Autonomous Robot Packing of Complex-shaped Objects

by

Fan Wang

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Kris Hauser, Supervisor

_____
Missy Cummings

_____
Miroslav Pajic

_____
Martin Brooke

_____
Aaron Franklin

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering in the Graduate School of Duke University
2025

Abstract

Autonomous Robot Packing of Complex-shaped Objects

by

Fan Wang

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Kris Hauser, Supervisor

_____
Missy Cummings

_____
Miroslav Pajic

_____
Martin Brooke

_____
Aaron Franklin

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering
in the Graduate School of Duke University
2025

Abstract

With the unprecedented growth of the E-Commerce market, robotic warehouse automation has attracted much interest and capital investment. Compared to a conventional labor-intensive approach, an automated robot warehouse brings potential benefits such as increased uptime, higher total throughput, and lower accident rates. To date, warehouse automation has mostly developed in inventory mobilization and object picking.

Recently, one area that has attracted a lot of research attention is automated packaging or packing, a process during which robots stow objects into small confined spaces, such as shipping boxes. Automatic item packing is complementary to item picking in warehouse settings. Packing items densely improves the storage capacity, decreases the delivery cost, and saves packing materials. However, it is a demanding manipulation task that has not been thoroughly explored by the research community.

This dissertation focuses on packing objects of arbitrary shapes and weights into a single shipping box with a robot manipulator. I seek to advance the state-of-the-art in robot packing with regards to optimizing container size for a set of objects, planning object placements for stability and feasibility, and increasing robustness of packing execution with a robot manipulator.

The three main innovations presented in this dissertation are:

1. The implementation of a constrained packing planner that outputs stable and collision-free placements of objects when packed with a robot manipulator.

Experimental evaluation of the method is conducted with a realistic physical simulator on a dataset of scanned real-world items, demonstrating stable and high-quality packing plans compared with other 3D packing methods.

2. The proposal and implementation of a framework for evaluating the ability to pack a set of known items presented in an *unknown order of arrival* within a given container size. This allows packing algorithms to work in more realistic warehouse scenarios, as well as provides a means of optimizing container size to ensure successful packing under unknown item arrival order conditions.

3. The systematic evaluation of the proposed planner under real-world uncertainties such as vision, grasping, and modeling errors. To conduct this evaluation, I built a hardware and software packing testbed that is representative of the current state-of-the-art in sensing, perception, and planing. An evaluation of the testbed is then performed to study the error sources and to model their magnitude. Subsequently, robustness measures are proposed to improve the packing success rate under such errors.

Overall, empirical results demonstrate that a success rate of up to 98% can be achieved by a physical robot despite real-world uncertainties, demonstrating that these contributions have the potential to realize robust, dense automatic object packing.

Acknowledgements

First and foremost, I would like to thank my advisor Dr. Kris Hauser for his continuous support and guidance throughout my Ph.D., and for providing me with the independence and freedom to work on a variety of research problems. I have been incredibly fortunate to work with Kris, an excellent role model who inspired me to pursue a life-long career in robotics.

I would also like to thank my dissertation committee members Dr. Missy Cummings, Dr. Martin Brooke, Dr. Pajic Miroslav, and Dr. Aaron Franklin, for their support and great advice. I am tremendously grateful for my friends and colleagues from the Duke Intelligent Motion Lab (Now the Intelligent Motion Lab at UIUC), including Yifan Zhu, Gao Tang, Shihao Wang, João Marcos Correia Marques and many others. It has been a great journey pursuing a Ph.D. with all of you, and I continue to learn and get inspired by many of you daily.

I would also like to thank the members of Team Duke when participating in the Amazon Robotics Challenge 2017 during my first year as a Ph. D. student, particularly Mark Draelos, Brenton Keller, and Anne Draelos. Taking part in the challenge together was both a thrilling and humbling learning experience, and you all have inspired me to be a more exceptional Ph.D. student.

I am also very fortunate to have had the opportunity to be supported and work as an intern at Amazon Robotics with Jane Shi, an inspiring mentor and friend. During my internship, I enjoyed working with many team members, including Joey

Durham, Chaitanya Mitash, and Sachal Dillion.

Finally, I would like to thank my partner Haibei Zhu for always being there and for putting my needs in front of his. I would also like to thank my parents and other family members for their unconditional love and support.

Contents

List of Figures

List of Tables

1

Introduction

With the fast expansion of the E-Commerce market, modern warehouses are handling more frequent shipments and more complex deliveries. Thus, the demand for warehouse automation technology is high. According to a Westernacher white paper [wes17], worldwide spending on warehouse automation technology is expected to reach $22.4 billion in market value by the end of 2021. Such demand is driven by the many advantages of automated warehouses. Compared to a conventional labor-intensive approach, an automated warehouse holds the promises of increased safety and more efficient operation. Among different warehouse automation technologies, robotic automation allows for fast deployment and high mobility, attracting much research interest and capital investment.

Robotic automation aims to expedite the transport of goods and streamline laborious processes such as object picking, sorting, and packing. This level of automation requires systems to scan and identify packages and individual goods, perform pick-and-place tasks for objects of various shapes and sizes, and sometimes handle complex decision logic. Developments in robotic automation are closely tied to advances in robot *perception* and *manipulation*, and significant progress has been made in recent

years in both areas.

Robot perception has witnessed tremendous progress in the last decade, largely due to the success of applying deep learning in computer vision tasks. Deep convolutional neural networks have now taken over most established rule-based algorithms and have dominated benchmarks in object classification [HZRS15a, TL19], object detection [TPL19, RHGS15, Gir15, ZWZ+20], object segmentation [RFB15, HGDG17, LDG+17], and many others [CHS+18, QSMG16, COR+16]. Deep learning methods show promise in achieving or surpassing human-level performance in many significant sensing problems. For example, human-level performance on object recognition for selected categories (e.g., ImageNet classes) is demonstrated by recent works[HZRS15c, SLWT15]. These advances in robot perception have shown potential in a commercial setting. For example, the successful establishments of Amazon Go stores and full-size grocery stores are largely related to powerful perception algorithms [Gro19]. They demonstrate that highly accurate detection and classification can be generalized to tens of thousands of unique items, which is an essential component of an automated warehouse.

At the same time, robots continue to gain manipulation dexterity. Several factors drive this enhancement. One is advances in hardware such as higher resolution tactile sensing [WLM+16, YDA17, TLK+09] and robot end-effectors that offer better precision or compliant control. Another is the development of realistic simulation environments [Hau20, Dra05, CB19], that run in real-time and above on modern computer architectures. By using these simulated platforms, one may perform testing and data-collection for data-driven methods that are magnitudes faster than doing so on real, physical systems. Finally, the emergence of reinforcement learning (RL) in robotics holds the promise of enabling autonomous robots to learn vast behavioral skills with minimal hand-engineered rules [LLS15, MBM+16, MKS+15, SLM+15, GLSL16], although applications of direct reinforcement learning algorithms have thus

far been restricted to simulated settings and relatively simple tasks [GLSL16].

Despite these advances, it remains a challenge to design an automated robot system for warehouses, particularly for a system that operates directly on an individual object. Interests in robust and versatile robotic manipulation for a wide variety of objects is almost as old as robotics. However, most robotic automation success is restricted to known objects in controlled environments with specialized hardware. A modern warehouse handles millions of unique objects, varying widely in size and appearance, ranging from loose objects wrapped in plastic, such as clothing, to soft objects like plush toys, as well as solid but highly concave objects like bowls and cups. This variety presents significant challenges for robot vision and manipulation. To date, robot tasks that require coordination of perception, sensing, and manipulation on complex objects have shown only limited success. While there have been a number of robots reported as being able to load dishware into dishwashers [JZLS12], to make pancakes [BKK+11], to assemble furniture [KLRR13], and to perform general pick-and-place in cluttered homes [CJCH12a], amongst other tasks [Sch18, WHLC10, WCH18], the execution is usually slow and the results are far from optimal. For example, a state-of-the-art autonomous laundry folding robot folds a single towel in 15 minutes [Sch18], a task humans can effortlessly do in seconds. A sophisticated grasping system such as the Google grasping project [LPKQ16], which trains industrial robot arms to grasp novel objects in a cluttered container, reported to have collected over 800,000 grasp attempts over two months, using between 6 and 14 robotic manipulators at any given time, achieves an 80% success rate among 100 grasping attempts.

Given that current technologies do not offer high reliability and efficient operations, which are critical for warehouse automation, this dissertation proposes theorems and algorithms that tackle a specific problem: **to pack objects of arbitrary shapes and weights into a single shipping box**, which belongs to a sub-area of

robotic warehouse automation called *dense robot packing*. In addition, I design and implement a robotic system that realizes dense automatic object packing with high reliability.

1.1 Warehouse Automation

Warehouse operations involve a complicated system that can be divided into six primary steps: receiving, sorting, storing, picking, packing, and shipping, shown in Fig. 1.1. Efforts in warehouse automation have been spent on optimizing these six processes 1) to reduce the sheer size of a warehouse facility and to speed up transportation of goods within it, and 2) to automate labor-intensive and repetitive operations.

Large areas are needed for warehouses to store items in racks, to move stock, to unload and load trailers and containers, and to allow people to pick from them. At the same time, moving items and people around inside such a large space becomes time-consuming. By the earlier 2000s, warehouses used largely mechanized automation to increase storage density and transportation efficiency. Examples include the use of Carousel, a rotatable circuit of shelving that allows the product to rotate to the person, A-frames, an automated dispensing machine that drops items onto a conveyor [BH10], as well as Automated Pallet Stacking and Destacking technologies and Autonomous Storage-and-Retrieval systems [AdKR17], and many others. Among these techniques, the usage of Autonomous Storage-and-Retrieval(AS/R) systems gives a big boost to storage density and transportation efficiency. These systems use racks with aisles and deploy autonomous shuttles that operate to transport inventory around the warehouse, and therefore allow for aisles that are extremely narrow [BH10].

However, one big disadvantage of mechanized automation is inflexibility: it performs well for a specific task for which it was designed, but becomes very expensive

to be adapted for changes [BH10].

Later in the 2000s, with advances in sensor technologies and robotics, mobile robots such as automated guided vehicles(AGVs) [CLR07, GKMS08, USSB97] and autonomous mobile robots(AMRs) [Hag04, Wan91, FM98] were employed to move inventory shelves around in a warehouse and present inventory for an operator to retrieve. The inventory could include storage racks, packages, or a tote of items. Mobilized storage racks can be stored at a higher density as there is no need to leave room in the aisle for a human to pass through. The fact that the robot brings the inventories to the operator instead of having the operator walk to them also increases the base pick rate. Compared to earlier warehouse automation technologies that relied on networks of conveyor belts and chutes, robotic automation allows for faster deployment and greater mobility.

The last ten years have seen a tremendous transformation in warehouse operation, driven by the fast growth of the e-commerce market. Many warehouses moved to operate 24/7, handle large and variable daily order volumes, and store millions of unique items. These movements have raised the demand for more comprehensive and sophisticated robotic automation. The hope is to fulfill a customer order fully automatically, from sorting inbound inventory to picking and packing each individual ordered item. These processes are great candidates for automation: They are laborious and repetitive, and often suffer from poor ergonomics.

This demand for a higher level of automation first drew broad research attention through the Amazon Robotics Challenge (ARC) [Rut15], held yearly from 2015-2017. The rule of the competitions tested the ability of robotic systems to fulfill an order by autonomously picking the requested items from a tote and placing them into a storage unit, and picking the items from the storage unit and packing them into a shipping box, which is a reasonably good representation of the key challenges in a warehouse setting. Some state-of-the-art pick-and-place system prototype such as [YFD+16,

SLG+18, ZSY+18] have resulted from the competition. Those systems can recognize and grasp both known and novel objects in relatively cluttered environments to a degree and place them in over-sized storage space.

Industry soon followed suit, such that by 2020 there were 278 robot warehouse automation companies opened globally [Tra20], many of which are tech start-ups. Some preliminary commercial success has been demonstrated: The number of robotic units in warehouses is expected to reach 620,000 units by 2021 [wes17], and many of those units are AGVs and AMRs, made by companies such as Kiva Systems, Quicktron, and Geek+. Recent start-ups have begun developing object picking/grasping systems that can handle individual objects. For example, RightHand Robotics developed a range of multi-finger and vacuum grippers with sensors for more general-purpose grasping. IAM Robotics designed a mobile picking robot with an extended arm that can perform a picking operation using a vacuum end-effector.

Furthermore, a few companies have started exploring experimental projects aimed at sorting and re-binning. For example, GreyOrange developed a fully automated sorting system to sort and divert outbound packets. XYZ Robotics developed a re-binning station that sorts batch-picked orders into each customer order. The sorting and re-binning process usually places objects onto a conveyor belt or in an over-sized tote or bin.

To conclude, the current warehouse automation developments are summarized in Fig. 1.1.

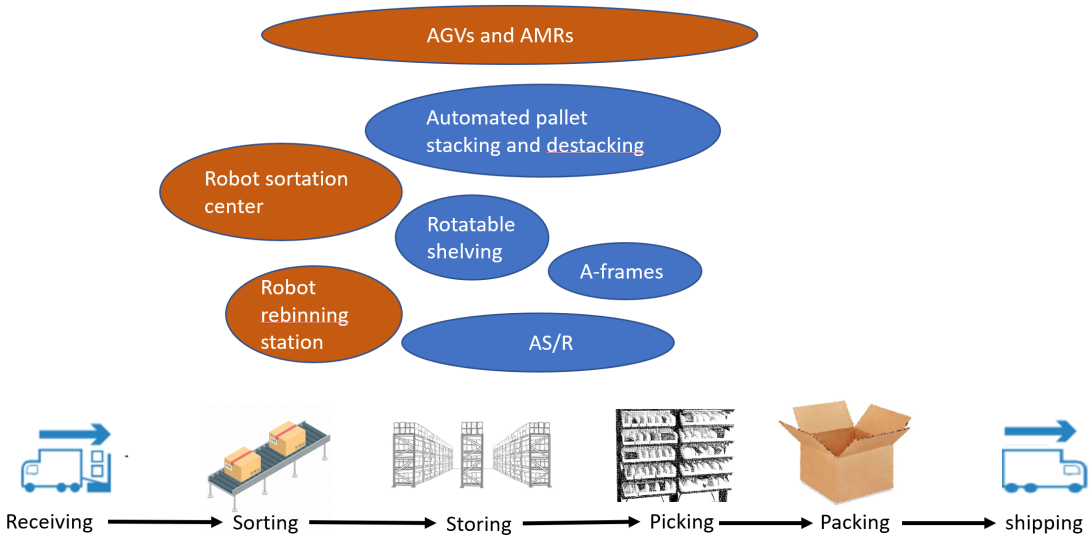


FIGURE 1.1: A summary of warehouse automation approaches and the processes they automate. Blue ovals represent mechanized technologies, while orange ovals represent robot technologies.

1.2 Robot Packing

One key aspect of warehouse automation that has attracted increasing amount of interest is robot packing, a final step of the fulfillment process when all ordered items from a customer are packed into a box for shipping. The current practice in fulfillment centers leaves the responsibility of container selection and packing largely to human worker intuition. Due to demanding schedules, workers cannot employ much foresight in the packing process and are reluctant to re-pack. As a result, over-sized containers are often used, incurring waste and higher shipping costs (Fig 1.2). Better containers and packing plans could be chosen using automated algorithms, whether packing is accomplished by humans or robots.

FIGURE 1.2: Examples of poor space utilization in shipping boxes.

E-commerce companies have been looking at automation methods to reduce shipping waste for a long time. Industry giants like Amazon have explored various strategies to use the smallest shipping container possible. Such measures include using machines known as the "SmartPac" that wraps a single customer order in patented envelopes , as well as the reported installation of two automatic 3D-box made-on-demand machines called CartonWrap, Cartonwrap can scan a single item and cut a box that fits around the object so that the waste is minimized [Das19, CMC20].

However, these measures stagnate when it comes to packing orders with more than one item: Arranging tight placements and optimizing a container for a set of irregular shapes is a difficult combinatorial optimization problem. Packing the objects densely with a robot manipulator also poses robot manipulation challenges.

Little prior work has been done in dense packing of multiple objects. Although boxing multiple items [YFD+16, SLG+18, ZSY+18] was previously demonstrated at the Amazon Robotics Challenge, the packing aspects were simplified by using over-sized containers. With an over-sized container, packing can be addressed with simple heuristics or even dropping items from the top center of the container, and the objects would still fit. Some recent works have studied dense packing as an extension to the classic "peg-in-hole" problem, a typical contact manipulation task that requires

precise position under some form of compliance [Dra05, HJJ+13, BDD95]. For example, Yu et al. [YR18] propose an insertion strategy that leverages tactile sensing to probe the gap for the targeted insertion pose while monitoring incipient slip to maintain a stable grasp on the object. Shome et al. [STS+19] use RGB-D data and a vacuum-based end-effector as a gripper and push finger. They monitor potential failures in real-time and use corrective pushing to achieve tight packing of an object close to the edge. However, these works are only concerned with inserting a single object into a tight designated space, but do not plan for where to place each object to achieve dense packing.

To address these shortcomings, this dissertation studies planning and robot execution for dense packing, with a focus on this specific problem: to pack objects of arbitrary shapes and weights into a single container. The goal is to optimize a placement plan and a container size while ensuring feasibility of the plan with a robot manipulator.

1.3 Cutting and Packing Problems

Traditionally, problems that involve the placement of objects within a container or a set of containers are referred to as cutting and packing problems. Most existing packing algorithms apply to idealized scenarios, such as rectilinear objects and floating objects not subject to the force of gravity. Under specific settings, such problems can be formulated and solved optimally using exact algorithms. One example of these state-of-the-art exact algorithms is the solution to the 3D bin packing problem using branch and bound, proposed by Martello et al. [MV98, MPV00], whose work is further extended by many including Boef et al. [dBKM+05] and Crainic et al. [CPT08]. Exact algorithms, although capable of finding the optimal solution if infinite time is spent, are strongly NP hard [GJ90] and do not guarantee optimal results within a reasonable amount of time especially when a large number of instances

are involved [MPV00]. Heuristic methods and meta-heuristic approaches have also been developed over the years, such as the popular Bottom-Left heuristic [BCR80] and the Best-Fit-Decreasing heuristic [JDU+74].

More recent work has addressed irregular shape packing, and only heuristic methods are practical because the search space is infinite. Meta-heuristics are commonly used in this setting such as Simulated Annealing (SA) [kam88, ZH04, LLCY15] and Guided Local Search (GLS) [FPZ03, Ege09, VT03, VVHS15, BLP13] that start with an initial placement and iteratively improve the placement by moving the pieces in the neighborhood while minimizing an objective function (e.g., overlap in the system). Other work has also proposed constructive positioning heuristics for 3D irregular objects, such as Deepest-Bottom-Left-Fill (DBLF), which places items in the deepest, bottom-most, left-most position; and Maximum Touching Area (MTA), which places an item in a position that maximizes the total contact area of its faces with the faces of other items [WGC+10].

To perform automatic packing in warehouses using a pre-computed packing plan, several real-world issues need to be addressed, such as stability under the force of gravity, and kinematics and clearance issues for the robot. If stability is not considered, the object pile may shift during execution, and therefore subsequent placements are unlikely to be executed as planned. If kinematics and clearance are not considered, the robot may be asked to perform infeasible motions (e.g., grip an item from underneath, bring an item to a target through another interlocked item, or pass through the container wall). Unstable placements and infeasible robot motions may cause failure to contain all items and even damage the robot and the items. Recovering from a failed packing operation can be complex and time-consuming, often requiring the robot to remove all items from the current box and place them in a larger container.

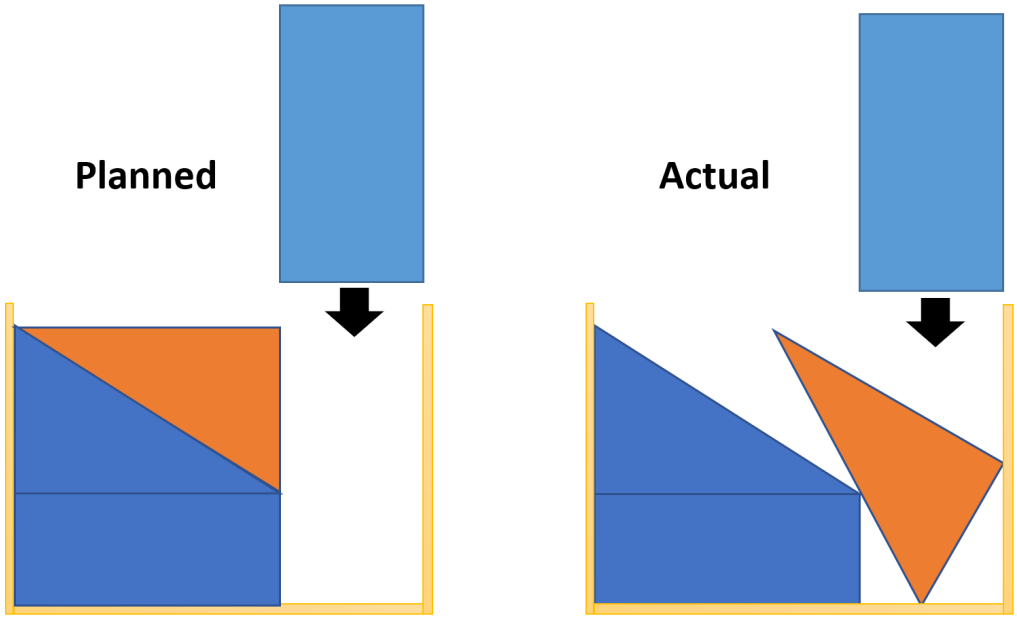


FIGURE 1.3: Unstable or floating placement is infeasible under force of gravity

Some research has taken aspects of stability into consideration during packing. Egeblad et al., for example, use a two-stage GLS packing algorithm that, in the first stage, optimizes for the center of gravity and inertia of the pile and, in the second stage, minimizes overlap in the system [Ege09]; Liu et al. propose a constructive method that packs irregular 3D shapes using a Minimum-Total-Potential-Energy heuristic [LLCY15]. This method performs a grid search for the lowest gravitational center height Z for each placement and can be hybridized with meta-heuristic SA to search for packing permutations that lead to lower total potential energy in the system. However, both proposed methods rely solely on heuristics and do not verify the stability of each placement. In contrast to these works, the method proposed in this work enforces stability explicitly using constraints.

One recent work on packing that takes into account robot manipulation feasibility is presented in [dBKM+05], in which the author proposes a variant of the orthogonal 3D box packing scheme such that no prior packed box is in front of, to the right

of, or above the current placing box, to avoid a possible collision with a vacuum gripper. This placing rule, however, cannot guarantee collision-free placements with other gripper geometries and neglects robot kinematic constraints and graspability constraints.

1.4 Summary of contributions

This dissertation is the first, to the best of my knowledge, to propose stability and robot feasibility constraints tailored to the automated warehousing domain and to implement a packing algorithm to solve those constraints. The algorithm guarantees the stability of the object pile during packing and the feasibility of the robot motion executing the placement plans. Experimental evaluation of the method is conducted with a realistic physical simulator on a dataset of scanned real-world items, demonstrating stable and high-quality packing plans compared with other 3D packing methods. Moreover, a prototype robot hardware and software system that executes the algorithm achieves a 98% success rate with proposed error-correcting measures. I hope that by highlighting the importance of robot-feasible offline planning and error-correcting measures, the ideas introduced in this dissertation can inform key design decisions for future robot warehouse automation technologies. My contributions are as follows:

1. A set of constraints for stable and feasible robot packing are formulated, and a constructive packing pipeline is proposed to solve these constraints. The pipeline is able to pack geometrically complex, non-convex objects while satisfying stability and robot packability constraints. Experimental evaluation of the algorithm conducted with a realistic physical simulator demonstrates stable and high-quality packing plans compared with other 3D packing methods. This work appeared previously as Wang et al. [WH19b], published in ICRA

2019 with co-author Kris Hauser. This work is presented in Chapter 2.

2. Two variants of packing problems in which the set of items is known, but the arrival order is unknown are formulated. This addresses the gap in the then-state-of-the-art algorithms that assumed a fully controllable arrival order of the items to be packed. The proposed algorithm provides certification that the items can be packed in a given container, as well as to optimize the size or cost of a container so that that the items are guaranteed to be packable, regardless of arrival order. This work appeared previously as Wang et al. [WH19c], published in RSS 2019 as best paper award nominee with co-author Kris Hauser. This work is presented in Chapter 3.

3. A technique for convenient 3D object model acquisition is presented in which an object is reoriented in front of a video camera with multiple grasps and regrasps. Experiments show that our method results in high quality reconstructed models. Moreover, testing with a novice user on a set of 200 objects demonstrates relatively rapid construction of complete 3D object models. This is done to help expedite the acquisition of 3D models used by a wide variety of robotic planners. This work appeared previously as Wang et al. [WH19a], published in ICRA 2019 with co-author Kris Hauser. This work is presented in Chapter 4.

4. Evaluation of the proposed planner under real-world uncertainties such as vision, grasping, and modeling errors. I build a hardware and software testbed that is representative of current state-of-the-art sensing, perception, and planning for warehouse manipulation. A systematic evaluation of the testbed is then performed to study the sources of error and models their magnitude. Exhaustive experiments are conducted in Monte Carlo simulation and on the

physical testbed to examine the feasibility of the packing placements under open-loop baseline conditions as well as utilizing two strategies for improving the robustness of robotic packing. Empirical results demonstrate a success rate of up to 98% can be achieved on a physical robot when using robustness measures despite cascading real-world uncertainties. This work is presented in Chapter 5.

5. During my thesis studies, I have also made contributions outside of the realm of warehouse automation. In order to conduct many desirable functions, service robots will need to actuate buttons and switches that are designed for humans. I designed a robot called SwitchIt that is small, relatively inexpensive, easily mounted on a mobile robot, and actuates buttons reliably. This work appeared previously as Wang et al. [WCH18], published in ICRA 2018 with co-author Kris Hauser. This work is presented in Chapter 6.

Finally, this dissertation concludes by summarizing contributions and discussing open challenges for future work in Chapter 7.

2

Offline Bin Packing under Stable and Robot-feasible Constraints

This chapter proposes a formulation of the offline packing problem that is tailored to the automated warehousing domain. Besides minimizing waste space inside a container, the problem requires stability of the object pile during packing and the feasibility of the robot motion executing the placement plans. To address this problem, a set of constraints are formulated, and a constructive packing pipeline is proposed to solve for these constraints. The pipeline is able to pack geometrically complex, non-convex objects while satisfying stability and robot packability constraints. In particular, a new 3D positioning heuristic called Heightmap-Minimization heuristic is proposed, and heightmaps are used to speed up the search. Experimental evaluation of the method is conducted with a realistic physical simulator on a dataset of scanned real-world items, demonstrating stable and high-quality packing plans compared with other 3D packing methods.[1]

[1] This chapter is reproduced from Fan Wang and Kris Hauser, "Stable Bin Packing of Non-convex 3D Objects with a Robot Manipulator," in 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 8698-8704. This work is funded by Amazon Research Award.

2.1 Introduction

Problems that involve the placement of objects within a container or a set of containers are generally referred to as cutting and packing problems. Most existing packing algorithms apply to idealized scenarios, such as rectilinear objects and floating objects not subject to the force of gravity. To perform automatic packing in warehouses using a pre-computed packing plan, several real-world issues need to be addressed, such as stability under force of gravity, and kinematics and clearance issues for the robot.

For a packing plan to be feasible with a robot manipulator, a comprehensive set of constraints need to be formulated.
In addition to the two standard packing constraints:

1. *Noninterference.* Each object is collision free,

2. *Containment.* All objects are placed within the internal space of the container,

we introduce the following constraints necessary for a robot-packable plan:

3. *Stability.* Each object is stable against previously packed objects and the bin itself, and

4. *Manipulation feasibility.* A feasible robot motion exists to load the object into the target placement. The robot must obey kinematic constraints, grasp constraints, and collision constraints during this motion.

In the following sections, we refer to constraints 1 and 2 as the *non-overlap* constraints, and constraints 1-4 as all constraints, or the robot-packable constraints.

While the application of robot-packable constraints is independent of the particular packing problem addressed, this chapter focuses on the problem of offline

packing of 3D irregular shapes into a single container. To solve this problem under robot-packable constraints, we present the following main contributions:

1. A polynomial time constructive algorithm to implement a resolution-complete search amongst feasible object placements, under *robot-packable constraints*.

2. A 3D positioning heuristic named Heightmap-Minimization (HM) that minimizes the volume increase of the object pile from the loading direction.

3. A fast prioritized search scheme that first searches for robot-packable placement in a three-dimensional space that likely contains a solution, and falls back to search in a five-dimensional space.

Our algorithm and others in comparison are tested in a realistic physics simulator, by packing large quantities of itemsets using highly complex, real-world object scannings. With item sizes of 3-5 objects (e.g., a common Amazon order size), the success rate is 99.9% for finding and executing packing plans using small Amazon order boxes. Large number of items are also packed in stress tests, in these tests, 80% of the placement plans were successfully executed in the physics simulator, which is significantly better than the 17% success rate from a standard packing solver under the same testing condition. Empirical results also show that the new Heightmap-Minimization heuristic finds more placements than existing heuristics.

2.2 Problem Defination

We address the problem of offline packing of 3D irregular shapes into a single container while ensuring the stability of each packed item and feasibility of the placement with a robot gripper.

Specifically, for a set N geometries $\mathcal{G}_1, \ldots, \mathcal{G}_\mathcal{N}$ where $\mathcal{G}_i \subset \mathcal{R}^3$, let $\mathcal{C}$ donate the free space volume of the container and ∂C as the boundary of the free space. Let

$T_i \cdot \mathcal{G}_i$ denote the space occupied by item i when the geometry is transformed by T_i. The problem is to find a placement sequence $S = (s_1, \ldots, s_N)$ of $\{1, \ldots, N\}$ and transforms $\mathcal{T} = (T_1, \ldots, T_N)$ such that each placement satisfies non-overlapping and containment constraints with geometries placed prior:

$$(T_i \cdot \mathcal{G}_i) \cap (P_j \cdot \mathcal{G}_j) = \emptyset,\ \forall i, j \in \{1, \ldots, N\}, i \neq j \tag{2.1}$$

$$T_i \cdot \mathcal{G}_i \subseteq \mathcal{C},\ \forall i \in \{1, \ldots, N\} \tag{2.2}$$

and for each $k = 1, \ldots, N$, stability constraints:

$$isStable\left(P_{s_k} \cdot \mathcal{G}_{s_k}, \mathcal{C}, T_{s_1} \cdot \mathcal{G}_{s_1}, \ldots, P_{s_{k-1}} \cdot \mathcal{G}_{s_{k-1}}\right) \tag{2.3}$$

and manipulation feasibility constraints:

$$isManipFeasible\left(P_{s_k} \cdot G_{s_k}, P_{s_1} \cdot \mathcal{G}_{s_1}, \ldots, P_{s_{k-1}} \cdot \mathcal{G}_{s_{k-1}}\right) \tag{2.4}$$

It is important to note that both stability and manipulation feasibility constraints must be satisfied for *every intermediate arrangement* of objects, not just the final arrangement.

2.2.1 Stability checking

Stability is defined as the condition in which all placed items are in static equilibrium under gravity and frictional contact forces. We model the stack using point contacts with a Coulomb friction model with a known coefficient of static friction. Let the set of contact points be denoted as $c_1, \ldots, c_K$, which have normals $n_1, \ldots, n_N$, and friction coefficients $\mu_1, \ldots, \mu_K$. For each contact c_k, let the two bodies in contact be denoted A_k and B_k. Let $f_1, \ldots, f_K$ denote the contact forces, with the convention that f_k is applied to B_k and the negative is applied to A_k. We also define m_i as the mass of object i, and cm_i as its COM. We take the convention that the container has infinite mass.

The object pile is in static equilibrium if there are a set of forces that satisfy the following conditions.

Force balance: $\forall i = 1, \ldots, N$,

$$- \sum_{k \,|\, i=A_k} f_k + \sum_{k \,|\, i=B_k} f_k + m_i g = 0. \tag{2.5}$$

Torque balance: $\forall i = 1, \ldots, N$,

$$- \sum_{k \,|\, i=A_k} (cm_i - c_k) \times f_k + \sum_{k \,|\, i=B_k} -(cm_i - c_k) \times f_k = 0.$$

Force validity: $\forall k = 1, \ldots, K$,

$$f_k \cdot n_k > 0, \tag{2.6}$$

$$\| f_k^{\perp} \| \leq \mu_k (f_k \cdot n_k). \tag{2.7}$$

where $f_k^{\perp} = f_k - n_k(f_k \cdot n_k)$ is the tangential component (i.e., frictional force) of f_k.

For a given arrangement of objects, an approximate set of contact points is obtained with the slightly scaled geometries in placement. A pyramidal approximation for the friction cone is used, and the conditions above are formulated as a linear programming problem over $f_1, \ldots, f_N$, solved using the convex programming solver CVXPY [DB16]. If no such forces can be found, the arrangement is considered unstable.

2.2.2 Manipulation feasibility

This constraint checks feasibility of a packing pose when executed by a robot manipulator. This requires that the object be graspable from its initial pose and can be packed in the desired pose via a continuous motion, without colliding with environmental obstacles.

In our system, we limit ourselves to the existence of a feasible top-down placement trajectory within the grasp constraints, as robots performing pick and place (e.g., box packing) commonly use vertical motion [dBKM+05]. We also assume the existence of a grasp generator that produces some number of candidate end effector(EE) transforms, specified relative to an object's geometry that may be used to grasp the object. The pseudo-code for this procedure is given in Alg. 1.

Algorithm 1: isManipFeasible

input : Desired placed geometry $T \cdot \mathcal{G}$ and a set of grasp candidates $\{T_1^{\mathcal{G}}, \dots T_n^{\mathcal{G}}\}$

1 **for** $T^{\mathcal{G}} \in \{T_1^{\mathcal{G}}, \dots T_n^{\mathcal{G}}\}$ **do**
2 Compute top-down EE path $\mathcal{P}_{ee}$ interpolating from an elevated pose to a final pose $T \cdot T^{\mathcal{G}}$;
3 **for** $P_{ee} \in \mathcal{P}_{ee}$ **do**
4 **if** $\neg$*(IKSolvable* (P_{ee}) $\wedge$ *inJointLimits*(P_{ee}) $\wedge$ *collisionFree*$(P_{ee}))$ **then Continue** with Line1;
5 **end**
6 **return** True
7 **end**
8 **return** False

2.3 Pipeline for Robot-packable Planning

We develop a constructive packing pipeline to solve for the set of robot-packable constraints proposed. Our algorithm accepts an itemset, a container dimension, a constructive positioning heuristic, and/or a packing sequence, to produce packing plans. The pipeline packs each item to its optimized feasible pose in sequential order, without backtracking.

Our pipeline primarily consists of 4 components, namely:

1. Placement sequence

2. Generate ranked transforms

3. Stability check

4. Manipulation feasibility check

The pipeline starts with a sequencing heuristic to sort all items in a tentative placement ordering and allocates them individually into the container in this sequence. For each object at the time of the allocation, a set of candidate transforms satisfying robot-packable constraints are generated and ranked based on the positioning heuristic. Constraint checks are performed in order until a transform satisfying all required constraints is returned.

2.3.1 Placement sequence

The placement sequence can be user-specified or generated by non-increasing bounding box volume rule. The generated sequence is subject to adjustment if a solution cannot be found in the specified ordering.

2.3.2 Generating ranked transforms

For a given item, a positioning heuristic (e.g., placement rule) identifies a free pose inside the container that is most preferred according to a specific criterion. Our pipeline accepts arbitrary positioning heuristics, but instead of applying the heuristic to obtain one optimal placement for each item, we use the score formulated from the positioning heuristic to rank candidate placements.

The candidate placements are obtained with a prioritized search among a discretized set of object poses. Instead of searching in the 6D space of SE(3), our algorithm first performs a grid search in a 3D space that likely contains robot-packable solutions. In the 3D search, the rolls and pitches of G are restricted to be a set of *planar-stable* orientations, which are a set of stable resting orientations of G on a planar surface, computed using the method of Goldberg et al. [GMZ+99]. This speeds

up the search for the common case of packing on the first layer and on horizontal supports. If no feasible solutions exist in the 3D space, the algorithm falls back to search in 5D, in which a grid search is performed for rolls and pitches as well.

The 3D search for collision-free placements of one object, given a set of rolls and pitches, is shown in Alg. 2. A grid search is performed for yaw, X, and Y at a given resolution, and the height Z of the placement is analytically determined as the lowest free placement. 2D heightmaps are used to accelerate the computation of Z to an efficient 2D matrix manipulation. Three heightmaps are computed: 1) a top-down heightmap H_c of the container and placed objects, 2) a top-down heightmap H_t of the object to be placed, and 3) a bottom-up heightmap H_b of the object to be placed. H_t and H_b are measured relative to the lower left corner of the orientated object. Raycasting is used to build these heightmaps, and rays that do not intersect with the object geometry are given height 0 in H_t and ∞ in H_b. The container heightmap is obtained once at the beginning of object placement search, and an object heightmap is computed once for each distinct searched orientation.

Given an object orientation and X, Y location, we calculate the lowest collision-free Z as follows:

$$Z = \max_{i=0}^{w-1} \max_{j=0}^{h-1} (H_c[x+i, y+j] - H_b[i, j]) \tag{2.8}$$

where (x, y) are the pixel coordinates of X, Y, and (w, h) to be the dimensions of H_t.

