Robotic Pick-and-Place of Partially Visible and Novel Objects

by

Marcus Gualtieri

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Khoury College of Computer Sciences

of

Northeastern University

in

Boston, Massachusetts

Committee in charge:

Associate Professor Robert Platt, Chair
Assistant Professor Lawson Wong
Assistant Professor Christopher Amato
Associate Professor Kris Hauser
Assistant Professor David Held

Summer 2021

Abstract

Robotic Pick-and-Place of Partially Visible and Novel Objects

by

Marcus Gualtieri

Doctor of Philosophy in Computer Science

Northeastern University, Boston, Massachusetts

Associate Professor Robert Platt, Chair

If robots are to be capable of performing tasks in uncontrolled, natural environments, they must be able to handle objects they have never seen before, i.e., novel objects. We study the problem of grasping a partially visible, novel object and placing it in a desired way, e.g., placing a bottle upright onto a coaster. There are two main approaches to this problem: policy learning, where a direct mapping from observations to actions is learned, and modular systems, where a perceptual module predicts the objects' geometry and a planning module calculates a sequence of grasps and places valid for the perceived geometry. We have two contributions. The first relates to policy learning. We develop efficient mechanisms for sampling six degree-of-freedom gripper poses. Efficient sampling enables the use of established value-based reinforcement learning algorithms for pick-and-place of novel objects. Our second contribution relates to modular systems. We show that perceptual uncertainty is relevant to regrasping performance, and we compare different ways of incorporating perceptual uncertainty into the regrasp planning cost. Overall, we increase the range of objects robots can pick-and-place reliably without human intervention. This gets us a step closer to robots that work outside of factories and laboratories, i.e., in uncontrolled environments.

To Lucia and Sophia.

Contents

Acknowledgments

I had a memorable experience as a PhD student at Northeastern, due to unique opportunities and challenges. I would not have made it without the support from an extraordinary group of people.

My adviser, Robert Platt, provided suggestions critical to this research, including the use of reinforcement learning for pick-and-place and the use of a neural network to predict probability of grasp success in regrasp planning, which led to the SP approach. I am grateful for his flexibility, which was useful for both pursuing my own interests as well as for overcoming the practical challenges induced by a global pandemic. Lawson Wong, Christopher Amato, Kris Hauser, and David Held also provided helpful feedback relating to the shape completion project and to the dissertation.

Many coworkers come to mind. Andreas ten Pas helped me get started by teaching me how to use the Baxter robot and by taking me into his grasp pose detection work. Andreas' suggestions are behind every paper of mine that ended up begin any good. He was there to help, from beginning to end. Abraham Shultz was the foundation of the scooter project. He drove Andreas and myself back and forth to the Lowell train station on the days we came to assist, and he acquainted us with the various dining options near UMass Lowell. Mordechai Rynderman traveled with me to Houston in the summer of 2015 to implement novel-object grasping on Robonaut 2, and Allison Thackston taught us how to use that robot. Yuchen Xiao provided feedback on some of my papers, and we had a memorable trip to Brisbane, Australia for ICRA 2018. Other coworkers, including Di Sun, Swagatika Panda, Ulrich Viereck, Matthew Corsaro, Colin Kohler, Dian Wang, Girik Malik, and Ojdrej Biza, provided helpful discussion and advice.

I thank Hsueh-Cheng Wang, George Konidaris, and Li Yang Ku for hosting or recommending me to talk at their respective research groups. Communication with several others, including Kevin Eckenhoff, Benjamin Burchfiel, Dimitris Kanoulas, Barrett Ames, Jacob Varley, Chen-Lung Lu, John Oberlin, Nakul Gopalan, Jeffrey Mahler, Michael Laskey, Takeshi Takahashi, Michael Lanighan, Khoshrav Doctor, Karthik Desingh, Zhen Zeng, Jay Wong, Peter Pastor, Dieter Fox, Stan Birchfield, Philip Strawser, and Julia Badger, was helpful and memorable.

Most importantly, I thank my parents, Mei-Chun Chen, Shyh-Jye Su, Mary Gualtieri, and Robert Gualtieri, for their support and patience, and Hui-Wen Su, without whom, I would have neither started nor finished.

Part I

Introduction and Background

Regrasping must be performed whenever a robot's grasp of an object is not compatible with the task it must perform.

Tournassoud, Lozano-Pérez, and Mazer in *Regrasping*

Chapter 1

Introduction

1.1 Robotic Pick-and-Place

The problem of robotic pick-and-place is controlling a manipulator to grasp an object and place it down in a task-specific way [62]. For example, placing a bottle upright onto a coaster. In this thesis, we are concerned with a couple of issues that arise with this problem in a realistic setting. First, objects are only partially observed by the robot's visual sensors. This is in part due to how the sensor discretizes the world into pixels and in part due to self occlusion or occlusion by other objects. Partial visibility makes it difficult to plan robot actions which lead to grasping the object rigidly and placing the object into its desired configuration. For instance, when a classical pick-and-place planner was used with a single view of the scene, a plan was found only 10.0% of the time (versus 95.7% if the geometry is fully observed) for a bin packing task (Table 6.1 in Chapter 6). One solution is to match a model of the objects to the perceived sensor data and then execute precomputed grasps and places, as in [110]. This, however, is not an option if the objects in the scene are drawn from an infinite or unknown set. This is the second issue considered in this thesis, called *novel objects*. Models of the objects are not available in some applications of practical interest, such as warehousing, exploration, or housekeeping, where it is impossible to know what objects will be manipulated ahead of time.

1.1.1 Practical applications

Novel objects are encountered in uncontrolled, natural environments, i.e., environments not highly engineered to enhance the performance of the robot. For example, uncontrolled environments include households, warehouses, disaster zones, space, and sea. Since an exact description of these environments cannot be obtained prior to robot activity, the robot must demonstrate the capability to generalize to novel situations.

A system was developed in our lab for grasping novel household objects for persons with mobility disorders (Figure 1.1) [19]. The operator indicates which object to grasp using a laser pointer. The robot then retrieves the object or indicates to the operator that the object is out of reach. Later, the system was extended to place objects in a location designated with the laser pointer [114]. For this application, it is impossible to know ahead of time what objects the user would like to manipulate.

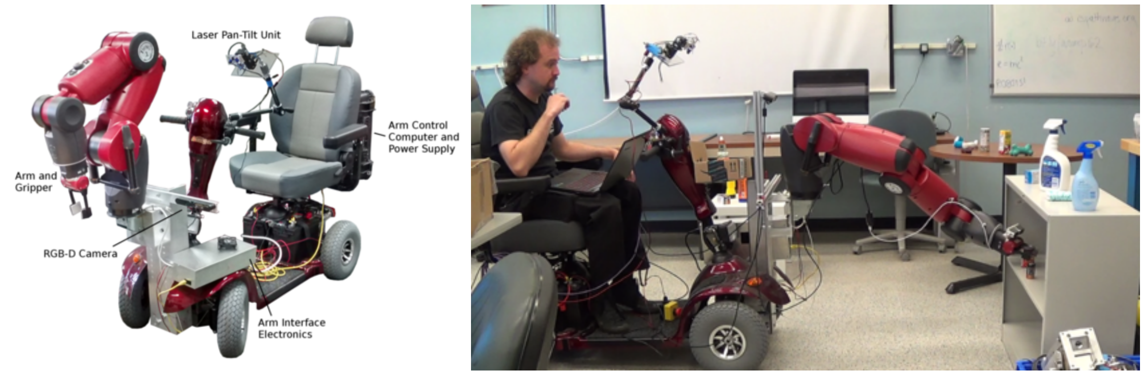

Laser Pan-Tilt Unit
Arm Control Computer and Power Supply
Arm and Gripper
RGB-D Camera
Arm Interface Electronics

Figure 1.1: Mobile grasping system to assist with activities of daily living [19].

Other applications in uncontrolled environments include litter removal [129] and warehousing, most notably the Amazon picking challenge (APC) [13, 29, 131, 77]. These applications remain, at the time of this writing, research projects rather than products because the systems have not been demonstrated to be robust to manipulating a vast set of novel objects, especially when the objects are presented in tight spaces with limited visibility.

1.1.2 Approaches overview

There are four fundamentally different ways to approach the problem of pick-and-place of partially visible, novel objects: modular architectures, policy learning, belief space planning, and active sensing. We do not argue which one is best; indeed, it could be that a combination of these approaches is ultimately adopted. Instead, the merits and disadvantages of each approach are briefly explained.

Modular architectures

A *modular* or *pipelined* system is composed of separate state estimation and control components (Figure 1.2). These components have evolved mostly independently. On the state estimation side, recent advances in deep learning have enabled accurate processing of image, point, and depth data into object segments (e.g., [28]), classifications (e.g., [47]), and completed shapes (e.g., [124]). But these perceptual components were not designed with planning in mind. On the planning side, efficient algorithms were given for regrasping a single object to get it into a goal pose [109], and manipulation planning extended this to multiple, movable objects [2]. But all of these planners assume complete and uniformly correct object models.

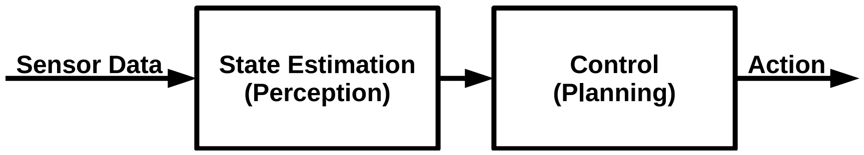


Figure 1.2: Modular architecture.

In special cases, such as linear quadratic systems, separately designed perception and planning components is optimal ([6] p. 200). But this is not the case in general. For example, consider placing a bottle upright onto a coaster. Suppose the bottom of the bottle is not visible to the robot's sensors, e.g., it is either self-occluded or occluded by another object. In this case, the perceptual component can only guess the height of the bottle. If the planner assumes the provided height is correct, it could decide to grasp the bottle near the bottom and position it just above the coaster. If the bottle is slightly shorter than guessed, the grasp will fail, and if the bottle is slightly longer than guessed, the place will be in collision with the coaster. Taking uncertainty into account, the planner would grasp a known part of the bottle and place it slightly higher than the estimated bottle height, to avoid the possibility of these failures. On the other hand, the way uncertainty is represented to the planner has implications on planning efficiency. For example, if the bottle is represented by a large, non-parametric distribution over shapes, it will take the planner a long time to consider all possible worlds in order to come up with a solution that is most likely to succeed. Thus, when designing a pick-and-place solution using a modular architecture, it is important for the planning component to be aware of the perceptual uncertainty inevitably present, and it is important for the perceptual component to represent the uncertainty in a way enabling efficient planning.

Policy learning

Another approach is to dispense with separate perception and planning components and use reinforcement learning (RL) to learn a *policy* – a function mapping observations (e.g., sensor data) to actions (e.g., grasps and places) that are likely to succeed (i.e., yield high sum of future rewards). This avoids the issues associated with separately designed perception and control components. Some examples of this approach applied to robotic manipulation include [55, 54, 20, 22, 88, 23, 115].

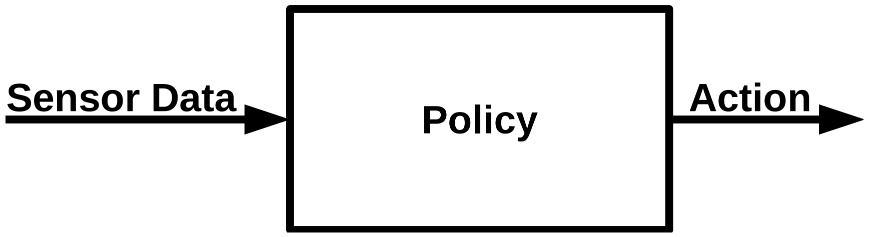


Figure 1.3: Policy architecture.

While it is convenient from an implementation point of view to have a single algorithm and a single component that learns robotic pick-and-place, this approach has a few drawbacks. Training is time-consuming, and, at the end, performance is often suboptimal, even for simple tasks (e.g., for placing mugs upright onto a shelf, a pipelined approach [67] outperforms a policy learning approach [20]). Furthermore, these systems are brittle in the sense that small variations to the environment in which they were trained can result in completely unsuccessful behavior [49].

Belief space planning

Another approach to combining planning and perception has been to plan in *belief space* – the space of probability distributions over the current state [37]. While this is a principled approach to accounting for partial observability, there are still a couple of important drawbacks. First, many of these methods require a detailed description of the observation and state transition models of the system, which can be very difficult to obtain (e.g., [38, 125, 17]). Second, planning takes place in the space of probability distributions over states, which is continuous and, for practical problems, high dimensional. For these reasons, this approach has been confined to problems with few dimensions or other simplifying structure.

Active sensing

Another option is to allow the robot to control the position of its sensor in order to get additional views of the scene. This has been shown to be helpful for increasing the grasp grasp success rate [40, 21, 25, 76], so we expect it would also be helpful for pick-and-place. A couple of issues encountered in this scenario include (i) matching the views together requires precise calibration of the sensor's pose relative to some fixed frame and (ii) it may not be possible for the robot to reach a desired viewing position, e.g., if the objects are tightly packed in a shelf, they cannot be viewed from behind, and (iii) it could be more time-consuming to move the sensor to get more views than applying other methods.

1.2 Thesis Contributions and Outline

1.2.1 Contributions

Generally speaking, the contribution is to enable more difficult instances pick-and-place of partially visible and novel objects than was previously possible. As this study began, much success had been seen with novel-object grasping, starting with Saxena et al.'s work in 2008 [95]. At this point, grasps were being picked from dense clutter at a success rate around 90% [21]. But novel-object pick-and-place is a non-trivial step up in complexity from novel-object grasping. First, pick-and-place has temporal dependencies: the way the object is grasped influences how or even if it is possible to place the object into a goal configuration. For instance, if a bottle is grasped so the hand is coming in from the bottom, it cannot be placed upright onto a coaster, due to the hand colliding with the coaster. Second, pick-and-place forces us to think about sampling gripper poses in a more general, task-independent way. Geometric heuristics were useful for sampling grasp candidates [85, 86], which were then encoded and passed into a machine learning tool for success/failure classification. But, when it comes to placing, any particular set of heuristics for sampling hand poses may fail, depending what the placement surface is and how the object is grasped. After Jiang et al. [35, 34], who considered novel-object placing without consideration to how grasps are generated, we were amongst the first to systematically investigate pick-and-place of novel objects. Specifically, our contributions are as follows:

- **Reduced pick-and-place action sampling complexity.** The space of 6-DoF, continuous, pick-and-place actions is too large to evaluate densely by uniform sampling. Two ways to reduce sampling complexity include relying on grasp detection to discretize grasp choices (Chapter 4) [20] and learning to constrain the space of samples (Chapter 5) [22, 23].

- **Extended modular systems to account for perceptual uncertainty for pick-and-place, and compared different representations of uncertainty.** We show that, with minimal modifications, existing perceptual components for object instance segmentation and shape completion can predict their own uncertainty. Additionally, we show how this can be used as a planning cost in existing pick-and-place planners. We compare four different ways of representing perceptual uncertainty (Chapter 6) [24].

1.2.2 Outline

- **Part I**. The rest of Part I is background. In Chapter 2, the problem of pick-and-place and regrasping of partially visible, novel objects is introduced, along with our assumptions. In Chapter 3, an overview of approaches to related problems is given.

- **Part II**. In Part II, two policy learning approaches to the pick-and-place problem are detailed. In both cases, a function assigning values to every action from every estimated world state is learned, and then the maximum valued feasible action from the current state is taken. The key issue that had not previously been addressed satisfactorily is the size of the action space. If the action space is infinite and high dimensional, it is not possible to evaluate all actions. In Chapter 4, grasp detection is used to discretize the choice of pick actions [20]. This limits grasp choices to a manageable set, but it also reveals a problem with the grasp detector. The grasp detector relies on geometric heuristics to sample grasp choices, and these heuristics cannot be extended to placing, which is somewhat task-specific. So in Chapter 5, the system learns to constrain the space in which grasps and places are sampled through a spatial hierarchy [22, 23].

- **Part III**. A modular approach to the pick-and-place problem is described in Chapter 6. With this approach, deep neural networks are used to segment object instances from sensor data and to predict their complete shapes [24]. After this, classical geometric planning algorithms are applied to plan the pick-and-place actions. As noted in Section 1.1.2, perceptual uncertainty can be an issue with this type of approach. This problem is addressed by having the perceptual modules estimate their own uncertainty and by incorporating this uncertainty as a planning cost. We compare four different cost functions corresponding to different ways of representing uncertainty in object instance segmentation and shape completion.

- **Part IV**. In Chapter 7, limitations of the current approaches are described, key lessons learned are discussed, and some ideas for future research are given.

Publications completed through the course of this PhD but not included in this thesis are related to grasping novel objects [21, 84], active sensing for grasping [25], assistive grasping [19], and a deictic representation for policy learning [88].

Chapter 2

Regrasping Partially Visible and Novel Objects

In this chapter, we describe the problem addressed in this thesis. In short, the problem is to find a sequence of grasps and places which results in placing an object into a goal pose. Tournassoud et al. give an efficient solution to this problem when the environment is fully observed [109]. However, in reality, we are faced with limitations in sensing, and we do not *a priori* know the objects' locations and shapes. This complicates the problem, as we can no longer guarantee a particular pick-and-place sequence will move an object to a goal. Instead, we must content ourselves with the *probability* a pick-and-place sequence solves the problem.

We start by defining an environment consisting of a robotic manipulator and partially visible, novel objects. To simplify planning, we aim for actions which either fix an object in the gripper or place an object stably. We then describe the regrasping problem, of which pick-and-place is a special case.

2.1 The Move-Binary-Effect System

The problem of regrasping a partially visible, novel object is defined in terms of a move-binary-effect system (cf. move-effect system [88, 23]). A visual description of this system is given in Figure 2.1.

Definition 2.1 (Move-binary-effect system). *A* move-binary-effect system *consists of one or more objects, one or more depth sensors, and a robotic manipulator, each situated in 3D Euclidean space. Objects are rigid masses $O_1, \dots, O_{n_{obj}} \subseteq \mathbb{R}^3$, sampled from an unknown probability distribution. The depth sensors acquire a point cloud $C \in \mathbb{R}^{n \times 3}$, consisting of points on object surfaces, each having an unoccluded line to* *a depth sensor. The manipulator is equipped with an effector with status* empty *or* holding. *The action of the robot is to move the effector to a target pose $T_{\text{eff}} \in SE(3)$, followed by an effector operation, either* open *or* close. *At each step, the robot acquires a point cloud, observes its effector status, and takes an action.*

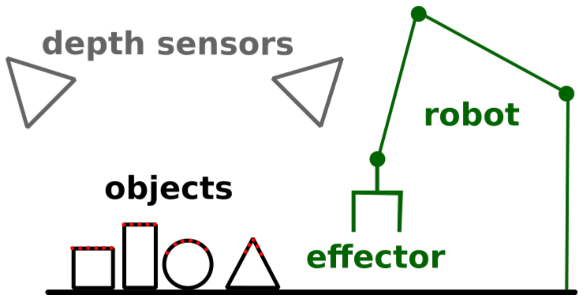


Figure 2.1: Abstract illustration of a move-binary-effect system.

For example, objects may include a table and several bottles and coasters. Depth sensors could be a combination of lidar, stereo, and structured-light sensors, carefully calibrated so point clouds from each sensor are combined into a common reference frame. The manipulator could be a 6- or 7-degree-of-freedom arm with a combination of prismatic and revolute joints. The effector could be a 2- or 3-finger gripper or a vacuum gripper.

We assume the following about the move-binary-effect system:

- The manipulator, effector, and object geometries do not intersect. In particular, normal forces prevent intersection when two bodies are in contact.
- Objects are rigid masses. This enables use of existing grasp and placement stability analysis. This analysis is difficult with deformable objects [94].
- Point clouds sample points on object surfaces, i.e., sensors do not give false or noisy readings. This is to focus on uncertainty in object shape rather than hardware-specific sensing uncertainty.
- Objects are uniformly dense and the coefficient of friction between objects and the effector is known. This is to focus on uncertainty in object shape rather than uncertainty in other object properties.
- If there are no collisions, the manipulator can be moved precisely to a target configuration. This is to focus on uncertainty in object shape rather than uncertainty in effector pose.

On the other hand, we make minimal assumptions about the objects' shapes. Objects are randomly sampled from an unknown and possibly infinite distribution. Sometimes, the objects' categories are given, where "category" ambiguously describes the distribution of objects by WordNet words (e.g., "mug" and "bottle") [69]. In this thesis, the robot never has the complete geometries of the objects it will manipulate. This arises in several, practical settings, such as household chores [19, 114] and warehousing [13, 29, 131, 77]. Thus, solutions which match a database object to sensor data (e.g., [61, 109, 8, 110]) are not directly applicable to this problem.

2.2 Static Pick-and-Place Actions

Like others before us (e.g., [109, 2, 1, 80, 101, 113]), we avoid dynamic actions, such as pushing, throwing, or rotating the object in the effector. In particular, we seek *close* actions which fix an object rigidly in the gripper and *open* actions which place an object at rest. This is to simplify planning: when applying a dynamic action to an unknown object, the outcome is difficult to model. While Yu et al. study pushing a single object of unknown shape [127], many pushes are required before outcomes are predictable, and having multiple unknown objects presents additional challenges. Besides, we want to focus on pick-and-place of partially visible, novel objects in the simplest setting possible.

Definition 2.2 (Rigid grasp). *Let $T_O^{\mathit{eff}} \in SE(3)$ be the pose of an object $O \subseteq \mathbb{R}^3$ relative to the effector pose $T_{\mathit{eff}} \in SE(3)$. Given a move-binary-effect system (Definition 2.1), an effector* close *operation forms a* rigid grasp *or a* stable grasp *on an object O if and only if T_O^{eff} remains constant from the beginning of the* close *operation until the beginning of the next effector* open *operation.*

Definition 2.3 (Stable placement). *Let $T_O \in SE(3)$ be the pose of an object $O \subseteq \mathbb{R}^3$ relative to a fixed frame, i.e., a reference frame not attached to the robot or any of the objects. Given a move-binary-effect system (Definition 2.1) where the effector is* holding *O, an effector* open *operation forms a* stable placement *or a* stable place *if and only if T_O remains constant from the beginning of the* open *operation until after the* open *operation completes and contact is again made between the robot and O.*

In the sequel, it will be useful to have efficiently verifiable sufficient conditions for rigid grasps and stable placements. When the effector is a parallel-jaw gripper, we assume an antipodal grasp (Definition 2.4) is sufficient to form a rigid grasp. Murray et al. provide technical justifications for this assumption ([79] p. 223). When an object's supporting surface is a fixed plane, horizontal with respect to gravity, Tournassoud et al. give necessary and sufficient conditions for stable placements (Definition 2.5) [109].

Definition 2.4 (Antipodal grasp, adapted from [79])**.** *A parallel-jaw gripper forms an* antipodal grasp *on an object if and only if the line connecting the contact points lies inside both friction cones.*

Definition 2.5 (Stable placement on a horizontal plane, adapted from [109])**.** *An object is placed* stably *on a horizontal (with respect to gravity) plane if and only if a face of the convex hull of the object is in contact with the plane and the ray from the object's center of mass in the direction of gravity intersects the interior of the face.*

Limiting ourselves to static pick-and-place actions will make the problem simpler, as the effects of actions will be easier to model. For the case of a parallel-jaw gripper and placements on a horizontal plane, we have efficiently computable sufficient conditions for static pick-and-place actions.

2.3 The Problem of Regrasping a Partially Visible and Novel Object

We now define the problem studied in this thesis:

Definition 2.6 (Regrasping a partially visible and novel object)**.** *Given a move-binary-effect system (Definition 2.1), a point cloud $C \in \mathbb{R}^{n \times 3}$, points on a target object $\bar{C} \subseteq C$, and a goal transform $T \in SE(3)$, find a sequence of actions where, with maximum probability, every* close *forms a rigid grasp on the target object and every* open *forms a stable placement on the target object, and if so, $\bar{C}$ is transformed by T.*

Unlike regrasping in a fully observed environment, which requires an object to be placed into a goal pose [109], we maximize the *probability* the object is placed into a goal pose. This is because, with partially visible, novel objects, it is impossible to guarantee a particular sequence of actions will displace the object to its goal, due to unexpected slippages in the hand and unstable placements.

When only one pick-and-place is allowed, Definition 2.6 reduces to a *pick-and-place problem* (Definition 2.7) [62]. We consider regrasping as it is not always possible to place the object into its goal with a single pick-and-place. Regrasps are necessary when the set of grasps reachable at the object's start pose does not intersect the set of grasps reachable at the object's goal pose [109]. Besides, with partially visible, novel objects, it is possible a plan with regrasps is more likely to succeed than a plan with just two steps. For example, if the only grasp reachable at both start and goal poses is likely unstable, it may be preferable to choose a plan with additional steps.

Definition 2.7 (Pick-and-place of a partially visible and novel object)**.** *The regrasping problem (Definition 2.6) constrained to a single* close *followed by a single* open *is a* pick-and-place problem of a partially visible, novel object*.*

2.3.1 Example

Suppose the task is to place a blue block long-end-up at the end of a line, as in Figure 2.2. At the block's initial pose, the only collision-free grasps are long-wise along the block, which are incompatible with the goal placement. Thus, the block is grasped long-wise, placed at a *temporary* (i.e., non-goal) location, regrasped, and then placed at the goal.

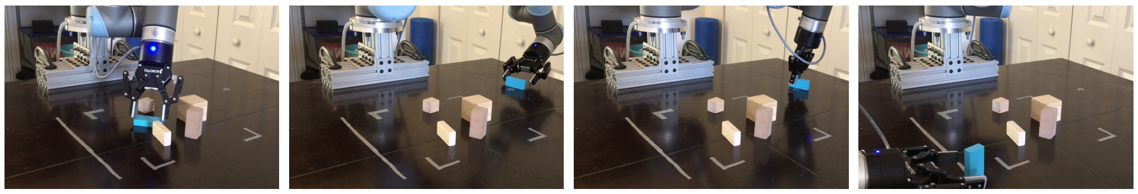


Figure 2.2: An example solution to the regrasping problem.

2.3.2 Extensions

The problem of Definition 2.6 is simplified for clarity of exposition. In the sequel, we encounter practical problems that are slightly more general.

Suppose the target object and goal pose are not given explicitly. For example, for the task "place a bottle upright onto a coaster", we only know the target object belongs to the "bottle" category and the goal pose satisfies the property "upright onto a coaster". The task specification is method-dependent: for modular architectures, separate components segment the object and plan goal placements; for policy-based methods, the task is specified via a reward function.

We also consider the case of multiple target objects, each with multiple goals, where any object-goal pair is a solution. This generalization has implications on system performance: this additional flexibility expands the space of regrasp plans and thus enables us to better maximize the probability the plan succeeds. For example, if the task is to place any bottle upright onto any coaster, we can choose to manipulate the bottle most visible in the point cloud.

2.3.3 The rearrangement problem

A more general problem is *rearrangement planning* or *manipulation planning*, where goals are specified for a set of target objects [120, 2, 1, 82, 81, 48]. This problem is more complex than regrasping since, as the robot completes the task, placed objects become obstacles [120]. The order of object placements is thus significant. For this thesis, we only consider rearrangement problems where the order of object placements is not important. In this case, the rearrangement problem breaks down into a sequence of independent regrasping problems. Krontiris and Bekris use pick-and-place as a component of a rearrangement planner [48].

Chapter 3

Approaches to Related Problems

In this chapter, we review approaches to problems related to pick-and-place of partially visible, novel objects. First is pick-and-place with full information. This is relevant because, as will be seen in Chapter 6, any of these solutions can be used for the planning component in a modular architecture. Second is pick-and-place of partially visible, known objects. Relaxing the assumption of full information gets us closer to the problem of interest. However, these solutions are not directly applicable to the novel-object case. Third is grasping novel objects and grasping objects with shape uncertainty. Since grasping is the first step in pick-and-place, ideas in this area are carefully carried over to pick-and-place (Chapters 4 and 6). Lastly, we review the most related topics – pick-and-place of novel objects and pick-and-place under uncertainty – to which we contribute.

3.1 Pick-and-Place in Fully Observed Environments

The structure of the regrasping problem was first explained by Tournassoud et al. [109]. There is a discrete search component, for sequencing grasp-place combinations, and a continuous search component, for connecting grasp-place combinations with a trajectory (i.e., a motion plan). Alami et al. generalized regrasping to multiple, movable objects and coined the term *manipulation planning* [2]. Wilfong showed this problem is PSPACE-hard [120]. Later, Alami et al. considered different cost functions for the discrete search phase, including path length and number of grasp changes [1]. Nielsen and Kavraki described a two-level planner for manipulation planning that interleaves execution between discrete search and continuous search levels [80]. This way, if a probabilistically complete motion planner gets stuck, the discrete search is able to continue. Stilman et al. considered a subset of manipulation planning problems called *monotone* problems, i.e., if a solution exists, it can be found by moving each object at most once [101]. The complexity of monotone problems, with the goal specified for one object, was estimated to be $O\left(m!(pe)^m\right)$, where m is the number of objects, p is the number of free placements, and e is the time required for motion planning [101]. Kaelbling and Lozano-Pérez proposed a hierarchical algorithm which integrates symbolic planning (with a hierarchical PDDL-like language) with low level pick-and-place actions [39]. This enables completion of high level, long term tasks such as, "Clean an object and put it into the storage room." Wan et al. employed a three-level planner, where the high-level planner provides a list of goal poses for the objects, the middle-level planner is a regrasp planner, and the low-level planner is a motion planner [113].

In Chapter 6, a regrasp planner like Tournassoud et al.'s is adapted with a cost function that accounts for perceptual uncertainty. But any of these methods could be applied in the same way.

3.2 Pick-and-Place of Partially Visible and Known Objects

Others have considered pick-and-place of imperfectly perceived objects with known shapes. One approach is to match object models to sensor data, as in Tremblay et al. [110]. However, model matching fails when there are significant differences between the perceived object and the model, as with novel objects [67].

3.3 Grasping Novel Objects

In the past decade, much work has been done on novel-object grasping. There are roughly four classes of approaches. The first class formulates the problem as binary classification: given a gripper pose, encoded by the point cloud in the vicinity of the gripper (or point features), label this as a successful grasp or not [95, 53, 21, 87, 65, 84, 59, 78]. These approaches have a grasp candidate sampling step followed by a classification step. The classification step relies on machine learning to generalize to novel objects. The second class of approaches aims to reduce computation time by avoiding the grasp candidate sampling step. Given an image or a point cloud, directly predict (or regress) gripper poses that are likely to result in a grasp [36, 75, 130]. The third class of approaches views grasping as a sequential, closed-loop problem, where images are taken and the grasp pose is adjusted as the gripper moves toward the objects, and is typically solved with reinforcement learning (RL) [55, 112, 41, 91]. This has the advantage of reactivity: if objects move while the gripper approaches, the gripper can move to compensate. The fourth class of approaches predicts the complete shape of an object and then plans grasps using the completed shape [111, 64, 63]. Shape completion is performed using a deep neural network, where the input and output are 3D voxel grids. Shape completion enables the use of grasp planning algorithms, which optimize for grasp stability given the input shape. Shape completion also enables the use of collision checking, to avoid collisions between the robot and (seen or unseen) object parts.

Since grasping is the first step in pick-and-place, methods for novel-object grasping inspire methods for novel-object pick-and-place. The third and fourth approach classes are particularly suitable for pick-and-place planning, as they immediately extend to acting over long time horizons. In Chapters 4 and 5, RL is applied to novel-object pick-and-place; in Chapter 6, shape completion is used.

3.4 Grasping with Object Shape Uncertainty

When an object is partially visible, the unobserved shape of the object can be thought of as being randomly distributed. In this case, the goal is to choose a grasp that is likely to be stable, no matter what the shape of the object turns out to be. Two approach classes are: (i) evaluating grasp stability over a Monte Carlo (MC) sampling of object shapes and (ii) evaluating a probabilistic model of grasp stability. For approach (i), an explicit distribution over shapes is not required: all that is needed is a mechanism for sampling shapes. For approach (ii), a simplified model of the distribution (e.g., a mixture of Gaussians) is fit to prior experience. If the number of MC samples is large, approach (i) tends to be more computationally expensive but more accurate than approach (ii) [66].

Christopoulos and Schrater may have been the first to consider grasping under object shape uncertainty [9]. They considered grasping planar objects, represented as closed, cubic splines. Probability of grasp success is estimated by evaluating force closure over a sample of gripper poses (with normally distributed error) and object shapes (from a given database of objects of the same category as the target object). Next, Kehoe et al. considered planar grasping with uncertain object shape, but uncertainty is represented as normally distributed polygonal vertices and center of mass with given means and variances [44, 43]. As with Christopoulos's work, MC sampling is used to estimate the probability of grasp success: the key difference is to consider a certain class of slip grasps which is more general than static force-closure analysis.

Soon afterward, Gaussian process implicit surfaces (GPISs) were proposed as a representation of object shape uncertainty [11, 12, 66, 51, 58]. GPISs combine multiple observations of the object's signed distance function (SDF) into a Gaussian process – a normal distribution over SDFs. Mahler et al. use GPISs for planar grasping of uncertain object shapes [66]. They formulate the problem as a non-convex optimization problem, Equations 3.1 to 3.4, where g is a grasp target, f is the GPIS, $\widetilde{P}_F$ is an estimation of the probability of force closure (given mean GPIS shape and Gaussian uncertainty in gripper pose), c_1 and c_2 are the contact points (given g and the mean shape), $\sigma^2(\cdot)$ is the variance of the GPIS at a query point, n_1 and n_2 are the surface normals at the contact points, v is the closing direction (given g), and $\lambda, \alpha, \beta > 0$ are given constants. Intuitively, the objective is to find a grasp maximizing probability of force closure and minimizing uncertainty at the contact points, and the constraints require grasps to be antipodal on the mean object shape. This is approximately solved using sequential convex programming with some number of random start grasps. They compare to baselines that do not account for the variance in object shape and show that uncertainty in object shape is important. Additionally, they compare to an MC method which has the best performance but the highest computational cost. Laskey et al. improve the efficiency of MC sampling from the GPIS by employing multi-armed bandit techniques to reduce the number of evaluations for grasps that are unlikely to succeed [51]. Li et al. use GPIS for closed-loop grasping of 3D objects; in their formulation, only variance at the grasp's contact points is considered in order to avoid the computational overhead of MC sampling [58].

$$
\begin{aligned}
&\underset{g \in \mathcal{G}}{\text{maximize}} \quad && \widetilde{P}_F(g, f) - \lambda\left(\sigma^2(c_1) + \sigma^2(c_2)\right) && (3.1)\\
&\text{subject to:} && \|n_1 + n_2\|_2^2 < \alpha && (3.2)\\
& && n_1^T v > \beta \|v\|_2^2 && (3.3)\\
& && -n_2^T v > \beta \|v\|_2^2 && (3.4)
\end{aligned}
$$

Later, advances in deep learning influenced grasping under shape uncertainty. Lundell et al. used a deep 3D convolutional neural network (CNN) to predict missing voxels, i.e., for shape completion. Uncertainty is represented with *dropout* – an MC sample is drawn by randomly dropping out nodes in the network [100]. Lundell et al. concluded that MC sampling improves grasp robustness, especially on novel objects.

