding the global minima of each quadratic piece. The first quadratic piece is denoted as the void piece with length $\bar{d}_i$, i.e. $J^i_{post}(d) = J^i_{post}(0)$ if $0 \le d \le \bar{d}_i$. The length $\bar{d}_i$ is denoted as the compression distance, i.e. the minimal distance that we have to move the pusher in order to touch the object. In the illustrative example of Figure 3(b), we have $\bar{d}_i = \bar{d}_{ia} + \bar{d}_{ib} + \bar{d}_{ic}$. $\bar{d}_i$ can be computed analytically by the recursive raycasting Algorithm 2. If the pushing distance is larger than $\bar{d}_i$, then the distance $\mathbf{dist}(\mathbf{o}_i(d), \mathbf{t}_j)$ will change according to the Voronoi region of $\mathbf{t}_j$ that $\mathbf{o}_i$ belongs to [14]. Within each Voronoi region, $\mathbf{dist}(\mathbf{o}_i(d), \mathbf{t}_j)$ is a quadratic function of d. In the planar case, there are only two types of Voronoi regions, corresponding to vertex and edge, respectively. In the example of Figure 3(b), we illustrate three quadratic pieces with $J^{1a}_{post}, J^{1c}_{post}$ corresponding to edge regions and J^{1b}_{post} to a vertex region. The dividing points between regions can be determined by computing the intersections between Voronoi region boundaries and the object's moving path.

We summarize our push planner Algorithm 3 by estimating the computational complexity. For each pushing direction, our planner first enumerates possible pusher locations, where there are at most $16N^2$ cases. For each case, there are at most N objects in the affected set. Each object contributes a piecewise quadratic cost model, with at most $1 + 2 \max_{j=1,\cdots,T} V(\mathbf{t}_j)$ pieces, where $V(\mathbf{t}_j)$ is the number of vertices of $\mathbf{t}_j$. After summing up J^i_{post} to get J_{post}, it has at most $N(1 + 2 \max_{j=1,\cdots,T} V(\mathbf{t}_j))$ pieces. If we assume solving for each quadratic piece takes constant time, then our algorithm has the following complexity: $\mathcal{O}(16DN^3(1 + 2 \max_{j=1,\cdots,T} V(\mathbf{t}_j)))$. Note that the complexity in practice is much lower than this upper bound because many pusher locations are colliding with objects and thus pruned.

VI. High-Level Receding-Horizon Planner

We can perform synergetic planning using low-level planners alone, by first computing the one-step greedy actions, $\mathcal{G}, \mathcal{P}$, and then picking the action with larger cost reduction. But this strategy has two drawbacks. First, our push planner is based on a simplified kinematic model, which might suffer from a high approximation error. Second, our cost model in the low-level planner does not take transit cost into consideration, which can slow down the overall efficacy [9].

Our high-level planning Algorithm 1 mitigates these two drawbacks. First, we use Box2D [2] to simulate $\mathcal{P}$ and compute a more accurate cost reduction by solving Equation 3 before and after each simulation. Second, our high-level planner considers the transit cost $J_{transit}$ and seeks to maximize the following modified cost function:

$$J_{rate} = (J(\mathbf{o}_i) - J_{post})/J_{transit}, \tag{6}$$

i.e. the rate of cost reduction per unit end-effector movement of the gripper. To effectively reduce J_{rate}, we have to expand the decision space. We observe that, when using low-level planners alone, the gripper will suffer from unnecessary transits by jumping between target regions (i.e. grasping an object to t_i, pushing another object to t_j, and then grasping a third object to t_i again). To avoid this artifact, we choose to not use the overall greedy actions $\mathcal{G}, \mathcal{P}$, but use the actions focused on a single target region, i.e. $\mathcal{G}_j, \mathcal{P}_j$. In other words, we allow the gripper to reduce the cost of one target region as much as possible, before transiting to another target region. In addition, our high-level planner seeks to reduce Equation 6 over multiple steps via an action-tree search. Whenever a push action is used in a tree node, then the state after the push is predicted using Box2D [2] (Line 7 of Algorithm 1). The branching factor of this search is $2T$, as we choose one action from the set $G_{1,\cdots,T}, P_{1,\cdots,T}$. We can further reduce the branching factor by half if we choose between grasping or pushing actions greedily for each target region, which does not degrade the overall performance empirically.