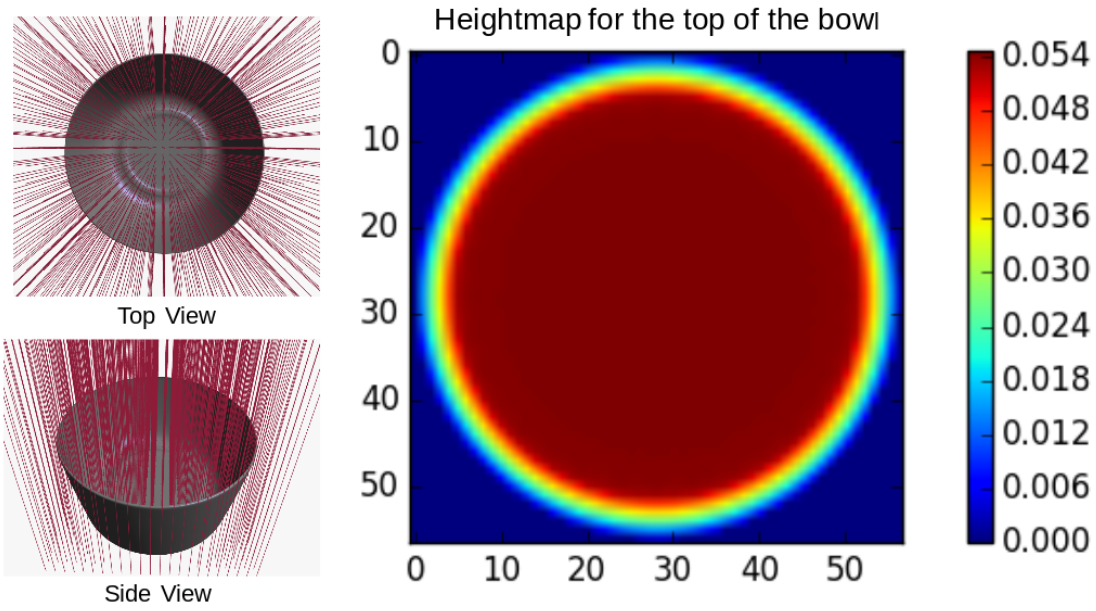


FIGURE 2.1: Heightmap of a bowl from ray-mesh intersection.

Algorithm 2: 3DGridSearch

input : Geometry $\mathcal{G}$, container $\mathcal{C}$, rolls and pitches O
output: All collision-free candidate transforms

for $(\phi, \psi) \in O$ **do**
  for $\theta \in \{0, \Delta r, 2\Delta r, \ldots, 2\pi - \Delta r\}$ **do**
    Let $R \leftarrow R_z(\theta)R_y(\phi)R_x(\psi)$;
    Discretize legal horizontal translations of $R \cdot \mathcal{G}$ into grid $\{(X_1, Y_1), \ldots (X_n, Y_n)\}$;
    for (X, Y) *in* $\{(X_1, Y_1), \ldots (X_n, Y_n)\}$ **do**
      Find the lowest collision free placement Z at translation X, Y;
      $T \leftarrow (R, (X, Y, Z))$;
      if $T \cdot \mathcal{G}$ *lies within* $\mathcal{C}$ **then** Add T to $\mathcal{T}$;
    end
  end
end
return $\mathcal{T}$

Once all collision-free candidate transforms are obtained, they are scored by a scoring function formulated from a positioning heuristic. For example, the Deepest-

Bottom-Left-First heuristic can be formulated as the score:

$$Z + c \cdot (X + Y) \tag{2.9}$$

where c is a small constant.

The candidates are then ranked by score (lower is better). If only robot-packable constraints are required, the placement candidate with the lowest score is returned. If additional constraints are specified, the ranked candidates will be checked for the additional constraints until a candidate satisfying all constraints is returned.

After a new object has been placed, we update the heightmap of the container H_c. This subroutine is also used in our heightmap minimization heuristic. Given a pose X, Y, Z of the object to be packed, and the top heightmap H_t at the given orientation, we calculate an updated heightmap H'_c adding the placed object as follows:

For all $i = 0, \ldots, w-1$, $j = 0, \ldots, h-1$, we let:

$$H'_c[x+i, y+j] = \max(H_t[i,j] + Z, H_c[x+i, y+j]) \tag{2.10}$$

if $H_t[i,j] \neq 0$, and otherwise

$$H'_c[x+i, y+j] = H_c[x+i, y+j]. \tag{2.11}$$

2.3.3 Pipeline summary and fall back procedures

A packing attempt for a single item is summarized in Alg. 3, and the overall pipeline for packing multiple objects is given in Alg. 4. Given a heuristic packing sequence, it calls Alg. 3 for each item with the set of planar-stable rolls and pitches. This first stage finds placements for most objects in typical cases. For the remaining unpacked items U, the algorithm activates the *fallback procedure*. The fallback procedure examines each unpacked item and attempts to perturb the planar stable orientations by iterating over rolls and pitches until a solution is found, and if no solution is found the algorithm terminates with failure.

Algorithm 3: packOneItem

input : item geometry $\mathcal{G}$, container $\mathcal{C}$, pitches and yaws O, sequence of the packed items $\{s_1, \ldots, s_i\}$, transforms of the packed items $\{P_1, \ldots, P_i\}$

output: Transform T or **None**

$\mathcal{T} \leftarrow$ 3DGridSearch$(\mathcal{G}, \mathcal{C}, O)$;
Score each T in $\mathcal{T}$ based on heuristic used;
for *up to N lowest values of* T *in* $\mathcal{T}$ **do**
    if $\neg$ *isStable*$(T \cdot \mathcal{G}, \mathcal{C}, P_1 \cdot \mathcal{G}_{s_1}, \ldots, P_i \cdot \mathcal{G}_{s_i})$ **then continue**;
    Compute grasp poses $T_1^{\mathcal{G}}, \ldots T_n^{\mathcal{G}}$ compatible with T;
    if *isManipFeasible*$(T \cdot \mathcal{G}, \{T_1^{\mathcal{G}}, \ldots T_n^{\mathcal{G}}\})$ **then return** T;
end
return ***None***

2.4 Heightmap-Minimization Heuristic

The performance and solution quality of a multi-dimensional packing problem is highly susceptible to the item-positioning rule [LMV04]. However, existing positioning heuristics for 3D packing are scarce and are commonly adapted directly from 2D packing, and therefore result in poor space utilization in the 3D container [LMV02, CPT08]. To address these shortcomings, we propose a novel positioning heuristic called the Heightmap-Minimization (HM) heuristic, which favors item placements that result in the smallest occupied volume in the container, as observed from the loading direction.

Specifically, HM scores a placement as follows. Given the candidate transform $T = (roll, pitch, yaw, X, Y, Z)$, compute a tentative container heightmap H'_c using the update routine described in Sec. 12. Suppose its shape is (w, h). The score for the placement using the HM heuristic is:

$$c \cdot (X + Y) + \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} H'_c[i, j] \tag{2.12}$$

where c is a small constant.

Algorithm 4: Robot-feasible packing with fall back procedures

input : Item geometries $\mathcal{G}_1, \ldots, \mathcal{G}_N$, container C, initial packing sequence $\{s0_1, \ldots, s0_N\}$
output: Transforms $\mathcal{T}$ and sequence S, or **None**

1 Initialize $\mathcal{T}, S, U, \mathcal{O}$ to empty lists;
2 **for** $\mathcal{G}_i \in \{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$ **do**
3 | Get planar-stable rolls and pitches for $\mathcal{G}_i$ with the top n highest quasi-static probabilities $O_i = \{(\phi_1, \psi_1), \ldots, (\phi_n, \psi_n)\}$;
4 | Add O_i to $\mathcal{O}$;
5 **end**
6 **for** $(s0_i \in \{s0_1, \ldots, s0_N\}$ **do**
7 | T = packOneItem(G_{s0_i}, C, O_{s0_i}, S, $\mathcal{T}$);
8 | **if** T **then** Add T to $\mathcal{T}$, Add $s0_i$ to S;
9 | **else** Add $s0_i$ to U;
10 **end**
11 **for** $u_i \in U$ **do**
12 | Let $\{(\phi_1, \psi_1), \ldots, (\phi_n, \psi_n)\}$ be the planar-stable orientations in O_{u_i};
13 | **for** $t_r \in \{0, \Delta r, 2\Delta r, \ldots, 2\pi - \Delta r\}$ **do**
14 | | **for** $t_p \in \{0, \Delta r, 2\Delta r, \ldots, 2\pi - \Delta r\}$ **do**
15 | | | $O^t = \{(\phi_1 + t_r, \psi_1 + t_p), \ldots, (\phi_n + t_r, \psi_n + t_p)\}$;
16 | | | T = packOneItem(G_{u_i}, C, O^t, S, $\mathcal{T}$);
17 | | | **if** T **then**
18 | | | | Add T to $\mathcal{T}$; Add u_i to S;
19 | | | | **continue** with Line 11
20 | | **end**
21 | **end**
22 | **return** ***None***
23 **end**
24 **return** $(\mathcal{T}, S)$

HM favors positions and orientations that result in good space utilization as it minimizes wasted space and holes that cannot be filled. HM also favors stable placements since the bottom of the object is encouraged to match the shape of the supporting terrain (Fig. 2.2).

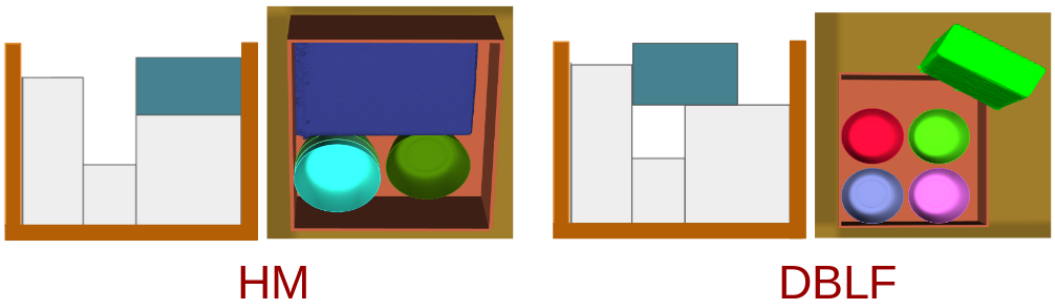


FIGURE 2.2: Example packing placements obtained by HM and DBLF. HM finds more compact and stable packing compared to DBLF.

2.5 Experiment

We tested our algorithm on different item sets and validated plan feasibility in a physics simulator. Objects were drawn at random from a set of 94 real-world object meshes from the YCB [CSB+17] and the APC 2015 object set [Rut15]. On average each mesh contains 10,243 vertices. Experiments are conducted on Amazon Web Services instance type m5.12xlarge. All computation times are measured on a single thread. Parameters used in the experiment are: heuristic constant $c = 1$; heightmap resolution 0.002m; step size in both X and Y 0.01m; $\Delta r = pi/4$ in range $[0, \pi)$; friction coefficient $\mu = 0.7$. Contact points are obtained using the exact geometry with a scale factor of 1.03. The top 4 planar-stable rolls and pitches with the highest quasi-static probabilities are used, and candidate number $N = 100$.

2.5.1 Robot manipulation feasibility with a vacuum gripper

The robot model used to verify robot feasibility constraints is a Staubli TX90 robot, equipped with a cylindrical vacuum gripper of 30 cm length and 2 cm diameter. The graspability constraints ask the vacuum gripper to grasp within a radius $r = 2$ cm in the horizontal plane to the object's center of mass when the object is sitting in flat orientations. The areas under the gripper should be solid planar areas (80% of the surface points directly below the tool are within 0.3mm to the estimated plane) for

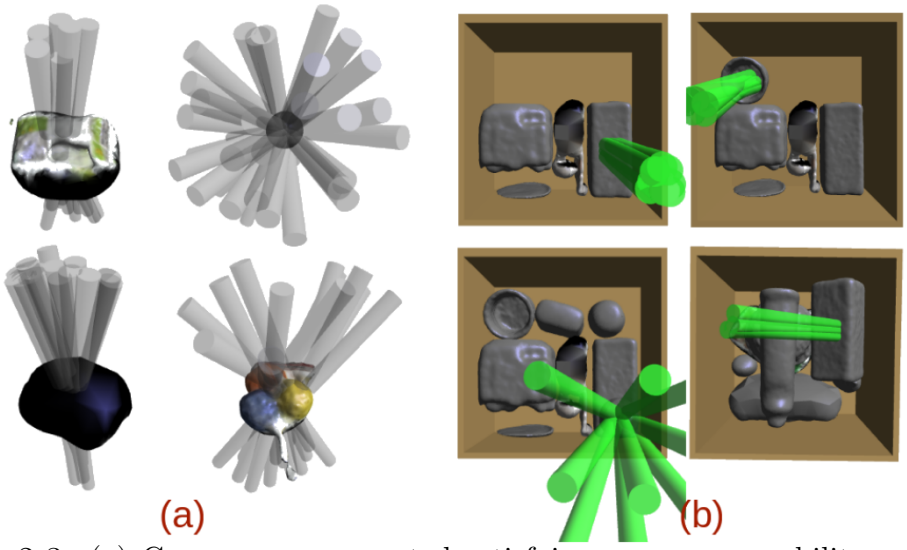


FIGURE 2.3: (a) Grasp poses generated satisfying vacuum graspbility constraints. (b) Compatible gripper poses with candidate object orientation are checked for clearance with the container and the object pile. Collision-free grasps are shown in green and colliding grasps are colored in transparent grey.

the vacuum opening to grasp normal to the surface, the resulting gripper axis needs to be within a tilting angle $\theta = \pi/4$ to the Z-axis.

2.5.2 Small Order Packing

Simulating problem settings in a typical warehouse, we performed a small order packing test. Per communication with personnel at Amazon, 3-5 items are a standard order size. We generated 1000 random itemsets consisting of 3-5 models. The itemsets are verified with various testing methods to fit in at least one of the five containers used in the Amazon Robotics Challenge 2017 [Rob17] under robot-packable constraints, and the smallest feasible container for each itemset is recorded.

We test our pipeline with HM heuristic and all constraints. Our algorithm seeks a feasible solution using the smallest container first, and if fails, the process is repeated on the second smallest container and so on until either a solution is found or all available containers are exhausted.

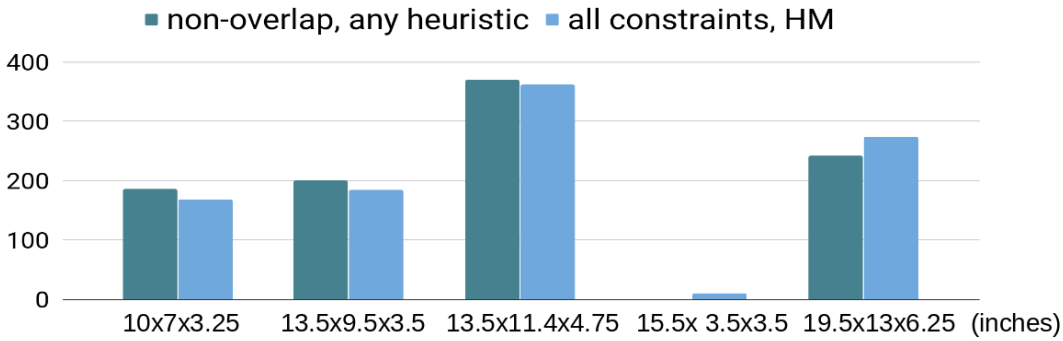


FIGURE 2.4: Distribution of the solution containers found under different level of constraints. The x axis is the 5 container dimensions tested.

The success rate is 99.9% averaging 9.54 s per order. The only failure occurs when a large object needs to be tilted sideways to fit within a tight space, and no feasible vacuum grasp exists in the specified orientation. Fig. 2.4 compares the smallest bin statistics using all tested methods and ours. It appears that HM packing, despite adding all constraints, enlarges the container needed only marginally.

FIGURE 2.5: Examples of packing plans for item sets of size 3–5.

FIGURE 2.6: Examples of packing plans for itemsets of size 10.

2.5.3 Comparisons on Large Itemsets

Next, we perform stress tests on itemsets of size 10. A tall container of size $32 \times 32 \times 30$ cm is chosen. 1000 itemsets of size 10 are generated and verified with all tested methods to have a non-overlap packing within the chosen container. Since the tilted gripper is likely to collide with the tall container chosen, we assume the gripper can grasp object of any orientation at the center of object's top surface, with the gripper axis vertically aligned to the Z axis.

We compare our HM heuristic against the DBLF and MTA heuristics [WGC+10], as well as an implementation of a guided local search (GLS) method as described by Egeblad et al. [Ege09]. The fast intersection area theorem in Egeblad's paper was not implemented. Therefore, for the fairness of the comparison, GLS was run with 5 random restarts, and each restart was terminated after 300 s if a solution could not be obtained. GLS is also not tested for all constraints, as implementing robot-packable constraints in GLS methods is very challenging. Table 2.1 reports the percentage of solutions found and the average computation time.

Empirically, HM finds more solutions than any other method in comparison. With robot-packable constraint, HM finds 99.9% of all feasible solutions, leading the 2nd place DBLF heuristic by 1.5%, while MTA and GLS are not as competitive. After adding all constraints, each technique drops in success rate by a few percents,

Table 2.1: Comparing planning techniques on 10-item orders with and without robot-packable constraints

	HM	DBLF [WGC$^+$10]	MTA [WGC$^+$10]	GLS [ENO07]
Success, non-overlap (%)	99.9	98.4	88.9	78.9
Time, non-overlap (s)	15.7	14.2	14.1	502
Success, all constraints (%)	97.1	96.3	86.3	—
Time, all constraints (s)	34.9	50.1	95.4	—

Table 2.2: Impact of Δr and candidate number N on the results

Rotation granularity Δr	$\pi/4$	$\pi/4$	$\pi/8$
Number of candidates N	100	500	500
Success, all constraints (%)	97.1	97.5	98.7
Time, all constraints (s)	34.9	70.05	89.40

but HM still leads the other methods. The mean running time of HM is also 30% shorter, indicating that the highest ranked placements are more likely to be stable than the other heuristics.

In addition, only 3.2% (320 out of 10,000) of the items are packed with the fallback procedure, indicating the 3D space searched is indeed highly likely to contain robot-packable solutions. The fallback procedure is nonetheless important, as, with no fallback procedure, the success rate with all constraints drops from 97.1% to 72.4%.

With finer rotation granularity Δr and more candidates to check against robot-packable constraints, the success rate can be further improved at the cost of increased computation time (Table 2.2).

2.5.4 Executing Packing Plans in Simulation

Finally, we test the open-loop execution feasibility of packing plans in the Klamp't robot physics simulator [Hau20]. In the simulation, the robot places one item after another using a top-down loading direction. The plan is considered a success if:

Table 2.3: Execution success rates in simulation, 10-item orders

	Success (%)	Drop (cm)	Horiz. Shift (cm)
Non-overlap constraint	17.11	1.95	1.29
All constraint	79.1	1.36	0.50

1) All items placed to the planned transforms without the robot and the objects colliding with items placed prior, and 2) all items contained within the container when placement is complete.

The robot used in the simulation is the TX90 robot model with the vacuum gripper described. We make the same assumption for the gripper as in the 10-item packing case. The robot places all items 1cm elevated from their planned transform; therefore there is an expected 1 cm drop. We allow 20 s for the items to settle before the next item is placed.

In the 3-5 items case, 100% of plans are executed successfully according to our success criteria. In the 10 item case, 768 out of 971 (≈80%) of robot-packable plans obtained with HM heuristic are executed successfully. This is significantly higher than the 17% success rate with robot-packable constraints. Further shown in Table 2.3, items packed with all constraints undergo smaller displacements during packing execution, indicating increased stability.

The 20% failure cases are caused by an object falling out of its desired placement, which prevents subsequent items from being packed. The stability checker may be too optimistic, especially for intrinsically unstable objects like balls. Moreover, the impact of dropping an object could shift supporting objects.

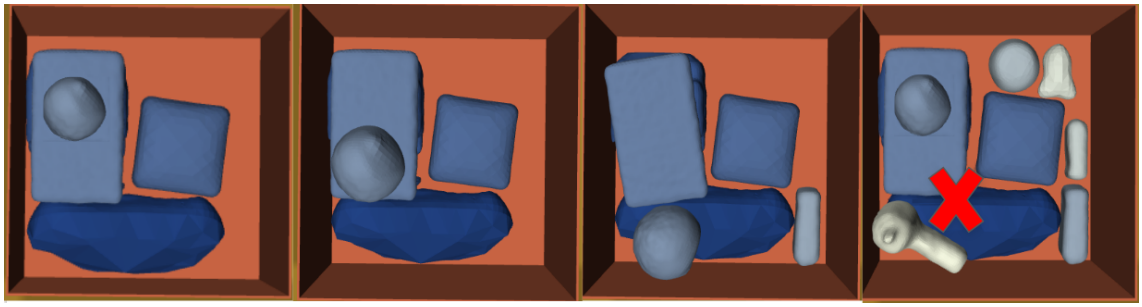


FIGURE 2.7: A typical execution failure case. The left three frames show a ball rolling out of its desired position preventing subsequent placement of the drill in the rightmost frame.

3

Robot Packing with Known Items and Nondeterministic Arrival Order

This chapter formulates two variants of packing problems on top of the proposed offline algorithm, in which the set of items is known but the arrival order is unknown. The goal is to certify that the items can be packed in a given container, and/or to optimize the size or cost of a container so that that the items are guaranteed to be packable, regardless of arrival order. The Nondeterministically ordered packing (NDOP) variant asks to generate a certificate that a packing plan exists for every ordering of items. Quasi-online packing (QOP) asks to generate a partially-observable packing policy that chooses the item location as each subsequent item is revealed. Theoretical analysis demonstrates that even the simple subproblem of verifying feasibility of a packing policy is NP-complete. Despite this worst-case complexity, practical solvers for both NDOP and QOP are developed. Multiple extensions to the basic nondeterministic problem are presented, including packing with a fixed-capacity buffer and packing with equivalent objects. Experiments demonstrate that these algorithms can be applied to packing irregular 3D shapes with stability

and manipulator loading constraints.[1]

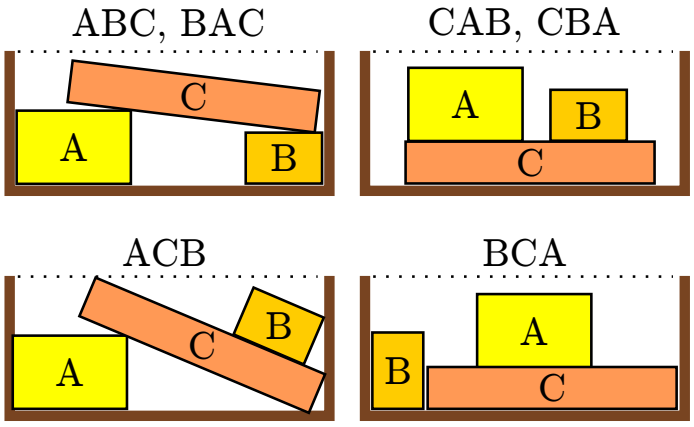
[1] This chapter is reproduced from Fan Wang and Kris Hauser, "Robot Packing with Known Items and Nondeterministic Arrival Order" in Robotics: Science and Systems(RSS), June 2019. The journal version of the paper is currently in submission to in submission to IEEE Transactions on Robotics (T-RO). This work is supported by an Amazon Research Award.

3.1 Introduction

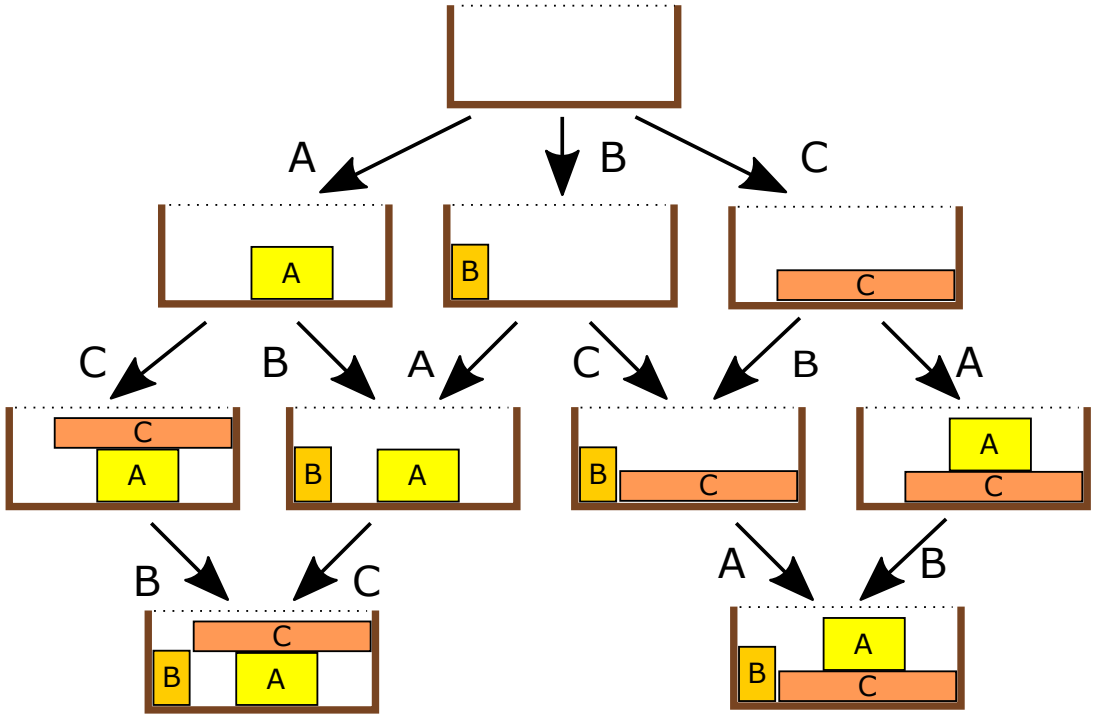
Interest in warehouse automation has grown rapidly with the growth of e-commerce and advances in robotics. Given the rapid progress in the field of robotic manipulation, the prospect of fully autonomous picking and packing robots is becoming increasingly likely in the near future [CBB+18], but relatively little attention has been paid to robotic packing. Packing algorithms have the potential to optimize containers and packing plans for both human and robot packers. In the current state of practice in fulfillment centers, human workers select containers and pack items largely according to intuition. Heuristic algorithmic assistance based on item bounding box dimensions may be employed, but these lead to conservatively large containers. When containers are chosen too small, items need to be repacked, causing delays and reducing efficiency. When containers are too large, excess material is wasted and shipping costs are increased.

A large variety of packing problems have been studied, including the bin and strip packing problem, knapsack problem, container loading problem, nesting problem, and others. In an *offline* setting, the items and container(s) are known, and a plan can place the items in arbitrary order [MPV00]. In an *online* setting, the items are not known a priori and need to be placed as they arrive [Sei02]. We consider a *robot packing* setting which addresses packing problems with the additional constraints that items must be loaded with a collision-free robot path, and that intermediate piles of items must be stable against gravity.

This chapter introduces two *nondeterministic* formulations of robot packing problems that lie between the offline and online settings. These formulations are practical for automated warehouses where the ultimate item set (e.g., shopping cart) is known, but some distinct, uncontrollable component of the packing system controls the item arrival order. For example, in Amazon's automated fulfillment centers, shelving units



(a) Nondeterministically ordered packing (NDOP)



(b) Quasi-online packing (QOP)

FIGURE 3.1: Feasible solutions for a 2D, 3-item instance of (a) NDOP and (b) QOP. All 3! = 6 possible arrival orders are collision-free, loadable from top-down, and yield intermediate piles that are stable under gravity. In QOP, an item is never moved after it is placed.

containing individual items are carried by thousands of mobile robots to several picking stations, and the order in which shelves arrive at a given station is controlled by a complex algorithm that is tuned to maximize delivery throughput for shelving units. In applications where item deliveries are human-controlled, such as in less than truckload (LTL) consolidation, it may be even less practical for an algorithm to dictate the arrival order. Hence, to guarantee that the items can fit in a given container, a packing planner should *certify* the validity of a plan under *all possible arrival orders*. In the NDOP variant, the feasibility of the container is verified under all nondeterministic orders, but the arrival sequence is revealed before packing is executed. In the QOP variant, each object must be packed before the next item is revealed (Fig. 3.1).

Our framework for solving NDOP and QOP problems uses a combination of an offline planner and a packing policy verifier. A packing policy is represented by a set of possible packing plans, each of which consists of a set of packing locations and a directed acyclic graph (DAG) of their dependencies. The offline planner is treated as an *oracle* that is supplied independent of the NDOP / QOP algorithm, and is assumed to handle all geometric and physical constraints required of the packing domain. The verifier will verify or disprove the feasibility of a policy under all permutations of arrival orders. We present a verification algorithm that uses pruning techniques, and in practice can check feasibility quickly even for a large number of objects and packing plans. However, in some cases exponential behavior is observed. We prove that the worst-case solution complexity of NDOP and QOP is $O(n!)$ and even feasibility verification for a polynomial-sized NDOP policy is NP-complete, via reduction from SAT.

Nevertheless, the solver is practical for small numbers of items, and even using an incomplete offline planner, it guarantees that a solution, when found, is feasible for all object orderings. Several packing heuristics are also introduced to improve

scalability of the approach, and experiments demonstrate that our approach can be realistically applied to irregular 3D shapes with item sets of size up to 10.

3.2 Problem formulation

Let $\mathcal{I} = \{v_1, \ldots, v_n\}$ be a set of n items. Item v_i has some geometry $A_i \subset \mathbb{R}^d$, and we wish to pack all items into a container volume $C \subset \mathbb{R}^d$. Here $d = 2$ or 3 is the dimension of the workspace. The offline packing problem is to compute a *feasible packing plan* given $A_1, \ldots, A_n$ and C. Such a plan is defined as follows:

Definition 1. *A* packing plan P *consists of an ordering* $\sigma_{1:n} = (\sigma_1, \ldots, \sigma_n)$ *and a tuple of transforms* $T_{1:n} = (T_1, \ldots, T_n)$*, in which* $\sigma_j \in \{1, \ldots, n\}$ *specifies that* v_{σ_j} *is the* j*'th item to be placed, and* $T_i \in SE(d)$ *specifies the target location (pose) of* v_i*.*

An ordering must be a permutation on n elements, and is hence an element of the symmetric group S_n

Definition 2. *A packing plan is* feasible *when it, and all prefix plans, satisfy certain constraints, as shown in Fig. 3.2 and defined in the below section.*

The prefix feasibility requirement means that for all $j < n$, the ordering $\sigma_{1:j}$ with the corresponding items in locations $T_{\sigma_1}, \ldots, T_{\sigma_j}$ must also satisfy the feasibility constraints. For example, we cannot require two blocks to be placed simultaneously on either ends of a see-saw when stability is violated with only a single block (Fig. 3.2.d).

3.2.1 Constraint formulation

In our formulation, the feasibility of a packing plan requires satisfying the following three constraints. For readability, for the ordering $\sigma_{1:n}$ let us denote the sequence number s_i of the i'th item to be the ordinal index in which it appears, i.e., $s_{\sigma_j} = j$ and $\sigma_{s_i} = i$.

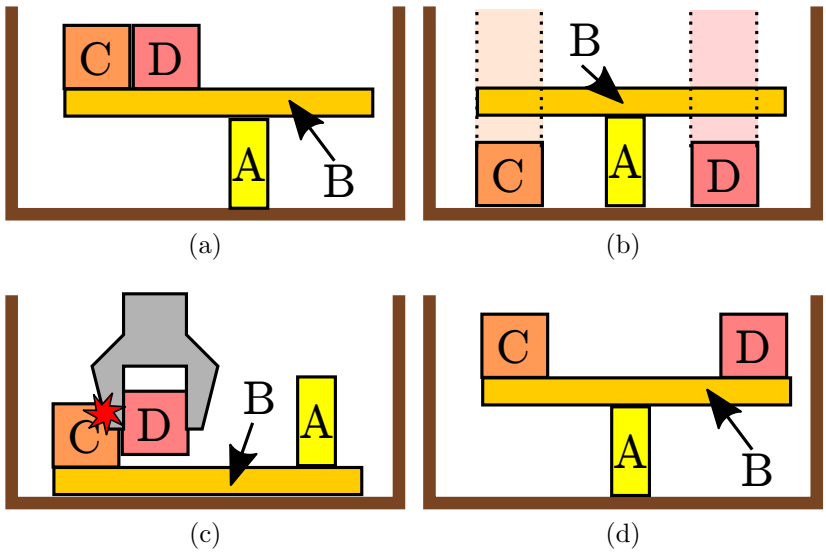


FIGURE 3.2: Examples of plans that are infeasible for arrival order ABCD: (a) unstable, (b) items C and D collide with B along the loading direction, (c) and the path for the robot manipulator to grasp and load item D is infeasible. In (d), although ABCD is feasible, the prefix requirement is violated because the sub-plan ABC is unstable.

Non interference All items do not overlap but can touch (3.1) and all items lie entirely inside the container (3.2):

$$T_i A_i^\circ \cap T_j A_j^\circ = \emptyset \text{ for all } i, j \text{ with } i \neq j, \tag{3.1}$$

$$T_i A_i \subseteq C \text{ for all } i. \tag{3.2}$$

Here $\cdot^\circ$ denotes a set's interior.

Equilibrium To prevent "floating" items and unbalanced stacks, the equilibrium constraint requires that each intermediate packing be stable under gravity and frictional contact (Fig. 3.2.a). An item is allowed to make contact with the container walls and previously placed items. We model these as a set of contact points, and require that there exist feasible forces at each contact point that respect Coulomb friction.

Manipulation feasibility Each item in the packing plan must be loadable by a manipulator without disturbing previously packed items (Figs. 3.2.b and 3.2.c). We

consider a robot gripper R and a top-down loading direction. In the packing plan, an item is also given a grasp transform T_G, such that the combined geometry of the ith item and the robot while grasped is $A_i \cup T_G R$. The swept volume of the item and robot while loading is $SV_i = \overline{ab} \oplus T_i(A_i \cup T_G R)$, where $a = (0, 0, 0)$ and $b = (0, 0, h)$, with h some "safe" height greater than the height of the container. This constraint states that, for all items i, the swept volume cannot intersect any previously-placed items (3.3) or the container walls ∂C (3.4):

$$SV_i^\circ \cap T_j A_j = \emptyset \text{ for all } j \text{ s.t. } s_j < s_i, \tag{3.3}$$

$$SV_i^\circ \cap \partial C = \emptyset. \tag{3.4}$$

3.2.2 Nondeterministic problems

Definition 3 (NDOP)**.** *The* nondeterministically ordered packing problem *asks whether there exists a feasible packing plan for every ordering* $\sigma_{1:n} \in S_n$*.*

To define QOP, we need to define the concept of a *feasible packing policy* as follows:

Definition 4. *A* packing policy *is a function* π *that takes as arguments the identities and locations of previously packed items* $(T_{\sigma_1}, \ldots, T_{\sigma_{j-1}})$ *and the next item* σ_j *to be packed, and returns the location* T_{σ_j} *of the next packed item.*

Definition 5. *A packing plan is* generated *by a packing policy* π *and an ordering* $\sigma_{1:n} \in S_n$ *via the recursive application of the policy:*

$$\begin{aligned} T_{\sigma_1} &= \pi((), \sigma_1), \\ T_{\sigma_2} &= \pi((T_{\sigma_1}), \sigma_2), \\ &\vdots \\ T_{\sigma_n} &= \pi((T_{\sigma_1}, \ldots, T_{\sigma_{n-1}}), \sigma_n). \end{aligned} \tag{3.5}$$

Definition 6. *A* packing policy is feasible *if for all item orders* $\sigma_{1:n} \in S_n$*, the generated packing plan is feasible.*

Since a packing policy is deterministic, we can also write the policy as a function of the prior order of the objects:

$$\pi((\sigma_1, \ldots, \sigma_{j-1}), \sigma_j) \equiv \pi((T_{\sigma_1}, \ldots, T_{\sigma_{j-1}}), \sigma_j). \tag{3.6}$$

A policy can also be viewed as a tree with depth n and each node has $n - \ell$ branches on level ℓ. There are $n \cdot (n-1) \cdots (n-\ell)$ nodes on level ℓ, and so this gives a total of $\sum_{\ell=1}^{n} n!/\ell! = O(n \cdot n!)$ nodes altogether.

Definition 7 (QOP)**.** *The* quasi-online packing problem *asks to compute a feasible packing policy.*

The main difference between NDOP and QOP is that with QOP, the items are revealed in sequence, and the location chosen for an item is fixed and may not be changed thereafter. QOP is at least as hard as NDOP, because any solution to QOP is also a solution to NDOP (but the converse does not hold).

3.2.3 Container optimization variants

Above we have stated these packing problems in their decision versions. We also consider container optimization variants, which assume a set of possible containers $\mathcal{C}$ and a cost function $cost : \mathcal{C} \to \mathbb{R}$, and are stated as follows:

- Offline: Find the container $C \in \mathcal{C}$ with minimum cost that yields a feasible packing plan for item set $\mathcal{I}$.
- Nondeterministically-ordered: Find the container $C \in \mathcal{C}$ with minimum cost that yields a feasible packing plan for any ordering of item set $\mathcal{I}$.