In Chapter 6, both MC and probabilistic modeling approaches are extended to pick-and-place.

3.5 Pick-and-Place of Novel Objects

A few projects have considered novel-object pick-and-place. Both policy learning and modular approaches have been explored. All approaches have a machine learning component enabling generalization to new objects. Our contribution was (a) to enable efficient action sampling for value based policy learning, (b) to explicitly maximize the probability of successfully executing a regrasp plan, and (c) to compare different representations of perceptual uncertainty.

3.5.1 Policy learning approaches

Finn et al. demonstrated a policy learning approach to placing a novel object into a novel container after just one demonstration [14]. Their system is trained with a loss function designed to facilitate learning of new tasks with few demonstrations. However, their system assumes the object is already grasped.

We used deep RL to address 6-DoF pick-and-place of novel objects within a given category [20, 22, 23]. A neural network estimates a *value* – the expected sum of future rewards – for a query grasp or place. The robot moves to the highest-valued, reachable grasp or place found. To handle the large space of grasps and places, the action choices can be limited by a grasp detector [20] or hierarchical sampling [22, 23]. By managing the large action space, these methods are efficient at runtime. However, they require long training times. Details are in Chapters 4 and 5.

Recently, Wang et al. proposed a policy learning system similar to our hierarchical sampling approach [115]. They demonstrated building structures with blocks and showed that it compares favorably to an actor-critic method. Berscheid et al. demonstrated policy learning of 4-DoF pick-and-place [5]. The most important aspect of this work is that the placement goal is specified by an image of the desired placement. Their system predicts separate values for (i) the grasp, without knowing the place (ii) the place, without knowing the grasp, and (iii) the grasp-place combination. This enables efficient in-plane sampling of grasp-place combinations, as grasps or places with low, independent values can be removed from consideration.

3.5.2 Modular architectures

Jiang et al. were one of the earliest teams to use machine learning to address novel-object pick-and-place [35, 34]. Their approach is similar to many of the early novel-object grasping approaches in that there is a sampling step followed by a prediction step. The focus is on localizing placements that are likely to be stable and satisfy human preference. However, their experiments assume known grasps on the objects.

Manuelli et al. proposed a modular system for pick-and-place of novel objects within a given category [67]. Their system, shown in Figure 3.1, is a 4-component pipeline: (a) object instance segmentation, (b) 3D key point detection, (c) optimization-based planning for computing object displacement given task-specific costs and constraints, and (d) a grasp detector and execution of the object displacement. The focus of the project was the (b) and (c) components. Objects are minimally represented with key points, which are 3D points indicating task-relevant object parts, e.g., the top, bottom, and handle of a mug. An integral network [102] is used to localize key points in the RGBD image of an object instance. The pick-and-place task is specified by costs and constraints on the key points. For example, a point in the center of a mug's handle is constrained to be aligned with a peg on the rack. Their robot reliably placed shoes on a shelf and mugs on a rack. Later, Gao and Tedrake augmented this line of research with shape completion, which proved to be useful for avoiding collisions when planning arm motions with the held object [16]. However, this system does not explicitly consider the probability of successfully executing a pick-and-place.

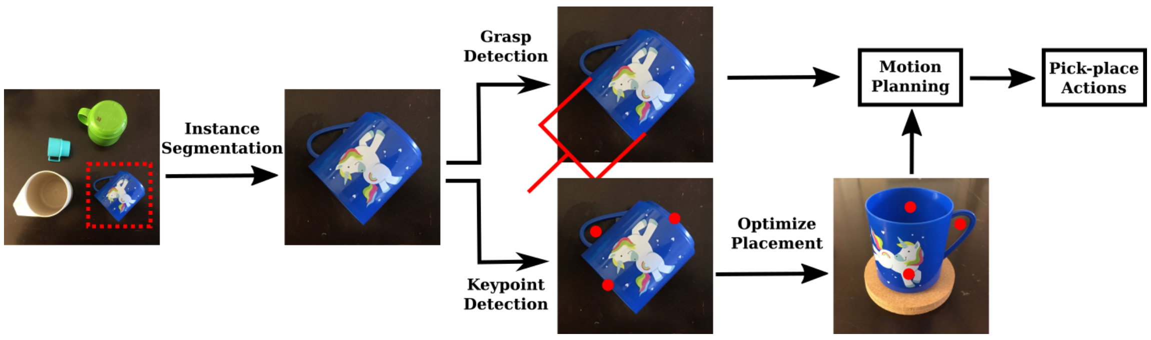


Figure 3.1: Diagram of Manuelli et al.'s system, called kPAM, adapted from [67].

Another modular approach was proposed by Mitash et al. [70]. They have (a) grasp detection (for either a vacuum or a parallel-jaw gripper), (b) regrasp planning with a single hand-off, and (c) additional views if regrasp planning fails (Figure 3.2). Objects are conservatively represented by both observed and occluded points. Regrasp planning decides the grasp, hand-off, and place. If the object is too large to be placed into the goal configuration (e.g., the occluded points make the object too large to fit into a narrow slot) additional views are taken until a plan is found. They concluded that (i) regrasp planning is important for avoiding errors from unnecessary views and hand-offs and (ii) including the unobserved region as part of the object helps avoid collisions when placing. While this system attempts to find robust placements, it does not consider uncertainty over the entire regrasp sequence and does not compare different ways of incorporating perceptual uncertainty.

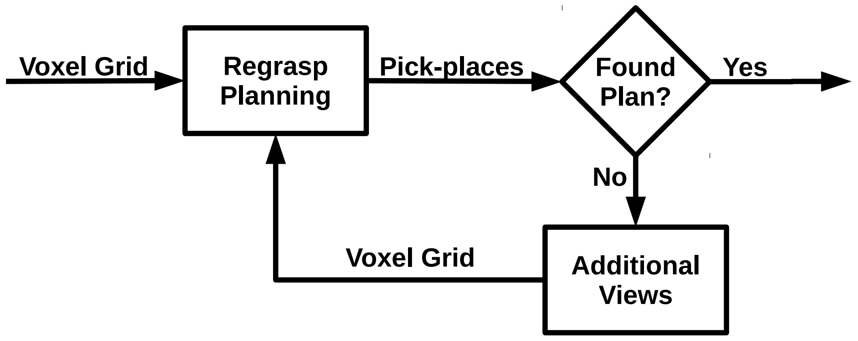


Figure 3.2: Rough outline of Mitash et al.'s system [70].

To address the limitations Manuelli et al. and Mitash et al.'s systems, we investigated a modular approach which explicitly reasons about the probability of successfully executing a regrasp plan, given that perception is uncertain. This system perceives the scene using object instance segmentation and shape completion and incorporates perceptual uncertainty into the regrasp planning cost. Details are in Chapter 6.

3.6 Pick-and-Place with Uncertainty

Research on pick-and-place with uncertainty has primarily assumed known objects in unknown poses, e.g., [39, 18, 125]. High-quality policies in these examples exhibit two types of behavior: (i) taking actions to improve confidence in the observations and (ii) avoiding actions with uncertain outcomes. A good policy for the case of uncertain object shapes should also exhibit these behaviors.

Part II

Policy Learning

The choice of a point in three-dimensional space may be considered to be a single-stage process wherein we choose (x, y, z), or a multi-stage process where we choose first x, then y, and then z.

Richard Bellman in *Dynamic Programming*

Chapter 4

Grasp Detection for Discretizing Pick Actions

In this chapter, we formulate the problem of pick-and-place of partially visible, novel objects as a Markov decision process (MDP) and solve it using reinforcement learning (RL).[1] Specifically, the robot learns an *action-value function* (or *Q-function*), which gives, for every observation-action pair, a number indicating the value of the action for performing the task. For example, if the task is to place a bottle upright onto a shelf, an action grasping the bottle from the bottom will have a value of 0, since it is not possible to place the bottle upright with this grasp, and an action grasping the bottle from the side will receive a value of 1, since this action is useful for completing the task. The robot learns the Q-function from hundreds of thousands of trial-and-error experiences in simulation, receiving rewards when a useful action is performed. After learning the action-value function, the robot's *policy* – a function mapping observations to actions – is to take the highest-valued, kinematically feasible action.

However, there is a critical issue learning a value function for pick-and-place. Since the action space is continuous, it is necessary to discretize the action choices before checking their values. Since the action space is high-dimensional, this discretization must concentrate on actions that are likely to be useful. While value-learning approaches like DQN can handle high-dimensional observations, they require a small number of actions in order for training and execution times to be reasonable [73].

We address this problem by using a grasp detector to produce a discrete sampling of pick actions which are all likely to be stable grasps. Not all stable grasps, though, are suitable for the pick-and-place task, as some goal placements are incompatible with some grasps. (For example, grasps on the bottom of a bottle are not compatible

[1]See Appendix A for background on MDPs and RL.

with placing the bottle upright.) The Q-function will indicate which of these grasps are suitable for the task.

Experiments demonstrate that the above approach generalizes well from training objects to held-out test (i.e., novel) objects and from simulation to the real world. Moreover, the RL approach consistently outperforms a model-based, shape primitives baseline which does not use machine learning for generalization. The main takeaway is that it is possible to automatically learn, from trial and error, a pick-and-place policy that can handle novel objects heavily occluded by clutter, and that prior knowledge of grasp detection is helpful to achieve this.

4.1 The Descriptor-Based MDP

In this section, we describe our MDP formulation of the pick-and-place problem. This MDP represents grasp actions similarly to how they were represented for novel-object grasping [21, 84]. We then describe how a Q-function is learned for this MDP. The main point is that grasp actions are discretized with grasp pose detection (GPD) [21, 84]. This way, the only pick actions available to the robot are those which are likely to result in stable grasps. After this, it is straightforward to use DQN to learn the Q-function [73]. We begin by explaining how GPD works.

4.1.1 Grasp pose detection

GPD follows the two steps in Algorithm 4.1 [21, 84]. Step 1 is to randomly sample gripper poses. The following heuristics are used to improve sample efficiency:

- Randomly sample a point in the cloud C, and set this as the gripper's position.
- For each sampled point, estimate the surface normal and axes of principle curvature. A method for estimating surface normals is given in [93], and a method for estimating axes of principle curvature is given in [104, 85]. Set the surface normal as the gripper's approach direction (the red arrow in Figure 4.1b), and set the axis of least curvature as the gripper's up direction (the green arrow in Figure 4.1b).
- Move the effector in the approach direction until contact is made with C.

Step 2 is to label these grasp candidates as antipodal or not, given local point cloud information. In particular, the cloud is transformed into the reference frame of the gripper, cropped to a predefined cuboid, and projected onto 2D images as

Algorithm 4.1: Grasp pose detection

```
Input  : A point cloud, $C \in \mathbb{R}^{n \times 3}$.
Output: A discrete set of gripper poses, $G \subset SE(3)$.
1 $G \leftarrow$ Sample($C$) // Sample gripper poses and generate descriptors.
2 $G \leftarrow$ Classify($G$) // Classify candidates as antipodal/not.
```

shown in Figure 4.1. These images are then fed into a convolutional neural network (CNN) which predicts the probability the grasp is antipodal (Definition 2.4). The set of accepted grasps G are those with predicted antipodal probability greater than a given threshold.

Training data for the grasp classification CNN is generated with either simulated or real point clouds, where both partial and complete views are available. Using a partial view of the object, grasp candidates are sampled and their corresponding descriptors are computed. Using the complete view of the object, the antipodal conditions in Definition 2.4 are checked to generate ground truth labels.

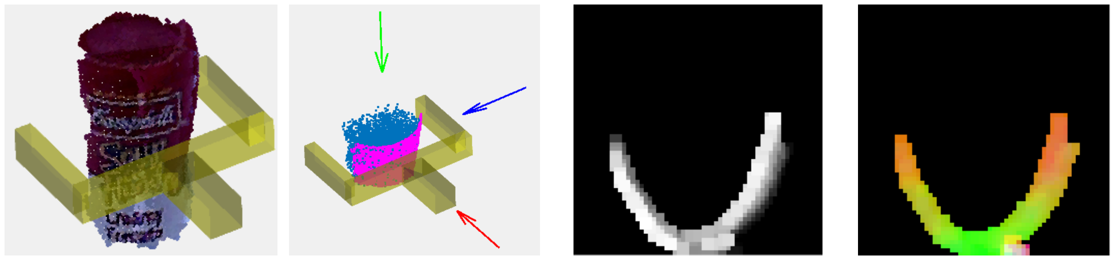
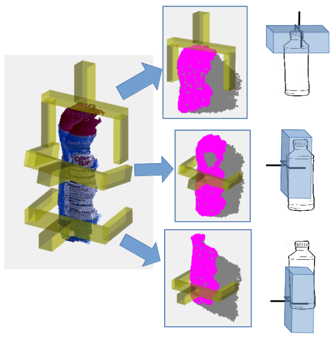


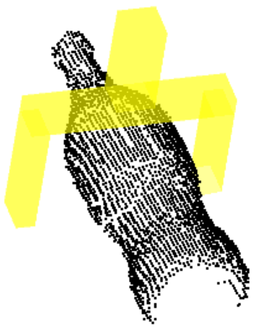
Figure 4.1: Descriptor for antipodal classification. (a) The pose of a candidate grasp, shown as a parallel-jaw gripper in yellow. (b) Images are projections of the points from these three directions. (c) Top-view height map of the observed points. (d) Top-view of unit-length surface normals: x, y, and z components correspond to red, green, blue. In total, 12 height maps are passed into the classification network: observed points ($\times 1$) and surface normals ($\times 3$) from 3 directions.

4.1.2 Pick descriptor

Similar to the GPD descriptor of the previous section – used to encode grasps for antipodal classification – the *pick descriptor* encodes grasp actions for the RL agent. Intuitively, the pick descriptor encodes the relative pose between the robot's gripper and an object in terms of the observed points in the vicinity of a candidate grasp.

Definition 4.1 (Pick descriptor). *Let Trans* $: SE(3) \times \mathbb{R}^{n\times 3} \rightarrow \mathbb{R}^{n\times 3}$ *denote the rigid transformation of a point cloud. Let Crop* $: \mathbb{R}^3_{>0} \times \mathbb{R}^{n\times 3} \rightarrow \mathbb{R}^{\bar{n}\times 3}$, *with* $\bar{n} \leq n$ *and arguments* $z \in \mathbb{R}^3_{>0}$ *and* $C \in \mathbb{R}^{n\times 3}$, *denote the function which removes all points in* C *besides those in the rectangular volume situated at the origin with length* $z(1)$, *width* $z(2)$, *and height* $z(3)$. *Let Proj* $: \mathbb{R}^{n\times 3} \rightarrow \mathbb{R}^{n_x\times n_y\times 12}$ *be a function which takes in a point cloud, estimates a surface normal for each point, and produces 12,* $n_x \times n_y$ *images, which are orthographic projections of the point cloud and surface normals from standard basis directions (Figure 4.2c). Then, given a point cloud* C, *a gripper pose* T, *and volume size* z, *the* pick descriptor *is* $I = \mathit{Proj}(\mathit{Crop}(z, \mathit{Trans}(T^{-1}, C)))$.[2]


(a)


(b)

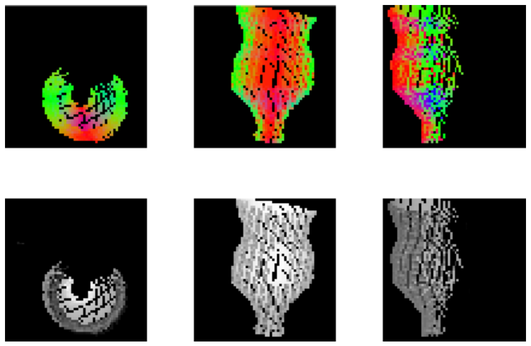

(c)

Figure 4.2: The pick descriptor. (a) Points are cropped to a fixed rectangular volume with respect to the gripper's reference frame. (c) Orthographic projections of the points around the bottle in (b). The top three images show x, y, and z components of unit length surface normals, and the bottom three images show height maps. In total, there are 12, 60×60 images per descriptor.

Compared to the grasp descriptor of the previous section, the cropped volume of the pick descriptor (specified by z) is larger. This is to capture information relevant to placing the object, e.g., object orientation in the gripper. For example, from Figure 4.2c, it is possible to recognize that the bottle is upside-down in the gripper. Otherwise, the pick descriptor is the same as the GPD descriptor.

4.1.3 Descriptor-based MDP

We formulate the pick-and-place problem as an MDP, which enables us to use standard RL algorithms, like DQN [73], to learn a pick-and-place policy. Grasp actions

[2]See Appendix B for implementations of *Trans*, *Crop*, and *Proj*.

are discretized using GPD and are represented by pick descriptors. Place actions are given, fixed gripper poses and are represented by 1-hot vectors.

Definition 4.2 (Descriptor-based MDP)**.** *Given a move-binary-effect system (Definition 2.1) and a set of goal poses for a target object* $\{T_1, T_2, \dots\} \subseteq SE(3)$*, a* descriptor-based MDP *is an MDP where:*

- ***Actions**: The set of actions available at the current time step, t, is* $\{I_{t,1}, I_{t,2}, \dots, I_{t,N_t}\} \cup \{1, 2, \dots, M\}$*, where* $\{I_{t,1}, I_{t,2}, \dots, I_{t,N_t}\}$ *is a finite set of* $N_t \in \{0, 1, \dots\}$ *pick descriptors (Definition 4.1) returned by GPD given the current point cloud* $C_t \in \mathbb{R}^{n_t \times 3}$*, and where* $\{1, 2, \dots, M\}$ *is a set of integers corresponding to M given, fixed place poses.*
- ***State**: The current state is the last* $k \in \{0, 1, 2\}$ *actions.*
- ***Transitions**: Selecting* $I_{t,i}$*, where* $i \in \{1, 2, \dots, N_t\}$*, moves the gripper to the pose used to generate* $I_{t,i}$ *and closes the gripper. Selecting* $j \in \{1, 2, \dots, M\}$ *moves the gripper to the corresponding place pose and opens the gripper. Then, objects move according to the dynamics of the move-binary-effect system (Definition 2.1), and the system acquires a point cloud,* $C_{t+1} \in \mathbb{R}^{n_{t+1} \times 3}$*. The next set of pick actions,* $\{I_{t+1,1}, I_{t+1,2}, \dots, I_{t+1,N_{t+1}}\}$*, is returned by GPD given* C_{t+1}*. The next set of place actions,* $\{1, 2, \dots, M\}$*, is unchanged.*
- ***Reward**: The reward is 1 whenever the pose of the target object is a member of the set* $\{T_1, T_2, \dots\}$ *and 0 otherwise.*

The process terminates after t_{max} *actions, after the target object is placed into a goal pose* $\{T_1, T_2, \dots\}$*, or after a grasp or place failure, whichever occurs first.*

The objective is for the robot to learn a policy maximizing the expected sum of future rewards. More specifically, we seek a sequence of actions maximizing the probability of placing the target object into a goal pose.

The transition probabilities of the descriptor-based MDP are not known explicitly. However, an exact description of the transition probabilities is not used by our learning algorithm. We assume a computer simulator of the move-binary-effect system. This way, the robot can learn what actions to take through trial-and-error experience without having to incur the cost of real-world trial-and-error learning. While it is not necessary to explicitly know the transition probabilities of the descriptor-based MDP, we do simulate the move-binary-effect system.

Transitions in the move-binary-effect system and the descriptor-based MDP appear stochastic to the robot. This is only because the objects are partially observed.

Thus, it would be straightforward to express the pick-and-place problem as a partially observable Markov decision process (POMDP) (Appendix A), where state includes the poses and shapes of the objects, and where observations are pick descriptors. One way to reduce this POMDP to an equivalent MDP is to store all past observations and actions in the MDP's state (Appendix A). Storing the past k actions approximations this reduction. This is the approach taken here.

Encoding grasps as pick descriptors enables efficient learning in two ways. First, the descriptor only shows the points in the vicinity of the grasp, and thus hopefully the robot only sees the object it is grasping. This way, the robot does not need to learn over all combinations of objects present. Second, the object is represented in the gripper's reference frame, i.e., in the reference frame of a grasp that is likely to be stable, rather than in an arbitrary reference frame. This way, the robot does not need to learn over arbitrary combinations of object poses. These two features enable efficient learning in the descriptor-based MDP.

4.1.4 Simulator

Deep RL requires such an enormous amount of experience that it is difficult to learn control policies on real robotic hardware without spending months or years in training [54, 55]. As a result, learning in simulation is basically a requirement. Fortunately, pick-and-place constraints allows us to simplify the simulation. Instead of simulating arbitrary contact interactions, we only simulate the effects of grasp and place actions. The former is simulated by evaluating standard grasp quality metrics, such as the antipodal condition (Definition 2.4). The latter is simulated by evaluating placement stability (Definition 2.5 for horizontal planes) and by checking whether or not a goal arrangement is satisfied. These are straight-forward to evaluate in OpenRAVE [10], the simulator used in the experiments for this chapter.

4.1.5 Action-value function

The Q-function is approximated using the CNN architecture depicted in Figure 4.3. The input includes an action to evaluate along with the current state. The output is a scalar, real value estimating the value of that state-action pair. This structure is slightly different than that used in DQN where the network has a number of outputs equal to the number of actions [73]. Since the descriptor-based MDP has a variable number of grasp actions, action must be an input to the network.

The action component of the input is comprised of the pick descriptor (a $60 \times 60 \times 12$ stacked image as in Figure 4.2c) and the place choice (a 43-element one-hot vector). When the robot takes a grasp action, the pick descriptor is populated

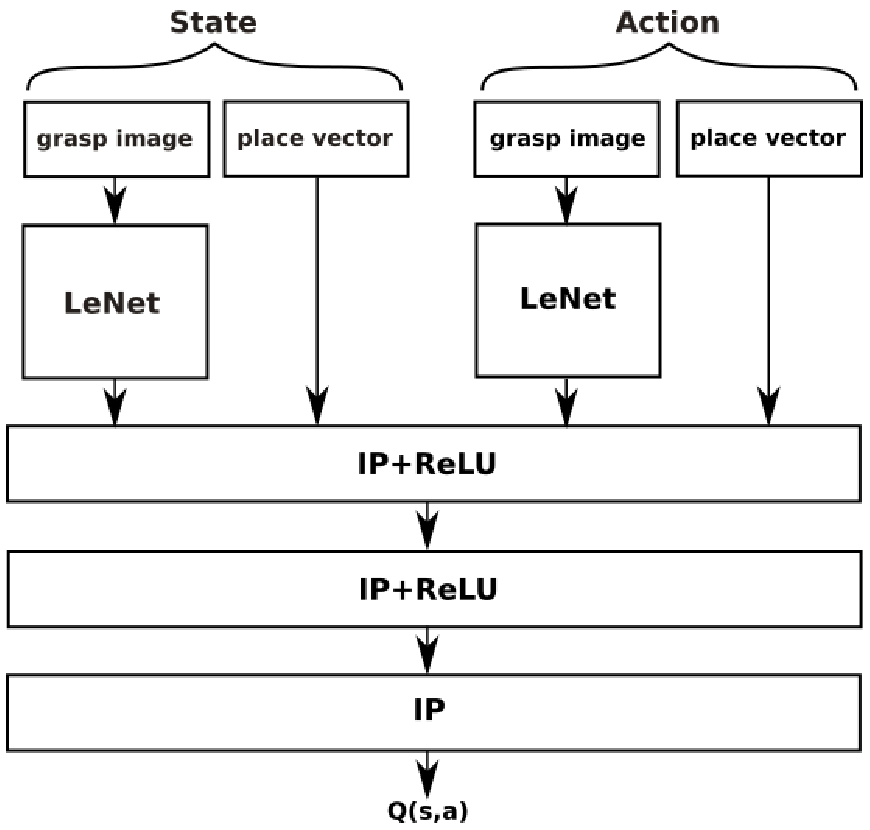


Figure 4.3: CNN architecture used to encode the Q-function.

and the place vector is set to zero. When the robot takes a place action, the pick descriptor is set to zero and the place vector is populated.

The state component of the input is also comprised of a pick descriptor and a place vector. These parameters encode the recent history of actions taken. After executing a grasp action, the pick descriptor is set to the previous action and the place vector is set to zero. After executing a place action, the pick descriptor retains the selected grasp and the place component is set to the just-executed place, thereby implicitly encoding the resulting pose of the object following the placement.

Each grasp image (in both action and state input branches) is processed by a CNN similar to LeNet [52], except there are 100 outputs instead of 10. These outputs, together with the place vector, are then concatenated and passed into two, 60-output inner product (IP) layers, each followed by rectifier linear units (ReLUs). After this, there is one more IP layer to produce the scalar output.

4.1.6 Learning algorithm

The algorithm for learning a Q-function for the descriptor-based MDP is shown in Algorithm 4.2. This is similar to DQN but with a couple of differences. First, Sarsa is used instead of Q-learning because the large action branching factor makes the $\max_{a \in A} Q(s, a)$ in Q-learning expensive to evaluate. Second, instead of running a step of stochastic gradient descent (SGD) after each experience, *nEpisodes* of experiences are collected before labeling the experience replay database using the most recent neural network weights. Every *nEpisodes* additional experiences, *nIterations* of SGD are run using Caffe [33]. For the experiments in this chapter, the learning algorithm is run only in simulation; although it could be used to fine-tune the network weights on the actual hardware.

Algorithm 4.2: Sarsa for the descriptor-based MDP

```
for $i \leftarrow 1, \ldots,$ nTrainingRounds do
    for $j \leftarrow 1, \ldots,$ nEpisodes do
        Choose random object(s) from the training set
        Place object(s) in a random configuration
        Sense point cloud $C$ and detect grasps $G$
        $s \leftarrow \mathbf{0}$
        $a \leftarrow$ Pick($s, C, G$) // Select grasp using $\epsilon$-greedy.
        for $t \leftarrow 1, \ldots, t_{max}$ do
            Execute $a$, run simulator forward, and observe $(r, s')$
            if a is a pick then
                $a' \leftarrow$ Place($s'$) // Select place using $\epsilon$-greedy.
            else if a is a temporary place then
                Sense point cloud $C$ and detect grasps $G$
                $a' \leftarrow$ Pick($s', C, G$) // Select grasp using $\epsilon$-greedy.
            else if a is a goal place then
                $a' \leftarrow$ null
            Add $(s, a, r, s', a')$ to database
            if $s'$ is terminal then break
            $a \leftarrow a'$; $s \leftarrow s'$
    Prune database if it is larger than maxExperiences
    Label each database entry $(s, a)$ with $r + \gamma Q(s', a')$
    Run Caffe for nIterations on database
```

4.2 Experiments in Simulation

We performed experiments in simulation to evaluate how well this approach performs on pick-and-place and regrasping problems with novel objects.[3] To do so, we obtained objects belonging to two different categories for experimentation: a set of 73 bottles and a set of 75 mugs – both in the form of mesh models from 3DNet [122]. Both object sets were partitioned into a 75%/25% train/test split.

4.2.1 Experimental scenarios

There were three different experimental scenarios: two-step-isolation, two-step-clutter, and multi-step-isolation. In *two-step-isolation*, an object was selected at random from the training set and placed in a random pose in isolation on a tabletop. The goal condition was a right-side-up placement in a particular position on the table. In this scenario, the robot was only allowed to execute one grasp action followed by one place action (hence the "two-step" label). *Two-step-clutter* was the same as two-step-isolation except a set of seven objects was selected at random from the same object category and placed in random poses on a tabletop as if they had been physically dumped onto the table (Figure 4.4).

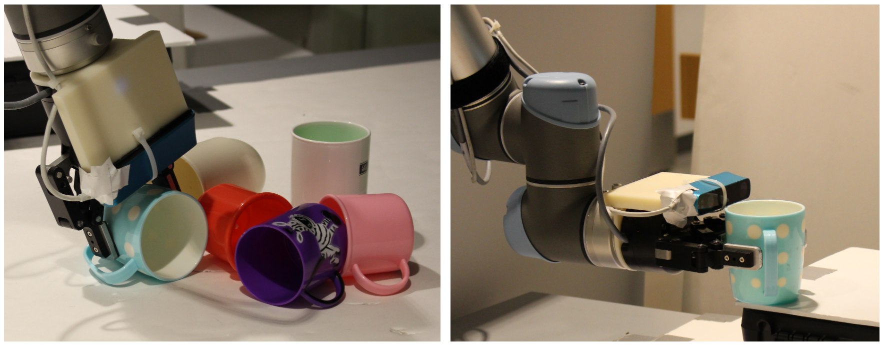


Figure 4.4: Example of the two-step-clutter scenario for mugs.

Multi-step-isolation was like two-step-isolation except multiple picks and places were allowed (i.e., regrasping) for up to 10 actions (i.e., $t_{\max} = 10$). Also, the goal condition was more restricted: the object needed to be placed upright, inside of a box rather than on a tabletop. Because the target pose was in a box, it became impossible to successfully reach it without grasping the object from the top before performing the final place (see Figure 4.8, bottom). Because the object could not

[3]Source code is available at github.com/mgualti/PickAndPlace.

always be grasped in the desired way initially, this additional constraint on the goal state sometimes forced the system to perform a regrasp in order to achieve the desired pose.

In all scenarios, point clouds were registered composites of two clouds taken from views above the object and 90° apart: a single point cloud performs worse, presumably because features relevant for determining object pose are unobserved. In simulation, we assumed picks always succeed, because the grasp detector was already trained to recognize stable grasps with high probability [21, 84].[4] A place was considered successful only if the bottom of the object was placed within 3 cm of the table and the vertical axis was within 20° of the desired pose.

4.2.2 Algorithm variations

Algorithm 4.2 was parameterized as follows. We used 70 training rounds (*nTrainingRounds* = 70) for the two-step scenarios and 150 for the multi-step scenario. We used $1,000$ episodes per training round (*nEpisodes* $= 1,000$). For each training round we updated the CNN with $5,000$ iterations of SGD with a batch size of 32. *maxExperiences* was $25,000$ for the two-step scenarios and $50,000$ for the multi-step scenario. For each episode, bottles were randomly scaled in height between 10 and 20 cm. Mugs were randomly scaled in height between 6 and 12 cm. We linearly decreased the exploration factor ϵ from 100% down to 10% over the first 18 training rounds.

We compared the performance of Algorithm 4.2 on two different types of pick descriptors. In the *standard* variation, we used descriptors of the standard size ($10 \times 10 \times 20$ cm). In the *large-volume* (LV) variation, we used descriptors evaluated over a larger volume ($20 \times 20 \times 40$ cm) but with the same image resolution.

We also compared with two baselines. The first was the *random baseline*, where grasp and place actions were chosen uniformly at random. The second was the *shape primitives baseline*, where object pose was approximated by segmenting the point cloud and fitting a cylinder. Although it is generally challenging to fit a shape when the precise geometry of the object to be grasped is unknown, we hypothesized that it could be possible to obtain good pick-and-place success rates by fitting a cylinder and using simple heuristics to decide which end should be up. We implemented this as follows. First, we segment the scene into k clusters, using k-means ($k = 1$ for isolation and $k = 7$ for clutter). Then we fit a cylinder to the most isolated cluster using MLESAC [108]. We select the grasp most closely aligned with and nearest to

[4]It is possible to train grasping from the same reward signal, but this would require longer simulations. Empirically, this assumption did not lead to many grasp failures on the real robot (Section 4.3).

the center of the fitted cylinder. The height of the placement action is determined based on the length of the fitted cylinder. The grasp up direction is chosen to be aligned with the cylinder half which contains fewer points. In order to get the shape primitive baseline to work, we had to remove points on the table plane from the point cloud. Although our learning methods do not require this and work nearly as well either way, we removed the table plane in all simulation experiments for consistency.

4.2.3 Results for the two-step scenarios

Figure 4.5 shows learning curves for the two-step-isolation and two-step-clutter contingencies for bottles (left) and mugs (right) averaged over 10 runs. Table 4.1 shows place success rates when the test objects were used.

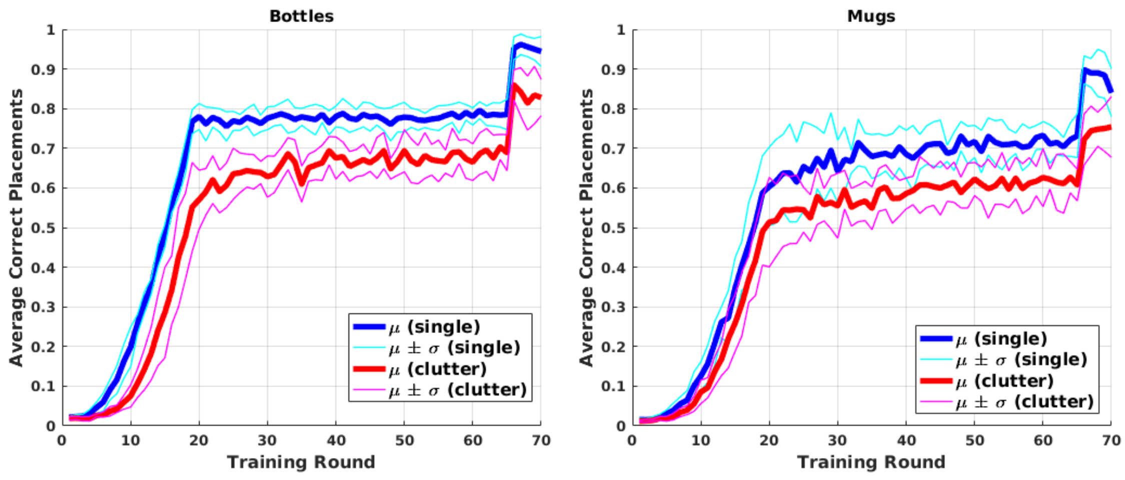


Figure 4.5: Average of 10 learning curves for the two-step scenario. The "training round" on the horizontal axis denotes the number of times Caffe had been called for a round of 5,000 SGD iterations. The **left** plot is for bottles and the **right** plot is for mugs. Blue denotes single objects and red denotes clutter. Curves for mean plus and minus standard deviation are shown in lighter colors. The sharp increase in performance during the last five rounds in each graph is caused by dropping the exploration factor ϵ from 10% to 0% during these rounds.

Several results are worth highlighting. First, our algorithm does very well with respect to the baselines. The random baseline (last row in Table 4.1) succeeds only 2% of the time – suggesting that the problem is indeed challenging. The shape primitives baseline (where we localize objects by fitting cylinders) also does relatively poorly: it succeeds at most only 43% of the time for bottles and only 12% of the

Trained With / Tested With	Bottle in Isolation	Bottles in Clutter
Isolation	1.00	0.67
Clutter	0.78	0.87
Isolation LV	0.99	0.47
Clutter LV	0.96	0.80
Shape Primitives Baseline	0.43	0.24
Random Baseline	0.02	0.02

Trained With / Tested With	Mug in Isolation	Mugs in Clutter
Isolation	0.84	0.60
Clutter	0.74	0.75
Isolation LV	0.91	0.40
Clutter LV	0.67	0.70
Shape Primitives Baseline	0.08	0.12
Random Baseline	0.02	0.02

Table 4.1: Average correct placements over 300 episodes for bottles (**top**) and mugs (**bottom**) using test set, after training.

time for mugs. Second, place success rates are lower when objects are presented in clutter compared to isolation: 100% success versus 87% success rates for bottles; 84% versus 75% success for mugs. Also, if evaluation is to be in clutter (respectively isolation), then it helps to train in clutter (respectively isolation) as well: if trained only in isolation, then clutter success rates for bottles drops from 87% to 67%; clutter success rates for mugs drops from 75% to 60%. Also, using the LV descriptor can improve success rates in isolation (an increase of 84% to 91% for mugs), but hurts when evaluated in clutter: a decrease from 87% to 80% for bottles; a decrease from 75% to 70% for mugs. We suspect this drop in performance reflects the fact that in clutter, the large receptive field of the LV descriptor encompasses "distracting" information created by other objects nearby the target object [72].

