Learning Visual Affordances for Robotic Manipulation

Andy Zeng

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Thomas Funkhouser

November 2019

Abstract

A humans remarkable ability to manipulate unfamiliar objects with little prior knowledge of them is a constant inspiration for robotics research. Despite the interest of the research community, and despite its practical value, robust manipulation of novel objects in cluttered environments still remains a largely unsolved problem. Classic solutions (*e.g.* involving 6D object pose estimation) typically require prior knowledge of the objects (*e.g.* class categories or 3D CAD models), which may not be available outside of highly constrained settings. More recent deep learning methods using end-to-end convolutional networks (e.g. raw pixels to motor torques) have the potential to model complex skills that generalize, but they remain highly data inefficient – and robot data (*e.g.* trial and error) is expensive.

In this thesis, we consider an approach to learning manipulation called visual affordances. The idea is to use classic controllers to design motion primitives, then use convolutional networks to map from visual observations (*e.g.* images) to the perceived affordances (*e.g.* confidence scores or action-values) of the primitives for every pixel of the input. By leveraging dense equivariant state and action representations, this formulation can be used to acquire complex vision-based manipulation skills (*e.g.* pushing, grasping, throwing) on real robot platforms that generalize to novel objects, while using orders of magnitude less data. While visual affordances may not be directly compatible with classic planning frameworks that involve explicit forward simulation or propagation, in this thesis we show that it is possible to workaround this limitation by extending it with model-free reinforcement learning to sequence primitive picking motions for more complex manipulation policies. We also study how it can be combined with residual physics (learning to predict residual values on top of control parameter estimates from an initial analytical controller) to enable learning end-to-end visuomotor policies that leverage the benefits of analytical models while still maintaining the capacity (via data-driven residuals) to account for real-world

dynamics that are not explicitly modeled. Finally, we conclude by discussing the limitations of learning visual affordances, which suggest directions for future work.

Acknowledgements

First and foremost, I would like to thank my advisor Thomas Funkhouser for his continuous support and guidance throughout my PhD, and for providing me with the independence and freedom to work on a variety of research problems. I'm also incredibly grateful to have worked with Alberto Rodriguez, who is an excellent role model and inspired me to pursue a career in robotics. Thank you to my thesis committee members Szymon Rusinkiewicz, Olga Russakovsky, and Adam Finkelstein.

I'm tremendously grateful for my friends and collaborators from the MIT-Princeton Amazon Picking Challenge Team, including Shuran Song, Peter (Kuan-Ting) Yu, Elliott Donlon, Daniel Suo, Ed Walker Jr., Jianxiong Xiao, Francois R. Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, and Weber Liu. Taking part in the challenge together was both a thrilling and humbling learning experience. I've also had the pleasure of working with a number of other students and faculty including Angel X. Chang, Manolis Savva, Matthias Niener, Matthew Fisher, Angela Dai. I'm also thankful for my labmates at Princeton, including Maciej Halber, Kyle Genova, Daniel Suo, Ari Seff, Riley Simmons-Edler, Yinda Zhang, Fisher Yu, Xinyi Fan, Elena Balashova, and Sema Berkiten. Our daily tea runs were both amazing and unforgettable.

I'm also very fortunate to have had the opportunity to spend a large part of my PhD as an intern at Google Brain with Johnny Lee, who is an inspiring mentor and friend. During my internship, I've enjoyed working with many team members, including Ivan Krasin, Ryan Hickman, and Stefan Welker.

Finally, I'd like to thank my parents, sister, and grandparents for their love and support.

To my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A humans remarkable ability to manipulate unfamiliar objects with little prior knowledge of them is a constant inspiration for robotics research. This ability to grasp the unknown is central to many applications: from picking packages in a logistic center to bin-picking in a manufacturing plant; from unloading groceries at home to clearing debris after a disaster.

Despite the interest of the research community, and despite its practical value, robust manipulation of novel objects in cluttered environments still remains a largely unsolved problem. Classic solutions for robotic picking require recognition and pose estimation prior to model-based grasp planning, or require object segmentation to associate grasp detections with object identities. These solutions tend to fall short when dealing with novel objects in cluttered environments, since they rely on prior information about the objects (*e.g.* class categories, 3D object CAD models) which may not be available.

Borrowing from the recent success of deep learning in computer vision, recent work suggests that it's possible to train end-to-end convolutional networks (*e.g.* mapping from raw pixels to control torques) to learn complex robotic manipulation skills. In practice however, these models experience limited generalization to novel settings and

remain highly data inefficient often requiring massive amounts of robot data collection through trial and error, which is both time-consuming and expensive. Furthermore, these methods have yet to be proven in the constraints and accuracy required by a real application with heavy clutter, severe occlusions, and object variability.

This outcome begs the question: why do the convolutional networks that work well for computer vision tasks remain so difficult to train for robotic manipulation? In this work, we argue that this problem is highly related to the architecture and data representations of these models. The best performing convolutional network architectures built for vision tasks (*e.g.* image classification) consist of modules that induce rotational and translational invariance (*e.g.* with pooling layers, fully connected layers, or data augmentation) across input image data. This is important for vision tasks like image classification, which benefit from being able to output the same labels for an object even as it appears under different rotational and translational perturbations. However, vision-based robotic manipulation in its rawest form (pixels to torques) does not necessarily benefit from such invariance. The same manipulation exerted on the same object, observed in two different locations, may require very different output torque trajectories. Instead, vision-based robotic manipulation may benefit from spatial equivariance. For example, if we change our output representation to be spatial control parameters (*e.g.* positioning of the end effector) for motion primitives, then as an object (*e.g.* a mug) translates or rotates in the input image, the output end effector positioning should change (*e.g.* translate or rotate) with the object respectively. As the result, the nuances of learning how to grasp the mug from one angle, could transfer to a different orientation of the mug. Generally, if we seek to enable end-to-end learning for robotic manipulation, we should aim to design network architectures and data representations that maximize spatial equivariance.

In this thesis, we introduce an approach to learning manipulation called visual affordances, which uses fully convolutional networks (FCNs) and dense state/action

representations to maximize spatial equivariance. The idea is to use classic controllers to design motion primitives, then use FCNs to map from visual observations (*e.g.* images) to the perceived affordances (*e.g.* confidence scores or action-values) of the primitives for every pixel in the input. For example, to learn the visual affordances of picking up objects with a suction cup, we can use a fully convolutional network that takes as input an RGB-D image of the scene (*e.g.* a bin full of objects), and output the dense pixel-wise probability of picking success for every observable surface in the image. The robot then executes a scripted motion primitive that controls the robot end effector equipped with a suction cup to contact the 3D location (assuming calibration) of the pixel with the highest predicted affordance value from the network. In this example, both the network architecture and action representation (dense pixel-wise sampling of end effector locations) provide translational equivariance for picking up objects with a suction cup *i.e.*, the predicted values of the pixel regions that belong to a good suction point on an object translates with the object as it moves around.

The general concept behind visual affordances is not new, and hearkens back to a theory from psychology developed in the 1970s by James J. Gibson, who theorized that animals (including humans) could directly perceive the environment by what it has to offer them [31]. In other words, visual stimuli that they perceive directly maps to information about the physical properties of different objects in the environment and what they can or cannot do with them (*e.g.* action possibilities). This theory has been revisited in prior work on the design and development of artificially intelligent agents [44]. In the advent of deep learning, this thesis provides a new perspective to learning visual affordances for robotic manipulation with convolutional networks.

Our formulation for learning visual affordances is simple, but yields surprisingly effective results. In this thesis, we show that it enables robotic systems to learn a variety of complex vision-based manipulation skills (*e.g.* pushing, grasping, throwing) that generalize to novel objects while using orders of magnitude less data. Visual

affordances is not the only approach that can facilitate spatial equivariance; but we hope that by highlighting the importance of spatial equivariance in learning manipulation, the ideas introduced in this thesis can inform key design decisions for future manipulation algorithms. Our contributions are as follows:

- In Chapter 2, we develop a vision algorithm that incorporates deep learning into the classic manipulation pipeline by leveraging convolutional networks for 6D object pose estimation. We describe its integration into a real world picking system, and discuss its limitations. The approach was part of the MIT-Princeton Team system that took 3rd place at the 2016 Amazon Picking Challenge, and appeared previously as Zeng et al. (2017) [123].

- In Chapter 3, we present a pick-and-place system that can grasp (with suction and parallel-jaw grasping) and recognize novel objects (appearing for the first time during testing) in cluttered environments without needing any additional data collection or re-training. At the core of the manipulation system, lies our main contribution: visual affordances. We compare it to prior grasping algorithms, and discuss how it addresses the limitations of the algorithm presented in Chapter 2. The approach was part of the MIT-Princeton Team system that took 1st place in the stowing task at the 2017 Amazon Robotics Challenge, and appeared previously as Zeng et al. (2018) [125].

- Chapter 4, we show that we can extend visual affordances using model-free reinforcement learning to discover and learn the synergies between non-prehensile (e.g. pushing) and prehensile (e.g. grasping) actions. The method remains sample efficient and generalizes to novel objects and scenarios. This work was published previously as Zeng et al. (2018) [124].

- Chapter 5, we show that we can combine visual affordances with residual physics (learning to predict residual values on top of control parameter estimates from an

initial analytical controller) to learn how to grasp and throw arbitrary objects for pick-and-place. This formulation enables learning end-to-end visuomotor policies that leverage the benefits of analytical models (e.g. generalization) while still maintaining the capacity (via data-driven residuals) to account for real-world dynamics that are not explicitly modeled. This work appeared previously as Zeng et al. (2019) [126].

- Finally, we conclude by summarizing our contributions and discussing open challenges for future work in Chapter 6.

Chapter 2

6D Object Pose Estimation for Manipulation with Deep Learning

In this chapter, we develop a vision algorithm that incorporates deep learning into the classic manipulation pipeline by leveraging convolutional networks for 6D object pose estimation. Specifically, we present an approach that leverages multi-view RGB-D data and self-supervised, data-driven learning to reliably recognize and locate objects amid cluttered environments, self-occlusions, sensor noise, and a large variety of objects. In the proposed approach, we segment and label multiple views of a scene with a fully convolutional neural network, and then fit pre-scanned 3D object models to the resulting segmentation to get the 6D object pose. Training a deep neural network for segmentation typically requires a large amount of training data. We propose a self-supervised method to generate a large labeled dataset without tedious manual segmentation. We demonstrate that our system can reliably estimate the 6D pose of objects under a variety of scenarios. The approach was part of the MIT-Princeton Team system that took 3rd- and 4th- place in the stowing and picking tasks, respectively at APC 2016.

2.1 6D Object Pose Estimation for Manipulation

The last two decades have seen a rapid increase in warehouse automation technologies, satisfying the growing demand of e-commerce and providing faster, cheaper delivery. Some tasks, especially those involving physical interaction, are still hard to automate. Amazon, in collaboration with the academic community, has led a recent effort to define two such tasks: 1) *picking* an instance of a given a product ID out of a populated shelf and place it into a tote; and 2) *stowing* a tote full of products into a populated shelf.

In this chapter we describe the vision system of the MIT-Princeton Team, that took 3rd place in the stowing task and 4th in the picking task at the 2016 Amazon Picking Challenge (APC), and provide experiments to validate our design decisions. Our vision algorithm estimates the 6D poses of objects robustly under challenging scenarios:

- **Cluttered environments**: shelves and totes may have multiple objects and could be arranged as to deceive vision algorithms (e.g., objects on top of one another).
- **Self-occlusion**: due to limited camera positions, the system only sees a partial view of an object.
- **Missing data**: commercial depth sensors are unreliable at capturing reflective, transparent, or meshed surfaces, all common in product packaging.
- **Small or deformable objects**: small objects provide fewer data points, while deformable objects are difficult to align to prior models.
- **Speed**: the total time dedicated to capturing and processing visual information is under 20 seconds.

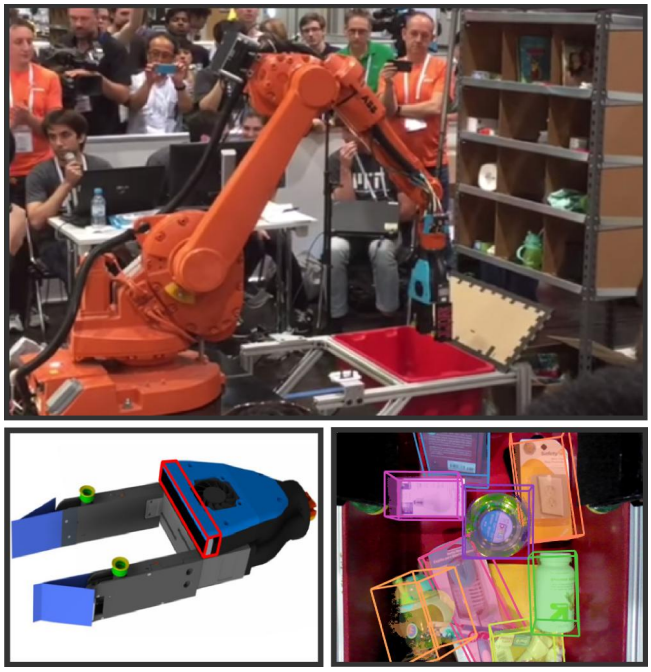


Figure 2.1: Top: The MIT-Princeton robotic picking system. Bottom-left: The gripper mounted with an Intel RealSense camera (outlined in red). Bottom-right: Predicted 6D object poses from our vision system during the stow-task finals of the APC 2016. Each prediction is highlighted with a colored 3D bounding box.