- Quasi-online: Find the container $C \in \mathcal{C}$ with minimum cost that yields a feasible packing policy for item set $\mathcal{I}$.

The container set is typically discrete, such as a set of available boxes, but could also be continuous, such as a varying height. This formulation can express the classical bin-packing problem, where $\mathcal{C}$ contains a container with 1 bin, a container with 2 bins, and so on, and cost measures the number of bins.

NDOP and QOP are adapted rather easily into discrete container optimization algorithms by enumerating containers in order of non-decreasing cost until a successful packing policy is found.

3.3 Method

We make use of our offline robot packing planner proposed in the last chapter with a small amount of modification. The responsibility of the offline planner is to generate a feasible packing plan given the constraints outlined above, while our key contributions are novel methods to invoke the planner and to validate plans under permutations of item orders. Our NDOP and QOP algorithms treat the offline planner as an oracle that is responsible for all constraint checking and dependency graph construction steps. As a result, the NDOP and QOP algorithms can be generalized to other packing settings by replacement of the oracle with other methods.

3.3.1 Offline planning oracle

The offline planner is required to accept some number of fixed items and a partial packing sequence for the remaining items. Its interface takes the form

$$P \leftarrow \text{Offline-Pack}(\sigma_{1:j}^{fixed}, P^{prior}, \sigma_{j+1:k}^{next}) \tag{3.7}$$

producing either a feasible plan $P = (\sigma_{1:n}, T_{1:n})$ or "failure." The inputs $\sigma_{1:j}^{fixed}$ specify that j items of the prior plan P^{prior} should be kept in their previous positions, and

$\sigma^{next}_{j+1:k}$ are a sequence of $k - j > 0$ items that should be placed next. The remaining $n - k$ items can be placed in arbitrary order. Specifically, the result must satisfy $\sigma_{1:j} = \sigma^{fixed}_{1:j}$, $\sigma_{j+1:k} = \sigma^{next}_{j+1:k}$, and each fixed transform T_j for $j \in \sigma^{fixed}_{1:k}$ matches the corresponding transform in P^{prior}.

The offline planner used here is a constructive, heuristic method that is easily modified to handle the required changes. In Sec. 3.4 we also consider offline packing heuristics that make the job of generating a nondeterministic plan easier, but these are not strictly necessary to ensure completeness of our method.

3.3.2 Compatibility

A naive algorithm to solve NDOP would compute a packing plan for all $n!$ orderings. However, the notion of *plan compatibility* allows us to validate large numbers of orderings for lightly-interdependent plans. For example, if we ignored manipulation feasibility and equilibrium constraints, there is no sequential dependence between any two items, and hence all orderings of items would be feasible under the following policy: *when an item arrives, just place it in its planned location.* We define compatibility as follows:

Definition 8 (Compatible ordering)**.** *A packing plan $P = (\sigma_{1:n}, T_{1:n})$ is* compatible with an ordering $\sigma'_{1:n}$ *if the reordered plan $P' = (\sigma'_{1:n}, T_{1:n})$ is feasible.*

Hence, we can recast the problem of generating a feasible packing policy as one of generating a set of feasible plans with sufficient coverage as follows:

Definition 9 (NDOP #2)**.** *Compute a set of feasible plans $P_1, \ldots, P_m$ such that for any order $\sigma_{1:n} \in S_n$, there is at least one plan compatible with $\sigma_{1:n}$.*

Our NDOP solver formulates a packing policy as a set of packing plans $P_1, \ldots, P_m$ along with their associated *constraint dependency graphs* (CDGs) $G_1, \ldots, G_m$ as defined in Sec. 3.3.3. An individual packing plan can be used for the set of orderings

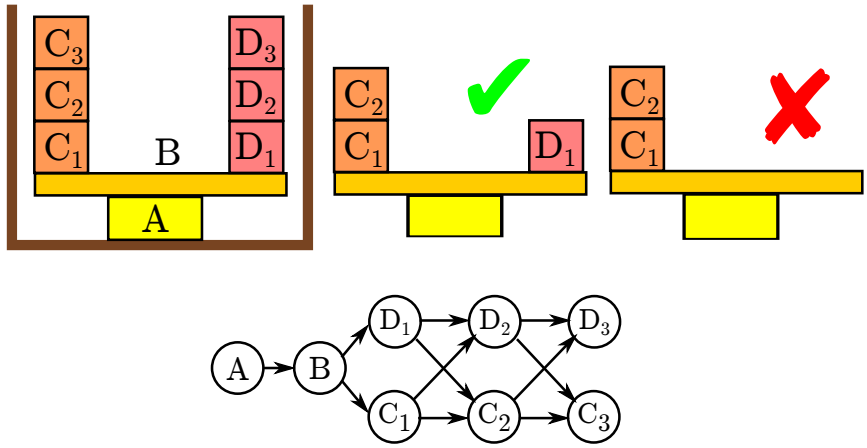


FIGURE 3.3: A plan and its dependency graph. C_2 requires D_1 to be present to maintain the stability constraint, because otherwise the imbalanced weight on B would cause tipping. Similarly, D_2 depends on C_1, and so forth for C_3 and D_3. This CDG is compatible with orders of the form $AB(C_1D_1)(C_2D_2)(C_3D_3)$ where the (XY) denotes either XY or YX.

that are compatible with its dependency graph. If the union of the m sets of compatible orderings covers S_n, then we are done. If not, we find an incompatible ordering using Alg. 5, and generate a new plan for this ordering.

A QOP solver must address the problem that if any two plans share the same order prefix, each item location in the prefix must be the same. Our algorithm uses the same CDG data structure to calculate compatibility while generating an optimized policy tree.

3.3.3 Constraint dependency graphs

A fundamental data structure that will allow us to verify compatibility is the constraint dependency graph (CDG). This structure (Fig. 3.3) explicitly models the dependencies between items, so that compatibility can be quickly verified.

Definition 10 (CDG)**.** *The CDG of a feasible plan P is a graph on vertices $\mathcal{I}$ that has an edge (u, v) if some feasibility constraint requires item u to be placed before item v.*

We can see that a CDG is a directed acyclic graph (DAG), because if there were a cycle in the graph, by transitivity an item on the cycle would need to be placed before itself. Moreover, a CDG can be replaced by its transitive reduction with no loss in compatibility information.

To construct a CDG $G = (\mathcal{I}, E)$ of a plan $P = (\sigma_{1:n}, T_{1:n})$, we do so in incremental fashion by testing all pairwise constraints. Observe that there is no edge $(\sigma_j, \sigma_i) \in E$ for $i < j$, and we need not add edges (u, σ_i) for any ancestors of σ_i already in the CDG. For each index i in increasing order, we check all $u \in \sigma_{1:i-1}$ for a dependency in reverse packing order. First, if u is an ancestor of σ_i, it is skipped because σ_i is already dependent on u. Next, the manipulation feasibility constraint of u is checked against σ_i. If so, we add an edge (u, σ_i). If not, we proceed to check equilibrium of the partial stack that includes $\sigma_{1:i}$ but omits u and all descendants of u. If there is no equilibrium solution, we add an edge (u, σ_i) (see Fig. 3.3). It should be noted that there exist scenarios that are stable if a single predecessor item is removed, but unstable if multiple predecessors are removed. These examples, however, are convoluted "multiple see-saw" constructions, and would be highly unlikely to be generated by an offline packing planner.

An ordering is compatible with a plan P iff it does not violate any dependency in P's CDG. In other words, a feasible packing plan P with a dependency-free CDG $G = (\mathcal{I}, \emptyset)$ is compatible with all orderings. More precisely, we can state:

Lemma 1. *Let $G = (\mathcal{I}, E)$ be the CDG of a feasible plan P. An ordering $\sigma'_{1:n}$ is incompatible with P iff there exists indices $u < v$ such that $(\sigma'_v, \sigma'_u) \in E$.*

In other words, a compatible ordering obeys all pairwise ordering constraints specified by the edges of the CDG.

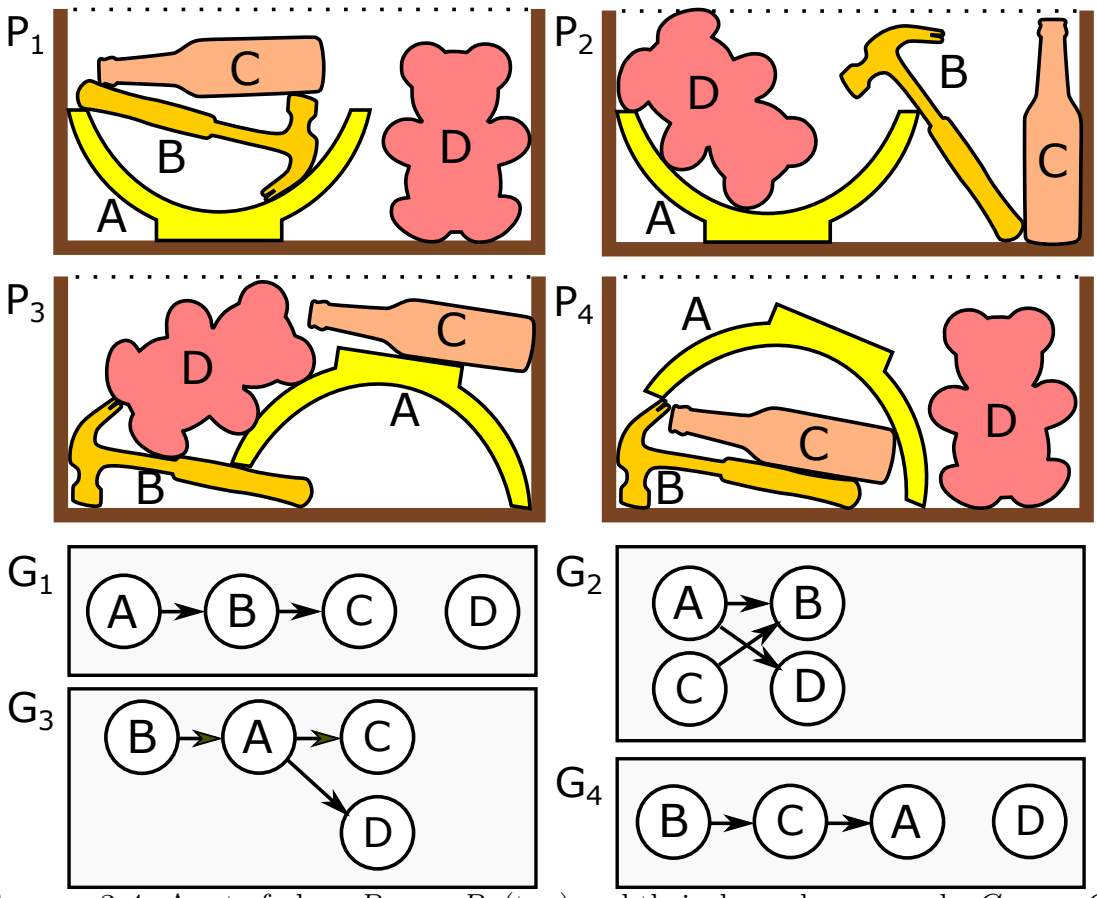


FIGURE 3.4: A set of plans $P_1, \ldots, P_4$ (top) and their dependency graphs $G_1, \ldots, G_4$ (bottom). For any ordering beginning with A, there is at least one plan (P_1 or P_2) compatible with it. But for any ordering beginning with BDA, CB, CD, DAC, or DC, no plans are compatible.

3.3.4 Coverage verification

A key subroutine in our algorithm is to verify whether a set of packing plans $P_1, \ldots, P_m$ is compatible with all orderings in S_n, and if not, to generate a counterexample (i.e., incompatible ordering). An example is shown in Fig. 3.4. Let us reduce this to a combinatorial problem of validating whether a set of dependency graphs is compatible with all orderings, and call it DEPSET-COMPAT.

We present a recursive algorithm, which tries assigning each unassigned vertex v, and recurses on the subset of plans in which v is a root. There are two base cases:

1. There exists a vertex v that is not a root in any G_i. Then, any ordering that starts with v is a counterexample.

2. G_i has no edges for some plan P_i. The policy is feasible because P_i is compatible with all orderings.

To verify faster, we also perform a *singleton pruning* step: if a vertex v is a singleton (has no neighbors) in every $G_1, \ldots, G_m$, then v can be safely ignored. This is because v can be assigned at any point without affecting dependencies.

Algorithm 5: Verify-CDG-Coverage($\mathcal{I}, \mathcal{G}$)

```
   input  : a set of items I
            list of dependency graphs G = (G_1, ..., G_m)
 1 if there exists v ∈ I that is not a root in any graph G_i then return "v
    incompatible";
 2 if any G_i ∈ G has no edges then return "all compatible";
 3 Remove all vertices v from I that are singletons in every graph G_i ∈ G;
 4 for v ∈ I do
 5 |   G^v ← ();
 6 |   for i = 1, ..., m do
 7 |   |   if v is a root in G_i then
 8 |   |   |   Append G_i to G^v, but with v removed;
 9 |   end
10 |   r ←Verify-CDG-Coverage(I/{v}, G^v);
11 |   if r = "σ_{1:j} incompatible" then
12 |   |   return "v, σ_{1:j} incompatible"
13 end
14 return "all compatible"
```

The overall algorithm is given a vertex set $\mathcal{I}$ and a list of CDGs $\mathcal{G} = (G_1, \ldots, G_m)$ of the CDGs of $P_1, \ldots, P_m$ as input, and is listed in Alg. 5. The return value is either "all compatible" or a subsequence of $\mathcal{I}$ that is incompatible with every dependency graph. Line 1 processes the first base case, and line 3 processes the second. Line 3 performs the singleton pruning step, and Lines 4–14 perform the recursion. Lines 5–10 compute the list $\mathcal{G}^v$ of dependency graphs that are compatible with assigning v

at the current step, but with v is removed. In Line 13, the vertex v is prepended to the counterexample of a recursive call, because the counterexample is reached after assigning v. (We note that instead of copying CDGs in Lines 5–10, it is more efficient to modify the CDGs in-place and then undo the operations after line 12.)

A counterexample can often be found faster by ordering the vertices in Line 4 using a heuristic. Our approach sorts the vertices v by the number of plans compatible with the assignment of v (i.e., have v as a root).

Alg. 6 solves NDOP using this subroutine.

Algorithm 6: NDOP

input : a set of items $\mathcal{I}$
output: a solution set of plans $\mathcal{P}$, or "failure"

```
1  E ←empty-list;
2  P ←empty-list;
3  while true do
4  |   r ← Verify-CDG-Coverage(I, E);
5  |   if r = "all compatible" then return P;
6  |   Let σ_{1:ℓ} be the incompatible ordering in r;
7  |   P ←Offline-Pack(nil, nil, σ_{1:ℓ});
8  |   if P = "failure" then return "failure";
9  |   Add P to P;
10 |   Add CDG(P) to E;
11 end
```

3.3.5 Quasi-online packing

Due to the need for shared transforms, QOP is not as amenable to elimination of orderings via compatibility verification. A naïve method for QOP would build a policy tree by enumerating all possible orders and ask for compatible plans.

Specifically, let N be a node in the policy tree at depth ℓ, which is associated with the ℓ'th step of a feasible plan $P = (\sigma_{1:n}, T_{1:n})$. For all non-placed items $\sigma'_{\ell+1} \notin \sigma_{1:\ell}$, we could call:

$$P' \leftarrow \text{Offline-Pack}(\sigma_{1:\ell}, P, \sigma'_{\ell+1}). \tag{3.8}$$

If P' ="failure", then failure is returned. Otherwise, P' is associated with a new child of N in the tree corresponding to the choice $\sigma'_{\ell+1}$, and the search can proceed recursively. Note that if $\sigma_{\ell+1}$ was already the $\ell+1$'th item in P, replanning is unnecessary and we can just set $P' = P$. With this check, only $O(n!)$ calls to the offline planner are needed.

This procedure can be optimized by observing that all items that are roots of the dependency subgraph $CDG(\sigma_{\ell+1:n}, (T_{\sigma_{\ell+1}}, \ldots, T_{\sigma_n}))$, can reuse P. In fact, all *combinations of roots* can reuse P. Moreover, once roots have been assigned, any newly created children can also reuse it.

To exploit this, our QOP planner performs a depth-first search according the the pseudocode given in Algs. 7 and 8. Each search node N is associated with:

- A packing sequence $\sigma_{1:\ell}$,
- A transform T_{σ_ℓ} for σ_ℓ,
- A list of plans $\mathcal{P}_N$ compatible with $\sigma_{1:\ell}$ (i.e., all T_j match, for each fixed $j \in \sigma_{1:\ell}$).

Each plan $P_i \in \mathcal{P}_N$ is associated with a dependency subgraph G_i, which is the subgraph P_i's CDG induced by the unpacked vertices $\{\sigma \notin \sigma_{1:\ell}\}$. Line 3 gives the single-layer packing base case that allows early termination. Lines 5–16 proceed in depth-first search fashion to enumerate children of N. For each item arrival $\sigma_{\ell+1}$ a child node C is generated. If at least one plan in $\mathcal{P}$ is compatible with $\sigma_{\ell+1}$, which means that $\sigma_{\ell+1}$ is a root of some G_i (Lines 6–7), then replanning is not performed, and the choice $T_{\sigma_{\ell+1}}$ is fixed. If multiple plans are compatible, the value of $T_{\sigma_{\ell+1}}$ that is compatible with the most plans is used. $\mathcal{P}_C$ is then set to the set of plans in $\mathcal{P}_N$ for which $\sigma_{\ell+1}$ is a root of the dependency subgraph, and whose placement of $\sigma_{\ell+1}$ matches $T_{\sigma_{\ell+1}}$. If no plan is compatible (Lines 9–13), then Offline-Pack is called as

normal, and $\mathcal{P}_C$ is set to contain only the newly generated plan P'. Moreover, we add P' to the sets $\mathcal{P}_A$ for any ancestor of N (inclusive), which enables subsequent siblings, siblings of parents, etc. to use P' and avoid additional planning (Line 13).

Algorithm 7: QOP-Recurse(N)

```
   input : policy tree node N at depth ℓ
 1 Let σ_{1:ℓ} be the sequence of packed items in N;
 2 Let P_N be the set of plans in N;
 3 if for any P ∈ P_N the subgraph of P induced by {σ ∉ σ_{1:ℓ}} has no edges
     then return "success";
 4 for all items σ_{ℓ+1} ∉ σ_{1:ℓ} do
 5     if any plan in P_N is compatible with σ_{1:ℓ+1} then
 6         Let T_{σ_{ℓ+1}} be the location compatible with the most plans in P_N;
 7         P_C ← {P' ∈ P_N |  P' is compatible with T_{σ_{ℓ+1}}};
 8     else
 9         Let P be any plan in P_N, or nil if P_N = ∅;
10         P' ←Offline-Pack(σ_{1:ℓ}, P, σ_{ℓ+1});
11         if P' = "failure" then return "failure";
12         P_C ← {P'};
13         For all ancestors A of N, add P' to P_A;
14     end
15     C ← add-child(N, σ_{ℓ+1}, P_C);
16     if QOP-Recurse(C) fails then return "failure";
17 end
18 return "success"
```

Algorithm 8: QOP()

```
19 root ← make-node(nil, ∅);
20 if QOP-Recurse(root) is successful then return root;
21 else return "failure";
```

3.3.6 Analysis

Here we show that NDOP inherits the completeness properties of the offline planner, but QOP is incomplete. Even when the offline planner is incomplete, when the NDOP or QOP result is not "failure", the solution is correct. We also analyze the behavior of Verify-CDG-Coverage and demonstrate that it is NP-complete.

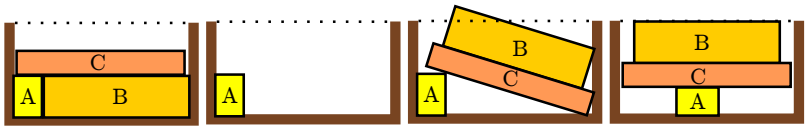


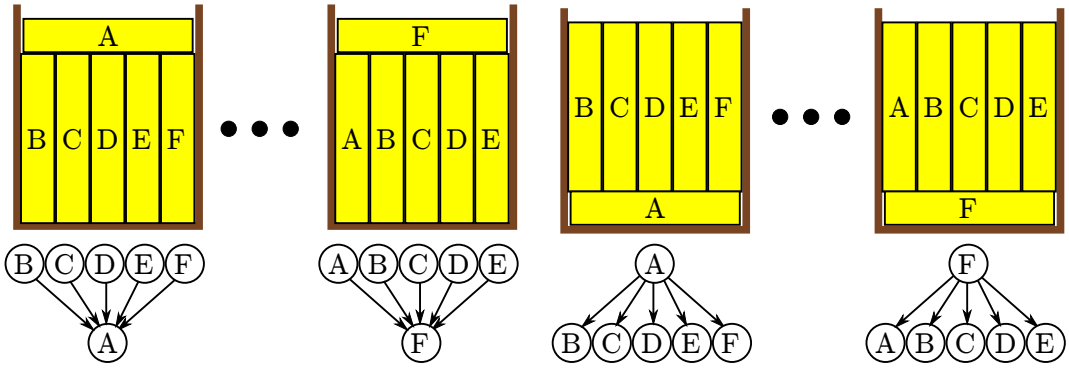
FIGURE 3.5: An example showing that QOP (Alg 8) is not necessarily complete even with a complete offline planner. (a) The first recursive call produces a feasible plan with A placed first. (b) Once item A is placed in the planned location, the plan is infeasible for order ACB, as shown in (c). On the other hand, if A were placed as in (d), a feasible QOP solution could be found.

Correctness and completeness

NDOP inherits its completeness from the offline packing planner. To see this, first observe that whenever NDOP returns a solution (Line 6), this solution is correct, because for all orderings $\sigma_{1:n} \in S_n$, Alg. 5 has shown that the solution contains some plan that is compatible with $\sigma_{1:n}$. Now consider the case where NDOP returns "failure." This can only occur when Offline-Pack returns failure for a partial ordering $\sigma_{1:j}$ (Line 7). If Offline-Pack is complete, then there is indeed no solution compatible with this ordering, and hence NDOP returns failure correctly. If it is incomplete, then NDOP may return failure incorrectly.

Assuming Verify-CDG-Coverage takes negligible time, the worst-case running time of NDOP occurs when all n items are stacked upon one another. In this case, all $n!$ possible orderings must be examined for feasibility.

Unlike NDOP, QOP is not necessarily complete even if the offline planner is complete. This is because the offline planner may commit early to a bad choice because assumes it has control over future item ordering, as illustrated in Fig. 3.5. However, if it does return a solution, then this solution is feasible even if a heuristic offline planner is used.



(a) Verification is $O(n!)$ (b) Verification is $O(n^2)$

FIGURE 3.6: (a) Worst-case behavior of Verify-CDG-Coverage occurs in an instance with n plans, where $n-1$ "books" are stacked vertically with the n'th book stacked horizontally on top. Every possible order of $k \leq n$ items is compatible with a set of $n-k+1$ plans, and a recursion depth of n is required. (b) With a slightly different stacking, the dependency graphs are reversed. Only a depth 1 recursion is needed due to the singleton pruning step, so running time is polynomial.

DEPSET-COMPAT is NP-Complete

We observe that Verify-CDG-Coverage terminates extremely quickly in many cases, but can exhibit exponential behavior. An example is shown in Fig. 3.6.a, in which each of the $m = n$ plans has one item depending on all other items. At each level ℓ of the recursion tree, there are $n - \ell$ valid CDGs, and all $n - \ell$ vertices are valid. Hence, the function is called $O(n!)$ times. In fact, we prove the following theorem:

Theorem 2. *DEPSET-COMPAT is NP-complete.*

Proof. The proof is via polynomial time reduction from 3-SAT. A 3-SAT instance consists of n Boolean variables $x_1, \ldots, x_n$ and a logical expression in disjunctive normal form, consisting of m clauses

$$(y_{11} \vee y_{12} \vee y_{13}) \wedge \cdots \wedge (y_{m1} \vee y_{m2} \vee y_{m3}) \tag{3.9}$$

where y_{ij} indicates either a variable or its negation, i.e., $y_{ij} = x_k$ or $y_{ij} = \neg x_k$. We transform any 3-SAT instance in this form into the complement of a DEPSET-COMPAT instance on $2n$ vertices and up to m dependency graphs. That is, when

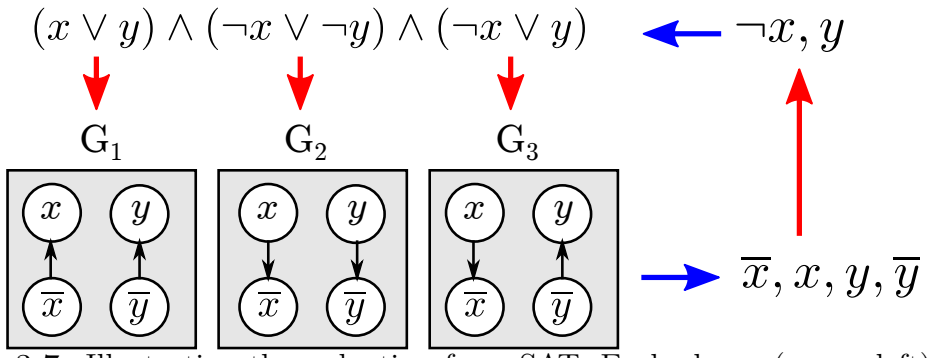


FIGURE 3.7: Illustrating the reduction from SAT. Each clause (upper left) is converted into a dependency graph (lower left), and a counterexample (lower right) ordering corresponds to a SAT solution (upper right).

3-SAT has a solution, the DEPSET-COMPAT version returns an incompatible ordering which corresponds to a 3-SAT solution, and when 3-SAT has no solution, the DEPSET-COMPAT version returns "all compatible." This construction is illustrated in Fig. 3.7 for a 2-SAT instance.

Specifically, let $\mathcal{I} = \{v_1, \ldots, v_n, \overline{v}_1, \ldots, \overline{v}_n\}$ be the vertex set. Consider the i'th conjunctive clause in (3.9). First, if the same variable and its negation appear in the same clause (e.g., $x_4 \vee \neg x_4 \vee x_6$), we drop the clause because it is satisfied via any assignment. Otherwise, we construct a dependency E_i as follows. If $y_{ij} = x_k$ for some k, construct an edge $(\overline{v}_k, v_k)$. If $y_{ij} = \neg x_k$, construct an edge $(v_k, \overline{v}_k)$. In the first case, this means that if this dependency is violated, then v_k will appear before $\overline{v}_k$ in the ordering. In the second case, the reverse is true. This is repeated for each $j = 1, 2, 3$ and $i = 1, \ldots, m$.

If the DEPSET-COMPAT$(\mathcal{I}, E_1, \ldots, E_m)$ instance constructed in this way returns an incompatible ordering, we observe whether each v_k appears before $\overline{v}_k$. If so, we assign $x_k \leftarrow T$, and if not, $x_k \leftarrow F$. The variables $x_1, \ldots, x_k$ are then a solution to 3-SAT. This holds because in every clause $y_{i1} \vee y_{i2} \vee y_{i3}$, the dependency graph E_i is violated in such a way that makes the clause true.

Conversely, if the 3-SAT instance has a solution $(x_1, \ldots, x_n)$, the DEPSET-

COMPAT instance has an incompatible ordering. It is constructed as follows: place v_k before $\overline{v_k}$ if $x_k = T$, and $\overline{v_k}$ before v_k if $x_k = F$. This ordering violates at least one constraint in each dependency graph.

Since each step in the reduction is polynomial time and NP-complete, DEPSET-COMPAT is NP-hard. It is also in NP, since a nondeterministic recursion could enumerate all possible orderings and check their validity in $O(mn)$ time. □

What is interesting about this reduction is that DEPSET-COMPAT is hard even if restricted to seemingly easy classes of dependency graphs, e.g., separable, bipartite graphs with at most 3 dependencies! Experimentally, we have observed that DEPSET-COMPAT problems corresponding to hard 3-SAT problems (e.g., with clause-to-variable ratio of ∼4.24 [Fre96]) also exhibit exponential complexity when solved via Alg. 5.

3.4 Planning Heuristics

Although DEPSET-COMPAT is NP-complete, computation time of NDOP and QOP is dominated by time spent in the offline planner, because each plan requires searching over 6D object pose. The number of plans requested, and hence overall running time, is greatly dependent on the number of orderings compatible with previous offline plans. Hence, it would be beneficial if the offline planner would generate packing plans that maximize compatibility. We employ some heuristics that speed up the approach in common scenarios.

3.4.1 Dependency minimization heuristic

Constructive packing chooses an item's location based on certain placement heuristics, such as deepest-bottom-left-first (DBLF) [WGC+10], or heightmap minimization (HM) to maximize packing density. For nondeterministic packing, we would like to generate plans with few dependencies. We introduce a *dependency count*

(DC) heuristic that measures the number of items underneath the item at the given placement (i.e., number of ancestor nodes in the CDG). Our implementation uses a heuristic that is a weighted sum of HM and DC.

3.4.2 Matching prior placements

In QOP it is beneficial for the offline planner to place as many "free" items (i.e., those not in $\sigma_{1:j}^{fixed}$ or $\sigma_{j+1:k}^{next}$) as possible in the same location as the prior plan, since this will maximize the likelihood that the plan is compatible with other branches in the search tree. To implement this heuristic, when packing a free item, the location in P^{prior} is checked for feasibility before any other locations are tested.

3.4.3 Container optimization heuristics

During container optimization, it is helpful to limit exponential growth in running time by replacing the infinite loop in Line 3 of Alg. 6 with a fixed number of iterations, or break the recursion of QOP after the policy graph has grown too large. As the containers grow wider / longer, the number of dependencies decreases because all items will be packable in fewer layers. With a large enough container, all items are packable in a single layer. Hence, if there exists a sufficiently large container, the optimization version will always terminate with a feasible, but possibly suboptimal solution.

3.5 Packing with Equivalence Classes

A straightforward extension of the NDOP and QOP problems is to handle equivalent items in the packing process. This is a common case for larger itemsets. Because items in the same equivalence class are replaceable, it reduces the number of possible orders that need to be considered by the nondeterministic packer. For example, if a consumer orders 5 item As and 3 item Bs, then the set of possible orderings is the

set of unique strings containing 5 As and 3 Bs. This gives only 56 possible orders compared to $8! = 40,320$ in the case where all 8 objects are distinct.

To address this case, an equivalence relation $\sim$ is defined over the itemset $\mathcal{I}$, and the quotient space $\tilde{\mathcal{I}} = \mathcal{I}/\sim$ is the set of unique items. The notion of CDG compatibility must be extended to allow for equivalence classes. We use a recursive definition. Let $G = (\mathcal{I}, E)$ be the CDG of a feasible plan P. An ordering $\sigma'_{1:n}$ is compatible with G if:

- $n = 0$ or
- σ'_1 is *equivalent to* a root in G AND $\sigma'_{2:n}$ is compatible with G', where G' is G with σ'_1 removed.

Determining whether $\sigma'_{1:n}$ is compatible with a plan is somewhat more expensive than the standard $O(|G|)$ procedure. Any arriving item can be matched with any root in the same equivalence class, so determining compatibility requires maintaining multiple hypothetical matches (and hence, multiple reduced CDGs G'). We verify compatibility in depth-first fashion, speeding up search using the all-root base case and treating all singletons as equivalent.

3.5.1 NDOP with Equivalence

Using this definition we can extend Verify-CDG-Coverage to handle equivalence classes as listed in Alg .9. The algorithm only has minor modifications to handle equivalence classes when testing for roots and singletons, and rather than looping over items it loops over classes. Simply using Verify-CDG-Coverage-Eq in NDOP (Alg. 6) will address the problem of NDOP planning with equivalence classes.

3.5.2 QOP with Equivalence

Alg. 10 presents a solver for QOP with equivalence classes that modifies QOP-Recurse in similar fashion to Verify-CDG-Coverage-Eq. The main difference in this case is

Algorithm 9: Verify-CDG-Coverage-Eq($\mathcal{I}, \mathcal{G}$)

input : a set of items $\mathcal{I}$
list of dependency graphs $\mathcal{G} = (G_1, \ldots, G_m)$

1 **if** *there exists* $v \in \mathcal{I}$ *that is not equivalent to a root in any graph* G_i **then return** "v incompatible";
2 **if** *any* $G_i \in \mathcal{G}$ *has no edges* **then return** "all compatible";
3 **while** $\exists c \in \tilde{\mathcal{I}}$ *such that there is a singleton of class* c *in every graph* $G_i \in \mathcal{G}$ **do**
4 | Remove an item of class c from $\mathcal{I}$;
5 | Remove an singleton of class c from each $G_i \in \mathcal{G}$;
6 **end**
7 **for** $c \in \tilde{\mathcal{I}}$ **do**
8 | $\mathcal{G}^c \leftarrow ()$;
9 | **for** $i = 1, \ldots, m$ **do**
10 | | **for** *roots* v *in* G_i *with class* c **do**
11 | | | Append G_i to $\mathcal{G}^c$, but with v removed;
12 | | **end**
13 | **end**
14 | Let v be an item in $\mathcal{I}$ with class c;
15 | $r \leftarrow$Verify-CDG-Coverage-Eq($\mathcal{I}/\{v\}, \mathcal{G}^c$);
16 | **if** $r =$ "$\sigma_{1:j}$ *incompatible*" **then**
17 | | **return** "$v, \sigma_{1:j}$ incompatible"
18 **end**
19 **return** "all compatible"

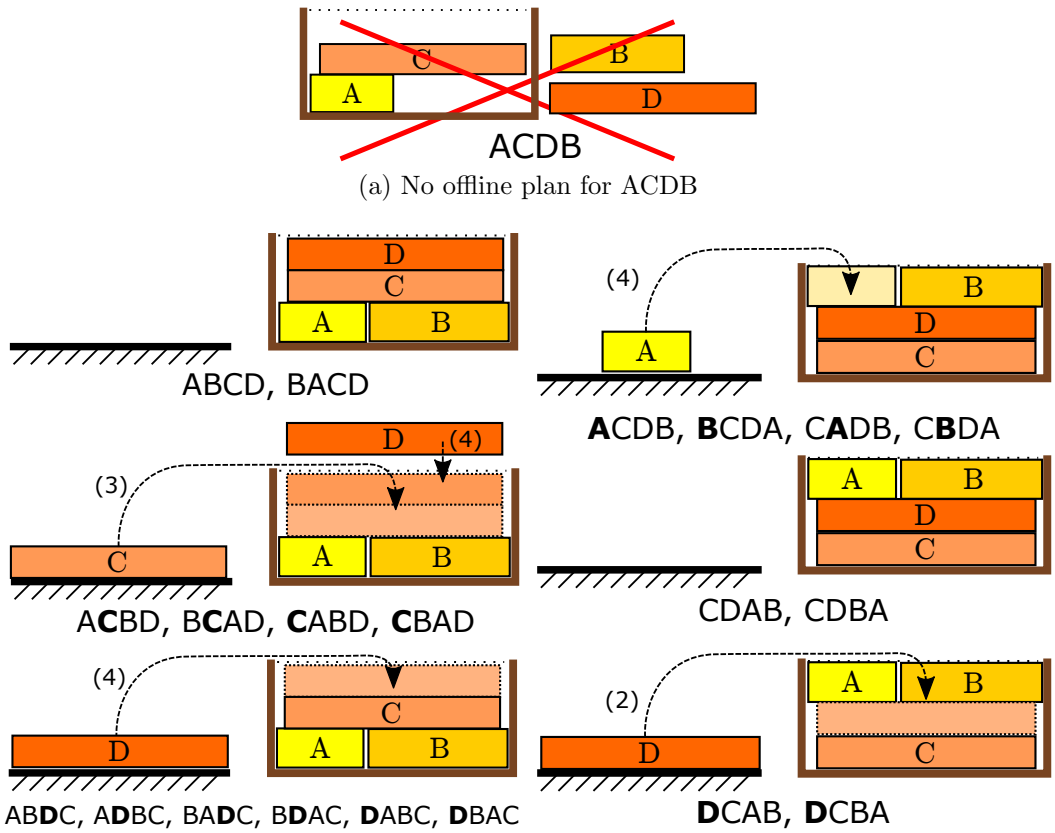
that the we maintain the item sequence while building the policy tree, but the branching should be understood as no longer performed on items but rather equivalence classes. Moreover, whereas in the prior QOP-Recurse algorithm each plan $P_i \in \mathcal{P}_N$ is associated with single dependency subgraph associated with the removal of packed items, in this case each plan has potentially multiple compatible subgraphs. These subgraphs can be propagated through the recursion along with N in a manner similar to Lines 8–13 of Verify-CDG-Coverage-Eq.