4.2.4 Results for the multi-step scenario

Training for the multi-step-isolation scenario was the same as in the two-step scenario except we increased the number of training rounds and the capacity of the experience replay database because the longer policies took longer to learn. We only performed this experiment using mugs because GPD did not find many grasps on the tops of bottles. Figure 4.6 shows the number of successful non-goal and goal placements as a function of training round.[5] Initially, the system does not make much use of its ability to perform intermediate placements in order to achieve the desired goal placement, i.e., to pick up the mug, put it down, and then pick it up a second time in a different way. This is evidenced by the low values for non-goal placements (the blue line) prior to round 60. However, after round 60, the system learns the value of the non-goal placement, thereby enabling it to increase its final placement success rate to is maximum value (around 90%). Essentially, the agent learns to perform a non-goal placement when the mug cannot immediately be grasped from the top or if the orientation of the mug cannot be determined from the sensor perception. After learning is complete, we obtain an 84% pick-and-place success rate averaged over 300 test set trials.

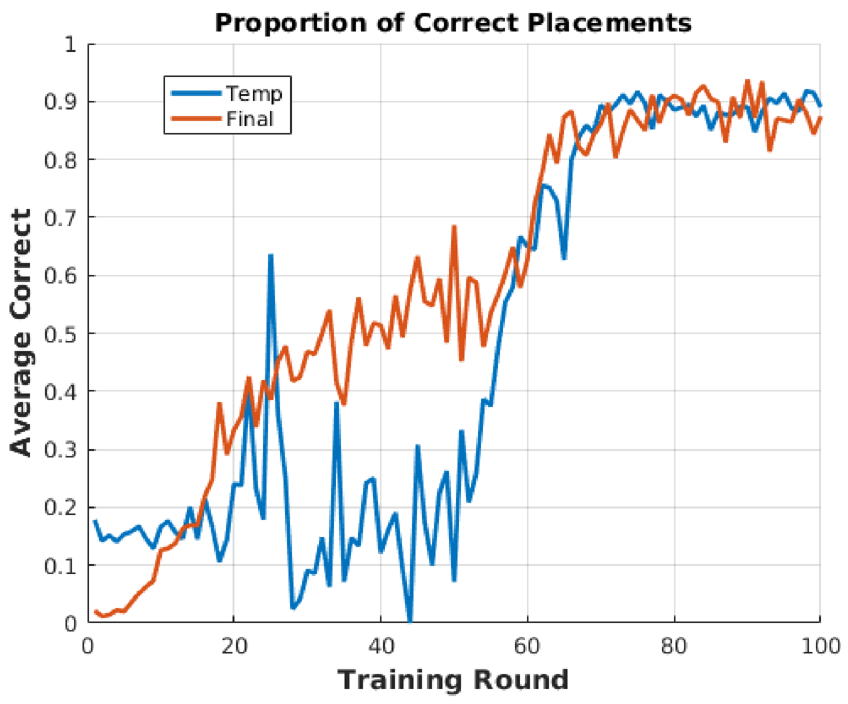


Figure 4.6: One multi-step realization with mugs in isolation. Red line: number of successful pick-and-place trials as a function of training round. Blue line: number of successful non-goal placements executed.

[5]Non-goal placements were considered successful if the object was 3 cm or less above the table. Any orientation was allowed. Unsuccessful non-goal placements terminate the episode.

4.3 Experiments on a Real Robot

We evaluated the same three scenarios on a real robot: two-step-isolation, two-step-clutter, and multi-step-isolation. As before, the two step scenarios were evaluated for both bottles and mugs, and the multi-step scenario was evaluated for only mugs. All training was done in simulation, and fixed CNN weights were used on the real robot.

The experiments were performed by a UR5 robot with 6 DOFs, equipped with a Robotiq parallel-jaw gripper and a wrist-mounted Structure depth sensor (Figure 4.8). Two sensor views were always taken from fixed poses, 90° apart. The object set included 7 bottles and 6 mugs, as shown in Figure 4.7. We used only objects that fit into the gripper, would not shatter when dropped, and had a non-reflective surface visible to our depth sensor. Some of the lighter bottles were partially filled so small disturbances (e.g., sticking to fingers) would not cause a failure. Figure 4.8 shows several examples of our two-step scenario for bottles presented in clutter.

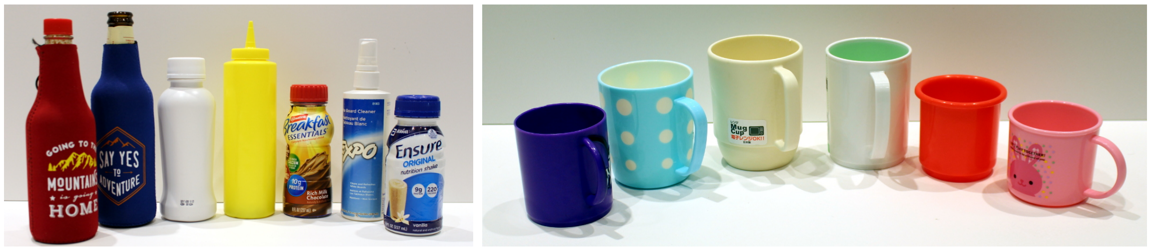


Figure 4.7: The seven novel bottles and six novel mugs used to evaluate our approach in the robot experiments.

Unlike in simulation, the UR5 requires an inverse kinematics (IK) solution and motion plan for any grasp or place pose it plans to reach to. For grasps, GPD returns many grasp choices. We sort these by their pick-and-place Q-values in descending order and select the first reachable grasp. For places, the horizontal position on the shelf and orientation about the vertical (gravity) axis do not affect object uprightness or the height of the object. Thus, these variables were chosen to suit reachability.

After testing some trials on the UR5, we found we needed to adjust a couple of training/simulation parameters. First, we changed the conditions for a successful place in simulation because, during our initial experiments, we found the policy sometimes selected placements that caused the objects to fall over. As a result, we adjusted the maximum place height in simulation from 3 cm to 2 cm and changed the reward function to fall off exponentially from +1 for altitudes higher than 2 cm.

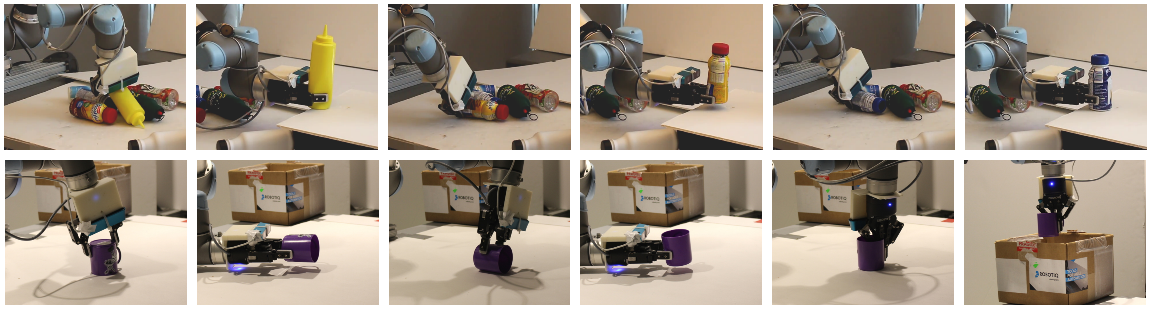


Figure 4.8: **Top**. Two-step-clutter scenario for bottles. First three objects are placed right-side-up and without falling over. **Bottom**. Multi-step-isolation scenario for a mug. The mug is initially upside-down, so must be flipped around before it can be put upright into the box.

Second, we raised the acceptance threshold used by GPD [21, 84].[6]

Table 4.2 summarizes the results from our robot experiments. We performed 483 pick-and-place trials over five different scenarios. Column one of Table 4.2 shows results for pick-and-place for a single bottle presented in isolation averaged over all bottles in the seven-bottle set. Out of 112 trials, 99% of the grasps were successful and 98% of the placements were successful, resulting in a complete task pick/place success rate of 97%. Column two shows similar results for the bottles-in-clutter scenario, and columns three and four include results for the same experiments with mugs. Finally, column five summarizes results from the multi-step-isolation scenario for mugs: overall, our method succeeded in placing the mug upright into the box 68% of the time. The temporary place success is perfect because a temporary placement only fails if the mug is so high it rolls away after dropped or too low it is pushed into the table, neither of which ever happened after 72 trials. The final placement is perfect because it always did get the orientation right (for all 72 trials that got far enough to reach the final placement), and it is hard for the mug to fall over in the box. The multi-step scenario has low task success rate because 12 trials failed to perform the final place after 10 time steps. Perhaps this is due to lower Q-function values on the real system (due to domain transfer issues), causing the robot to never become confident enough with its given state information to perform the final place.

Our experimental results are interesting for several reasons beyond demonstrating that the method can work. First, we noticed consistently lower place performance for the mug category relative to the bottle category. The reason for this is there is more

[6]GPD outputs a machine-learned probability of an antipodal grasp. The *threshold* is the grasp antipodal probability above which grasps are accepted.

	1 Bottle	7 Bottles	1 Mug	6 Mugs	Regrasp
Grasp	0.99	0.97	0.96	0.93	0.94
Final Placement	0.98	0.94	0.93	0.87	1.00
Temporary Placement	-	-	-	-	1.00
Entire Task	0.97	0.92	0.90	0.80	0.68
n Trials	112	107	96	96	72
Upside Down	0	4	5	10	0
Sideways	0	0	0	2	0
Fell Over	2	2	1	0	0
$t > 10$	-	-	-	-	12

Table 4.2: **Top**. Success rates for grasp, temporary place, final place, and entire task. **Bottom**. Placement error counts by type. Results are averaged over the number of trials (middle).

perceptual ambiguity involved in determining the orientation of a mug compared to that of a bottle. In order to decide which end of a mug is "up", it is necessary for the sensor to view into at least one end of the mug. Second, the robot had trouble completing the multi-step task in a reasonable number of steps with the real hardware compared with simulation. This may be because fewer grasps are available on the real robot versus the simulated robot due to collision modeling. Another unexpected result was our learned policies typically prefer particular types of grasps, e.g., to grasp bottles near the bottom; e.g., see Figure 4.8 (top). We suspect this is a result of the link between the location of a selected grasp and the pick descriptor used to represent state. In order to increase the likelihood that the agent will make high-reward decisions in the future, it selects a pick descriptor that enables it to easily determine the pose of the object. In the case of bottles, descriptors near the base of the bottle best enable it to determine which end is "up".

4.4 Discussion

To summarize, the problem of sampling 6-DoF gripper poses for grasps for pick-and-place tasks, with partially visible and novel objects, was addressed by combining grasp detection with Q-function learning. Grasp detection finds a small, discrete set of grasps that are likely to be stable, and the learned Q-function indicates which of these grasps are useful for the task.

4.4.1 Limitations

The descriptor-based MDP introduces additional partial observability into the problem because, instead of seeing the entire point cloud, the robot only sees parts of the point cloud in the vicinity of a gripper pose. The advantage of this is that learning is focused on the most task-relevant part of the scene – the robot's gripper. The disadvantage is that, to perform some tasks, an arbitrarily long history of pick descriptors is required. For example, if the task is to pack a bin as densely as possible, the robot must remember all previous placements in order to choose placements that will make the packing dense. Further, the robot would have trouble making good long-term packing decisions, because it sees only one unpacked object at a time.

Another limitation is fixed place choices. Tasks such as placing bottles onto coasters, where the coaster can appear in arbitrary positions on the table, require adaptive place choices. One approach to this is to learn a place detector for discretizing place choices, analogous to how GPD is used to discretize grasp choices. However, it would be convenient to not have to rely on task-specific heuristics for sampling grasps and placements. While very powerful for grasping, we already got into trouble with the grasp sampling heuristics, where grasps were not found on the tops of bottles, which happened to be the only grasps useful for the multi-step scenario. This problem is addressed in the next chapter.

4.4.2 Related work

This chapter is based on findings published in [20]. To our knowledge, this was the first successful demonstration of policy learning for pick-and-place and regrasping of novel objects. Previous policy learning methods considered only grasping [54, 112], placing [14], or the same objects used for training [56, 15, 55, 3]. With later methods, the problem of selecting high-dimensional pick-and-place actions was addressed by breaking down gripper pose selection into steps [22, 23, 115].

The action representation of the descriptor-based MDP is considered a *deictic* representation because the actions are described with respect to the effector's reference frame [88]. Platt et al. later explained how this representation results in faster, more generalizable learning [88].

Harada et al. successfully demonstrated picking fruit using shape primitives [26]. In particular, they used cylinder fitting to pick bananas from a cluttered bin.

Chapter 5

Hierarchical Spatial Attention

In Chapter 4, grasp pose detection (GPD) limits the grasp choices for pick-and-place, enabling more efficient policy learning. However, this does not immediately extend to limiting place choices, as it relies on grasp-specific heuristics. In this chapter, we describe a unified approach to limiting both grasp and place choices by learning to focus on areas containing actions relevant to the task. For instance, if the task is to place bottles upright onto coasters, the robot learns to focus on a bottle during the grasping stage and on a coaster during the placement stage. This is inspired by biological visual attention.

Visual attention has long been suggested to improve efficiency of policy learning. Focused perceptions can ignore irrelevant details, and generalization is improved by the elimination of the many irrelevant combinations of object arrangements [119]. Additionally, as we later show, attention can result in a substantial reduction to the number of actions that need considered. Indeed, when selecting position, the number of action choices can become logarithmic rather than linear in the volume of the robot's workspace. But there is a catch. Visual attention adds two new challenges: (i) the burden of learning where to attend and (ii) partial observability caused by the narrowed focus.

We address challenge (i) – efficiently learning where to attend – by constraining the system to a spatial hierarchy of attention. This means the robot must first see a large part of the scene in low detail, select a position within that observation, and see the next observation in more detail at the position previously selected, and so on for a fixed number of gazes. We address challenge (ii) – partial observability induced by the narrowed focus – by identifying attention with a type of state abstraction which preserves the ability to learn optimal policies with efficient reinforcement learning (RL) algorithms. With attention, we solve more difficult instances of pick-and-place of partially visible, novel objects than was previously possible with value learning methods.

5.1 Problem Statement

The problem is learning to control a move-effect system (Figure 5.1, left). This is similar to the move-binary-effect system of Chapter 2 but without the arm:

Definition 5.1 (Move-effect system)**.** *A* move-effect system *is a discrete time system consisting of a robot, equipped with a depth sensor and an effector, and rigid objects of various shapes and configurations. The robot perceives a history of point clouds* $C_1, \dots, C_k$*, where* $C_i \in \mathbb{R}^{n_c \times 3}$ *is acquired by the depth sensor; an effector status,* $h \in \{1, \dots, n_h\}$*; and a reward* $r \in \mathbb{R}$*. The robot's action is move-effect*(T_e, o)*, where* $T_e \in W$ *is the pose of the effector,* $W \subseteq SE(3)$ *is the robot's workspace, and* $o \in \{1, \dots, n_o\}$ *is a preprogrammed controller for the effector. For each stage* $t = 1, \dots, t_{max}$*, the robot receives a new perception and takes an action.*

The reward is instrumented by the system engineer to indicate progress toward completion of some desired task. The robot initially has no knowledge of the system's state transition dynamics. The objective is, by experiencing a sequence of episodes, for the robot to learn a policy – a mapping from observations to actions – which maximizes the expected sum of per-episode rewards.

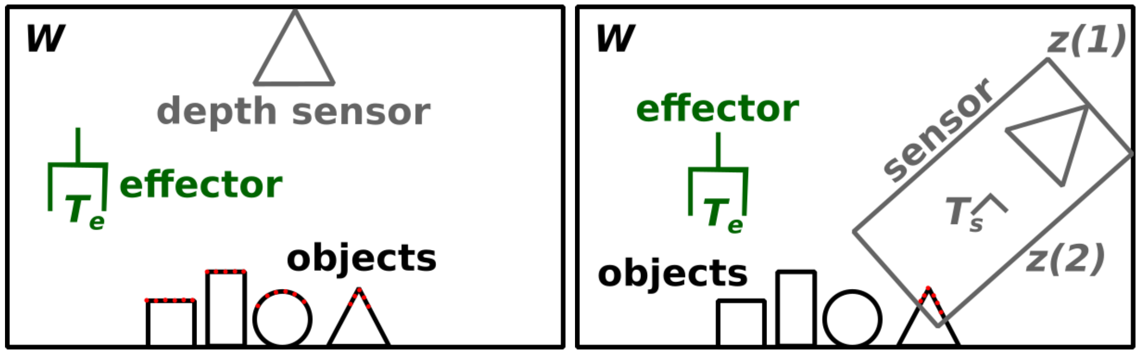


Figure 5.1: **Left**. The move-effect system. The robot has an effector which can be moved to pose T_e to perform operation o. **Right**. The sense-move-effect system adds a virtual, mobile sensor which observes points in a rectangular volume at pose T_s with size z.

For example, suppose the effector is a 2-fingered gripper, $o \in \{open, close\}$, $h \in \{empty, holding\}$, the objects are bottles and coasters, and the task is to place all the bottles on the coasters. The reward could be 1 for placing a bottle on a coaster, -1 for removing a placed bottle, and 0 otherwise.

5.2 Approach

Our approach has two parts. The first part is to reformulate the problem as a Markov decision process (MDP) with abstract states and actions. With this reformulation, the resulting state representation is substantially reduced, and it becomes possible for the robot to learn to restrict attention to task-relevant parts of the scene. The second part is to add constraints to the actions so the effector pose is decided sequentially. After these improvements, the problem is amenable to solution via standard RL algorithms like DQN. (Appendix A has more on MDPs and RL.)

5.2.1 Sense-move-effect MDP

The sense-move-effect system adds a controllable, virtual sensor which perceives a portion of the point cloud from a parameterizable perspective (Figure 5.1, right).

Definition 5.2 (Sense-move-effect system)**.** *A* sense-move-effect system *is a move-effect system where the robot's actions are augmented with sense*(T_s, z) *(where* $T_s \in W$ *and* $z \in \mathbb{R}^3_{>0}$*) and the point cloud observations* $C_1, \ldots, C_k$ *are replaced with a history of k images,* $I_1, \ldots, I_k$ *(where* $I \in \mathbb{R}^{n_{ch} \times n_x \times n_y}$*). The sense action has the effect of adding* $I = \mathit{Proj}(\mathit{Crop}(\mathit{Trans}(T_s^{-1}, C_k), z))$ *to the history.*[1]

The *sense* action makes it possible for the robot to get either a compact overview of the scene or to attend to a small part of the scene in detail. Since the resolution of the images is fixed, large values of z correspond to seeing more objects in less detail, and small values of z correspond to seeing less objects in more detail.

The robot's memory need not include the last k images – it can include any previous k images selected according to a predetermined strategy. Because the environment only changes after *move-effect* actions, we keep the latest image, I_k, and the last $k-1$ images that appeared just before *move-effect* actions. Figure 5.2 shows an example in the bottles on coasters domain.

[1] $\mathit{Proj} : \mathbb{R}^{n_c \times 3} \to \mathbb{R}^{n_{ch} \times n_x \times n_y}$ is n_{ch} orthographic projections of points onto n_{ch}, $n_x \times n_y$ images. Each image plane is positioned at the origin with a different orientation. Image values are the point to plane distance, ambiguities resolved with the nearest distance. $\mathit{Crop} : \mathbb{R}^{n_c \times 3} \to \mathbb{R}^{n_{c'} \times 3}$ returns the $n_{c'} \le n_c$ points of C which lie inside a rectangular volume situated at the origin with length, width, height z. $\mathit{Trans}(T_s^{-1}, C)$ expresses C (initially expressed w.r.t. the world frame) w.r.t. T_s.

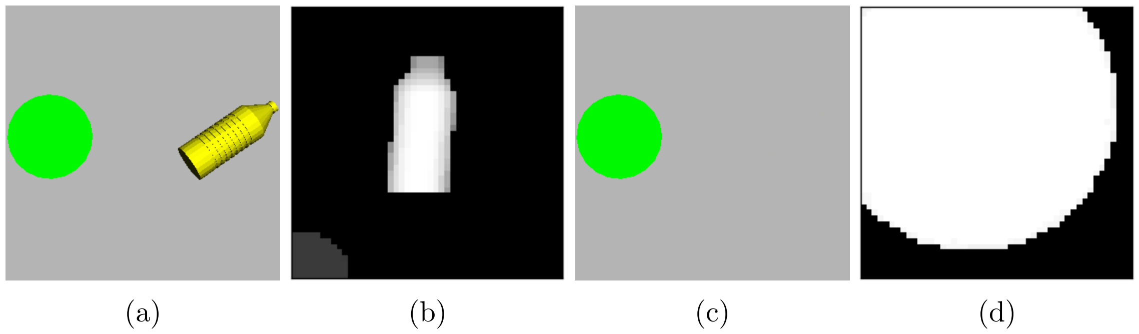


Figure 5.2: Scene and observed images for $k = 2$ and $n_{ch} = 1$. **(a)** Scene's initial appearance. **(b)** *sense* image (large z) just before *move-effect*(T_e, *close*). **(c)** Scene's current appearance. **(d)** Current *sense* image, focused on the coaster (small z). The robot remembers how the object was grasped (b) as it is adjusting the pose of the effector for the placement (d).

In order to apply standard RL algorithms to the problem of learning to control a sense-move-effect system, we define the sense-move-effect MDP. (See Appendix A for more on MDPs and standard RL algorithms.)

Definition 5.3 (Sense-move-effect MDP)**.** *Given a sense-move-effect system, a reward function, and transition dynamics, a* sense-move-effect MDP *is a finite horizon MDP where states are sense-move-effect system observations and actions are sense-move-effect system actions.*

The reward function and transition details are task and domain specific, respectively, examples of which are given in Section 5.3.

5.2.2 Hierarchical spatial attention

The observation is now similar to that of DQN – a short history of images plus the effector status – and can be used by a Q-network to approximate Q-values. However, the action space remains large due to the 6-DoF choice for T_s or T_e and the 3-DoF choice for z. Additionally, it may take a long time for the robot to learn which *sense* actions result in useful observations. To remedy both issues, we design constraints to the sense-move-effect actions.

Definition 5.4 (Hierarchical spatial attention)**.** *Given a sense-move-effect system, $L \in \mathbb{N}_{>0}$, $T_s^1 \in W$, and the list of pairs $[(z_1, d_1), \ldots, (z_L, d_L)]$, (where $z_i \in \mathbb{R}^3_{>0}$ and $d_i \in \mathbb{R}^6$),* hierarchical spatial attention (HSA) *constrains the robot to take L*

sense(T_s, z) *actions, with* $z = z_i$ *for* $i = 1, \ldots, L$, *prior to each move-effect action. Furthermore, the first sensor pose in this sequence must be* T_s^1*; the sensor poses* T_s^{i+1}, *for* $i = 1, \ldots, L-1$, *must be offset no more than* d_i *from* T_s^i*; and effector pose* T_e *must be offset no more than* d_L *of* T_s^L.[2]

The process is thus divided into t_{max} *overt stages*, where, for each stage, L *sense* actions are followed by 1 *move-effect* action (Figure 5.3). The constraints should be set such that the observation size z_i and offset d_i decrease as i increases, so the point cloud under observation decreases in size, and the volume within which the effector pose can be selected is also decreasing. These constraints are called hierarchical spatial attention because the robot is forced to learn to attend to a small part of the scene (e.g., Figure 5.4).

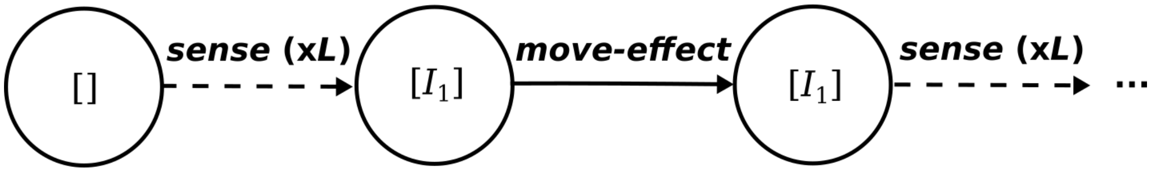


Figure 5.3: Initially, the state is empty. Then, L sense actions are taken, at each point the latest image is state. After this, the robot takes 1 *move-effect* action. Then, the process repeats, but with the last image before *move-effect* saved to memory.

To see how HSA can improve action sample efficiency, consider the problem of selecting position in a 3D volume. Let α be the largest volume allowed per sample. With naive sampling, the required number of samples n_s is proportional to the workspace volume $v_0 = d_1(1)d_1(2)d_1(3)$, i.e., $n_s = \lceil v_0/\alpha \rceil$. But with HSA, we select position sequentially, by say, halving the volume size in each direction at each step, i.e., $d_{i+1} = 0.5d_i$. In this case $8L$ samples are needed, i.e., a sample for each octant at each step. The volume represented by each sample at step i, for $i = 1, \ldots, L$, is $v_i = v_0/8^i$. To get $v_L \leq \alpha$, i.e., to get the volume represented by samples used for selecting effector position to be no more than α, $L = \lceil \log_8(v_0/\alpha) \rceil$. Thus, with HSA, the sample complexity becomes logarithmic, rather than linear, in v_0.

5.2.3 Lookahead sense-move-effect

So far we have not specified how action parameters T_s, T_e, and z are encoded. For *standard sense-move-effect*, these are 6 floating point numbers representing the pose

[2]Concretely, $d_i = [x, y, z, \theta, \phi, \rho]$ indicates a position offset of $\pm x/2$, $\pm y/2$, and $\pm z/2$ and a rotation offset of $\pm\theta/2$, $\pm\phi/2$, and $\pm\rho/2$.

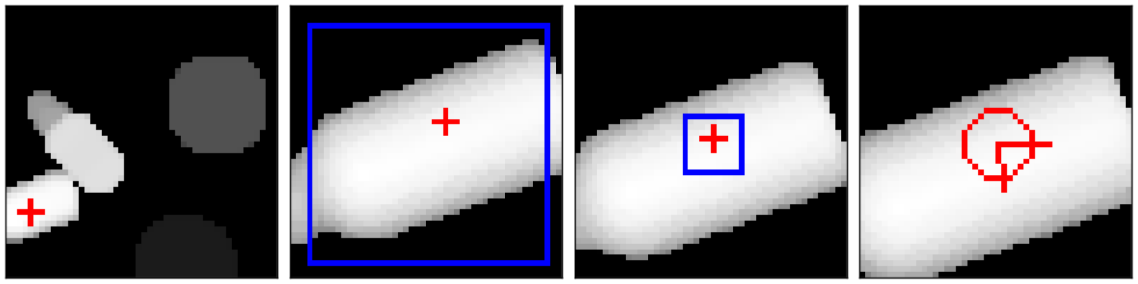


Figure 5.4: HSA applied to grasping in the bottles on coasters domain (Section 5.3.3). There are 4 levels (i.e. $L = 4$). The sensor's volume size z is $36 \times 36 \times 23.75$ cm for level 1, $10.5 \times 10.5 \times 47.5$ cm for levels 2 and 3, and $9 \times 9 \times 47.5$ cm for level 4. As indicated by blue squares, d constrains position in the range of $\pm 18 \times 18 \times 6.875$ cm for level 1, ± 4.5 cm^3 for level 2, and ± 1.125 cm^3 for level 3. Orientation is selected for level 4 in the range of $\pm 90°$ about the hand approach axis. Red crosses indicate the x, y position selected by the robot, and the red circle indicates the angle selected by the robot. Positions are sampled uniformly on a $6 \times 6 \times 6$ grid and 60 orientations are uniformly sampled. Pixel values normalized and height selection not shown for improved visualization.

T and 3 floating point numbers representing the volume size z. Alternatively, the pair (T, z) could be encoded as the *sense* image that *would* be seen if the sensor were to move to pose T with zoom z. This is as if the action was "looking ahead" at the pose the sensor or effector would move to if this action were selected.

In particular, the *lookahead sense-move-effect* MDP has actions $sense(T_s, z_s)$ and $move\text{-}effect(T_e, z_e, o)$, the difference being the additional parameter $z_e \in \mathbb{R}^3_{>0}$ for *move-effect*. The action samples are encoded as the height map that would be generated by $sense(T, z)$. Because action has this rich encoding, state is just the effector status and a history of k actions.

The HSA constraints for the lookahead variant have the same parameterization – an initial pose T_s^1 and a list of pairs $[(z_1, d_1), \ldots, (z_L, d_L)]$. The semantics are slightly different. z_i for $i = 1, \ldots, L-1$ is the z_s parameter for the ith *sense*, and z_L is the z_e parameter. The d_i for $i = 1, \ldots, L-1$ specify the offset of the *sense* action samples relative to the last pose decided, T_s^i. d_L specifies the offset of T_e relative to T_s^L.

5.2.4 Relation to other approaches in the literature

We show that DQN [73] is a 1-level standard HSA and that deictic image mapping [88] is a 1-level lookahead HSA. The problem with 1-level approaches is that the number of actions scales exponentially with the dimension of the action space.

DQN

Consider a sense-move-effect MDP with HSA constraints $L = 1$, T_s^1 centered in the robot's workspace, and z_1 and d_1 large enough to capture the entire workspace. The only free action parameters for this system are the effector pose, which is sampled uniformly and spaced appropriately for the task, and the effector operation. In this case, the observations and actions are similar to that of DQN [73], and the DQN algorithm can be applied to the resulting MDP.

However, this approach is problematic in robotics because the required number of action samples is large, and the image resolution would need to be high in order to capture the required details of the scene. For example, a pick-and-place task where effector poses are in $SE(3)$, the robot workspace is 1 m^3, the required position precision is 1 mm, and the required orientation resolution is 1° per Euler angle requires on the order of 10^{17} actions. Adding more levels (i.e. $L > 1$) alleviates this problem.

Deictic Image Mapping

With $L = 1$, T_s^1 centered in the robot's workspace, z_1 the deictic marker size (e.g., the size of the largest object to be manipulated), and d_1 large enough to capture the entire workspace, HSA applied to the lookahead sense-move-effect MDP is the deictic image mapping representation [88]. Similar to the case with DQN, if the space of effector poses is large, and precise positioning is needed, many actions need to be sampled. In fact, the computational burden with the deictic image mapping representation is even larger than that of DQN due to the need to create images for each action. Yet, the deictic representation has significant advantages over DQN in terms of efficient learning due to its small observations [88].

HSA generalizes and improves both DQN and deictic image mapping by overcoming the burden for the agent to select from many actions in a single time step. Instead, the agent sequentially refines its choice of effector pose over a sequence of L decisions. We compare these approaches experimentally in Sections 5.3.1 and 5.3.2.

5.2.5 Implementation methods

To implement HSA for a sense-move-effect MDP, it is necessary to select values for HSA parameters and a training algorithm. Here are rough guidelines for making both choices for standard HSA.

HSA parameter values

Ideal values for T_s^1, L, and $[(z_1, d_1), \ldots (z_L, d_L)]$ depend on the position and size of the robot's workspace, the desired effector precision, and available computing resources. In our implementations, we have separate levels for selecting position and orientation, with position selecting levels occurring first. The procedure for deciding position selecting levels is as follows. First, the position component of the initial sensor pose T_s^1 is set to the center of the robot's workspace. Second, the number of action samples n_s depends on computing resources, e.g., the number of Q-values that can be evaluated in parallel. If $n_s = n^3$, where n is the number of position samples spaced evenly along an axis, then n is set to the largest integer such that n_s samples can be evaluated efficiently. Third, the number of levels L is the minimum number of times the workspace needs divided to achieve the desired effector precision. If $p \in \mathbb{R}^3_{>0}$ is the desired effector precision and $w \in \mathbb{R}^3_{>0}$ is the size of the workspace, $L = \max_{i=1,\ldots,3} \lceil \log_n(w(i)/p(i)) \rceil$. Fourth, sampling regions d_i for $i = 1, \ldots, L$ should be large enough so that, if patches size d_i are centered on samples in level $i-1$, the entire region is covered: $d_i = w/n^{i-1}$. Lastly, observation sizes z_i for $i = 1, \ldots, L$ should be equal to d_i or the size of the largest object to be manipulated, whichever is largest. The latter condition is necessary if the entire object must be visible to determine the appropriate action. For example, when grasping bottles to be placed upright, either the top or bottom of the bottle must be visible to determine bottle orientation in the hand. Deciding orientation selecting levels is simpler: add 1 level per Euler angle, each with the desired angular effector precision. We used roll-pitch-roll ordering, where roll is about the hand approach axis and pitch is about the axis connecting the fingers.

Training algorithm

Algorithm 5.1 is a variant of DQN [73] that follows the HSA constraints. For concreteness, this implementation stores experiences for Q-learning; modification for other temporal difference (TD) update rules, such as Sarsa [92] or Monte Carlo (MC) [103], is straight-forward. For simplicity of exposition, we also restrict to the case where image history consists of the current image I and the image I_h before the last grasp, effector status is binary *empty* or *holding*, and the effector operation is binary *open* or *close*.

Initially, the Q-function gets random weights, the experience replay database is empty, and the probability of taking random actions $\epsilon = 1$ (line 1). The environment is initialized to a scene unique to each episode (line 2). For each time step, the effector status is observed (line 5), and I_h is the previously observed image if the effector is

Algorithm 5.1: Train standard HSA.

```
Input: $nEpisodes$, $t_{max}$, $T_1$, $n_s$, $L$, $[(z_1, d_1), \ldots, (z_L, d_L)]$, $maxExperiences$,
       $trainEvery$
1  Initialize $Q$, $D$, $\epsilon$
2  for $i \leftarrow 1, ..., nEpisodes$ do
3      $env \leftarrow$ InitializeEnvironment($i$)
4      for $t \leftarrow 1, \ldots, t_{max}$ do
5          $h =$ GetEffectorStatus($env$)
6          $I_h = \text{NULL}$
7          if $t > 1 \wedge h = holding$ then
8              $I_h \leftarrow I$
9          for $l \leftarrow 1, \ldots, L$ do
10             $I \leftarrow sense(T_l, z_l)$
11             $o' \leftarrow (h, I_h, I)$
12             $T_{l+1} \leftarrow$ GetPose($Q, o', T_l, d_l, n_s, \epsilon$)
13             $a' \leftarrow T_l^{-1} T_{l+1}$
14             if $t > 1 \vee l > 1$ then
15                 $D \leftarrow D \cup \{(o, a, o', r)\}$
16             $o \leftarrow o'$; $a \leftarrow a'$; $r \leftarrow 0$
17         $op \leftarrow$ GetEffectorOperator($h$)
18         $overtAct \leftarrow move\text{-}effect(T_{L+1}, op)$
19         $r \leftarrow$ Transition($env, overtAct$)
20     $D \leftarrow D \cup \{(o, a, \text{NULL}, r)\}$
21     if $i$ mod $trainEvery = 0$ then
22         $D \leftarrow$ PruneExperiences($D, maxExperiences$)
23         $Q \leftarrow$ UpdateQFunction($D, Q$)
24     $\epsilon \leftarrow$ Decrement-$\epsilon$($|D|$)
```

holding something (lines 6-8). Then, for each HSA level, a *sense* action is taken (line 10), the pose of the next *sense* action is determined either randomly or according to Q (line 12), and the experience is saved (line 15). Actions are encoded relative to the previous sense pose (line 13). Next, the robot moves the effector to T_{L+1} and performs an operation *op*, after which a reward is observed (lines 17 - 19). Finally, after *trainEvery* episodes, the Q function is updated with the current experiences (lines 21 - 23), and ϵ is set inversely proportional to the number of experiences (line

24).

