Tight Robot Packing in the Real World
A Complete Manipulation Pipeline with Robust Primitives

Rahul Shome*+, Wei N. Tang*, Changkyu Song, Chaitanya Mitash, Hristiyan Kourtev, Jingjin Yu, Abdeslam Boularias, and Kostas E. Bekris

***Abstract*— Many order fulfillment applications in logistics, such as packing, involve picking objects from unstructured piles before tightly arranging them in bins or shipping containers. Desirable robotic solutions in this space need to be low-cost, robust, easily deployable and simple to control. The current work proposes a complete pipeline for solving packing tasks for cuboid objects, given access only to RGB-D data and a single robot arm with a vacuum-based end-effector, which is also used as a pushing or dragging finger. The pipeline integrates perception for detecting the objects and planning so as to properly pick and place objects. The key challenges correspond to sensing noise and failures in execution, which appear at multiple steps of the process. To achieve robustness, three uncertainty-reducing manipulation primitives are proposed, which take advantage of the end-effector's and the workspace's compliance, to successfully and tightly pack multiple cuboid objects. The overall solution is demonstrated to be robust to execution and perception errors. The impact of each manipulation primitive is evaluated in extensive real-world experiments by considering different versions of the pipeline. Furthermore, an open-source simulation framework is provided for modeling such packing operations. Ablation studies are performed within this simulation environment to evaluate features of the proposed primitives.**

I. Introduction

The past decade has witnessed a growth of autonomous robot solutions for logistics and warehouse automation, such as the *Amazon/Kiva* fulfillment system [1]. Many tasks, however, still rely on repetitive human labor, such as building customer orders. In particular, picking packaged products from unstructured piles and tightly packing them into another container is a frequent task but often is performed manually [1]. This is because packing objects in confined spaces, such as shipping boxes, as in Fig. 1 (top), is challenging. It requires placing objects in close vicinity to each other, in an ordered manner and aligned to the container's boundary. This demands high accuracy both from perception and manipulation. Apart from recent insights into the combinatorial, and geometric bottlenecks [2], [3], relatively little work exists that addresses real-world complexities of this task, let alone using commodity hardware, such as a robotic arm, a suction-based end-effector and an RGB-D sensor.

The authors are affiliated with the Computer Science Dept. of, Rutgers University, New Brunswick, and +Rice University. Email:{jingjin.yu,abdeslam.boularias,kostas.bekris}@cs.rutgers.edu

*These authors have equally contributed to the work.

The authors would like to acknowledge the support of NSF 1723869, 1734492 and JD-X Research and Development Center (RDC). Any opinions expressed here or findings do not reflect those of the sponsor.

[1]The authors would like to thank Dr. Hui Cheng and members of the JD-X Research and Development Center for introducing the packing task.

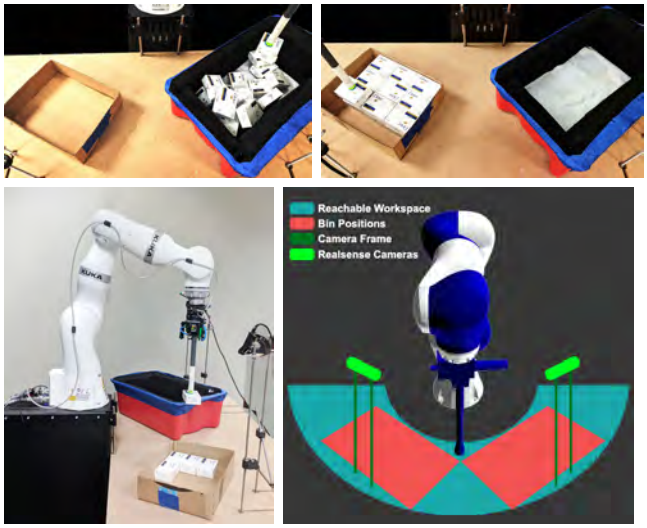


Fig. 1. (*Top row:*) Tight packing for cuboid products: initial configuration (left), and achieved packing (right). (*Bottom left:*) The problem is solved using a *Kuka LBR iiwa* arm equipped with a suction-based end-effector and depth-sensing cameras *SR300*. (*Bottom right:*) Two bins are placed within the reachability region of the arm given overhand picks.

Warehouse Automation: This article is motivated by real-world warehouse automation tasks. Most automation efforts so far have focused on picking. Product placement is also critical but less automated[2]. Examples of placement tasks include: (i) *Induction:* This relatively easier task requires dropping a picked object in an unobstructed surface (e.g., conveyor), potentially with a desired orientation. (ii) *Specialized Packaging:* This often involves mechanisms that build the container around the product. The objects often need to be properly pre-positioned. (iii) *Stowing:* Placing a product in a container for storage without regard to its final pose. (iv) *Aggregation:* Bringing together batches of identical products (depanning) or containers (denesting). Palletizing requires containers to be packed into pallets with some degree of uniformity and careful positioning. (v) *Tight Packing:* The focus task of this work requires precise positioning of objects and containers. It can be addressed without sensing if the object and container positions are guaranteed by a conveying mechanism, which is not always available and requires additional investments. Robust, and efficient autonomous packing can provide significant economic, and environmental savings by reducing packaging material, storage space, and shipping costs. These costs typically form a good fraction of expenses in order-fulfilment and online retail.

[2]See an online survey of industrial warehouse automation here: `https://robotpacking.org/industrial_automation.html`

Contribution: This work develops a robot packing pipeline using a minimalistic end-effector for different cuboid products presented in an unstructured pile that must be placed in arbitrarily sized open boxes. The focus is on making this process robust for real-world deployment. To help narrow the application gap and enable the reliable, fully autonomous execution of packing, this article:

A. Proposes minimalistic hardware and accompanying software architecture. The hardware shown in Fig.1 (bottom-left) integrates a single robot arm, depth-imaging technology and a suction-based end-effector, which is also used as a pushing or dragging finger. While objects can be initially presented in configurations that require reorientation for proper placement, the suction-based end-effector is shown sufficient to solve such challenges. The result is a fully autonomous integrated system, as shown in Fig. 1 (top-right).

B. Develops corrective manipulation primitives to increase robustness due to uncertainty arising both from actuation errors and noise in visual sensing. These closed-loop primitives intentionally use *contacts* and *compliance* of objects, bins, and the end-effector to resolve the following tasks in real-time: (i) object *toppling* in the unstructured bin to expose a desirable object surface for picking; (ii) *adaptive pushing* of objects towards their target placement while displacing neighboring objects to further pack them given point cloud data; and (iii) real-time monitoring of potential failures and *fine correction* to achieve tight packing.

C. Provides an open-source simulation framework for robot packing. The simulation framework is made available to the research community to facilitate benchmarking in this space[3]. The simulator allows the modeling of high-fidelity RGB-D sensor information and compliant interactions.

D. Extensively evaluates the pipeline and its primitives both in simulation and with the real platform of Fig. 1 (bottom-left). The experiments show that the proposed primitives provide robustness despite the setup's minimalism.

II. Related Work

This work lies at the intersection of robot manipulation, perception, and solutions for industrial deployment.

Picking Objects in Clutter: Traditionally, grasping used fingered hands [4] for form- or force-closure for standalone objects [5] or for clutter [6]–[8]. Suction-based grippers were shown successful in the Amazon Picking Challenge [9]–[14] and attracted more interest. Model-based or data-driven grasping [6] often identifies points on object surfaces and hand poses that result in grasps. Model-based methods rely on object models to identify picks that are stable. They can be sensitive to modeling errors. Data-driven techniques rely on examples of grasps to regress successful grasps [15] without depending on models. They tend to be computationally efficient during inference [16] but require a sufficient amount of training data. This work follows a model-based approach using suction and aims for simplicity, robustness and reduced computational cost.

[3]https://robotpacking.org/simulator.html

Non-prehensile Manipulation: Actions beyond grasping, such as pushing, are helpful in manipulating objects in clutter, such as for reducing the uncertainty of a target object's pose [17], [18] and for rearrangement [19], [20]. The current work uses non-prehensile manipulation to topple objects, inspired by early work [21], and for placement to counter the effects of inaccurate object localization. A related approach [22] performs within-hand manipulation of an object by pushing it against its environment. The proposed system takes advantage of the end-effector's compliance and leverages contact with the environment to accurately place objects or topple them. Recent work is exploring robust toppling [23], and the modeling of compliant contacts [24].