Algorithm 10: QOP-Recurse-Eq(N)

Let $\mathcal{I}$ be the set of items not in $\sigma_{1:\ell}$;
if *for any* $P \in \mathcal{P}_N$ *the subgraph of* P *induced by* $\mathcal{I}$ *has no edges* **then**
  return "success";
for *all classes* $c \in \tilde{\mathcal{I}}$ **do**
  Let $\sigma_{\ell+1}$ be an item in $\mathcal{I}$ with class c;
  if *any plan in* $\mathcal{P}_N$ *is compatible with an item of class* c **then**
    Let T_c be the location compatible with the most plans in $\mathcal{P}_N$;
    $\mathcal{P}_C \leftarrow \{P' \in \mathcal{P}_N |$ $\quad P'$ is compatible with $T_c\}$;
  else
    Let P be any plan in $\mathcal{P}_N$, or *nil* if $\mathcal{P}_N = \emptyset$;
    $P' \leftarrow$Offline-Pack$(\sigma_{1:\ell}, P, \sigma_{\ell+1})$;
    if $P' =$ *"failure"* **then return** "failure";
    $\mathcal{P}_C \leftarrow \{P'\}$;
    For all ancestors A of N, add P' to $\mathcal{P}_A$;
  end
  $C \leftarrow$ add-child$(N, \sigma_{\ell+1}, \mathcal{P}_C)$;
  if *QOP-Recurse-Eq*(C) *fails* **then return** "failure";
end
return "success"

3.6 Buffered Nondeterministic Planning

Lastly, we present *k-buffered* NDOP and QOP variants, which are hybrids of NDOP / QOP and the buffered online packing problem, in which the packer has a "buffer" that can hold up to k items before having to make a container choice [CJ01, EK09]. This formulation spans a continuum between offline packing ($k \geq n-1$) and nondeterministic planning ($k = 0$). It is also fairly realistic for packing setups in warehouses, since it is typically possible to augment the container with a temporary space where items may be set aside. Moreover, multi-armed packing robots (or humans) can temporarily "store" items in their hands, where k is the number of hands minus 1. The overall benefit of this approach is that the number of plans that must be computed by the offline planner decreases with k. Specifically, if $N(k)$ is the number of offline plans computed, $N(k)$ is non-increasing with k, and if the buffer is large enough the



(a) No offline plan for ACDB

(b) Plan 1

(c) Plan 2

FIGURE 3.8: (a) A problem that has no standard NDOP solution, but is feasible with a buffer of size $k = 1$. Three plans are suffice to cover all 4! orderings. (b) Plan 1 covers 12 orderings: 2 without using the buffer, 4 placing C in the buffer, and 6 placing D in the buffer. Bold letters indicate the buffered item, and numbers indicate the packing order. (c) Plan 2 covers 8 orderings. Plan 3 (not shown) swaps C and D to cover the remaining 4 orderings.

problem becomes equivalent to standard offline planning ($N(n-1) = 1$).

The other potential benefit of a buffered approach is that some infeasible NDOP / QOP problems can become feasible, because a buffer provides more control over the packing order so items might be packed more densely. Fig. 3.8 gives an example of such a problem.

3.6.1 Buffered NDOP

In order to compute and use k-buffered NDOP policies, we must cope with fact that the arrival and packing order are not necessarily the same; an arrived item not immediately compatible with a plan can be set aside for later packing. Three changes are made to the NDOP algorithm. First, the notion of compatibility must be revised to manage the buffer and decide when to put an item aside. Second, the NDOP search must be augmented to maintain buffer states. Third, the approach of generating a single counterexample ordering for offline planning is not necessarily going to succeed, because there may be a feasible ordering that uses the buffer differently than expected.

We start by defining the notion of *buffer compatibility*. Given a plan, its CDG, and a new order, there is a straightforward greedy policy for packing while maintaining a minimal buffer. There are two rules to follow for a given item v and buffer B:

1. If all predecessors of v in the CDG are packed, then pack v. Otherwise, put v in the buffer.

2. If, for any buffered item $w \in B$, all predecessors of w in the CDG are packed, then pack w (removing it from the buffer).

If the number of items held in the buffer at any time during this process is less than or equal to k, then there exists a feasible packing order for this plan. This can be checked in $O(nk)$ time by incrementally removing root nodes from the CDG when the corresponding item is packed, since Step 1 is $O(1)$ time and Step 2 is $O(k)$ time. Let us call Step 2 *cleaning the buffer*. Note that cleaning the buffer must apply the check in Step 2 repeatedly if there are multiple items in the buffer that are unblocked by the packing of a new item. For example, if the CDG is $C \rightarrow B \rightarrow A$ and the packing order is A, B, C, then the buffer is $\{A, B\}$ when item C is packed, and packing C

unblocks B then A in turn.

We use this reasoning in a buffered Verify-CDG-Coverage algorithm, given in Alg. 11. Given a set of plans $P_1, \ldots, P_m$ with CDGs $G_1, \ldots, G_m$, the new subroutine explores partial orderings of items while maintaining the minimal buffer $B_1, \ldots, B_m$ for each plan. The buffers are initialized to empty sets. The base cases are modified as follows:

1. there exists a vertex v that is not a root in any G_i and all $|B_j| = k$. Then, any ordering that starts with v is a counterexample.

2. The number of root nodes in some G_i is at least $n - k + |B_i|$. The policy is feasible because P_i is buffer-compatible with all orderings.

The first base case is less aggressive at declaring failure than in the non-buffered case, because even if a vertex is non-root, then it can still be added to the buffer while there is room. The second base case is more aggressive at declaring success, because we can terminate recursion even when a CDG is non-empty. If the number of non-root nodes is less than or equal to k, then the buffer can hold them all regardless of the presentation order.

The recursive step is modified so that, first, the buffer for each plan is cleaned. Then, upon visiting a vertex v, a plan P_i and its buffer B_i are retained if either v can be packed in P_i or there is room in the buffer ($|B_i| < k$). If v can be packed directly there is no sense in considering the option of placing it in the buffer, since the greedy policy is optimal.

Simply replacing Verify-CDG-Coverage with Verify-CDG-Coverage-Buf in NDOP (Alg. 6) will correctly solve buffered problems in which a non-buffered NDOP solution exists. But for infeasible NDOP problems, e.g., the one in Fig. 3.8, this approach may fail to find a buffered solution. Specifically, the Offline-Pack call in Line 7 of NDOP

Algorithm 11: Verify-CDG-Coverage-Buf($\mathcal{I}, \mathcal{G}, \mathcal{B}, k$)

input : a set of items $\mathcal{I}$
dependency graphs $\mathcal{G} = (G_1, \ldots, G_m)$
buffers $\mathcal{B} = (B_1, \ldots, B_m)$
buffer size k

Clear buffers B_i in their respective graphs G_i;
if *there exists* $v \in \mathcal{I}$ *for which* v *is not a root in any* G_i *and all* $|B_i| = k$ **then return** "v incompatible";
if *there exists* $G_i \in \mathcal{G}$ *with at least* $n - k + |B_i|$ *root nodes* **then return** "all compatible";
Remove all vertices v from $\mathcal{I}$, graphs, and buffers that are singletons in every G_i, $i = 1, \ldots, m$;
for $v \in \mathcal{I}$ **do**
    $\mathcal{G}^v \leftarrow ()$, $\mathcal{B}^v \leftarrow ()$;
    for $i = 1, \ldots, m$ **do**
        if v *is a root in* G_i **then**
            Append G_i to $\mathcal{G}^v$, but with v removed;
            Append B_i to $\mathcal{B}^v$;
        else if $|B_i| < k$ **then**
            Append G_i to $\mathcal{G}^v$;
            Append $B_i \cup \{v\}$ to $\mathcal{B}^v$;
    end
    $r \leftarrow$Verify-CDG-Coverage-Buf($\mathcal{I}/\{v\}, \mathcal{G}^v, \mathcal{B}^v$);
    if $r =$ "$\sigma_{1:j}$ *incompatible*" **then**
        return "$v, \sigma_{1:j}$ incompatible"
end
return "all compatible"

may fail on the counterexample $\sigma_{1:\ell}$ even though some offline plan may be is buffer-compatible with this order (e.g., the counterexample ACDB to plan 1 in Fig. 3.8). To ensure completeness, the offline planner must be made buffer-aware. Specifically, if there is an offline plan that *packs, buffers, or de-buffers* items in the same order as they appear in $\sigma_{1:\ell}$, it's a valid solution to the requested counterexample. Note that the plan must describe how all n items must ultimately be packed, but the planner now has more freedom to choose which items are packed first. Let us call this modified oracle *Offline-Pack-Buf*.

If the offline planner uses a heuristic item ordering strategy, as it does in our

implementation, the best way to do implement Offline-Pack-Buf is to allow it to delay packing up to k items if subsequent items have higher priority, or an item fails to be placed. (It is an open question about how to accomplish this efficiently for a complete offline planner; a correct but slow approach would enumerate all subsequences of $\sigma_{1:\ell}$ with up to k buffered objects, and try finding an offline plan for each of them.)

3.6.2 Buffered QOP

For QOP, each node in the policy tree may buffer the item rather pack it in a given location, delaying the decision of the packing location to some descendant node. This requires each QOP node N maintain a *packing list* of items and their locations $(\sigma_{N1}, T_{N1}), \ldots, (\sigma_{Nj}, T_{Nj})$ packed upon the arrival of an item. An empty list ($j = 0$) indicates the arrived item is buffered.

To build intuition, we describe how a naïve k-buffered QOP variant can be implemented by modification of the naïve QOP algorithm of Sec. 3.3.5. Each node is associated with an arrival order $\sigma_{1:\ell}$, a packing list L, a feasible plan $P = (\sigma_{1:n}, T_{1:n})$, and an optimal buffer B at step ℓ. We also define the dependency subgraph G of the CDG of P when the items in $\sigma_{1:\ell} - B$ are removed. In each recursive call, perform the following steps:

1. For each item $\sigma_{\ell+1}$ not in $\sigma_{1:\ell}$, repeat Steps 3–6:
2. Call Offline-Pack-Buf, replacing $\sigma_{1:\ell}$ in (3.8) with $\sigma_{1:\ell} - B$ and using B as the initial buffer. If this fails, return failure. (Note that this may choose to buffer $\sigma_{\ell+1}$)
3. Create a child node whose pack list is initialized with the pack or buffer decision for $\sigma_{\ell+1}$. Initialize its dependency subgraph G and buffer B
4. Continue recursively on the child.

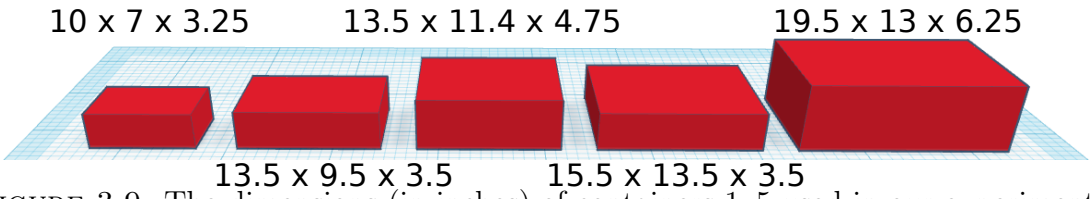


FIGURE 3.9: The dimensions (in inches) of containers 1–5 used in our experiments, in order of increasing length + girth.

If a node is a leaf, then the node's buffers should be cleared. Note that unlike NDOP, QOP actually gains power by delaying packing decisions until the buffer is full. Hence, there is no need to pack until $|B| = k$. Delaying also maximizes our ability to reuse offline plans across many policy nodes.

A more refined QOP algorithm reuses offline plans like Alg. 7. We modify each QOP search node N to contain elements:

- The arrival sequence $\sigma_{1:\ell}$ leading to and including N,
- The packing list $L_N = (\sigma_{N1}, T_{N1}), \ldots, (\sigma_{Nj}, T_{Nj})$ of decisions made upon the arrival of item σ_ℓ,
- The buffer B_N associated with the packing sequence leading to and including N
- A list of plans $\mathcal{P}_N = (P_1, \ldots, P_m)$ that are buffer-compatible with $\sigma_{1:\ell}$,
- A list of dependency subgraphs $\mathcal{G}_N = (G_1, \ldots, G_m)$ corresponding to the plans in $\mathcal{P}_N$, restricted to the complement of $(\sigma_{1:\ell} - B_N)$.

The resulting algorithm is listed in Alg. 12.

3.7 Experiments

Our experiments test the NDOP and QOP algorithms with random item sets of 3D scanned objects from the APC 2015 [Rut15] and YCB [CSB+17] datasets (94

Algorithm 12: QOP-Recurse-Buf(N, k)

input : policy tree node N at level ℓ
buffer size k

Compute dependency subgraphs $\mathcal{G}_N$;
if *any* $G_i \in \mathcal{G}_N$ *would have at least* $n - k + |B_N|$ *root nodes after cleaning* B_N **then return** "success";
for *all items* $\sigma_{\ell+1} \notin \sigma_{1:\ell}$ **do**
    if $|B_N| < k$ **then**
        $L_C \leftarrow ()$;
        $B_C \leftarrow B_N \cup \{\sigma_{\ell+1}\}$;
        $\mathcal{P}_C \leftarrow \mathcal{P}_N$;
    else if *any item in* $B_N \cup \{\sigma_{\ell+1}\}$ *is a root of any* G_i **then**
        Let $\sigma \in B_N \cup \{\sigma_{\ell+1}\}$ and T_σ be the item and location compatible with the most plans in $\mathcal{P}_N$;
        $L_C \leftarrow (\sigma, T_\sigma)$;
        $B_C \leftarrow B_N$;
        $\mathcal{P}_C \leftarrow \{P' \in \mathcal{P}_N \mid \quad P'$ is compatible with σ and $T_\sigma\}$;
    else
        $P \leftarrow$ any plan in $\mathcal{P}_N$, or *nil* if it is empty;
        $P' \leftarrow$Offline-Pack-Buf($\sigma_{1:\ell} - B_N, P, \sigma_{\ell+1}, B_N$);
        if $P' =$ *"failure"* **then return** "failure";
        $L_C \leftarrow$ the next packed item in P';
        $B_C \leftarrow$ the resulting buffer;
        $\mathcal{P}_C \leftarrow \{P'\}$;
        For all ancestors A of N, add P' to $\mathcal{P}_A$;
    end
    $C \leftarrow$ add-child($N, \sigma_{\ell+1}, L_C, \mathcal{P}_C, B_C$);
    if *QOP-Recurse-Buf*(C, k) *fails* **then return** "failure";
end
return "success"

objects total). The containers used in these experiments are the five boxes used in the Amazon Robotics Challenge 2017 (Fig. 3.9), and are sorted by the length + girth metric (length + 2×width + 2×height), a commonly-used shipping measurement. All experiments were performed on an Amazon EC2 m5d.12xlarge instance.

Our experiments use three testing datasets in which the item sets have different size: a) typical shopping carts of 2–5 items, b) large sets of 5 items, and c) stress tests with 10 items. For each category, we generate 1,000 random item sets by drawing

Table 3.1: NDOP container optimization results

Items	Success (%)	Time (mean / max, s)	# planner calls (mean / max)
2–5	99.3	73.0 / 1,813	1.3 / 9
5	96.9	94.4 / 1,417	1.6 / 14
10	64.0	1,048 / 33,300	5.2 / 118

items at random, and verifying with an offline planner that there exists is a feasible packing in one of the five containers. In the 2–5 category, 250 item sets of each size are included.

3.7.1 NDOP Results

The results for the NDOP planner, running in container optimization mode, are summarized in Tab. 3.1, and Fig. 3.10 illustrates a solution. As might be expected, the running time and the number of offline planner calls increases with the number of items, but we do not observe the exponential running time of worst-case instances. Other experiments suggest the dependency minimization heuristic reduces mean and maximum running times by approximately 20% and 50%, respectively.

Observe also that the success rate drops with increasing numbers of items, as the 10-item offline plans tend to be tightly packed even in the largest container. Note that it is not known whether an NDOP solution exists in these instances, so we cannot determine whether the solver is failing incorrectly. Fig. 3.11 shows the distribution of the minimum-cost container found. In cases with 5 or fewer items, NDOP often successfully packs in the same box as the offline planner. Observe that with 5 items, approximately 5% of test cases require container 5, even though they can be packed offline in containers 1–4. Fig. 3.12 illustrates an example of such a case.

FIGURE 3.10: An NDOP solution for packing 5 items in container 5.

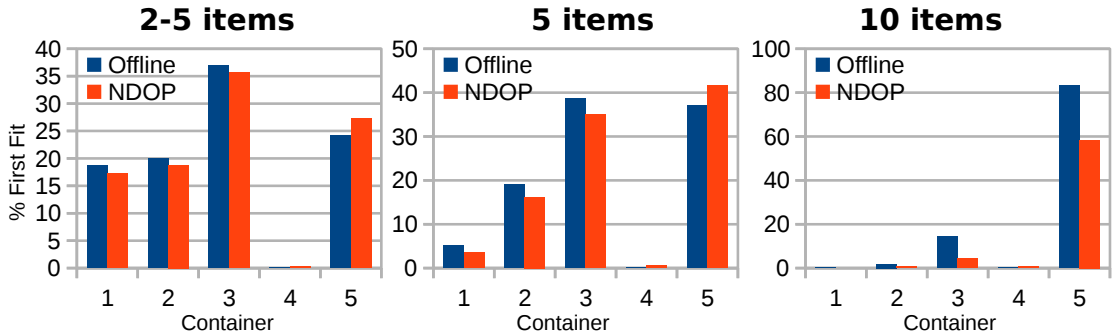
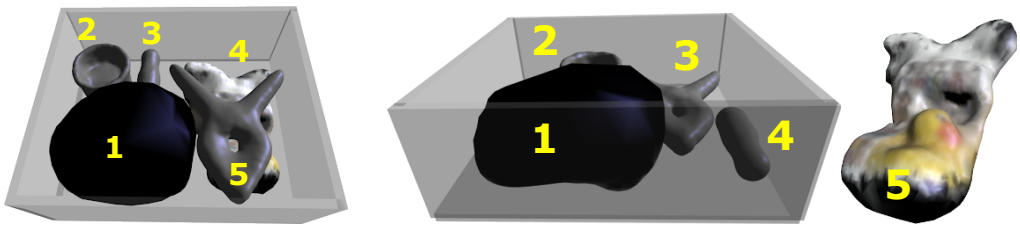


FIGURE 3.11: Distribution of the solution container for offline / NDOP container optimization and various itemset sizes. Most small instances can be packed in any order, but with more items the variability in order requires larger boxes.



(a) (b)

FIGURE 3.12: A failure case for packing 5 items in container 3. The offline planner solves for the arrival order in (a) but fails on the order in (b) because there is insufficient remaining space for the fifth item.

Table 3.2: QOP planning results in container 5

Items	Success (%)	Time (mean / max, s)	# planner calls (mean / max)
2–5	99.4	22.4 / 1,520	1.4 / 43
5	97.0	65.1 / 5,800	2.1 / 46
10	43.7	2,850 / 85,478	45.8 / 5,363

3.7.2 QOP Results

Performance results for QOP in container 5 are given in Tab. 3.2, with a representative solution shown in Fig. 3.13. Up to 5 items, the success rates are quite similar to NDOP, but the maximum running times and number of offline planner calls tend to be significantly larger. Other experiments suggest that employing the matching heuristic improves average and maximum running time by over 50%, which explains the surprising result that QOP is faster than NDOP on average. QOP struggles with 10 items, with a long-tailed distribution: 24 instances could not be solved within a 24-hour cutoff.

3.7.3 Results for Equivalence and Buffered Variants

Fig. 3.14 shows how performance changes when the buffered and equivalence class extensions to the NDOP planner are introduced. This experiment uses an artificial model of an offline planner, where the oracle always returns a valid plan, and its

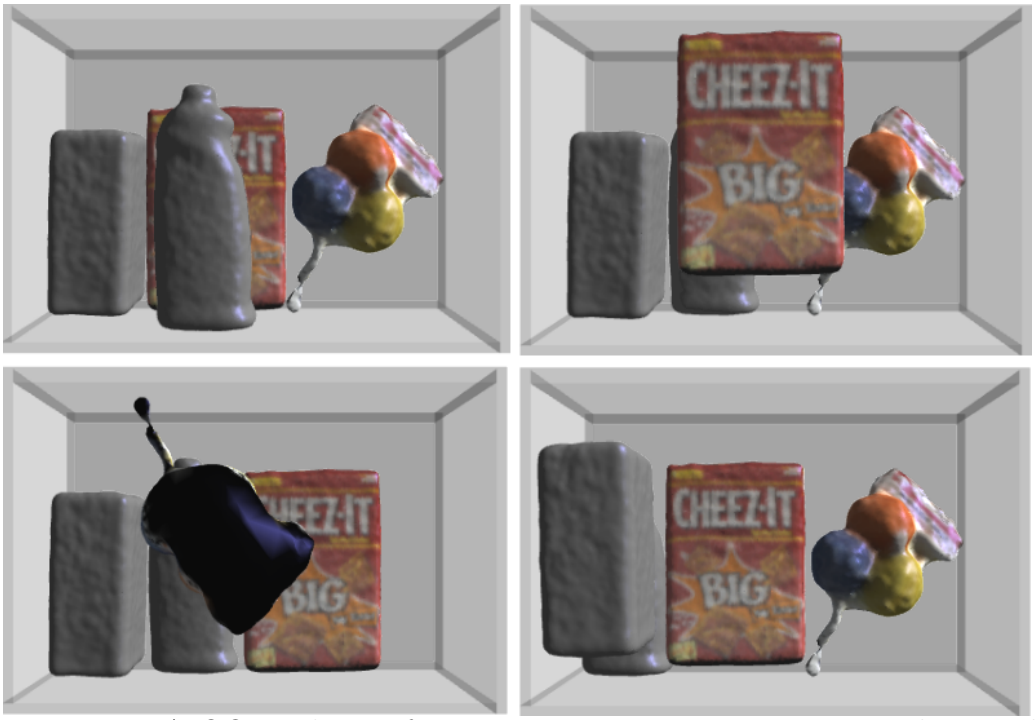
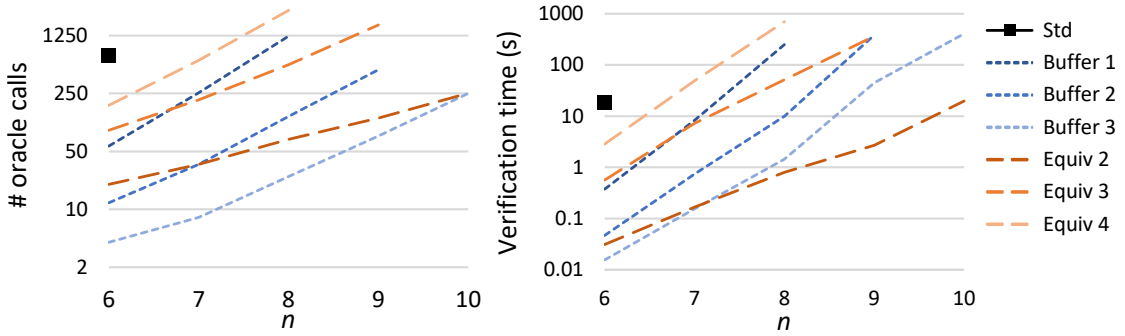


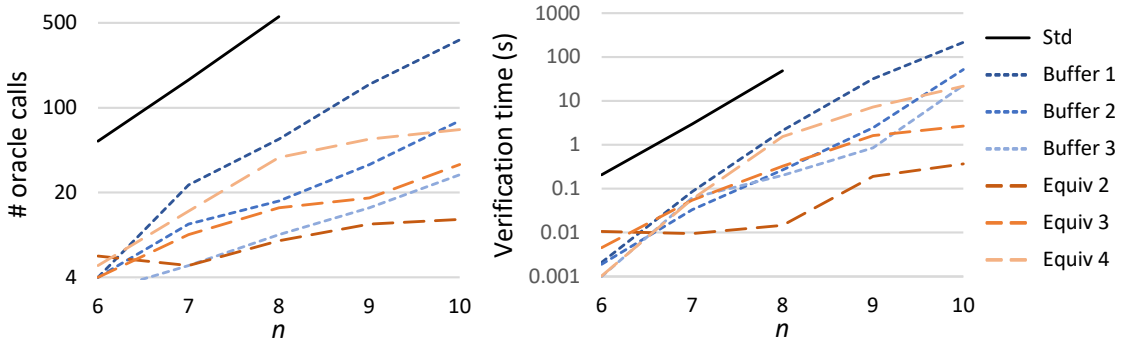
FIGURE 3.13: A QOP solution for 4 items in container 5. Due to the matching heuristic, only four plans are needed (one for each item placed last).

CDG is a random subset of the maximally dependent CDG, where each item is dependent on all prior items. (Offline planning time is negligible in this model.) Fig. 3.14.a is the worst-case where each CDG is maximally dependent, so each plan is only compatible with a single ordering and hence the standard NDOP problem must generate $n!$ oracle calls. In Fig. 3.14.b, each edge of the maximally dependent CDG is retained with probability 0.5. The total verification time budget is capped at 10 minutes, so standard NDOP cannot solve Fig. 3.14.a for $n \geq 7$ and Fig. 3.14.b for $n \geq 9$. "Equiv k" indicates that items are allocated among k equivalence classes. Each class is balanced to hold approximately the same number of items. For both dependency models, larger buffer sizes and fewer equivalence classes lead to more scalable performance.

Figs. 3.15 and 3.16 demonstrate these variants on random itemsets drawn from the APC / YCB datasets. To test the effects of equivalence classes, we generated



(a) 100% dependency model

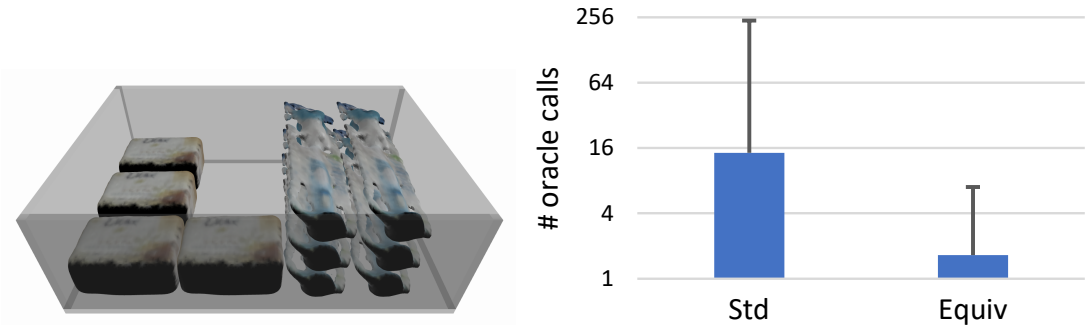


(b) 50% dependency model

FIGURE 3.14: Comparing standard NDOP performance against k-buffered and equivalence class variants, artificial examples. Number of offline plans and coverage verification time are plotted for varying numbers of items n (note the logarithmic scale). (a) Worst-case performance, with each plan inducing a fully dependent CDG. (b) Performance when each plan contains 50% of dependencies, chosen at random.

1,000 new 10-item itemsets where only 2 item classes were chosen at random, and the split between classes was chosen between 4–6 items at random. For each itemset, it was verified that with offline planning, all items fit in at least one of the 5 Amazon containers. Fig. 3.15.a demonstrates that all 10! orders of the given items can be covered in a single plan. In Fig. 3.15.b, we compare standard NDOP to the equivalence-class aware variant. With equivalence class information, the average number of offline planner calls is 1.65 on average, and 7 at most. Without, the average / maximum are 14.45 / 239.

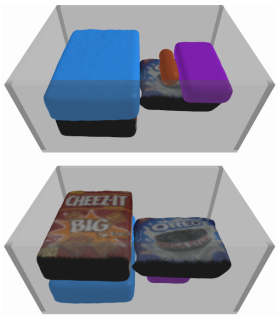
Next, Fig. 3.16 evaluates buffering. Fig. 3.16.a shows a representative output of the planner. For this example, standard NDOP finds a covering with 13 offline



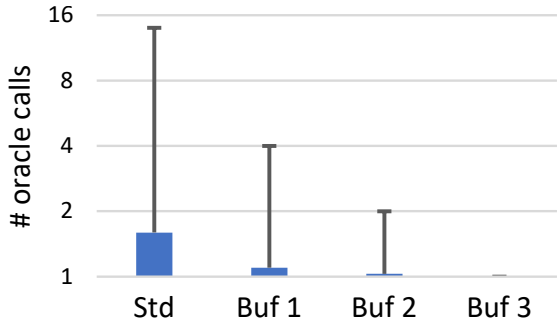
(a) (b)

FIGURE 3.15: Equivalence-class NDOP on items from the APC / YCB datasets. Two item classes are selected at random, with 4–6 items per class. (a) The planner covers all orderings with a single offline plan. (b) Oracle call counts, over 1000 itemsets. Column indicates average, error bar indicates maximum. (Note logarithmic scale.)

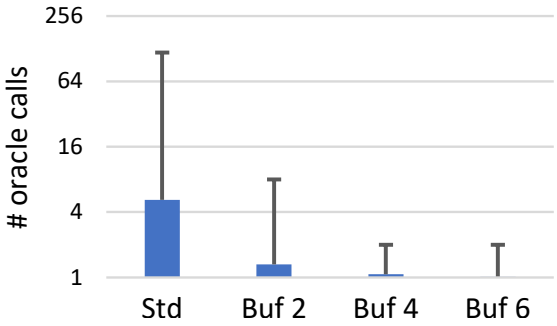
plans, but 1-buffered NDOP finds a covering with the 2 offline plans shown. The statistics in Fig. 3.16.b show that with 5-item packing, the number of oracle calls drops very close to 1 as the buffer size increases. A similar pattern is observed in 10-item packing (Fig. 3.16.c). Moreover, the success rate increases dramatically as the buffer size increases, because the larger buffer gives the planner more opportunity to solve badly-ordered arrival sequences in boxes with high packing density.



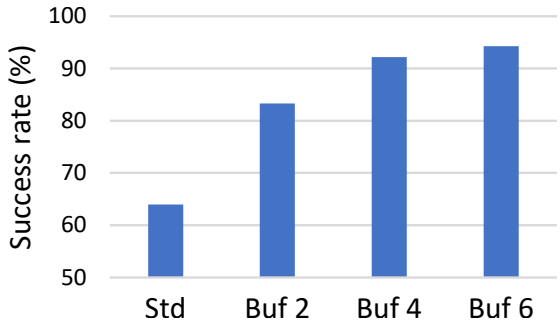
(a) 5-item, 1-buffered plan



(b) 5-item oracle calls



(c) 10-item oracle calls



(d) 10-item success rate

FIGURE 3.16: Buffered NDOP on items drawn from the APC / YCB datasets. (a) With a buffer size of 1, a 5-item solution can be covered with 2 offline plans. (b-d) Statistics over 1000 itemsets. Columns indicate average, error bars indicate maximum. (Note: oracle call counts plotted on logarithmic scale.)

4

In-hand Object Scanning via RGB-D Video Segmentation

This chapter introduces useful tools to gather 3D shape information for an arbitrary object. We propose a technique for building full 3D models of an arbitrary object via in-hand manipulations, in which the object is reoriented in front of a video camera with multiple grasps and regrasps. 3D models with colors and accurate geometry are known to increase robustness in a robot's perception and manipulation of objects. Therefore, building accurate 3D models of task-relevant objects and recording their identities could greatly enhance the autonomy of many robotic tasks in the environment.

In-hand object scanning is a convenient and inexpensive way to model a hand-sized object. Yet, many significant challenges exist in tracking the manipulated object due to fast movement, rapid appearance changes, and occlusions. We propose a novel video-segmentation-based object tracking algorithm that tracks arbitrary in-hand objects more effectively than existing techniques. It also describes a novel RGB-D in-hand object manipulation dataset consisting of several common house-

hold objects. Experiments show that the new method achieves a 6% increase in accuracy compared to top-performing video tracking algorithms and results in noticeably higher quality reconstructed models. Moreover, testing with a novice user on a set of 200 objects demonstrates relatively rapid construction of complete 3D object models.[1]

[1] This chapter is reproduced from Fan Wang and Kris Hauser, "In-hand Object Scanning via RGB-D Video Segmentation," in 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 3296-3302.

4.1 Introduction

The ability to build 3D models of the geometry and appearance of real-world objects has long been essential in many application scenarios such as art and design, manufacturing and packaging, augmented reality, and robotics. While early 3D model acquisition relied on multiple calibrated scanners or accurate motion apparatuses [RHHL02, BR02, KHRF11], the last decade has seen the introduction of low-cost RGB-D cameras like Microsoft Kinect and advances in feature detection and automatic feature matching algorithms that have made 3D object model acquisition easier than ever.

Among various 3D scanning techniques, in-hand scanning has attracted much research attention [TG17, PASV15, WWLG09, PKA15]. Methods that require an object to be stationary during the scan inherently carry the missing side problem [BR02], while in-hand scanning can reveal all sides of the object, making it possible to reconstruct the complete object without multiple model alignment in post-processing. The human hand is capable of intricate motion when manipulating objects through gripping, grasping and turning, so this is a natural way of presenting an object for scanning. It is also low-cost and flexible since it does not require expensive equipment and calibration. Many reconstruction pipelines from object-hand interaction require only an off-the-shelf RGB-D camera [TG17, PASV15].

Despite these attractive characteristics, 3D object modeling from in-hand object manipulation remains a challenging problem. To obtain a high-quality object model, the target object should be accurately separated from non-target objects. This separation is difficult since hands may be similar to the target object in terms of color, motion, or appearance, and furthermore, the target object itself undergoes significant changes in a short period of time.

This chapter presents a 3D model acquisition pipeline from in-hand interaction.

FIGURE 4.1: Some successfully reconstructed objects from a 200-item test set, scanned by a novice user of our system.

The pipeline includes a novel object tracking technique and a set of reconstruction and post-processing procedures. With this pipeline, a non-expert can scan arbitrary objects with only a hand-held single RGB-D camera and light manual annotation.

To evaluate our work, we constructed a public in-hand object manipulation dataset consisting of 13 objects from the publicly available YCB object set [CWS+15] being manipulated by hand in front of an RGB-D camera. Experiments on this dataset demonstrate that our method outperforms many state-of-the-art video segmentation algorithms in terms of tracking performance and results in higher quality 3D reconstructed models. We also asked a novice user to use our pipeline to scan 200 arbitrary items, requiring approximately 2.5 minutes of manual effort per object including scanning and annotation, and demonstrates the capability to produce high-quality reconstructed models, shown in Fig. 4.1.

4.2 Related Work

An object scanning pipeline commonly consists of object-background segmentation, multi-view registration, and registration refinements steps. Several in-hand scanning systems have been presented [WLG08, WWLG11, TG17, RHHL02, PKA15]. Weise et al.[WLG08] demonstrated real-time system reconstruction of textured objects from in-hand manipulation using coarse to fine registration and online loop closure. Tzionas et al.[TG17] showed the possibility of reconstructing featureless objects from hand motion cues, by minimizing objective functions that consist of both visual correspondences and contact correspondences.

While the registration and refinement aspects of an object scanning system have been extensively studied[WLG08, WWLG09, RHHL02, BR02, TG17], little work has been done to address the unique challenges present in foreground - background segmentation in an in-hand manipulation task, despite its importance being commonly acknowledged [WWLG09, KHRF11, SMZ+16, PKA15].

Existing in-hand scanning systems address the segmentation task with simple approaches. Hand-object segmentation are commonly addressed by skin pixel identification based on statistically color model [WLG08, TG17], or by having operators wear black gloves [WWLG09, WWLG11, RHHL02]. To remove other unwanted backgrounds, depth thresholding and black backgrounds are commonly used [TG17, RHHL02]. A more sophisticated method is to perform segmentation in phase image as in [WLG08] to remove background that remains stationary.

However, those methods are not robust enough to achieve highly accurate object-background segmentation. Color based skin-pixel segmentation, for example, can result in a high false negative rate with hands under different lighting conditions and a high false positive rate when objects being manipulated are close in color to human hands, as illustrated in Fig. 4.2. The phase image segmentation cannot remove non-

stationary background objects such as hands and arms. The black glove method might be the most robust among all, but nonetheless requires only the hand and the object to be visible during the entire scanning, which can be difficult for the operator to accomplish.

Our pipeline uses similar registration approaches to prior work, but to address hand-object separation with more generality and to higher accuracy, we solve a semi-supervised binary video segmentation problem, tailored for the hand-object segmentation task. Our method can separate objects accurately from all unwanted backgrounds, including hands of various skin tones, as well as other stationary or non-stationary objects not of interest.

Recent research on video segmentation-based object tracking can be roughly categorized into unsupervised, semi-supervised, and human-in-the-loop approaches.

Unsupervised methods are fully automatic and require no human effort during run time [LTHF17, XL16, WSP15, CTWY17]. While this approach is preferred when real-time tracking is necessary, a challenge is to propose a foreground hypothesis without specific knowledge on the actual object to track. Proposals are commonly formed from pre-learned models of "object-like" regions using static cues such as convex regions, pixel group similarities, or classification based on learning [HPL17, RT16, KMV$^+$16] as well as motion cues [TYB16, CTWY17], given that rigid foreground objects should move with coherent motion different from the background. Unsupervised segmentation for object scanning is difficult because pre-training is unlikely to apply to the large variety of arbitrary objects to be scanned, and motion cues from human hand movement can mislead the algorithm.

Semi-supervised and human-in-the-loop methods require varying amount of human interaction to specify the object of interest for higher accuracy [DAV]. In semi-supervised methods, the human provides an initial ground truth, typically for the first frame of a video sequence. This ground truth could be a segment

FIGURE 4.2: Skin regions identified by skin-pixel based segmentation described in [VSA03]. This method cannot detect shaded areas of hand as well as accessories on hand (jewelry, nail polish, etc.). It also falsely includes object regions with colors close to human hands.

mask [CMPT+17, CTWY17] or user scribbles annotating foreground and background [RKB04, BK04]. The algorithm then propagates this annotation segment to the unlabeled frames. Some semi-supervised methods use the first frame for fine-tuning a learned object model [CMPT+17, PKB+17, VL17] and then perform per-frame segmentation. However, most semi-supervised methods use a propagation scheme to propagate the initial labels to successive frames [TYB16, PS14, CTWY17].