Our approach makes careful use of known constraints in the task—the list of possible objects and the expected background. The algorithm first segments the object from a scene by feeding multiple-view images to a deep neural network and then fits a 3D model to a segmented point cloud to recover the object's 6D pose. The deep neural network provides speed, and in combination with a multiple-view approach boosts performance in challenging scenarios.

Training a deep neural network for segmentation requires a large amount of labeled training data. We have developed a self-supervised training procedure that automatically generated 130,000 images with pixel-wise category labels of the 39 objects in the APC. For evaluation, we constructed a testing dataset of over 7,000 manually-labeled images.

In summary, this chapter contributes with:

- A robust multi-view vision system to estimate the 6D pose of objects;

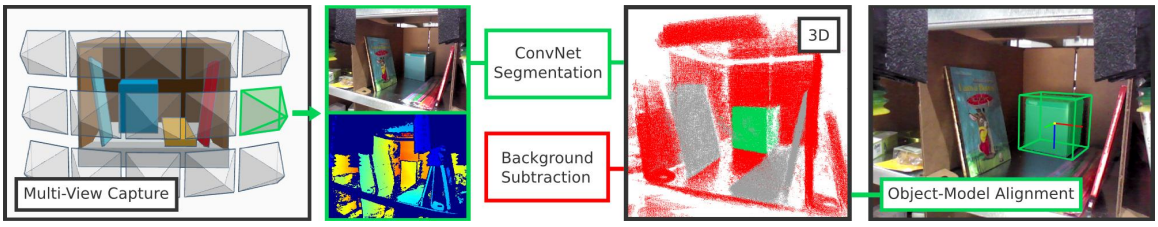


Figure 2.2: Overview of the vision algorithm. The robot captures color and depth images from 15 to 18 viewpoints of the scene. Each color image is fed into a fully convolutional network [64] for 2D object segmentation. The result is integrated in 3D. The point cloud will then go through background removal and then aligned with a pre-scanned 3D model to obtain its 6D pose.

- A self-supervised method that trains deep networks by automatically labeling training data;
- A benchmark dataset for estimating object poses.

All code, data, and benchmarks are publicly available [1].

2.2 Related Work

Vision algorithms for robotic manipulation typically output 2D bounding boxes, pixel-level segmentation [55, 26], or 6D poses [127, 121] of the objects. The choice depends primarily on manipulation needs. For example, a suction based picker might have sufficient information with a 2D bounding box or with a pixel-level segmentation of the object, while a grasper might require its 6D pose.

Object segmentation. While the 2015 APC winning team used a histogram back-projection method [63] with manually defined features [26, 55], recent work in computer vision has shown that deep learning considerably improves object segmentation [64]. In this work, we extend the state-of-the-art deep learning architecture used for image segmentation to incorporate depth and multi-view information.

Pose estimation. There are two primary approaches for estimating the 6D pose of an object. The first aligns 3D CAD models to 3D point clouds with algorithms

such as iterative closest point [12]. The second uses more elaborated local descriptors such as SIFT keypoints [23] for color data or 3DMatch [122] for 3D data. The former approach is mainly used with depth-only sensors, in scenarios where lighting changes significantly, or on textureless objects. Highly textured and rigid objects, on the other hand, benefit from local descriptors. Existing frameworks such as LINEMOD [41] or MOPED [17] work well under certain assumptions such as objects sitting on a table top with good illumination, but underperform when confronted with the limited visibility, shadows, and clutter imposed by the APC scenario [18].

Benchmark for 6D pose estimation. To properly evaluate our vision system independent from the larger robotic system, we have produced a large benchmark dataset with scenarios from APC 2016 with manual labels for objects' segmentation and 6D poses. Previous efforts to construct benchmark datasets include Berkeley's dataset [110] with a number of objects from and beyond APC 2015 and Rutgers's dataset [95] with semi-automatically labeled data.

2.3 Amazon Picking Challenge 2016

The APC 2016 posed a simplified version of the general picking and stowing tasks in a warehouse. In the picking task, robots sit within a 2x2 meter area in front of a shelf populated with objects, and autonomously pick 12 desired items and place them in a tote. In the stowing task, robots pick all 12 items inside a tote and place them in a pre-populated shelf.

Before the competition, teams were provided with a list of 39 possible objects along with 3D CAD models of the shelf and tote. At run-time, robots were provided with the initial contents of each bin in the shelf and a work-order containing which items to pick. After picking and stowing the appropriate objects, the system had to report the final contents of both shelf and tote.

2.4 System Description

Our vision system takes in RGB-D images from multiple views, and outputs 6D poses and a segmented point cloud for the robot to complete the picking and stowing tasks.

The camera is compactly integrated in the end-effector of a 6DOF industrial manipulator ABB IRB1600id, and points at the tip of the fingers (2.1). This configuration gives the robot full controllability of the camera viewpoint, and provides feedback about grasp or suction success. The camera of choice is the RealSense F200 RGB-D because its depth range (0.2m–1.2m) is appropriate for close manipulation, and because it is a consumer-level range sensor with a decent amount of flexibility on the data capture process.

Due to the tight integration of the camera, the gripper fingers, even when fully open, occupy a small portion of the view frustum. We overcome this limitation by combining different viewpoints, making use of the accurate forward kinematic reported by the robot controller.

2.5 6D Object Pose Estimation

The algorithm estimates the 6D pose of all objects in a scene in two phases (Figure 2.2). First, it segments the RGB-D point clouds captured from multiple views into different objects using a deep convolutional neural network. Second, it aligns prescanned 3D models of the identified objects to the segmented point clouds to estimate the 6D pose of each object. Our approach is based on well-known methods. However, our evaluations show that when used alone, they are far from sufficient. In this section we present brief descriptions of these methods followed by in-depth discussions of how we combine them into a robust vision system.

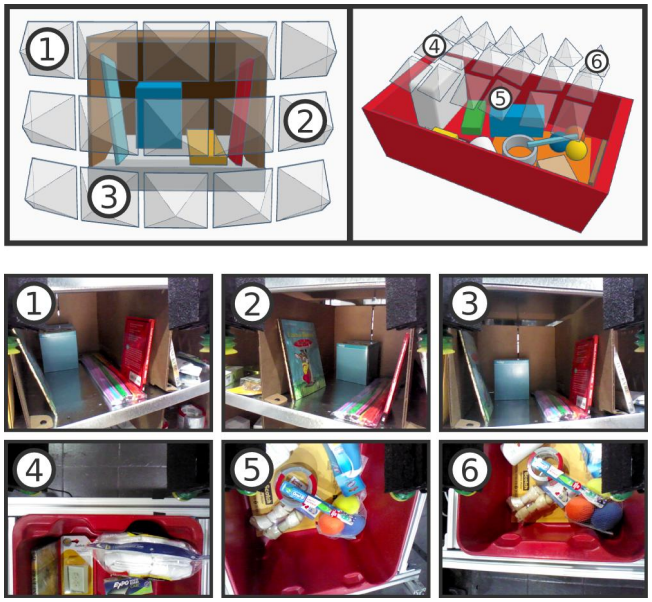


Figure 2.3: Camera viewpoints of the RGB-D frames captured for bins and tote, and captured color images from 6 selected viewpoints. The 15 viewpoints of a shelf bin (upper-left) are arranged in a 3x5 grid. The 18 viewpoints of a tote (upper-right) are arranged in a 3x6 grid.

2.5.1 Object Segmentation with Fully Convolutional Networks

In recent years, ConvNets have made tremendous progress for computer vision tasks [33, 64]. We leverage these advancements to segment camera data into the different objects in the scene. More explicitly, we train a VGG architecture [109] Fully Convolutional Network (FCN) [64] to perform 2D object segmentation. The FCN takes an RGB image as input and returns a set of 40 densely labeled pixel probability maps–one for each of the 39 objects and one for the background–of the same dimensions as the input image.

Object segmentation using multiple views. Information from a single camera view and from a given object, is often limited due to clutter, self-occlusions, and bad reflections. We address the missing information during the model-fitting phase by combining information from multiple views so that the object surfaces are more

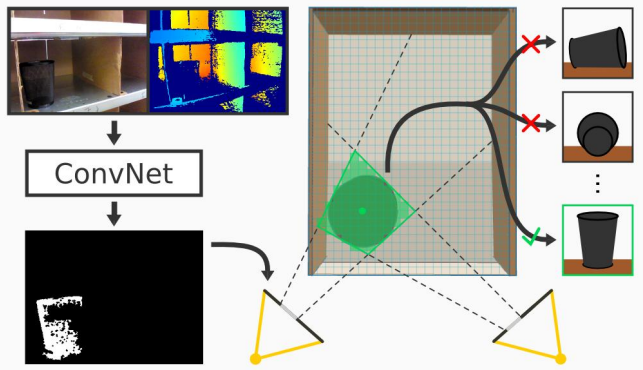


Figure 2.4: Pose estimation for objects with no depth. 2D object segmentation results from a fully convolutional network are triangulated between the different camera views to create a 3D convex hull (green) of the object. For simplicity, only two camera views (yellow) are illustrated. The centroid and aspect ratio of the convex hull are used to estimate the geometric center of the object and its orientation (from a predefined set of expected orientations).

distinguishable. In particular, we feed the RGB images captured from each viewpoint (18 for stowing from the tote and 15 for picking from the shelf) to the trained FCN, which returns a 40 class probability distribution for each pixel in each RGB-D image. After filtering by the list of expected objects in the scene, we threshold the probability maps (three standard deviations above the mean probability across all views) and ignore any pixels whose probabilities for all classes are below these thresholds. We then project the segmented masks for each object class into 3D space and directly combine them into a single segmented point cloud with the forward kinematic feedback from the robot arm (note that segmentation for different object classes can overlap each other).

Reduce noise in point cloud. Fitting pre-scanned models to the segmented point cloud directly often gives poor results because of noise from the sensor and noise from the segmentation. We address this issue in three steps: First, to reduce sensor noise, we eliminate spatial outliers from the segmented point cloud, by removing all point farther than a threshold from its k-nearest neighbors. Second, to reduce segmentation noise, especially on the object boundaries, we remove points that lie outside the shelf

bin or tote, and those that are close to a pre-scanned background model. Finally, we further filter outlier points from each segmented group of points by finding the largest contiguous set of points along each principal axis (computed via PCA) and remove any points that are disjoint from this set.

Handle object duplicates. Warehouse shelves commonly contain multiple instances of the same object. Naively segmenting RGB-D data will treat two distinct object with the same label as the same object. Since we know the inventory list in the warehouse setting, we know the number of identical objects we expect in the scene. We make use of k-means clustering to separate the segmented and aggregated point cloud into the appropriate number of objects. Each cluster is then treated independently during the model-fitting phase of the algorithm.

2.5.2 3D Model-Fitting

We use the iterative closest point (ICP) algorithm [97] on the segmented point cloud to fit pre-scanned 3D models of objects and estimate their poses. The vanilla ICP algorithm, however, gives nonsensical results in many scenarios. We describe here several such pitfalls along with our solutions.

Point clouds with non-uniform density. In a typical RGB-D point cloud, surfaces perpendicular to the sensor's optical axis have often denser point clouds. The color of the surface changes its reflectivity on the IR spectrum, which also affects the effective point cloud density. These non-uniformities are detrimental to the ICP algorithm because it biases convergence toward denser areas. By applying a 3D uniform average grid filter to the point clouds, we are able to give them consistent distributions in 3D space.