6D Pose Estimation: A lot of recent efforts in 6D pose estimation employ deep learning by performing regression over object orientations and centers from images [25]. An alternative approach first predicts 3D object coordinates, followed by a RANSAC scheme to predict the object's pose [26]. Geometric consistency has also been used to refine the predictions from learned models [27]. The current work is based on the authors' prior effort [28], which first performs image segmentation to then guide a geometric search process for estimating 6D poses of objects [29]. Uncertainty over pixel labels is returned by a convolutional neural network and used to register 3D models into the point cloud. Recent progress, motivated by the packing application, deals specifically with pose estimation of regular arrangements of similar objects closely situated in a container [30].

Bin Packing Algorithms: This NP-hard problem [31], [32] considers the spatial arrangement of objects of different or similar volumes so that they are packed into a cubical bin. Most strategies search for ε-optimal solutions via greedy algorithms [33]. To the best of the authors' knowledge, these solutions are not deployed in real robotic workcells, where inaccuracies in vision and control must also be dealt with. Recent work [2], [3] has focused on computing incremental stable arrangements for packing objects in a container. Such methods could be integrated with the proposed pipeline to specify target arrangements.

Relation to Previous Work by the Authors: This article extends previous work [34] as follows: *A.* The simulator is a new contribution. New ablation experiments using the simulator test: 1) the robustness of the primitives against different levels of simulated noise, and 2) the impact of underlying parameters. *B.* A novel extension to the adaptive pushing primitive solves problems where there is no clearance for the target objects given sufficient compliance. *C.* A real world demonstration is included in Section VI where the containers' size is exactly the size of the packed arrangement. *D.* Additional demonstrations performed on the robot show new capabilities of the proposed system: 1) toppling the object to its narrow, less-stable side, and 2) handling a pile with different objects and packing them into separate boxes. *E.* The text expands upon the primitives' technical details in Section IV, including pseudocode, and the coverage of the related literature in Section II.

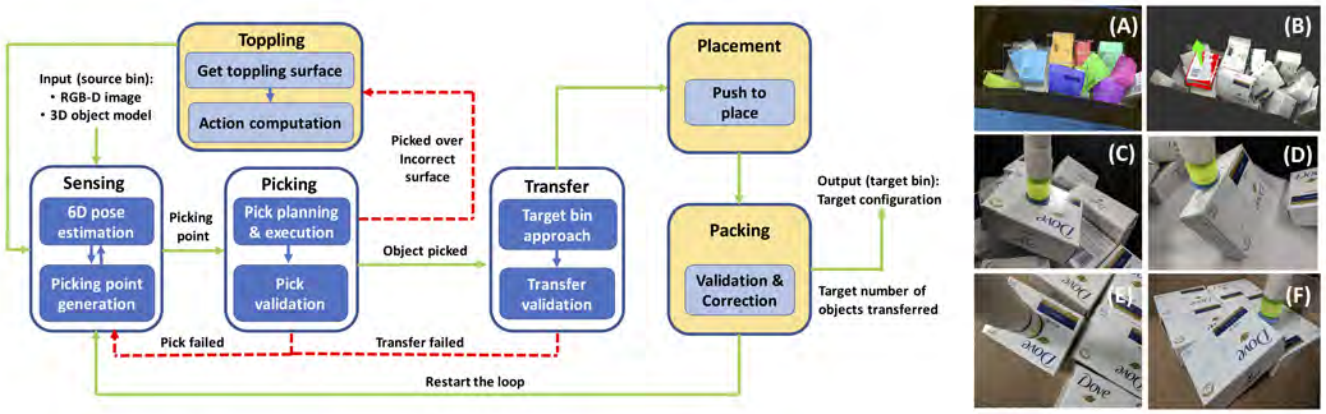


Fig. 2. *Left*: Pipeline with the system modules (blocks) highlighting data flow (green lines) and failure handling (red lines). Sensing receives an RGBD image and object CAD models to return a grasp point. Depending on the picking surface, the object is either transferred to the place scene or is handled by the `Toppling` module, which flips the object in the pick scene. When the object is transferred, a robust `Placement` module places the object at the target pose. The `Packing` module validates and corrects the placement to achieve tight packing. *Right*: a) Instance segmentation. b) Pose estimation and pick point selection, c) Picking d) Toppling e) Placement and f) Packing.

III. Problem Setup and Notation

Consider a workspace $\mathcal{W}$ with: a) a robot arm, b) static obstacles, c) n movable cuboid objects $\mathcal{O} = \{o^1, \dots, o^n\}$, d) two static container bins $\mathcal{B}_{init}$ and $\mathcal{B}_{goal}$, which are compliant bodies in known poses that define cuboid volumes, where the objects can be placed.

A labeled arrangement $A = \{p^1, \dots, p^n\}$ is an assignment of poses $p^i \in SE(3)$ to each object o^i. The objects start at an initial, random but "stable" arrangement A_{init} inside $\mathcal{B}_{init}$, i.e., the objects are stably resting and not moving. The A_{init} arrangement is *not* known a-priori to the robot. The objective is to move $\mathcal{O}$ to an unlabeled arrangement $\hat{A}_{\text{goal}} = \{\hat{p}^1, \dots, \hat{p}^n\}$, which achieves a tight packing in $\mathcal{B}_{goal}$. $\hat{A}_{\text{goal}}$ depends on the pose of $\mathcal{B}_{goal}$, its dimensions and the objects' dimensions. The target unlabeled arrangement is input for the proposed process. $\hat{A}_{\text{goal}}$ is a regular grid packing for the cuboid objects, which maximizes the number of objects inside $\mathcal{B}_{goal}$, as they rest on a stable face, and minimizes the convex hull of their volume. The unlabeled nature of $\hat{A}_{\text{goal}}$ means it is satisfied as long as one of the objects is placed at each target pose, i.e.,

$$\forall \hat{p}^j \in \hat{A}_{\text{goal}} : \exists\ o^i \in \mathcal{O} \text{ so that } D(\hat{p}^j, p^i) < \epsilon, \quad (1)$$

where ϵ is a threshold for achieving the target pose; $D(\cdot, \cdot)$ is distance between object poses, which considers the 3-axis symmetry of the cuboid objects, i.e., if two poses result into an object occupying the same volume, their distance is 0. Such a distance metric for 6D poses is the ADI metric [35].

The arm has d joints that define the arm's configuration space $\mathbb{C}_{\text{full}} \subset \mathbb{R}^d$, which has a subset $\mathbb{C}_{\text{free}}$ that is collision-free given the static obstacles and the bins. Valid arm motions correspond to a continuous C-space curve $\pi : [0, 1] \rightarrow \mathbb{C}_{\text{free}}$. The arm has an end-effector, such as a suction cup, for which discrete operations $\{\texttt{pick}, \texttt{release}\}$ give rise to discrete manipulation modes: $\mathcal{M} = \{\texttt{transfer}, \texttt{transit}\}$. No within hand manipulation operations are available. The state space of the arm is: $\mathcal{X} = \mathbb{C}_{\text{free}} \times \mathcal{M}$. Sensing is used to reason about the current object poses. Overall, the *robot operations* involve (i) rotating the joints, (ii) picking or releasing objects and (iii) sensing.

The arm's forward kinematics define a mapping $FK : \mathbb{C}_{\text{full}} \rightarrow SE(3)$, which provides the pose $g \in SE(3)$ of the end-effector given $q \in \mathbb{C}_{\text{full}}$. The reachable task space defines the set of end-effector poses that can be reached without collisions: $\mathcal{T} = \{\forall\ q \in \mathbb{C}_{\text{free}} : FK(q) \in SE(3)\}$. For the arm to pick o^i at p^i, it has to be that the end-effector's pose g satisfies a binary output function: $\texttt{is\_pick\_feasible}(o^i, p^i, g)$, where $g \in \mathcal{T}$. For instance, the pose g of a suction cup must align with one of the surfaces of an object o^i at p^i. Then, the set of end-effector poses that allow to pick an object at a specific pose are:

$$\mathcal{G}(o^i, p^i) = \{g \in \mathcal{T} : \texttt{is\_pick\_feasible}(o^i, p^i, g) = true\}.$$

Assume $\mathcal{G}(o_i, p_i)$ is non-empty for all objects $\mathcal{O}$ and poses in A_{init} or $\hat{A}_{\text{goal}}$. Otherwise, the task is not feasible. Note it may be necessary to reconfigure the objects inside $\mathcal{B}_{init}$ to pick them from an appropriate face before placing them, given the lack of within-hand manipulation. Then, the task is to identify a sequence of *robot operations* to transfer objects $\mathcal{O}$ from the unknown initial stable arrangement A_{init} in $\mathcal{B}_{init}$ to a tight, grid-based packing inside $\mathcal{B}_{goal}$ that satisfies Eq. 1 given an unlabeled arrangement $\hat{A}_{\text{goal}}$.