5.3 Application Domains

In this section we compare the HSA approach in four application domains of increasing complexity. The complexity increases in terms of the size of the action space and in terms of the diversity of object poses and geometries. We analyze simpler domains because the results are more interpretable and learning is faster (Table 5.1). More complex domains are included to demonstrate the practicality of the approach. All training is in simulation, and the last two domains include test results for a psychical robotic system.[3]

	Tabular Pegs on Disks	Upright Pegs on Disks	Bottles on Coasters	6-Dof Pick-and-Place
Time (hours)	0.23	1.29	8.12	96.54

Table 5.1: Average simulation time for the four domains. Times are averaged over 10 or more simulations on 4 different workstations, each equipped with an Intel Core i7 processor and an NVIDIA GTX 1080 graphics card.

5.3.1 Tabular pegs on disks

Here we analyze the HSA approach applied to a simple, tabular domain, where the number of states and actions is finite. The domain consists of two types of objects – pegs and disks – which are situated on a 3D grid (Figure 5.5). The robot can move its effector to a location on the grid and open/close its gripper. The goal is for the robot to place all the pegs onto disks.

As this domain has finite state and action spaces, standard RL algorithms are guaranteed to converge to an optimal solution [118, 31]. However, the number of state-action pairs is too large for practical implementation unless some abstraction is applied. The main question addressed here is if convergence guarantees are maintained with the HSA abstraction.

Ground MDP

Tabular pegs on disks is first described without the sense-move-effect abstraction.

[3]Source code is available at github.com/mgualti/Seq6DofManip (for pegs on disks and bottles on coasters) and github.com/mgualti/DeepRLManip (for 6-DoF pick-and-place).

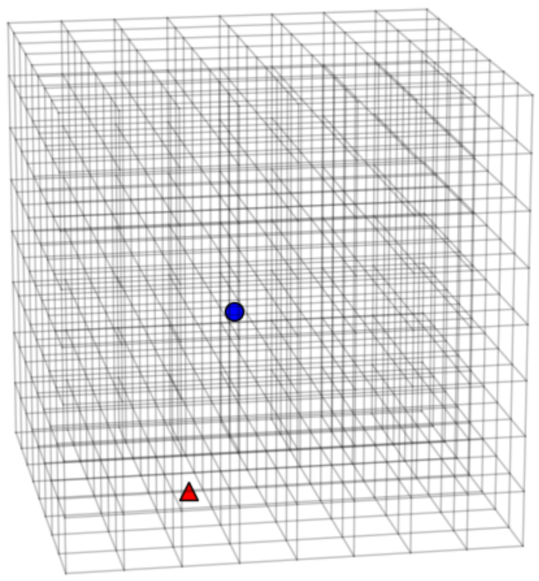


Figure 5.5: Tabular pegs on disks with an $8 \times 8 \times 8$ grid, 1 peg (red triangle), and 1 disk (blue circle).

- **State.** A set of pegs $P = \{p_1, \dots, p_n\}$, a set of disks $D = \{d_1, \dots, d_n\}$, and the current time $t \in \{1, \dots, t_{max}\}$. A peg (respectively disk) is a location $(x, y, z) \in \{1, \dots, m\}^3$ except peg locations are augmented with a special in-hand location h. Pegs (respectively disks) cannot occupy the same location at the same time, but one peg and one disk can occupy the same location at the same time.
- **Action.** *move-effect*(x, y, z), which moves the effector to $(x, y, z) \in \{1, \dots, m\}^3$ and opens/closes. It opens if a peg is located at h and closes otherwise.
- **Transition.** t increments by 1. If no peg is at h and a peg p is at the action location, then the peg is grasped ($p = h$). If a peg is located at h and the action location a does not contain a peg, the peg is placed ($p = a$). Otherwise, the state remains unchanged.
- **Reward.** 1 if a peg is placed on an unoccupied disk, -1 if a placed peg is removed, and 0 otherwise.

Initially, pegs and disks are at distinct locations, and no peg is in the effector. The time horizon is $t_{max} = 2n$, where there is just enough time to grasp and place each peg. This MDP satisfies the Markov property because the next state is completely determined from the current state and action. The number of possible states is shown in Equation 5.1, and the number of actions is $|A| = m^3$. It is not practical to learn the optimal policy by enumerating all state-action pairs for this MDP: for example, if $m = 16$ and $n = 3$, the state-action value lookup table size is on the order of 10^{24}.

$$|S| = \binom{m^3+1}{n}\binom{m^3}{n} t_{max} \tag{5.1}$$

Sense-move-effect MDP

We apply the sense-move-effect abstraction (Section 5.2.1) and the standard HSA constraints (Section 5.2.2) to the tabular pegs on disks problem. The process is illustrated in Figure 5.6. At level 1, the sensor perceives the m^3 grid as 8 cells, each summarizing the contents of an octant in the underlying grid. The robot then selects one of these cells to attend to. At levels $2, \ldots, L-1$, the sensor perceives 8 cells revealing more detail of the octant selected in the previous level. At level L, the sensor perceives 8 cells in the underlying grid, and the location of the underlying action is determined by the cell selected here. Without loss of generality, assume the grid size m of the ground MDP is a power of 2 and the number of levels L is $\log_2(m)$.

- **State.** The current level $l \in \{1, \ldots, L\}$, the overt time step $t \in \{1, \ldots, t_{max}\}$, a bit $h \in \{0, 1\}$ indicating if a peg is held, and the tuple $G = (G_p, G_d, G_e)$ where each $G_i \in \{0, 1\}^8$. G_p indicates the presence of unplaced pegs, G_d unoccupied disks, and G_e empty space.
- **Action.** The action is $a \in \{1, \ldots, 8\}$, a location in the observed grids.
- **Transition.** For levels $l = 1, \ldots, L-1$, the robot selects a cell in G which corresponds to some partition of space in the underlying grid. The sensor perceives this part of the underlying grid and generates the observation at level $l+1$. For level L, the L selections determine the location of the underlying *move-effect* action, l is reset to 1, and otherwise the transition is the same as in the ground MDP.
- **Reward.** The reward is 0 for levels $1, \ldots, L-1$. Otherwise, the reward is the same as for the ground MDP.

The above process is no longer Markov because a history of states and actions could be used to better predict the next state. For instance, for a sufficiently long random walk, the exact location of all pegs and disks could be determined from the history of observations, and the underlying grid could be reconstructed.

On the other hand, this abstraction substantially reduces the number of states (Equation 5.2) and actions ($|A| = 8$). The only nonconstant term (besides t_{max}) is

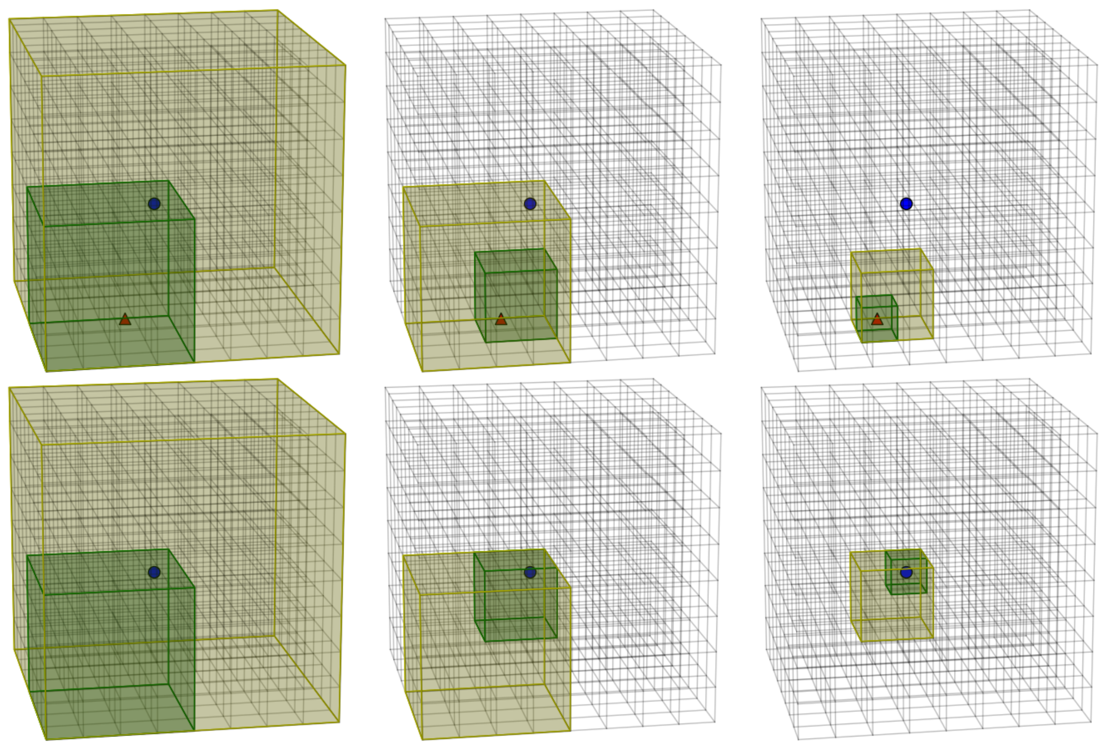

(a) Level 1 ($l = 1$). (b) Level 2 ($l = 2$). (c) Level 3 ($l = 3$).

Figure 5.6: HSA applied to the grid in Figure 5.5 ($m = 8$ and $n = 1$). The observed volume appears yellow, and the octant selected by the robot appears green. **Top**. Robot selects the peg and is holding it afterward. **Bottom**. Robot selects the disk.

logarithmic in m. Referring to the earlier example with $m = 16$ and $n = 3$, the state-action lookup table size is on the order of 10^9.

$$|S| \leq 2^{25} \log_2(m) t_{max} \tag{5.2}$$

Theoretical Results

We classify the sense-move-effect MDP with HSA constraints according to the state abstraction ordering defined in Li et al. [57]. In particular, we show Q^*-irrelevance, which is sufficient for the convergence of a number of RL algorithms, including Q-learning, to a policy optimal in the ground MDP.

Definition 5.5 (Q^*-irrelevance abstraction [57])**.** *Given an MDP $M = (S, A, P, R, \gamma)$, any states $s_1, s_2 \in S$, and an arbitrary but fixed weighting function $w(s)$, a Q^*-*

irrelevance abstraction ϕ_{Q^*} *is such that for any action* a, $\phi_{Q^*}(s_1) = \phi_{Q^*}(s_2)$ *implies* $Q^*(s_1, a) = Q^*(s_2, a)$.

ϕ_{Q^*} is a mapping from ground states to abstract states and defines the abstract MDP.[4]

Theorem 5.1 (Convergence of Q-learning under Q^*-irrelevance [57])**.** *Assume that each state-action pair is visited infinitely often and the step-size parameters decay appropriately. Q-learning with abstraction* ϕ_{Q^*} *converges to the optimal state-action value function in the ground MDP. Therefore, the resulting optimal abstract policy is also optimal in the ground MDP.*

Because Li et al. do not consider action abstractions, we redefine the ground MDP to have the same actions as the sense-move-effect MDP. Additionally, to keep the ground MDP Markov, we add the current level l and the current point of focus $v \in \{1, \ldots, m\}^3$, to the state. This does not essentially change the tabular pegs on disks domain but merely allows us to rigorously make the following connection.

Denote states of the ground MDP by s and states of the sense-move-effect MDP by $\bar{s}$. Actions are denoted a. Let $\phi_{SME} : S \rightarrow \bar{S}$ be the observation function.

Theorem 5.2 (ϕ_{SME} is Q^*-irrelevant)**.** *The sense-move-effect abstraction,* ϕ_{SME}*, is a* Q^**-irrelevance abstraction.*

Proof. We show that $Q^*(s, a)$ is uniquely determined by $\bar{s}$ and a. This proves the theorem as all states satisfying $s = \phi_{SME}(\bar{s})$ must then have the same Q^*-value.

- The reward achievable immediately after the current overt stage depends only on h (which is in $\bar{s}$), whether or not it is possible to select a peg or a disk (which is determined by G_p and G_d, respectively), and whether or not it is possible to avoid selecting a placed peg (which is determined by G_p, G_d, and G_e).

- Due to $t_{max} = 2n$ and the fact that all pegs are initially unplaced, the sum of rewards after the current stage, following an optimal policy, depends only on (a) whether or not a peg will be held after the current stage and (b) the amount of time left. (a) is determined by h, G_p, G_d, and G_e, each in $\bar{s}$. (b) is $t_{max} - t$, where t_{max} is fixed and t is in s.

[4]Although the definition is for infinite-horizon problems (due to γ), our finite-horizon problem readily converts to an infinite-horizon problem by adding an absorbing state that is reached after t_{max} overt stages. The weight $w(s)$ is the probability the underlying state is s given its abstract state $\phi(s)$ is observed. Any fixed policy, e.g. ϵ-greedy with fixed ϵ, induces a valid $w(s)$ and satisfies the definition.

Since the sum of future rewards, when following an optimal policy, is uniquely determined by $\bar{s}$ and a, $Q^*(s, a)$ is uniquely determined by $\bar{s}$ and a. $\square$

Simulation results

In these experiments, there were $n = 3$ objects, and the grid size was $m = 16$. Besides deictic image mapping (where $L = 1$), the number of levels was $L = 4$. A comparison with no abstraction (i.e., HSA with $L = 1$) was not possible because the system quickly ran out of memory (Equation 5.1). The learning algorithm was Sarsa [92], and actions were taken greedily with respect to the current Q-estimate. An optimistic initialization of action-values and random tie-breaking were relied on for exploration.

The proof to Theorem 5.2 suggests the observability of unplaced pegs, unoccupied disks, and empty space are all important for learning the optimal policy. Note that it is important to distinguish unplaced pegs from placed pegs and unoccupied disks from occupied disks. Figure 5.7 shows learning curves for the HSA agent with the sense-move-effect MDP versus an agent showing pegs/disks irregardless of whether or not they are placed/occupied.

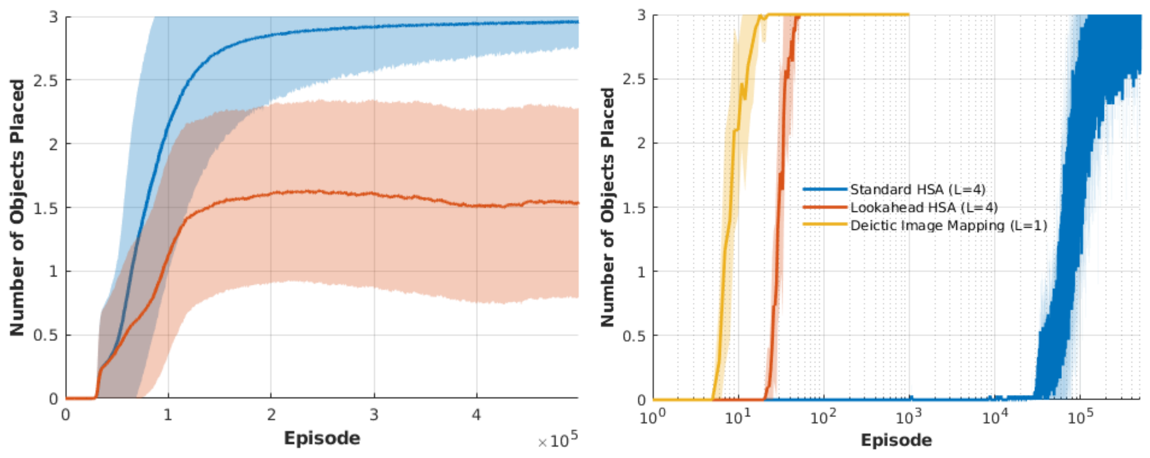


Figure 5.7: Learning curves (mean $\pm\sigma$ over 30 realizations) for the tabular pegs on disks domain. **Left**. Standard HSA (blue) versus standard HSA with a faulty sensor (red). Curves averaged over 1,000-epsisode segments for improved visualization. **Right**. Standard HSA (blue), lookahead HSA (red), and deictic image mapping (yellow). x-axis (episode number) is in log scale.

Lookahead HSA (Section 5.2.3) and deictic image mapping variants (Section 5.2.4) result in an even smaller state-action space than standard HSA. In the tabular domain, this means faster convergence (Figure 5.7). Although the deictic representation seems superior in these results, it has a serious drawback. The action-selection time scales linearly with m^3 because there is one action for each cell in the underlying grid. The lookahead variant captures the best of both worlds – small representation and fast execution. Thus, in the tabular domain, lookahead appears to be the satisfactory middle ground between the two approaches. However, for domains requiring Q-function approximation, we found the time needed to generate action images becomes more significant, and the advantage of lookahead in terms of episodes to train diminishes.

5.3.2 Upright pegs on disks

In this domain, pegs and disks are modeled as tall and flat cylinders, respectively, where the cylinder axis is always vertical (Figure 5.8, left). Unlike the tabular domain, object size and position are sampled from a continuous space. Grasp and place success are checked with a set of simple conditions appropriate for upright cylinders.[5] The reward is 1 for grasping an unplaced peg, -1 for grasping a placed peg, 1 for placing a peg on an unoccupied disk, and 0 otherwise.

Observations consist of 1 or 2 images ($k = 2$, $n_{ch} = 1$, $n_x = n_y = 64$); the current HSA level, $l \in \{1, 2, 3\}$; and the effector status, $h \in \{\mathit{empty}, \mathit{holding}\}$. Each HSA level selects (x, y, z) position (Figure 5.8, right). Gripper orientation is not critical for this problem.

Network architecture and training algorithm

The Q-function consists of 6 convolutional neural networks (CNNs), 1 for each level and effector status, with identical architecture (Table 5.2). This architecture results in faster execution time compared with our previous version [22]. The loss is the squared difference between predicted and actual Q-value target, averaged over a mini-batch. The Q-value target is the reward received at the end of the current overt stage.[6] For CNN optimization, Adam [45] is used with a base learning rate of 0.0001, weight decay of 0.0001, and mini-batch size of 64.

[5]Grasp conditions: gripper is collision-free and the top-center of exactly one cylinder is in the gripper's closing region. Place conditions: entire cylinder is above an unoccupied disk and the cylinder bottom is at most 1 cm below or 2 cm above the disk surface.

[6]With standard MC and $\gamma = 1$, the action-value target would be the sum of rewards received after the current time step [103]. Since, for this problem, no positively rewarding grasp precludes a positively rewarding place, ignoring rewards after the current overt stage is acceptable.

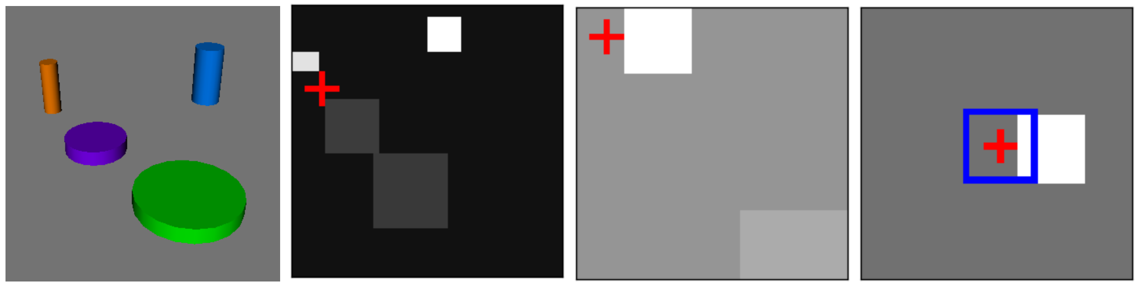


Figure 5.8: **Left**. Example upright pegs on disks scene. **Right**. Level 1, 2, and 3 images for grasping the orange peg. Red cross denotes the (x, y) position selected by the robot and the blue rectangle denotes the allowed (x, y) offset. $z_x = z_y = 36$ cm^2 for level 1 and 9 cm^2 for levels 2 and 3. $d_x = d_y = 36$ cm^2 for level 1, 9 cm^2 for level 2, and 2.25 cm^2 for level 3. Pixel values normalized and height selection not shown for improved visualization.

Layer	Kernel Size	Stride	Output Size
Conv-1	7×7	2	$32 \times 32 \times 32$
Conv-2	7×7	2	$16 \times 16 \times 64$
Conv-3	7×7	2	$8 \times 8 \times 32$
Conv-4	7×7	2	$4 \times 4 \times 32$
Conv-5	7×7	1	$4 \times 4 \times 4$

Table 5.2: CNN architecture for the upright pegs on disks domain. Each layer besides Conv-4 and Conv-5 has a rectified linear unit (ReLU) activation.

Simulation results

We tested standard HSA with 1, 2, and 3 levels. The number of actions (CNN outputs) per level was adjusted so that each case had the same 5.625 mm precision in positioning of the effector: 1 level had 4^9 outputs, 2 levels had 4^3 outputs and 4^6 outputs, and 3 levels each had 4^3 outputs. Note that with 1 level this is the DQN (i.e. no-hierarchy) approach. Exploration was ϵ-greedy with $\epsilon = 0.04$.

Results are shown in Figure 5.8, left. The 1 level case trains faster in terms of episodes because learning is over fewer time steps. The 2 levels case initially learns faster for the same reason. The 1 and 2 level cases converge to higher values because, with 3 levels, there is a higher chance of taking a random action during an overt stage. This is because more levels imply more time steps over which a random action could be selected with probability ϵ. What is important is that, in the last $5,000$ episodes when $\epsilon = 0$, all scenarios have similar performance. However, HSA trains faster than DQN in terms of wall clock time (1.29 versus 2.55 hours) because fewer actions need evaluated (192 versus 262, 144). This advantage becomes more staggering as the dimensionality of the action space increases, as in following sections.

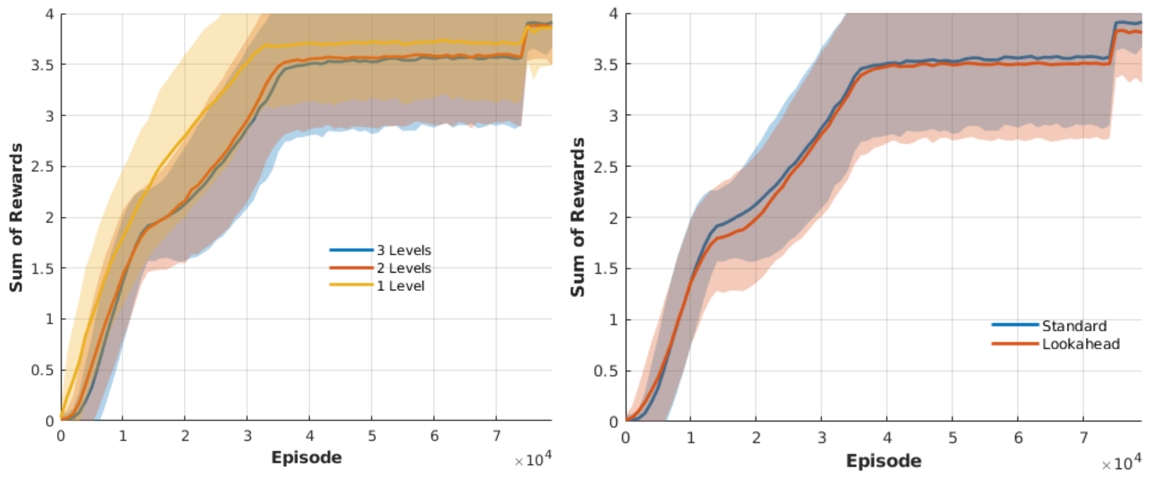


Figure 5.9: **Left**. Standard HSA with varying number of levels. (Blue) $L = 3$, (red) $L = 2$, and (yellow) $L = 1$. **Right**. Standard HSA (blue) versus lookahead HSA (red). Curves are mean $\pm\sigma$ over 10 realizations then averaged over 1,000 episode segments.

In another experiment, we tested the sensitivity of standard HSA to the choice of z and d parameters. As explained in Section 5.2.5, these parameters are selected based on task geometry. If z (respectively d) is too small, parts of the workspace will not be perceivable (respectively reachable). On the other hand, if z is too large, the scene will not be visible in detail (because the perceived images are of fixed resolution), and if d is too large, the samples at the last level will not be dense, resulting in low effector precision. Results for different values of z and d are shown in Table 5.3. The "Ideal" values are those selected according to the principles in Section 5.2.5 and correspond to the 3-levels case in Figure 5.8, left. As expected, performance is much worse when selecting z and d without consideration to task geometry.

We also compared standard HSA to lookahead HSA, both with 3 levels. We did not compare to the deictic image mapping approach (i.e., lookahead HSA with 1 level) because computation of all 4^9 images was prohibitively expensive. Results are shown in Figure 5.8, right. In contrast to the tabular results, both scenarios perform similarly. We hypothesize that the advantage of lookahead HSA is lost due to the equivariance property of CNNs. Since execution time for standard HSA is less

	Small	Ideal	Large
Level-1, $z_{xy} =$	36.0	36.0	36.0
Level-1, $d_{xy} =$	36.0	36.0	36.0
Level-2, $z_{xy} =$	6.00	9.00	12.00
Level-2, $d_{xy} =$	6.00	9.00	12.00
Level-3, $z_{xy} =$	6.00	9.00	12.00
Level-3, $d_{xy} =$	1.50	2.25	3.00
μ Return	2.69	3.91	2.83
σ Return	1.32	0.01	1.75

Table 5.3: Varying standard HSA parameters z_{xy} and d_{xy} (in cm). "Ideal" values were selected according to Section 5.2.5. "Small" (respectively "Large") values are smaller (respectively larger) than ideal. Last two rows are average and standard deviation over sum of rewards per episode, after 10 different training sessions and $1,000$ episodes per session.

than half that of lookahead (1.29 versus 3.67 hours), from now on we only consider standard HSA.

5.3.3 Bottles on coasters

The main question addressed here is if HSA can be applied to a practical problem and implemented on a physical robotic system. The bottles on coasters domain is similar to the pegs on disks domain, but now objects have complex shapes and are required to be placed upright. The reward is 1 for grasping an unplaced object more than 4 cm from the bottom (as placing with bottom grasps is kinematically infeasible with the physical system), -1 for grasping a placed bottle, 1 for placing a bottle, and 0 otherwise.[7]

Observations are similar to those for the upright pegs on disks domain except now the image resolution is $n_x = n_y = 48$ and the overt time step is input to grasp networks (but not the place networks). HSA has three levels selecting (x, y, z) position and one level selecting orientation about the gripper approach axis (Figure 5.4).

To achieve the target precision in effector pose (3.75 mm position and $6°$ orientation for grasping), DQN (or 1-level HSA) would need to evaluate over 53 million

[7]Grasp conditions: gripper closing region intersects exactly one object and the antipodal condition (Definition 2.4) with $30°$ friction cone. Place conditions: bottle is upright, center of mass (x, y) position at least 2 cm inside an unoccupied coaster, and bottom within ± 2 cm of coaster surface.

actions. Evaluation was prohibitively expensive with our computing hardware. HSA only needs 404 actions; although we use 708 to achieve redundancy, with little loss in computation time as the evaluation is done in parallel.

Network architecture and training algorithm

The network architecture is shown in Table 5.4. There is one network for each HSA level and effector status. The loss is the mean squared difference between predicted and actual sum of future rewards. For optimization, Adam [45] is used with a base learning rate of 0.0001, weight decay of 0, and mini-batch size of 64.

Layer	Kernel Size	Stride	Output Size
Conv-1	8×8	2	$24 \times 24 \times 64$
Conv-2	4×4	2	$12 \times 12 \times 64$
Conv-3	3×3	2	$6 \times 6 \times 64$
Conv-4 / IP-1	2×2 / -	1 / -	$6^3/n_{orient}$

Table 5.4: CNN architecture for the bottles on coasters domain. Each layer besides the last has a ReLU activation. The last layer is a convolution layer for levels 1-3 (selecting position) and an inner product (IP) layer for level 4 (selecting orientation). $n_{orient} = 60$ for grasp networks and $n_{orient} = 3$ for place networks.

Simulation results

70 bottles from 3DNet [122] were scaled uniformly at random from 10 to 20 cm in height. Bottles were placed upright with probability 1/3 and on their sides with probability 2/3. Episodes were initialized with two bottles and two coasters not in contact. Learning curves are shown in Figure 5.10. Performance is lower than that of the upright pegs on disks domain, reflective of the additional problem complexity.

To test robustness of the system to background noise, we ran the same experiment with the addition of distractor objects. These distractors are three rectangular blocks, with side lengths 1 to 4 cm, scattered randomly in the scene (e.g., Figure 5.11, left). Learning performance is only slightly lower (Figure 5.11, right). However, if clutter is present at test time, it is important to train the system with clutter. The robot trained without clutter places an average of 1.24 bottles in the cluttered environment (versus 1.55 if trained with clutter). The distractors are visible at some levels (e.g., level 1), so the robot does need to learn to avoid them.

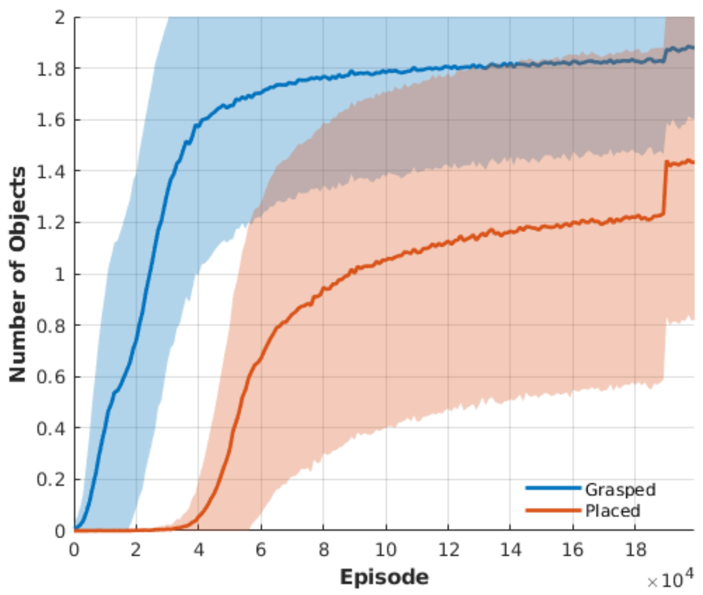


Figure 5.10: Number of bottles grasped (blue) and placed (red). Curves are mean $\pm\sigma$ over 10 realizations then averaged over $1,000$ episode segments. Standard HSA with $L = 4$.

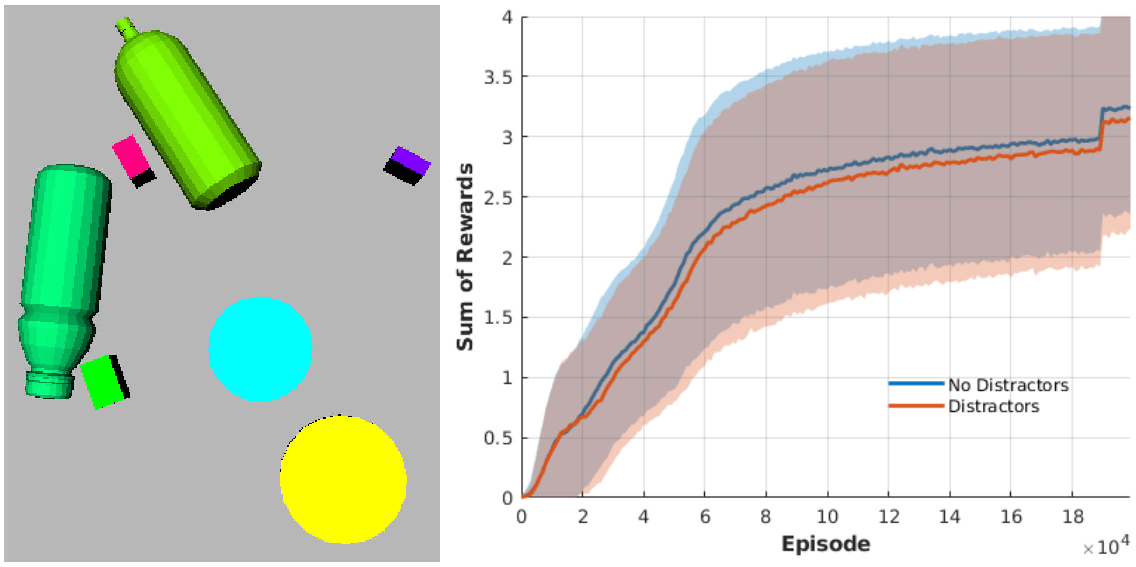


Figure 5.11: **Left.** Scene with clutter. **Right.** Learning curves comparing average sum of rewards when distractors are not present (blue) and present (red).

Top-n sampling

Before considering experiments on a physical robotic system, we address an important assumption of the move-effect system of Section 5.1. The assumption is the effector can move to any pose T_e in the robot's workspace. Motion planning algorithms make this a reasonable assumption for the most part; nonetheless, a pose can still be unreachable due to obstacles, no inverse kinematics (IK) solution, or motion planning failure.

To address this issue, multiple, high-valued actions are sampled from the policy learned in simulation. In particular, for each level l of an overt stage, we take the top-n samples according to Equation 5.3, where Q_l is the Q-value estimate at level l, Q_{max} is the maximum possible Q-value, Q_{min} is the minimum possible Q-value, and $p_0 = 1$. For the bottles on coasters domain, $Q_{max} = 4$ if at most two objects can be placed, and $Q_{min} = 0$, since, in the worst case, nothing is placed at the end. During test time, the resulting n, T_e samples are checked for IK and motion plan solution in descending order of p_L value.

$$p_l = p_{l-1} \frac{Q_l - Q_{min}}{Q_{max} - Q_{min}}, \qquad l = 1, \ldots, L \tag{5.3}$$

Improved chances of finding a reachable gripper pose is just one benefit of top-n sampling. We also observe improved performance. Sampling top-n p_l values is an ensemble method where each level votes on the value of the final overt action (cf. [4]). Since different CNNs vote on the value of n actions, a mistaken value of one action at one level is not catastrophic, as it is for the greedy (top-1) policy.

Real robot results

We tested the bottles on coasters task with the system depicted in Figure 5.12, left. The system consists of a UR5 arm, a Robotiq 85 parallel-jaw gripper, and a Structure depth sensor. The test objects (Figure 5.12, right) were not observed during training. Scenes were initialized as follows. Two coasters were randomly selected and placed in arbitrary positions in the back half of the robot's workspace (as too close resulted in unreachable places). Two bottles were randomly selected and placed upright with probability 1/3 and on the side with probability 2/3. The objects were not placed in contact with each other. Top-n sampling with $n = 200$ was used.