Pose initialization. ICP is an iterative local optimizer, and as such, it is sensitive to initialization. The principal directions of the segmented point cloud, as estimated by PCA, give us a reasonable first approximation to the orientation of objects with

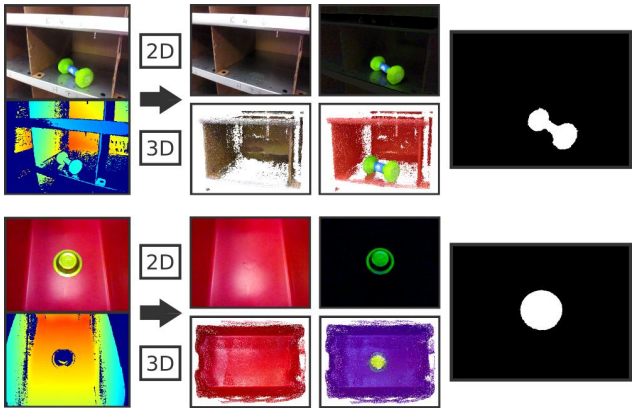


Figure 2.5: To automatically obtain pixel-wise object labels, we separate the target objects from the background to create an object mask. There are a 2D and a 3D component in this data process. Both use color and depth information. The 2D pipeline is robust to thin objects and objects with no depth, while the 3D pipeline is robust to an unstable background.

uneven aspect ratios. We have observed experimentally that the choice of initial orientation for objects with even aspect ratios has little effect on the final result of ICP. Analogously, one would use the centroid of the point cloud as the initial guess for the geometric center of the object, however we have observed that since captured point clouds are only partial, those two centers are usually biased from each other. To address this, we push back the initial pose of the pre-scanned object back along the optical axis of the RGB-D camera by half the size of the object's bounding box, under the naive assumption that we are only seeing "half" the object. This initialization has proven more successful in avoiding local optimums.

Coarse to fine ICP. Even after reducing noise in the segmentation step, the resulting point cloud may still have noise (e.g., mislabeled points from adjacent objects). We address this with two passes of ICP, acting on different subsets of the point cloud: we define the *inlier* threshold of an ICP iteration as the percentile L2 distance above which we ignore. ICP with a 90% inlier ratio keeps the closest pairs of points between the two point clouds up to the 90th percentile. The main assumption is that regions of the point cloud that are correctly labeled are denser than regions with incorrect label.

A first pass with a high inlier threshold (90%) moves the pre-scanned complete model closer to the correct portion of the partial view than the noisy portion. Starting now from a coarse but robust initialization, the second pass uses a lower inlier threshold (45%) to ignore the noisy portion of the point cloud and converge to a more accurate pose.

2.5.3 Handling Objects with Missing Depth.

Many objects in the APC, as it is typical in retail warehouses, have surfaces that challenge infrared-based depth sensors, e.g., with plastic wrapping returning noisy or multiple reflections, or transparent or meshed materials which may not register at all. For these objects the captured point cloud is noisy and sparse, and our pose estimation algorithm performs poorly.

Our solution leverages the multi-view segmentation to estimate a convex hull of the object by carving a 3D gridded space of voxels with the segmented RGB images. This process results in a 3D mask that encapsulates the real object. We use the convex hull of that mask to estimate the geometric center of the object and approximate its orientation (assuming that the object is axis-aligned).

2.6 Self-supervised Training

By bringing deep learning into the approach we gain robustness. This, however, comes at the expense of amassing quality training data, which is necessary to learn high-capacity models with many parameters. Gathering and manually labeling such large amounts of training data is expensive. The existing large-scale datasets used by deep learning (e.g. ImageNet [98]) are mostly Internet photos, which have very different object and image statistics from our warehouse setting.

To automatically capture and pixel-wise label images, we propose a self-supervised method, based on three observations:

- Batch-training on scenes with a single object can yield deep models that perform well on scenes with multiple objects [33] (i.e., simultaneous training on cat-only or dog-only images enables successful testing on cat-with-dog images);
- An accurate robot arm and accurate camera calibration, gives us at will control over camera viewpoint;
- For single object scenes, with known background and known camera viewpoint, we can automatically obtain precise segmentation labels by foreground masking.

The captured training dataset contains 136,575 RGB-D images of 39 objects, all automatically labeled.

Semi-automatic data gathering. To semi-autonomously gather large quantities of training data, we place single known objects inside the shelf bins or tote in arbitrary poses, and configure the robot to move the camera and capture RGB-D images of the objects from a variety of different viewpoints. The position of the shelf/tote is known to the robot, as is the camera viewpoint, which we use to transform the collected RGB-D images in shelf/or tote frame. After capturing several hundred RGB-D images, the objects are manually re-arranged to different poses, and the process is repeated several times. Human involvement sums up to re-arranging the objects and labeling which objects correspond to which bin/tote. Selecting and changing the viewpoint, capturing sensor data, and labeling each image by object is automated. We collect RGB-D images of the empty shelf and tote from the same exact camera viewpoints to model the background, in preparation for the automatic data labeling.

Automatic data labeling. To obtain pixel-wise object segmentation labels, we create an object mask that separates foreground from background. The process is composed of 2D and 3D pipelines. The 2D pipeline is robust to thin objects (objects

not sufficient volume to be reliably segmented in 3D when placed too close to a walls or ground) and objects with no depth information, while the 3D pipeline is robust to large miss-alignments between the pre-scanned shelf bin and tote. Results from both pipelines are combined to automatically label an object mask for each training RGB-D image.

The 2D pipeline starts by fixing minor possible image misalignments by using multimodal 2D intensity-based registration to align the two RGB-D images [111]. We then convert the aligned color image from RGB to HSV, and do pixel-wise comparisons of the HSV and depth channels to separate and label foreground from background.

The 3D pipeline uses multiple views of an empty shelf bin and tote to create their pre-scanned 3D models. We then use ICP to align all training images to the background model, and remove points too close to the background to identify the foreground mask. Finally, we project the foreground points back to 2D to retrieve the object mask.

Training neural network. To leverage features trained from a larger image domain, we use the sizable FCN-VGG network architecture from [109] and initialize the network weights using a model pre-trained on ImageNet for 1000-way object classification. We fine-tune the network over the 40-class output classifier (39 classes for each APC object and 1 class for background) using stochastic gradient descent with momentum. Due to illumination and object viewpoint biases, we maximize performance by training two such segmentation networks: one for shelf bins and one for the tote. The segmentation labels automatically generated for the training data can be noisy. However, we find that the networks are still capable of working well during test time due to the sheer size of available training data.

Figure 2.6: Examples from our benchmark dataset. The dataset contains 477 scenes with 2,087 unique object poses seen from multiple viewpoints. In total, there are 7,713 images with manually-annotated ground truth 6D object poses and segmentation labels.

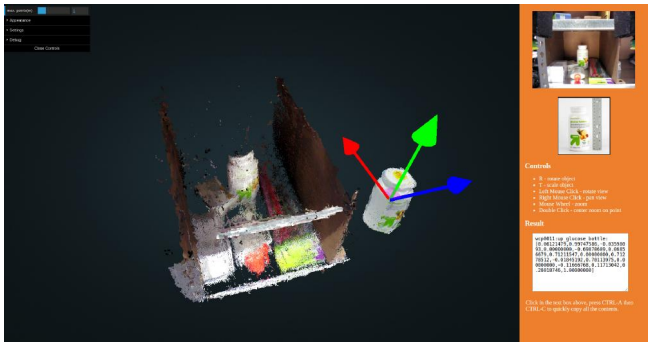


Figure 2.7: The 3D online annotation tool used to label the benchmark. The drag-and-drop UI allows annotators to navigate in 3D space and manipulate point clouds with ease. Annotators are instructed to move and rotate a pre-scanned object model to its ground truth location in a 3D point cloud generated from RGB-D data. Labeling one object takes about 1 minute.

2.7 Implementation

All components of the vision system are modularized into reusable ROS packages, with CUDA GPU acceleration. Deep models are trained and tested with Marvin [119], a ROS-compatible and lightweight deep learning framework. Training our models takes up to 16 hours prior to convergence.

Our robot is controlled by a computer with an Intel E3-1241 CPU 3.5 GHz and an NVIDIA GTX 1080. The run-time speeds per component are as follows: 10ms for ROS communication overhead, 400ms per forward pass of VGG-FCN, 1200ms for denoising per scene, and 800ms on model-fitting per object. On average, pose estimation time is 3-5 seconds per shelf bin and 8-15 seconds for the tote. Combined with multi-view robot motions, total visual perception time is 10-15 seconds per shelf bin and 15-20 seconds for the tote.

2.8 Evaluation

We evaluate variants of our method in different scenarios in the benchmark dataset to understand (1) how segmentation performs under different input modalities and training dataset sizes and (2) how the full vision system performs.

2.8.1 Benchmark Dataset

Our benchmark dataset, 'Shelf&Tote', contains over 7,000 RGB-D images spanning 477 (2.6) scenes at 640×480 resolution. We collected the data during practice runs and competition finals for the APC and manually labeled 6D object poses and segmentations using our online annotator (2.7). The data reflects various challenges found in the warehouse setting: reflective materials, variation in lighting conditions, partial views, and sensor limitations (noisy and missing depth) over cluttered environments.

Figure 2.8: Example results from our vision system. 6D pose predictions are highlighted with a 3D bounding box. For deformable objects (cloth in a,c,i) we output the center of mass. We additionally illustrate successful pose predictions for objects with missing depth (water bottle, black bin, green sippy cup, green bowl)

Figure 2.9: Several common failure cases. These include low-confidence predictions due to severe occlusion (missing object labels in m,o,p), completely incorrect pose predictions due to confusion in texture (m,p,r) or bad initialization (n,q), and model-fitting errors (o).

Tables 2.1 and 2.2 summarize our experimental results and highlight the differences in performance over different overlapping scene categories:

- **cptn:** during competition at the APC finals.
- **environment:** in an office (off); in the APC competition warehouse (whs).
- **task:** *picking* from a shelf bin or *stowing* from a tote.
- **clutter:** with multiple objects.
- **occlusion:** with % of object occluded by another object, computed from ground truth.
- **object properties:** with objects that are deformable, thin, or have no depth from the RealSense F200 camera.

2.8.2 Evaluating Object Segmentation

We test several variants of our FCN on object segmentation to answer two questions: (1) can we leverage both color and depth segmentation? (2) is more training data useful?

Metrics. We compare the predicted object segmentation from our trained FCNs against the ground truth segmentation labels of the benchmark dataset using pixel-wise precision and recall. Table 2.1 displays the mean average F-scores ($F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$).

Depth for segmentation. We use HHA features [38] to encode depth information into three channels: horizontal disparity, height above ground, and angle of local surface normal with the inferred direction of gravity. We compare AlexNet trained on this encoding, VGG on RGB data, and both networks concatenated in Table 2.1.

We find that adding depth does not yield any notable improvements in segmentation performance, which could be in part due to the noisiness of the depth information from our sensor. On the other hand, we observe that the FCN performs significantly better when trained on color data, with the largest disparity for deformable objects and thin objects, whose textures provide more discriminative power than their geometric structure.

Size of training data. Deep learning models have seen significant success, especially if given large amounts of training data. However in our scenario—instance-level object segmentation on few object categories—it is not clear whether such a large dataset is necessary. We create two new datasets by randomly sampling 1% and 10% of the original and use them to train two VGG FCNs (Table 2.1). We confirm marked improvements in F-score across all benchmark categories going from 1% to 10% to 100% of training data.

2.8.3 Evaluating Pose Estimation

We evaluate several key components of our vision system to determine whether they increase performance in isolation.

Metrics. We report the percentage of object pose predictions with error in orientation smaller than 15°, and the percentage with error in translation smaller than 5cm. The metric also recognizes the structural invariance of several objects, some of which are axially-symmetric (cuboids), radially-symmetric (bottles, cylinders), or deformable (see web page [1] for further details). We have observed experimentally that these bonds of 15° and 5cm are sufficient for picking with sensor-guarded motions.

Multi-view information. With multiple views the system overcomes missing information due to self-occlusions, other-object occlusions, or clutter. Multi-view information also alleviates problems with illumination on reflective surfaces.

To quantify the effect of the multiple-view system, we test the full vision system on the benchmark with three different subsets of camera views:

- [Full] All 15 views for shelf bins $a_{1_{\text{shelf}}} = \{0 \dots 14\}$ and all 18 views for the tote $a_{1_{\text{tote}}} = \{0 \dots 17\}$.
- [5v-10v] 5 views for shelf $a_{2_{\text{shelf}}} = \{0,4,7,10,14\}$ and 10 views for tote $a_{2_{\text{tote}}} = \{0,2,4,6,8,9,11,13,15,17\}$, with a sparse arrangement and a preference for wide-baseline view angles.
- [1v-2v] 1 view for shelf bins $a_{3_{\text{shelf}}} = \{7\}$ and 2 views for the tote $a_{3_{\text{tote}}} = \{7,13\}$.

The viewpoint ids are zero-indexed in row-major order as depicted in Figure 2.3. Our results show that multiple views robustly address occlusion and heavy clutter in the warehouse setting (Table 2.2 [clutter] and [occlusion]). They also present a clear contrast between the performance of our algorithm using a single view of the scene, versus multiple views of the scene (Table 2.2 [Full] v.s [1v-2v]).

Denoising. The denoising step described in 2.5 proves important for achieving good results. With this turned off, the accuracy in estimating the translation and rotation decreases by 6.0% and 4.4% respectively (Table 2.2).

ICP improvements. Without the pre-processing steps to ICP, we observe a drop in prediction accuracy of 0.9% in translation and 3.1% in rotation (Table 2.2).