VII. Evaluations

We implement our method using mixed Python/C++, where we use C++ to perform multi-threaded distance computations as in Figure 3. All the experiments are conducted on a desktop machine with a 10 core Intel(R) Xeon(R) W-2155 CPU. We evaluate our method in a simulated environment with a STAUBLI 6-axis Industrial robotic Arm TX90 as well as the bimanual hardware platform in Figure 4 equipped with a 20cm-long pushing bar and a Robotiq vacuum gripper.

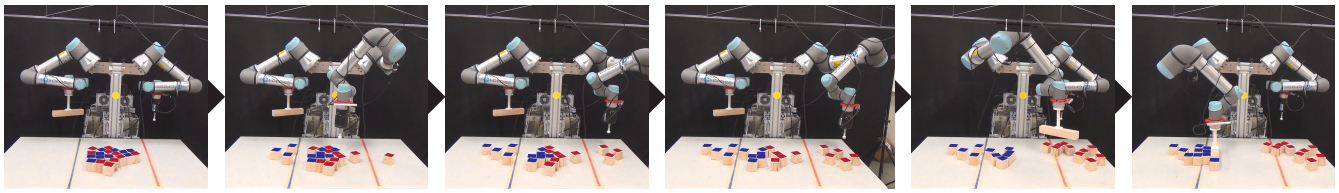

Fig. 4: We show several frames of hardware execution with 2 object categories (red and blue).

The simulation is performed using ODE [17] where the control signals are provided by a PID controller. Our goal is to push $N = 50 - 200$ cubical bricks ($5 \times 5 \times 5cm^3$) to $T = 1 - 4$ target regions ($100 \times 50cm^2$).

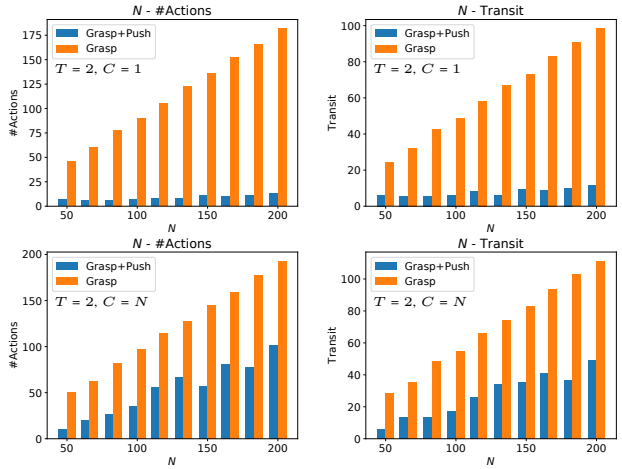

Fig. 5: We show the speedup/number-of-actions/transit-cost with/without pushing actions, plotted against the number of objects. Top Row: $T = 2$, $C = 1$, $c_0(\mathbf{t}_j) = N/2$; Bottom Row: $T = 2$, $C = N$.

Speedup Using Pushing Actions: In Figure 5, we show the speedup using grasping+pushing actions, as compared with grasping actions alone. We use two settings: unlabeled ($C = 1$) and fully labeled categories ($C = N$). When $C = 1$, the robot is mostly using pushing actions, and the pushing action could provide $6 - 15\times$ speedup in terms of number of actions and $3-10\times$ speedup in terms of the transit cost. When $C = N$, the robot is forced to use more grasping actions due to the fixed assignment, and the pushing actions could only provide $1.4 - 3.1\times$ speedup in terms of number of actions and $1.6 - 4.2\times$ speedup in terms of the transit cost. In each test case, the initial object poses are sampled randomly.

Speedup Using The High-Level Planner: In Figure 6, we show the speedup with/without using the high-level planner. We randomly generate 10 sorting tasks with parameters sampled uniformly in range: $50 \leq N \leq 200$, $2 \leq T \leq 4$, $C = 2$, $c_0(\mathbf{t}_j) = 0.8N/2$, $c_1(\mathbf{t}_j) = 0.2N/2$. We observe that the high-level planner does not help reducing the number of actions except in one task. However, the high-level planner does reduce the transit cost in most cases, achieving up to $2\times$ speedup. Using $H = 3$ increases the computational time of each decision making by $8\,s$ as compared with $H = 1$. We observe that further increasing H was not worth the extra computational time.

Brute-Force Push Planner: We design a simple push planner by sampling a grid of points with spacing R and then connect each points with its 8 neighbors, resulting in a set of edges each representing a candidate pushing action. We test each pushing action using Box2D and pick the one that minimizes the cost. Altogether, there are 2500 candidate pushing actions that fall inside the reachable set. The two methods return pushing actions of comparable quality, but our method uses 1.5s to find a greedy pushing action, while the brute-force method takes 37s on average.