Results are summarized in Table 5.5, and a successful sequence is depicted in Figure 5.13. A grasp was considered successful if a bottle was lifted to the "home" configuration; a place was considered successful if a bottle was placed upright on an unoccupied coaster and remained there after the gripper withdrew. Failures were:

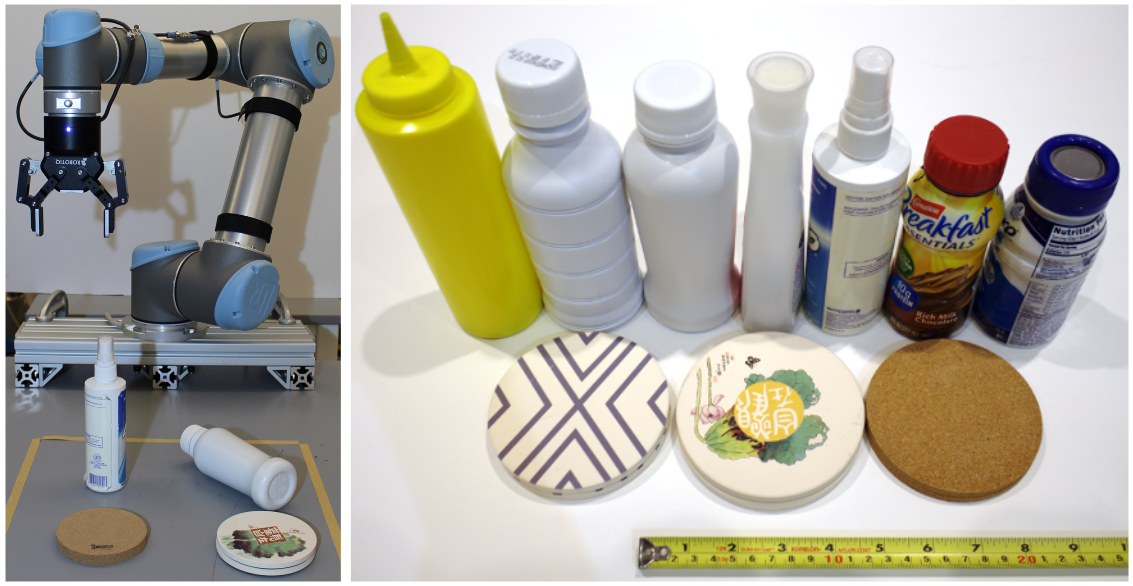


Figure 5.12: **Left**. Test setup for bottles on coasters task: a UR5 arm, Robotiq 85 gripper, Structure depth sensor (mounted out of view above the table and looking down), 2 bottles, and 2 coasters. **Right**. Test objects used in UR5 experiments.

grasped a placed object (×3), placed too close to the edge of a coaster and fell over (×3), placed upside-down (×2), object slip in hand after grasp caused a place failure (×1), and object fell out of hand after grasp (×1).

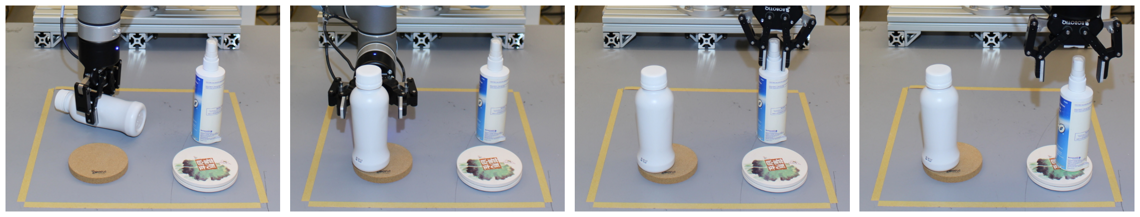


Figure 5.13: Successful trial – all bottles placed in four overt stages.

5.3.4 6-DoF pick-and-place

Here we ask if HSA can be applied to 6-DoF action spaces. With an action space of this size, naive sampling is out of the question. The HSA constraints are similar those for the bottles on coasters domain except we add two additional levels for orientation (Figure 5.14).

	Grasp	Place
Attempts	60	59
Success Rate	0.98	0.90
Number of Objects	1.97 ± 0.18	1.67 ± 0.48

Table 5.5: Performance for UR5 experiments placing two bottles on two coasters averaged over 30 episodes with $\pm\sigma$. Task success rate with $t_{max} = 4$ was 0.67.

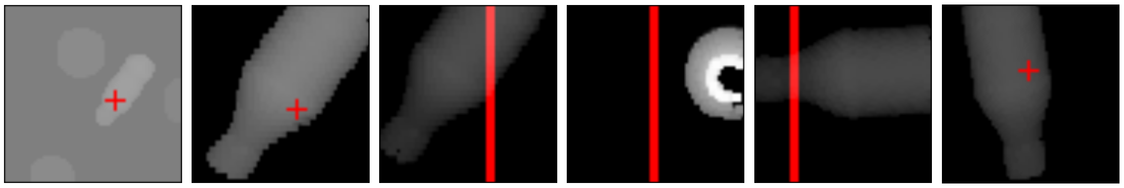
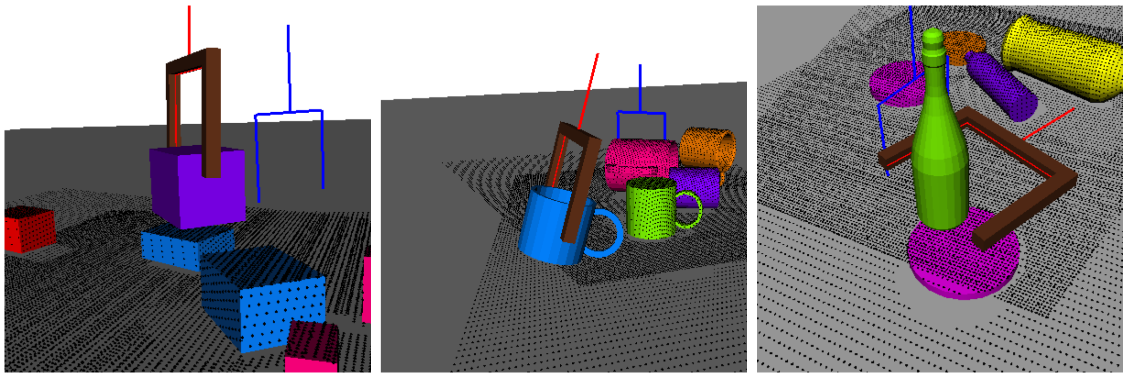

(a) Level 1. (b) Level 2. (c) Level 3. (d) Level 4. (e) Level 5. (f) Level 6.

Figure 5.14: Example sequence for grasping a bottle: **(a)** coarse position, **(b)** fine position, **(c)** orientation about gripper's approach, **(d)** orientation about gripper's closing direction, **(e)** orientation about gripper's approach, and **(f)** final position. Crosses denote the position selected for the next action, and lines denote the orientation selected for the next action (where the center is 0 and edges $\pm\pi$). For all images besides (d), the gripper approaches the image plane with the fingers on the left and right sides. For (d), the gripper approaches the image plane from the left with the fingers on the top and bottom sides. Orientation is selected about the in-plane axis.

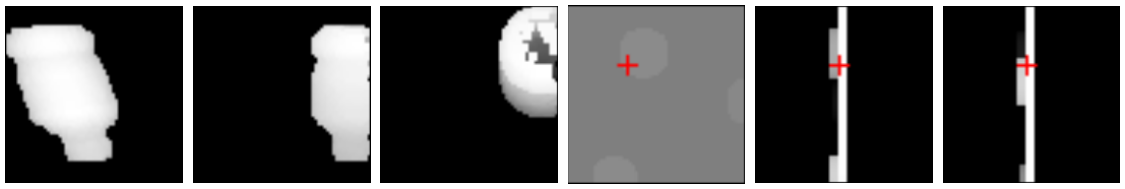
We test HSA on three different 6-DoF pick-and-place tasks: placing a block on top of another block, placing a mug upright onto a table, and placing a bottle onto a coaster (Figure 5.15). Since the orientation of mugs is often ambiguous from a single top-view, point clouds are obtained from two depth sensors, situated on either side of the robot's workspace, 45° above the table. Each observation in the HSA hierarchy includes projections from three directions, as in Figure 5.16. Observations include more 3D information compared to the bottles on coasters domain.

Ideally, the reward function should simply describe the set of goal states, e.g., +1 when the object is placed correctly and 0 otherwise. However, successful 6-DoF pick-and-places are rare at the beginning of learning when actions are random. To make the reward signal more informative, we assign a reward in $[0, 1]$ for a successful grasp and an additional reward in $[0, 1]$ for a successful place. For both grasping and placing there are required conditions and partial credit conditions. All required conditions must be met for the reward to be nonzero; otherwise the reward is proportional to the number of partial credit conditions met. For example, when placing bottles,



(a) Blocks task. (b) Mugs task. (c) Bottles task.

Figure 5.15: 6-DoF pick-and-place tasks: **(a)** place a block on top of another block, **(b)** place a mug upright onto the table, and **(c)** place a bottle upright onto a coaster. The blue gripper shows the grasp, and the red gripper shows the place. Scenes are cluttered with objects of the same category.



(a) $I_1(1)$. (b) $I_1(2)$. (c) $I_1(3)$. (d) $I_2(1)$. (e) $I_2(2)$. (f) $I_2(3)$.

Figure 5.16: Images observed when placing a bottle onto a coaster with $k = 2$, $n_{ch} = 3$, and $n_x = n_y = 60$. **(a-c)**. Projections of the point cloud along z, y, and x directions just before the last *move-effect(close)* was called. **(d-f)**. Projections of the current point cloud along z, y, and x directions. The red cross indicates the next position the robot intends to move to in order to place the bottle onto a coaster.

the required conditions include: (a) the bottle is within 30° of upright, (b) above a coaster, (c) no more than 4 cm above the coaster, and (d) not in collision or not in collision if moved up 2 cm. The partial credit conditions include: (a) within 15° of upright, (b) no more than 2 cm above the coaster, and (d) not in collision. Similar reward functions were developed for both grasping and placing stages for the other tasks.

Network architecture and training algorithm

The network architecture used to approximate the Q-function is shown in Figure 5.17. Unlike the CNNs in Tables 5.2 and 5.4, action is an input and there are several IP layers. The advantage of this architecture is that the action sampling strategy need not be fixed. For instance, actions can be sampled more densely at test time than during training, when performance is more important than speed. Disadvantages of this architecture include it (a) evaluates slower and (b) has higher loss values versus those in Tables 5.2 and 5.4.

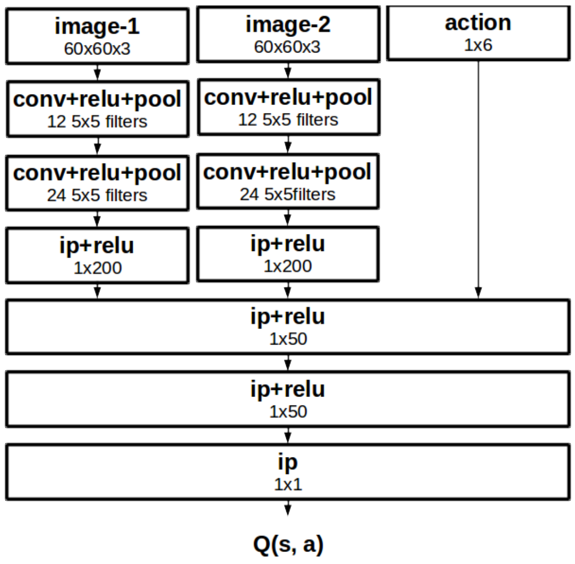


Figure 5.17: CNN architecture for 6-DoF experiments. A different set of weights is trained for each level and for each overt stage.

As with the bottles on coasters domain, the training algorithm was gradient MC, where the loss of a mini-batch is the average squared difference between predicted and actual sum of future rewards [103].

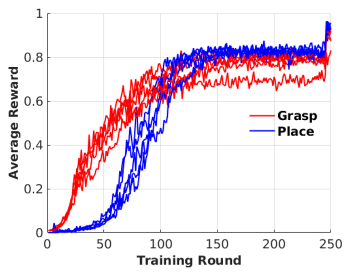
n-trial sampling

Instead of the top-n sampling that was used for bottles on coasters, the 6-DoF pick-and-place experiments relied on n-trial sampling. HSA is run in n independent trials, where the samples are different for each trial, and the trial that ends with the highest valued action is selected. This is analogous to how Monte Carlo tree search is used on-line to improve policies [106]. The benefits of n-trial sampling are similar to top-

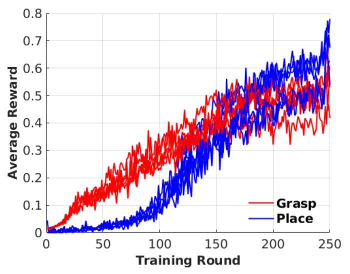
n sampling: a reachable gripper pose is more likely to be found and performance improves.

Simulation results

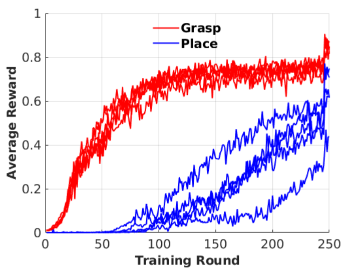
Learning curves for the three 6-DoF tasks are shown in Figure 5.18. In terms of task difficulty, placing blocks appears easiest, followed by placing bottles on coasters, followed by placing mugs on the table. This ordering is consistent with the ambiguity of the object pose given a partial point cloud: blocks have three planes of symmetry, bottles have two orthogonal planes of symmetry, and mugs have at most one plane of symmetry (an upright plane dividing the handle). We also notice there is a high variance between training realizations placing bottles on coasters. This may be due to the difficulty of finding the first several successful placements: the space of goal placements is small compared to the blocks scenario (where the desired orientation is less specific) or the mugs scenario (where anywhere on the table is acceptable).



(a) Blocks.



(b) Mugs.



(c) Bottles.

Figure 5.18: Learning curves for 6-DoF pick-and-place. Grasp and place rewards shown in different colors – episode return is the sum of the two. Rewards are averages over all of the episodes in a training round. Each training round consisted of $2,000$ episodes followed by $3,000$ iterations of stochastic gradient descent. Maximum possible reward for a grasp/place is 1. Each plot includes learning curves for five independent runs.

Recall that grasp pose detection (GPD) relies on heuristics for sampling grasp poses (Chapter 4 and [84]). We ask if HSA can do better, as the sampling strategy is learned rather than hand-coded. We compared the grasp performance of HSA versus GPD with blocks, bottles, and mugs. The result is summarized in Table 5.6. HSA with top-10 sampling outperforms GPD except on bottles. This makes sense as GPD's heuristics enable finding many grasp samples on cylindrical objects [84].

	Blocks A	Blocks A∧CF	Mugs A	Mugs A∧CF	Bottles A	Bottles A∧CF
GPD	0.95	0.51	0.77	0.52	**0.98**	0.71
1-Trial	0.86	0.82	0.65	0.58	0.83	0.81
10-Trial	**0.99**	**0.98**	**0.85**	**0.80**	0.94	**0.93**

Table 5.6: Grasp success rates averaged over $1,000$ independent trials. Performance is shown with the antipodal condition (Definition 2.4) only (A) and with both antipodal and collision-free conditions (A∧CF). GPD used 500 grasp samples and the highest-scoring grasp for evaluation. Both GPD and HSA were trained with 3DNet objects in the named category.

Real robot results

We tested 6-DoF pick-and-place on the same three tasks using a UR5 robot, Robotiq 85 gripper, and wrist-mounted Structure depth sensor. n-trial sampling with $n = 100$ was used. The test objects used in the experiments are shown in Figure 5.19, and successful executions for each task are shown in Figure 5.20.

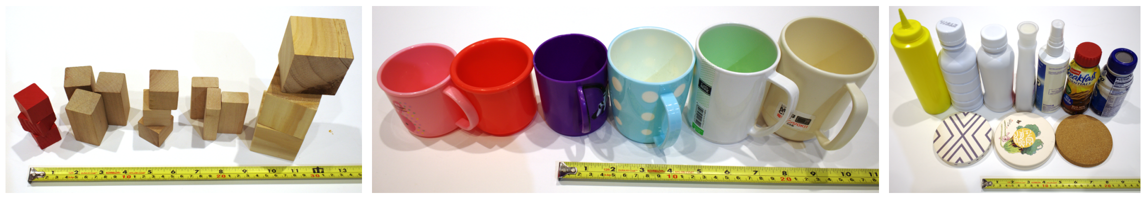
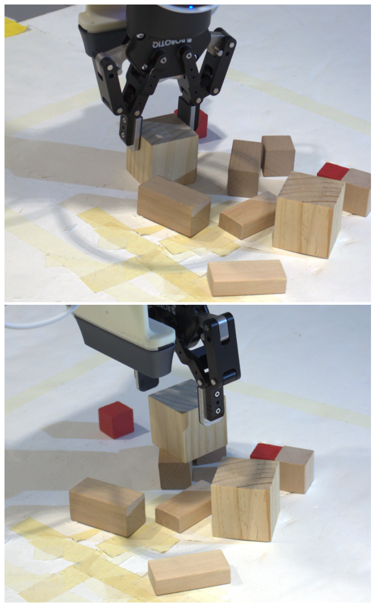


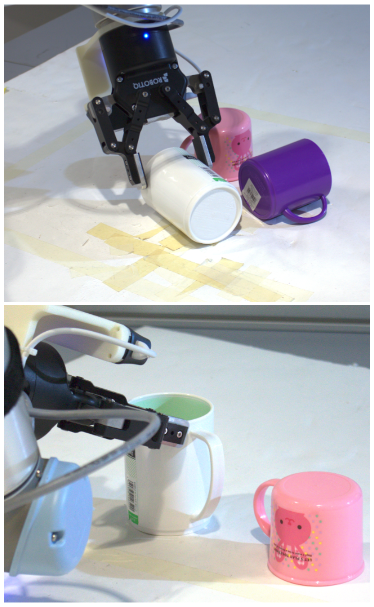
Figure 5.19: **Left**. Blocks test set. 10/15 blocks were randomly selected for each episode. **Center**. Mugs test set. 5/6 mugs were randomly selected for each episode. **Right**. Bottles test set. 3/7 bottles were randomly selected for each episode. All 3 coasters were present in each episode.

Results are shown in Table 5.7. Performance for the 6-DoF bottles task is worse than that for the upright bottle on coasters domain (64% versus 90% place success rate). We think this is due to (i) the lossier CNN architecture and (ii) the image changes significantly after an out-of-plane rotation, so predicting the value of a level-4 action is difficult. Nonetheless, we demonstrate the feasibility of learning 6-DoF pick-and-place, with variable place choices, using value-based RL.

5.4 Discussion

The sense-move-effect abstraction, when coupled with HSA, is an effective way to simultaneously handle (a) high-resolution, 3D observations and (b) high-dimensional,


(a) Blocks task.


(b) Mugs task.

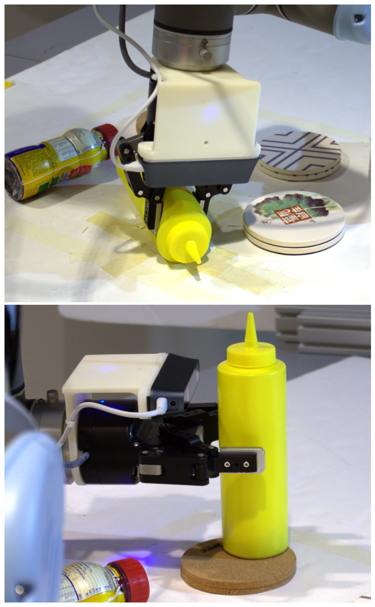

(c) Bottles task.

Figure 5.20: Example 6-DoF grasps (**top**) and placements (**bottom**). Notice for the mugs example **(b)**, the grasp is diagonal to the mug axis, and the robot compensates for this by placing diagonally with respect to the table surface.

	Blocks	Mugs	Bottles
Grasp	0.96	0.86	0.89
Place	0.67	0.89	0.64
Task	0.64	0.76	0.57
n Grasps	50	51	53
n Places	48	44	47

Table 5.7: **Top.** Grasp, place, and task success rates for the three 6-DoF tasks with $t_{max} = 2$ (i.e., pick-and-place). **Bottom.** Number of grasp and place attempts.

continuous action spaces. These two issues are intrinsic to the problem of pick-and-place of partially visible, novel objects. More specifically:

- Compared to a flat representation, e.g., DQN or deictic image mapping, HSA has exponentially fewer actions that need evaluated (Section 5.2.2, hierarchical spatial attention, and Section 5.3.1, tabular pegs on disks).

- HSA generalizes DQN, and lookahead HSA generalizes deictic image mapping (Section 5.2.4, relation to other approaches in the literature).
- The partial observability induced by HSA does not necessarily preclude learning an optimal policy (Section 5.3.1, tabular pegs on disks).
- HSA may take longer to learn than DQN in terms of the number of episodes to convergence, but HSA executes faster (Section 5.3.2, upright pegs on disks).
- HSA can be applied to 4-DoF and 6-DoF pick-and-place, where the action space is too large for naive sampling; although, additional research is needed to improve value prediction for out-of-plane rotations (Section 5.3.3, bottles on coasters, and Section 5.3.4, 6-DoF pick-and-place).

5.4.1 Limitations

A concern with all deep RL methods is that modeling and optimization errors induced by the use of function approximation prevent the robot from learning an optimal policy. This is true for even simple problems, such as the upright pegs on disks problem. Also, Q-functions do not readily transfer to different tasks or changes in the environment. For example, even small changes to the task, such as the inclusion of distractor objects, requires complete retraining of the system for maximum performance. For these reasons, we turn our investigation to modular approaches.

5.4.2 Related work

This chapter is based on the findings first published in [20, 23]. We were inspired by previous work in RL for robotic manipulation and attention models.

Reinforcement learning for robotic manipulation

Like several others, we apply RL to the problem of robotic manipulation [46, 55, 41, 91, 20, 3]. RL is appealing for robotic control for several reasons. First, several algorithms (e.g., [117, 92]) do not require a complete model of the environment. This is of particular relevance to robotics, where the environment is dynamic and difficult to describe exactly. Additionally, observations are often encoded as camera or depth sensor images. Deep Q-Networks (DQN) demonstrated an agent learning difficult tasks (Atari games) where observations were image sequences and actions were discrete [73]. An alternative to DQN that can handle continuous action spaces are actor-critic methods like DDPG [60]. Finally, RL – which has its roots in optimal control – provides tools for the analysis of learning optimal behavior (e.g. [118, 31, 57]), which we refer to in the tabular pegs on disks domain in Section 5.3.1.

Models of visual attention

Following the early work of Whitehead and Ballard [119], we distinguish overt actions (which directly affect change to the environment) from perceptual actions (which retrieve information). Similar to their agent model, our abstract robot has a virtual sensor which can be used to focus attention on task-relevant parts of the scene. The present work updates their methodology to address more realistic problems, and we extend their analysis by describing a situation where an optimal policy can be learned even in the presence of "perceptual aliasing" (i.e., partial observability).

Attention mechanisms have also been used with artificial neural networks to identify an object of interest in a 2D image [99, 50, 72, 32]. Our situation is more complex in that we identify 6-DoF poses of the robot's hand. Improved grasp performance has been observed by active control of the robot's sensor [25, 76]. These methods attempt to identify the best sensor placement for grasp success. In contrast, our robot learns to control a virtual sensor for the purpose of reducing the complexity of action selection and learning.

Work contemporary with ours considered attention for grasping [123]. In addition to controlling a 6-DoF gripper pose, their system also decides joint angles for a 3-fingered gripper. Their system learns parameters analogous to our manually specified HSA parameters z, d, and L. Instead of a value-based algorithm like DQN, their system uses policy gradient. Their results provide further evidence that attention is a useful prior for high-DoF manipulation.

Part III

Modular Architectures

Each mental agent by itself can only do some simple thing that needs no mind or thought at all. Yet when we join these agents in societies – in certain very special ways – this leads to true intelligence.

Marvin Minksy in *The Society of Mind*

Chapter 6

Regrasp Planning with Uncertain Object Instance Segmentation and Shape Completion

In this chapter, we explore a modular approach to regrasping partially visible, novel objects. A modular system has separate perception and planning components. Perception takes in sensor data and produces models of the objects. Planning calculates a sequence of grasps and places displacing a model to a goal pose, assuming the perceived model is correct. Communities specializing in perception (e.g., [90, 89, 128, 126]) and planning (e.g., [109, 2, 1, 80, 48, 113]) advance their respective components, and, in the end, the components are combined into a working system.

Specifically, consider the case where sensor data is a point cloud and where object models are completed point clouds, i.e., occlusion-free point clouds. The goal of perception is to predict the objects' shapes in occluded regions: a difficult task given that objects are partially visible and novel. The goal of planning is to determine a sequence of grasps and places that will geometrically transform a point cloud to a goal pose.

While this basic modular, shape completion approach has been demonstrated to be effective [111, 64, 16], it ignores the uncertainty inherently present in the perceived object models. With novel objects, geometries in unobserved areas are just guesses. For example, if a bottle is lying on its side with the bottom occluded by another object, the length of the bottle cannot be determined exactly, as there may be many bottles with similar shapes but different lengths. If the planner is unaware of this uncertainty, it could calculate grasps on the supposed bottom of the bottle, which could result in a grasp failure. Thus, it is important for perception to communicate to planning its uncertainty and for planning to efficiently incorporate this uncertainty.

This example suggests additional integration between perception and planning is needed for reliable manipulation.

In this chapter we take the following approach: (a) use perception to predict the complete geometry of the objects as well as segmentation and shape completion uncertainty and (b) incorporate perceptual uncertainty into the regrasp planning cost. The cost function guides the planner toward plans that are likely to be executed successfully. We compare six different cost functions, four of which explicitly model the probability of successfully executing a regrasp plan, including grasp quality (GQ), Monte-Carlo (MC) sampling, uncertainty at contact points (CU), and success prediction (SP). With only small modifications to existing planners, we efficiently account for perceptual uncertainty.

We tested this approach with bin packing and bottle arrangement tasks in both simulation and the real world. Results show perception is indeed a significant source of error and shape completion is critical to regrasp planning. Also, the SP method consistently outperforms three other methods (no cost, step cost, and GQ) which do not account for perceptual uncertainty in terms of avoiding grasp failures. Furthermore, the SP cost evaluates faster than the MC cost.

6.1 Problem Statement

Consider the problem of planning robot motions to place a partially visible object of unknown shape into a goal pose. In particular, we consider this problem in the context of the following system:[1]

Definition 6.1 (Move-open-close system)**.** *A* move-open-close system *consists of one or more objects, a robotic manipulator, and one or more depth sensors, each situated in 3D Euclidean space. Objects are rigid masses $O_1, \ldots, O_{n_{obj}} \subseteq \mathbb{R}^3$, sampled randomly from a fixed, unknown probability distribution. The manipulator is equipped with a parallel-jaw gripper with status* empty *or* holding*. The action of the robot is to move the gripper to a target pose $T_e \in SE(3)$, followed by either gripper* open *or* close*. At each step, the robot acquires a point cloud $C \in \mathbb{R}^{n \times 3}$, observes its gripper status, and takes an action.*

As in Chapter 2, to simplify planning, we avoid dynamic actions (e.g., pushing). In particular, *close* actions should fix an object rigidly in the gripper, and *open* actions should place an object at rest. Specifically, we aim for actions which result in either an antipodal grasp (Definition 2.4) or a stable, horizontal placement (Definition 2.5). We now state the problem as follows:

[1]This is a move-binary-effect system (Definition 2.1) with a parallel-jaw gripper as the effector.

Definition 6.2 (Regrasping under perceptual uncertainty)**.** *Given a move-open-close system, a point cloud for each object* $\{\bar{C}_i \in \mathbb{R}^{\bar{n}_i \times 3}\}_{i=1}^{n_{obj}}$ *with corresponding segmentation uncertainties* $\{U_i \in \mathbb{R}^{n_i}\}_{i=1}^{n_{obj}}$ *and shape completion uncertainties* $\{\bar{U}_i \in \mathbb{R}^{\bar{n}_i}\}_{i=1}^{n_{obj}}$*, and a set of goal poses for each object* $\{\{T_{ij} \in SE(3)\}_{j=i}^{n_{goal}}\}_{i=1}^{n_{obj}}$*, find a sequence of antipodal grasps and stable, horizontal placements maximizing the probability of displacing an object to a goal pose.*

The segmentation uncertainty $U_i \in \mathbb{R}^{n_i}$ estimates the probability each point belongs to the ith object segment. The shape completion uncertainty $\bar{U}_i \in \mathbb{R}^{\bar{n}_i}$ estimates the probability each point is within a threshold distance of the true object shape. Intuitively, actions should account for uncertainty in perception, as grasping and placing on uncertain object parts is likely to result in unpredictable movements of the object. In this chapter, we obtain these uncertainty estimates with BoNet [126] and point completion network (PCN) [128], as explained in Section 6.2.1.

6.2 System Overview

Consider a modular, perception-planning pipeline for displacing partially visible, novel objects, where the regrasp planner addresses the problem of regrasing under perceptual uncertainty (Definition 6.2). Such a system is summarized in Figure 6.1. For each perception-action cycle, the environment produces a point cloud, the geometry of the scene is estimated, a partial plan for displacing an object is found, and the first pick-and-place of the plan is executed. We re-sense and re-plan after the first pick-and-place, similar to model predictive control [74]. In this section, each component is briefly described. Regrasping under segmentation and completion uncertainty – the main contribution of this chapter – is detailed in Section 6.3.

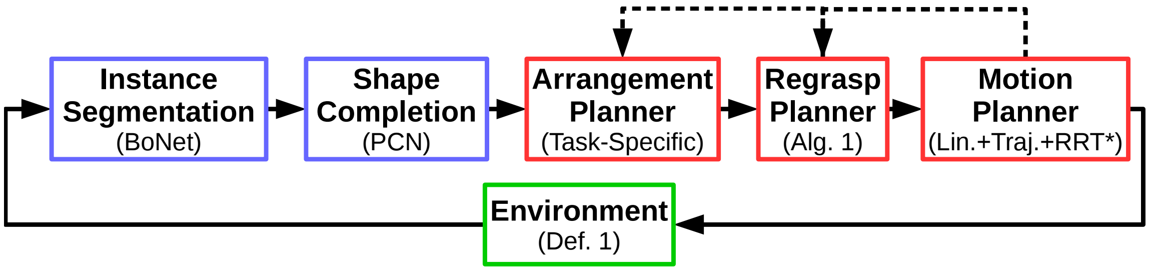

Instance Segmentation (BoNet)
Shape Completion (PCN)
Arrangement Planner (Task-Specific)
Regrasp Planner (Alg. 1)
Motion Planner (Lin.+Traj.+RRT*)
Environment (Def. 1)

Figure 6.1: Diagram of our system architecture. Green represents the environment, blue the perceptual modules, and red the planning modules. Dashed arrows are followed up to a number of times if no plan is found.

6.2.1 Perception

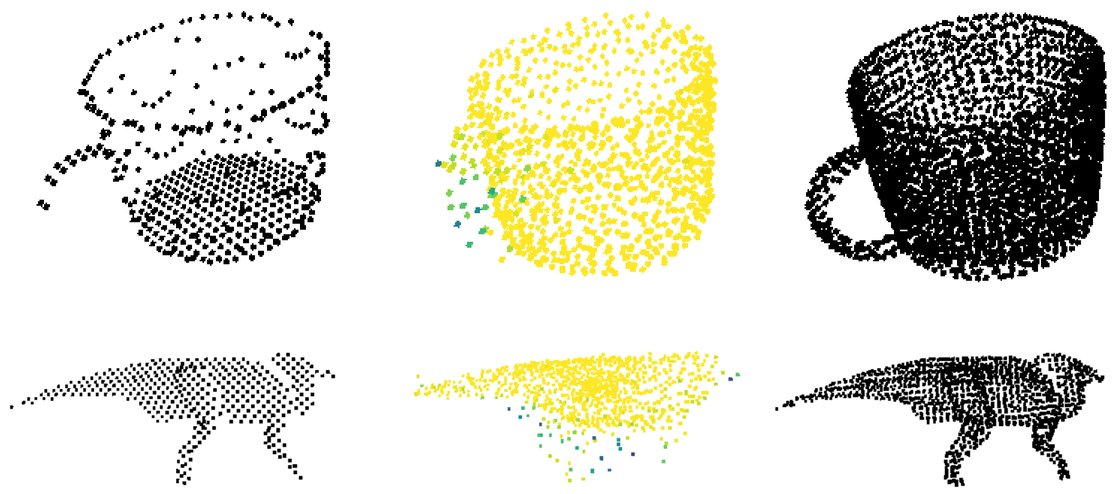
The purpose of the perceptual modules is to reconstruct the geometry of the scene so we can apply geometric planning algorithms. Additionally, they must quantify their own uncertainty so plans unlikely to succeed can be avoided. For both instance segmentation and shape completion, we have chosen point clouds as the input/output representation of objects. A point representation consumes less memory than uncompressed voxel grids, enables efficient planning, and, from our previous experience, exhibits good simulation-to-real domain transfer [20, 22, 23].

Object instance segmentation

The input to the segmentation module is a point cloud with n points, $C \in \mathbb{R}^{n \times 3}$. The output is a point cloud for each object, $\{C_i \in \mathbb{R}^{n_i \times 3}\}_{i=1}^{n_{obj}}$ with $\sum_{i=1}^{n_{obj}} n_i \leq n$ (points with a U_i value below a threshold are not assigned a segment), and segmentation uncertainties $\{U_i \in \mathbb{R}^{n_i}\}_{i=1}^{n_{obj}}$. Although any object instance segmentation method with this interface can be used in the proposed architecture, our implementation uses BoNet [126]. BoNet produces an $n \times K$ matrix, where K is a predefined maximum number of objects, and each row is a point's probability distribution over object ID. U_i is the max of the ith row, which is interpreted as the estimated probability each point is correctly segmented. Architectural details of BoNet are in Appendix C.

Shape completion

The input to the shape completion module is the point cloud of the ith object segment $C_i \in \mathbb{R}^{n_i \times 3}$, and the output is a point cloud $\bar{C}_i \in \mathbb{R}^{\bar{n}_i \times 3}$ that is a dense sampling of points on all object faces, including faces not visible to the sensor. We also require a completion uncertainty estimate for each point, $\bar{U}_i \in \mathbb{R}^{\bar{n}_i}$. Although any shape completion method with this interface can be used in the proposed architecture, our implementation uses a modified version of PCN [128]. PCN consists of an encoder (two PointNet layers [90]) and a decoder (three fully connected, inner product layers) (details in Appendix C). We augmented the original version of PCN with a second decoder for uncertainty estimates. In particular, the uncertainty decoder is trained using a binary cross-entropy loss to predict the probability each point is within Euclidean distance $\beta \in \mathbb{R}_{>0}$ of the nearest ground truth point. So the uncertainty values should be interpreted as the estimated probability each completed point is accurate. Example completions are shown in Figure 6.2b.