Performance upper bound. We also evaluated how well the model-fitting part of our algorithm alone performs on the benchmark by using ground truth segmentation labels from the benchmark as the performance upper bound.

2.8.4 Common Failure Modes

Here we summarize the most common failure modes of our vision system, which are illustrated in Figure 2.9:

- The FCN segmentation for objects under heavy occlusion or clutter are likely to be incomplete resulting in poor pose estimation (Fig. 2.8.e), or undetected (Fig. 2.9.m and p). This happens with more frequency at back of the bin with poor illumination.

- Objects color textures are confused with each other. Figure 2.9.r shows a Dove bar (white box) on top of a yellow Scotch mail envelope, which combined have a similar appearance to the outlet plugs.

- Model fitting for cuboid objects often confuses corner alignments (marker boxes in Fig. 2.9.o). This inaccuracy, however, is still within the range of tolerance that the robot can tolerate thanks to sensor-guarded motions.

Filtering failure modes by confidence score. We compute a confidence score per object pose prediction that favors high precision for low recall. Specifically, the confidence score of a pose prediction equals the mean value of confidence scores over all points belonging to the segmentation of the object. We observe that erroneous

Table 2.1: 2D object segmentation evaluation (pixel-level object classification average % F-scores).

			environment		task	
network	all	cptn	off	whs	shelf	tote
color	**45.5**	**42.7**	**46.8**	**44.2**	**47.7**	**43.7**
color+depth	43.8	41.5	44.8	42.6	45.8	41.9
depth	37.1	35.0	38.6	35.5	39.8	34.9
10% data	20.4	18.8	19.5	21.3	21.7	20.3
1% data	8.0	9.2	7.2	8.8	15.8	6.5

	clutter (# of objects)			occlusion (%)			object-specific properties		
network	1 - 3	4 - 5	6 +	< 5	5 - 30	30 +	dfrm.	no depth	thin
color	**53.0**	**46.0**	**42.2**	**49.9**	**41.4**	**33.3**	**54.0**	**47.9**	**41.7**
color+depth	52.2	43.5	40.0	47.5	39.1	32.6	51.1	47.7	37.2
depth	45.5	37.0	33.5	40.8	33.2	26.3	44.1	42.3	29.1
10% data	36.0	21.6	18.0	21.2	25.5	0.0	41.9	17.2	33.3
1% data	17.3	7.5	6.0	7.7	8.3	7.8	10.1	5.7	3.5

poses (especially those due to partial occlusions) more often have low confidence scores. The robot system uses this value to target only predictions with high scores.

We evaluate the usefulness of the confidence scores by recalling the output of the perception system to only consider predictions with confidence scores larger than 10% and 70% respectively (see Table 2.2). These confidence percentages are important thresholds, because the full robot system, predictions with $< 10\%$ confidence (conf-10, at 78% recall) are ignored during planning, and prediction with $> 70\%$ confidence (conf-70, at 23% recall) trigger a pick attempt.

2.9 Discussion

Despite tremendous advances in computer vision, many state-of-the-art well-known approaches are often insufficient for relatively common scenarios. We describe here two observations that can lead to improvements in real systems:

Make the most out of every constraint. External constraints limit what systems can do. Indirectly they also limit the set of states in which the system can be, which

Table 2.2: Full vision system evaluation (average % correct rotation and translation predictions for object pose)

			environment		task	
algorithm	all	cptn	off	whs	shelf	tote
Full (rot.)	**49.8**	**62.9**	**52.5**	**47.1**	**50.4**	**49.3**
Full (trans.)	**66.1**	**71.0**	**66.3**	**65.9**	**63.4**	**68.1**
5v-10v (rot.)	44.0	48.6	50.9	35.9	50.9	38.9
5v-10v (trans.)	58.4	50.0	63.7	52.1	61.0	56.5
1v-2v (rot.)	38.9	60.0	41.1	36.5	45.0	35.3
1v-2v (trans.)	52.5	50.0	56.3	48.2	53.8	51.8
conf-70 (rot.)	58.3	77.3	65.0	49.0	64.2	53.2
conf-70 (trans.)	84.5	95.5	84.7	84.2	82.6	86.1
conf-10 (rot.)	55.0	70.8	57.0	52.7	54.9	55.0
conf-10 (trans.)	76.5	81.2	76.7	76.3	73.4	79.1
no denoise (rot.)	43.8	45.6	46.9	40.6	45.3	42.7
no denoise (trans.)	61.7	66.4	61.9	61.5	60.4	62.6
no ICP+ (rot.)	48.9	60.8	51.2	46.7	49.1	48.8
no ICP+ (trans.)	63.0	67.2	63.2	62.9	59.7	65.4
gt seg rot.	63.4	74.4	65.8	60.9	68.1	60.1
gt seg trans.	88.1	90.4	85.7	90.4	86.9	88.9

	clutter (# of objects)			occlusion (%)			object-specific properties		
algorithm	1 - 3	4 - 5	6 +	< 5	5 - 30	30 +	dfrm.	no depth	thin
Full (rot.)	**56.1**	**54.6**	**45.4**	**56.9**	**43.2**	**33.9**	**-**	**55.6**	**54.7**
Full (trans.)	**76.7**	**66.7**	**61.9**	**79.4**	**57.4**	**27.3**	**75.4**	**63.3**	**58.1**
5v-10v (rot.)	53.9	53.1	34.4	47.6	40.0	26.7	-	47.4	42.4
5v-10v (trans.)	69.4	63.0	50.3	66.2	49.8	21.3	54.7	67.3	35.4
1v-2v (rot.)	45.7	45.2	32.7	43.6	33.9	14.8	-	40.9	35.4
1v-2v (trans.)	60.4	56.5	46.7	58.2	47.8	16.7	52.9	55.9	33.3
conf-70 (rot.)	63.8	69.3	49.0	63.7	43.1	36.4	-	64.5	81.6
conf-70 (trans.)	86.2	84.1	83.2	87.1	77.1	72.7	83.1	77.4	85.7
conf-10 (rot.)	58.6	59.3	51.0	59.8	50.0	34.2	-	53.1	60.2
conf-10 (trans.)	80.8	74.4	75.4	84.0	70.0	40.0	78.1	72.0	70.1
no denoise (rot.)	52.0	46.7	39.5	51.1	37.3	28.1	-	48.8	54.1
no denoise (trans.)	74.8	62.7	56.4	76.5	52.9	19.9	75.0	62.3	53.8
no ICP+ (rot.)	55.4	54.1	44.4	55.8	41.9	36.2	-	53.6	52.5
no ICP+ (trans.)	72.1	64.4	59.1	75.2	57.0	24.6	67.3	62.8	53.2
gt seg rot.	69.1	68.8	59.1	67.6	60.0	53.5	-	58.0	74.1
gt seg trans.	88.3	88.0	88.0	90.7	90.3	71.4	90.5	71.5	79.8

can lead to opportunities for simplifications and robustness in the perception system. In the picking task, each team received a list of items, their bin assignments, and a model of the shelf. All teams used the bin assignments to rule out objects from consideration and the model of the shelf to calibrate their robots. These optimizations are straightforward and useful. However, further investigation yields more opportu-

nity. By using these same constraints, we constructed a self-supervising mechanism to train a deep neural network with significantly more data. As our evaluations show, the volume of training data is strongly correlated with performance.

Designing robotic and vision systems hand-in-hand. Vision algorithms are too often designed in isolation. However, vision is one component of a larger robotic system with needs and opportunities. Typical computer vision algorithms operate on single images for segmentation and recognition. Robotic arms free us from that constraint, allowing us to precisely fuse multiple views and improve performance in cluttered environments. Computer vision systems also tend to have fixed outputs (e.g., bounding boxes or 2D segmentation maps), but robotic systems with multiple manipulation strategies can benefit from variety in output. For example, suction cups and grippers might have different perceptual requirements. While the former might work more robustly with a segmented point cloud, the latter often requires knowledge of the object pose and geometry.

2.10 Summary

In this chapter, we develop a vision algorithm that incorporates deep learning into the classic manipulation pipeline by leveraging convolutional networks for 6D object pose estimation. To address the challenges posed by the warehouse setting, our framework leverages multi-view RGB-D data and data-driven, self-supervised deep learning to reliably estimate the 6D poses of objects under a variety of scenarios. We also provide a well-labeled benchmark dataset of APC 2016 containing over 7,000 images from 477 scenes.

Chapter 3

Learning Visual Affordances

Classic algorithms for robotic picking typically require recognition and pose estimation prior to model-based grasp planning. In the previous chapter, we showed that it is possible to improve such solutions using deep learning. However, these solutions are still limited because they are susceptible to propagating errors (through independent perception, planning, and control modules) and cannot handle novel objects, since they require prior information about the objects in the form of class categories or 3D object CAD models.

To address these limitations, this chapter presents a robotic pick-and-place system that is capable of grasping and recognizing both known and novel objects in cluttered environments, without needing any task-specific training data for novel objects. To achieve this, it first uses a visual affordance framework to map from visual observations to actions: inferring dense pixel-wise probability maps of the affordances for four different grasping primitive actions. It then executes the action with the highest affordance and recognizes picked objects with a cross-domain image classification framework that matches observed images to product images. Since product images are readily available for a wide range of objects (e.g., from the web), the system works out-of-the-box for novel objects without requiring any additional data collection or

re-training. Exhaustive experimental results demonstrate that our visual affordance grasping achieves high success rates for a wide variety of objects in clutter, and our recognition algorithm achieves high accuracy for both known and novel grasped objects. The approach was part of the MIT-Princeton Team system that took 1st place in the stowing task at the 2017 Amazon Robotics Challenge.

3.1 Learning Visual Affordances for Robotic Pick-and-place of Novel Objects

A human's remarkable ability to grasp and recognize unfamiliar objects with little prior knowledge of them is a constant inspiration for robotics research. This ability to grasp the unknown is central to many applications: from picking packages in a logistic center to bin-picking in a manufacturing plant; from unloading groceries at home to clearing debris after a disaster. The main goal of this work is to demonstrate that it is possible – and practical – for a robotic system to pick and recognize novel objects with very limited prior information about them (e.g. with only a few representative images scraped from the web).

Despite the interest of the research community, and despite its practical value, robust manipulation and recognition of novel objects in cluttered environments still remains a largely unsolved problem. Classical solutions for robotic picking require recognition and pose estimation prior to model-based grasp planning, or require object segmentation to associate grasp detections with object identities. These solutions tend to fall short when dealing with novel objects in cluttered environments, since they rely on 3D object models that are not available and/or large amounts of training data to achieve robust performance. Although there has been inspiring recent work on detecting grasps directly from RGB-D pointclouds as well as learning-based recognition systems to handle the constraints of novel objects and limited data, these

methods have yet to be proven in the constraints and accuracy required by a real task with heavy clutter, severe occlusions, and object variability.

In this chapter, we propose a system that picks and recognizes objects in cluttered environments. We have designed the system specifically to handle a wide range of objects novel to the system without gathering any task-specific training data from them.

To make this possible, our system consists of two components. The first is a multi-affordance grasping framework which uses fully convolutional networks (FCNs) to take in visual observations of the scene and output dense predictions (arranged with the same size and resolution as the input data) measuring the affordance (or probability of picking success) for four different grasping primitive actions over a pixel-wise sampling of end-effector orientations and locations. The primitive action with the highest inferred affordance value determines the picking action executed by the robot. This picking framework operates without a priori object segmentation and classification and hence is agnostic to object identity.

The second component of the system is a cross-domain image matching framework for recognizing grasped objects by matching them to product images using a two-stream convolutional network (ConvNet) architecture. This framework adapts to novel objects without additional re-training. Both components work hand-in-hand to achieve robust picking performance of novel objects in heavy clutter.