4.3 In-hand Object Reconstruction Pipeline

Our pipeline takes an RGB-D video as input, in which a user rotates the object in front of the camera. The user can perform multiple regrasps to reveal all sides of the object. Our method then uses a novel semi-supervised video segmentation technique, called BackFlow, to segment the object from the hands and background. These object segments are then registered to a global reference frame, and the overall model is post-processed to reduce noise.

4.3.1 BackFlow Video Segmentation Method

BackFlow is designed to handle many unique challenges for hand-object segmentation in RGB-D videos. For example, the **new sides problem** occurs because objects

FIGURE 4.3: The new sides problem poses a challenge for video segmentation algorithms that rely on coherent object appearance. Tracking a can of tomato soup with a hierarchical graph-based method [GKHE10] fails to grow the segmentation to newly appeared top of the can.

being manipulated quickly undergo significant shape and appearance changes. These changes are difficult to predict without prior knowledge of the object. Conventional video segmentation methods have difficulty maintaining tracking (Fig. 4.3). **Object-hand appearance similarity** is also quite common. Although the appearance of human hands varies from person to person, various skin-tone colors are commonly represented on the packaging. Additionally, the hand could also be wearing nail polishes or jewelry, which further complicates the identification of hand from the object. **Occlusion and large deformation** are especially prominent with human hands. Hands can occlude large portions of the object, and deform quickly and significantly. **Finger-object motion similarity** also tends to confuse algorithms that rely on distinct motion cues.

BackFlow derives new object appearance hypotheses by deduction, rather than through foreground coherence. It preferentially tracks background pixels and derives foreground accordingly through a graph-based framework. Essentially, it assumes that what is not background is likely foreground, and it attempts to track and detect the disappearance and emergence of the background. The rationale behind this approach is that hands and background objects exhibit stronger spatiotemporal coherence compared with the target object, and therefore are the better candidate to be tracked.

BackFlow requires no prior training and forms initial object proposals based on color GMMs learned in the first segment. The proposals are adjusted during run-time with more available frames. This method is most effective when the appearance and shape of the foreground object evolve quickly while the background remains relatively stable.

The four main steps of BackFlow are described below:

Prepossessing and initialization

BackFlow first eliminates unnecessary background tracking. A depth cut-off on the color images is performed that eliminates areas more than 1 m away from the camera. Then it requires a human to initialize *sure background* pixels, either by providing background scribbles or a segment mask. These annotations serve as hard constraints in graph-cutting for the first frame.

Background pixel propagation

Next, we use optical flow [Far03] to propagate raw background pixels between two successive frames. The flow is cross-examined by performing backward flow, and background pixels with the inconsistent flow are dropped. Since most optical flow inconsistencies occur near the motion edge, this helps prevent background labels from drifting into the foreground.

Superpixels are employed to guide the background propagation and to increase robustness against drift. Specifically, we perform SLIC segmentation [ASS+12] with N = 20,000 superpixels on frames with 640*480 resolution. When propagating background pixels from frame t to frame t+1, all frame t superpixels that contain raw background pixels are marked as the background superpixels. We track the flow of all pixels within the background superpixels to the second frame and label frame t+1 superpixels that receive enough propagated background pixels as new background su-

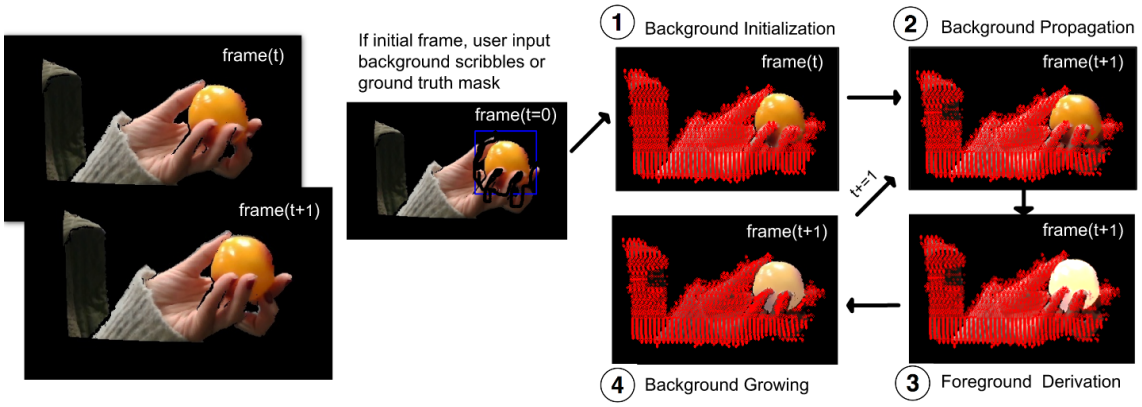


FIGURE 4.4: BackFlow takes user annotation of background or ground truth segment on the first frame and propagates labels as follows. 1) Initialize background labels (superpixels) from user input. 2) Propagate background labels from frame t to frame t+1 through optical flow. 3) Perform graph-based foreground derivation. 4) Remove background labels in segmented foreground and grow background labels to neighbourhood superpixels.

perpixels. The use of superpixels provides additional color information and prevents drifts into neighboring groupings that differ in color. This measure also replenishes background labels within a superpixel under the assumption that all pixels within a superpixel likely belong to the same label.

Segmentation with relaxed foreground derivation

Step 3 performs graph-based segmentation after the background pixels have been propagated to the current frame. Pixels form vertices connected to neighbors and both background and foreground labels by weighted edges. The segmentation is formulated as an energy minimization problem in which the energy function has the form:

$$E(L) = \sum_{p \in \mathcal{P}} D_p(L_p) + \gamma \sum_{(p,q) \in \mathcal{N}} V_{(p,q)}(L_p, L_q), \tag{4.1}$$

where $L = \{L_p | p \in \mathcal{P}\}$ is a labeling of image $\mathcal{P}$, $D_p(\cdot)$ is the data penalty function assigning labels to the vertex, $V_{(p,q)}$ is the smoothness term, and $\mathcal{N}$ is a set of all pairs of neighboring pixels [BK04]. γ is a parameter chosen to balance data and

smoothness terms. The minimum-energy cut of the graph is therefore a partition of the vertices into two disjoint sets, solved by min-cut/max-flow algorithms [BK04].

We use Gaussian Mixture Models (GMMs) to describe background and foreground distributions similar to [RKB04]. We modify standard graph segmentation algorithms to encourage foreground propagation and to rely more heavily on motion cues in the presence of drastic appearance changes.

Background GMMs with K=8 components are learned on the first frame and used throughout the video sequence, and foreground GMMs are learned on a sample of raw pixels in the previous foreground segments. Specifically, we store past foreground pixels, and after each new frame is segmented, we downsample the foreground set by 90% and add 10% new foreground pixels; this is to ensure a stable background proposal and a smoothly transitioning foreground proposal as the object evolves.

In BackFlow, we formulate initial data penalty terms similar to Grabcut [RKB04], which indicate the log likelihood of a pixel belonging to the foreground and background GMMs, respectively:

$$\begin{aligned}\tilde{D}(\alpha_p, k_p, \underline{\theta}, z_p) = &- \log \pi(\alpha_p, k_p) + \frac{1}{2} \log \det \sum (\alpha_p, k_p) \\ &+ \frac{1}{2}[z_p - \mu(\alpha_p, k_p)]^\intercal \sum (\alpha_p, k_p)^{-1} [z_p - \mu(\alpha_p, k_p)],\end{aligned} \tag{4.2}$$

where α_p indicates background or foreground assignments and has value of 0 or 1 in hard segmentation. $\underline{\theta}$ stands for the GMM parameters. k_p is the GMM component assigned to the pixel and $k_p \in \{1, ..., K\}$ where K is the number of components in the GMMs. z_p the a pixel color in RGB colour space. The weights π, mean μ and covariance Σ are the parameters for the Gaussians from the foreground and background distributions.

The data cost for each node is by default $D \leftarrow \tilde{D}$, except for the following exceptions. For nodes already associated with background labels, we assign D to

a high constant penalty to be connected to the foreground and zero penalty to be connected to the background. For unlabeled nodes, we further add a comparison step in which we equalize data penalties assigned to both labels if a pixel only has a slightly lower probability belonging to the foreground than the background proposal. The background-foreground assignment of such node is therefore determined by the assignments of neighborhood pixels that have a high interacting potential to the node. Specifically, if

$$\tilde{D}(\alpha_{p=1}, k_p, \underline{\theta}, z_p) > \tilde{D}(\alpha_{p=0}, k_p, \underline{\theta}, z_p), \tag{4.3}$$

and

$$\tilde{D}(\alpha_{p=1}, k_p, \underline{\theta}, z_p) - \tilde{D}(\alpha_{p=0}, k_p, \underline{\theta}, z_p) < N, \tag{4.4}$$

then

$$D(\alpha_{p=1}, k_p, \underline{\theta}, z_p) \leftarrow \tilde{D}(\alpha_{p=0}, k_p, \underline{\theta}.z_p), \tag{4.5}$$

where N is the relaxation constant. Experiments find that the segmentation accuracy is not greatly impacted by the value of N for $8 < N < 16$.

The introduction of the relaxation constant encourages the inclusion of the newly appeared foreground that is adjacent to the existing foreground. While this measure would otherwise introduce false positives if the background is not densely labeled, the majority of the background area is already correctly labeled. We also observed that the difference in data penalties for newly appeared background (e.g., hands, arms) are usually much greater than the relaxation constant.

A commonly used smoothness term (e.g., in Grabcut) is:

$$V(\underline{\alpha}, z) = \sum_{(m,n)\in\mathbf{C}} I[\alpha_n \neq \alpha_m] \exp(-\beta||Z_m - Z_n||^2)$$

z is the given image data, $\underline{\alpha}$ is the background or foreground assignments for each pixel, $\mathbf{C}$ is the set of all neighbouring pixels, $||Z_m - Z_n||^2$ is the Euclidean distance

between neighbouring pixels in RGB color space. β measures the inverse expectation of contrast within the image.

In BackFlow we introduce motion cues into the smoothness term. To encourage consistent labeling for areas with coherent motions, we use optical flow along with color dissimilarities for smoothness term calculations. We convert the flow vectors into images in HSV color space where the Hue and Value are computed from the angle and magnitude of the flow. The smoothness penalties are then taken as the smaller of the discontinuity in color and flow:

$$V(\underline{\alpha}, z) = \sum_{(m,n)\in\mathbf{C}} I[\alpha_n \neq \alpha_m] \min(\exp(-\beta_z||Z_m - z_n||^2), \exp(-\beta_f||F_m - F_n||^2)).$$

The expectation value β_z and β_f are calculated respectively for RGB image and the flow image. In our implementation we use $\gamma = 50$ and $N = 10$. The cut is then performed using Min-Cut/Max-Flow algorithms. The largest connected component in the result becomes the foreground segment.

Removing and growing background labels

Finally, we extend background tracking into the newly appeared background. After segmentation of each frame, all superpixels that are spatially adjacent to the existing background and are not segmented as foreground become background superpixels and will be tracked in the next frame as shown in Fig. 4.5.

4.3.2 3D Reconstruction

The segmented frames are registered to produce the final model. To avoid redundant information and bad frames with severe motion blur/noise, keyframes are selected for registration. Pairwise registration between keyframes is performed using sparse

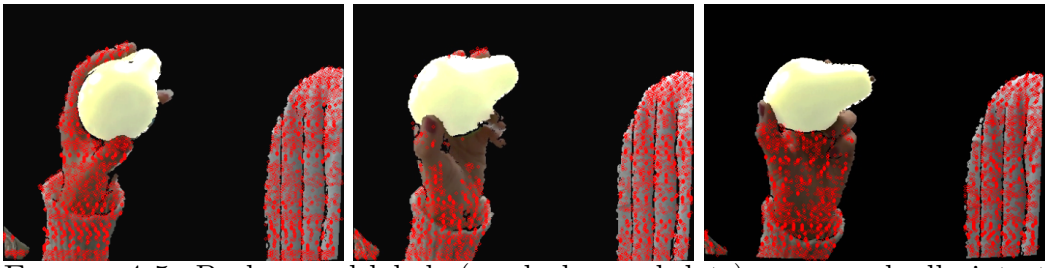


FIGURE 4.5: Background labels (marked as red dots) grow gradually into the previously unseen palm while maintaining separation from the foreground object.

feature-point matching followed by dense Iterative Closest Point (ICP) [BM92] alignment. Finally, we run pose-graph alignment and postprocessing.

By default, we select every 10th frame as the keyframe if it either 1) matches enough feature points from the previous keyframe or 2) its point cloud can be matched above a certain percentage with the previous keyframe with ICP. Otherwise, we try to match the previous keyframe with the 9th frame, 8th frame ... until a match is found. If no match exists, we will select the 10th frame and continue with the process.

For pairwise registration we extract SIFT (Scale-Invariant Feature Transform) [Low04a] features from consecutive keyframes and match feature pairs through a 2D RANSAC (Random sample consensus) homography process, and use the depth correspondence of the matched feature point to find the best 3D rigid body transformation between the two frames. If insufficient matching features are found, dense ICP is performed between two point clouds to estimate the transformation. We use the colored point cloud registration by Park et al. [PZK17] implemented in Open3D [ZPK18]. This ICP variant jointly optimizes both point-to-plane distances and minimizes the error of projected color. Pose graph optimization is used to obtain the final registration transforms in the global frame, with methods described in [KGS+11].

For postprocessing, the aligned colored point clouds are stored in voxel grids of dimension 2mm, and each voxel is optimized to reject color outliers, and grids with

less than 2 points are not used. Registered point clouds then go through Poisson surface reconstruction [KH13] to create watertight surfaces, and isolated components whose diameter is smaller than 10% of the mesh diameter are removed to achieve the final models.

4.4 Experiments

4.4.1 RGB-D in-hand object manipulation dataset

Although there are several popular video segmentation datasets, there is no existing in-hand object scanning RGB-D video dataset with dense pixel level annotations. To systematically evaluate our methods and to aid the research community, we introduce the RGB-D In-hand Object Manipulation Dataset (Fig. 4.6). This dataset contains 13 sequences of in-hand manipulation of objects from the YCB object set [CWS+15], recorded with an Intel RealSense SR300 RGB-D camera. Each sequence ranges from 300 to 800 frames in length (filmed at 30 fps) and contains in-hand manipulation of the objects revealing all sides. The dataset is complete with color images, depth cut-off images, and depth mages. The pixel-wise annotation aligned to the depth cut off image is provided every 10 frames. The dataset and results can be downloaded at the RGB-D In-hand Object Manipulation website at https://www.rgbdinhandmanipulation.com.

The performance of BackFlow is compared to other state-of-the-art segmentation methods in Tab. 4.1. OSVOS is a semi-supervised fully-convolutional neural network (FCNN) that was one of the top-performers on DAVIS 2017 [CMPT+17]. SFL indicates the unsupervised FCNN technique SegFlow [CTWY17]. Both OSVOS and SFL were pre-trained on the RGB-D Object Dataset [LBRF11] while OSVOS was additionally fine-tuned on the initial segment of each sequence. HVS is the hierarchical graph-based video segmentation of [GKHE10]. SPBS indicates skin-pixel based segmentation, which is a GMM method described in [VSA03], trained on the UCI

FIGURE 4.6: RGB-D in-hand object manipulation dataset contains RGB-D video of 13 non-duplicate items from YCB food category, including Pringles, mustard, Cheez-It, sugar, Spam, tomato soup, banana, pear, plum, strawberry, orange, lemon and Jell-O.

Table 4.1: Segmentation Results

	OSVOS	SFL	HVS	SPBS	BackFlow
IoU	87.97%	29.61%	52.05%	31.52%	**93.52%**
FP	4.17%	50.59%	**1.7%**	49.99%	2.12%
FN	7.85%	13.90%	46.29%	18.48%	**4.36%**

skin segmentation dataset [Bha18].

These experiments separate each object handling sequence into sub-sequences of 100 images. A ground truth annotation is given on the first frame of each sub-sequence, and the segmentation accuracy is tested on the remaining 99 frames. We measure the intersection over union (IoU), areas of false positives over union (FP) and areas of false negatives over union (FN). BackFlow outperforms other state-of-the-art approaches by almost 6% percent in overall accuracy and achieves lower false positive and false negative rate. Per-sequence results are shown in Fig. 4.7.

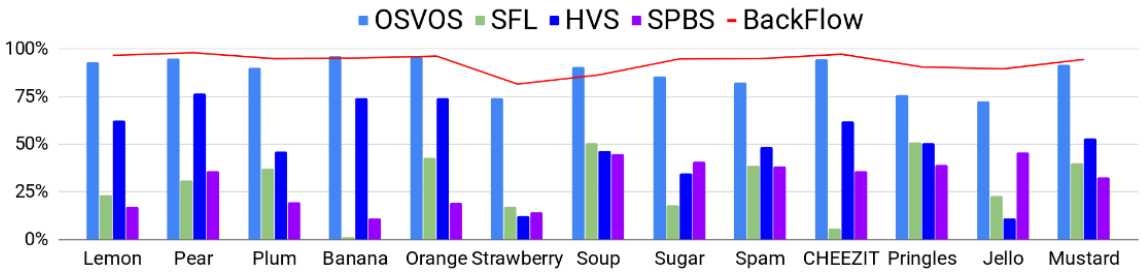


FIGURE 4.7: Per sequence accuracy (IoU) comparison of BackFlow to other state-of-the-art video segmentation methods.

Admittedly, BackFlow is tuned for in-hand object tracking, while many state-of-the-art fully-convolutional neural network architecture we compared to are pre-trained for more general segmentation scenarios such as tracking animals, humans and large objects. In those scenarios, they will perform better than BackFlow. For example, on the DAVIS 2017 dataset [PPTM$^+$16], BackFlow achieved an overall IoU accuracy of 43.5% compared to 86.1% from the best performer OnAVOS. Given other training datasets, it is possible that the algorithms we compared to could be trained to perform better on our test sequences. However, given the randomness of the object that may appear in a hand-object interaction task, we argue that no specific dataset could prepare the best object proposal for any arbitrary object to be potentially modeled. It was also observed that even the FCN models that were fine-tuned on the first frame struggled to track the newly appeared sides accurately, indicating the fundamental limitations of using classification based tracking methods for arbitrary objects with new sides problems.

The reconstructed models of several objects from the dataset are shown in Figure 4.8. The textured (color or geometry) objects are reconstructed complete with all sides, except when the surface material (e.g., black, highly reflective) cannot be captured by a depth camera. Our reconstruction pipeline currently cannot obtain a complete model with symmetrical and texture-less objects, such as pear and orange, and the registered point clouds appear as incomplete "shell".

FIGURE 4.8: Reconstructed models from BackFlow segmentation results.

Fig. 4.9 shows some example of unprocessed point cloud reconstruction from BackFlow against the next-best performer OSVOS. To qualitatively measure the quality of the reconstructed models, we further compare all 13 registered point clouds constructed from BackFlow and OSVOS to the ground truth Poisson meshes provided in the YCB dataset. The evaluation metric used is the one-sided Hausdorff Distance implemented in MeshLab. To measure it, the registered point cloud and the YCB mesh are aligned using manual key points matching (we use the same rigid transformation for both registered point clouds), then the point cloud is sampled to 20,000 points and for each sample, the Hausdorff Distance to the closest point over the YCB mesh is computed.

Shown in Tab. 4.2, the Hausdorff distance measurements for BackFlow recon-

FIGURE 4.9: Comparison of reconstructions from OSVOS and BackFlow.

structed models are around 50% smaller than OSVOS in all categories including max distance, mean distance, and RMS. The max distance in BackFLow is caused by failure to segment out a human finger in parts of the banana sequence, the artifact is reflected on the reconstruct (see Fig. 4.8), yet the max distance is still much smaller than that of OSVOS. While BackFlow only improves segmentation performance by 6%, the resulting reconstructions are of significantly better quality, as these segments maintain a better consistency and have a lower false positive rate.

Table 4.2: Hausdorff Distance measurements w.r.t. bounding box size of the reconstructed models

	Max distance	Mean distance	RMS
BackFlow	0.195	0.014	0.017
OSVOS	0.427	0.024	0.031

4.4.2 Novice scanning of many items

To evaluate the ease of use and robustness of the pipeline when handled by a novice user, we presented our software to a volunteer who is unfamiliar with 3D object modeling. 200 grocery items were purchased satisfying the following criteria: 1) the size of the object should be easily manipulated with a single hand, 2) the object does not contain large black or transparent areas, which cannot be captured with a structured light RGB-D camera, 3) the object is textured or non-symmetric, 4) objects should not significantly deform during manipulation.

The volunteer was taught how to use the software and was told to reveal all sides of the object. No other instruction on how to manipulate the objects was given. We record the total amount of time needed, including the instruction time, and several small breaks, for the volunteer to finish scanning the objects. The volunteer then annotates every 100 frames of the scan sequence as initial labels.

On average, the volunteer took 102 seconds to scan each item, plus another 36 seconds for labeling (averaging 4.9 frames per item). Segmentation and reconstructions then automatically run to generate object reconstructions. The total labor time used for reconstructing 200 models is 7 hours and 40 minutes, or 1 fps for segmentation and reconstruction on a CPU computer.

As shown in Fig. 4.1, the pipeline is robust enough to generate many visually high-quality reconstructions with no hands or any other background objects found in any reconstructed model. There are also failed reconstructions, which are most frequently caused by misaligned frames during registration. Better registration techniques are

likely to improve the results further.

5

Systems and Analysis for Robust Robotic Packing

While the offline packing algorithm is guaranteed to be feasible under perfect modeling and precise execution, making packing reliable enough on an actual physical system remains a challenge due to numerous sources of system uncertainties. This chapter studies how errors in vision, grasping, and modeling give causes to cascading errors in the overall robotic packing pipeline. A systematic evaluation on a physical packing testbed studies the sources of error and models their magnitude. Experiments are conducted to quantify the impact of such errors on the overall packing success rate in Monte Carlo simulation and on the physical testbed. It also presents strategies for improving the robustness of robotic packing: 1) using conservative planning to ensure feasibility under uncertain ranges of model parameters, and 2) employing closed-loop vision and manipulation so that a robot can dynamically react to errors and correct for them. Empirical results demonstrate that the closed-loop packing and robust planning increase the overall packing success rate by 15%, from 83% to 98%, when compared to the baseline system. [1]

[1] This chapter is reproduced from Fan Wang and Kris Hauser, "Systems and Analysis for Robust Robotic Packing," in submission to IEEE Transactions on Robotics (T-RO). This work is supported

5.1 Introduction

There exists various uncertainties in a physical packing system, such as incorrectly estimated object shapes, failed grasps, object reorientation during grasping, as well as calibration error. Small disturbances can cause serious, cascading failures, particularly if the packing arrangement is dense: if an object unexpectedly shifts during execution, it will likely leave insufficient space for remaining items in the selected container. Failures can also cause damages to the items, since an industrial robot is stiffly geared and blind execution of pre-planned paths can exert very high force when an unexpected obstacle is encountered. Such failures are also difficult and time-consuming to recover from, e.g., remove all placed items in the old box and replace them in a larger box.

In this chapter we analyse various sources of uncertainty inherent to automated packing systems and develop strategies to mitigate the impact of uncertainties and increase the robustness of the packing execution. This analysis is carried on a packing system we build that uses state-of-the-art perception and planning components. The system picks from a set of arbitrarily shaped objects on a flat picking area and packs them in a shipping box. This system executes offline planning from our previous work that produces robot-packable object placements

Mitigating errors in dense packing have been explored by a few previous works. For example, Ye et al. [YR18] develop a real-time state estimation system that can recover the pose and contact formation of a rectilinear object in grasp relative to its environment. By fusing visual and force sensing, they demonstrate successful inserting an object picked by a suction cup into a tight space. Shome et al. [STS+19] designed a complete robot packing system using minimalistic hardware stack of a single robot with suction gripper equipped with RGB-D cameras, similar to ours.

by an Amazon Research Award.

They have also employed different techniques to achieve robust packing that minimizes failure conditions, One example of such technique is using the suction gripper as a push finger that executes three pre-defined key manipulation primitives to tight packing in against perception and positioning errors.

Both methods introduces collision to the object during manipulation to force tight placement, which could cause damage to fragile object. Instead, our method against uncertainties focus on planning. We propose two strategies for improving the reliability of packing under uncertainties. The first strategy is to plan conservatively during the offline planning phase to ensure execution feasibility under positioning error. The second strategy is to employ closed-loop vision and manipulation so that a robot can dynamically react to errors and correct them. Specifically, the system re-senses the object pile after each placement using visual and depth information and then replans the best subsequent placements if the pile deviates from the plan significantly.

We perform exhaustive testing in both Monte Carlo simulation as well as on a physical robot, under different packing strategies. In Monte Carlo simulation, we test the packing success rate while systematically introduce positioning errors of arbitrary magnitudes on thousands of object sets. We have also tested those strategies on the physical platform (Fig. 5.1). Empirically, the execution success rate on the physical platform with an open-loop baseline system is quite low, at 83%, due to multiple sources of uncertainty causing cascading errors. The proposed strategies for robust robot packing raise the packing success rate to 98%.

5.2 System setup and error analysis

In this section, we first introduce our hardware and software setup is fairly representative of current state-of-the-art sensing, perception, and planning for warehouse manipulation. Then we identify the sources of error in such a setup that may lead

to failed packing, and propose two strategies for mitigating these errors.

5.2.1 Experimental setup

A 6R industrial robot (Universal Robot UR5e) is equipped with a suction gripper (Robotiq EPick) and a compliant suction cup (Fig. 5.1). One RGB-D camera (Realsense SR300) and two high-resolution stereo / structured light black-and-white depth cameras (Ensenso N35) are used. An RGB-D / stereo pair is mounted overhead looking down over the picking area, and the other stereo camera is mounted looking down the packing area, shown in Fig. 5.2. This setup allows us to combine both the color information from the RGB-D camera and the high-quality point clouds from the stereo cameras. The high-quality point cloud enables the pose recognition pipeline to achieve better accuracy compared to the RGB-D data.

The entire system layout is illustrated in Fig. 5.2. In the picking area, objects are placed in arbitrary locations. The system assumes that objects are drawn from a known set, whose 3D models are available. In addition, all objects in the picking area will be packed, with their initial locations not touching. In the packing area, a predetermined box footprint area is marked off, and a box with matching dimensions is placed by hand to align with the footprint.

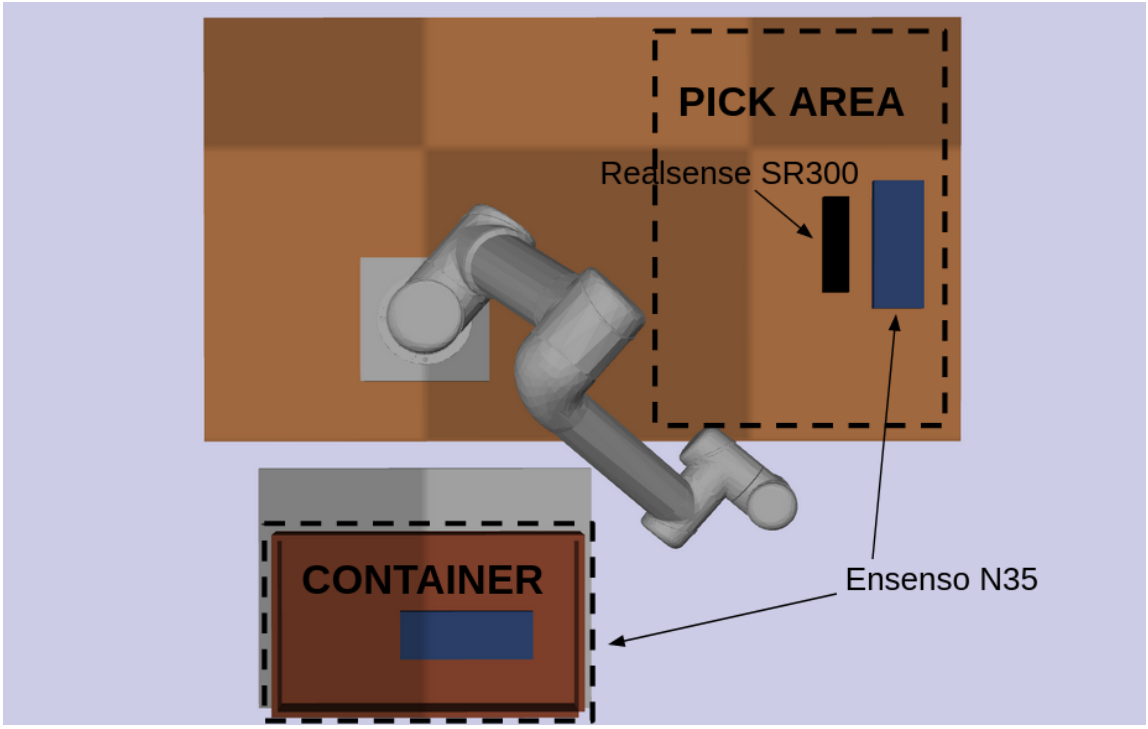


FIGURE 5.2: Top-down diagram of the packing experiment. Objects to be packed are arranged in the picking area, and must be packed in the container (e.g., a shipping box). Overhead cameras (Realsense SR300 and Ensenso N35) provide color and depth information.

The software pipeline consists of calibration, perception, and planning. The calibration steps include:

- Factory-specified tool center point and camera intrinsic parameters were used.
- Camera extrinsic calibration using robot-mounted checkerboard
- The box footprint location is calibrated by jogging the robot to touch the footprint corner locations. The footprint corners are set to the tool center point coordinates relative to the robot base.

Object identification and pose estimation are then performed with the following steps:

1. Plane segmentation removes the tabletop from the point cloud.

2. Region growing segmentation [AB94] is used to segment object point clouds.

3. Pointcloud segments are projected onto the RGB image to obtain an object ROI.

4. The object is identified using a convolutional neural network model [HZRS15b], trained on the experiment objects.

5. A CNN pose estimation model gives an initial estimate of object pose [WXZ+19].

6. If fitness [ZPK18] of the CNN estimation is below a threshold, point-cloud only template matching is performed.

7. The pose is fine-tuned with iterative closest points (ICP) [RL01b].

The packing algorithm [WH19b] chooses an object packing sequence and generates a grasp location and packing pose for each object. Top-down loading is performed. The planner checks several constraints: 1) no excessive force on the gripper caused by misalignment, 2) the loading trajectory is collision-free with the container position and objects already placed, and 3) stability of the object under gravity in the stack of existing items.

We suppose two forms of sensor feedback during/after packing execution with the robot arm:

- Force feedback, via the force/torque sensor on the UR5e's wrist, is used for guarded moves during packing. This prevents crushing items.

- RGB-D and stereo cameras provide an overhead image of how items have been packed in the container.

5.2.2 Factors Affecting Packing Success

Packing success is defined by a final state having all objects unbroken and packed within the container's dimensions.

Because the planner performs collision and stability checking, if the world behaved exactly as modeled, then the executed plan would be guaranteed to yield a successful packing. However, several errors are introduced by the execution pipeline.

We identify and describe nine key **error sources**:

1. Camera calibration error ϵ_{cal}.
2. Object identification error ϵ_{ID}.
3. Pose recognition error ϵ_{pose}.
4. 3D geometry modeling errors ϵ_{geom}.
5. Center of mass specification errors ϵ_{cm}.
6. Grasp positioning error ϵ_{grasp}.
7. Grasp acquisition (lifting) error ϵ_{lift}.
8. In-transit manipulation error ϵ_{manip}.
9. Box placement error ϵ_{box}.

Camera calibration error in the picking area causes deviations in the robot-relative object pose estimates, adding to problems acquiring a grasp. Camera calibration over the container leads to collisions with objects and walls of the container in the closed-loop packing strategy.

Object identification error leads the planner to choose a different 3D reference model than the actual object's geometry. In packing setups where the correct subset

of items must be picked from a larger collection, this could lead to more serious errors where the wrong item is packed. Our problem definition assumes that all presented objects must be packed, so object identification error is correctable during packing.

Pose recognition error leads to misalignment of the true object geometry and the 3D reference model used in grasp and packing planning. This causes grasp failures and inadvertent collisions during loading.

The 3D geometry of each object was acquired using a 3D scanner and multiple viewpoints. There is inevitable noise in these models, particularly noticeable in the underside of the object. Moreover, the scanned items were newly bought objects, and as items exhibited wear and tear during our experiments, some items underwent slight deformations. Although we do not have ground truth, we expect overall geometry errors to be less than 5 mm.

Centers of mass of objects are not measured precisely, and may also change with the object orientation (e.g., product settling in a box). This causes potential problems with grasp acquisition and inaccurate stability tests in the planner.

Grasp positioning errors occur when the suction gripper shifts the object during grasping. The related grasp acquisition errors occur when the gripper cannot achieve adequate suction to lift the object. In-transit manipulation errors occur when the robot prematurely loses grip of the object, e.g., by causing large accelerations.

Finally, box placement errors can cause collisions with the sidewalls during packing.

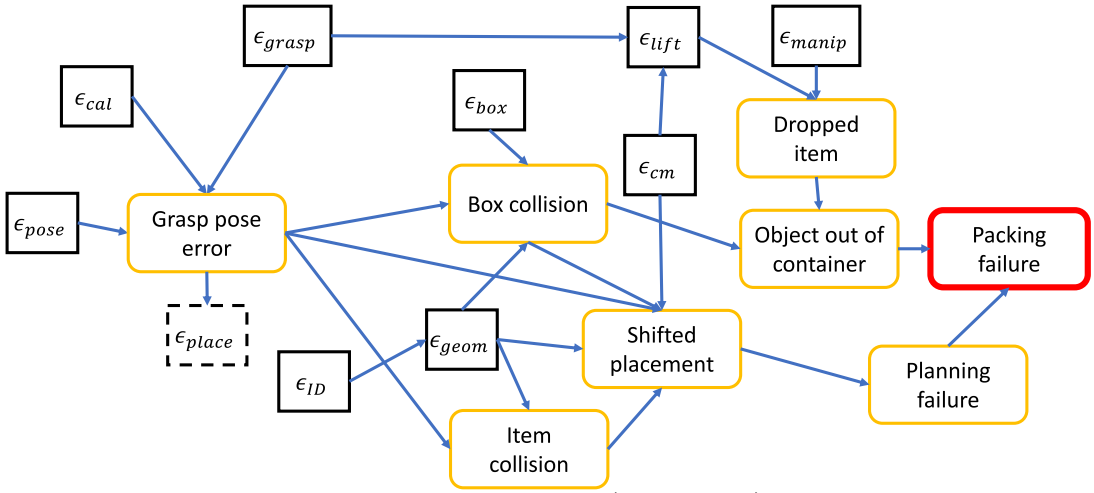


FIGURE 5.3: A diagram of how error sources (rectangles) contribute to unexpected events (rounded rectangles) and ultimately to packing failure. We use ϵ_{place} as a measure for grasp pose error.

Overall, these errors form cascading effects down the pipeline to contribute to unexpected events, such as toppling object placements or unexpected collisions, which lead to packing failures, as shown in Fig. 5.3. Informally, we refer to events in which an object hits a container wall or other objects prematurely as "smashing," and events in which an object falls over or shifts as "toppling." Note that the packing loop is performed N times, where N is the number of objects, so the system has an opportunity to measure and correct for errors accumulated during packing.

5.3 Error reduction methods

We propose two methods to reduce errors. The first strategy is closed-loop packing, which makes use of sensing feedback to adjust the packing plan so that if an object is grasped incorrectly or falls over during placement, subsequent steps can adapt to the new knowledge. The second strategy is robust planning, which predicts the future effects of uncertainty so that unexpected behaviors are less likely to occur. To evaluate the effectiveness, we perform experiments with four variations of planning and executing strategies.

1. **V1- Baseline:** Plans are generated and executed, assuming perfect models.
2. **V2- Robust planning:** The packing planner is modified to avoid tight fits.
3. **V3- Closed-loop vision:** The container sensor re-senses the object pile after each placement. If the heightmap of the pile deviates from the plan, then a replan is triggered.
4. **V4- Robust planning and closed-loop vision:** Both V2 and V3 strategies are employed.

Strategies V3 and V4 do not consider significant rearrangement of objects within the container (e.g., repacking), although we do allow the robot to push objects that have stuck outside the container. We describe these strategies in more detail below.

We implement a grasping controller based on end effector force feedback, which is used in all experiments. During grasping, if the robot's force sensor detected that the object was not successfully lifted, the robot will attempt the next computed grasp pose that can achieve the planned object placement with a collision-free loading motion. Moreover, while packing an object into the container, if a contact force exceeding 20N is sensed, the object will be released. This avoids crushing objects during "smash" events.

5.3.1 Open-loop packing baseline

This section provides details on the packing planner and vision system, which are used as-is in the baseline packing condition.

Packing planner

The packing planner algorithm from [WH19b] can be summarized as follows. The input is a set of N objects with reference geometries $\mathcal{G}_1, \ldots, \mathcal{G}_\mathcal{N}$, where $\mathcal{G}_i \subset \mathcal{R}^3$, and initial poses $\mathcal{P}^0 = (P_1^0, \ldots, P_N^0)$, where $P_i^0 \in SE(3)$. Also, $\mathcal{C}$ is given as the

free space volume of the container. The problem is to find a placement sequence $S = (s_1, \ldots, s_N)$ of $\{1, \ldots, N\}$, grasp poses $G_1, ..., G_N$, and placement poses $\mathcal{P} = (P_1, \ldots, P_N)$, $P_i \in SE(3)$, such that each placement satisfies a set of robot-packable constraints.