(a) Observed cloud. (b) Complete cloud. (c) Ground truth.

Figure 6.2: Shape completions with PCN. Yellow represents high $\bar{U}$ values (near 1) and blue represents low $\bar{U}$ values (near 0.5). Additional examples are shown in Appendix C.

6.2.2 Planning

We use a three-level planner, similar to Wan et al. [113]. The output action at each level has a different level of abstraction: the arrangement planner produces object displacements, the regrasp planner produces a pick-and-place sequence, and the motion planner produces an arm trajectory.

Arrangement planner

The purpose of the arrangement planner is to determine a set of goal poses for the objects that satisfies a given task. An example arrangement planner for the task of placing a bottle upright onto a coaster samples goal poses for the bottle as follows: align the longest axis with gravity, center the widest end just over the coaster, and apply random rotations about gravity. While this is in general a difficult problem, as there is uncertainty in the objects' shapes and locations, we do not dwell on it here. Instead, we implement an arrangement planner specific to each task and assume the output goal set satisfies the task.

The input to the arrangement planner is a list of completed clouds, $\bar{C}_1, \dots, \bar{C}_{n_{obj}}$, and the output is a set of triples $\{(T, c, i)_j\}_{j=1}^{n_{goal}}$, where T is a goal pose for the ith object and c is an associated goal cost. Each object can have zero or more goals associated with it. Compared to having one goal for one object (e.g. [109]), it is better if the arrangement planner produces multiple goals for multiple objects, since the minimum regrasp planning cost can only stay the same or decrease as additional goals are considered. The goal cost c specifies preference between different goals. For example, for bin packing, c could be equal to the height of the pile in the bin. The same interface is implemented by each task-specific arrangement planner.

Regrasp planner

The regrasp planner takes in the triples from the arrangement planner and produces a sequence of picks and places, i.e., effector poses, that displaces one object. If a regrasp plan is not found, more goals can be requested from the arrangement planner (as indicated by dashed lines in Figure 6.1). The main contribution of this chapter concerns the regrasp planning cost, which is detailed in Section 6.3.

Motion planner

The motion planner finds a continuous motion between picks and places. Any off-the-shelf motion planner will do. We use a three-level planner that first attempts a linear motion, then Trajopt [96], and then RRT* with timeout [42]. If no motion plan is found, the regrasp planner can be resumed from where it left off, but marking the infeasible section so the same solution is not found again.

6.3 Regrasp Planning Under Uncertainty

The problem of regrasp planning under uncertainty is: given a set of completed objects with corresponding perceptual uncertainty and given a set of goal poses for each object, find a sequence of antipodal grasps and stable places maximizing the probability of displacing an object to a goal pose. Regrasps are needed due to kinematic constraints: the grasps at the object's current pose may all be in collision or out of reach at the object's goal poses. In this case, a number of temporary places (i.e., non-goal places) are needed. Our regrasp planner (Algorithm 6.1) extends Tournassoud et al.'s [109] to handle multiple goals for multiple objects, arbitrary additive costs, and discrete grasp/place sampling. Related planners (e.g, [2, 1, 80, 48]) could also have been adapted to the purpose: the main point is to incorporate segmentation and shape completion uncertainty into the cost.

A key part of Algorithm 6.1 is the regrasp graph, RG. An example regrasp graph is shown in Figure 6.3. The nodes of the *regrasp graph* are grasp-place combinations. Edges are between nodes sharing either a grasp or a place: when the object is

Algorithm 6.1: Regrasp planner: run for each object.

```
Input  : Number of sampling iterations $N$, completed cloud $\bar{C}$, segmentation
         uncertainty $U$, shape completion uncertainty $\bar{U}$, goal poses and costs
         $\{(T,c)_j\}_{j=1}^{n_{goal}}$, and $costLowerBound$.
Output: A list of gripper poses, $plan$, alternating between grasp and place
         poses.
1 $RG \leftarrow []$
2 for $i \leftarrow 1, \dots, N$ do
3     $G, gc \leftarrow$ SampleGrasps$(\bar{C}, U, \bar{U})$
4     $P, pc \leftarrow$ SampleTemporaryPlaces$(\bar{C}, U, \bar{U})$
5     $RG \leftarrow$ UpdateRegraspGraph$(RG, \{(T,c)_j\}_{j=1}^{n_{goal}}, G, gc, P, pc)$
6     $plan, cost \leftarrow$ A$^*(RG)$
7     if $cost \le costLowerBound$ then break
8 return $plan$
```

grasped, a change in place is allowed, and when the object is placed, a change in grasp is allowed. The regrasp graph is organized into a matrix where rows refer to grasps and columns refer to places. At any point in time, either a row change is allowed to change the grasp, keeping the place fixed, or a column change is allowed to change place, keeping the grasp fixed [109]. To Tournassoud et al.'s regrasp graph we add costs: matrix values are the sum of the corresponding grasp and place costs if the grasp-place combination is feasible (i.e., there is a collision-free inverse kinematics (IK) solution) and infinity otherwise.

Algorithm 6.1 works as follows. For N steps, additional grasps (G with costs gc) and temporary places (P with costs pc) are randomly sampled. Given the shape completion $\bar{C}$, grasp samples are constrained to satisfy the geometric antipodal conditions (Definition 2.4), and place samples are constrained to satisfy the stability conditions (Definition 2.5). Grasp and place costs are computed using the perceptual uncertainty estimates U and $\bar{U}$ from Section 6.2.1. (More on how these costs are computed is in the next section.) The function `UpdateRegraspGraph` adds a row for each sampled grasp and a column for each sampled place to RG. Then, it checks IK and collisions for the new grasp-place combinations and sets RG values for infeasible combinations to infinity. Finally, A* with a consistent heuristic finds an optimal pick-and-place sequence, given the current samples [27]. In the next section we define the cost function used by A* and give different ways of calculating grasp and place costs. Algorithm 6.1 is run in parallel for each object that has at least one goal pose.

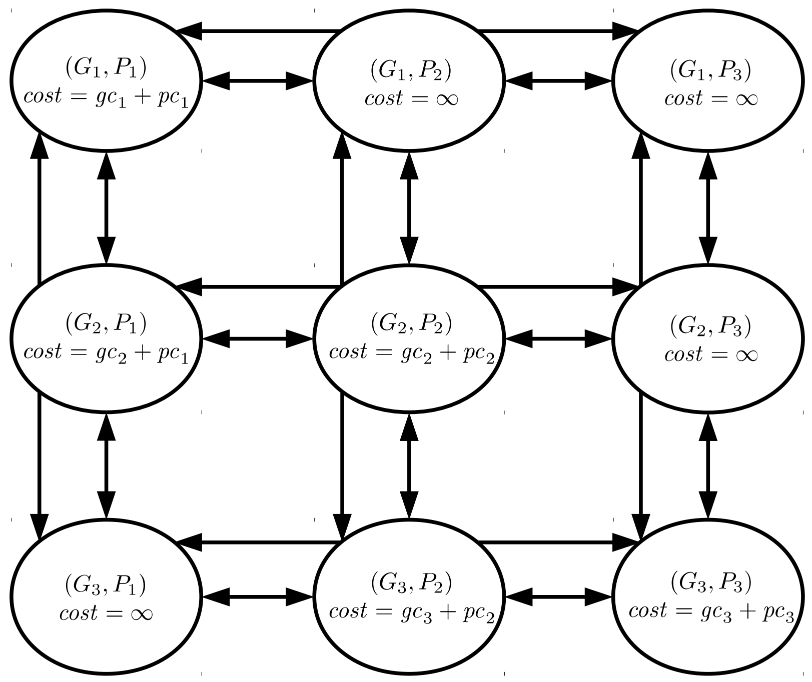

(G_1, P_1) $cost = gc_1 + pc_1$
(G_1, P_2) $cost = \infty$
(G_1, P_3) $cost = \infty$
(G_2, P_1) $cost = gc_2 + pc_1$
(G_2, P_2) $cost = gc_2 + pc_2$
(G_2, P_3) $cost = \infty$
(G_3, P_1) $cost = \infty$
(G_3, P_2) $cost = gc_3 + pc_2$
(G_3, P_3) $cost = gc_3 + pc_3$

Figure 6.3: An example regrasp graph with three grasps, $G = \{G_1, G_2, G_3\}$ and three places, $P = \{P_1, P_2, P_3\}$. The cost of arriving at a reachable grasp-place combination is the sum of the grasp and the place costs, $gc_i + pc_j$. The cost of arriving at an unreachable grasp-place combination is infinity. Edges allow changing row (grasp) or column (place) but not both. In this example, the lowest cost solution for moving the object from P_1 to P_3 (assuming positive costs) is to use G_2 to move the object to P_2, regrasp the object using G_3, and then place the object at P_3.

6.3.1 Maximize probability of regrasp plan execution success

The aim is to choose a regrasp plan that maximizes the joint probability each grasp is antipodal and each temporary place is stable, i.e., maximize Equation 6.1, where G_i is the event the ith grasp is antipodal, P_i is the event the ith place is stable, and m is total number of picks or places. We assume each G_i and P_i is independent, which gives the right side of Equation 6.1:

$$\Pr(G_1, P_1, \ldots, P_{m/2}) = \Pr(G_1) \cdots \Pr(P_{m/2}). \tag{6.1}$$

Taking the log and abbreviating $\Pr(G_i)$ as g_i and $\Pr(P_i)$ as p_i yields Equation 6.2:

$$\log\left[\Pr(G_1,\ldots,P_{m/2})\right] = \sum_{i=1}^{m/2} \log(g_i) + \sum_{i=1}^{m/2} \log(p_i). \tag{6.2}$$

Negating Equation 6.2 results in a non-negative, additive cost: the form required by A*. The cost of taking a grasp/place action corresponds to an edge cost in the regrasp graph.

Our final cost function is Equation 6.3, which adds two additional terms: the plan length $m \in \mathbb{N}$ and the task cost $c \in \mathbb{R}$ associated with the goal placement (from the arrangement planner, Section 6.2.2). $w_1, \ldots, w_4 \in \mathbb{R}_{>0}$ are given trade-off parameters. To complete the description, we next look at different ways of estimating g_i and p_i.

$$J = w_1 m - w_2 \sum_{i=1}^{m/2} \log(g_i) - w_3 \sum_{i=1}^{m/2} \log(p_i) + w_4 c \tag{6.3}$$

6.3.2 Probability grasps are antipodal and places are stable

In this section, we introduce four different ways (including GQ, MC, CU, and SP) to estimate the probability the ith grasp in the regrasp plan is antipodal, g_i, and three different ways (including MC, CU, and SP) to estimate the probability the ith placement in the regrasp plan is stable, p_i .

Grasp quality (GQ)

One way to estimate g_i is via a measure of "robustness" of the grasp to small perturbations in the nominal shape completion. For antipodal grasps, Murray et al. suggest choosing grasps where the line between contacts is inside and maximally distant from the edges of both friction cones ([79] p. 233). This way, a grasp will satisfy the geometric antipodal conditions under small perturbations to the object's shape.

We place this idea into our probabilistic framework. For both contact points $j = 1, 2$, let $\theta_j \in [0, \pi]$ be the angle between the object's surface normal and the line connecting both contact points. In light of the definition of an antipodal grasp (Definition 2.4 in Chapter 2), if θ_j exceeds half the (given) angle of the friction cone θ_{max}, for either contact, the grasp will not be antipodal. θ_j is unknown because the object's shape is unknown.

Given an object's shape completion, let n_j be an estimate the object's surface normal at the jth contact point, and let b_j be an estimate of the unit-length ray extending in the direction of the line connecting both contact points in the direction of the finger making contact. We assume θ_j is distributed according to a truncated normal distribution with mode $\mu_j = \arccos(b_j \cdot n_j)$ and given scale σ (Figure 6.4, left). The probability b_j lies in the friction cone is then $\Pr(\theta_j \leq \theta_{max}) = F(\theta_{max}; \mu_j, \sigma, 0, \pi)$, where F is the cumulative density function of the truncated normal distribution (Figure 6.4, right). We make the simplifying assumption that this probability is independent between contacts, giving Equation 6.4.

$$g_i = \prod_{j=1}^{2} F(\theta_{max}; \mu_j, \sigma, 0, \pi) \tag{6.4}$$

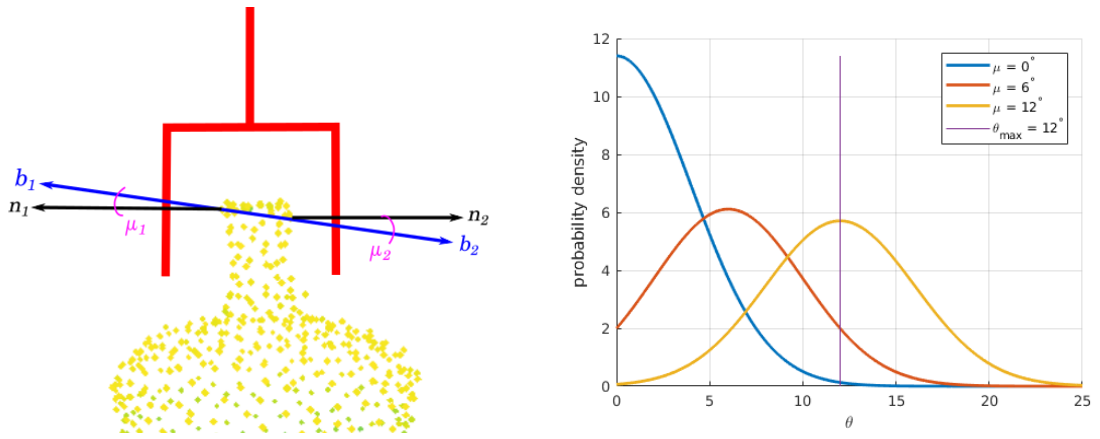


Figure 6.4: **Left**. Relationship between the surface normals n_1 and n_2 (black) at the contact points, the line b_1 and b_2 (blue) connecting the contact points, and the angles μ_1 and μ_2 (magenta) not to exceed θ_{max}. These quantities are estimates as they depend on the (uncertain) shape completion. **Right**. Truncated normal probability density functions with $\mu = 0°$ (blue), $6°$ (red), and $12°$ (yellow), and $\sigma = 0.10$. $\Pr(\theta \leq \theta_{max})$ is the area under each curve up to the vertical line at $\theta_{\max} = 12°$. Notice that $\Pr(\theta \leq \theta_{max})$ can never be 1 (so every grasp has a non-zero cost) and reduces as μ increases.

The effect of the GQ estimator is to choose grasps that are as centered as possible in both friction cones, given the estimated object shape. The scale parameter σ makes the trade-off between regrasp plan length and centering of grasps: small σ prefers centered grasps over short plans and large σ prefers short plans over centered grasps.

Monte Carlo (MC)

Another approach is to estimate g_i and p_i via segmentation and completion samples, as was done for grasping under shape uncertainty [44, 66, 51, 64]. The idea is to randomly generate multiple segmentations then completions and set g_i (or p_i) equal to the average grasp (place) antipodal (stability).

Let $\Pr(\bar{C}_i|C)$, for $i = 1, \dots, n_{obj}$, be the probability the shape completion is $\bar{C}_i \in \mathbb{R}^{\bar{n}_i \times 3}$ given the input point cloud is $C \in \mathbb{R}^{n \times 3}$. We use the point-wise uncertainty estimates U_i and $\bar{U}_i$ from Section 6.2.1 to sample from the probability distribution $\Pr(\cdot|C)$ as follows.

To sample a shape: (a) sample a segmentation point-wise using the segmentation mask and (b) compute the shape completion given this segmentation. To sample a segmentation (a), the object ID for each point is independently sampled from the probability distributions given by the segmentation matrix. (To reduce noise, we only sample points whose U-value is below a threshold.) To sample a shape completion (b), assume the ith point's offset from the nominal point is i.i.d. $\sim \mathcal{N}(0, \sigma_i^2)$. Since $\bar{U}_i$ is the estimated probability the point is offset no more than β, the standard deviation of the point's offset, σ_i, is derived from the Gaussian cumulative distribution function Φ as in Equation 6.5 to 6.6 (where the full derivation is shown in Appendix C). Then, for each point in the completion, (b.1) sample a direction uniformly at random and (b.2) sample an offset along this direction from a normal distribution with 0 mean and standard deviation given by Equation 6.6. Example samples applying this method to a scene with six objects are shown in Figure 6.5.

$$\bar{U}_i = 1 - 2\Phi(-\beta; 0, \sigma_i^2) \tag{6.5}$$

$$\sigma_i = \frac{\beta}{\sqrt{2}\,\mathrm{erf}^{-1}(\bar{U}_i)} \tag{6.6}$$

g_i is estimated as #antipodal/M and p_i is estimated as #stable/M where M is the number of shape samples and #antipodal is the number of shapes for which the ith grasp is antipodal and #stable is the number of shapes for which the ith place is stable.

Contact uncertainty (CU)

Computing g_i and p_i with MC is computationally expensive if M is large. This motivates considering uncertainty only at contact points. For instance, placing an object on its unseen, predicted geometry could likely be unstable, so we penalize

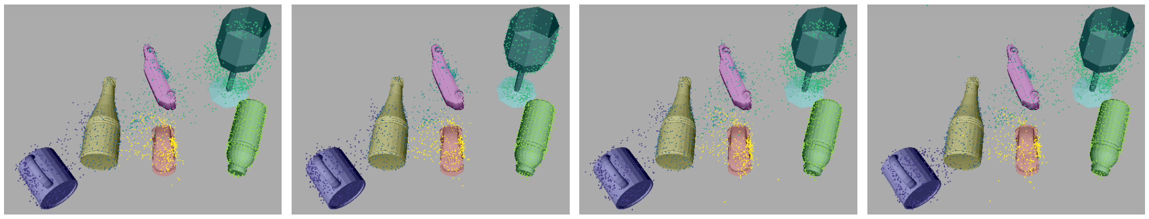


Figure 6.5: Four Monte Carlo point cloud segmentation-completion samples for a scene with six objects. Each color represents a different object segment. Since the green bottle on the bottom-right has little variation between samples, grasps and places for this object will be preferred.

grasps/places on uncertain object parts. The same idea is behind penalizing high-variance grasp contacts [66, 58].

Formally, suppose we estimate $\Pr(V_i)$, where V_i is the event the ith point in the completed cloud is segmented correctly. Suppose we also estimate $\Pr(\bar{V}_i|V_i)$, where $\bar{V}_i$ is the event the ith point in the completed cloud is within Euclidean distance β of a ground truth point, given the point is segmented correctly. Here, we assume whether a grasp (place) is antipodal (stable) depends only on V_i and $\bar{V}_i$ for each predicted contact point (where contacts are explained in Figure 6.6). Assuming independence between contacts, g_i and p_i are estimated via Equation 6.7 and 6.8.

$$g_i = \Pr(\bar{V}_l|V_l)\Pr(V_l)\,\Pr(\bar{V}_r|V_r)\,\Pr(V_r) \tag{6.7}$$

$$p_i = \prod_{i=1}^{3} \Pr(\bar{V}_{t^i}|V_{t^i})\,\Pr(V_{t^i}) \tag{6.8}$$

The uncertainty values from PCN ($\bar{U}_i$ in Section 6.2.1) estimate $\Pr(\bar{V}_i|V_i)$ – the probability ith point in the completed cloud is within Euclidean distance β of a ground truth point given the point is segmented correctly. Estimating $\Pr(V_i)$ – the probability the ith point in the shape completion is segmented correctly – from the uncertainty values from BoNet (U_i in Section 6.2.1) is less straight-forward. This is because there is no one-to-one correspondence between segmented points and completed points. (In fact, there are usually more completed points than segmented points.) We assign the nearest point in the segmented cloud to each point in the completed cloud, i.e., use U_i as an estimate of V_j where the ith point in the segmented cloud is nearest to the jth point in the completed cloud.

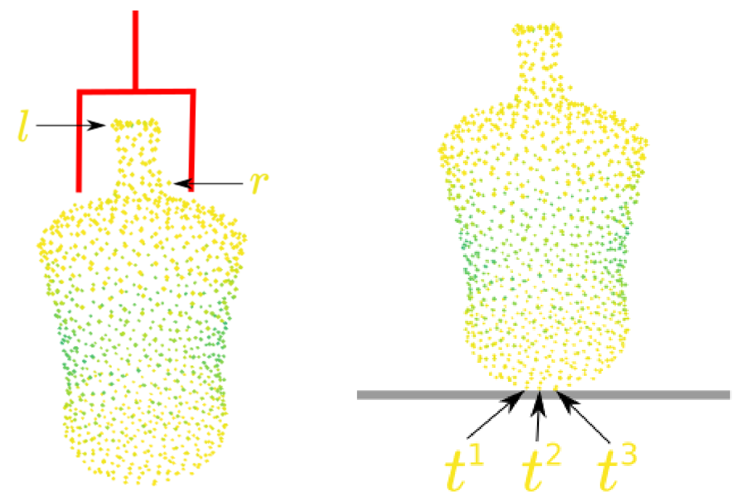


Figure 6.6: **Left**. For an antipodal grasp (shown in red), there are at least two contact points, l and r. **Right**. For a stable placement on a flat surface, there are at least three contact points, t^1, t^2, and t^3. Colors represent estimates of $\Pr(\bar{V}_i|V_i)\Pr(V_i)$, where yellow represent higher probabilities.

Success prediction (SP)

g_i and p_i can also be directly estimated with a neural network. Here, we encode grasps as the points from the shape completion, $\bar{C}$, inside the gripper's closing region with respect to the gripper's reference frame (Figure 6.7), as in [59]. For places, the completed cloud, $\bar{C}$, is transformed to the place pose and translated with the bottom-center of the cloud at the origin (Figure 6.8).

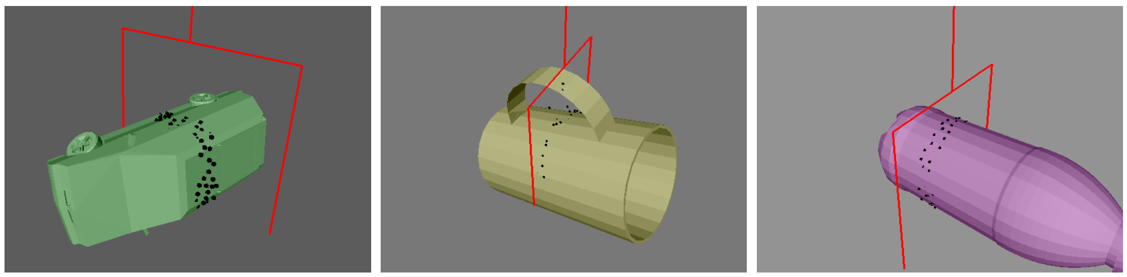


Figure 6.7: The SP grasp descriptor is the completed points in the gripper's closing region with respect to the gripper's reference frame. Liang et al. also employ this idea for ranking grasps [59].

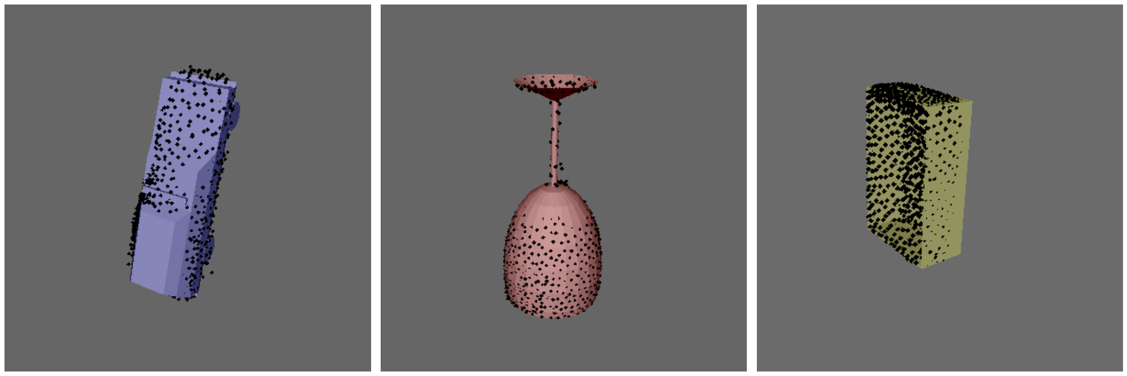


Figure 6.8: The SP place descriptor is the completed points with respect to the target place's orientation and with the bottom-center at the origin.

Training data is generated in simulation as follows. Antipodal grasps are sampled from the shape completion. Then, training labels are generated by testing the antipodal condition for the same grasps on the ground truth point cloud. Similarly, stable places are sampled from the shape completion, and training labels are generated by testing the stability conditions for the same places on the ground truth point cloud.

For network architecture, we use PCN [128] with a single output with sigmoid activation, trained with the binary cross-entropy loss. (A diagram of PCN is in Appendix C.) We train with 1 million examples in mini-batches of 32 using Adam [45] with base learning rate 0.0005. Grasps are trained for 100 epochs, and places are trained for 50 epochs.

6.4 Experiments

We ran experiments in simulation and the real world to compare the different ways in Section 6.3.2 for accounting for object shape uncertainty in regrasp planning. The hypothesis is that a cost using perceptual uncertainty estimates will choose regrasp plans that are executed successfully more often on average compared to common methods which do not. GQ is considered a baseline as it does not use perceptual uncertainty estimates: it only aims for robustness to small perturbations in object shape. We also compare to two other baselines: *no cost*, which takes the first regrasp plan found, and *step cost*, which includes the step cost term only ($w_1 = 1$ in Equation 6.3). The step cost appears almost exclusively in the regrasping literature, e.g., [109, 2, 1, 113].

6.4.1 Setup

The experimental environment is illustrated in Figure 6.9, left. We evaluate the proposed system on the following tasks:

1. **Canonical placement.** Place any 1 of 5 objects into a goal pose. The arrangement planner is an oracle which consistently gives the same goal pose for an object. The purpose is to analyze regrasp performance independent from arrangement planner errors.

2. **Bin packing.** Place 6 objects into a box minimizing packing height. This is known as the 3D irregular-shaped open dimension problem [116].

3. **Bottle arrangement.** Place 2 bottles upright onto 2 coasters (from our prior work [22, 23]).

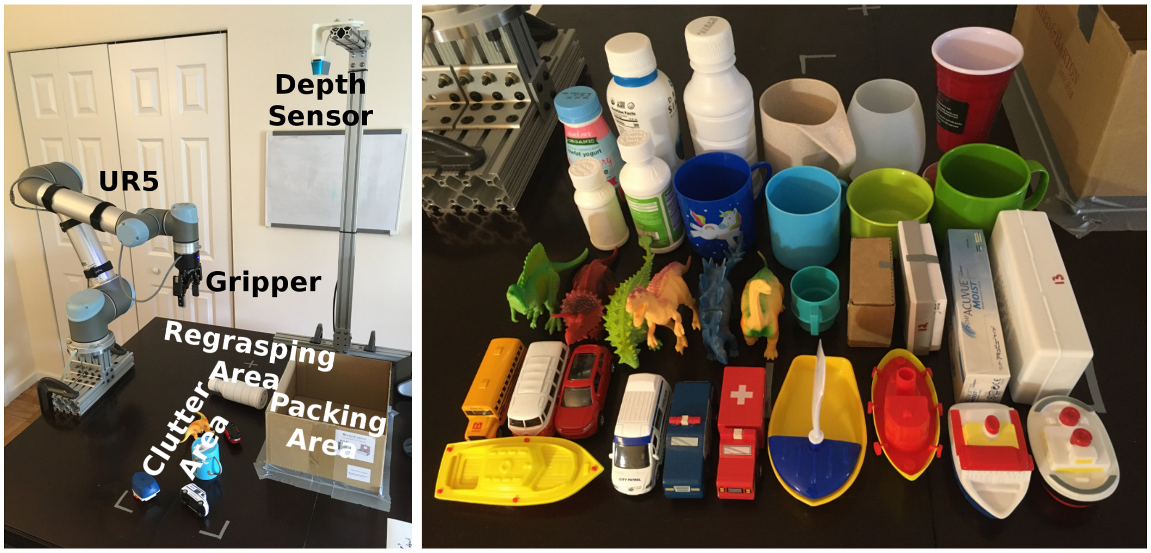


Figure 6.9: **Left**. Environment includes a UR5 arm, a Robotiq 85 gripper, and a Structure depth sensor. **Right**. 34 same-category novel objects used for real-world packing experiments.

6.4.2 Simulation experiments

The environment is simulated by OpenRAVE [10] using 3DNet objects [122]. Objects are partitioned into *Train* and *Test* sets of the same category (boat, bottle, box, car, dinosaur, mug, and wine glass) and the *Test-2* set for novel categories (airplane, bowl, and stapler). A depth sensor, situated above the objects, captures a point cloud which is then passed into BoNet for segmentation. Grasps succeed if (a) exactly 1 object intersects the hand closing region, (b) the antipodal condition (Definition 2.4) with 24° friction cone is met, and (c) the robot is collision-free. Places are stable if the conditions of Definition 2.5 are met.

Evaluation

We evaluate *place execution success rate* – the proportion of regrasp plans where an object is moved to a goal (i.e., no grasp failures) – and *temporary place stability rate* – the proportion of temporary places that are stable. These metrics are fast to compute and indicate how well the regrasp plans succeed in placing objects. It is also important to consider *plan length* – the number of grasps and places – and *regrasp planning time* – the amount of time spent by the regrasp planner – as these affect how long it takes to complete the task. We use a 1-sided, same-variance, unpaired t-test to decide if one method significantly outperforms another. (If $p \leq 0.05$, the difference is considered to be significant.) Green cells in the tables indicate significant improvements over the best-performing baseline.

Perception ablation study

We quantify the potential benefit of accounting for uncertainty for the bin packing task. We evaluate performance with ground truth perception (GT Seg. & Comp.), imperfect completion (GT Seg.), imperfect segmentation and completion (Percep.), and without shape completion (GT Seg. & No Comp.) "Imperfect" means the objects' segmentation/completion is estimated from the observed point cloud. Step and task costs are used, i.e., $w_1 = w_4 = 1$ and $w_2 = w_3 = 0$ in Equation 6.3, where the task cost, c, is the estimated final packing height in centimeters.

	GT Seg. & Comp.	GT Seg. (Train)	GT Seg. (Test)	Percep. (Train)	Percep. (Test)	GT Seg. & No Comp.
Place Execution Success	0.929 ± 0.008	0.767 ± 0.013	0.747 ± 0.013	0.718 ± 0.014	0.710 ± 0.014	0.508 ± 0.046
Regrasp Plan Found	0.957 ± 0.006	0.882 ± 0.009	0.939 ± 0.007	0.879 ± 0.009	0.941 ± 0.007	0.100 ± 0.009
Temporary Place Stable	1.000 ± 0.000	0.769 ± 0.122	1.000 ± 0.000	0.828 ± 0.071	0.826 ± 0.081	0.500 ± 0.500
Regrasp planning time (s)	35.62 ± 1.103	38.46 ± 1.115	38.68 ± 1.141	35.76 ± 1.059	35.05 ± 1.077	15.86 ± 1.482

Table 6.1: Perception ablation study for bin packing. Showing average ± standard error over 200 episodes.

Results (shown in Table 6.1) are as expected. A clear drop in performance is observed as perception becomes imperfect (down 18% for imperfect completion and another 4% for imperfect segmentation). Thus, a large source of error is due to perception, so there is space for improvement by accounting for perceptual uncertainty. Without shape completion, regrasp planning is crippled (regrasp plan found rate drops from 94.1% for Percep. Test to just 10.0%). This is because insufficient grasp and place samples are found to displace objects. Perception ablation studies for bin packing Test-2 and bottle arrangement show similar trends (Appendix C).

Regrasp cost comparison

We test the hypothesis that a method estimating perceptual uncertainty (either MC, CU, or SP from Section 6.3.2) selects regrasp plans that execute successfully more often on average than the baselines (no cost, step cost, and GQ).

Results for the canonical task are shown in Table 6.2. In this case, MC, CU, and SP methods have significantly higher temporary place stability rates than no cost (which happened to do better than step cost and GQ). There is no doubt SP outperforms GQ for place execution success rate ($p = 9.7 \times 10^{-9}$). On the other hand, no cost has the shortest regrasp planning time, and step cost has the shortest regrasp plans on average. This is expected, as no cost optimizes for planning time and step cost optimizes for plan length.

	No Cost	Step Cost	GQ	MC	CU	SP
Place Execution Success	0.727 ± 0.010	0.777 ± 0.009	0.856 ± 0.008	0.852 ± 0.008	0.830 ± 0.008	**0.913** ± 0.006
Temporary Place Stable	0.785 ± 0.015	0.623 ± 0.067	0.700 ± 0.031	0.852 ± 0.022	0.885 ± 0.029	**0.967** ± 0.012
Plan Length	3.061 ± 0.029	**2.079** ± 0.009	2.273 ± 0.016	2.286 ± 0.016	2.157 ± 0.013	2.239 ± 0.015
Regrasp planning time (s)	**2.462** ± 0.061	6.413 ± 0.353	62.19 ± 0.326	117.6 ± 0.724	54.88 ± 0.366	61.54 ± 0.900

Table 6.2: Cost comparison for the **canonical** task. Showing average ± standard error over 2,000 episodes. The left three columns (Step Cost, No Cost, and GQ) are baselines. The right three columns (MC, CU, and SP) are uncertainty-aware costs. A cell is highlighted green iff the corresponding value is significantly higher (i.e. $p \leq 0.05$) than the highest baseline value. While planning time is less for no cost and plan length is shorter for step cost, SP has higher place execution success and temporary place stable rates.

Results for bin packing are shown in Table 6.3. SP again performs best in terms of place execution success and temporary place stability. However, for bin packing, we do not see a significant improvement for place stability over the step cost, but this is because regrasps are rare with the step cost, obscuring the significance of the results. We again note that no cost has the shortest regrasp planning time and step cost has the shortest regrasp plans on average.

	No Cost	Step Cost	GQ	MC	CU	SP
Place Execution Success	0.651 ± 0.013	0.725 ± 0.012	0.748 ± 0.012	0.756 ± 0.012	0.712 ± 0.013	**0.779** ± 0.012
Temporary Place Stable	0.784 ± 0.024	0.857 ± 0.097	0.845 ± 0.030	0.904 ± 0.028	0.848 ± 0.054	**0.959** ± 0.018
Plan Length	2.665 ± 0.031	**2.038** ± 0.008	2.293 ± 0.021	2.222 ± 0.019	2.105 ± 0.013	2.233 ± 0.019
Regrasp planning time (s)	**4.904** ± 0.230	7.201 ± 0.393	84.56 ± 0.827	90.10 ± 0.892	72.00 ± 0.835	86.61 ± 1.040

Table 6.3: Cost comparison for the **bin packing** task. Showing average ± standard error over 230 episodes. The left three columns (Step Cost, No Cost, and GQ) are baselines. The right three columns (MC, CU, and SP) are uncertainty-aware costs. A cell is highlighted green iff the corresponding value is significantly higher (i.e. $p \leq 0.05$) than the highest baseline value. As with the canonical task, planning time is less for no cost and plan length is shorter for step cost, but SP has higher place execution success. The Temporary Place Stable rate comparison is less meaningful since the standard error for the best-performing baseline, Step Cost, is high.