IV. Proposed Solution

Fig. 2 describes a pipeline for solving packing given the setup of Fig. 1. More details about the hardware are provided in Section VI. A straightforward baseline involves: a) *"Sensing"* objects $\mathcal{O}$ in $\mathcal{B}_{init}$ and selecting an object o^i; b) *"Picking"* o^i by executing action $\{\texttt{pick}\}$; c) "Transfer" of o^i to the next available target pose $\hat{p}^j$ in $\mathcal{B}_{goal}$ by executing $\{\texttt{release}\}$ at that pose. Experiments show that this baseline performs poorly given pose uncertainty, calibration errors, object non-uniformity and unexpected contacts. This motivates remedies, which actively increase robustness. To this end, three manipulation primitives are designed to increase robustness and are integrated with the overall architecture: i) *"Toppling"*; ii) *"Adaptive Pushing"*; and iii) *"Corrective Actions"*. This section describes first a baseline and then each one the proposed robust manipulation primitives.

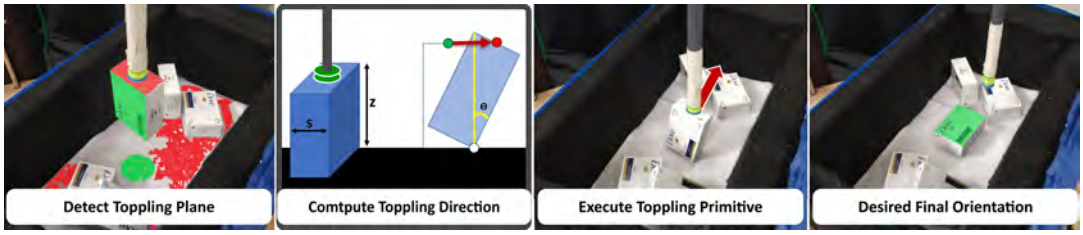


Fig. 3. Steps for toppling when the object's pose does not directly allow the desired placement configuration: (from left to right) a plane is detected in the scene that can allow toppling; the toppling direction is computed, where z is the height of the face to be picked; toppling is executed by moving the end-effector along a vector of motion for the top face; the reoriented object is picked from a pose that allows the desired placement.

A. Baseline: Sense, Pick and Transfer

Given an RGB-D image of the source bin $\mathcal{B}_{init}$ and a CAD model of the target object category, the objective is to select o^i that can achieve a pose p^i given the next available target pose $\hat{p}^j$ so that $D(p^i, \hat{p}^j) \leq \epsilon$. To achieve this, the image is passed through `Mask-RCNN` [36] trained to perform segmentation and retrieve the visible subset of object instances $\mathcal{O}$. An image segment is ignored if it has a number of pixels below a threshold or if `Mask-RCNN` has small confidence the segment is of the target object category.

The remaining segments are arranged in a descending order given the mean global Z-coordinate of all the RGB-D pixels in each segment. Then, 6D pose estimation is performed for the highest-order instance [37] [28]. If the detected 6D pose reveals that the object's top-facing surface does not allow its placement via a top-down pick and place, the next segment instance in the ordering is evaluated. If no object reveals a top-facing surface, then the object with the maximum mean global Z-coordinate is chosen for picking.

For the chosen object, a picking point, i.e., a point where the suction cup is attached, is computed over the point cloud registered against the object model. The approach uses a pre-computed picking score on the object model, which indicates the pick's stability by minimizing the distance to the object's center. A continuous neighborhood of planar pickable points are required to make proper contact between the suction cup and the object's surface. A local search is performed around the best score point to maximize its pickable surface. The pick is performed at that maximum and the picked object is transferred to the target pose.

B. Toppling

The toppling primitive is invoked if there exists no object, which exposes the desirable top-facing surface, or if the object was erroneously picked from the wrong face. The latter is detected after the pick by performing pose estimation once the object is attached to the suction cup. For instance, this can happen for the soaps in Figure 2 (right)(D), if only the thin side is available for pick but the soap needs to be placed on its wide side. In these cases, toppling is performed to reorient the object. Given the starting pose of an object p_{start} and a toppling action of the arm, the object ends up at pose p_{topple}. The requirement is for p_{topple} and the next available target pose $\hat{p}^j$ to have the same top-facing surface.

Prior work [21] has shown the efficacy of minimal end-effectors used in tandem with the environment to achieve toppling. In the previous work, the friction against a conveyor belt is used to topple an object about a resting surface. The conveyor belt's motion is parallel to the initial resting surface plane. In the current setup, the compliance of the suction cup is used to emulate the same effect using a lateral motion on the same plane as the top-surface along the direction of the desired transformation between p_{start} and p_{topple}. Due to the symmetry of cuboidal objects, at least one neighboring surface allows a successful toppling to exist. The accompanying results show this to be highly effective in the target setup.

Algorithm 1 outlines the toppling process. It returns a sequence of poses, which constitute the discrete steps of the toppling maneuver. When executed, they drop the object to expose a desired face, denoted by the target orientation. It is invoked once the object is already attached to the end-effector using a top-down pick, and is also visible to the camera. Line 2 checks if reorientation is necessary by measuring the distance between the current object rotation r and the target rotation r_{target}. The toppling plane is computer over the point cloud based on the dimensions of the object's bottom surface. p_{plane} denotes the new object pose when its bottom surface touches the toppling plane.

Algorithm 1: TOPPLE

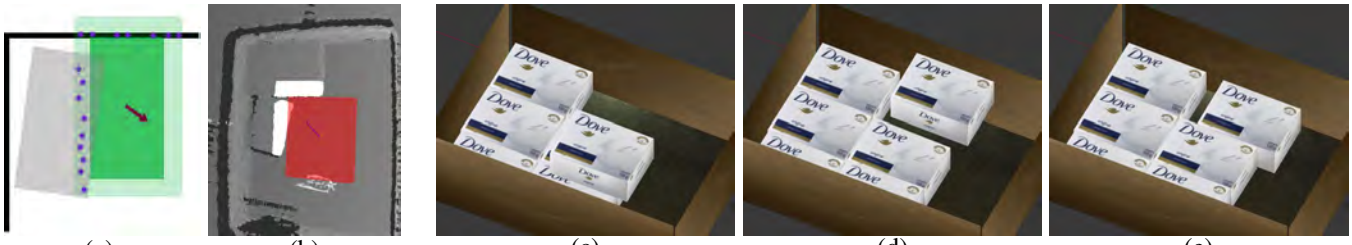
Input: Point cloud X, object o, top-down current object pose $p_{start} \leftarrow (t, r)$, target orientation r_{target}
Output: Toppling primitive Π

1 $\Pi \leftarrow \emptyset$;
2 **if** $D(p_{start}, (t, r_{target})) > \epsilon$ **then**
3 $\quad t_{\text{plane}} \leftarrow$GETTOPPLINGPLANE($X$,DIMENSIONS($o$));
4 $\quad p_{\text{plane}} \leftarrow (t_{\text{plane}}, r)$
5 $\quad \Pi \leftarrow \Pi \cup p_{\text{plane}}$;
6 $\quad (\theta, \Delta x, \Delta z) \leftarrow$GETOFFSET($o, p_{\text{plane}}, r_{target}$);
7 $\quad \Pi \leftarrow \Pi \cup$ APPLYOFFSET($p_{\text{plane}}, \theta, \Delta x, \Delta z$);
8 **return** Π

The GETOFFSET primitive is described in the second image of Fig 3. Given the pose p_{plane} of the object, the height of the top face z, and the dimension of the edge(s) that are not adjacent to the desired face s, define θ as $\tan^{-1}\frac{s}{z}$. The toppling operation needs to change the orientation of the object such that its center of gravity moves out of the support point along the contact edge denoted by the white circle. This object rotation can be achieved by displacing the pick's point of contact (green circle) to the toppled configuration (red circle). This offset is computed as:

$$\Delta x = \frac{s}{2} + z\sin\theta - \frac{s}{2}\cos\theta + \epsilon;\ \ \Delta z = z\cos\theta + \frac{s}{2}\sin\theta - z.$$

Here, Δx and Δz describe the motion vector for the end-effector shown as a red arrow in the third image of Fig 3.