We provide exhaustive experiments, ablation, and comparison to evaluate both components. We demonstrate that our affordance-based algorithm for grasp planning achieves high success rates for a wide variety of objects in clutter, and the recognition algorithm achieves high accuracy for known and novel grasped objects. These algorithms were developed as part of the MIT-Princeton Team system that took 1st place in the stowing task of the Amazon Robotics Challenge (ARC), being the only system to have successfully stowed all known and novel objects from an unstructured

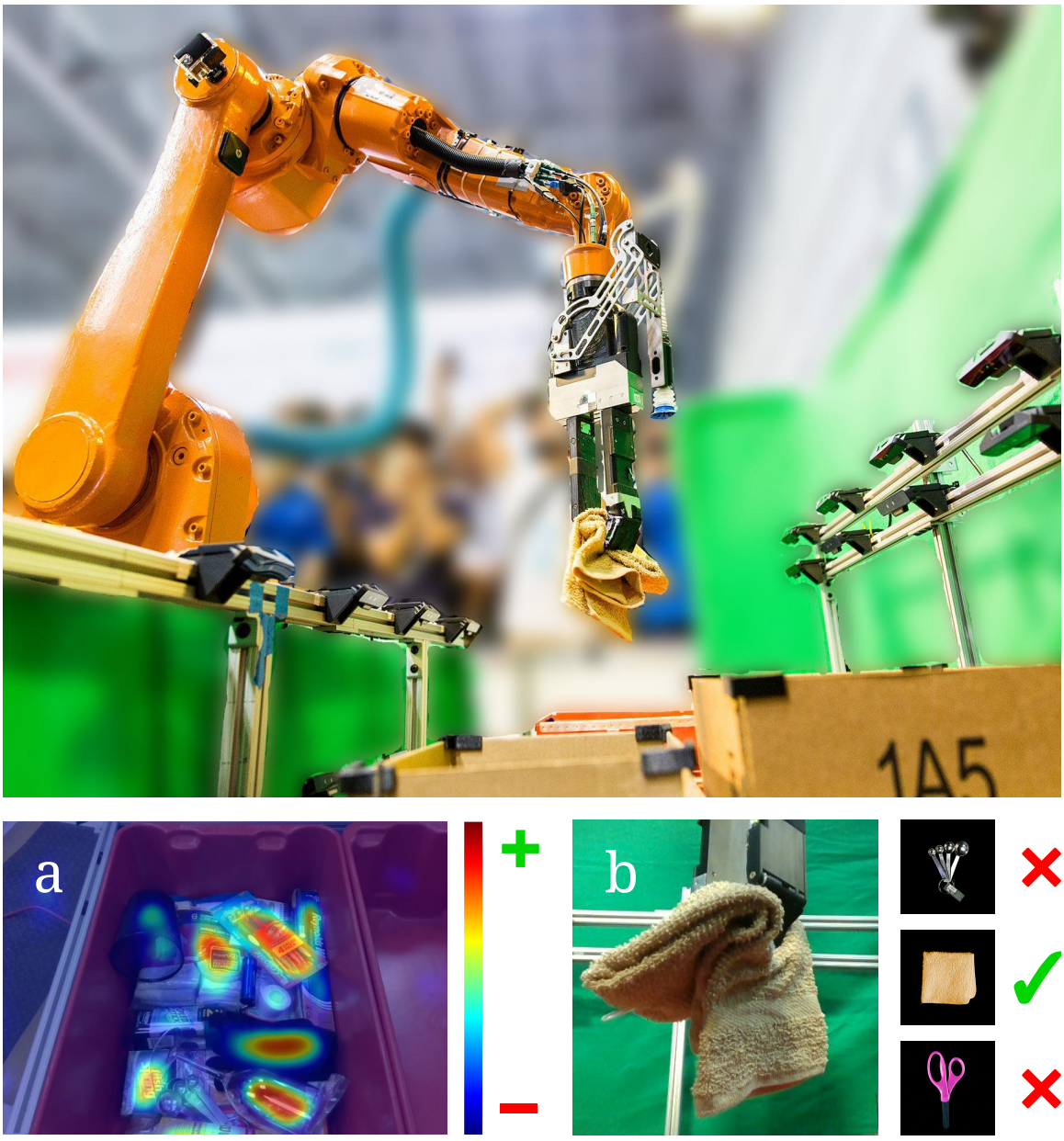


Figure 3.1: **Our picking system** computing pixel-wise affordances for grasping over visual observations of bins full of objects, (a) grasping a towel and holding it up away from clutter, and recognizing it by matching observed images of the towel (b) to an available representative product image. The key contribution is that the entire system works out-of-the-box for novel objects (unseen in training) without the need for any additional data collection or re-training.

tote into a storage system within the allotted time frame. Figure 3.1 shows our robot in action during the competition.

In summary, our main contributions are:

- An affordance-based object-agnostic perception framework to plan grasps using four primitive grasping actions for fast and robust picking. This utilizes fully convolutional networks for inferring dense pixel-wise affordances of each primitive (Section 3.4).

- A perception framework for recognizing both known and novel objects using only product images without extra data collection or re-training. This utilizes a two stream convolutional network to match images or picked objects to product images (Section 3.5).

- A system combining these two frameworks for picking novel objects in heavy clutter.

All code, datasets, and pre-trained models are available online at http://arc.cs.princeton.edu. We also provide a video summarizing our approach at https://youtu.be/6fG7zwGfIkI.

3.2 Related Work

In this section, we review works related to robotic picking systems. Works specific to grasping (Section 3.4) and recognition (Section 3.5) are in their respective sections.

3.2.1 Recognition followed by Model-based Grasping

A large number of autonomous pick-and-place solutions follow a standard two-step approach: object recognition and pose estimation followed by model-based grasp planning. For example, [55] designed object segmentation methods over handcrafted image features to compute suction proposals for picking objects with a vacuum.

More recent data-driven approaches ([40, 123, 104, 117]) use ConvNets to provide bounding box proposals or segmentations, followed by geometric registration to estimate object poses, which ultimately guide handcrafted picking heuristics ([13, 75]). [83] improve many aspects of this pipeline by leveraging robot mobility, while [63] adds a pose correction stage when the object is in the gripper. These works typically require 3D models of the objects during test time, and/or training data with the physical objects themselves. This is practical for tightly constrained pick-and-place scenarios, but is not easily scalable to applications that consistently encounter novel objects, for which only limited data (i.e. product images from the web) is available.

3.2.2 Recognition in parallel with Object-Agnostic Grasping

It is also possible to exploit local features of objects without object identity to efficiently detect grasps ([77, 59, 93, 113, 86, 87, 66, 37, 61]). Since these methods are agnostic to object identity, they better adapt to novel objects and experience higher picking success rates in part by eliminating error propagation from a prior recognition step. [72] apply this idea in a full picking system by using a ConvNet to compute grasp proposals, while in parallel inferring semantic segmentations for a fixed set of known objects. Although these pick-and-place systems use object-agnostic grasping methods, they still require some form of in-place object recognition in order to associate grasp proposals with object identities, which is particularly challenging when dealing with novel objects in clutter.

3.2.3 Active Perception

The act of exploiting control strategies for acquiring data to improve perception ([9, 16]) can facilitate the recognition of novel objects in clutter. For example, [52] describe a robotic system that actively rearranges objects in the scene (by pushing) in order to improve recognition accuracy. Other works [118, 51] explore next-best-view

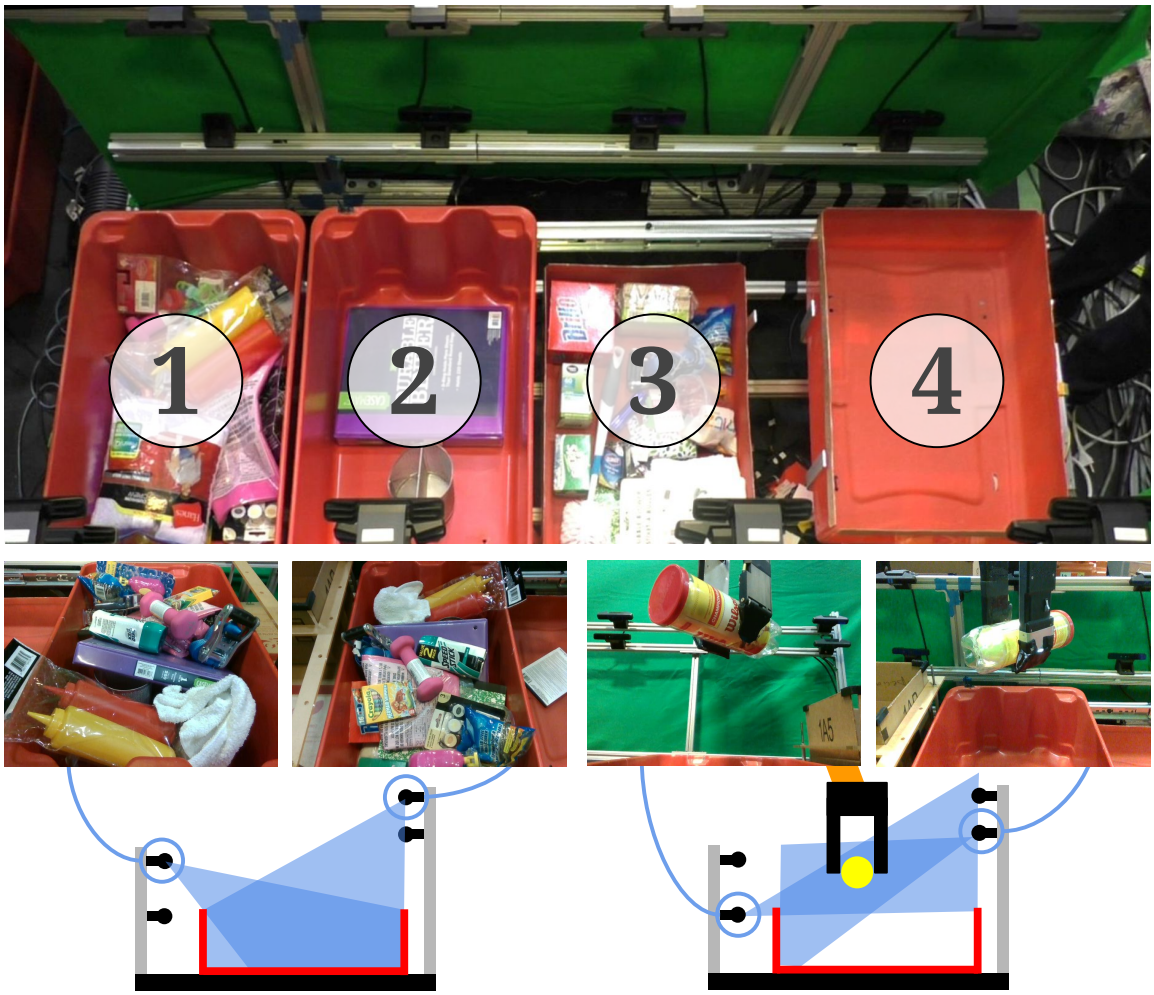


Figure 3.2: **The bin and camera setup.** Our system consists of 4 units (top), where each unit has a bin with 4 stationary cameras: two overlooking the bin (bottom-left) are used for inferring grasp affordances while the other two (bottom-right) are used for recognizing grasped objects.

based approaches to improve recognition, segmentation and pose estimation results. Inspired by these works, our system uses a form of active perception by using a grasp-first-then-recognize paradigm where we leverage object-agnostic grasping to isolate each object from clutter in order to significantly improve recognition accuracy for novel objects.

3.3 System Overview

We present a robotic pick-and-place system that grasps and recognizes both known and novel objects in cluttered environments. We will refer by "known" objects to those that are provided to the system at training time, both as physical objects and as representative product images (images of objects available on the web); while "novel" objects are provided only at test time in the form of representative product images.

The pick-and-place task presents us with two main perception challenges: 1) find accessible grasps of objects in clutter; and 2) match the identity of grasped objects to product images. Our approach and contributions to these two challenges are described in detail in Section 3.4 and Section 3.5 respectively. For context, in this section we briefly describe the system that will use those two capabilities.

Overall approach. The system follows a *grasp-first-then-recognize* work-flow. For each pick-and-place operation, it first uses FCNs to infer the pixel-wise affordances of four different grasping primitive actions: from suction to parallel-jaw grasps (Section 3.4). It then selects the grasping primitive action with the highest affordance, picks up one object, isolates it from the clutter, holds it up in front of cameras, recognizes its category, and places it in the appropriate bin. Although the object recognition algorithm is trained only on known objects, it is able to recognize novel objects

through a learned cross-domain image matching embedding between observed images of held objects and product images (Section 3.5).

Advantages. This system design has several advantages. First, the affordance-based grasping algorithm is model-free and agnostic to object identities and generalizes to novel objects without re-training. Second, the category recognition algorithm works without task-specific data collection or re-training for novel objects, which makes it scalable for applications in warehouse automation and service robots where the range of observed object categories is large and dynamic. Third, our grasping framework supports multiple grasping modes with a multi-functional gripper and thus handles a wide variety of objects. Finally, the entire processing pipeline requires only a few forward passes through deep networks and thus executes quickly (run-times reported in Table 3.2).

System setup. Our system features a 6DOF ABB IRB 1600id robot arm next to four picking work-cells. The robot arm's end-effector is a multi-functional gripper with two fingers for parallel-jaw grasps and a retractable suction cup (Fig. 3.3). This gripper was designed to function in cluttered environments: finger and suction cup length are specifically chosen such that the bulk of the gripper body does not need to enter the cluttered space.