For both packing and canonical tasks, SP significantly outperforms all baselines in terms of place execution success, which supports the hypothesis. (This is also true for test objects from novel categories: see Appendix C).

6.4.3 Real world experiments

We seek to (a) verify the importance of uncertainty seen in simulation results and (b) see if the perceptual components, trained with simulated data, work well with real sensor data. For these experiments, same-category novel objects are used (Figure 6.9, right).

To answer part (a), a regrasp cost comparison for bin packing is shown in Table 6.4. Both MC and SP methods significantly outperform the step cost (which outperforms GQ). Example packing and regrasp sequences are shown in Figure 6.10. To answer part (b), no domain transfer was needed for bin packing. For bottles, BoNet (but not PCN) overfit to simulation data. This problem was mitigated by adding simulated sensor noise.

	Step Cost	GQ	MC	SP
Place Success Rate	0.839 ± 0.027	0.833 ± 0.028	0.911 ± 0.021	**0.917** ± 0.021
Grasp Success Rate	0.883 ± 0.023	0.866 ± 0.024	**0.947** ± 0.016	0.933 ± 0.017
Grasp Attempts	196	201	207	210
Number of Regrasps	**17**	21	27	30

Table 6.4: Packing performance on the real robot. Showing average ± standard error over 30 episodes, each with 6 objects.

We also compare bottle arrangement performance to our previous method, which uses RL to learn a pick-and-place policy [23]. Many of the same bottles as before are included, but 4/15 of them are more challenging. Two of the bottles are difficult to distinguish orientation (size of tops near size of bottoms), and two are near the 8.5 cm gripper width. Results are shown in Table 6.5. With the proposed method, all places are correct. Only the grasp success rate is lower than before, but all 3 grasp failures are with the wider bottles. Overall, we conclude the modular approach performs better (80% vs. 67% task success rate).[2]

	Shape Completion	HSA [23]
Number of Objects Placed	1.800 ± 0.074	1.667 ± 0.088
Task Success Rate	0.800 ± 0.074	0.667 ± 0.088
Grasp Success Rate	0.948 ± 0.029	0.983 ± 0.017
Place Success Rate	1.000 ± 0.000	0.900 ± 0.040

Table 6.5: Bottles performance for the proposed method versus [23]. Showing average ± standard error over 30 episodes.

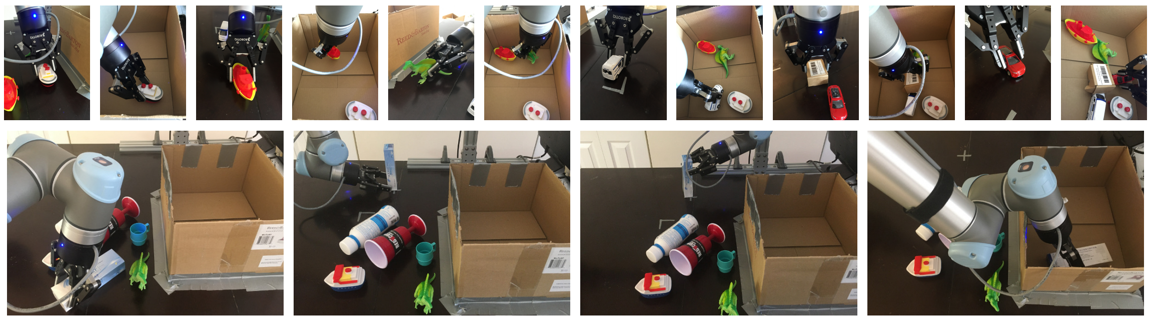


Figure 6.10: **Top**. Example packing sequence. **Bottom**. Example situation requiring a regrasp. Since the box is initially right-side-up, and the goal is to place it bottom-side-up, a regrasp is needed to flip the box over.

6.5 Discussion

Object instance segmentation and shape completion enable use of existing planning algorithms for pick-and-place of sensed objects. However, perceptual errors are still a

[2]Source code, additional results, and a video of some of the real robot experiments are available at `https://github.com/mgualti/GeomPickPlace`.

major source of failure. To compensate for this, we compare different planning costs modeling probability of successfully executing a regrasp plan. Results show the SP cost, which uses separate networks to predict grasp/place success, consistently performs nearly as well as or outperforms all other costs. We attribute this to: (a) unlike baseline and GQ costs, SP can detect when perception is uncertain based on the distribution of perceived points; (b) unlike the CU cost, which considers uncertainty only at contact points, SP considers uncertainty at many points; and (c) unlike the MC cost, which requires sampling and evaluating multiple shapes, SP is computationally cheaper. On the other hand, when shape completion is accurate, e.g., when trained with one category like bottles, the step cost is a reasonable choice as planning and execution is faster than SP.

6.5.1 Limitations

We note some limitations with this approach. First, the regrasp planner is much slower with a more sophisticated cost function than the step cost. This is because the step cost can exit the sampling loop when a two step plan is found, which occurs often in our experiments, while the other costs have no easy stopping criterion. Second, segmentation and completion accuracy is much lower with novel-object categories (cf. Appendix C). Third, integrating additional views to decrease uncertainty is an important aspect not considered.

6.5.2 Related Work

This chapter is based on material first published in [24].

Deep shape completion for robotic manipulation

Varley et al. may have been the first to use a deep neural network to predict an object's 3D shape for the purpose of planning grasps [111]. Their network is a 3D CNN with a 3D occupancy grid input/output. The output occupancy grid is post-processed into a mesh model of the object, which is used by GraspIt! [68] to plan grasps. As they predicted, "This ability to infer occluded geometries can be applied to a multitude of robotic tasks." Instead of relying on a deep network, Mitash et al. relied on integrating multiple views of an object to complete its shape for regrasping [70]. They demonstrated inserting an object into a small opening in the table. Uncertainty was handled conservatively by assuming the entire unobserved region was part of the object (and thus would be too large for the slot until more views were integrated). We appear to be amongst the first to investigate deep shape completion for pick-and-place and regrasping.

Grasping under uncertainty

Many of the cost functions here were inspired by grasp planning under uncertainty. Mahler et al. compared a probabilistic model (based on the variance of the GPIS at contact points, analogous to the CU method) to an MC approach [66]. The MC approach did better but has higher computational cost. Lundell et al. represented objects as voxels, used a deep network to complete objects, and performed MC sampling using dropout [64]. Uncertainty was more important for novel objects than for training objects. The SP cost was inspired by our prior work with GPD [21, 84] and Liang et al.'s PointNetGPD [59], which builds off of GPD by taking in point clouds as input instead of depth images.

Part IV

Conclusion

…the Scientist, like the Pilgrim, must wend a straight and narrow path between the Pitfalls of Oversimplification and the Morass of Overcomplication.

Richard Bellman in *Dynamic Programming*

Chapter 7

Discussion

We studied the pick-and-place problem where objects are partially visible and where objects' shapes are not given. Two fundamentally different approaches were taken, including policy learning and modular, perception-planning architectures. While each approach required different innovations for effective pick-and-place of partially visible, novel objects, they both required innovations to efficient sampling of 6-DoF effector poses for candidate grasp and place actions. In this chapter we begin by discussing this common challenge. Afterward, takeaways specific to each approach and research opportunities extending from this work are discussed.

7.1 Takeaways

7.1.1 Sampling 6-DoF effector poses

Both policy learning and modular approaches repeat the following algorithmic steps:

1. Sample effector poses in the robot's workspace, a subset of $SE(3)$.
2. Rank poses by probability of grasp or placement stability and task relevance.
3. Find a motion plan moving the effector to the highest-ranking, reachable pose.

In our implementations, Steps 2 and 3 heavily relied on existing machine learning and motion planning technology, respectively. However, an existing technology could not be immediately adapted for implementing Step 1. In Chapters 4 to 6, we explored different ways to implement Step 1:

- **Local geometry heuristics.** In Chapter 4, we showed how GPD heuristics for sampling grasp poses [21, 84] are used to limit the grasp choices for pick-and-place. Similar heuristics could be developed for sampling place poses, e.g., align the hand to a horizontal plane.
- **Spatial attention.** In Chapter 5, we showed how learning to sequentially focus attention enables dense sampling of candidate effector poses.
- **Shape completion.** In Chapter 6, we showed how shape completion enables efficient sampling of grasps and placements which are stable given the estimated object's shape.

While the feasibility of each sampling mechanism was shown for a range of tasks, examples demonstrating a lack of generalizability can be found. For local geometry heuristics, grasps on the tops of bottles or near the edges of mugs were not found due to limitations in GPD's sampling heuristics. For spatial attention, the system had to be retrained for each new task, even when the modifications to the task were fairly minor, e.g., adding distractor objects. For shape completion, the grasp and placement samples were useful if the predicted shape was reasonably accurate, which was not the case for novel-object categories. It may be there is no best approach for every task and situation, so efficient effector pose sampling will likely remain a key issue for pick-and-place for some time.

7.1.2 Abstraction in policy learning

An important takeaway for pick-and-place policy learning is that abstraction is essential. In the context of reinforcement learning, an *abstraction* is a function mapping high-dimensional states or actions to a lower dimensional space. This mapping can *alias* states/actions, meaning many states/actions map to the same abstract state/action (i.e., the abstraction is not invertible). Aliasing can prevent the system from learning an optimal policy. On the other hand, without abstraction, the space of observations and actions is simply too large for efficient learning.

In Chapter 4, actions were the point cloud in the vicinity of effector pose samples, and observations were a history of actions. This is in contrast to the "ground MDP": 6-DoF effector poses for actions and a history of full-scene point clouds for observations. In Chapter 5, observations were again restricted to regions of space targeted for manipulation, and virtual *sense* actions enabled efficient sampling of effector poses. As many types of abstraction are possible, and it is not at all clear which are most appropriate for any given task, abstractions will likely remain a rich source of creativity for some time to come.

7.1.3 Uncertainty in planning

In Chapter 6, we showed perceptual uncertainty is relevant to regrasping performance. This is hardly surprising, as there are entire books dedicated to the importance of uncertainty in robotics [107]. What is not clear is how to efficiently represent and algorithmically account for uncertainty. We explored different approaches, and we found that using separate neural networks to predict probability of grasp and place stability works well in practice. But there are other options, including belief space planning [38, 125], or better implementations of Monte Carlo sampling [64], which may also be appropriate.

7.2 Opportunities for Future Research

7.2.1 Active sensing

No system proposed in this thesis plans additional views of the scene. Intuitively, integrating additional views should decrease perceptual uncertainty, as more of the objects will become visible, and thus result in improved performance. While active sensing has been investigated for grasping [25, 76], surprisingly few have systematically investigated this problem for pick-and-place [70].

7.2.2 Rearrangement

If the progression of robotic manipulation for novel objects mimics that of fully observed objects, we will see grasping, regrasping, and then rearrangement. *Rearrangement planning* is the problem of moving the objects into a *goal arrangement*, i.e., a set of goal poses for each object. As rearrangement is still very difficult for fully observed settings [120, 48], not much has been done to address this problem for partially observed, novel objects. Tasks in Chapters 5 and 6 involved rearranging a scene, but there we assumed the order of the arrangement was not significant in terms of collision-free arm motions. This, in general, is not the case, as placed objects become obstacles. Indeed, for the partially observed setting we have the new problem of how placed objects affect the visibility of the scene. Again, policy learning approaches immediately suggest themselves for solving this problem, but learning over long time horizons takes even longer.

7.2.3 Pushing

Of course, static pick-and-place actions are not the only useful actions. Some rearrangement tasks are more time consuming than necessary when we are constrained to static pick-and-place. For example, if the target object is in the middle of a bunch of other objects, it would be more efficient to push some of the objects out of the way before grasping the target object. However, predicting the effects of pushing for partially visible, unknown objects is challenging (e.g., see [127] for an example of pushing a single, unknown object). Finding a unified framework for novel-object pick-and-place and pushing is an interesting challenge for the future.

Appendix A

Reinforcement Learning

This chapter provides a brief introduction to reinforcement learning. A more complete treatment can be found in Sutton and Barto's book [103]. This gives the necessary background for understanding the models and algorithms used in Part II.

A.1 Markov Decision Processes

Reinforcement learning (RL) is a set of techniques for solving Markov decision processes, particularly when the transition and reward functions are unknown:[1]

Definition A.1 (Markov decision process (MDP)). *A* Markov decision process *is a 4-tuple, $\mathcal{M} = \langle S, A, \mathcal{T}, R \rangle$, consisting of a set of states S, a set of actions A, a transition function $\mathcal{T} : S \times A \times S$ – where $\mathcal{T}(s, a, s') = \Pr(s'|a, s)$ is the probability the next state is $s' \in S$ given the current action is $a \in A$ and the current state is $s \in S$ – and a reward function $R : S \times A \times S \to \mathbb{R}$.*

The MDP is a mathematical model of the robot's environment. At each time step $t \in \{1, 2, \ldots, t_{max}\}$, the robot (*agent*) observes the current state, takes an action, and receives a reward. (See Figure A.1 for an example.) Throughout this chapter, we assume S and A are finite and R and t are bounded. A consequence of assuming a finite number of time steps is that the state variable must include a component indicating the current time step. This is because, for a finite-horizon MDP, process termination is itself considered a state, and the transition probabilities must include the probability of transitioning to this state.

A *policy* $\pi : S \to A$ is a function mapping states to actions. Given an MDP $\mathcal{M}$ and a start state $s_1 \in S$, the objective of RL is to find a *policy* π^* maximizing the

[1]*Dynamic programming* treats the case of known transition and reward functions.

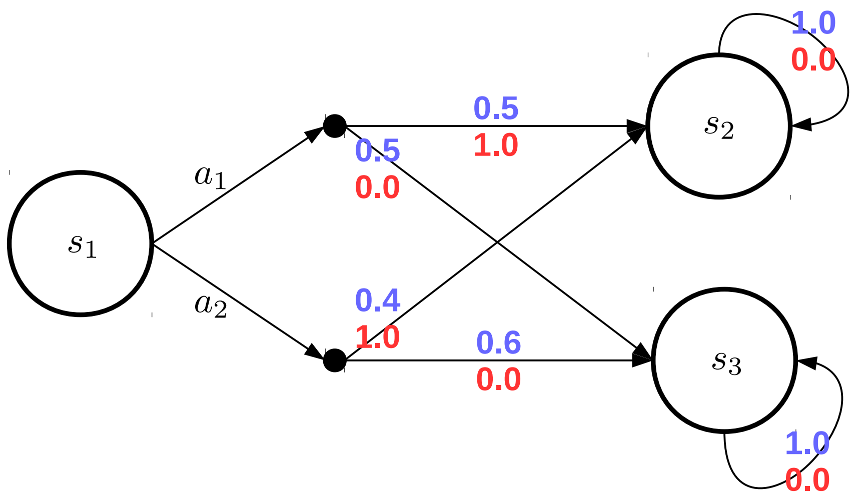


Figure A.1: An MDP with $S = \{s_1, s_2, s_3\}$ and $A = \{a_1, a_2\}$. Transition probabilities are shown above the arrows in blue, and rewards are shown below the arrows in red. Evidently, the optimal action is a_1.

expected sum of future rewards, i.e., Equation A.1, where s_t is the tth state visited and a_t is the tth action taken. The expectation is taken with respect to $\mathcal{T}$.

$$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t=1}^{t_{max}} R(s_t, \pi(s_t), s_{t+1})\right] \tag{A.1}$$

A.2 Policies

Instead of searching the space of policies, it can be more efficient to incrementally learn an action-value function. An *action-value* function is the expected sum of future rewards given the current state s_t, action a_t, and policy π (Equation A.2). The optimal action-value function satisfies the Bellman equation (Equation A.3), and the objective (Equation A.1) is restated in terms of the optimal action-value function (Equation A.4). A policy which takes an action maximizing the action-value function is called a *greedy policy*.[2]

[2]There are multiple greedy policies when the argmax is ambiguous; however, $q*$ is unique.

$$q_\pi(s_t, a_t) = \sum_{i=t}^{t_{max}} \mathbb{E}\left[R(s_i, a_i, s_{i+1})\right] \tag{A.2}$$

$$q^*(s_t, a_t) = \mathbb{E}\left\{R(s_t, a_t, s_{t+1}) + \max_{a \in A}\left[q^*(s_{t+1}, a)\right]\right\} \tag{A.3}$$

$$\pi^*(s_t, a_t) = \operatorname*{argmax}_{a \in A} q^*(s_t, a) \tag{A.4}$$

Because the agent does not know the transition and reward functions ahead, the action-value function must be estimated. An estimated action-value function is denoted $Q : S \times A \to \mathbb{R}$ and is called the Q-function. The greedy policy with respect to Q is problematic as the agent avoids actions with underestimated value. The trade-off between receiving high expected sum of future rewards, with respect to Q, and improving the accuracy of Q is called *exploration versus exploitation*.

To facilitate exploration, sometimes stochastic policies are used. The simplest and most common of these is ϵ-greedy (Equation A.5, where, abusing notation, π is a probability distribution). In some implementations, the parameter $\epsilon \in [0, 1]$ is decremented as learning progresses, so the ϵ-greedy policy becomes more like the greedy policy.

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \operatorname{argmax}_{a'} Q(s, a') \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases} \tag{A.5}$$

A.3 Algorithms

With finite state and action spaces, algorithms exist for computing q^* to arbitrary precision. When state or action spaces are continuous, approximate methods must be used instead. Approximate methods resemble the exact algorithms, but a different data structure is used to represent the Q-function.

A.3.1 Exact algorithms

There are three similar algorithms for iteratively improving an action-value function from experience. The first is Q-learning (Algorithm A.1) [117]. With an appropriate schedule for decreasing α, and if all states continue to be visited, Q-learning converges to q^* as the number of iterations goes to infinity [118]. Interestingly, this is independent of the policy used by Q-learning. The second algorithm is Sarsa (Algorithm A.2) [92]. Sarsa uses the next action chosen by the policy instead of the action maximizing Q for the Q-function update. (Compare line 7 in Algorithm A.1 to line 8 in Algorithm A.2.) Sarsa converges to q^* under the same conditions as Q-learning and with the additional condition that the policy used for training converges in the limit to the greedy policy [97]. The third algorithm is based on Monte Carlo sampling of sums of future rewards (Algorithm A.3). The disadvantage of this algorithm is that the Q-function is not updated until the end of the episode. This is problematic only for long time horizons. It is not known if the MC algorithm always converges [103].

Algorithm A.1: Q-learning

```
  Input  : $N, t_{max} \in \mathbb{N}$, $\alpha, \epsilon \in [0, 1]$
1 $Q(s, a) \leftarrow 0, \forall s \in S, \forall a \in A$  // Initialize $Q$-function.
2 for $i \leftarrow 1, \dots, N$ do
3     $s \sim I$ // Sample initial state.
4     for $t \leftarrow 1, \dots, t_{max}$ do
5         $a \sim \pi(\cdot|s, \epsilon)$ // Sample action using $\epsilon$-greedy.
6         Take $a$ and observe $s'$ and $r = R(s, a, s')$
7         $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\,[r + \max_{a'} Q(s', a')]$
8         $\epsilon \leftarrow$ Decrement-$\epsilon(i, t)$; $\alpha \leftarrow$ Decrement-$\alpha(i, t)$
9         $s \leftarrow s'$; $a \leftarrow a'$
```

Algorithm A.2: Sarsa

```
   Input  : $N, t_{max} \in \mathbb{N}$, $\alpha, \epsilon \in [0, 1]$
1  $Q(s, a) \leftarrow 0, \forall s \in S, \forall a \in A$  // Initialize $Q$-function.
2  for $i \leftarrow 1, \dots, N$ do
3      $s \sim I$ // Sample initial state.
4      $a \sim \pi(\cdot|s, \epsilon)$ // Sample action using $\epsilon$-greedy.
5      for $t \leftarrow 1, \dots, t_{max}$ do
6          Take $a$ and observe $s'$ and $r = R(s, a, s')$
7          $a' \sim \pi(\cdot|s, \epsilon)$ // Sample next action using $\epsilon$-greedy.
8          $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\,[r + Q(s', a')]$
9          $\epsilon \leftarrow$ Decrement-$\epsilon(i, t)$; $\alpha \leftarrow$ Decrement-$\alpha(i, t)$
10         $s \leftarrow s'$; $a \leftarrow a'$
```

Algorithms A.1 and A.2 are called 1-step Q-learning and Sarsa, respectively, because the update rule (lines 7 and 8, respectively) include the reward for exactly

Algorithm A.3: Monte Carlo

```
Input : $N, t_{max} \in \mathbb{N}$, $\alpha, \epsilon \in [0,1]$
1  $Q(s,a) \leftarrow 0, \forall s \in S, \forall a \in A$ // Initialize $Q$-function.
2  $C(s,a) \leftarrow 0, \forall s \in S, \forall a \in A$ // Initialize counts.
3  $U(s,a) \leftarrow 0, \forall s \in S, \forall a \in A$ // Initialize sums.
4  for $i \leftarrow 1, \dots, N$ do
5      $s \sim I$ // Sample initial state.
6      for $t \leftarrow 1, \dots, t_{max}$ do
7          $a \sim \pi(\cdot|s, \epsilon)$ // Sample action using $\epsilon$-greedy.
8          Take $a$ and observe $s'$ and $r = R(s,a,s')$
9          $s_t \leftarrow s$; $a_t \leftarrow a$; $r_t \leftarrow r$ // Store experience.
10         $s \leftarrow s'$
11     $g \leftarrow 0$ // Sum of future rewards.
12     for $t \leftarrow t_{max}, \dots, 1$ do
13         $g \leftarrow g + r_t$
14         $C(s_t, a_t) \leftarrow C(s_t, a_t) + 1$
15         $U(s_t, a_t) \leftarrow U(s_t, a_t) + g$
16         $Q(s_t, a_t) \leftarrow U(s_t, a_t)/C(s_t, a_t)$
17     $\epsilon \leftarrow$ Decrement-$\epsilon(i)$
```

1 time step. n-step Q-learning and Sarsa are obtained by incorporating the last n rewards into the update rule. For example, the n-step update for Sarsa is given in Equation A.6, where r_t is the most recently observed reward, r_{t-1} is the previously observed reward, and so on. n-step versions with $n > 1$ often outperform the original algorithms [103], especially when there is no reward for a fixed number of steps [23].

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r_{t-n+1} + \cdots + r_{t-1} + r_t + Q(s',a')\right] \tag{A.6}$$

A.3.2 Approximate algorithms

A popular method for approximating an optimal value function, even when the state space is high-dimensional and continuous, is deep Q-learning [73]. The method is shown in Algorithm A.4. There are three differences from standard Q-learning. First, the Q-function, now denoted $Q(s,a;\theta)$, is implemented as a deep neural network and is parameterized by a vector θ. Second, experiences are stored in a database D, and updates are performed using random batches from the database. Third, the

Q-function parameters (weights) are updated with gradient descent, using the loss function Equation A.7, where the expectation is taken with respect to a distribution over the experience database (uniform by default) and $\hat{\theta}$ is an outdated copy of θ. It is straight-forward to implement a Sarsa version of DQN using the loss function Equation A.8.

$$L(\theta_i) = \mathbb{E}\left[\Big(r + \max_{a'} Q(s,a';\hat{\theta}) - Q(s,a;\theta)\Big)^2\right] \tag{A.7}$$

$$L(\theta_i) = \mathbb{E}\left[\Big(r + Q(s,a';\hat{\theta}) - Q(s,a;\theta)\Big)^2\right] \tag{A.8}$$

A.4 Beyond Value Learning

A problem with DQN is it requires a discrete, small set of action choices.[3] An alternative without this issue is to search in policy space. Algorithms of this type include REINFORCE [121], DDPG [60], and A3C [71]. Since these are not used in this thesis, we will not cover them further.

Another important issue that arises in robotics is partial observability. The definition of an MDP makes it clear that the transition function is purely a function of the current state, the current action, and the next state. In reality, it is easy to represent the state in a way that violates this assumption. For instance, consider a robot which can move its point cloud sensor. If the object to be manipulated is occluded from the current view, the probability the manipulation action will succeed could change if past views are considered. In this case, the above algorithms would not find an "optimal" policy in the sense that a better policy could be found if the robot remembered past observations. In this case the environment is actually a POMDP:

Definition A.2 (Partially observable Markov decision process (POMDP))**.** *A* partially observable Markov decision process *is a 6-tuple, $\mathcal{P} = \langle S, A, \mathcal{T}, R, \Omega, O\rangle$, where $\langle S, A, \mathcal{T}, R\rangle$ is an MDP, Ω is a set of observations, and $O : A \times S \times \Omega \to \mathbb{R}$ is $O(a,s',o) = \Pr(o|a,s')$, i.e., the probability the observation is o given the state after taking action a is s'.*

In this case, the robot does not observe the state directly, but instead gets an observation, which stochastically depends only on the action and next state. This

[3]Otherwise, it would not be possible to evaluate $\max_a$ when evaluating the policy or the loss.

Algorithm A.4: Deep Q-learning

```
   Input  : $K, M, N, t_{max} \in \mathbb{N}$, $\epsilon \in [0,1]$
 1 $D \leftarrow \{\}$ // Initialize replay database to empty.
 2 $\theta \sim \Theta$ // Randomly sample weights of $Q$.
 3 $\hat{\theta} \leftarrow \theta$ // Copy $\theta$ to target weights.
 4 $k \leftarrow 0$ // Reset counter for $\hat{\theta}$.
 5 for $i \leftarrow 1, \ldots, N$ do
 6     $s \sim I$ // Sample initial state.
 7     for $t \leftarrow 1, \ldots, t_{max}$ do
 8         $a \sim \pi(\cdot|s, \epsilon)$ // Sample action using $\epsilon$-greedy.
 9         Take $a$ and observe $s'$ and $r = R(s, a, s')$
10         $D \leftarrow D \cup \{(s, a, s', r)\}$ // Store transition in database.
11         $D \leftarrow$ PruneDatabase($D$) // Remove an experience if full.
12         $L \leftarrow 0$
13         for $j \leftarrow 1, \ldots, M$ do
14             $(\hat{s}, \hat{a}, \hat{s}', \hat{r}) \sim D$ // Sample an experience from database.
15             $L \leftarrow \frac{1}{M}\left(\hat{r} + \max_{\hat{a}'} Q(\hat{s}, \hat{a}'; \hat{\theta}) - Q(\hat{s}, \hat{a}; \theta)\right)^2$
16         Perform a gradient descent step on $L$ w.r.t. $\theta$
17         $k \leftarrow k + 1$
18         if $k \geq K$ then
19             $k \leftarrow 0$
20             $\hat{\theta} \leftarrow \theta$ // Update target weights.
21         $\epsilon \leftarrow$ Decrement-$\epsilon(i, t)$
22         $s \leftarrow s'$; $a \leftarrow a'$
```

is useful because the robot no longer needs to have a Markov representation of the state, which is often difficult to obtain.

A POMDP can be reduced to an MDP by representing state with the entire history of observations and actions up to the current time step t, i.e., $\hat{s}$ in Equation A.9 ([6] pp. 187-188). With this reduction, all of the above methods are applicable. However, the dimension of the state space is now linear in t_{max}, making the problem PSPACE-hard [83]. For this reason, in practice, often only the last $k \ll t_{max}$ observations and actions are stored as an approximation to the Markov state.

$$\hat{s} = (o_1, o_2, \ldots, o_t, a_1, a_2, \ldots, a_{t-1}) \tag{A.9}$$

A.5 Further Reading

Most of the notation used here is due to Sutton and Barto [103]. Bertsekas gives an overview of dynamic programming for finite horizon [6] and infinite horizon [7] problems. DQN was preceded by TD-Gammon, a high-performing backgammon player using a neural network representation of the Q-function [105]. When TD-Gammon was applied to other problems, such as Atari games, learning was unstable [73]. DQN was aimed at resolving this issue [73]. Many improvements have been made to DQN, several of which were combined in [30]. Smallwood and Sondik give exact solution methods for small POMDPs [98], and a brief introduction to POMDPs is given by Kaelbling et al. [37]. Kober et al. survey RL applied to robotics [46].

Appendix B

Additional Data for Chapter 4

This appendix contains additional implementation details for Chapter 4.

B.1 Pick Descriptor

From Section 4.1.2, we saw that the pick descriptor is an $n_x \times n_y \times 12$ image I formed as $I = \mathit{Proj}(\mathit{Crop}(z, \mathit{Trans}(T^{-1}, C)))$. Here we provide implementations for the functions *Trans*, *Crop*, and *Proj*.

The function *Trans* takes in a rigid transformation $T \in SE(3)$ represented by (i) a *rotation matrix*, i.e., an orthogonal matrix R with $\det(R) = 1$, and (ii) an offset $d \in \mathbb{R}^3$. *Trans* also takes in a point cloud C represented by an $3 \times n$ matrix. The ith point of the point cloud C is denoted $C(1:3, i)$ and is a column vector.

Algorithm B.1: *Trans*

```
   Input  : Rotation matrix $R \in SO(3)$, offset $d \in \mathbb{R}^3$, point cloud $C \in \mathbb{R}^{3\times n}$
   Output: Point cloud $\bar{C} \in \mathbb{R}^{3\times n}$
1  $\bar{C} \leftarrow 0$
2  for $i \leftarrow 1, \ldots, n$ do
3      $\bar{C}(1:3, i) \leftarrow RC(1:3, i) + d$
4  return $\bar{C}$
```

The function *Crop* takes in the size of an axis-aligned, rectangular cuboid centered at the origin with length $d(1)$, width $d(2)$, and height $d(3)$. *Crop* also takes in a point cloud C, represented by an $3 \times n$ matrix. The output is another point cloud with only the points of C contained within the rectangular cuboid specified by d.

Algorithm B.2: *Crop*

```
  Input  : $d \in \mathbb{R}^3_{>0}$, $C \in \mathbb{R}^{3 \times n}$
  Output: $\bar{C} \in \mathbb{R}^{3 \times \bar{n}}$
1 $\bar{C} \leftarrow 0$
2 $\bar{n} \leftarrow 0$
3 for $i \leftarrow 1, \ldots, n$ do
4     $[x, y, z]^T \leftarrow C(1:3, i)$
5     if $x \geq -0.5d(1) \wedge x \leq 0.5d(1) \wedge y \geq -0.5d(2) \wedge y \leq 0.5d(2) \wedge z \geq -0.5d(3) \wedge z \leq 0.5d(3)$ then
6         $\bar{n} \leftarrow \bar{n} + 1$
7         $\bar{C}(1:3, \bar{n}) \leftarrow C(1:3, i)$
8 return $\bar{C}(1:3, 1:\bar{n})$
```

The function *Proj* takes a rectangular cuboid size $d \in \mathbb{R}^3_{>0}$ and a point cloud $C \in \mathbb{R}^{3 \times n}$. The result is 12, $n_x \times n_y$ images. The first 3 images are height maps, and the last 9 images include surface normal information. We assume we have a function `EstimateSurfaceNormals` which produces a $3 \times n$ matrix N, where the ith column is a surface normal estimate for the ith point in C. A function for estimating surface normals is described in [93].

Algorithm B.3: *Proj*

```
Input  : $d \in \mathbb{R}^3_{>0}$, $C \in \mathbb{R}^{3\times n}$
Output: $I \in \mathbb{R}^{n_x \times n_y \times 12}$
$I \leftarrow 0$
$N \leftarrow \texttt{EstimateSurfaceNormals}(C)$
for $i \leftarrow 1, \dots, n$ do
    $j \leftarrow \lfloor (C(1,i) + 0.5d(1)) \frac{n_x - 1}{d(1)} \rfloor$
    $k \leftarrow \lfloor (C(2,i) + 0.5d(2)) \frac{n_y - 1}{d(2)} \rfloor$
    $v \leftarrow 1 - (C(3,i) + 0.5d(3))/d(3)$
    if $v > I(j,k,1)$ then
        $I(j,k,1) \leftarrow v$
        $I(j,k,4:6) \leftarrow N(1:3,i)$
    $j \leftarrow \lfloor (C(1,i) + 0.5d(1)) \frac{n_x - 1}{d(1)} \rfloor$
    $k \leftarrow \lfloor (C(3,i) + 0.5d(3)) \frac{n_y - 1}{d(3)} \rfloor$
    $v \leftarrow 1 - (C(2,i) + 0.5d(2))/d(2)$
    if $v > I(j,k,2)$ then
        $I(j,k,2) \leftarrow v$
        $I(j,k,7:9) \leftarrow N(1:3,i)$
    $j \leftarrow \lfloor (C(2,i) + 0.5d(2)) \frac{n_x - 1}{d(2)} \rfloor$
    $k \leftarrow \lfloor (C(3,i) + 0.5d(3)) \frac{n_y - 1}{d(3)} \rfloor$
    $v \leftarrow 1 - (C(1,i) + 0.5d(1))/d(1)$
    if $v > I(j,k,3)$ then
        $I(j,k,3) \leftarrow v$
        $I(j,k,10:12) \leftarrow N(1:3,i)$
return $I$
```

Appendix C

Additional Data for Chapter 6

This appendix contains implementation details and additional experimental results for the modular approach of Chapter 6.

C.1 System Overview

This section provides details on our implementation of BoNet [126] and point completion network (PCN) [128].

C.1.1 BoNet

We use BoNet (Figure C.1) to segment a point cloud and provide segmentation uncertainty estimates. The first output of BoNet is *pmask*. This is an $n \times K$ matrix where the value at row i and column j predicts the probability point i belongs to object j and K is a predefined maximum number of objects. The second output is *bbvert*. This is a $3 \times 2 \times K$ tensor predicting the (x, y, z) positions of two corners of K axis-aligned bounding boxes. We do not use this output at test time. The third output is *bbscores*. This predicts the probability each bounding box contains an object as there may be more bounding boxes than objects present in the scene. The authors of BoNet estimate the segmentation *mask* by element-wise multiplying *bbscores* to each row of *pmask*. The argmax of *mask* along each row yields the segmentation $\{C_i\}_{i=1}^{n_{obj}}$ and the max of *mask* along each row yields the uncertainty vectors $\{U_i\}_{i=1}^{n_{obj}}$.

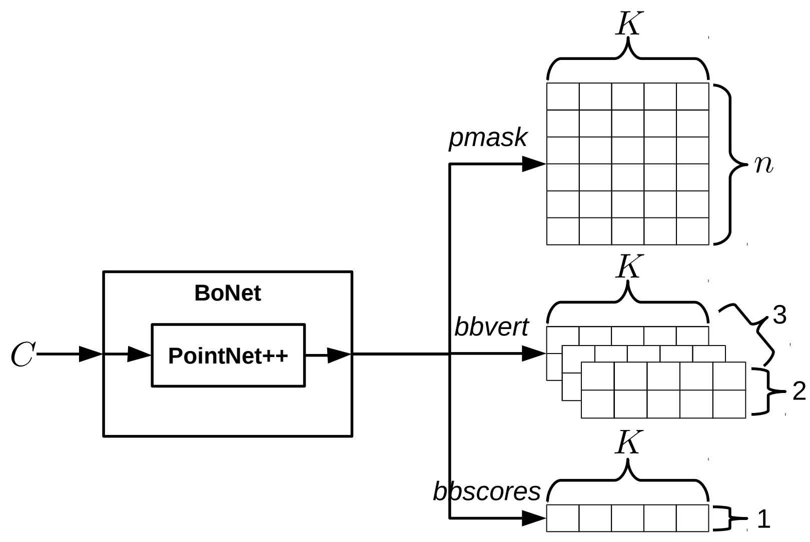


Figure C.1: BoNet [126] takes in a point cloud C, internally uses a PointNet++ [89] "backbone", and outputs *pmask*, *bbvert*, and *bbscores*.