(a) (b) (c) (d) (e)

Fig. 4. Adaptive pushing: (a) The black border is the bin and the gray rectangle is a previously placed object. The green rectangle represents the target pose for the current object. The light green boundary represents an ε-expanded model of the target object that intersects the point cloud at the purple points. These points result in the black vector that pushes the object away from them. (b) A screen shot of a scene's point cloud. The white points are collision points with previously placed objects. The red volume shows the computed pre-push pose for the new object. (c) The rest of the images show adaptive pushing for placing an object at the top-right corner, where there is no free space given the bin. (d) The target object is pushed against the boundary and then pushed down to the level of the objects. (e) The target object is moved towards its target pose at the bottom level.

C. Point Cloud Driven Adaptive Pushing

Directly placing the i^{th} object at the target pose $\hat{p}^i$ is prone to placement failures. Previous placement errors can cause surrounding objects to block $\hat{p}^i$. Executing a direct placement may result in damaging the objects. This means that the object can only be lowered safely to an offset pose that avoids surrounding objects. Once lowered to this offset, "pre-push" pose, a key observation is that *pushing in the direction of the target pose* can a) enable the current object to reach its target pose with low error, and b) correct local errors through compliant interactions with other objects. These observations motivate the proposed *adaptive pushing primitive*.

In order to compute a collision-free "pre-push" pose p_{pre} the method operates over a point cloud snapshot X taken of the target bin with all the previously placed objects. The cuboidal model of the object B_{box} is evaluated *in simulation* at the target pose $\hat{p}^i$. Due to errors in prior placements, parts of X intersect with B_{box}. The set of intersecting points can be used to compute a planar offset direction to reduce collisions. Doing this repeatedly by offsetting the pose with small planar displacements will succeed once the offset pose has no intersection with X. Once computed, the push direction $\vec{d}$ takes the object from the safe p_{pre} to target $\hat{p}^i$.

There is another source of error arising from sensing and calibration. This means that p_{pre} is collision-free only as long as the detected object pose and current execution is perfect. Real-world noise means that attempting to lower the object at p_{pre} will encounter errors that cause the ground-truth object pose to differ from p_{pre}. Expanding the planar dimensions of the cuboidal object by ϵ to compute B_{box} means that as long as the noisy ground-truth is contained within the expanded B_{box} the real-world execution is *still guaranteed to be collision free*. Similarly, a small raise in height h is also incorporated to account for vertical noise, so that the pushing action is clear of the floor of the container. Fig 4 (a,b) is an illustration of the offset computation.

Algorithm 2 outlines a single ADAPTIVEPUSH primitive. Line 3 generates the bounding box region with expanded object dimensions at a raised target pose. The loop between Lines 4-8 iteratively checks for neighboring *collisions*, and displaces the pre-push pose by a step $\vec{u}$. Line 8 updates the bounding box. The primitive returns pre-push pose p_{pre} and push vector $\vec{d}$. Certain implementation details are important to consider. The magnitude of $\vec{u}$ is kept sufficiently small ($5mm$) and there is a local minimum check if no progress is made, which can shrink ϵ if needed.

It should be noted that a robust ADAPTIVEPUSH implies there exists a clearance of $|\vec{d}|$ from the target pose away from the neighboring objects. If the dimensions of the container are exactly (or close to) the dimensions of the packing arrangement, such a clearance will not exist. An alternative outlined in Algorithm 3 leverages the compliance of the container walls to solve the low-clearance case.

Algorithm 2: ADAPTIVEPUSH

Input: Pointcloud X, target pose $\hat{p}^i$, expansion ε, raise height h, object o
Output: Push vector $\vec{d}$, pre-push pose p_{pre}

1 $p_{pre} \leftarrow$ RAISE($\hat{p}^i, h$)
2 $S \leftarrow$EXPAND(DIMENSIONS(o) , ε)
3 $B_{box} \leftarrow$ GENERATEBOUNDINGBOX(p_{pre}, S)
4 **while** $B_{box} \cap X \neq \emptyset$ ***or*** $MaxIters$ **do**
5 $\quad X_{collision} \leftarrow B_{box} \cap X$
6 $\quad \vec{u} \leftarrow$ GENERATEPUSHVECTOR($X_{collision}, p_{pre}$)
7 $\quad p_{pre} \leftarrow p_{pre} + \vec{u}$
8 $\quad B_{box} \leftarrow$ GENERATEBOUNDINGBOX(p_{pre}, S)
9 $\vec{d} \leftarrow$RAISE($\hat{p}^i, h$) $- p_{pre}$
10 **return** $(\vec{d}, p_{pre})$

Algorithm 3: TIGHTPUSHPLACE

Input: Pointcloud X, target pose $\hat{p}^i$, expansion ε, raise height h, container height H, object o
Output: Tight Push-place primitive Π

1 $p^{high} \leftarrow$RAISE($\hat{p}^i, H$)
2 $\Pi \leftarrow$ ADAPTIVEPUSH($X, p^{high}, \varepsilon, 0, o$)
3 $\Pi \leftarrow \Pi \cup$ ADAPTIVEPUSH(X, $\hat{p}^i, \varepsilon, h, o$)
4 **return** Π

Adaptive Pushing in Low-clearance Packing: Algorithm 3 outlines how repeated calls to Algorithm 2 can achieve packing in tight containers. $\hat{p}^i$ in such a case might have container walls surrounding it. The single push-primitive is first called on Line 2 for the object at the target pose raised by the container height H, which is clear of the surrounding objects, but still alongside the container wall. For poses along the edge of the container the first step pushes against the wall(s). For poses near the boundary, as in Fig. 4 (c,d,e), the first displacement (Line 2) pushes *into the walls of the container*. If the container is compliant enough this push can take the object away from local errors for safely lowering to the p_{pre} (computed on Line 3), and then the planar push towards the final pose $\hat{p}^i$.

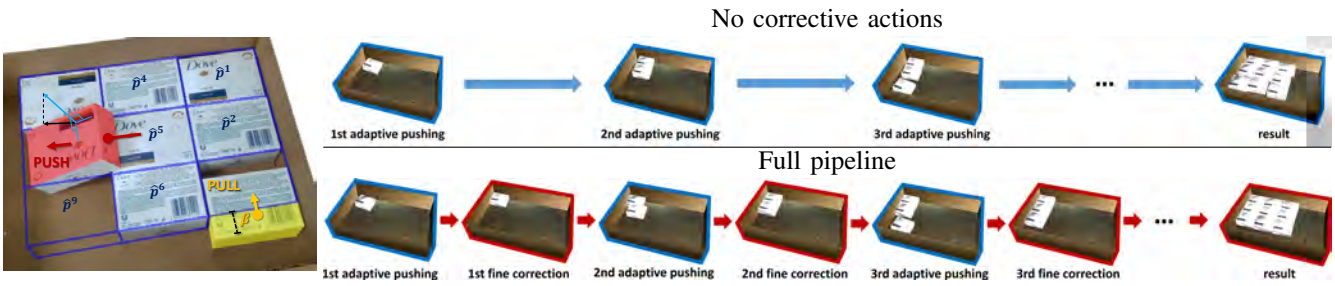


Fig. 5. (*Left:*) The two kinds of errors that invoke fine corrections. (*Right:*) Alternating adaptive push, and fine corrections for packing an object sequence.

D. Fine Correction using Push and Pull Primitives

The final primitive deals with the remaining failure cases. Fine corrections are required because objects can be placed in incorrect poses due to unexpected collisions as well as calibration and pose estimation errors. The proposed corrective manipulation procedure continuously monitors the scene and triggers corrective actions whenever necessary.