Each work-cell has a storage bin and four statically-mounted RealSense SR300 RGB-D cameras (Fig.3.2): two cameras overlooking the storage bins are used to infer grasp affordances, while the other two pointing upwards towards the robot gripper are used to recognize objects in the gripper. For the two cameras used to infer grasp affordances, we find that placing them at opposite viewpoints of the storage bins provides good visual coverage of the objects in the bin. Adding a third camera did not significantly improve visual coverage. For the other two cameras used for object recognition, having them at opposite viewpoints enables us to immediately reconstruct a near-complete 3D point cloud of the object while it is being held in

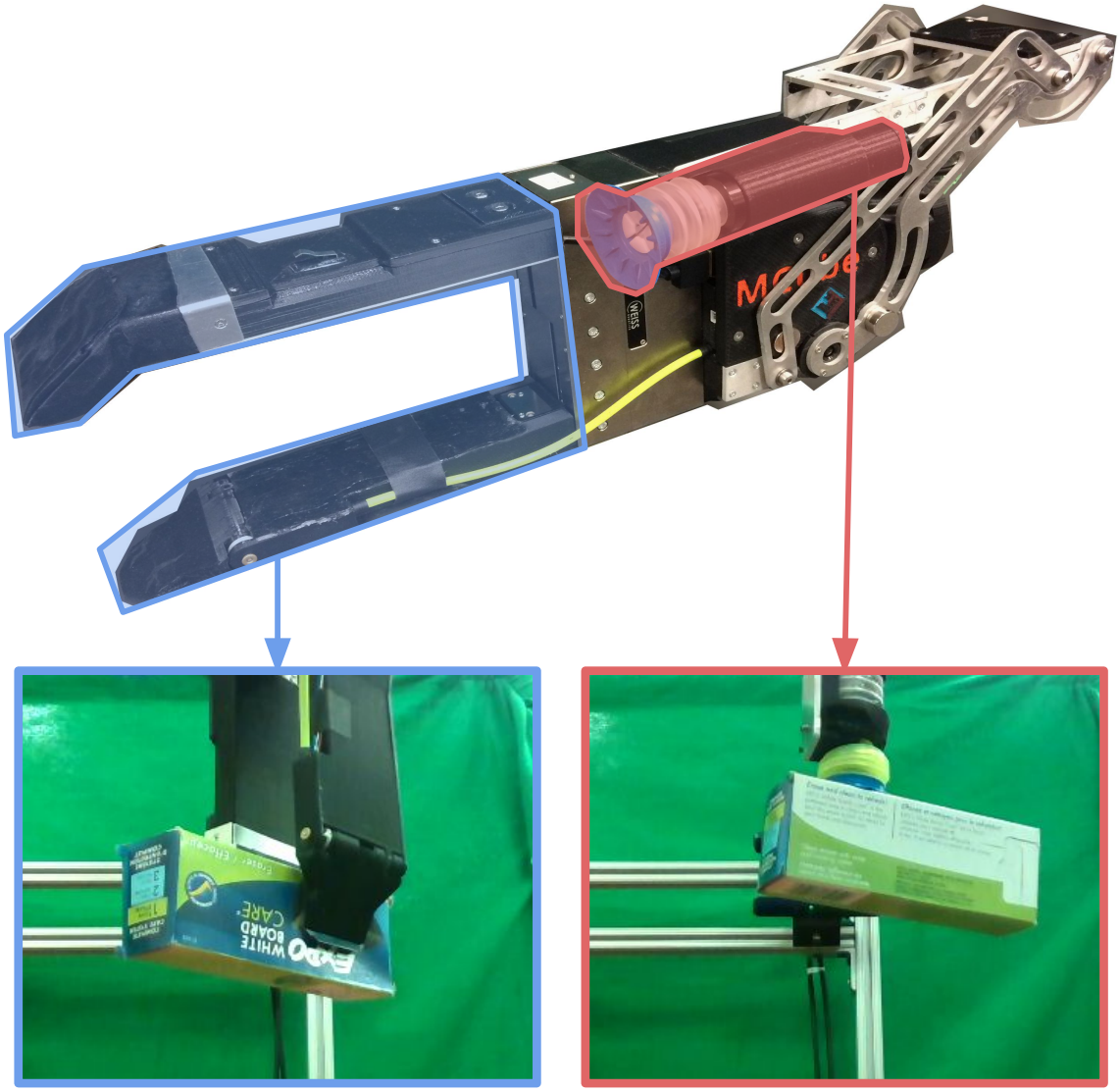


Figure 3.3: **Multi-functional gripper** with a retractable mechanism that enables quick and automatic switching between suction (pink) and grasping (blue).

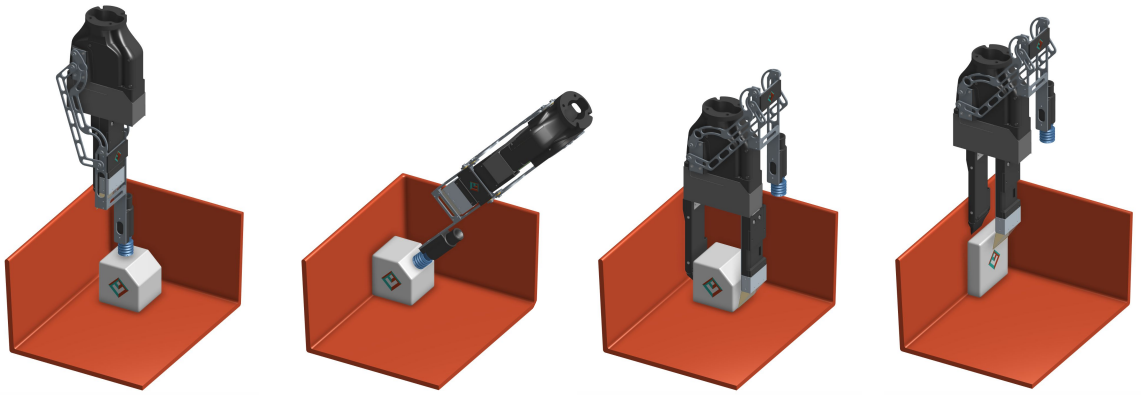


Figure 3.4: **Multiple motion primitives** for suction and grasping to ensure successful picking for a wide variety of objects in any orientation.

the gripper. These 3D point clouds are useful for planning object placements in the storage system.

Although our experiments were performed with this setup, the system was designed to be flexible for picking and placing between any number of reachable workcells and camera locations. Furthermore, all manipulation and recognition algorithms in this chapter were designed to be easily adapted to other system setups.

3.4 Challenge I: Planning Grasps with Multi-Affordance Grasping

The goal of the first step in our system is to robustly grasp objects from a cluttered scene without relying on their object identities or poses. To this end, we define a set of four grasping primitive actions that are complementary to each other in terms of utility across different object types and scenarios – empirically broadening the variety of objects and orientations that can be picked with at least one primitive. Given RGB-D images of the cluttered scene at test time, we infer the dense pixel-wise affordances for all four primitives. A task planner then selects and executes the primitive with the highest affordance.

3.4.1 Grasping Primitives

We define four grasping primitives to achieve robust picking for typical household objects. Figure 3.4 shows example motions for each primitive. Each of them is implemented as a set of guarded moves with collision avoidance using force sensors below the work-cells. They also have quick success or failure feedback mechanisms using either flow sensing for suction or force sensing for grasping. Robot arm motion planning is automatically executed within each primitive with stable inverse kinematic-based controllers [22]. These primitives are as follows:

Suction down grasps objects with a vacuum gripper vertically. This primitive is particularly robust for objects with large and flat suctionable surfaces (e.g. boxes, books, wrapped objects), and performs well in heavy clutter.

Suction side grasps objects from the side by approaching with a vacuum gripper tilted at a fixed angle. This primitive is robust to thin and flat objects resting against walls, which may not have suctionable surfaces from the top.

Grasp down grasps objects vertically using the two-finger parallel-jaw gripper. This primitive is complementary to the suction primitives in that it is able to pick up objects with smaller, irregular surfaces (e.g. small tools, deformable objects), or made of semi-porous materials that prevent a good suction seal (e.g. cloth).

Flush grasp retrieves unsuctionable objects that are flushed against a wall. The primitive is similar to grasp down, but with the additional behavior of using a flexible spatula to slide one finger in between the target object and the wall.

3.4.2 Learning Affordances with Fully Convolutional Networks

Given the set of pre-defined grasping primitives and RGB-D images of the scene, we train FCNs ([65]) to infer the affordances for each primitive across a dense pixel-

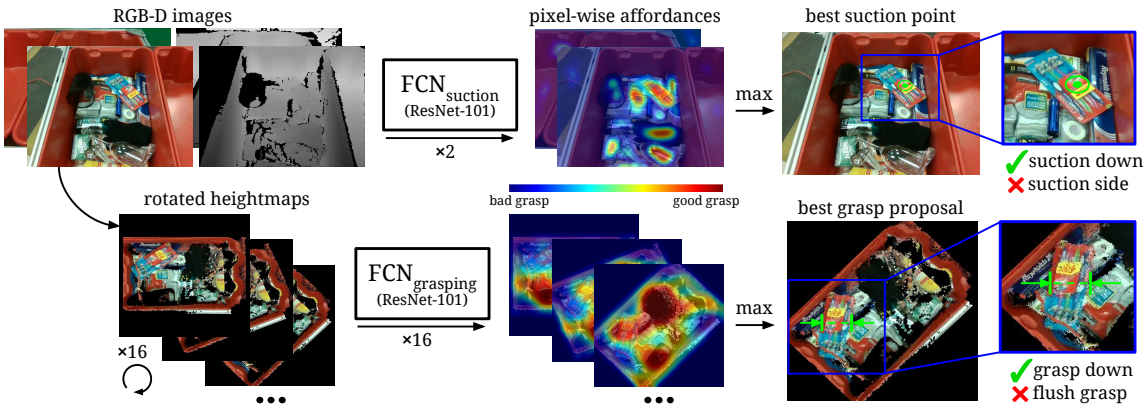


Figure 3.5: **Learning pixel-wise affordances for suction and grasping.** Given multi-view RGB-D images, we infer pixel-wise suction affordances for each image with an FCN (top row). The inferred affordance value at each pixel describes the utility of suction at that pixel's projected 3D location. We aggregate the inferred affordances onto a 3D point cloud, where each point corresponds to a suction proposal (down or side based on surface normals). In parallel, we merge RGB-D images into an orthographic RGB-D heightmap of the scene, rotate it by 16 different angles, and feed them each through another FCN (bottom row) to estimate the pixel-wise affordances of horizontal grasps for each heightmap. This effectively produces affordance maps for 16 different top-down grasping angles, from which we generate grasp down and flush grasp proposals. The suction or grasp proposal with the highest affordance value is executed.

wise sampling of end-effector orientations and locations (*i.e.* each pixel correlates to a different position on which to execute the primitive). Our approach relies on the assumption that graspable regions can be deduced from local geometry and visual appearance. This is inspired by recent data-driven methods for grasp planning [77, 102, 59, 93, 86, 87, 66, 37, 61], which do not rely on object identities or state estimation.

Inferring Suction Affordances. We define suction points as 3D positions where the vacuum gripper's suction cup should come in contact with the object's surface in order to successfully grasp it. Good suction points should be located on suctionable (*e.g.* nonporous) surfaces, and nearby the target object's center of mass to avoid an unstable suction seal (e.g. particularly for heavy objects). Each suction proposal is defined as a suction point, its local surface normal (computed from the projected 3D

point cloud), and its affordance value. Each pixel of an RGB-D image (with a valid depth value) maps surjectively to a suction point.

We train a fully convolutional residual network (ResNet-101 [39]), that takes a 640×480 RGB-D image as input, and outputs a densely labeled pixel-wise map (with the same image size and resolution as the input) of affordance values between 0 and 1. Values closer to one imply a more preferable suction location. Visualizations of these densely labeled affordance maps are shown as heat maps in the first row of Fig. 3.5. Our network architecture is multi-modal, where the color data (RGB) is fed into one ResNet-101 tower, and 3-channel depth (DDD, cloned across channels, normalized by subtracting mean and dividing by standard deviation) is fed into another ResNet-101 tower. The depth is cloned across channels so that we can use the ResNet weights pre-trained on 3-channel (RGB) color images from ImageNet [21] to process depth information. Features from the ends of both towers are concatenated across channels, followed by 3 additional spatial convolution layers to merge the features; then spatially bilinearly upsampled and softmaxed to output a binary probability map representing the inferred affordances.

Our FCN is trained over a manually annotated dataset of RGB-D images of cluttered scenes with diverse objects, where pixels are densely labeled either positive, negative, or neither. Pixel regions labeled as neither are trained with 0 loss back-propagation. We train our FCNs by stochastic gradient descent with momentum, using fixed learning rates of 10^{-3} and momentum of 0.99. Our models are trained in Torch/Lua with an NVIDIA Titan X on an Intel Core i7-3770K clocked at 3.5 GHz. Training takes about 10 hours.

During testing, we feed each captured RGB-D image through our trained network to generate dense suction affordances for each view of the scene. As a post-processing step, we use calibrated camera intrinsics and poses to project the RGB-D data and aggregate the affordances onto a combined 3D point cloud. We then compute surface

normals for each 3D point (using a local region around it), which are used to classify which suction primitive (down or side) to use for the point.