Let $P_i \cdot \mathcal{G}_i$ denote the space occupied by item i when the geometry is at pose P_i. The solution packing should satisfy the non-overlap constraints

$$(P_i \cdot \mathcal{G}_i) \cap (P_j \cdot \mathcal{G}_j) = \emptyset, \ \forall i, j \in \{1, \ldots, N\}, i \neq j, \tag{5.1}$$

containment constraints

$$P_i \cdot \mathcal{G}_i \subseteq \mathcal{C}, \ \forall i \in \{1, \ldots, N\}, \tag{5.2}$$

for each $k = 1, \ldots, N$, stability constraints

$$isStable\left(P_{s_k} \cdot \mathcal{G}_{s_k}, \mathcal{C}, P_{s_1} \cdot \mathcal{G}_{s_1}, \ldots, P_{s_{k-1}} \cdot \mathcal{G}_{s_{k-1}}\right) \tag{5.3}$$

and manipulation feasibility constraints:

$$isManipFeasible\left(P_{s_k} \cdot G_{s_k}, P_{s_1} \cdot \mathcal{G}_{s_1}, \ldots, P_{s_{k-1}} \cdot \mathcal{G}_{s_{k-1}}\right). \tag{5.4}$$

Non-overlap and containment constraints are calculated using standard collision detection methods. Stability is checked using a convex program, while manipulation feasibility requires 1) finding a feasible grasp pose and 2) ensuring that a top-down loading motion is collision-free both for the selected object as well as the robot.

For grasp planning, we use a point cloud-based planner that generates a set of vacuum grasp candidates, ensuring that the grasp location is nearly flat, the suction direction is nearly normal to the surface, and the center of mass is almost underneath the grasp point.

Specifically, the grasp point must lie within a radius $r = 2\,\text{cm}$ in the horizontal plane to the centroid of the observed point cloud segment. The areas under the

gripper should be planar areas (80% of the surface points directly below the tool are within 0.3mm to the estimated plane), which helps the vacuum opening to grasp normal to the surface.

For use in the planner, the set of grasp candidates is tested one by one for manipulation feasibility, i.e., with the chosen grasp pose, the robot can transfer $\mathcal{G}_i$ from P_i^0 to P_i without collision and with successful IK solutions being found along the trajectory. Moreover, in the packing pose, the resulting gripper axis needs to be within a tilting angle $\theta = \pi/4$ to the Z-axis to prevent excessive torques being applied at the gripper.

Pose recognition

The pose recognition pipeline estimates P_i^0 for $i \in \{1, \ldots, N\}$ for use in the planner.

First, we acquire partial pointcloud segments for $\mathcal{G}_1, \ldots, \mathcal{G}_\mathcal{N}$. This is done through a process of plane segmentation and distance calculation that removes all observation not on the picking station. The remaining pointcloud is segmented by Point Cloud Library(PCL)'s [RC11] implementation of region growing segmentation [AB94]. The algorithm partitions the points that are close enough in terms of a smoothness constraint.

Next, the partitioned pointcloud segments are projected on the color image obtained with a RealSense camera. Pixels inside the bounding box of the projects are cropped into a color image of each individual object. The ID of the object is classified with ResNet18 [HZRS15b], trained on the experiment dataset using Pytorch [PGM+19].

For the experiment, we assume that $\forall i \in \{1, \ldots, N\}, \mathcal{G}_i$ is sitting on a tabletop with pose P_i^0, where the orientation component of P_i^0 is a *planar stable* orientation for $\mathcal{G}_i$. Therefore, pose recognition estimates the translation of $\mathcal{G}_i$ while R_i^0 is from a finite subset modulo rotations about the vertical axis. We initialize the estimate

R_i^0 from the Dense Fusion [WXZ+19] algorithm. Inputs to Dense Fusion include a pointcloud, cropped color image, and ID for the corresponding object segment. If fitness [ZPK18] for this estimation is below a certain threshold, a template matching technique is used as a fallback. Template object point clouds are generated at planar-stable orientations, and the best-fit match to the point cloud observation is selected.

Whether Dense Fusion or template matching is used for the initial alignment, we follow this with a dense point-to-plane Iterative Closest Point (ICP) [RL01b] alignment for fine-tuning. This is performed using the implementation from Open3D [ZPK18].

5.3.2 Closed-loop packing

In closed-loop packing, we monitor and remodeled the object pile after each placement. A failure is detected if the pile shifted significantly from expected, preventing subsequent items from being executed as planned. In case of failure, we replan the packing locations of the remaining items while maintaining consistency in the previous plan where possible. We also detect whether an object is lying partially outside of the container, and perform a *push maneuver* to attempt to push the object back in.

Specifically, with the overhead depth camera, we capture a pre-placement depth map of the container. If the incoming item's 3D model at its planned packing location is in collision with the pre-placement heightmap, we count the number of colliding pixels. A "significant deviation" is triggered when this count exceeds a threshold of 20% of pixels belonging to the item.

If no significant deviation is determined, the item is placed to the planned transform as usual, which helps maintain coherence with the existing plan. If a significant deviation is detected, the strategy replans a new optimal placement for the infeasible object given the pre-placement heightmap. Rather than trying to identify the pose of each object in the pile, which is prone to significant errors, we simply treat the

heightmap as the geometry of an object rigidly-fixed to the container. The use of the sensed heightmap as a collision geometry suffices for collision checking between the robot and gripper, and stability checking still verifies whether there is sufficient friction between a rigid pile and subsequent placed objects. However, treating the pile as a rigid, unmovable object incurs some loss of information for the stability constraint checks, so the planner may decide to put the object in a location an underlying object may shift or topple.

To perform the push maneuver, we capture a post-placement heightmap, and detect whether any object's horizontal extent surpasses the dimensions of the container. If so, we aim the robot's gripper to push in a straight line perpendicularly to the box wall, with the end effector's lowest portion slightly above the box's maximum height. This movement is aimed at the geometry center of the portion of the object outside the container dimensions.

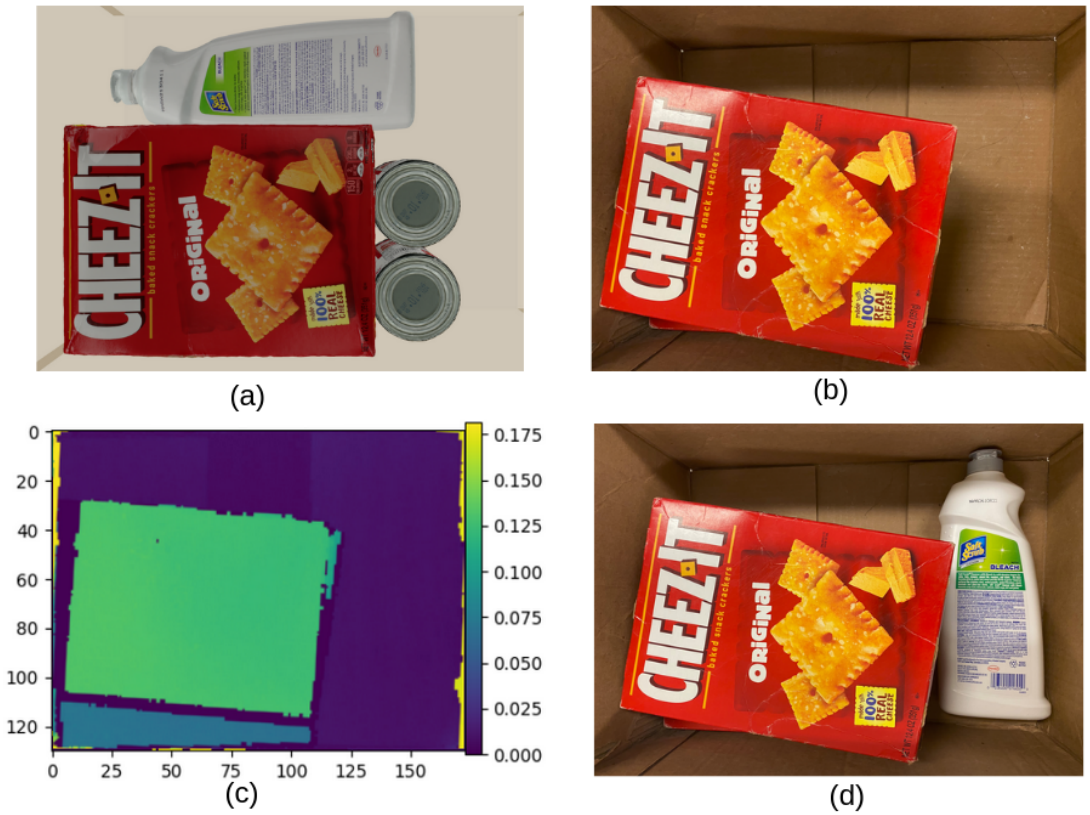


FIGURE 5.4: Closing the loop around vision leads to more robust packing. (a) Offline plan. The 2 *cracker boxes* are stacked first, followed by *bleach* and 2 *soup cans*. (b) The second *cracker box* is shifted during packing, blocking the access for *bleach*. Open-loop packing would lead to a failure. (c) The closed-loop strategy captures a heightmap of the placed pile, detects a violation for the *bleach* placement, and plans a new location. (4) *Bleach* is placed successfully in the empty location.

5.3.3 Robust planning

Robust planning avoids tight fits with a threshold distance of δ. Placements closer than δ to the container wall and placed objects will be penalized in the offline planner.

In [WH19b], a candidate placement is scored as follows:

$$c \cdot (X + Y) + \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} H'[i, j] \tag{5.5}$$

Where X and Y are the translation of the object inside the container at a given orientation. H' is the sum of the container heightmap assuming the object is placed

at the candidate placement. The heightmap is a 2D image of width w and height h, and c is a small constant. For robust planning, we add an extra term to the scoring function to penalize tight fit:

$$c \cdot (X + Y) + \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} H'[i, j] + C/\max(d_{min}, \delta) \tag{5.6}$$

Where d_{min} is the minimum horizontal distance between the item and the pile or the container wall. This is computed from the heightmaps of the pile and the item being placed. To compute the minimum distance, we only consider pixels in the pile/container heightmap whose pixel height is larger than object placement height Z and x-y location within and near the object 2D projection onto the x-y plane. The minimum of all horizontal distances between object pixels and the pile/container pixels is taken as d_{min}. The value C is taken to be a large constant such that C/δ dominates the other two terms in the sum.

It should be noted that conservative planning may fail to find a plan when non-conservative planning would succeed. To address these cases, the planner tries to replan with a smaller δ that decreases linearly until it is 0. If the $\delta = 0$ case fails, we declare planner failure.

FIGURE 5.5: Conservative geometry margins reduce inadvertent collisions. (a) A non-conservative plan (left) causes a failure during the execution of the plan (right). Specifically, the shifted *cracker box* caused a failure and crash of the *sugar* item planned on top of it. (b) A conservative plan with δ =1 cm margin. During execution, all items are placed within the container without causing an inadvertent collision.

5.3.4 Closed-loop packing and robust planning

In our final experimental strategy, we combine both closed-loop repacking and robust planning. This is mostly a straightforward combination, except that we note that when δ is decreased due to a planning failure, we maintain the decreased value for subsequent planning steps.

5.4 Analysis and experimentation

A systematical evaluation is performed on the proposed packing system that studies the sources of error and model its magnitude. Experiments are conducted to quantify the impact of such errors on the overall packing success rate in Monte Carlo simulation with bullet simulator [CB19] and on the physical testbed.

5.4.1 Itemset

15 real-world items from the YCB video object dataset [XSNF17] are selected that are mostly rigid, opaque, graspable, visible to the depth camera, and in a wide range of shapes and weights (Fig. 5.6). High-quality object meshes for the 15 items are available so that we can simultaneously perform the evaluation on both physical and simulation testbeds.

FIGURE 5.6: 15 items from YCB video object dataset are used in the packing experiment

5.4.2 Uncertainty evaluation

We collected a dataset of all experiment items placed in various poses and positions on the picking area, complete with RGB images, depth images, and pointclouds, taken by Realsense and Ensenso cameras in their overhead mounting positions. Around 200 poses and positions are taken for each object. 70% of the dataset was used for training, and the remaining 30% of the dataset was withheld for testing.

Object identification

In our test set, object segmentation was 100% successful because the input objects were placed in isolation. Our object recognition pipeline achieved a 98.5% top-1 accuracy on the testing set, which indicates that for 7.2% of 5-item order sets, at least one of the objects will be misidentified. Note that because all objects in the packing area need to be packed, a misidentified object will not necessarily lead to failed packing. Instead, it contributes to a (potentially substantial) geometry modeling error.

6D pose estimation

To obtain the pose estimation error, the 6D pose estimation result from our pipeline is further corrected by manual alignment of the projected and ground truth (using the Meshlab software). We considered using an alternative approach to pre-define a ground truth pose and manually align the object to this pose, but we found that symmetric objects (e.g., boxes, cylinders) induced large pose error measurements, even though these errors would have virtually no effect on picking and packing. On our test set, the average translation and rotation error is 0.7 cm and 0.3 rad.

Rare errors

Grasp acquisition errors were not observed in our setup (so $\epsilon_{lift} \approx 0$), in large part because we chose objects from the YCB dataset that were more easily lifted by suction. In-transit manipulation errors never occurred during our experiments ($\epsilon_{manip} \approx 0$). Box placement error is also quite low ($\epsilon_{box} \approx 0$) compared to other sources of error, e.g., calibration, because it is relatively simple to align the box to within a millimeter or two of the calibrated footprint. Moreover, it would be a simple matter to add rigid fixtures to the packing area to constrain the box even further. Since these errors are so low, we do not include them in our analysis.

5.4.3 Summary statistic: placement error

Observe that calibration, pose estimation, and grasp positioning errors all contribute to the pose of the object-in-hand, which leads to an ultimate error in where the object is packed. Specifically, this is the final translation and rotation error of an object from a planned packing location P_i, assuming no collisions occurred during loading. We call this summary error *placement error* ϵ_{place}, and can be evaluated experimentally. The evaluation performs 80 single item manipulations by a known transform that transforms object from one planar orientation to another. Before the manipulation, the pointcloud of the object is captured and the pose of the segment is estimated. After manipulation, the pointcloud of the object is captured again, and the observation is manually aligned to the projected object pose. ϵ_{place} is taken to be the RMSE of these alignment errors. Our experiments show that placement error is on average 1.02 cm in translation and 0.41 rad in rotation.

A summary of all error sources is given in Tab. 5.1.

Table 5.1: Summary of empirically evaluated error sources

Error source	Magnitude
ϵ_{ID}	1.5%
ϵ_{pose}	0.7 cm (translation), 0.3 rad (orientation)
ϵ_{geom}	Estimate $\approx$ 0.5 mm
ϵ_{place}	1.02 cm (translation), 0.41 rad (orientation)
ϵ_{lift}	≈ 0
ϵ_{manip}	≈ 0
ϵ_{box}	≈ 0

5.4.4 Simulation testing

A more extensive set of experiments were conducted in simulation, where it is more practical to systematically introduce different magnitudes of errors and larger test sets. Varying degrees of error from ground truth object poses and depth sensor readings are used as input to each of our packing strategies. The simulations are run on several desktop computers with 32GB of RAM and Intel I7 processors.

For these tests, we randomly generated 1000 sets of 10-item orders from YCB [CSB+17] and the APC 2015 object set [Rut15]. For consistency, we use the same item sets across the range of packing strategies. For picking, we rigidly attach objects to the robot's gripper with a randomly-generated pose error, using the same standard deviations as ϵ_{place}, but scaled with an error scaling factor from 0-200%. The object is dropped when the robot reaches the desired placement pose, or a 20N force threshold is reached.

The packing success rates for strategies V1-V4 are plotted in Fig. 5.7. For V2 and V4, a δ value of 1 cm is used. At 0% error, all experimental variations performed similarly, just under 98% success rate, while closed-loop strategy performs slightly worse. This can be largely attributed to the treatment of the pile as a rigid, infinite-mass object during replanning, which fails to capture the nuances of the pile's stability.

When an error greater than 100% is introduced, the success rate for the baseline strategy drops off sharply as the error increases. Meanwhile, the rate of drop off is noticeably slower for the other 3 comparison strategies that employ robustness measures. The best performer is V4 that uses both robust plannings as well as dynamic replanning, achieving the highest or near highest success rate at all positioning errors introduced, and maintain a packing success rate at 94.5% at 2.04 cm translation error and 0.82 rad in rotation error.

It should be noted that increasingδ does not always lead to better performance under uncertainties. A δ of 2 cm was also tested in V2 and V4, achieving similar and slightly worse performance than a δ of 2 cm. With a higher margin, more planning under margin will fail and fall back to use a smaller margin.

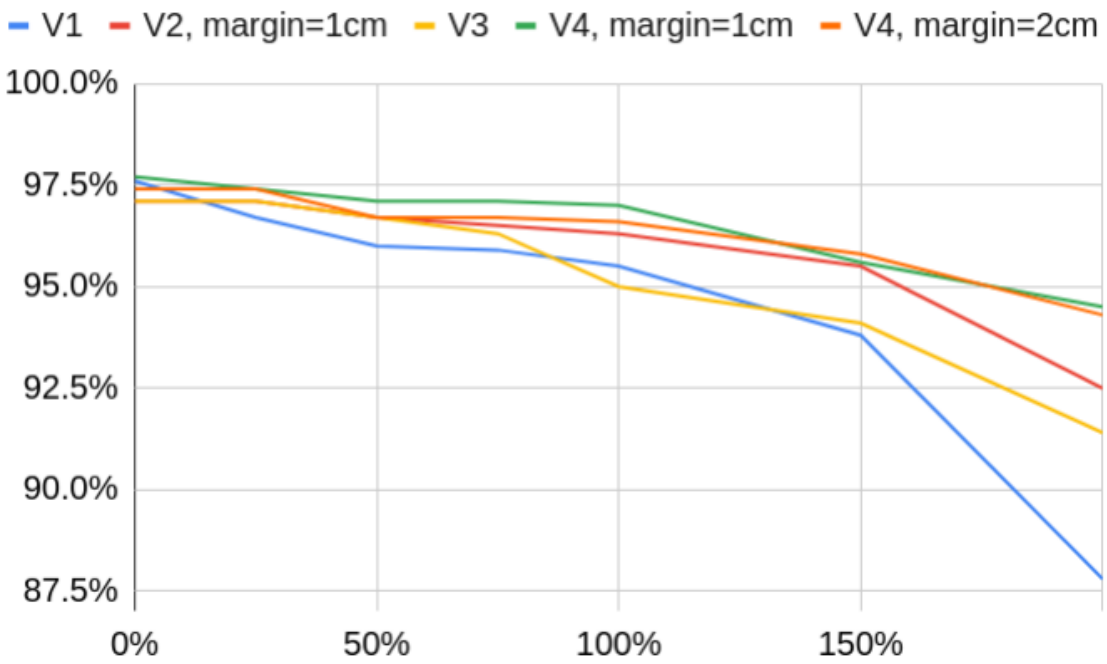


FIGURE 5.7: Simulation packing success rates for 10-item orders. Pose error scaling factor is given on the horizontal axis, and the success rate is given on the vertical axis. The baseline (V1) success rate drops off dramatically as pose error increases, while the robust technique (V4) is much less sensitive to error.

Table 5.2: Success rates on the physical packing platform, 5-item orders

	V1	V2	V3	V4
Success rate (%)	83	95	95	98
Max force exceeded (# per order)	2.22	1.40	1.12	0.7
Push performed (# per order)	—	—	0.07	0.03

5.4.5 testing on physical platform

For testing on the physical platform, we generated 100 sets of 5-item orders from our experiment items. The robust planning strategies (V2 and V4) use δ =1.02 cm. Tab. 5.2 summarizes the results. Both closed-loop and robust methods (V2 and V3) perform significantly better than the baseline, while the combined method (V4) performs best of all, beating the baseline success rate 98% to 83%.

We also show the number of times the maximum loading force (20 N) was exceeded, which indicates approximately how often a smash event was observed and how much damage the object sustained during packing. The same pattern is observed, showing that both closed-loop and robust strategies decrease the rate of smashing. The last row shows how many times a closed-loop push maneuver was performed, showing that these are rather rare, but are still helpful to improve success rates by a few percentage points.

FIGURE 5.8: Example of failed packing executions, for each experimental condition. In the baseline (V1), all items are planned tightly, and the *cracker box* hit a container wall, causing it to tilt, and for all subsequent items planned on top of it to smash. In robust planning (V2), although margins were enforced in planning, a substantially incorrect pose estimation caused the *bleach* item to stick out of the container, and no action is taken to correct it. In closed-loop packing (V3), the first-placed *cracker box* caught on the side and failed to be pushed in. This was the only case when the push maneuver did not work in our examples. Although this caused a failed plan, replanning prevented the subsequent items from smashing onto the *cracker box*. With both closed-loop packing and robust planning (V4), a replan was triggered after the *cracker box* and before the *bleach* item. Because the pile was treated as a fixed, rigid object, the *bleach* was placed on top of the *cracker box*, but since it was unstable, this caused it to tip over.

We recorded object classification results, pose estimation results, as well as planned and final placement for each experiments, which allows us to examine causes of cascading failures more closely. In V1, 9 failures occurred with one or more items protruding outside the box footprint, while 7 failed for objects within the container but exceeding the maximum height. A final case failed when a rigid object tipped over and stood up underneath the gripper, causing the robot to report a "maximum load exceeded" fault. Some failure cases can be traced to one predominant factor. 4 failures can be traced back to failed object identification, which leads to wrong shape estimation of the object to pack. 3 cases can be traced back to significant pose estimation error (e.g., rotation of 90 degrees, etc.). 1 case can be traced back to object tipping, which led to sudden force changes too rapid to be caught by the force control loop. The remaining failure cases are caused by cascading errors, e.g.,

slightly misplaced objects caused the object underneath to tip over, resulting in the subsequent objects to be packed exceeding the container height.

In V2, 3 cases failed when one or more item protruding outside the box footprint, and the other 2 failed during toppling events, causing items to exceed the max height. Among the failure cases, 2 can be traced back to misidentified objects, while 1 can be traced back to significant pose estimation error.

In V3, all 5 cases failed due to exceeding the max height. All objects protruding outside the box footprint were corrected by pushing maneuvers. Among the failure cases, 2 can be traced back to misidentified objects, and 2 can be traced back to significant pose estimation error.

In V4, the 2 failure cases appear similar – the *cracker box* item toppled and exceeded the max height. One case was caused by a significant pose estimation error, while in the other, the *bleach* item was pushed onto the *cracker box*, and the impact caused it to tip over.

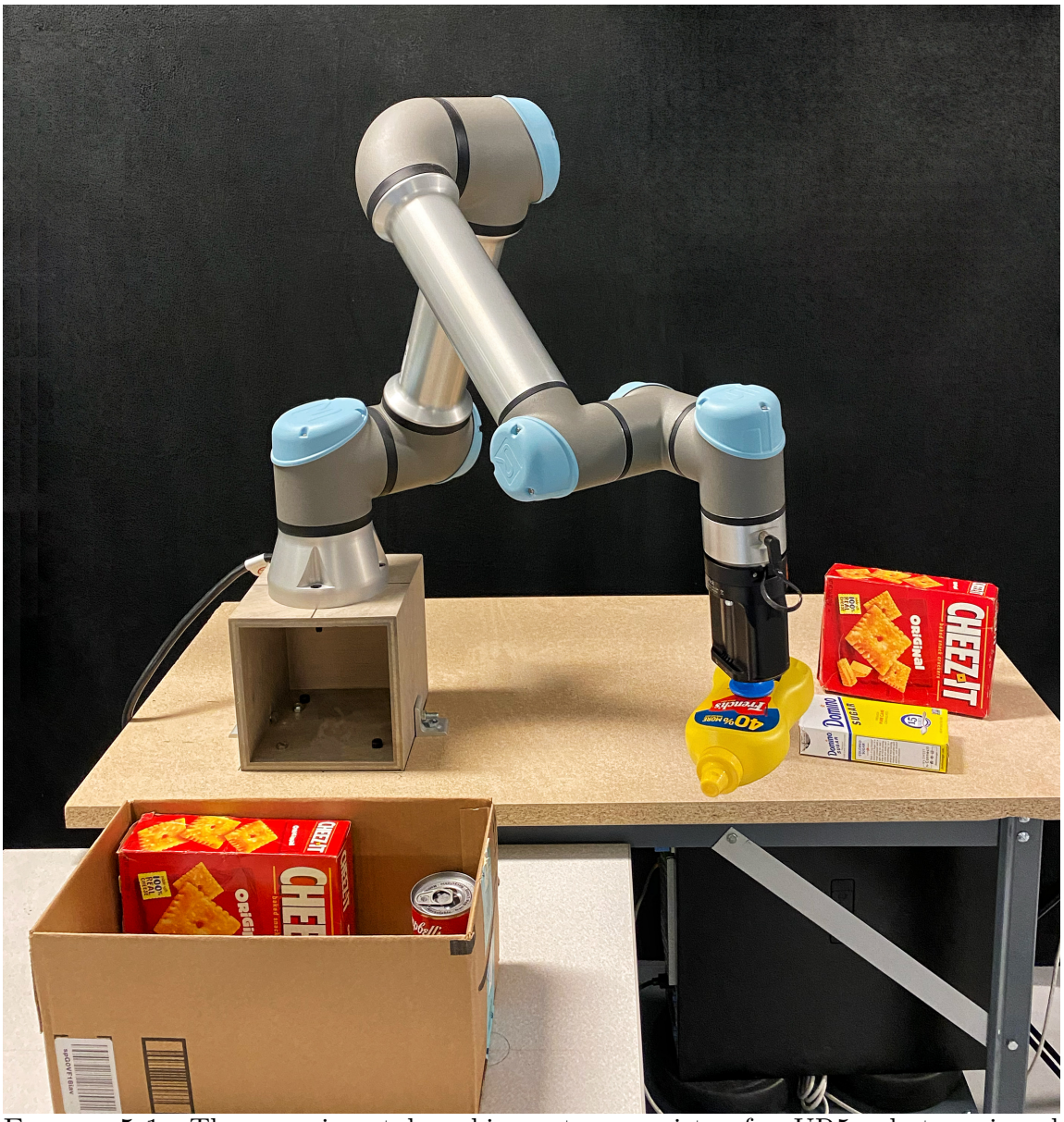


FIGURE 5.1: The experimental packing setup consists of a UR5 robot equipped with E-pick suction gripper, and overhead cameras (not pictured). Errors from pose recognition, camera-robot calibration, box location calibration, grasp planning, the center of mass estimation, and geometric modeling affect the overall success rate of packing.

6

Robot Button Pressing In Human Environments

In order to conduct many desirable functions, service robots will need to actuate buttons and switches that are designed for humans. This chapter presents the design of a robot named SwitchIt that is small, relatively inexpensive, easily mounted on a mobile robot, and actuates buttons reliably. Its operating characteristics were developed after conducting a systematic study of buttons and switches in human environments. From this study, we develop a categorization of buttons based on a set of physical properties relevant for robots to operate them. After a human calibrates and annotates buttons in the robot's environment using a hand-held tablet, the system automatically recognizes, pushes, and detects the state of a variety of buttons. Empirical tests demonstrate that the system succeeds in operating 95.7% of 234 total buttons/switches in an office building and a household environment.[1]

[1] This chapter is reproduced from Fan Wang, Gerry Chen, and Kris Hauser, "Robot Button Pressing in Human Environments,", in 2018 International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 2019, May 2018, pp. 7173–7180.

6.1 INTRODUCTION

Recent years have seen a growing interest in service robots that assist humans in their daily lives, such as in households, offices, factories, and hospitals. Interacting with physical switches and buttons is a pervasive part of human life, used to operate lighting, appliances, computers, elevators, and machinery, and will therefore be an important capability for these robots. As a result, these control devices have been designed for simple, intuitive, and reliable operation by humans, both in terms of their ergonomic mechanical properties and distinctive physical appearance. There are many types of control devices in human environments, including push buttons, toggles, slides, and knobs, and these devices will hereafter be considered synonymous to a *button* or a *switch*. Manipulating a button/switch to perform a desired effect may also be variously referred to hereafter as *button pressing*, *switch operation*, or *switching*. Operating switches with a robot with human-level ease and reliability remains a challenging task, due to the fundamentally different sensing and actuation modalities on robots vs. humans.

This chapter presents the system development and design of a compact autonomous button operating robot called SwitchIt. It is a 3DOF device based on relatively inexpensive sensing and actuator hardware (Fig. 6.1). A short manual calibration setup is performed once for a given environment using a handheld tablet and fiducial markers to identify the identity and purpose of each button. After calibration, the system recognizes and operates a wide variety of buttons automatically. It can also use sensor feedback to detect the state of many buttons and whether they have been successfully operated.

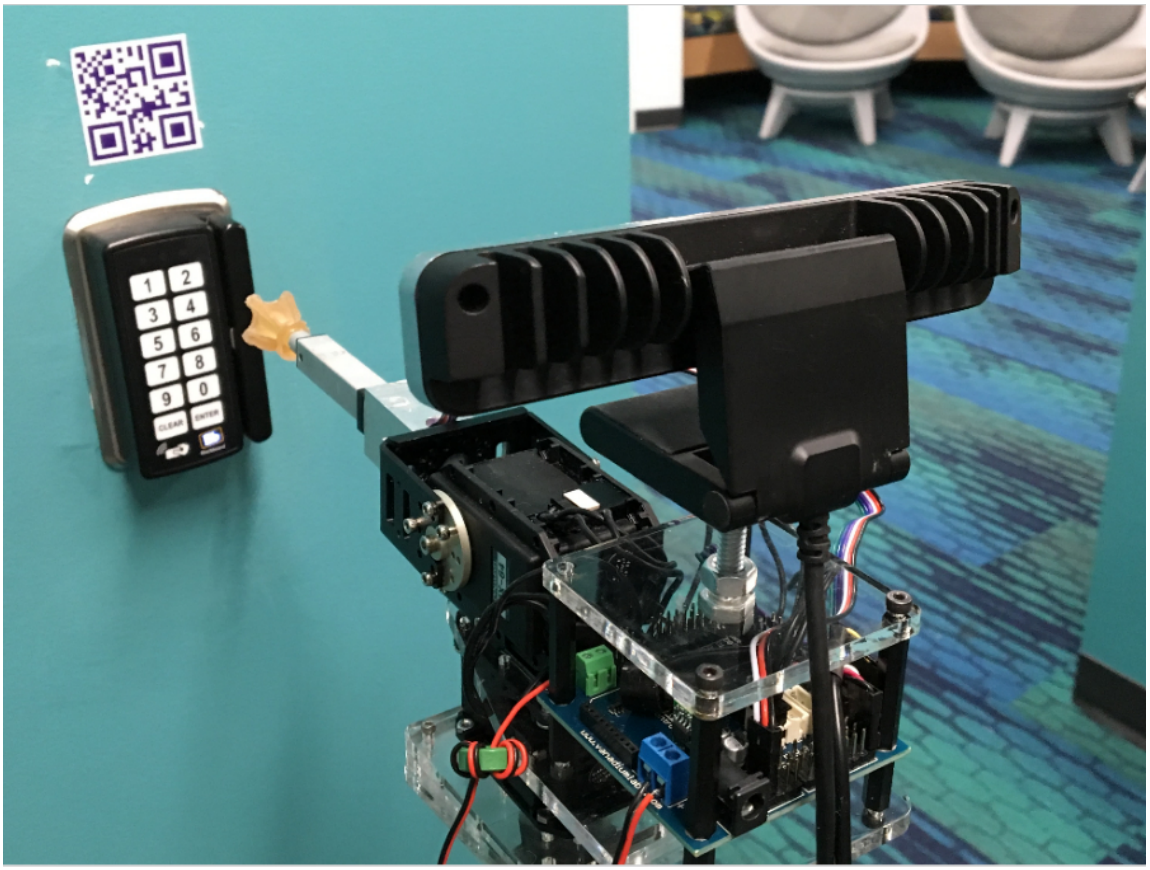


FIGURE 6.1: SwitchIt is a spherical robot equipped with an RGB-D camera. Button panels are annotated using a QR code sticker affixed during a manual setup phase. Shown here mounted on a tripod, the robot is preparing to press buttons on an electronic passcode panel.

The system integrates three primary contributions:

1. We perform a systematic categorization of over 600 buttons and switches found in offices and homes into 6 classes based on their physical properties required for robotic actuation. We propose a taxonomy of buttons from a robot's operational point of view, and characterize several relevant physical properties of these buttons, including travel, size, shape, and operating force.

2. We develop an annotation, calibration, and perception subsystem that achieves high-reliability button recognition, localization, and state detection. The calibration process also handles reflective and dark surfaces.

3. We design a compliant, scalloped end effector tip that can actuate pull buttons and turn knobs, and is robust to positioning error.

In controlled testing, our perception system localizes buttons with $< 2\,\text{mm}$ error and detects the state of toggle switches with 100% accuracy. For typical localization errors, the scalloped tip design achieves 99% repeatability compared with 89% for a cylindrical tip. We also test the system in an uncontrolled office and home environment, with 234 switches attempted in total. The platform succeeds at operating 224 (95.7%) total switches. In particular it was highly reliable at operating push buttons, sliders, rockers, and switches.

6.2 RELATED WORK

Computer vision techniques have long been employed for service robot to navigate and interact with objects in human environments [MWG$^+$10, PLS13]. Specifically, several authors have addressed button identification issues that uses visual features in RGB images to identify and locates buttons. Identifying features could either come from a priori knowledge of the type of button [WHLC10, KAO07] or based on results of machine learning [KCRN10]. Most prior research using features based on prior knowledge is performed on elevators buttons [WHLC10, KCRN10, KAO07]. The advantage of this approach both seen and unseen buttons can be detected and to some degree understood automatically. However, these algorithms usually must rely on contextual cues such as grid layout and sequential arrangement of the buttons. Machine learning strategies are also commonly used[SUB96, BKP11]. Sukhoy and Stoychev (2010) use an active machine learning strategy to identify and trigger a button autonomously. This method was used to train a robot to identify the active part of a door bell, and to trigger it effectively [SS10]. The auditory feedback of the feedback is used to determine whether the button was successfully pressed.

Several authors have addressed button identification issues with autonomous recognition algorithms that use visual features in RGB images to identify and locate buttons. Identifying features could either come from a priori knowledge of the type of button [WHLC10, KAO07] or based on the results of machine learning [KCRN10]. Most prior research using features based on prior knowledge has been performed on elevator buttons [WHLC10, KCRN10, KAO07]. The advantage of this approach is that both seen and unseen buttons can be detected and, to some degree, understood automatically. However, these algorithms must usually rely on contextual cues such as a grid layout and sequential arrangement of the buttons, or be applied to only a limited class of buttons.

Our approach asks for a small amount of environment augmentation and manual labeling to identify each button definitively. Environment argumentation has been used in other robotics systems as well to aid in object identifications. This removes ambiguity and achieves much higher accuracy than automated identification while require minimal setup time. Tools most commonly used are augmented reality tags such as RFID tags [NDRK09], QR code [LLTK14] and other artificial marks [KOM$^+$03, KOT$^+$03]. Those tags usually provide information on the location of the objects, instructions on how to interact with this specific object and a task completion criteria. We use a similar approach with QR codes, which was also previously used to enable a mobile manipulator to plug itself in [LLTK14].

The work arguably closest to ours is Nguyen et al (2009). They uses a combination of an augmented environment and a variety of sensors to help a robot interact with its environment [NDRK09]. This work does allow the robot to operate certain light switches. Force sensing and visual feedback, in the form of a change in lighting condition, is used to detect the change in button condition. In our work,we further demonstrate that high accuracy localization can be achieved using AR marker and RGB-D sensor alone, we also propose parameterized motion primitives for each class

of buttons as opposed to the explicit defining instructions for each individual object to interact with.

6.3 CHARACTERIZATION OF SWITCHES IN HUMAN ENVIRONMENTS

A button or switch changes its internal electrical connection or signal based on the force applied to its external active mechanical component. For a human or robot operator, the underlying circuitry of a switch can be considered a black box and can be mostly ignored. The main focus is performing the appropriate physical action to correctly and safely trigger the switch.

Although some industrial settings employ switches that require significant force (or even tools) to be applied, here we focus on switches that are designed to be operated by one or two human fingers. These switches are designed with size, shape, material, and mechanical resistance that are comfortable for human fingers to manipulate. Moreover, a switch usually has a distinctive appearance that indicates its mechanical functionality and semantic meaning, and usually provides feedback that can be promptly perceived and interpreted.

Although switches are ergonomically designed, the movement needed to trigger a switch safely and reliably is actually a delicate skill, acquired by humans through years of practice. Humans use memory, visual feedback, tactile feedback, and a variety of finger and hand contact strategies, and also progressively improve the efficiency and comfort of switching motions. For example, to operate stiff switches, a senior citizen with reduced hand strength will adopt finger postures that apply more leverage to stiff switches.

We collected data for over 600 switches in office and home environments to help design our robots operational characteristics. This section describes their typical distribution and operational characteristics.

6.3.1 Button Taxonomy

Laypeople usually address buttons by a common name that references its function, such as light switch, toggle button, touch pad, dimmer, or keyboard. Electricians categorized them on the basis of their electrical connection such as single-pole single-throw (SPST) or double-pole single-throw (DPST), or by their triggering mechanism (sliders, push buttons) or the type of the application it is used on (light switches, dimmers). For a robot, perhaps the most useful categorization of switches is in terms of its physical triggering mechanism.