Algorithm 4: FINECORRECTION

Input: $\hat{A}_{\text{goal}} = \{\hat{p}^1, \ldots, \hat{p}^n\}$, threshold τ, corner point x_{pivot}, support surface normal $\vec{N}_s$

1 **repeat**
2 $\{X^i\}_{i=0}^n \leftarrow$TOPFACESEGMENTS()
// Sort centers of X^i based on increasing distance to x_{pivot}
3 $\mathcal{X} \leftarrow$SORTBYDISTANCE($\{X^i\}_{i=0}^n, x_{pivot}$)
4 **for** $X^i \in \mathcal{X}$ **do**
// Check if X^i is not horizontally placed
5 **if** $\exists x \in X^i,\ normal(x) \cdot \vec{N}_s \leq 1 - \epsilon$ **then**
6 $x_c \leftarrow center(X^i)$
// Push segment X^i from a side to translate its center x_c after projecting vector $normal(x_c)$ to the XY plane, and where α is a small constant
7 $\Pi \leftarrow$PROJECTXY$(normal(x_c), \vec{N}_s) \times \alpha$
8 EXECUTE(Π)
9 **break**
// Check if X[i] is horizontally offset
10 $\beta \leftarrow$HAUSDORFF$(\hat{p}^i, X^i)$
11 **if** $\beta > \tau$ **then**
// Pull X^i toward $\hat{p}^i$ with distance β
12 $\Pi \leftarrow$PULLTOWARDS$(X^i, \hat{p}^i, \beta)$
13 EXECUTE(Π); break
14 **until** *NoError* ***or*** *Timeout*

The proposed realignment process is illustrated in Algorithm 4. The process first removes the background, the box, the robot's arm, end-effector, and the side-faces of the objects from the observed point cloud. The remaining point cloud corresponds to only top surfaces of objects, and is segmented into disjoint subsets $\{X^i\}_{i=0}^n$, where X^i is the point cloud corresponding to object i. The observed point cloud is then compared against the desired alignment of the objects in their target poses $\{\hat{p}^1, \ldots, \hat{p}^n\}$. The algorithm iterates through the objects in the order of their increasing distances to a corner point x_{pivot}. Misalignment is systematically detected by comparing the observed point cloud X^i of each object to its desired one $\hat{p}^i$. As shown in Fig. 5, two types of misalignment errors can occur. The first type occurs when the top surface normal of an object, denoted by $normal(x_c)$, is not perpendicular to the support surface (line 5). This error is corrected by pushing the object along a direction and for a distance given as the projection of the surface normal on the support surface. This process is repeated until the surface normal becomes perpendicular to the support surface.

The second type of error happens when a peripheral object is not entirely within the desired footprint of the pile (line 11). The proposed procedure systematically detects pivot points that are outside the desired footprint of the pile and pulls the objects inside accordingly (line 12). The correction is repeated until the point cloud is aligned with the desired goal poses, within a given threshold τ, or a timeout occurs.

V. SIMULATION FRAMEWORK

A simulation framework is developed to model the physical aspects of real-world tight packing. Given the focus of developing and evaluating sensor-based control algorithms, such as *adaptive pushing* and *fine-correction*, the framework simulates an RGB-D sensor to capture the state of the process. Fig.6 shows the simulation setup in Blender[4].

The software framework loads a model of the *Bin* into the simulation environment. A *point lighting source* and a *camera* are initialized based on the parameters specified in a configuration file. A target packing sequence is specified for objects with known 3D models as $\{(o^i, \hat{p}^i) | i = 1 : n\}$, where o^i identifies the object and $\hat{p}^i$ denotes the desired target pose of the object. An example of the desired configuration is shown in Fig.6 (left).

An intermediate simulation state corresponds to the partial placement of objects in the bin. The achieved placement might be inaccurate due to the noise (artificially added in the simulation framework) from execution or sensing. The resulting state is captured via an RGB-D rendering of the scene. The simulation can either be set to render images via a fast rasterization-based rendering process or to generate photo-realistic images via physically based rendering. In addition, the framework generates depth images, instance segmentation masks and consequently the 3d point cloud data. This state information is provided as input to *control algorithms*, such as adaptive pushing or fine correction. The control algorithms generate motions of the end-effector for

[4] http://www.Blender.org

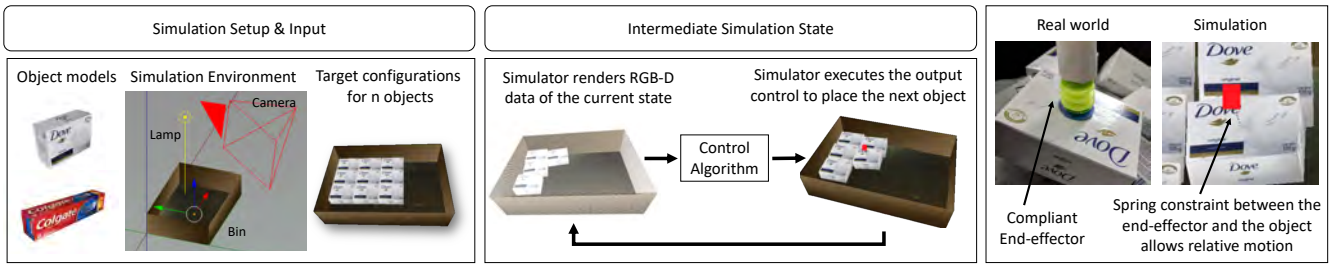


Fig. 6. The simulation environment is set up with a bin, lights and camera parameters. Objects are placed sequentially to achieve a desired target configuration. Before each placement, the state of the bin is captured by rendering RGB-D images. The state is provided as input to the control algorithm that computes the motion for the target object. Finally, the controls are executed in the simulator to place the object in the bin. To accurately model the compliance of the real-world end-effector, a spring constraint is applied in simulation between the end-effector and the object. The constraint allows small relative motion and prevents unrealistic collision and jumping of objects that are often observed in rigid-body simulation.

the placement of the target object which are then physically simulated via the Bullet Physics Engine[5].

The simulator reasons about the physical interactions between the end-effector and the object being manipulated as well as the interactions between different objects. The simulator is developed to mimic the compliance of the real-world setup as shown in Fig.6 (right). This is achieved by introducing a *spring constraint* between the end-effector and the object being manipulated. The spring constraint allows the link between the end-effector and the object to be stretched during the packing operation, which is similar to the behavior achieved by the soft suction gripper. In the absence of this constraint, i.e. in a strictly rigid simulation, the objects would pop out to resolve the collision.

The simulation framework is publicly released with this work and can be used for two purposes. The first is for developing and evaluating sensor-based control algorithms as in the case of the current work. This allows extensive evaluation and ablation studies of algorithms and environments with varying parameters, as well as a variety of objects and configurations without the need for an expensive robotic setup. The second use-case is in the context of a learning-based control algorithms, such as reinforcement learning, where this framework can be used to generate a large-scale training dataset. The framework gives access to photo-realistic image rendering as well as 3D point cloud data. It also generates corresponding instance masks and pose labels. This state information can be associated with the corresponding action sequences. Furthermore, the framework allows domain randomization over different parameters of the environment, which can help transfer learning and deployment in real-world setups.

VI. Evaluation

To evaluate the performance of the proposed pipeline, extensive experiments were executed both in simulation and on a real robotic platform, which encompasses the motivating complications that arise in real-world setups.

A. Hardware and Software Setup

First, this section describes the hardware setup, the workspace design and software choices, which influence the baseline pipeline and the proposed manipulation primitives:

[5]`http://www.bulletphysics.org`

a) Hardware Setup:: The robot is a `Kuka IIWA14` 7-DoF arm (Fig. 1 middle). A customized end-effector solution extrudes a cylindrical end-effector that ends with a compliant suction cup. Two *RealSense* SR300 cameras are mounted on a frame and pointed to the $\mathcal{B}_{init}$ and $\mathcal{B}_{goal}$ containers from the other side relative to the robot as Fig. 1 (right) shows. The cameras' frame is statically attached to the robot's base such that calibration errors are minimized in estimating the camera poses in the robot's coordinate system.

b) Workspace Design:: Fig. 1 (right) shows the setup designed for the target task. The annular blue region represents the subset of the reachable workspace that allowed for top-down picks with the robot's end-effector. This region is computed by extensively calling an inverse kinematics (IK) solver for top-down picks with the end-effector. The IK solutions indicate that the radial region between $40cm$ and $70cm$ from the robot center maximizes reachability and IK solution success given the setup. The bins (red rectangles) are placed so that they lie inside the optimal reachable region.

c) Software Dependencies: `MoveIt!` [38] is used for motion planning. Most of the motions are performed using *Cartesian Control*, which guides the arm using end-effector waypoints. Ensuring the motions occur in reachable parts of the space increases the success of *Cartesian Control*, simplifies motion planning and speeds-up execution. To decrease planning time, motion between the bins is precomputed using `RRT`* [39] and simply replayed at appropriate times.