To handle objects that lack depth information, e.g., finely meshed objects or transparent objects, we use a simple hole filling algorithm [107] on the depth images, and project inferred affordance values onto the hallucinated depth. We filter out suction points from the background by performing background subtraction [123] between the captured RGB-D image of the scene with objects and an RGB-D image of the scene without objects (captured automatically before any objects are placed into the picking work-cells).

Inferring Grasp Affordances. Grasp proposals are represented by 1) a 3D position which defines the middle point between the two fingers during top-down parallel-jaw grasping, 2) an angle which defines the orientation of the gripper around the vertical axis along the direction of gravity, 3) the width between the gripper fingers during the grasp, and 4) its affordance value.

Two RGB-D views of the scene are aggregated into a registered 3D point cloud, which is then orthographically back-projected upwards in the gravity direction to obtain a "heightmap" image representation of the scene with both color (RGB) and height-from-bottom (D) channels. Each pixel of the heightmap represents a $2x2$mm vertical column of 3D space in the scene. Each pixel also correlates bijectively to a grasp proposal whose 3D position is naturally computed from the spatial 2D position of the pixel relative to the heightmap image and the height value at that pixel. The gripper orientation of the grasp proposal is always kept horizontal with respect to the frame of the heightmap.

Analogous to our deep network inferring suction affordances, we feed this RGB-D heightmap as input to a fully convolutional ResNet-101 [39], which densely infers affordance values (between 0 and 1) for each pixel – thereby for all top-down parallel-jaw grasping primitives executed with a horizontally orientated gripper across all 3D

locations in heightmap of the scene sampled at pixel resolution. Visualizations of these densely labeled affordance maps are shown as heat maps in the second row of Fig. 3.5. By rotating the heightmap of the scene with n different angles prior to feeding as input to the FCN, we can account for n different gripper orientations around the vertical axis. For our system $n = 16$; hence we compute affordances for all top-down parallel-jaw grasping primitives with 16 forward passes of our FCN to generate 16 output affordance maps.

We train our FCN over a manually annotated dataset of RGB-D heightmaps, where each positive and negative grasp label is represented by a pixel on the heightmap as well as an angle indicating the preferred gripper orientation. We trained this FCN with the same optimization parameters as that of the FCN used for inferring suction affordances.

During post-processing, the width between the gripper fingers for each grasp proposal is determined by using the local geometry of the 3D point cloud. We also use the location of each proposal relative to the bin to classify which grasping primitive (down or flush) should be used: flush grasp is executed for pixels located near the sides of the bins; grasp down is executed for all other pixels. To handle objects without depth, we triangulate no-depth regions in the heightmap using both RGB-D camera views of the scene, and fill in these regions with synthetic height values of 3cm prior to feeding into the FCN. We filter out inferred grasp proposals in the background by using background subtraction with the RGB-D heightmap of an empty work-cell.

3.4.3 Other Architectures for Parallel-Jaw Grasping

A significant challenge during the development of our system was designing a deep network architecture for inferring dense affordances for parallel-jaw grasping that 1) supports various gripper orientations and 2) could converge during training with less than 2000 manually labeled images. It took several iterations of network architecture

designs before discovering the one that worked (described above). Here, we briefly review some deprecated architectures and their primary drawbacks:

Parallel trunks and branches (n copies). This design consists of n separate FCNs, each responsible for inferring the output affordances for one of n grasping angles. Each FCN shares the same architecture: a multi-modal trunk (with color (RGB) and depth (DDD) data fed into two ResNet-101 towers pre-trained on ImageNet, where features at the ends of both towers are concatenated across channels), followed by 3 additional spatial convolution layers to merge the features; then spatially bilinearly upsampled and softmaxed to output an affordance map. This design is similar to our final network design, but with two key differences: 1) there are multiple FCNs, one for each grasping angle, and 2) the input data is not rotated prior to feeding as input to the FCNs. This design is sample inefficient, since each network during training is optimized to learn a different set of visual features to support a specific grasping angle, thus requiring a substantial amount of training samples with that specific grasping angle to converge. Our small manually annotated dataset is characterized by an unequal distribution of training samples across different grasping angles, some of which have as little as less than 100 training samples. Hence, only a few of the FCNs (for grasping angles of which have more than 1,000 training samples) are able to converge during training. Furthermore, attaining the capacity to pre-load all n FCNs into GPU memory for test time requires multiple GPUs.

One trunk, split to n parallel branches. This design consists of a single FCN architecture, which contains a multi-modal ResNet-101 trunk followed by a split into n parallel, individual branches, one for each grasping angle. Each branch contains 3 spatial convolution layers followed by spatial bilinearly upsampling and softmax to output affordance maps. While more lightweight in terms of GPU memory consumption (*i.e.*, the trunk is shared and only the 3-layer branches have multiple copies), this FCN still runs into similar training convergence issues as the previous

architecture, where each branch during training is optimized to learn a different set of visual features to support a specific grasping angle. The uneven distribution of limited training samples in our dataset made it so that only a few branches are able to converge during training.

One trunk, rotate, one branch. This design consists of a single FCN architecture, which contains a multi-modal ResNet-101 trunk, followed by a spatial transform layer [50] to rotate the intermediate feature map from the trunk with respect to an input grasp angle (such that the gripper orientation is aligned horizontally to the feature map), followed by a branch with 3 spatial convolution layers, spatially bilinearly upsampled, and softmaxed to output a single affordance map for the input grasp angle. This design is even more lightweight than the previous architecture in terms of GPU memory consumption, performs well with grasping angles for which there is a sufficient amount of training samples, but continues to performs poorly for grasping angles with very few training samples (less than 100).

One trunk and branch (rotate n times). This is the final network architecture design as proposed above, which differs from the previous design in that the rotation occurs directly on the input image representation prior to feeding through the FCN (rather than in the middle of the architecture). This enables the entire network to share visual features across different grasping orientations, enabling it to generalize for grasping angles of which there are very few training samples.

3.4.4 Task Planner

Our task planner selects and executes the suction or grasp proposal with the highest affordance value. Prior to this, affordance values are scaled by a factor γ_ψ that is specific to the proposals' primitive action types $\psi \in \{\text{sd}, \text{ss}, \text{gd}, \text{fg}\}$: suction down (sd), suction side (ss), grasp down (gd), or flush grasp (fg). The value of γ_ψ is

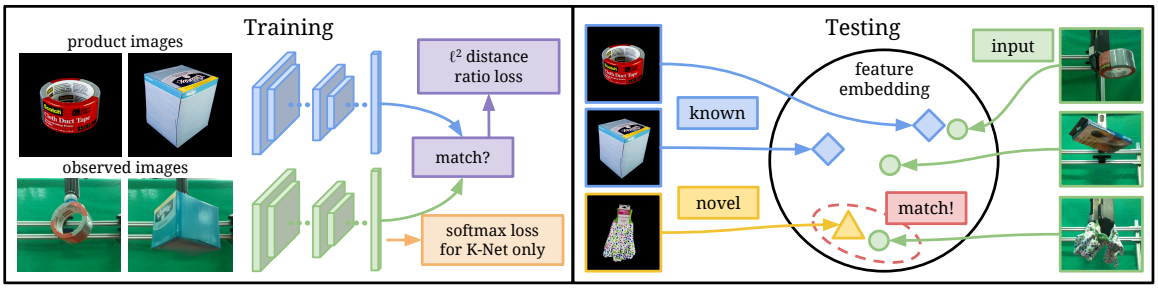


Figure 3.6: **Recognition framework for novel objects.** We train a two-stream convolutional neural network where one stream computes 2048-dimensional feature vectors for product images while the other stream computes 2048-dimensional feature vectors for observed images, and optimize both streams so that features are more similar for images of the same object and dissimilar otherwise. During testing, product images of both known and novel objects are mapped onto a common feature space. We recognize observed images by mapping them to the same feature space and finding the nearest neighbor match.

determined by several task-specific heuristics that induce more efficient picking under competition settings at the ARC. Here we briefly describe these heuristics:

Suction first, grasp later. We empirically find suction to be more reliable than parallel-jaw grasping when picking in scenarios with heavy clutter (10+ objects). Among several factors, the key reason is that suction is significantly less intrusive than grasping. Hence, to reflect a greedy picking strategy that initially favors suction over grasping, $\gamma_{\text{gd}} = 0.5$ and $\gamma_{\text{fg}} = 0.5$ for the first 3 minutes of either ARC task (stowing or picking).

Avoid repeating unsuccessful attempts. It is possible for the system to get stuck repeatedly executing the same (or similar) suction or grasp proposal as no change is made to the scene (and hence affordance estimates remain the same). Therefore, after each unsuccessful suction or parallel-jaw grasping attempt, the affordances of the proposals (for the same primitive action) nearby within radius 2cm of the unsuccessful attempt are set to 0.

Encouraging exploration upon repeat failures. The planner re-weights grasping primitive actions γ_ψ depending on how often they fail. For primitives that have been unsuccessful for two times in the last 3 minutes, $\gamma_\psi = 0.5$; if unsuccessful for

more than three times, $\gamma_\psi = 0.25$. This not only helps the system avoid repeating unsuccessful actions, but also prevents it from excessively relying on any one primitive that doesn't work as expected (*e.g.* in the case of an unexpected hardware failure preventing suction air flow).

Leveraging dense affordances for speed picking. Our FCNs densely infer affordances for all visible surfaces in the scene, which enables the robot to attempt multiple different suction or grasping proposals (at least 3cm apart from each other) in quick succession until at least one of them is successful (given by immediate feedback from flow sensors or gripper finger width). This improves picking efficiency.

3.5 Challenge II: Recognizing Novel Objects with Cross-Domain Image Matching

After successfully grasping an object and isolating it from clutter, the goal of the second step in our system is to recognize the identity of the grasped object.

Since we encounter both known and novel objects, and we have only product images for the novel objects, we address this recognition problem by retrieving the best match among a set of product images. Of course, observed images and product images can be captured in significantly different environments in terms of lighting, object pose, background color, post-process editing, etc. Therefore, we require an algorithm that is able to find the semantic correspondences between images from these two different domains. While this is a task that appears repeatedly in a variety of research topics (*e.g.* domain adaptation, one-shot learning, meta-learning, visual search, etc.), in this chapter we refer to it as a *cross-domain image matching* problem ([99, 106, 11]).

3.5.1 Metric Learning for Cross-Domain Image Matching

To perform the cross-domain image matching between observed images and product images, we learn a metric function that takes in an observed image and a candidate product image and outputs a distance value that models how likely the images are of the same object. The goal of the metric function is to map both the observed image and product image onto a meaningful feature embedding space so that smaller ℓ_2 feature distances indicate higher similarities. The product image with the smallest metric distance to the observed image is the final matching result.

We model this metric function with a two-stream convolutional neural network (ConvNet) architecture where one stream computes features for the observed images, and a different stream computes features for the product images. We train the network by feeding it a balanced 1:1 ratio of matching and non-matching image pairs (one observed image and one product image) from the set of known objects, and back-propagate gradients from the distance ratio loss (Triplet loss [42]). This effectively optimizes the network in a way that minimizes the ℓ_2 distances between features of matching pairs while pulling apart the ℓ_2 distances between features of non-matching pairs. By training over enough examples of these image pairs across known objects, the network learns a feature embedding that encapsulates object shape, color, and other visual discriminative properties, which can generalize and be used to match observed images of novel objects to their respective product images (Fig. 3.6).

Avoiding metric collapse by guided feature embeddings. One issue commonly encountered in metric learning occurs when the number of training object categories is small – the network can easily overfit its feature space to capture only the small set of training categories, making generalization to novel object categories difficult. We refer to this problem as metric collapse. To avoid this issue, we use a model pre-trained on ImageNet ([21]) for the product image stream and train only the stream that computes features for observed images. ImageNet contains a large collection

of images from many categories, and models pre-trained on it have been shown to produce relatively comprehensive and homogenous feature embeddings for transfer tasks ([47]) – i.e. providing discriminating features for images of a wide range of objects. Our training procedure trains the observed image stream to produce features similar to the ImageNet features of product images – i.e., it learns a mapping from observed images to ImageNet features. Those features are then suitable for direct comparison to features of product images, even for novel objects not encountered during training.

Using multiple product images. For many applications, there can be multiple product images per object. However, with multiple product images, supervision of the two-stream network can become confusing - on which pair of matching observed and product images should the backpropagated gradients be based? For example, matching an observed image of the front face of the object against a product image of the back face of the object can easily confuse network gradients. To solve this problem during training, we add a module called "multi-anchor switch" in the network. Given an observed image, this module automatically chooses which "anchor" product image to compare against (*i.e.*, to compute loss and gradients for) based on ℓ_2 distance between deep features. We find that allowing the network to select nearest neighbor "anchor" product images during training provides a significant boost in performance in comparison to alternative methods like random sampling.