The operating mechanism of buttons and switches can be described with respect to a normal direction facing outward from a *button panel* (a plane behind which the electrical circuits are hidden). Our proposed taxonomy divides all household buttons into following 6 types based on their operating mechanism (Fig. 6.2).

- **Push-buttons** operate via application of a force that moves the operating part inward toward the panel. The movement of the button is linear.

- **Toggles**, such as in household light switches, generally have a rod-shaped protrusion (known as a *level*) that can be rotated about an axis to toggle between 2 distinct states. Some toggle switches have 3 or even more states. Internally, a spring and plunger mechanism is used to aid in operation, and equilibrium is only achieved in the extreme positions.

- **Sliders** require lateral movement, but the level can remain in equilibrium in any state in its travel range. Typically a slider is operated with a linear motion and stays in place using friction. However, some have discrete equilibrium states enforced by internal springs.

- **Rockers** are pressed on one of 2 ends like a seesaw and toggle between two discrete states. Unlike a toggle, a rocker is triggered by applying an inward

force, primarily normal to the button panel.

- **Turn knobs** rotate along a center axis perpendicular to the panel face to adjust either a continuous or discrete value. Humans typically operate knobs using much of 2 or more fingers to achieve sufficient tangential friction about the axis.

- **Pull buttons** operate by a pulling action that moves the operating part away from the base to open or close the contacts. To operate the button, it must be solidly grasped either on the back or via friction on the sides.

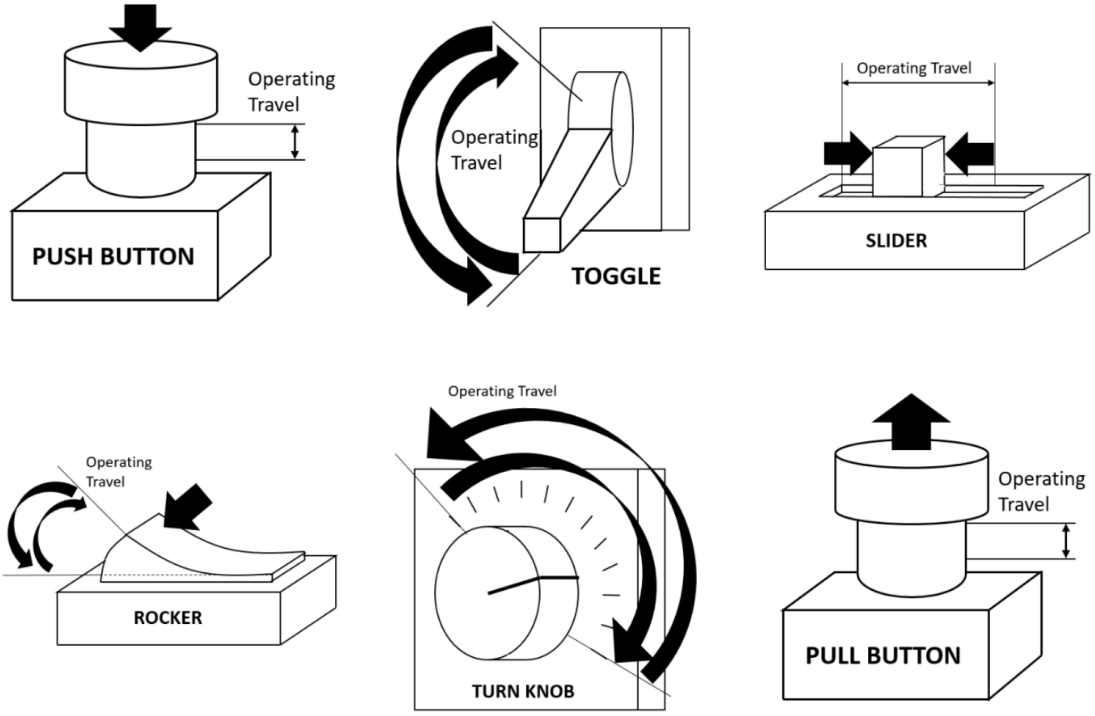


FIGURE 6.2: Buttons characterized by type.

6.3.2 Operation

Switches operate in three typical patterns: Momentary, Alternating, and Latching. In momentary operation, the switch is in an active state only while a force is being

applied, and then once unloaded, a spring returns the switch to its original position. For alternating operation, the switch's state is held after it is released, and a different force must be applied to change state (this characterizes almost all toggles, pull buttons, and rockers). In latching operation, a second force in the same location/direction of the initial force returns the switch to its initial position, which is accomplished by some spring-loaded toggling mechanism. Latching characterizes several types of push-buttons.

6.3.3 Location and Geometry

The location of the button in the environment, direction of travel, and the travel distance affect whether the operational capabilities are within the workspace of the robot. We are primiarily concerned with height, but to ensure accessibility of a mobile base it may also be important for a robot designer to consider surrounding obstacles and clutter. The geometry of the button and its relationship to surrounding buttons is also an important aspect of finger design, since it is important to be able to press the button without accidentally activating nearby buttons.

In our survey, the vast majority of buttons on the walls or doors have a height of 1.06–1.44m above the ground. The exception is elevator buttons, which have height range 0.88–1.72m. (The highest button is designed to be difficult to reach; it is to be used only in case of emergency.)

6.3.4 Force

There are different types of forces associated with button pressing, but we primarily focused on Operating Force (OF) which is the peak force needed to change the state of the button. The typical force-stroke characteristic of a button displays no movement until a breakaway force is reached, after which the force increases with increasing displacement until the peak at the operating point. Afterwards, the force follows

Table 6.1: Breakdown of button characteristics by type

	Push	Toggle	Slider	Rocker	Turn	Pull
Prevalence	66%	25%	2%	2%	2%	4%
Force (N)	0.7-12.5	2.6-6.2	0.3-0.7	2.4-9	0.7-15	3-20
$\overline{\text{Force}}$ (N)	7.15	4.61	0.5	4.5	4.2	10.2
Trav. (cm)	0.1-0.35	1-2	0.5-11	0.2-0.45	1-5.2	0.2-0.75
$\overline{\text{Trav.}}$ (cm)	0.167	1.53	4.3	0.3	2.4	0.4
$\overline{\text{Sep.}}$ (cm)	1.6	8.5	X	X	X	X
$\overline{\text{Height}}$ (cm)	119	126	122	118	76	132

a sharp decrease, and then gradually increases again until reaching the total travel distance [OMR17]. If tactile sensing is available, force profiles can be very helpful for detecting the success of button pressing. However, our system uses encoder derivatives to estimate the applied force.

Tab. 6.1 gives results of our survey, listing range and mean value of operating force (Force and $\overline{\text{Force}}$, respectively). Most buttons have OF in the 4.5–8.5N. 19.7% of buttons have OF $>$ 8.5N and 20.8% have OF $<$4.5N. Only two buttons exceeded OF $>$ 12N. For example, an elevator emergency stop pull button required 22N to operate. However, we find these buttons are not designed to be used on a daily basis.

It should be noted that all buttons we surveyed have been in operation for at least one year. New buttons are usually much harder to activate and require some usage before their internal springs soften.

6.3.5 Surface material

The body of electronic switches are usually made with metal or plastic. While highly polished plastic and metal makes attractive appearances, reflective surfaces are challenging for depth estimation with sensors. Our survey shows that 39% of buttons are made with highly polished plastic or metal and 10% are made with black material.

6.3.6 Travel distance

There are two parameters regarding travel distance: the total travel (TT) and operating travel (OT). TT indicates distance to a hard stop, while OT indicates distance until the switch is triggered. We are primarily interested in determining OT, although TT may be useful for tactile feedback. Most switch specifications suggested TT-OT to be between 0.5mm-1.8mm [C&17, OMR12].

In our survey, OT varies significantly, particularly among sliders (Tab. 6.1). However, we are most interested with the travel of push buttons, since they provide the least reliable visual cue of button state. From this data, we determined that an open-loop position controlled robot could press down a maximum of 5mm before linear actuator motor stop is detected to maximize its chance of activating a button successfully, while also being unlikely to cause damage to the button.

6.4 THE SWITCHIT PLATFORM

With the above data in mind, the SwitchIt robot accessory is designed to operate a large number of switches and to be easily mounted on a mobile robot platform. Setting up the system for use in a new environment requires a human to first perform a calibration procedure, which involves affixing QR codes to button panels and annotating reference models of the panels using a hand-held tablet. Afterward, the system will autonomously recognize any visible panel, suggest a reference position for the robot's base, and once in position, press a requested button or sequence of buttons.

6.4.1 Hardware

The robot arm used in our system is a custom 3DOF spherical robot that can pan, tilt and extend. The pan-tilt DOFs are ScorpionX MX-64 Robot Turret kit item number KIT-SXT-MX64 and extension is provided with a 50mm Firgelli Linear

actuator. We have tested the physical capability of the arm in its workspace and measured a output force exerted by the tip in a range from 6N to 18N. Although the robot is generally weaker and more susceptible to flexing the further it moves away from the center of the workspace, it should nonetheless be strong enough to trigger most buttons, which have operating force $<$ 12N. The positioning accuracy of the robot after calibration is measured to be sub-millimeter on average, and less than 3mm maximum.

For sensing we use a single Intel RealSense F200 RGB-D camera to do colour and depth capture. These cameras are inexpensive (purchase price $129), have a depth range of 20–120 cm, and work optimally in well-lit, indoor environments and diffuse objects.

Both the robot and camera are mounted on a fixed base using Plexiglass and a camera mount. We assume that the mount is attached to a mobile robot or an arm that has sufficient rigidity to keep the unit roughly in place while it presses buttons.

6.4.2 Scallop fingertip

We considered using a cylindrical rubber-coated tip, with similar shape and size to a human-finger. However, a novel scallop design proved to manipulate buttons much more reliably. The principle of the design is to increase tolerance against positioning error, and to enable motion primitives of pulling and turning which are usually difficult to achieve with a 3-DOF robot.

The design contains four rigid scallops protruding from its lateral edges (Fig. 6.3). For switches with activating rods, a scalloped channel guides motion towards the center-line, which corrects for positioning error and increases effective lateral friction. An underlying skeleton is 3D printed from rigid plastic, and this is coated with a 1-3mm thick PMC-121/30 rubber compound. The rubber coating provides compliance and large friction that helps correct for positioning errors and reduce

slippage. The coating at the forward tip of the finger is curved with a dimension and shape similar to a human's index finger. The scallop protrusions are also coated with longer rubber "skirts" to establish larger contact area when turning knobs. Finally, the rigid skeleton is also designed with a narrow "hook" located on one side of the finger, which is designed for holding onto pull buttons.

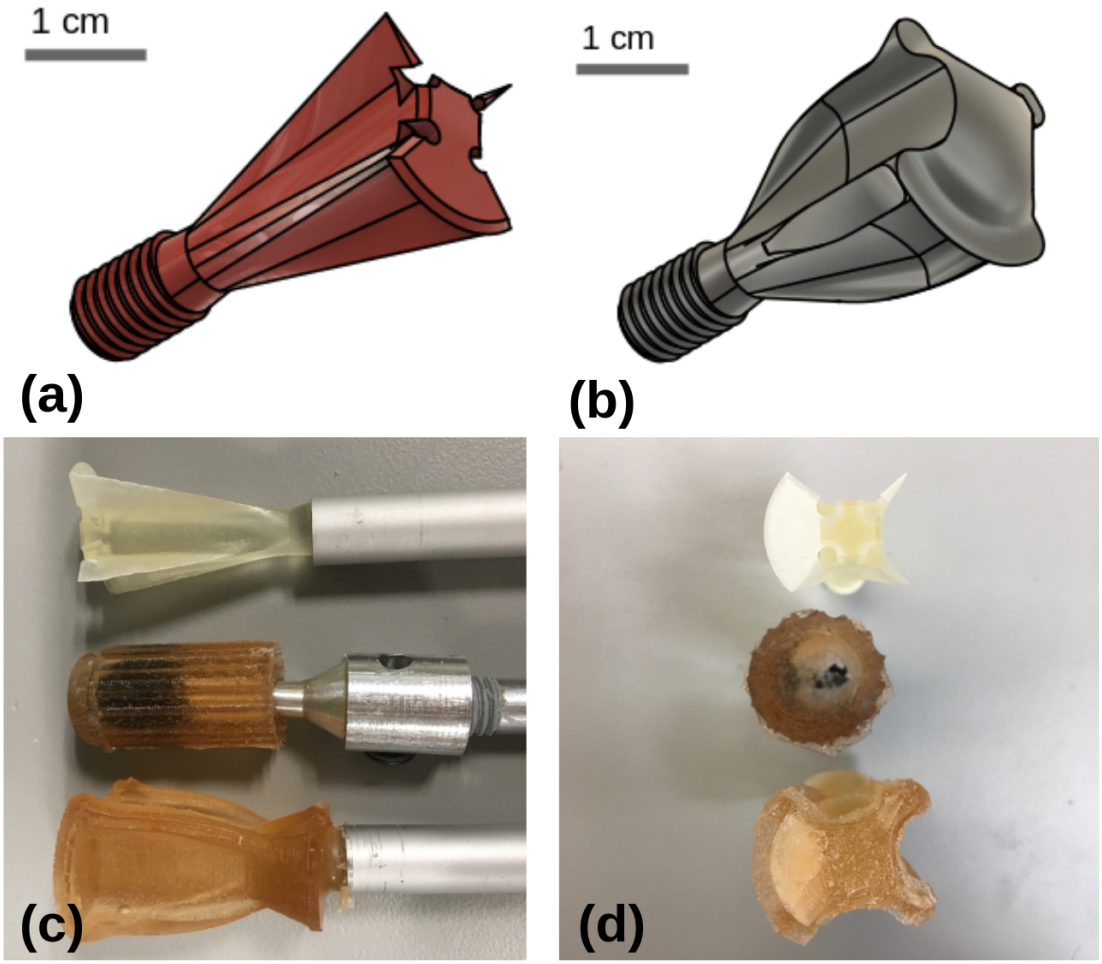


FIGURE 6.3: Cylindrical and scallop tip designs: (a) Scallop tip skeleton, (b) Scallop tip with coating, (c) Side view of tips. From top to bottom: scallop skeleton, cylindrical, scallop coated with rubber, (d) Top view of tips. From top to bottom: scallop skeleton, cylindrical, scallop coated with rubber

6.4.3 Environmental annotation and calibration

To apply the method to a new environment, a manual setup procedure must be performed to populate a database of known button panels. The process is relatively fast and the environment is minimally altered. For example, to complete annotation, calibration, and information entry for a medical device panel with 10 buttons takes less than 4 minutes (Fig. 6.4). The procedure consists of the following steps:

- The user affixes a QR code on or near the button panel.
- Using a tablet with attached RGB-D camera, the user takes a picture of the panel and provides an identifier for the panel.
- Guided by the annotation GUI, the user adds each button by name, type, and designates areas of interest on the picture.
- The panel identifier, QR code, RGB-D information, button names, type, location, size, and areas of interest are saved to a database.

It should be noted that dark or highly reflective panels and buttons cause problems with depth estimation, which could lead to erroneous 3D button location estimates. To accommodate these types of materials, the user should temporarily apply matte tape (masking or painter's tape) to the button panel when capturing the reference RGB-D image. After calibration the tape may be removed.

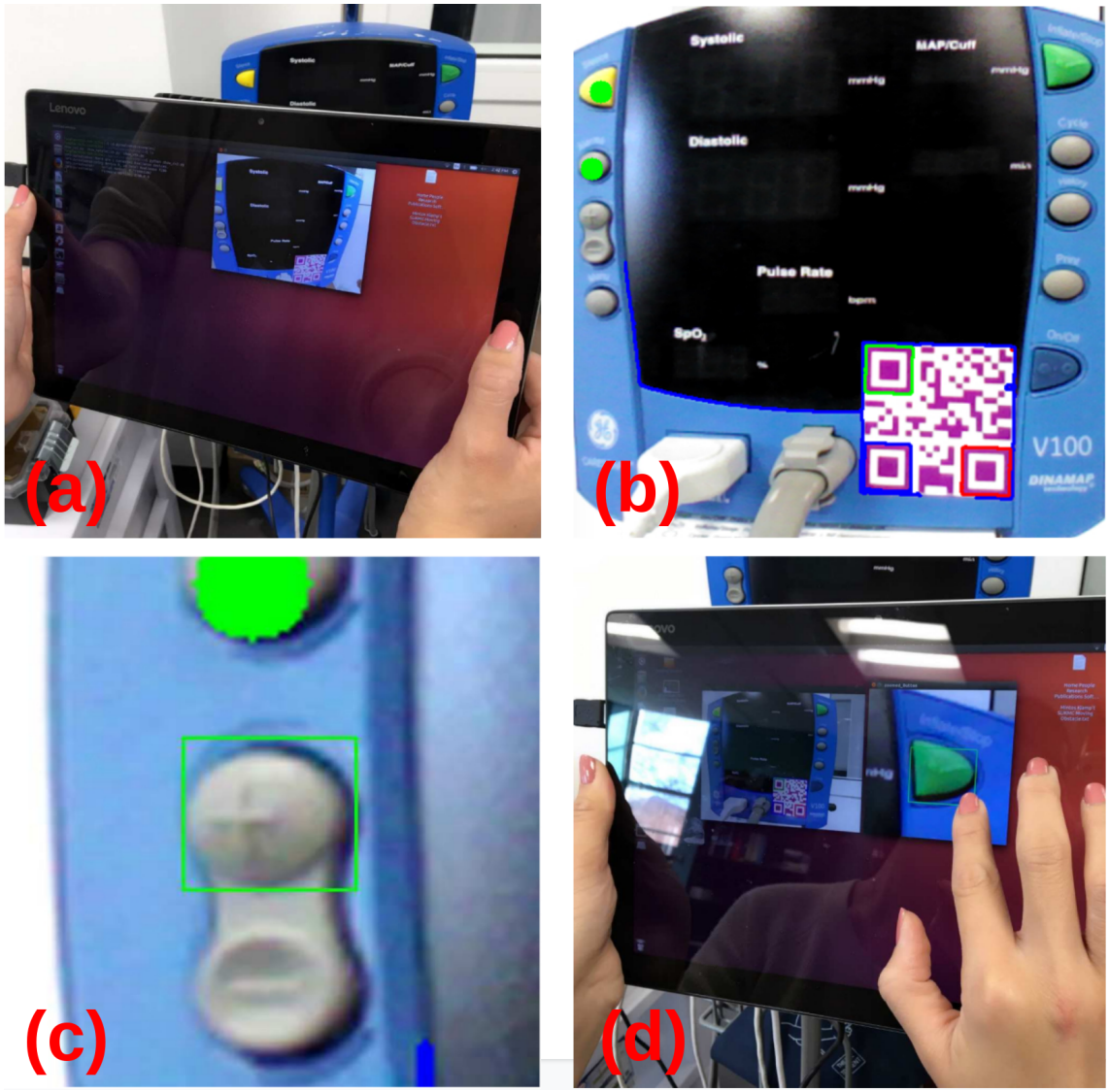


FIGURE 6.4: Steps of the panel calibration process: (a) Taking a picture of the panel panel with RGB-D camera, (b) Tap on a button to zoom in. Already calibrated buttons will be marked, (c) Draw rectangles in the area of interest as guided for each type of buttons, (d) Zoomed in button details make it easier for operators to calibrate with higher accuracy

One omission of the current procedure is that we do not store a 3D map of button panel locations. As a result, to use our system, a mobile base must be able to first position the camera to observe the panel's QR code. Future iterations of our system might record panel location, and incorporate simultaneous localization and mapping (SLAM) software to guide a mobile base to a desired panel.

6.4.4 Button Panel Recognition and Localization

Recognition and localization consists of an imprecise QR code localization followed by a more accurate point cloud registration via Iterative Closest Points (ICP) algorithm [BM92, CM92] . When a QR code is detected, the panel reference RGB-D image and all button annotations become available. A first guess is obtained from the QR code, which gives an estimated affine transformation between the reference image coordinates and the current camera coordinates. Since QR codes are relatively small, this estimate is often inaccurate.

To improve accuracy, we then apply ICP to match the point cloud corresponding to the reference RGB-D image to the currently observed point cloud. The QR code localization gives a reasonable initial guess for this optimization. It should be noted when depth data is missing or corrupted by dark or highly reflective surfaces, ICP is not as effective, and localization relies more on the QR code and surrounding non-reflective surfaces.

FIGURE 6.5: Point cloud and RGB view showing localization of button surface in real-time. Computed locations of the buttons are drawn as red rectangles.

Although RGB-D camera is relatively cheap, one significant drawback of this kind of camera is that they are unable to do depth measurements on objects with black or highly reflective surface. Therefore people turn to lasers and other more expensive sensing equipments for robust sensing. Buttons and buttons panels are often painted

in black or are made in very shiny material such as stainless steel. However, since we are only tracking the QR locations in real time and the current position of the buttons are calculated from the location in the database transformed by the affine transformation, the program is not actually detecting the real time position of the buttons with the RGB-D camera so that it is no issue that the RGB-D is incapable of doing the depth measurement on those buttons. However, if the entire button panel and it's surroundings are invisible to the depth camera, our program don't have the enough point cloud data to perform ICP and need to rely on the transformation provided by the QR code only.

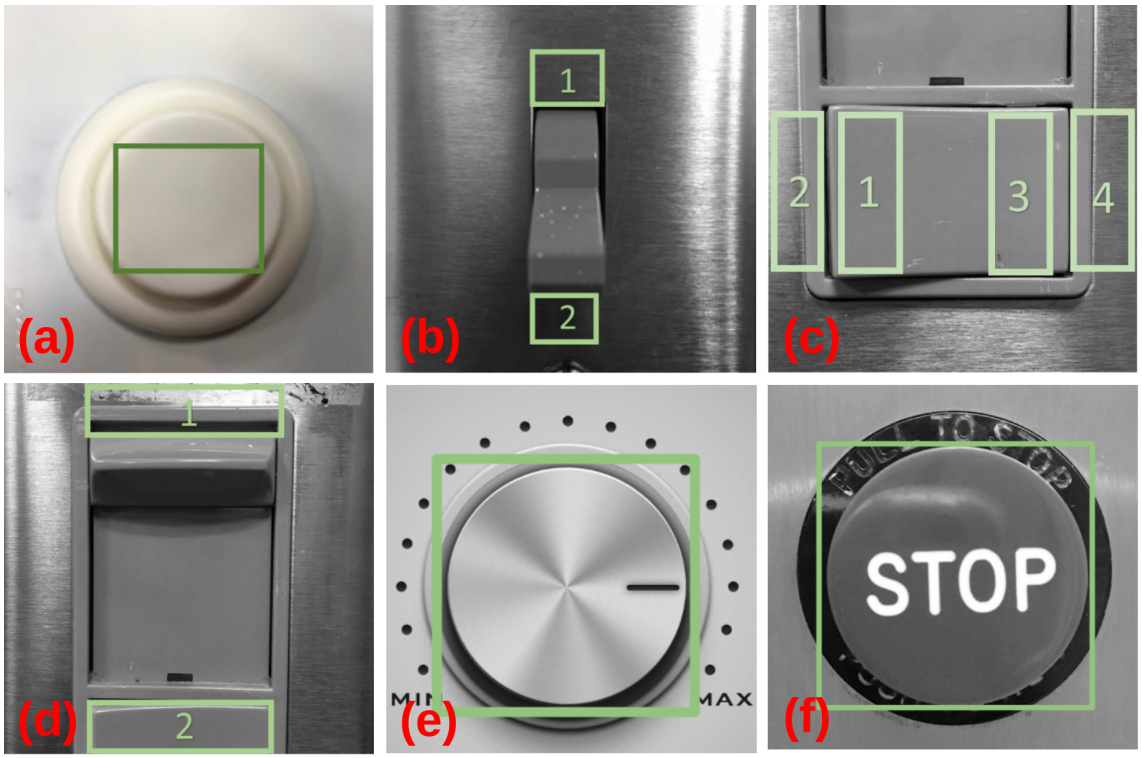


FIGURE 6.6: Calibration areas by button type: (a) Push button, (b) Toggle, (c) Rocker, (d) Slider, (e) Turn knob, (f) Pull button.

6.4.5 Button State Recognition

Many buttons provide tactile and/or auditory feedback primarily intended for humans, which can also be used by the robot to determine whether it has successfully

been switched. For robot not equipped with microphone or force sensor, it can be quite difficult to determine the state of the button and to confirm the completion of the task. Some of the previous research uses heuristics such dimming in the lighting condition to determine if light switch has been successfully switched off [NDRK09]. However, those heuristics are not always available or can be relied on.

SwitchIt uses RGB-D information to detect the state of toggle, slider, and rocker switches, as well as push-buttons with back-lit LEDs. The shape of different switch positions can be quite distinctive. The lever rod of a toggle switch rests on 2 opposite sides at different states, and rocker surfaces tilt up at different sides. Take a light switch as an example. If the switch is in the up position, the average distance from the button surface to the underlying plane will be greater on the upper side of the button.

Using a region of interest from the calibration data, our method calculates the average displacement from the panel plane in both halves of the switch, and detects the switch position by the maximum displacement. We filter out noise at 5 cm distance from the panel since this is most likely caused by obstacles in front of the camera, e.g., the robot itself. We use the same method for detecting states of rocker buttons and sliders.

State detection is challenging for push-buttons, since many do not change in appearance and shape after activation. Some push buttons do provide visual cues, for example, an LED back-light. We therefore focused on detecting these differences in the RGB image. However, we find that very few push buttons provide visual cues, and the interpretation is not always consistent (e.g. a backlight turns on vs the button itself lights up).

6.4.6 Control

The controller of the robot is initiated when the robot is in reach of the button panel, and a button pressing sequence has been specified. The robot performs one or more guarded end effector moves in Cartesian space, determined by the button type and areas of interest collected during calibration. The pushing strategy for each type of button is as follows:

- Push: Approach the center of the marked zone and push down. Pushing stops if one of the following conditions have been met: 1) the linear actuator has fully extended by 5mm, or 2) encoder readings indicate that the linear actuator has been stopped for 0.2s.

- Toggle: Linearly interpolate between center of 2 zones, in the direction needed to switch off / on.

- Rocker: Linearly interpolate from zones 1 to 2 or from 3 to 4, depending on the operational state. (We found this diagonal movement to slip less frequently than pushing straight downward.)

- Slider: Same as the toggle, except that the user / supervisor can specify a fractional travel amount.

- Turn Knob: Button center locations, button radius, and knob depth are determined from the marked region. During actuation, the tip touches the side of the knob and moves in a circular motion in the direction specified.

- Pull: First, approach the side of the button with the "hook" pointing inward and then move inward by 5mm. Interpolate toward a point 5mm in front of the pull button surface center, or until the encoder reading indicates that the linear actuator motor has been stopped for 0.2s.

6.5 EXPERIMENTS

We have done an full accuracy measurement of our system and separate localization errors with and without ICP. We also test our system with test panels that contains buttons of various types, shapes and stiffness, and finally an exhaustive real-world test in office and home environments.

6.5.1 Calibration

The robot and camera are automatically calibrated using a colored cross shape fiducial fixed to the end of the robot with a pin. Colored blob detection and averaging produces a relatively accurate measurement of the 3D tip position in the camera frame. The robot is driven to 30 random locations within its workspace and its joint coordinates are recorded. When the four circles are visible in both color and depth images and agree on the tip position within 5 mm, we consider the tip to be accurately measured. A least-squares transformation matrix between joint coordinates and sensed positions is then fit to the data. After calibration we find the mean average error in the range of 0.8mm-1mm with the maximum error less than 3mm.

6.5.2 Measurement of system accuracy

We built an apparatus to measure the cumulative positioning error of the system including human set-up error, calibration error, localization error, and hardware inaccuracies. We built a button panel with a "virtual button" in the center, whose coordinates are at the center of two colored diagonal visual features (Fig. 6.7). We conducted 500 test pushes using panel localization to determine the button location. Between each push we changed the position and orientation of the panel, with the entire panel oriented on each of its four sides and with yaw altered to up to 45°. The tip position was measured using the calibration cross marker, and "true" button center position was measured using the larger cross features. Results show the

average euclidean distance from the tip center to the sensed button center is 1.9mm, with an outlier of 8.39mm (Fig. 6.8). This outlier was likely caused by a hardware fault on the linear actuator.

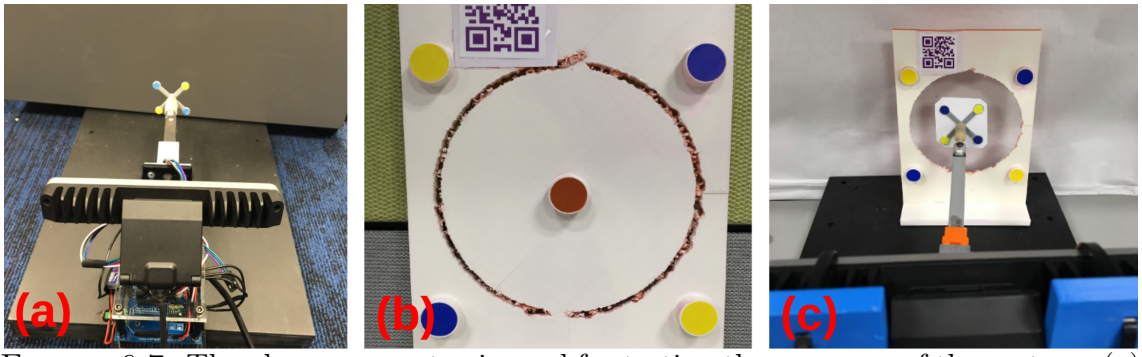


FIGURE 6.7: The above apparatus is used for testing the accuracy of the system: (a) Robot and camera are calibrated using a cross marker, (b) Center button is located at the intersection of opposing diagonal circles, (c) Center plate is removed during the test to not interfere with tip motion.

To isolate performance of our panel localization system we performed 500 localization readings on a different test panel that has several buttons (slider, rocker, push button, and switch) and a color-based fiducial on the push button. We compare the estimated button position from panel localization against the "ground truth" location from color tracking. Testing repeatability for a static panel (i.e., the effect of camera noise) gives a maximum euclidean distance error of 2.56mm using only QR localization, which is reduced to 0.5mm with ICP. Repeating this experiment while shifting and rotating the panel between each reading, localization with QR code only had a maximum error 2.76mm, reduced to 1.3mm with ICP.

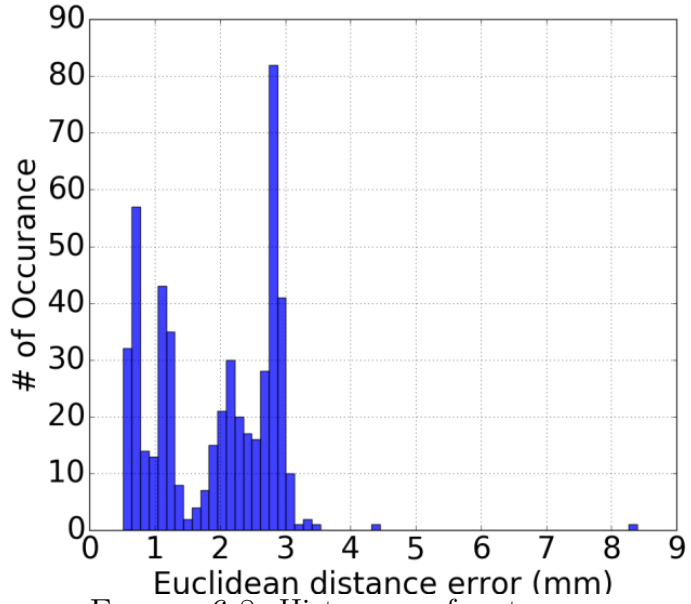


FIGURE 6.8: Histogram of system errors.

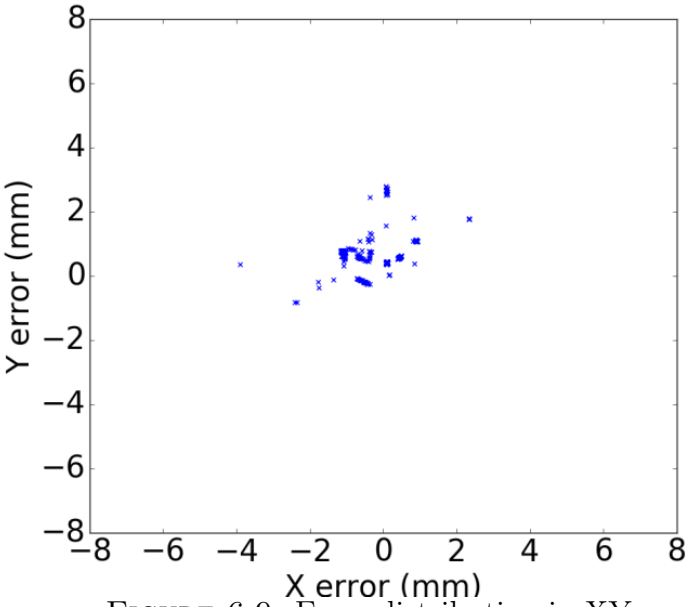


FIGURE 6.9: Error distribution in XY.

6.5.3 Test panel experiments

As a controlled test of our platform's reliability in operating switches, we built 2 test panels, one with toggle switches and another with push buttons. The push

Table 6.2: Test Panel Experiments

Button Type	Toggle	Toggle	Toggle	Toggle	Push	Push	Push	Push
Tip	Cylinder	Scallop	Cylinder	Scallop	Cylinder	Scallop	Cylinder	Scallop
ICP	Off	Off	On	On	Off	Off	On	On
Success rate	79%	96%	84%	99%	98%	99%	98%	100%
State detection	100%	100%	100%	100%	n/a	n/a	n/a	n/a
Duration (min)	20	20	22	22	13	13	17	16
Time / push (s)	12	12	13.2	13.2	7.8	7.8	10.2	9.6

button panel consists of 5mm radius buttons made with polished plastic. Each pair of buttons is separated by at least 2mm. The toggle test panel holds 5 toggles of different size, shape, and operating force ranging from 2.5N to 8.2N. All buttons are located within a 12cm by 10cm rectangle centered on a 30cm by 23cm flat panel, placed approximately 21cm away from the robot's base.

In the switch test, the robot localizes and switches on the 5 switches in sequence and then reports the perceived state of the switches. Then, it switches them off and again reports their perceived state. In case of a failure, we manually flip the switch to the desired state for the next sequence. For the pushbutton test, robot localizes and pushes 10 buttons in sequence. In each test, the robot runs 10 sequences and attempts 100 actuations in total. The panel is illuminated with indoor office lighting. To judge the impact of different components on performance, we performed these tests with the two tips (cylindrical and scallop) and with and without ICP activated.

Tab. 6.2 shows that although the cylindrical tip performed well at button pushing, it failed in roughly one fifth of the switch attempts. We observed that errors occurred due to slippage or flexing of the robot structure. The scallop tip eliminated slippage, although it still failed 4 times when activating the lower left button, which had a relatively high 8.2N OF. The failure case was further reduced to only 1 when ICP was used in localization. The scallop tip design still has a high success rate on push buttons, even though it has a larger cross section than the small buttons we tested on. This is due to the curved tip that can direct force within a small area.

6.5.4 Experiment pressing in an office building and home

We tested our system on all accessible buttons on doors/walls and large electrical appliances in a 4-story office building and a 2 story residential town house. We mounted our robot on an adjustable-height stand with lockable wheels and pushed the robot along the corridor while testing on all buttons that are accessible and safe to test on.

We have tested on 98 different button panels that contain a total of 379 individual buttons. The test set covers 39 distinct classes of panels and all 6 button types. We tested all distinct button on every panel, but when panels contain many identical buttons, we did not test every button. Specifically, if a panel contains exact duplicates of one button (such as a numeric keypad), we only tested 2 or 3 of them at extreme positions. In total, we asked the robot to operate 234 buttons.

Our robot succeeded in activating 224 of 234 buttons at the first attempt. We note that several of these buttons were quite challenging (Fig. 6.10). Successes include office passcode entry with small, stiff, and slippery metallic buttons; small and stiff rockers; turn knobs on a classroom stereo control; and non-conventional light switches that are activated from the side.

FIGURE 6.10: A selection of button panels that SwitchIt succeeded in activating: (a) Door passcode, (b) Disabled door exit, (c) Thermostat buttons, (d) Switch with dimmer, (e) Office stereo control, (f) Security door entrance, (g) Electronic keypad, (h) Light switch

6.5.5 Test panel experiments

Out of the 10 failures, 3 switches are within the capability of the device, but failed due to various positioning errors. The other 7 were of 4 button designs our system currently cannot actuate (Fig. 6.11). They include "push to stop, pull to run" emergency stop button with a shallow smooth indentation for human finger to pinch. The hook in our finger cannot establish a solid hold onto this indentation. The other 3 are variations of turn knobs such as an old-style timer that requires more than 20N to turn, 3 oven temperature knobs that must pushed in while turning, and 2 washing machine controls that are "pull to start, push to stop, and turn to select." More work is needed to develop actuation strategies for these atypical button types.