B. Experimental Design

The experiments are designed to showcase the hardness of tight packing as well as the benefits of adding robust environment-aware manipulation primitives that aid in increasing success rate and accuracy. To evaluate the performance of the primitives over different noise levels and to study the sensitivity of the primitives with respect to the underlying parameters, extensive experiments were executed on a physics-based simulator. The output of *real-world experiments* and *simulation experiments* are evaluated over three distinct metrics:

Volumetric Error: This measures the percentage of unoccupied volume within the ideal target placement volume. This is measured by computing an occupancy grid based on the sensing data acquired after the objects have been moved to the target bin. Given the partial observation, an

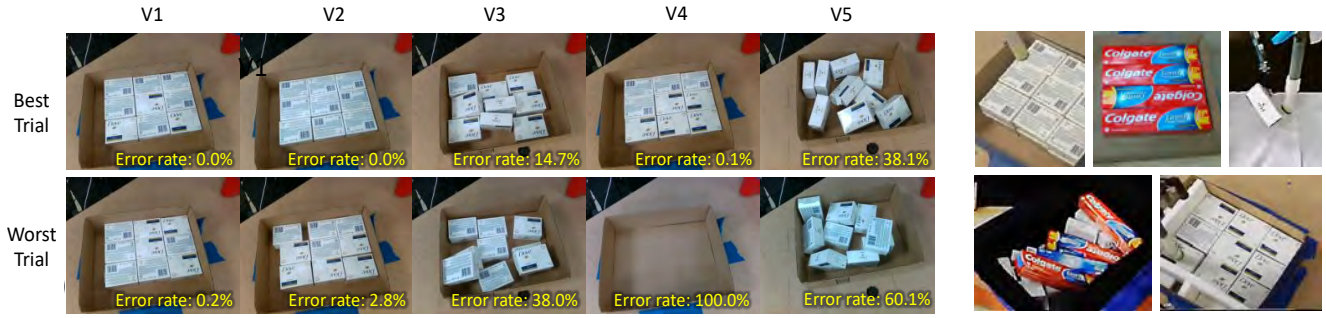


Fig. 7. The final set of object poses in the target bin at the end of every experiment. Different columns represent different versions. The *Left top row* is the best case, and the *Left bottom row* is the worst case. (*Right:*) Demonstrative real-world trials performed with the same pipeline for (*from top left*) multi-layer packing, different objects, narrow-face toppling, heterogeneous piles, and no-clearance packing.

approximation of the volume is computed by projecting the sensed points down to the bottom surface of the bin and marking all voxels along the projection as occupied.

ADI Error [35] : This is often used in the pose estimation literature. Given two poses, the ADI metric measures the average distance between corresponding points on the object model placed at the two poses.

Pose Recall: A pose is assigned to each of the object placements by aligning a 3D model of the object to the observed point cloud data. This pose is then compared to the closest target pose to compute an ADI error. A pose is considered correct if the ADI error is less than a distance threshold. Pose recall curves indicate the number of successful object placements that are within different threshold margins.

C. Real-world Packing Trials

For consistency, an identical version of the problem is tested, with "dove soap bars" that are randomly thrown into the source bin $\mathcal{B}_{init}$, which is placed on one side of the robot's reachable workspace (Fig 1). Only top-down grasps are allowed within a given alignment threshold. The start arrangement A_{init} of objects is intended to reflect a random pile, with 10 repetitions of each experimental condition. The target bin $\mathcal{B}_{goal}$ contains a 3×3 grid arrangement of 9 objects, on the same plane, with the stable face of the object targeted for placement. The complete pipeline uses a) corrective actions for fine adjustments, b) push-to-place actions for robust placement, c) toppling actions for increasing successes, and d) pose estimation for adjusting the object. The improvements introduced by these strategies are evaluated through the following comparison points, within the context of the proposed pipeline:

V1 - Full pipeline: The complete pipeline with all the primitives achieves the highest accuracy and success rate.

V2 - No corrective actions: The experiment corresponds to **V1** without the fine correction module of Fig. 5.

V3 - No push-to-place actions: This version is **V2** without the use of the robust placement module (Fig. 4) that performs push actions to achieve robust placement.

V4 - No toppling actions: These experiments used **V2** without considering toppling actions to deal with objects not exposing a valid top surface that allows the target placement.

V5 - (Baseline) No push-to-place, toppling, pose-estimation: The naive baseline that solely uses a pose-unaware grasping module that reports locally graspable points and drops the grasped object at an end-effector pose raised from the center of the desired object position, with no adjustment in orientation.

The metrics evaluated include the fraction of successful object transfers that succeed in moving objects to the target bin. The accuracy is captured in the threshold mentioned in Eq. (1) that is expressed in terms of a percentage of unoccupied volume within the ideal target placement volume. This was measured with a voxel discretization sufficient to elucidate the difference between the methods. The average data recorded is reported in Fig. 7 (*left*). This error measure is proportional to the accuracy. The key observations from these experiments, regarding each variant, are detailed as follows.

The low error for **V1** corroborates the final bin placement evidence. On average, seven corrective actions per experiment were invoked to achieve the high degree of accuracy.

The accuracy improvement obtained from corrective actions is evaluated in **V2**. While this version succeeded in dropping all the objects close to the correct target poses, application use-cases where a higher degree of accuracy is desired motivate the use of corrective actions. The integration of the corrective actions was done with higher error threshold during intermediate steps, and a much finer one for the final adjustment. Errors can typically arise from execution failure and pose misalignments. The less accurate these underlying processes are, the more important corrective actions become.

V3 only performs adjustments using pose estimation, and toppling. While, this is sufficient to successfully transfer all the objects, any difference of accuracy to **V2** would be introduced by the lack of push-to-place actions. Here, there is complete reliance on the exactness of the execution and pose adjustments. Due to the proximity of adjacent object surfaces in the target grid arrangement, even minor errors get aggravated. However, due to the ability to reason about toppling, all the objects can be transferred to the target bin, even with this low accuracy. This is demonstrated in the occurrence of the failure to transfer all the objects.

In **V4**, any object that does not expose a permitted picking surface that makes the prehensile placement possible, is not picked. Any instance of the source bin that ends with no

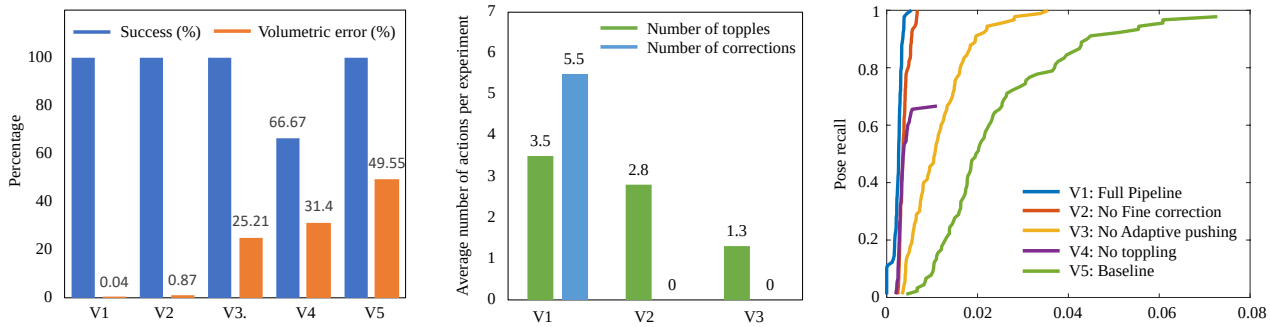


Fig. 8. (*Left*): The blue bars represent the fraction of successful object transfers. The orange bars represent the percentage of unoccupied volume within the ideal target placement volume. (*Middle:*) The blue bar represents the average number of correction actions happened per experiment. The green bars represent the average number of toppling actions occurred per experiment. (*Right:*) The pose recall percentage over different thresholds is shown for the pipeline versions.