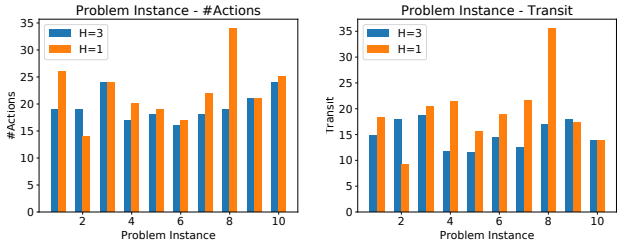

Fig. 6: On 10 randomly generated sorting tasks, we show the speedup/number-of-actions/transit-cost with/without high-level planner. (H is the high-level planning horizon, and $H = 1$ means no high-level planner is used)

VIII. Conclusion & Limitations

We propose a synergetic push-grasp planner for large-scale object sorting tasks. Our planner uses the grasping action to ensure feasibility of the task, and we use pushing actions to accelerate the execution. We show that one-step greedy grasping actions can be found by solving MILP, and with the help of a simplified kinematic model, one-step greedy pushing actions can be found by analyzing and enumerating pusher configurations. Finally, we take the transit cost into consideration using a high-level planner to perform multi-step action selection. As a major limitation, our method assumes perfect sensing and requires the exact knowledge of object configurations. In practice, objects can be occluded and thus cannot be localized exactly, in which case the two low-level planner should be modified to account for uncertainties. Furthermore, our simplified kinematic model is similar to [19], which assumes that characteristic length of each object is much smaller than that of the pusher or the target region size. If larger objects are sorted, our assumptions on object motions during pushing will be violated. Finally, although we have shown that grasping actions are feasible for object sorting, the pushing actions can violate this guarantee. This is because objects might be pushed too far away and they leave the reachable set of the gripper. In practice, a hardware-side or software-side safety mechanism can be implemented to ensure reachability.

REFERENCES

[1] S. Akella and M. T. Mason, "Posing polygonal objects in the plane by pushing," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, 1992, pp. 2255–2262 vol.3.

[2] E. Catto, "Box2d," *Available fro m: http://www. box2d. org*, 2010.

[3] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 4627–4632.

[4] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction part 1. limit surface and moment function," *Wear*, vol. 143, no. 2, pp. 307–330, 1991.

[5] K. Hauser and S. Emmons, "Global redundancy resolution via continuous pseudoinversion of the forward kinematic map," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 932–944, 2018.

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

[7] Jiaji Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 372–377.

[8] M. Kiatos and S. Malassiotis, "Robust object grasping in clutter via singulation," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 1596–1600.

[9] Y. Koga and J.-C. Latombe, "On multi-arm manipulation planning," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 945–952.

[10] C. Leopoldo, "Fruit and potato sorting machine," Oct. 17 1961, uS Patent 3,004,663.

[11] J. Mahler and K. Goldberg, "Learning deep policies for robot bin picking by simulating robust grasping sequences," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 515–524.

[12] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, p. eaau4984, 2019.

[13] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[14] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Transactions On Graphics (TOG)*, vol. 17, no. 3, pp. 177–208, 1998.

[15] H. T. Odquist and S. Benjamin, "Sorting machine," Mar. 2 1943, uS Patent 2,312,357.

[16] Z. Pan and K. Hauser, "Decision making in joint push-grasp action space for large-scale object sorting," *arXiv preprint arXiv:2010.10064*, 2020.

[17] R. Smith *et al.*, "Open dynamics engine," 2005.

[18] 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," 2019. [Online]. Available: http://arxiv.org/abs/1912.07024

[19] H. T. Suh and R. Tedrake, "The surprising effectiveness of linear models for visual foresight in object pile manipulation," in *The 14th International Workshop on the Algorithmic Foundations of Robotics*, 2020.

[20] W. N. Tang and J. Yu, "Taming combinatorial challenges in clutter removal," in *The 2019 International Symposium on Robotics Research*, 2019.

[21] P. R. Wurman and J. M. Romano, "Amazon picking challenge 2015," *AI Magazine*, vol. 37, no. 2, pp. 97–99, 2016.

[22] K.-T. Yu, N. Fazeli, N. Chavan-Dafle, O. Taylor, E. Donlon, G. D. Lankenau, and A. Rodriguez, "A summary of team mit's approach to the amazon picking challenge 2015," *arXiv preprint arXiv:1604.03639*, 2016.

[23] 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*, 2018.

[24] 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.