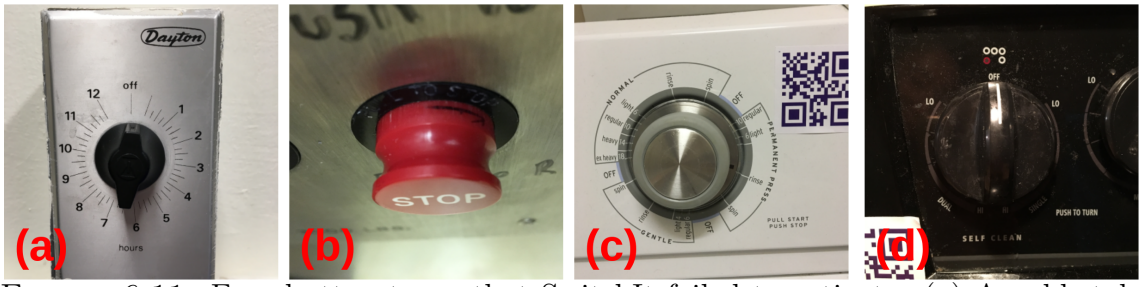


FIGURE 6.11: Four button types that SwitchIt failed to activate: (a) An old-style timer, (2) An elevator pull button, (3) A washer control button, (4) A turn knob on oven.

7

Conclusions

An automated robot warehouse brings potential benefits such as increased uptime, higher total throughput, and lower accident rates. In particular, autonomous dense packing of arbitrary objects improves the storage capacity, decreases the delivery cost, and saves packing materials. This dissertation proposes a formulation of the packing problem that is tailored to the automated warehousing domain. Besides minimizing waste space inside a container, the problem requires stability of the object pile during packing and the feasibility of the robot motion executing the placement plans. To address this problem, a set of stability and robot packability constraints are formulated, and a constructive packing pipeline is proposed to pack geometrically complex, non-convex objects while satisfying these constraints. To evaluate the proposed planner under real-world uncertainties such as vision, grasping, and modeling errors, a systematic evaluation on a state-of-the-art physical packing testbed is performed to study the sources of error and to model their magnitude. Exhaustive experiments are conducted in a Monte Carlo simulation and on the physical testbed to examine the packing placements' feasibility under open-loop baseline conditions as well as using two strategies for improving the robustness of robotic packing. Em-

pirical results demonstrate the proposed planner produces stable and high-quality packing plans compared with other 3D packing methods, and a success rate of up to 98% can be achieved on a physical robot when using robustness measures despite cascading real-world uncertainties.

7.1 Summary of Contributions

My contributions are as follows:

1. In Chapter 2, a constructive pipeline is developed that can pack geometrically complex, non-convex objects with stability while satisfying robot constraints. A new Heightmap-Minimization heuristic is proposed as a positioning heuristic for efficient 3D irregular shape packing. Simulation results on exhaustive datasets demonstrate the pipeline's effectiveness and the advantage of the new heuristic in finding stable and robot-packable plans. Robot-packable plans are shown to be far more successful in open-loop execution than non-overlap methods used in prior work.

2. In Chapter 3, two novel packing problems with nondeterministic item ordering are formulated. We presented practical solvers that handle irregular 3D shapes and item sets up to size 10. This work poses several interesting theoretical and practical questions, such as the minimal number of plans needed to guarantee NDOP coverage, whether complete QOP algorithms exist for rectilinear items, whether efficiency gains are possible with multiple identical items, and whether restrictions on item shape can overcome exponential worst-case complexity.

3. In Chapter 4, an easy to use and highly accurate in-hand object scanning pipeline is proposed. Our method improved tracking accuracy for fast-changing objects and led to a better reconstruction of 3D models. The only equipment

needed for the method is an RGB-D camera and standard PC. Moreover, the software is easy to use for novices, taking approximately 2 minutes of manual input per object.

4. In Chapter 5, a thorough analysis of multiple sources of error in an integrated, automatic robotic packing system is conducted. This chapter also presents two strategies for overcoming these errors to increase packing success rates. In the robust planning strategy, geometric collision margins were added to avoid inadvertent contact. In the closed-loop strategy, vision was used to sense the state of the object pile, to push items poking out of the container, and to replan if items slipped from their planned positions. The combination of both strategies led to higher success rates than either alone, and much higher success than open-loop planning. In all strategies, force feedback was also important to ensure items were successfully grasped and to avoid crushing items.

7.2 Future Research

More robust stability checking One issue we encounter is that the method is as good as the input models. In the case of a very bad object modeling, the placed pile is still likely to be disturbed and collapse. It is possible to further increase the execution success rate by implementing even more conservative planning, such as implementing a better stability checker than using checking only static stability. One method may be to check multiple perturbed gravity directions for feasibility [OR06], but it is still a challenge to ensure robustness with varying centers of mass and surface friction. We are also interested in improving replanning stability tests by identifying the object pile's shifted configuration rather than treating it as a fixed, infinite-mass object.

Packing "difficult" objects Our proposed methods are mostly based on rigid objects and assume that these objects can be reliably captured with vision sensors.

However, warehouses handle a wide variety of objects, many of which have translucent or transparent packaging and reflective surfaces, and some of them are soft and deformable. Handling the "difficult" objects poses new challenges relating to sensing and manipulation in dense packing. A more versatile robot system may need to be developed that utilizes multi-sensor fusion, tactile sensing and/or multi-finger hand to learn a better representation of such objects. Improvement of the current planning algorithm is needed to plan tight placement and use space more efficiently for soft objects based on its level of deformation.

Bibliography

[AAB18] Steven Diamond Akshay Agrawal, Robin Verschueren and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[AAD07] Pedram Azad, Tamim Asfour, and R. Dillmann. Stereo-based 6d object localization for grasping with humanoid robot systems. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 919–924, 2007.

[AB94] R. Adams and L. Bischof. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(6):641–647, June 1994.

[ACVB09] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[AdKR17] Kaveh Azadeh, M.B.M. de Koster, and Debjit Roy. Robotized warehouse systems: Developments and research opportunities. *ERIM report series research in management Erasmus Research Institute of Management*, 2017.

[Art66] Jr Art, Richard Carl. *An approach to the two dimensional irregular cutting stock problem.* PhD thesis, Massachusetts Institute of Technology, 1966.

[ASA+17] A. A. Arabi, P. Sarkar, F. Ahmed, W. R. Rafie, M. Hannan, and M. A. Amin. 2d mapping and vertex finding method for path planning in autonomous obstacle avoidance robotic system. In *2017 2nd International Conference on Control and Robotics Engineering (ICCRE)*, pages 39–42, April 2017.

[ASS+12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(11):2274–2282, 2012.

[BCR80] B. Baker, E. Coffman, Jr., and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[BDD95] H. Bruyninckx, S. Dutre, and J. De Schutter. Peg-on-hole: a model based solution to peg and hole alignment. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 2, pages 1919–1924 vol.2, 1995.

[BETG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[BH10] John J. Bartholdi and Steven T. Hackman. Warehouse and distribution science release 0.94 check for the latest version at www.warehouse-science.com. In *Georgia Institute of Technology*, 2010.

[Bha18] Rajen Bhatt. Skin segmentation data set. "https://archive.ics.uci.edu/ml/datasets/skin+segmentation, 2018. Accessed: 2018-09-12.

[BHKW10] E. K. Burke, R. S. R. Hellier, G. Kendall, and G. Whitwell. Irregular packing using the line and arc no-fit polygon. *Operations Research*, 58(4-part-1):948–970, 2010.

[BK04] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1124–1137, 2004.

[BKK+11] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth. Robotic roommates making pancakes. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 529–536, Oct 2011.

[BKP11] Abdeslam Boularias, Oliver Kroemer, and Jan Peters. Learning robot grasping from 3-d images with markov random fields. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1548–1553. IEEE, 2011.

[BLP13] Julia A. Bennell, Lai Soon Lee, and Chris N. Potts. A genetic algorithm for two-dimensional bin packing with due dates. *International Journal of Production Economics*, 145(2):547 – 560, 2013.

[BM92] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[BMAK14] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, April 2014.

[BR02] Fausto Bernardini and Holly Rushmeier. The 3d model acquisition pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002.

[BV02] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2383–2388 vol.3, Sept 2002.

[C&17] C&K Switches. Product catalog. "http://www.ckswitches.com/media/1416/ps.pdf, 2017. (accessed March 2, 2017).

[CB19] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. "http://pybullet.org, 2016–2019.

[CBB+18] 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 first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188, Jan 2018.

[CC15] B. Chandrasekaran and J. M. Conrad. Human-robot collaboration: A survey. In *SoutheastCon 2015*, pages 1–8, April 2015.

[CCSS01] Yung-Yu Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 2, pages II–264–II–271 vol.2, 2001.

[CFI+04] M. Callieri, A. Fasano, G. Impoco, P. Cignoni, R. Scopigno, G. Parrini, and G. Biagini. Roboscan: an automatic system for accurate and

unattended 3d scanning. In *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.*, pages 805–812, 2004.

[CFL13] Camille Couprie, Clément Farabet, and Yann LeCun. Causal graph-based video segmentation. *CoRR*, abs/1301.1671, 2013.

[CHS+18] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018.

[CJ01] János Csirik and David S Johnson. Bounded space on-line bin packing: Best is better than first. *Algorithmica*, 31(2):115–138, 2001.

[CJCH12a] S. Chitta, E. G. Jones, M. Ciocarlie, and K. Hsiao. Mobile manipulation in unstructured environments: Perception, planning, and execution. *IEEE Robotics Automation Magazine*, 19(2):58–71, June 2012.

[CJCH12b] S. Chitta, E. G. Jones, M. Ciocarlie, and K. Hsiao. Mobile manipulation in unstructured environments: Perception, planning, and execution. *IEEE Robotics Automation Magazine*, 19(2):58–71, June 2012.

[cl12] Flexible collision library, 2012.

[CLR07] Ayoub Insa Corréa, André Langevin, and Louis-Martin Rousseau. Scheduling and routing of automated guided vehicles: A hybrid approach. *Computers and Operations Research*, 34(6):1688 – 1707, 2007. Part Special Issue: Odysseus 2003 Second International Workshop on Freight Transportation Logistics.

[CLT+17] J. Cheng, S. Liu, Y.-H. Tsai, W.-C. Hung, S. Gupta, J. Gu, J. Kautz, S. Wang, and M.-H. Yang. Learning to segment instances in videos with spatial propagation network. *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*, 2017.

[CM92] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145 – 155, 1992.

[CM16] Benjamin Chandler and Ennio Mingolla. Mitigation of effects of occlusion on object recognition with deep neural networks through low-level image completion. *Intell. Neuroscience*, 2016:13–, June 2016.

[CMC20] CMC. Cmc. “https://www.cmcmachinery.com, 2020.

[CMPT+17] S. Caelles, K. K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. V. Gool. One-shot video object segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5320–5329, 2017.

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[CPT08] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing*, 20(3):368–384, 2008.

[CSB+17] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, April 2017.

[CTWY17] J. Cheng, Y. H. Tsai, S. Wang, and M. H. Yang. Segflow: Joint learning for video object segmentation and optical flow. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 686–695, 2017.

[CW13] H. S. Chang and Y. C. F. Wang. Superpixel-based large displacement optical flow. In *2013 IEEE International Conference on Image Processing*, pages 3835–3839, 2013.

[CWS+15] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, 2015.

[Das19] Jeffrey Dastin. Amazon rolls out machines that pack orders and replace jobs. “https://www.reuters.com/article/us-amazon-com-automation-exclusive/exclusive-amazon-rolls-out-machines-that-pack-orders-and-replace-jobs-idUSKCN1SJ0X1, 2019.

[DAV] DAVIS2017. Benchmark video object segmentation on davis. “http://davischallenge.org/soa˙compare.html. Accessed: 2018-09-12.

[DB16] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[dBKM+05] Edgar den Boef, Jan Korst, Silvano Martello, David Pisinger, and Daniele Vigo. Erratum to "the three-dimensional bin packing problem": Robot-packable and orthogonal variants of packing problems. *Operations Research*, 53(4):735–736, 2005.

[DCS+17] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CoRR*, abs/1702.04405, 2017.

[DGH07] Zeng Dehuai, Xu Gang, and Wang Hai. Study on teleoperated home care mobile robot. In *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 43–46, Dec 2007.

[DNRK10] T. Deyle, H. Nguyen, M. Reynolds, and C. Kemp. Rfid-guided robots for pervasive automation. *IEEE Pervasive Computing*, 9(2):37–45, April 2010.

[DR19] Siyuan Dong and Alberto Rodríguez. Tactile-based insertion for dense box-packing. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7953–7960, 2019.

[Dra05] Samuel Drake. *Using Compliance in Lieu of Sensory Feedback for Automatic Assembly*. PhD thesis, MIT, 08 2005.

[DVD02] Kathryn A. Dowsland, Subodh Vaid, and William B. Dowsland. An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research*, 141(2):371 – 381, 2002.

[Ege09] Jens Egeblad. Placement of two- and three-dimensional irregular shapes for inertia moment and balance. *International Transactions in Operational Research*, 16:789 – 807, 06 2009.

[EK09] Leah Epstein and Elena Kleiman. Resource augmented semi-online bounded space bin packing. *Discrete Applied Mathematics*, 157(13):2785–2798, 2009.

[ENO07] Jens Egeblad, Benny K. Nielsen, and Allan Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249 – 1266, 2007.

[EW15] Stefan Edelkamp and Paul Wichern. Packing irregular-shaped objects for 3d printing. *Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol 9324*, pages 45–58, 09 2015.

[FA03] T. Fraichard and H. Asama. Inevitable collision states. a step towards safer robots? In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 388–393 vol.1, Oct 2003.

[Far03] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[FM98] D. Floreano and F. Mondada. Evolutionary neurocontrollers for autonomous mobile robots. *Neural Networks*, 11(7):1461 – 1478, 1998.

[FPZ03] Oluf Faroe, David Pisinger, and Martin Zachariasen. Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing*, 15(3):267–283, 2003.

[Fre96] Jon W. Freeman. Hard random 3-sat problems and the davis-putnam procedure. *Artificial Intelligence*, 81(1):183 – 198, 1996.

[GHLL16] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *ArXiv*, abs/1610.00633, 2016.

[GHLL17] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, May 2017.

[Gir15] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[GKHE10] M. Grundmann, V. Kwatra, M. Han, and I. Essa. Efficient hierarchical graph-based video segmentation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2141–2148, 2010.

[GKMS08] Ewgenij Gawrilow, Ekkehard Köhler, Rolf H. Möhring, and Björn Stenzel. *Dynamic Routing of Automated Guided Vehicles in Real-time*, pages 165–177. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[GLSL16] Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016.

[GMM90] H. Gehring, K. Menschner, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44(2):277 – 288, 1990.

[GMZ+99] Kenneth Y. Goldberg, Brian Mirtich, Yan Zhuang, John Craig, Brian Carlisle, and John F. Canny. Part pose statistics: estimators and experiments. *IEEE Trans. Robotics and Automation*, 15:849–857, 1999.

[GO06] A. Miguel Gomes and José F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811 – 829, 2006. Feature Cluster: Heuristic and Stochastic Methods in Optimization Feature Cluster: New Opportunities for Operations Research.

[GPS89] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(2):271–279, 1989.

[GR80] J.A. George and D.F. Robinson. A heuristic for packing boxes into a container. *Computers and Operations Research*, 7(3):147 – 156, 1980.

[Gro19] Ryan Gross. How the amazon go store's ai works. "https://towardsdatascience.com/how-the-amazon-go-store-works-a-deep-dive-3fde9d9939e9, 2019.

[Hag04] H. A. Hagras. A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots. *IEEE Transactions on Fuzzy Systems*, 12(4):524–539, 2004.

[Hau20] Kris Hauser. Klampt - intelligent motion laboratory at duke university, 2020.

[HCT+11] Xin Han, Francis YL Chin, Hing-Fung Ting, Guochuan Zhang, and Yong Zhang. A new upper bound 2.5545 on 2d online bin packing. *ACM Transactions on Algorithms (TALG)*, 7(4):50, 2011.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[HHC+11] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 International Conference on Computer Vision*, pages 858–865, Nov 2011.

[HJJ+13] Hyeonjun Park, Ji-Hun Bae, Jae-Han Park, Moon-Hong Baeg, and Jaeheung Park. Intuitive peg-in-hole assembly strategy with a compliant manipulator. In *IEEE ISR 2013*, pages 1–5, 2013.

[HPL17] Guoheng Huang, Chi-Man Pun, and Cong Lin. Unsupervised video co-segmentation based on superpixel co-saliency and region merging. *Multimedia Tools Appl.*, 76(10):12941–12964, May 2017.

[HSKMG09] H. Hamer, K. Schindler, E. Koller-Meier, and L. V. Gool. Tracking a hand manipulating an object. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1475–1482, 2009.

[HT01] E. Hopper and B. C. H. Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16(4):257–300, Dec 2001.

[HZRS15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[HZRS15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[HZRS15c] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[IKH+11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 559–568, 2011.

[JDU+74] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[JPS13] Bonfim A. Junior, Plácido R. Pinheiro, and Rommel D. Saraiva. A hybrid methodology for nesting irregular shapes: Case study on a textile industry. *IFAC Proceedings Volumes*, 46(24):15 – 20, 2013. 6th IFAC Conference on Management and Control of Production and Logistics.

[JR02] Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.

[JZLS12] Y. Jiang, C. Zheng, M. Lim, and A. Saxena. Learning to place new objects. In *2012 IEEE International Conference on Robotics and Automation*, pages 3088–3095, May 2012.

[kam88] Thomas kampke. Simulated annealing: Use of a new tool in bin packing. *Annals of Operations Research*, 16(1):327–332, Dec 1988.

[KAO07] Jeong-Gwan Kang, Su-Yong An, and Se-Young Oh. Navigation strategy for the service robot in the elevator environment. In *Int. Conf. on Control, Automation and Systems*, pages 1092–1097. IEEE, 2007.

[KCRN10] Ellen Klingbeil, Blake Carpenter, Olga Russakovsky, and Andrew Y Ng. Autonomous operation of novel elevators for robot navigation. In *IEEE Int. Conf. on Robotics and Automation*, pages 751–758. IEEE, 2010.

[KGS+11] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, May 2011.

[KH13] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, July 2013.

[KHRF11] Michael Krainin, Peter Henry, Xiaofeng Ren, and Dieter Fox. Manipulator and object tracking for in-hand 3d object modeling. *Int. J. Rob. Res.*, 30(11):1311–1327, 2011.

[KLRR13] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE International Conference on Robotics and Automation*, pages 855–862, May 2013.

[KMV+16] M. Khodabandeh, S. Muralidharan, A. Vahdat, N. Mehrasa, E. M. Pereira, S. Satoh, and G. Mori. Unsupervised learning of supervoxel embeddings for video segmentation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2392–2397, 2016.

[KOM+03] Rie Katsuki, Jun Ota, Takahisa Mizuta, Tomomi Kito, Tamio Arai, Tsuyoshi Ueyama, and Tsuyoshi Nishiyama. Design of an artificial mark to determine 3d pose by monocular vision. In *IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 995–1000. IEEE, 2003.

[KOT+03] Rie Katsuki, Jun Ota, Yusuke Tamura, Takahisa Mizuta, Tomomi Kito, Tamio Arai, Tsuyoshi Ueyama, and Tsuyoshi Nishiyama. Handling of objects with marks by a robot. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 130–135. IEEE, 2003.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[LBRF11] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *2011 IEEE International Conference on Robotics and Automation*, pages 1817–1824, May 2011.

[LDG+17] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[Lee17] Jangwon Lee. A survey of robot learning from demonstrations for human-robot collaboration. *CoRR*, abs/1710.08789, 2017.

[LH15] Jun-Ming Lu and Yeh-Liang Hsu. *Telepresence Robots for Medical and Homecare Applications*, chapter 21, pages 725–735. Wiley-Blackwell, 2015.

[LISD10] H. Liao, T. Inomata, I. Sakuma, and T. Dohi. 3-d augmented reality for mri-guided surgery using integral videography autostereoscopic image overlay. *IEEE Transactions on Biomedical Engineering*, 57(6):1476–1486, 2010.

[LKH+13] F. Li, T. Kim, A. Humayun, D. Tsai, and J. M. Rehg. Video segmentation by tracking many figure-ground segments. In *2013 IEEE International Conference on Computer Vision*, pages 2192–2199, 2013.

[LLCY15] Xiao Liu, Jia-min Liu, An-xi Cao, and Zhuang-le Yao. Hape3d—a new constructive algorithm for the 3d irregular packing problem. *Frontiers of Information Technology & Electronic Engineering*, 16(5):380–390, May 2015.

[LLS15] H. M. La, R. Lim, and W. Sheng. Multirobot cooperative learning for predator avoidance. *IEEE Transactions on Control Systems Technology*, 23(1):52–63, 2015.

[LLTK14] Seok Ju Lee, Jongil Lim, Girma Tewolde, and Jaerock Kwon. Autonomous tour guide robot by using ultrasonic range sensors and qr code recognition in indoor environment. In *IEEE Int. Conf. on Electro/Information Technology (EIT)*, pages 410–415. IEEE, 2014.

[LMO+16] Xiaoping Liao, Junyan Ma, Chengyi Ou, Fengying Long, and Xiangsha Liu. Visual nesting system for irregular cutting-stock problem based on rubber band packing algorithm. *Advances in Mechanical Engineering*, 8(6):1687814016652080, 2016.

[LMV02] Andrea Lodi, Silvano Martello, and Daniele Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410 – 420, 2002.

[LMV04] Andrea Lodi, Silvano Martello, and Daniele Vigo. Tspack: A unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, 131(1):203–213, Oct 2004.

[Low04a] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[Low04b] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer VisioN*, pages 60–91, 2004.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 131–144, 2000.

[LPK+18] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping

with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[LPKQ16] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.

[LTHF17] Y. G. Lee, Z. Tang, J. N. Hwang, and Z. Fang. Inter-camera tracking based on fully unsupervised online learning. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2607–2611, 2017.

[MA04] A. T. Miller and P. K. Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122, Dec 2004.

[MBM+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[MPV00] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.

[MS15] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, 2015.

[MŠA+13] Marcus Mast, Michal Španěl, Georg Arbeiter, Vít Štancl, Zdeněk Materna, Florian Weisshardt, Michael Burmester, Pavel Smrž, and Birgit Graf. Teleoperation of domestic service robots: Effects of global 3d environment maps in the user interface on operators' cognitive and performance metrics. In Guido Herrmann, Martin J. Pearson, Alexander Lenz, Paul Bremner, Adam Spiers, and Ute Leonards, editors, *Social*

Robotics, pages 392–401, Cham, 2013. Springer International Publishing.

[MV98] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[MWG+10] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. B. Rusu, B. Marthi, G. Bradski, K. Konolige, B. Gerkey, and E. Berger. Autonomous door opening and plugging in with a personal robot. In *IEEE Int. Conf. on Robotics and Automation*, pages 729–736, 2010.

[Naw28] B. Nawahi. Acoustic production in liquid filled ceramic environments. *J. Chem. Phys.*, 108:9893–9904, 1928.

[NCHK13] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp. Ros commander (rosco): Behavior creation for home robots. In *2013 IEEE International Conference on Robotics and Automation*, pages 467–474, May 2013.

[NDRK09] Hai Nguyen, Travis Deyle, Matt S Reynolds, and Charles C Kemp. Ppstags: Physical, perceptual and semantic tags for autonomous mobile manipulation. In *Georgia Institute of Technology*. Georgia Institute of Technology, 2009.

[Ngu88] Van-Duc Nguyen. Constructing force- closure grasps. *The International Journal of Robotics Research*, 7(3):3–16, 1988.

[NSLV86] A.Y.C. Nee, K.W. Seow, S.L. Long, and V.C. Venkatesh. Designing algorithm for nesting irregular shapes with and without boundary constraints. *CIRP Annals*, 35(1):107 – 110, 1986.

[OKA11] I. Oikonomidis, N. Kyriazis, and A. A. Argyros. Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints. In *2011 International Conference on Computer Vision*, pages 2088–2095, 2011.

[OMR12] OMRON. Product catalog. "http://www.ckswitches.com/media/1416/ps.pdf, 2012. (accessed September 9, 2012).

[OMR17] OMRON. Switches: Basic operating characteristics. "https://www.omron.com/ecb/products/sw/special/switch/basic01-08.html, 2017. (accessed March 2, 2017).

[OR06] Yizhar Or and Elon Rimon. Computation and graphical characterization of robust multiple-contact postures in two-dimensional gravitational environments. *The International Journal of Robotics Research*, 25(11):1071–1086, 2006.

[PALvdP18] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *CoRR*, abs/1804.02717, 2018.

[PASV15] J. Prankl, A. Aldoma, A. Svejda, and M. Vincze. Rgb-d object modelling for object recognition and tracking. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 96–103, 2015.

[PC08] Guido Ranzuglia Paolo Cignoni, Massimiliano Corsini. Meshlab: an open-source 3d mesh processing system. *ERCIM NEWS*, 73:47–48, 04 2008.

[PGD+17] Deepak Pathak, Ross Girshick, Piotr Dollár, Trevor Darrell, and Bharath Hariharan. Learning features by watching objects move. In *CVPR*, 2017.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PKA15] Paschalis Panteleris, Nikolaos Kyriazis, and Antonis A. Argyros. 3d tracking of human hands in interaction with unknown objects. In *BMVC*, 2015.

[PKB+17] F. Perazzi, A. Khoreva, R. Benenson, B. Schiele, and A. Sorkine-Hornung. Learning video object segmentation from static images. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3491–3500, 2017.

[PLC+12] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari. Learning object class detectors from weakly annotated video. In *2012 IEEE*

Conference on Computer Vision and Pattern Recognition, pages 3282–3289, 2012.

[PLS13] A. Pratkanis, A. E. Leeper, and K. Salisbury. Replacing the office intern: An autonomous coffee run with a mobile manipulator. In *IEEE Int. Conf. on Robotics and Automation*, pages 1248–1253, 2013.

[PPTM+16] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 724–732, 2016.

[PS03] Soon-Yong Park and Murali Subbarao. An accurate and fast point-to-plane registration technique. *Pattern Recognition Letters*, 24(16):2967 – 2976, 2003.

[PS14] S. Poullot and S. Satoh. Vabcut: A video extension of grabcut for unsupervised video foreground object segmentation. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 2, pages 362–371, 2014.

[PZK17] J. Park, Q. Zhou, and V. Koltun. Colored point cloud registration revisited. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 143–152, Oct 2017.

[QSMG16] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[RA17] Martin Rünz and Lourdes Agapito. Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. *CoRR*, abs/1706.06629, 2017.

[RC11] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, May 2011.

[RCC+16] L. Rozo, S. Calinon, D. G. Caldwell, P. Jiménez, and C. Torras. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 32(3):513–527, June 2016.

[RD05] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1508–1515 Vol. 2, 2005.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[RHHL02] Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. Real-time 3d model acquisition. *ACM Trans. Graph.*, 21(3):438–446, July 2002.

[RI94a] R.L. Rao and S.S. Iyengar. Bin-packing by simulated annealing. *Computers and Mathematics with Applications*, 27(5):71 – 82, 1994.

[RI94b] R.L. Rao and S.S. Iyengar. Bin-packing by simulated annealing. *Computers & Mathematics with Applications*, 27(5):71 – 82, 1994.

[RKB04] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.

[RKG+17] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *CoRR*, abs/1709.10087, 2017.

[RL01a] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[RL01b] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. *Efficient Variants of the ICP Algorithm*, pages 145–152, 02 2001.

[RMF12] Alberto Rodriguez, Matthew T Mason, and Steve Ferry. From caging to grasping. *The International Journal of Robotics Research*, 31(7):886–900, 2012.

[Rob17] Amazon Robotics. Amazon robotics challenge official rules, 2017.

[RRKB11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[RT00] M. A. Ruzon and C. Tomasi. Alpha estimation in natural images. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, volume 1, pages 18–25 vol.1, 2000.

[RT16] H. Ramadan and H. Tairi. Automatic human segmentation in video using convex active contours. In *2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV)*, pages 184–189, 2016.

[Rut15] Rutgers APC RGB-D Dataset, 2015. (Accessed Jan 28, 2019).

[San05] F. Sanford. Oxidized ferrous materials. *Journal of Junk*, 5(4):324–345, 2005.

[Sch18] Helene Schumacher. Is this the end of household chores? "http://www.bbc.com/future/story/20180730-could-robots-do-our-household-chores-like-laundry, 2018. (accessed Oct 2, 2018).

[Sei02] Steven S Seiden. On the online bin packing problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

[SGF10] Zvi Shiller, Oren Gal, and Thierry Fraichard. The Nonlinear Velocity Obstacle Revisited: the Optimal Time Horizon. In *Guaranteeing Safe Navigation in Dynamic Environments Workshop*, Anchorage, United States, May 2010.

[SH70] M. Smith and I. A. Hall. *Handbook of Interstellar Travel*. Dover, New York, 7th edition, 1970.

[SLG+18] Max Schwarz, Christian Lenz, Germán Martín García, Seongyong Koo, Arul Selvam Periyasamy, Michael Schreiber, and Sven Behnke. Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3347–3354, 2018.

[SLM+15] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[SLWT15] Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. Deepid3: Face recognition with very deep neural networks. *CoRR*, abs/1502.00873, 2015.

[SMZ+16] Srinath Sridhar, Franziska Mueller, Michael Zollhoefer, Dan Casas, Antti Oulasvirta, and Christian Theobalt. Real-time joint tracking of a hand manipulating an object from rgb-d input. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.

[SS10] Vladimir Sukhoy and Alexander Stoytchev. Learning to detect the functional components of doorbell buttons using active exploration and multimodal correlation. In *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, pages 572–579. IEEE, 2010.

[ST08] Ufuk Sakarya and Ziya Telatar. Graph-based multilevel temporal video segmentation. *Multimedia Systems*, 14(5):277–290, Nov 2008.

[STD09] Giovanna Sansoni, Marco Trebeschi, and Franco Docchio. State-of-the-art and applications of 3d imaging sensors in industry, cultural heritage, medicine, and criminal investigation. *Sensors*, 9:568–601, 2009.

[STS+19] Rahul Shome, Wei N. Tang, Changkyu Song, Chaitanya Mitash, Hristiyan Kourtev, Jingjin Yu, Abdeslam Boularias, and Kostas E. Bekris. Towards robust product packing with a minimalistic end-effector. *CoRR*, abs/1903.00984, 2019.

[SUB96] Marcos Salganicoff, Lyle H Ungar, and Ruzena Bajcsy. Active learning for vision-based robot grasping. *Machine Learning*, 23(2-3):251–278, 1996.

[TBS+16] Dimitrios Tzionas, Luca Ballan, Abhilash Srikantha, Pablo Aponte, Marc Pollefeys, and Juergen Gall. Capturing hands in action using discriminative salient points and physics simulation. *Int. J. Comput. Vision*, 118(2):172–193, 2016.

[TG17] Dimitrios Tzionas and Juergen Gall. 3d object reconstruction from hand-object interactions. *CoRR*, abs/1704.00529, 2017.

[TK14] Sugitha. T.S. and Gladwin Jos K.T. Optical flow based on feature match and super pixel segmentation. In *2014 Fourth International Conference on Advances in Computing and Communications*, pages 243–246, Aug 2014.

[TL19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[TLK+09] Klas Tybrandt, Karin C. Larsson, Sindhulakshmi Kurup, Daniel T. Simon, Peter Kjäll, Joakim Isaksson, Mats Sandberg, Edwin W. H. Jager, Agneta Richter-Dahlfors, and Magnus Berggren. Translating electronic currents to precise acetylcholine–induced neuronal signaling using an organic electrophoretic delivery device. *Advanced Materials*, 21(44):4442–4446, 2009.

[TPL19] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *ArXiv*, abs/1911.09070, 2019.

[Tra20] Tracxn. Top warehouse automation startups. "https://tracxn.com/d/trending-themes/Startups-in-Warehouse-Automation, 2020. (accessed June 27, 2020).

[TYB16] Y. H. Tsai, M. H. Yang, and M. J. Black. Video segmentation via object flow. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3899–3908, 2016.

[USSB97] Gunduz Ulusoy, Funda Sivrikaya-Serifoglu, and Umit Bilge. A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles. *Computers and Operations Research*, 24(4):335 – 351, 1997.

[VL17] Paul Voigtlaender and Bastian Leibe. Online adaptation of convolutional neural networks for video object segmentation. In *BMVC*, 2017.

[VSA03] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A survey on pixel-based skin color detection techniques. In *IN PROC. GRAPHICON-2003*, pages 85–92, 2003.

[VT03] Christos Voudouris and Edward P. K. Tsang. *Guided Local Search*, pages 185–218. Springer US, Boston, MA, 2003.

[VVHS15] Joaquim L. Viegas, Susana M. Vieira, Elsa M. P. Henriques, and João M. C. Sousa. A tabu search algorithm for the 3d bin packing problem in the steel industry. In António Paulo Moreira, Aníbal Matos, and Germano Veiga, editors, *CONTROLO'2014 – Proceedings of the 11th Portuguese Conference on Automatic Control*, pages 355–364, Cham, 2015. Springer International Publishing.

[Wan91] P. K. C. Wang. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

[WCH18] Fan Wang, Gerry Chen, and Kris Hauser. Robot button pressing in human environments. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7173–7180, 2018.

[wes17] westernacher. The trend towards warehouse automation. "https://westernacher-consulting.com/wp-content/uploads/2017/11/Whitepaper˙Trend˙to˙Automation˙FINAL˙s.pdf, 2017. (accessed April 29, 2020).

[WGC+10] Lei Wang, Songshan Guo, Shi Chen, Wenbin Zhu, and Andrew Lim. Two natural heuristics for 3d packing with practical loading constraints. In Byoung-Tak Zhang and Mehmet A. Orgun, editors, *PRICAI 2010: Trends in Artificial Intelligence*, pages 256–267, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[WH19a] F. Wang and K. Hauser. In-hand object scanning via rgb-d video segmentation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3296–3302, 2019.

[WH19b] F. Wang and K. Hauser. Stable bin packing of non-convex 3d objects with a robot manipulator. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8698–8704, 2019.

[WH19c] Fan Wang and Kris Hauser. Robot packing with known items and nondeterministic arrival order. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[WHLC10] Wen-June Wang, Cheng-Hao Huang, I-Hsian Lai, and Han-Chun Chen. A robot arm for pushing elevator buttons. In *SICE Annual Conference 2010, Proceedings of*, pages 1844–1848. IEEE, 2010.

[WJL+05] B. Wang, L. Jiang, J. W. Li, H. G. Cai, and H. Liu. Grasping unknown objects based on 3d model reconstruction. In *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.*, pages 461–466, 2005.

[WL08] Meng Wang and James N.K. Liu. Fuzzy logic-based real-time robot navigation in unknown environment with dead ends. *Robotics and Autonomous Systems*, 56(7):625 – 643, 2008.

[WLG08] T. Weise, B. Leibe, and L. Van Gool. Accurate and robust registration for in-hand modeling. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[WLM+16] Y. Wang, G. Liang, D. Mei, L. Zhu, and Z. Chen. A flexible capacitive tactile sensor array with high scanning speed for distributed contact force measurements. In *2016 IEEE 29th International Conference on Micro Electro Mechanical Systems (MEMS)*, pages 854–857, 2016.

[WSMG+16] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[WSP15] Wenguan Wang, Jianbing Shen, and F. Porikli. Saliency-aware geodesic video object segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3395–3402, 2015.

[WWLG09] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1630–1637, 2009.

[WWLG11] Thibaut Weise, Thomas Wismer, Bastian Leibe, and Luc Van Gool. Online loop closure for real-time interactive 3d scanning. *Computer Vision and Image Understanding*, 115(5):635 – 648, 2011.

[WXZ+19] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[XL16] F. Xiao and Y. J. Lee. Track and segment: An iterative unsupervised approach for video object proposals. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 933–942, 2016.

[XSNF17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *CoRR*, abs/1711.00199, 2017.

[YCB19] YCB Benchmarks - Object and Model Set, 2019. (Accessed Jan 28, 2019).

[YDA17] Wenzhen Yuan, Siyuan Dong, and Edward H. Adelson. Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors (Basel, Switzerland)*, 17, 2017.

[YFD+16] Kuan-Ting Yu, Nima Fazeli, Nikhil Chavan Dafle, Orion Taylor, Elliott Donlon, Guillermo Diaz Lankenau, and Alberto Rodriguez. A summary of team mit's approach to the amazon picking challenge 2015. *CoRR*, abs/1604.03639, 2016.

[YR18] Kuan-Ting Yu and Alberto Rodriguez. Realtime state estimation with tactile and visual sensing for inserting a suction-held object. *CoRR*, abs/1803.08014, 2018.

[ZH04] Defu Zhang and Wenqi Huang. A simulated annealing algorithm for the circles packing problem. In Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *Computational Science - ICCS 2004*, pages 206–214, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[ZPK18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[ZSQ99] B. D. Zarit, B. J. Super, and F. K. H. Quek. Comparison of five color models in skin pixel classification. In *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 1999. Proceedings. International Workshop on*, pages 58–63, 1999.

[ZSY+17] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois Robert Hogan, Maria Bauzá, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, Weber Liu, Thomas A. Funkhouser, and Alberto Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *CoRR*, abs/1710.01330, 2017.

[ZSY+18] A. Zeng, S. Song, K. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Dafle, R. Holladay, I. Morena, P. Qu Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3750–3757, 2018.

[ZWZ+20] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi-Li Zhang, Haibin Lin, Yu e Sun, Tong He, Jonas Mueller, R. Manmatha, Mengnan

Li, and Alexander J. Smola. Resnest: Split-attention networks. *ArXiv*, abs/2004.08955, 2020.