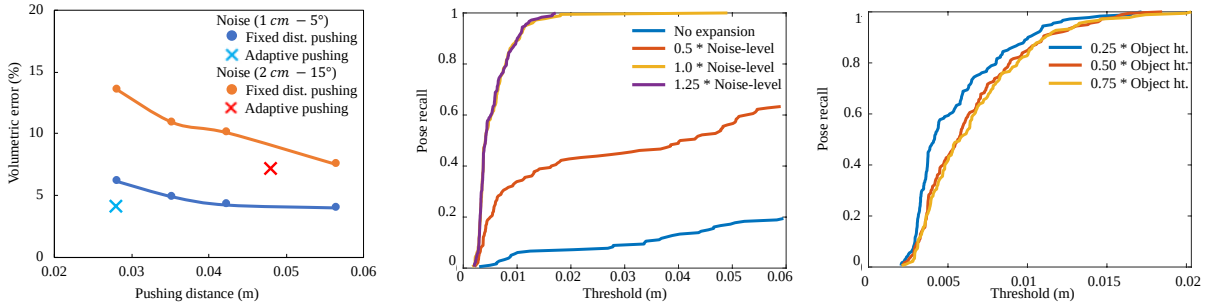


Fig. 9. (*left*) Volumetric error (ADI) of proposed adaptive pushing primitive compared with baseline method under two noise levels. Pushing distance is a parameter for the baseline method. Pose recall against the parameter for adaptive pushing. (*middle*) pose recall against the expansion size of object model, when the expansion size is above the noise level, the performance is similar; as the expansion size decreases below noise level, the performance becomes worse. From the 0 noise level to 1.25 times noise level, the successful rates are $7.2\%, 42.2\%, 99.4\%, 100\%$, *(right)* pose recall against height of pre-push pose, it shows that the performance drops rapidly after the height of pre-push pose increases over 1/4 of object height.

such objects results in no valid picking actions that can make the approach proceed, and a failure is thus declared. The current behavior of **V4** drops the object if it is mistakenly grasped from the wrong surface. This can itself be used as a naive toppling primitive. It is important to note that there might be other alternative strategies that can deal with this failure, but the intent of this comparison is to demonstrate the importance of having a deliberate toppling strategy in the pipeline, that can change the object's orientation in the context of random starting arrangements of the object. On average, the toppling primitive was required four times per experiment in **V1**, **V2**, and **V3**. This highlights the necessity of this reasoning. Deliberate toppling however requires at least one additional pick action. The number of pick attempts per successful object transfer was 2.56 for **V4**, whereas, in **V2** the same was 1.98. This indicates that toppling is indeed necessary both in terms of success and efficiency of actions.

Expectedly, **V5** has the lowest accuracy. However, since there is no reasoning about the pick surface, every object was transferred to the space of the bin. This has no guarantee to work if the object is larger. This drives the motivation for using a set of robust primitives for the packing problem.

Overall, the time for the experiments shows a trend of increasing with the increasing complexity of the pipeline. The trade-off of accuracy versus time persists. On average, **V1** ran for 945s per bin while **V5** ran for 323s.

D. Real-world Demonstrations

Various demonstrative packing tasks, highlighted in Fig 7 (*right*) were executed to showcase the versatility of the proposed pipeline.

Multi-layer packing: Objects can also be packed in multiple layers. Running the method beyond the first plane of the grid shifts all the operations to the higher plane. This is demonstrated in a standalone run of the attached video.

Different objects: To validate the method's applicability to cuboid objects of different dimensions, **V1** was performed for toothpastes. Over 5 experiments, with 4 objects, every run succeeded in placing the objects inside the bin.

Narrow-face reorientation: The packing orientation is tested with the narrow (unstable) face of the cuboid being the contact surface. Here toppling has to reorient the object from its more stable wider face to the desired narrow face.

Heterogenous pile: The source bin was filled up with two different kinds of objects and the pipeline was executed for each object sequentially, i.e., pack the *toothpaste* first, and then the *soap*, into two separate containers. The method is correctly capable of retrieving, and packing the desired objects from such a heterogeneous pile.

No-clearance Packing: The target bin is resized to be of the exact dimensions as the cross section of the target arrangement. This leaves no clearance for placement along the edges and corners of the 3×3 grid. The compliance of the end-effector, and the container's walls are leveraged by executing the meta-primitive of Algo 3.

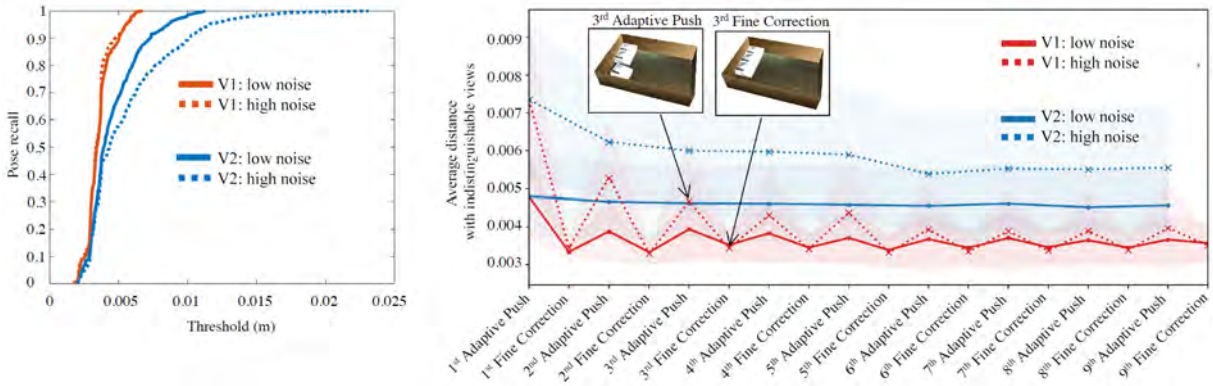


Fig. 10. Pose recall (left) and average distance error (ADI) against the number of placed objects (right) with and without the fine correction module under different noise levels.

E. Ablation Studies in Simulation

The simulator described in Sec V is used to perform ablation studies for both adaptive pushing, and fine correction.

Noise Level: Real world trials introduce errors due to pose estimation, execution, calibration, and non-prehensile interactions. All of these cause a disparity between the objects' ground truth poses versus where the pipeline models it to be. The objective of adaptive pushing and fine correction is to account for this difference. In order to approximate the noise introduced by all of these sources of error, two noise levels are evaluated with the adjustment primitives: the lower noise level uniformly samples up to a $1cm$ planar translation offset and up to a $5°$ deviation to the object's orientation given the ground truth. The higher noise level applies up to a $2cm$ translation, and $15°$ orientation. This means that the simulator changes the *ground-truth* values given the noise level for perception purposes, while executing the manipulation operations over the unperturbed poses.

Adaptive Pushing: Simulated experiments examine the effectiveness of the proposed adaptive pushing primitive on reducing the packing error. The comparison is between the proposed method and a simple baseline method. For each new object to place, given the target pose, the baseline method calculates a pre-push pose using a *constant* displacement. During the simulation, noise (sampled from either of the above two levels) is applied to the pre-push pose. In the experiment, the average volumetric error, and displacement computed from the proposed method is compared against the baseline method with different fixed displacements as parameters. The result is shown in Fig 9 (*left*). The results show that as the pushing distance increases for the baseline method, its performance improves, and is better than adaptive pushing beyond $5cm$. Nevertheless, this means such a large clearance would be required between the dimensions of the container and the packing arrangement. Therefore the adaptive pushing, which achieves better performance for a lower average pushing displacement is more amenable to tighter packing scenarios.

To further evaluate the adaptive pushing primitive, the influence of the expansion parameter ε from Algorithm 2 is examined. The evaluation studies the effect of this parameter on the performance of the primitive. The result is shown in Fig. 9 (*middle*). For the expansion size, different expansions relative to the maximum translation noise level and its effect on pose recall were studied. It is observed that once the expansion size drops below the noise level, the pose recall curve drops drastically. The lower expansion curves achieve low success due to situations where the pre-push pose is in collision with neighboring objects, triggering a failure of that run (remainder from 9 objects). From the no expansion to 1.25 times the noise level, the success rate increases from 7.2% to 100%. This motivates using a large enough expansion ε that is larger than the estimate for the noise that errors in the system can introduce.