C.1.2 Point completion network

PCN consists of an encoder (two PointNet layers [90]) and a decoder (three fully connected, inner product layers) (Figure C.2).[1] We augmented the original version of PCN with a second decoder for uncertainty estimates. This "uncertainty decoder" has the same architecture as the point decoder except the output size is $\bar{n} \times 1$ instead of $\bar{n} \times 3$. The uncertainty decoder is trained after the other parameters of the network have been trained and fixed. The uncertainty decoder is trained using the binary cross-entropy loss where labels indicate if each point is within Euclidean distance $\beta \in \mathbb{R}_{>0}$ from the nearest ground truth point. Additional completion examples with uncertainty values are shown in Figure C.3b.

C.2 Regrasp Planning Under Uncertainty

This section contains additional details about the regrasp planning cost.

[1]The "detailed output" layers were omitted in our implementation, and the Chamfer distance (CD) loss was used to train the shape completion branch. (See [128].)

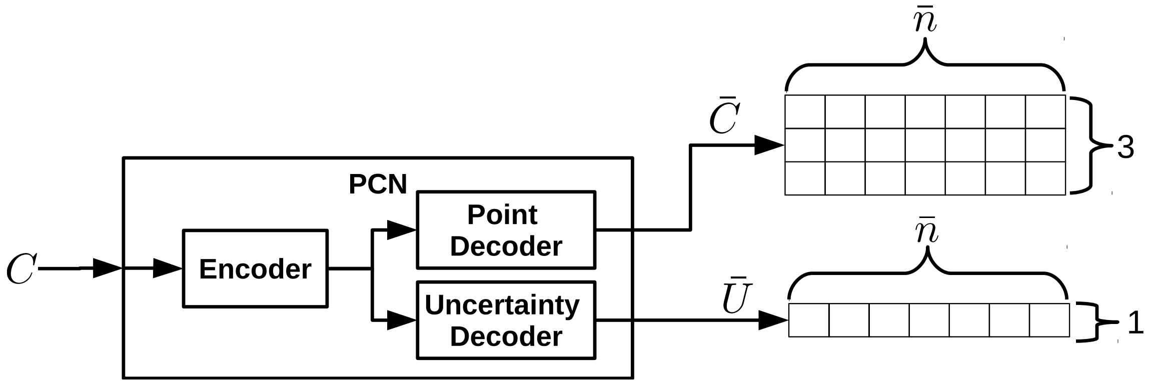


Figure C.2: Summary of PCN [128]. The input is a partial point cloud, C, and the output includes the completed point cloud $\bar{C}$ and uncertainty values $\bar{U}$.

C.2.1 Derivation of σ for MC sampling

The probability the offset x of a point from the nearest ground truth point is less than β is modeled by Equation C.1, where Φ is the Gaussian cumulative distribution function with mean 0 and variance σ^2. The integral of the Gaussian probability density function from $-\infty$ to $-\beta$ (and from β to ∞, due to symmetry) is $\Phi(-\beta; 0, \sigma^2)$. We subtract these tails from 1 to get the probability x lies in the region $[-\beta, \beta]$ (Equation C.1). Equation C.2 results from expanding the definition of Φ. Equation C.3 and C.4 result from elimination and rearrangement of terms. Equation C.5 results from the fact that erf^{-1} is an odd function. Equation C.6 results from rearranging terms and is Equation 6.6.

$$\begin{aligned}
\Pr(-\beta \leq x \leq \beta) = \bar{U} &= 1 - 2\Phi(-\beta; 0, \sigma^2) && \text{(C.1)}\\
&= 1 - 2\frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{-\beta}{\sqrt{2}\sigma}\right)\right] && \text{(C.2)}\\
&= -\,\mathrm{erf}\left(\frac{-\beta}{\sqrt{2}\sigma}\right) && \text{(C.3)}\\
\mathrm{erf}^{-1}(-U) &= \frac{-\beta}{\sqrt{2}\sigma} && \text{(C.4)}\\
\mathrm{erf}^{-1}(U) &= \frac{\beta}{\sqrt{2}\sigma} && \text{(C.5)}\\
\frac{\beta}{\sqrt{2}\,\mathrm{erf}^{-1}(\bar{U})} &= \sigma && \text{(C.6)}
\end{aligned}$$

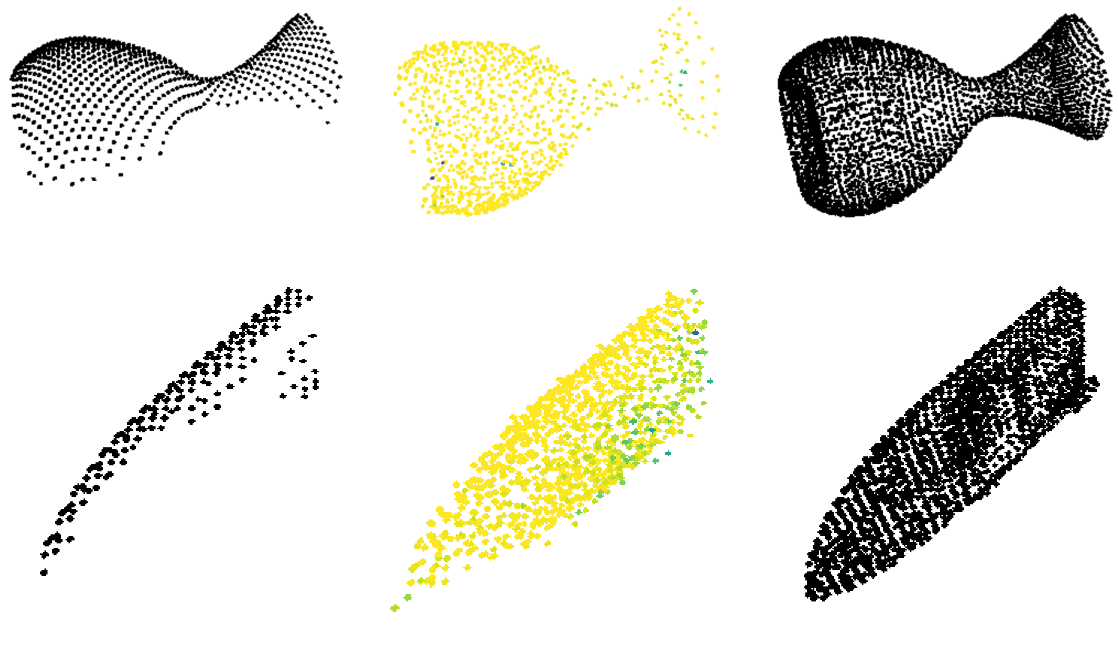

(a) Observed cloud. (b) Complete cloud. (c) Ground truth.

Figure C.3: Shape completions with PCN. Yellow represents high $\bar{U}$ values (near 1) and blue represents low $\bar{U}$ values (near 0.5).

C.3 Experiments

We provide additional simulation results for the canonical placement, bin packing, and bottle arrangement tasks. Details of the scenarios are described in Section 6.4.

C.3.1 Perception ablation study

Results for bin packing Test-2 (i.e., novel-object categories, including airplane, bowl, and stapler) are shown in Table C.1. Results for bottle arrangement are shown in Table C.2. We again see that perception is a significant source of error (place execution success rate down 41.7% from GT Seg. & Comp. for bin packing Test-2 and down 14.4% for bottle arrangement). Also, we again find that shape completion is critical to finding a regrasp plan (regrasp plan found rate down 56.7% for bin packing Test-2 and down 92% for bottle arrangement).

	GT Seg. & Comp.	GT Seg.	Percep.	GT Seg. & No Comp.
Place Execution Success	0.849 ± 0.011	0.459 ± 0.017	0.432 ± 0.017	0.304 ± 0.034
Regrasp Plan Found	0.878 ± 0.009	0.708 ± 0.013	0.718 ± 0.013	0.151 ± 0.010
Temporary Place Stable	1.000 ± 0.000	0.786 ± 0.114	0.167 ± 0.112	0.500 ± 0.500
Regrasp planning time (s)	36.85 ± 1.614	26.50 ± 0.891	25.22 ± 0.869	23.007 ± 2.145

Table C.1: Perception ablation study for **bin packing** Test-2. Showing average ± standard error over 200 episodes.

	GT Seg. & Comp.	GT Seg. (Train)	GT Seg. (Test)	Percep. (Train)	Percep. (Test)	GT Seg. & No Comp.
Place Execution Success	0.970 ± 0.005	0.868 ± 0.011	0.860 ± 0.011	0.853 ± 0.011	0.826 ± 0.012	0.234 ± 0.053
Regrasp Plan Found	0.996 ± 0.002	0.998 ± 0.001	0.999 ± 0.001	0.988 ± 0.003	0.984 ± 0.004	0.064 ± 0.008
Temporary Place Stable	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	–
Regrasp planning time (s)	4.486 ± 0.100	1.680 ± 0.083	1.596 ± 0.065	1.598 ± 0.070	1.550 ± 0.066	1.144 ± 0.059

Table C.2: Perception ablation study for **bottle arrangement**. Showing average ± standard error over 500 episodes.

C.3.2 Regrasp cost comparison

We also compare regrasp planning costs for objects from Test-2 (i.e., novel categories including airplane, bowl, and stapler). Results for the canonical task are shown in Table C.3, and results for the bin packing task are shown in Table C.4. The main takeaway is that in both cases SP significantly outperforms the best-performing baseline in terms of place execution success.

	No Cost	Step Cost	GQ	MC	CU	SP
Place Execution Success	0.446 ± 0.011	0.535 ± 0.012	0.520 ± 0.012	0.543 ± 0.012	0.533 ± 0.012	**0.591** ± 0.011
Temporary Place Stable	0.690 ± 0.021	0.555 ± 0.046	0.608 ± 0.030	0.717 ± 0.032	0.671 ± 0.036	**0.742** ± 0.027
Plan Length	3.265 ± 0.035	**2.323** ± 0.018	2.686 ± 0.025	2.501 ± 0.022	2.419 ± 0.020	2.518 ± 0.023
Regrasp planning time (s)	**4.278** ± 0.156	14.84 ± 0.539	68.87 ± 0.657	99.36 ± 0.818	60.05 ± 0.633	74.08 ± 0.732

Table C.3: Cost comparison for **canonical placement** with novel-object categories (over $2,000$ episodes). A cell is highlighted green iff the corresponding value is significantly higher (i.e. $p \leq 0.05$) than the highest baseline value.

	No Cost	Step Cost	GQ	MC	CU	SP
Place Execution Success	0.412 ± 0.017	0.417 ± 0.017	0.395 ± 0.017	0.458 ± 0.017	0.429 ± 0.017	**0.465** ± 0.017
Temporary Place Stable	0.704 ± 0.051	0.714 ± 0.125	0.533 ± 0.075	0.750 ± 0.083	**0.778** ± 0.101	0.686 ± 0.080
Plan Length	2.514 ± 0.036	**2.094** ± 0.015	2.247 ± 0.024	2.167 ± 0.020	2.118 ± 0.017	2.193 ± 0.022
Regrasp planning time (s)	**6.030** ± 0.237	8.484 ± 0.408	51.61 ± 1.113	58.56 ± 1.064	50.92 ± 1.177	53.35 ± 1.159

Table C.4: Cost comparison for **bin packing** with novel-object categories (over 200 episodes). A cell is highlighted green iff the corresponding value is significantly higher (i.e. $p \leq 0.05$) than the highest baseline value.

Finally, Table C.5 shows the regrasp cost comparison for bottle arrangement. Although the advantage of SP over GQ is not statistically significant, the advantage of SP over no cost and step cost is statistically significant in terms of place execution success rate. We attribute the good performance GQ to more accurate instance segmentation and shape completion (because there are fewer objects to segment and there is only one category for shape completion).

	No Cost	Step Cost	GQ	MC	CU	SP
Place Execution Success	0.831 ± 0.012	0.824 ± 0.012	0.860 ± 0.011	0.867 ± 0.011	0.820 ± 0.012	$\mathbf{0.877} \pm 0.011$
Temporary Place Stable	0.889 ± 0.043	–	0.926 ± 0.051	$\mathbf{1.000} \pm 0.000$	–	0.972 ± 0.028
Plan Length	2.122 ± 0.016	$\mathbf{2.000} \pm 0.000$	2.061 ± 0.011	2.024 ± 0.007	$\mathbf{2.000} \pm 0.000$	2.073 ± 0.012
Regrasp planning time (s)	$\mathbf{1.378} \pm 0.032$	1.442 ± 0.045	31.98 ± 0.181	33.33 ± 0.458	30.55 ± 0.253	32.97 ± 0.259

Table C.5: **Bottle arrangement** performance (over 500 episodes). A cell is highlighted green iff the corresponding value is significantly higher (i.e. $p \leq 0.05$) than the highest baseline value.

Bibliography

[1] Rachid Alami, Jean-Paul Laumond, and Thierry Siméon. "Two manipulation planning algorithms". In: *Proceedings of the Workshop on Algorithmic Foundations of Robotics*. A. K. Peters, Ltd., 1995, pp. 109–125.

[2] Rachid Alami, Thierry Siméon, and Jean-Paul Laumond. "A geometrical approach to planning manipulation tasks. The case of discrete placements and grasps". In: *Int'l Symp. on Robotics Research*. Cambridge, MA, USA: MIT Press, 1991, pp. 453–463. ISBN: 0262132532.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. "Hindsight experience replay". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[4] Oron Anschel, Nir Baram, and Nahum Shimkin. "Averaged-DQN: variance reduction and stabilization for deep reinforcement learning". In: *Int'l Conf. on Machine Learning*. JMLR. 2017, pp. 176–185.

[5] Lars Berscheid, Pascal Meißner, and Torsten Kröger. "Self-supervised learning for precise pick-and-place without object model". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4828–4835. DOI: 10.1109/LRA.2020.3003865.

[6] Dimitri Bertsekas. *Dynamic programming and optimal control*. 4th ed. Vol. 1. Athena scientific, 2017.

[7] Dimitri Bertsekas. *Dynamic programming and optimal control*. 4th ed. Vol. 2. Athena scientific, 2017.

[8] Paul Besl and Neil McKay. "Method for registration of 3D shapes". In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. Int'l Society for Optics and Photonics. SPIE, 1992, pp. 586–606. DOI: 10.1117/12.57955.

[9] Vassilios Christopoulos and Paul Schrater. "Handling shape and contact location uncertainty in grasping two-dimensional planar objects". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. 2007, pp. 1557–1563.

[10] Rosen Diankov. "Automated construction of robotic manipulation programs". PhD thesis. Robotics Institute, Carnegie Mellon University, 2010.

[11] Stanimir Dragiev, Marc Toussaint, and Michael Gienger. "Gaussian process implicit surfaces for shape estimation and grasping". In: *IEEE Int'l Conf. on Robotics and Automation*. 2011, pp. 2845–2850.

[12] Stanimir Dragiev, Marc Toussaint, and Michael Gienger. "Uncertainty aware grasping and tactile exploration". In: *IEEE Int'l Conf. on Robotics and Automation*. 2013, pp. 113–119.

[13] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martín-Martín, Arne Sieverling, Vincent Wall, and Oliver Brock. "Lessons from the Amazon picking challenge: four aspects of building robotic systems." In: *Robotics: Science and Systems*. 2016.

[14] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-shot visual imitation learning via meta-learning". In: *Proceedings of the 1st Annual Conf. on Robot Learning*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 357–368.

[15] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. 2016, pp. 4019–4026.

[16] Wei Gao and Russ Tedrake. "kPAM-SC: generalizable manipulation planning using keypoint affordance and shape completion". In: *arXiv preprint arXiv:1909.06980* (2019).

[17] Neha Garg, David Hsu, and Wee Sun Lee. "Learning to grasp under uncertainty using POMDPs". In: *IEEE Int'l Conf. on Robotics and Automation*. 2019, pp. 2751–2757.

[18] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. "Online replanning in belief space for partially observable task and motion problems". In: *IEEE Int'l Conf. on Robotics and Automation*. 2020, pp. 5678–5684. DOI: 10.1109/ICRA40945.2020.9196681.

[19] Marcus Gualtieri, James Kuczynski, Abraham Shultz, Andreas ten Pas, Robert Platt, and Holly Yanco. "Open world assistive grasping using laser selection". In: *IEEE Int'l Conf. on Robotics and Automation*. 2017, pp. 4052–4057.

[20] Marcus Gualtieri, Andreas ten Pas, and Robert Platt. "Pick and place without geometric object models". In: *IEEE Int'l Conf. on Robotics and Automation.* 2018.

[21] Marcus Gualtieri, Andreas ten Pas, Kate Saenko, and Robert Platt. "High precision grasp pose detection in dense clutter". In: *IEEE Int'l Conf. on Intelligent Robots and Systems.* 2016.

[22] Marcus Gualtieri and Robert Platt. "Learning 6-DoF grasping and pick-place using attention focus". In: *Proceedings of The 2nd Conference on Robot Learning.* Vol. 87. Proceedings of Machine Learning Research. Oct. 2018, pp. 477–486.

[23] Marcus Gualtieri and Robert Platt. "Learning manipulation skills via hierarchical spatial attention". In: *IEEE Transactions on Robotics* 36.4 (2020), pp. 1067–1078.

[24] Marcus Gualtieri and Robert Platt. "Robotic pick-and-place with uncertain object instance segmentation and shape completion". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1753–1760. DOI: 10.1109/LRA.2021.3060669.

[25] Marcus Gualtieri and Robert Platt. "Viewpoint selection for grasp detection". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems.* 2017, pp. 258–264.

[26] Kensuke Harada, Kazuyuki Nagata, Tokuo Tsuji, Natsuki Yamanobe, Akira Nakamura, and Yoshihiro Kawai. "Probabilistic approach for object bin picking approximated by cylinders". In: *2013 IEEE Int'l Conf. on Robotics and Automation.* 2013, pp. 3742–3747.

[27] Peter Hart, Nils Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[28] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *IEEE Int'l Conf. on Computer Vision.* 2017, pp. 2961–2969.

[29] Carlos Hernandez, Mukunda Bharatheesha, Wilson Ko, Hans Gaiser, Jethro Tan, Kanter van Deurzen, Maarten de Vries, Bas Van Mil, Jeff van Egmond, Ruben Burger, et al. "Team Delft's robot winner of the Amazon picking challenge 2016". In: *Robot World Cup.* Springer. 2016, pp. 613–624.

[30] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. “Rainbow: combining improvements in deep reinforcement learning”. In: *AAAI*. 2018.

[31] Tommi Jaakkola, Michael Jordan, and Satinder Singh. “Convergence of stochastic iterative dynamic programming algorithms”. In: *Advances in Neural Information Processing Systems*. 1994, pp. 703–710.

[32] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. “Spatial transformer networks”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2017–2025.

[33] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. “Caffe: convolutional architecture for fast feature embedding”. In: *Proceedings of the 22nd ACM Int'l Conf. on Multimedia*. ACM. 2014, pp. 675–678.

[34] Yun Jiang, Marcus Lim, Changxi Zheng, and Ashutosh Saxena. “Learning to place new objects in a scene”. In: *The Int'l Journal of Robotics Research* 31.9 (2012), pp. 1021–1043.

[35] Yun Jiang, Changxi Zheng, Marcus Lim, and Ashutosh Saxena. “Learning to place new objects”. In: *Int'l Conf. on Robotics and Automation*. 2012, pp. 3088–3095.

[36] Edward Johns, Stefan Leutenegger, and Andrew Davison. “Deep learning a grasp function for grasping under gripper pose uncertainty”. In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. 2016, pp. 4461–4468.

[37] Leslie Pack Kaelbling, Michael Littman, and Anthony Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[38] Leslie Pack Kaelbling and Tomás Lozano-Pérez. “Integrated task and motion planning in belief space”. In: *The Int'l Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227.

[39] Leslie Kaelbling and Tomás Lozano-Pérez. “Hierarchical task and motion planning in the now”. In: *IEEE/RSJ Int'l Conf. on Robotics and Automation*. 2011, pp. 1470–1477.

[40] Gregory Kahn, Peter Sujan, Sachin Patil, Shaunak Bopardikar, Julian Ryde, Ken Goldberg, and Pieter Abbeel. “Active exploration using trajectory optimization for robotic grasping in the presence of occlusions”. In: *IEEE Int'l Conf. on Robotics and Automation*. May 2015.

[41] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. "Scalable deep reinforcement learning for vision-based robotic manipulation". In: *Proceedings of The 2nd Conference on Robot Learning*. Vol. 87. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 651–673.

[42] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The Int'l Journal of Robotics Research* 30.7 (2011), pp. 846–894.

[43] Ben Kehoe, Dmitry Berenson, and Ken Goldberg. "Estimating part tolerance bounds based on adaptive cloud-based grasp planning with slip". In: *IEEE Int'l Conf. on Automation Science and Engineering*. 2012, pp. 1106–1113.

[44] Ben Kehoe, Dmitry Berenson, and Ken Goldberg. "Toward cloud-based grasping with uncertainty in shape: estimating lower bounds on achieving force closure with zero-slip push grasps". In: *IEEE Int'l Conf. on Robotics and Automation*. 2012, pp. 576–583.

[45] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *Int'l Conf. on Learning Representations* (2015).

[46] Jens Kober, Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: a survey". In: *The Int'l Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[48] Athanasios Krontiris and Kostas Bekris. "Dealing with difficult instances of object rearrangement." In: *Robotics: Science and Systems*. 2015.

[49] Brenden Lake, Tomer Ullman, Joshua Tenenbaum, and Samuel Gershman. "Building machines that learn and think like people". In: *Behavioral and Brain Sciences* 40 (2017).

[50] Hugo Larochelle and Geoffrey Hinton. "Learning to combine foveal glimpses with a third-order Boltzmann machine". In: *Advances in Neural Information Processing Systems*. 2010, pp. 1243–1251.

[51] Michael Laskey, Jeff Mahler, Zoe McCarthy, Florian Pokorny, Sachin Patil, Jur van den Berg, Danica Kragic, Pieter Abbeel, and Ken Goldberg. "Multi-armed bandit models for 2D grasp planning with uncertainty". In: *IEEE Int'l Conf. on Automation Science and Engineering*. 2015, pp. 572–579.

[52] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[53] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The Int'l Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.

[54] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies". In: *Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[55] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with large-scale data collection". In: *Int'l Symp. on Experimental Robotics*. Springer. 2016, pp. 173–184.

[56] Sergey Levine, Nolan Wagener, and Pieter Abbeel. "Learning contact-rich manipulation skills with guided policy search". In: *IEEE Int'l Conf. on Robotics and Automation*. 2015, pp. 156–163.

[57] Lihong Li, Thomas Walsh, and Michael Littman. "Towards a unified theory of state abstraction for MDPs". In: *Int'l Symp. on Artificial Intelligence and Mathematics*. 2006.

[58] Miao Li, Kaiyu Hang, Danica Kragic, and Aude Billard. "Dexterous grasping under shape uncertainty". In: *Robotics and Autonomous Systems* 75 (2016), pp. 352–364.

[59] Hongzhuo Liang, Xiaojian Ma, Shuang Li, Michael Görner, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang. "PointNetGPD: detecting grasp configurations from point sets". In: *IEEE Int'l Conf. on Robotics and Automation*. 2019, pp. 3629–3635.

[60] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[61] Tomas Lozano-Pérez. "Motion planning and the design of orienting devices for vibratory part feeders". In: *MIT Artificial Intelligence Laboratory Technical Report* (1986).

[62] Tomás Lozano-Pérez, Joseph Jones, Emmanuel Mazer, and Patrick O'Donnell. "Task-level planning of pick-and-place robot motions". In: *Computer* 22.3 (1989), pp. 21–29.

[63] Jens Lundell, Francesco Verdoja, and Ville Kyrki. "Beyond top-grasps through scene completion". In: *arXiv preprint arXiv:1909.12908* (2020).

[64] Jens Lundell, Francesco Verdoja, and Ville Kyrki. "Robust grasp planning over uncertain shape completions". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. 2019, pp. 1526–1532.

[65] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Ojea, and Ken Goldberg. "Dex-net 2.0: deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *Robotics: Science and Systems* 13 (2017).

[66] Jeffrey Mahler, Sachin Patil, Ben Kehoe, Jur van den Berg, Matei Ciocarlie, Pieter Abbeel, and Ken Goldberg. "GP-GPIS-OPT: grasp planning with shape uncertainty using gaussian process implicit surfaces and sequential convex programming". In: *IEEE Int'l Conf. on Robotics and Automation*. 2015, pp. 4919–4926.

[67] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. "kPAM: keypoint affordances for category-level robotic manipulation". In: *Int'l Symp. on Robotics Research*. 2019.

[68] Andrew Miller and Peter Allen. "Graspit! a versatile simulator for robotic grasping". In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122.

[69] George Miller. "Wordnet: a lexical database for english". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[70] Chaitanya Mitash, Rahul Shome, Bowen Wen, Abdeslam Boularias, and Kostas Bekris. "Task-driven perception and manipulation for constrained placement of unknown objects". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5605–5612.

[71] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. 2016, pp. 1928–1937.

[72] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. "Recurrent models of visual attention". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2204–2212.

[73] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Has-

sabis. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[74] Manfred Morari and Jay Lee. “Model predictive control: past, present and future”. In: *Computers & Chemical Engineering* 23.4-5 (1999), pp. 667–682.

[75] Douglas Morrison, Peter Corke, and Jürgen Leitner. “Closing the loop for robotic grasping: a real-time, generative grasp synthesis approach”. In: *Robotics: Science and Systems*. 2018.

[76] Douglas Morrison, Peter Corke, and Jürgen Leitner. “Multi-view picking: next-best-view reaching for improved grasping in clutter”. In: *IEEE Int’l Conf. on Robotics and Automation*. 2019.

[77] Douglas Morrison, Adam Tow, Matt Mctaggart, R Smith, Norton Kelly-Boxall, Sean Wade-Mccue, Jordan Erskine, Riccardo Grinover, Alec Gurman, T Hunn, et al. “Cartman: the low-cost cartesian manipulator that won the Amazon robotics challenge”. In: *IEEE Int’l Conf. on Robotics and Automation*. 2018, pp. 7757–7764.

[78] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. “6-DOF GraspNet: variational grasp generation for object manipulation”. In: *IEEE/CVF Int’l Conf. on Computer Vision*. 2019, pp. 2901–2910. DOI: 10.1109/ICCV.2019.00299.

[79] Richard Murray, Zexiang Li, and Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[80] Christian Nielsen and Lydia Kavraki. “A two level fuzzy prm for manipulation planning”. In: *IEEE/RSJ Int’l Conf. on Intelligent Robots and Systems*. Vol. 3. 2000, pp. 1716–1721.

[81] Jun Ota. “Rearrangement of multiple movable objects: integration of global and local planning methodology”. In: *IEEE Int’l Conf. on Robotics and Automation*. Vol. 2. 2004, pp. 1962–1967.

[82] Jun Ota. “Rearrangement planning of multiple movable objects by using real-time search methodology”. In: *IEEE Int’l Conf. on Robotics and Automation*. Vol. 1. 2002, pp. 947–953.

[83] Christos Papadimitriou and John Tsitsiklis. “The complexity of markov decision processes”. In: *Mathematics of operations research* 12.3 (1987), pp. 441–450.

[84] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. “Grasp pose detection in point clouds”. In: *The Int’l Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473.

[85] Andreas ten Pas and Robert Platt. "Localizing handle-like grasp affordances in 3d point clouds". In: *Int'l Symp. on Experimental Robotics*. 2014.

[86] Andreas ten Pas and Robert Platt. "Using geometry to detect grasps in 3d point clouds". In: *CoRR* abs/1501.03100 (2015). URL: `http://arxiv.org/abs/1501.03100`.

[87] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: learning to grasp from 50k tries and 700 robot hours". In: *IEEE Int'l Conf. on Robotics and Automation*. 2016, pp. 3406–3413.

[88] Robert Platt, Colin Kohler, and Marcus Gualtieri. "Deictic image maps: an abstraction for learning pose invariant manipulation policies". In: *AAAI Conf. on Artificial Intelligence*. 2019.

[89] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas Guibas. "PointNet++: deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems*. 2017, pp. 5099–5108.

[90] Charles Qi, Hao Su, Kaichun Mo, and Leonidas Guibas. "PointNet: deep learning on point sets for 3D classification and segmentation". In: *Conf. on Computer Vision and Pattern Recognition*. IEEE. 2017, pp. 652–660.

[91] Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. "Deep reinforcement learning for vision-based robotic grasping: a simulated comparative evaluation of off-policy methods". In: *IEEE Int'l Conf. on Robotics and Automation*. 2018.

[92] G. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. CUED/F-INFENG/TR 166. Cambridge University Engineering Department, Sept. 1994.

[93] Radu Rusu. "Semantic 3d object maps for everyday manipulation in human living environments". PhD thesis. Technical University of Munich, Oct. 2009.

[94] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey". In: *The Int'l Journal of Robotics Research* 37.7 (2018), pp. 688–716.

[95] Ashutosh Saxena, Justin Driemeyer, and Andrew Ng. "Robotic grasping of novel objects using vision". In: *The Int'l Journal of Robotics Research* 27.2 (2008), pp. 157–173.

[96] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. "Finding locally optimal, collision-free trajectories with sequential convex optimization." In: *Robotics: Science and Systems*. Vol. 9. 1. 2013, pp. 1–10.

[97] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. "Convergence results for single-step on-policy reinforcement-learning algorithms". In: *Machine learning* 38.3 (2000), pp. 287–308.

[98] Richard Smallwood and Edward Sondik. "The optimal control of partially observable markov processes over a finite horizon". In: *Operations research* 21.5 (1973), pp. 1071–1088.

[99] Nathan Sprague and Dana Ballard. "Eye movements for reward maximization". In: *Advances in Neural Information Processing Systems*. 2004, pp. 1467–1474.

[100] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[101] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. "Manipulation planning among movable obstacles". In: *IEEE Int'l Conf. on Robotics and Automation*. 2007, pp. 3327–3332.

[102] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. "Integral human pose regression". In: *European Conf. on Computer Vision*. 2018, pp. 529–545.

[103] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. Second. MIT Press Cambridge, 2018.

[104] Gabriel Taubin. "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11 (1991), pp. 1115–1138.

[105] Gerald Tesauro. "TD-Gammon, a self-teaching backgammon program, achieves master-level play". In: *Neural computation* 6.2 (1994), pp. 215–219.

[106] Gerald Tesauro and Gregory Galperin. "On-line policy improvement using Monte-Carlo search". In: *Advances in Neural Information Processing Systems*. 1997, pp. 1068–1074.

[107] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. 2005.

[108] Philip Torr and Andrew Zisserman. "MLESAC: a new robust estimator with application to estimating image geometry". In: *Computer Vision and Image Understanding* 78.1 (2000), pp. 138–156.

[109] Pierre Tournassoud, Tomás Lozano-Pérez, and Emmanuel Mazer. "Regrasping". In: *IEEE Int'l Conf. on Robotics and Automation.* Vol. 4. 1987, pp. 1924–1928.

[110] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. "Deep object pose estimation for semantic robotic grasping of household objects". In: *Proceedings of The 2nd Conference on Robot Learning.* Vol. 87. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 306–316.

[111] Jacob Varley, Chad DeChant, Adam Richardson, Joaquín Ruales, and Peter Allen. "Shape completion enabled robotic grasping". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems.* 2017, pp. 2442–2447.

[112] Ulrich Viereck, Andreas ten Pas, Kate Saenko, and Robert Platt. "Learning a visuomotor controller for real world robotic grasping using easily simulated depth images". In: *Proceedings of the Conf. on Robot Learning.* 2017.

[113] Weiwei Wan, Hisashi Igawa, Kensuke Harada, Hiromu Onda, Kazuyuki Nagata, and Natsuki Yamanobe. "A regrasp planning component for object reorientation". In: *Autonomous Robots* 43.5 (2019), pp. 1101–1115.

[114] Dian Wang, Colin Kohler, Andreas ten Pas, Alexander Wilkinson, Maozhi Liu, Holly Yanco, and Robert Platt. "Towards assistive robotic pick and place in open world environments". In: *arXiv preprint arXiv:1809.09541* (2019).

[115] Dian Wang, Colin Kohler, and Robert Platt. *Policy learning in SE(3) action spaces.* 2020. arXiv: `2010.02798 [cs.RO]`.

[116] Gerhard Wäscher, Heike Haußner, and Holger Schumann. "An improved typology of cutting and packing problems". In: *European journal of operational research* 183.3 (2007), pp. 1109–1130.

[117] Christopher Watkins. "Learning from delayed rewards". PhD thesis. King's College, Cambridge, 1989.

[118] Christopher Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[119] Steven Whitehead and Dana Ballard. "Learning to perceive and act by trial and error". In: *Machine Learning* 7.1 (1991), pp. 45–83.

[120] Gordon Wilfong. "Motion planning in the presence of movable obstacles". In: *Annals of Mathematics and Artificial Intelligence* 3.1 (1991), pp. 131–150.

[121] Ronald Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[122] Walter Wohlkinger, Aitor Aldoma, Radu Rusu, and Markus Vincze. "3DNet: large-scale object class recognition from CAD models". In: *IEEE Int'l Conf. on Robotics and Automation*. 2012, pp. 5384–5391.

[123] Bohan Wu, Iretiayo Akinola, and Peter Allen. "Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. 2019.

[124] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D ShapeNets: a deep representation for volumetric shapes". In: *IEEE Conf. on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[125] Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen, and Christopher Amato. "Online planning for target object search in clutter under partial observability". In: *IEEE Int'l Conf. on Robotics and Automation*. 2019, pp. 8241–8247.

[126] Bo Yang, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. "Learning object bounding boxes for 3D instance segmentation on point clouds". In: *Advances in Neural Information Processing Systems*. 2019, pp. 6737–6746.

[127] Kuan-Ting Yu, John Leonard, and Alberto Rodriguez. "Shape and pose recovery from planar pushing". In: *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*. 2015, pp. 1208–1215.

[128] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. "PCN: point completion network". In: *Int'l Conf. on 3D Vision*. 2018, pp. 728–737.

[129] Brayan Zapata-Impata, Vikrant Shah, Hanumant Singh, and Robert Platt. "AutOTranS: an autonomous open world transportation system". In: *CoRR* abs/1810.03400 (2018). arXiv: `1810.03400`. URL: `http://arxiv.org/abs/1810.03400`.

[130] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan-Dafle, Rachel Holladay, Isabella Morona, Qu-Nair Prem, Druck Green, Ian Taylor, Weber Liu, Thomas Funkhouser, and Alberto Rodriguez. "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching". In: *IEEE Int'l Conf. on Robotics and Automation*. 2018.

[131] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. "Multi-view self-supervised deep learning for 6D pose estimation in the Amazon picking challenge". In: *IEEE Int'l Conf. on Robotics and Automation*. 2017, pp. 1386–1383.