For the height of the pre-push pose (h), different values relative to the object's height were checked for how the pose recall changes. The result is shown in Fig. 9 (*right*). The primitive is less sensitive to the value of h. Performance is best for a fraction (0.25) of object height.

Fine Corrections: The fine correction module was evaluated on the same simulation setup and noise levels. The following alternatives were compared: **V1**, which includes adaptive pushing and the fine correction modules, and **V2**, which only performs adaptive pushing. Fine corrections are repeated at most 4 times per object until the point cloud is aligned with the desired goal poses within a threshold of $5mm$. An expansion of 1.25 times noise level was used since it achieved 100% pose recall. In this section, the focus is on how the fine correction module can improve the *average distance error*. Fig 10 (left) shows the pose recall rate under different noise levels. In this experiment, an average of $1.03(\pm 0.80)$ corrective actions is executed per placement. Without the corrective module (**V2**), the pose recall declines substantially as the noise level increases. **V1**, however, preserves a high recall rate regardless of noise level. Fig 10 (right) shows the distance error (ADI) averaged over the number of placed objects. The error drops periodically after each execution of the fine correction module in **V1**, while the same error remains mostly unchanged when **V2** is used instead of **V1**. The fine correction module not only corrected the misplacement due to noise, but also prevented accumulative misalignment. The quality improvements increase with the increase in noise, thereby motivating fine corrections where larger amounts of noise is expected.

VII. Discussion

The proposed pipeline indicates intriguing nuances of the packing problem. The use of a minimal, suction-based end-effector is a cost-effective, simple and relatively robust way to pick objects but does not easily allow for complex grasp reasoning, regrasping, or within-hand manipulation. The proposed pipeline achieves high accuracy by leveraging the compliance of the suction cup and the environment, while the object is attached. It uses robust reasoning to incrementally correct errors, instead of compounding them. While it can be argued that better baseline components can be developed to minimize uncertainty, the overall philosophy of robust, minimal, and compliant reasoning remains unchanged.

The proposed system and primitives can also deal with cubic objects with different sizes and can be extended to non-cubic objects by adapting the object models. The key adaptation corresponds to identifying an appropriate packing arrangement $\hat{A}_{\text{goal}}$ (potentially labeled in this case) in the target bin and the corresponding picking order from the initial bin. Future work will also explore speeding up performance and dealing with algorithmic challenges: the combinatorial reasoning over possible placements, physics-based reasoning to further improve pushing and toppling, as well as extending to more adaptive end-effectors. The platform can also be utilized as a training testbed for reinforcement learning to automatically discover robust primitives for solving packing tasks.

References

[1] J. J. Enright and P. R. Wurman, "Optimization and Coordinated Autonomy in Mobile Fulfillment Systems," in *AAAIWS'11-09*, 2011.

[2] F. Wang and K. Hauser, "Stable bin packing of non-convex 3d objects with a robot manipulator," in *IEEE ICRA*, 2019, pp. 8698–8704.

[3] ——, "Robot packing with known items and nondeterministic arrival order," in *R:SS*, 2019.

[4] A. Sahbani, S. El-Khoury, and P. Bidaud, "An Overview of 3D Object Grasp Synthesis Algorithms," *RAS*, vol. 60, no. 3, 2012.

[5] K. Shimoga, "Robot grasp synthesis algorithms: A survey," *IJRR*, vol. 15, no. 3, pp. 230–266, 1996.

[6] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-Driven Grasp Synthesis: A Survey," *IEEE TRO*, vol. 30, no. 2, pp. 289–309, 2014.

[7] A. Boularias, J. A. Bagnell, and A. Stentz, "Learning to manipulate unknown objects in clutter by reinforcement," in *AAAI*, 2015.

[8] A. Kimmel, R. Shome, and K. E. Bekris, "Anytime motion planning for prehensile manipulation in dense clutter," *Adv. Robotics*, 2020.

[9] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Analysis and Observations from the 1^{st} Amazon Picking Challenge," *IEEE T-ASE*, 2016.

[10] M. Schwarz, C. Lenz, G. M. García, S. Koo, A. S. Periyasamy, M. Schreiber, and S. Behnke, "Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing," in *IEEE ICRA*, 2018, pp. 3347–3354.

[11] D. Morrison, A. W. Tow, M. McTaggart, R. Smith, N. Kelly-Boxall, S. Wade-McCue, J. Erskine, R. Grinover, A. Gurman, T. Hunn, *et al.*, "Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge," in *IEEE ICRA*, 2018, pp. 7757–7764.

[12] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *IEEE ICRA*, 2018, pp. 1–8.

[13] Z. Littlefield, S. Zhu, C. Kourtev, Z. Psarakis, R. Shome, A. Kimmel, A. Dobson, A. Ferreira De Souza, and K. E. Bekris, "Evaluating end-effector modalities for warehouse picking: A vacuum gripper vs a 3-finger underactuated hand," in *IEEE CASE*, 2016.

[14] C. Rennie, R. Shome, K. E. Bekris, and F. A. De Souza, "A dataset for improved rgbd-based object detection and poe estimation for warehouse pick-and-place," *IEEE RA-L*, 2016.

[15] D. Morrison, J. Leitner, and P. Corke, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," in *R:SS*, 2018.

[16] J. Mahler and K. Goldberg, "Learning deep policies for robot bin picking by simulating robust grasping sequences," in *CoRL*, vol. 78, 2017, pp. 515–524.

[17] M. Dogar and S. Srinivasa, "A Framework for Push-Grasping in Clutter," in *R:SS*, 2011.

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

[19] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *IEEE ICRA*, 2015.

[20] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push Planning for Object Placement on Cluttered Table Surfaces," in *IEEE IROS*, 2011.

[21] K. M. Lynch, "Toppling Manipulation," in *IEEE ICRA*, 1999.

[22] N. Chavan-Dafle and A. Rodriguez, "Prehensile Pushing: In-Hand Manipulation with Push-Primitives," in *IEEE/RSJ IROS*, 2015.

[23] C. Correa, J. Mahler, M. Danielczuk, and K. Goldberg, "Robust Toppling for Vacuum Suction Grasping," in *IEEE CASE*, 2019.

[24] X. Cheng, Y. Hou, and M. T. Mason, "Manipulation with Suction Cups Using External Contacts," in *ISRR*, 2019.

[25] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," in *R:SS*, 2018.

[26] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6d Object Pose Estimation using 3D Object Coordinates," in *ECCV*, 2014.

[27] F. Michel, A. Kirillov, E. Brachmann, A. Krull, S. Gumhold, B. Savchynskyy, and C. Rother, "Global Hypothesis Generation for 6D Object Pose Estimation," in *CVPR*, 2017.

[28] C. Mitash, A. Boularias, and K. E. Bekris, "Robust 6D Object Pose Estimation with Stochastic Congruent Sets," in *British Machine Vision Conference*, 2018.

[29] V. Narayanan and M. Likhachev, "Discriminatively-guided Deliberative Perception for Pose Estimation of Multiple 3D Object Instances," in *R:SS*, 2016.

[30] C. Mitash, B. Wen, K. Bekris, and A. Boularias, "Scene-level Pose Estimation for Multiple Instances of Densely Packed Objects," in *CoRL*, 2019.

[31] J. O. Berkey and P. Y. Wang, "Two-Dimensional Finite Bin-Packing Algorithms," *Operational Research Society*, vol. 38, no. 5, 1987.

[32] S. Martello, D. Pisinger, and D. Vigo, "The Three-Dimensional Bin Packing Problem," *Operations Research*, vol. 48, no. 2, 2000.

[33] S. Albers and M. Mitzenmacher, "Average-case Analyses of First Fit and Random Fit Bin Packing," in *ACM SODA*, 1998, pp. 290–299.

[34] 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 ICRA*, 2019, pp. 9007–9013.

[35] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," in *ACCV*, 2012.

[36] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *ICCV*, 2017.

[37] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *IROS*, 2017.

[38] S. Chitta, I. Sucan, and S. Cousins, "Moveit!" *IEEE RAM*, 2012.

[39] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *IJRR*, vol. 30, no. 7, June 2011.