3.5.2 Two Stage Framework for a Mixture of Known and Novel Objects

In settings where both types of objects are present, we find that training two different network models to handle known and novel objects separately can yield higher overall matching accuracies. One is trained to be good at "over-fitting" to the known objects

(K-net) and the other is trained to be better at "generalizing" to novel objects (N-net).

Yet, how do we know which network to use for a given image? To address this issue, we execute our recognition pipeline in two stages: a "recollection" stage that determines whether the observed object is known or novel, and a "hypothesis" stage that uses the appropriate network model based on the first stage's output to perform image matching.

First, the recollection stage infers whether the input observed image from test time is that of a known object that has appeared during training. Intuitively, an observed image is of a novel object if and only if its deep features cannot match to that of any images of known objects. We explicitly model this conditional by thresholding on the nearest neighbor distance to product image features of known objects. In other words, if the ℓ_2 distance between the K-net features of an observed image and the nearest neighbor product image of a known object is greater than some threshold k, then the observed images is a novel object. Note that the novel object network can also identify known objects, but with lower performance.

In the hypothesis stage, we perform object recognition based on one of two network models: K-net for known objects and N-net for novel objects. The K-net and N-net share the same network architecture. However, during training the K-net has an "auxiliary classification" loss for the known objects. This loss is implemented by feeding in the K-net features into 3 fully connected layers, followed by an n-way softmax loss where n is the number of known object classes. These layers are present in K-net during training then removed during testing. Training with this classification loss increases the accuracy of known objects at test time to near perfect performance, and also boosts up the accuracy of the recollection stage, but fails to maintain the accuracy of novel objects. On the other hand, without the restriction of

the classification loss, N-net has a lower accuracy for known objects, but maintains a better accuracy for novel objects.

By adding the recollection stage, we can exploit both the high accuracy of known objects with K-net and good accuracy of novel objects with N-net, though incurring a cost in accuracy from erroneous known vs novel classification. We find that this two stage system overall provides higher total matching accuracy for recognizing both known and novel objects (mixed) than all other baselines (Table 3.3).

3.6 Experiments

In this section, we evaluate our affordance-based grasping framework, our recognition algorithm over both known and novel objects, as well as our full system in the context of the Amazon Robotics Challenge 2017.

3.6.1 Evaluating Multi-affordance Grasping

Datasets. To generate datasets for learning affordance-based grasping, we designed a simple labeling interface that prompts users to manually annotate good and bad suction and grasp proposals over RGB-D images collected from the real system. For suction, users who have had experience working with our suction gripper are asked to annotate pixels of suctionable and non-suctionable areas on raw RGB-D images overlooking cluttered bins full of various objects. Similarly, users with experience using our parallel-jaw gripper are asked to sparsely annotate positive and negative grasps over re-projected heightmaps of cluttered bins, where each grasp is represented by a pixel on the heightmap and an angle corresponding to the orientation (parallel-jaw motion) of the gripper. On the interface, users directly paint labels on the images with wide-area circular (suction) or rectangular (grasping) brushstrokes. The diameter and angle of the strokes can be adjusted with hotkeys. The color of the

strokes are green for positive labels and red for negative labels. Examples of images and labels from this dataset can be found in Fig. 3.7. During training, we further augment each grasp label by adding additional labels via small jittering (less than 1.6cm). In total, the grasping dataset contains 1837 RGB-D images with pixel-wise suction and grasp labels. We use a 4:1 training/testing split of these images to train and evaluate different grasping models.

Although this grasping dataset is small for training a deep network from scratch, we find that it is sufficient for fine-tuning our architecture with ResNets pre-trained on ImageNet. An alternative method would be to generate a large dataset of annotations using synthetic data and simulation, as in [66]. However, then we would have to bridge the domain gap between synthetic and real 3D data, which is difficult for arbitrary real-world objects (see further discussion on this point in the comparison to DexNet in Table 3.1). Manual annotations make it easier to embed in the dataset information about material properties which are difficult to capture in simulation (*e.g.* porous objects are non-suctionable, heavy objects are easier to grasp than to suction).

Evaluation. In the context of our grasping framework, a method is robust if it is able to consistently find at least one suction or grasp proposal that works. To reflect this, our evaluation metric is the precision of inferred proposals versus manual annotations. For suction, a proposal is considered a true positive if its pixel center is manually labeled as a suctionable area (false positive if manually labeled as an non-suctionable area). For grasping, a proposal is considered a true positive if its pixel center is nearby within 4 pixels and 11.25 degrees from a positive grasp label (false positive if nearby a negative grasp label).

We report the precision of our inferred proposals for different confidence percentiles across the testing split of our grasping dataset in Table 3.1. We compare our method to a heuristic baseline algorithm as well as to a state-of-the-art grasping algorithm Dex-Net [66, 68] versions 2.0 (parallel-jaw grasping) and 3.0 (suction) for which code

Figure 3.7: **Images and annotations from the grasping dataset** with labels for suction (top two rows) and parallel-jaw grasping (bottom two rows). Positive labels appear in green while negative labels appear in red.

is available. We use Dex-Net weights pre-trained on their original simulation-based dataset. As reported in [66, 68, 69], fine-tuning Dex-Net on real data does not lead to substantial increases in performance.

Table 3.1: Multi-affordance Grasping Performance

Primitive	**Method**	**Top-1**	**Top 1%**	**Top 5%**	**Top 10%**
Suction	Baseline	35.2	55.4	46.7	38.5
	Dex-Net	69.3	71.8	62.5	53.4
	ConvNet	**92.4**	83.4	66.0	52.0
Grasping	Baseline	92.5	90.7	87.2	73.8
	Dex-Net	80.4	87.5	79.7	76.9
	ConvNet	**96.7**	91.9	87.6	84.1

% precision of grasp proposals across different confidence percentiles.

The heuristic baseline algorithm computes suction affordances by estimating surface normal variance over the observed 3D point cloud (lower variance = higher affordance), and computes anti-podal grasps by detecting hill-like geometric structures in the 3D point cloud with shape analysis. Baselines details and code are available on our project webpage [2]. The heuristic algorithm for parallel-jaw grasping was highly fine-tuned to the competition scenario, making it quite competitive with our trained grasping ResNets. We did not compare to the other network architectures for parallel-jaw grasping described in Section 3.4 since those models could not completely converge during training.

The top-1 proposal from the baseline algorithm performs quite well for parallel jaw grasping, but performs poorly for suction. This suggests that relying on simple geometric cues from the 3D surfaces of objects can be quite effective for grasping, but less so for suction. This is likely because successful suction picking not only depends on finding smooth surfaces, but also highly depends on the mass distribution and porousness of objects – both attributes of which are less apparent from local geometry alone. Suctioning close to the edge of a large and heavy object may cause the object to twist off due to external wrench from gravity, while suctioning a porous object may prevent a strong suction contact seal.

Figure 3.8: **Common Dex-Net failure modes** for suction (left column) and parallel-jaw grasping (right column). Dex-Net's top-1 predictions are labeled in red, while our method's top-1 predictions are labeled in green. Our method is more likely to predict grasps near objects' center of mass (*e.g.* bag of salts (top left) and water bottle (top right)), more likely to avoid unsuctionable areas such as porous surfaces (*e.g.* mesh bag of marbles(bottom left)), and less susceptible to noisy depth data (bottom right).

Dex-Net also performs competitively on our benchmark with strong suction and grasp proposals across top 1% confidence thresholds, but with more false positives across top-1 proposals. By visualizing Dex-Net top-1 failure cases in Figure 3.8, we can observe several interesting failure modes that do not occur as frequently with our method. For suction, there are two common types of failures. The first involves false positive suction predictions on heavy objects. For example, shown in the top left image of Figure 3.8, the heavy ($2kg$) bag of Epsom salt can only be successfully suctioned near its center of mass (*i.e.*, near the green circle), which is located towards the bottom of the bag. Dex-Net is expectedly unaware of this, and often makes

predictions on the bag but farther from the center of mass (*e.g.* the red circle shows Dex-Net's top-1 prediction). The second type of failure mode involves false positive predictions on unsuctionable objects with mesh-like porous containers. For example, in the bottom left image of Figure 3.8, Dex-Net makes suction predictions (*e.g.* red circle) on a mesh bag of marbles – however the only region of the object that is suctionable is its product tag (*e.g.* green circle).

For parallel-jaw grasping, Dex-Net most commonly experiences two other types of failure modes. The first is that it frequently predicts false positive grasps on the edges of long heavy objects – regions where the object would slip due to external wrench from gravity. This is because Dex-Net assumes objects to be lightweight to conform to the payload ($< 0.25kg$) of the ABB YuMi robot where it is usually tested. The second failure mode is that Dex-Net often predicts false positive grasps on areas with very noisy depth data. This is likely because Dex-Net is trained in simulation with rendered depth data, so Dex-Net's performance is less optimal without higher quality 3D cameras (*e.g.* industrial Photoneo cameras).

Overall, these observations show that Dex-Net is a competitive grasping algorithm trained from simulation, but falls short in our application setup due to the domain gap between synthetic and real data. Specifically, the discrepancy between the 90%+ grasping success achieved by Dex-Net in their reported experiments [66, 68, 69] versus the 80% on our dataset is likely due to two reasons: our dataset consists of 1) a larger spectrum of objects, e.g., heavier than $0.25kg$; and 2) noisier RGB-D data, i.e., less similar to simulated data, from substantially more cost-effective commodity 3D sensors.

Speed. Our suction and grasp affordance algorithms were designed to achieve fast run-time speeds during test time by densely inferring affordances over images of the entire scene. Table 3.2 compares our run-time speeds to several state-of-the-art alternatives for grasp planning. Our numbers measure the time of each FCN forward pass,

Table 3.2: Grasp Planning Run-Times (sec.)

Method	Time
Lenz et al. [59]	13.5
Zeng et al. [123]	10 - 15
Hernandez et al. [40]	5 - 40 [a]
Schwarz et al. [104]	0.9 - 3.3
Dex-Net 2.0 [66]	0.8
Matsumoto et al. [72]	0.2
Redmon et al. [93]	0.07
Ours (suction)	0.06
Ours (grasping)	$0.05 \times n$ [b]

[a] times reported from [72] derived from [40].
[b] n = number of possible grasp angles (in our case n =16).

reported with an NVIDIA Titan X on an Intel Core i7-3770K clocked at 3.5 GHz, excluding time for image capture and other system-related overhead. Our FCNs run at a fraction of the time required by most other methods, while also being significantly deeper (with 101 layers) than all other deep learning methods.

3.6.2 Evaluating Novel Object Recognition

We evaluate our recognition algorithms using a 1 vs 20 classification benchmark. Each test sample in the benchmark contains 20 possible object classes, where 10 are known and 10 are novel, chosen at random. During each test sample, we feed to the recognition algorithm the product images for all 20 objects as well as an observed image of a grasped object. In Table 3.3, we measure performance in terms of average % accuracy of the top-1 nearest neighbor product image match of the grasped object. We evaluate our method against a baseline algorithm, a state-of-the-art network architecture for both visual search [11] and one-shot learning without retraining [58], and several variations of our method. The latter provides an ablation study to show the improvements in performance with every added component:

Nearest neighbor is a baseline algorithm where we compute features of product images and observed images using a ResNet-50 pre-trained on ImageNet, and use nearest neighbor matching with ℓ_2 distance. For nearest neighbor evaluation, the

difference between the matching accuracy for known objects and novel objects reflects the natural difference in distribution of objects in the testing set – the novel objects are more distinguishable from each other using ImageNet features alone than known objects.

Siamese network with weight sharing is a re-implementation of Bell et al. [11] for visual search and Koch et al. [58] for one shot recognition without retraining. We use a Siamese ResNet-50 pre-trained on ImageNet and optimized over training pairs in a Siamese fashion. The main difference between this method and ours is that the weights between the networks computing deep features for product images and observed images are shared.

Two-stream network without weight sharing is a two-stream network, where the networks' weights for product images and observed images are not shared. Without weight sharing the network has more flexibility to learn the mapping function and thus achieves higher matching accuracy. All the later models describe later in this section use this two stream network without weight sharing.

Two-stream + guided-embedding (GE) includes a guided feature embedding with ImageNet features for the product image stream. We find this model has better performance for novel objects than for known objects.

Two-stream + guided-embedding (GE) + multi-product-images (MP) By adding a multi-anchor switch, we see more improvements to accuracy for novel objects. This is the final network architecture for N-net.

Two-stream + guided-embedding (GE) + multi-product-images (MP) + auxiliary classification (AC) By adding an auxiliary classification, we achieve near perfect accuracy of known objects for later models, however, at the cost of lower accuracy for novel objects. This also improves known vs novel (K vs N) classification accuracy for the recollection stage. This is the final network architecture for K-net.

Two-stage system As described in Section 3.5, we combine the two different models - one that is good at known objects (K-net) and the other that is good at novel objects (N-net) - in the two stage system. This is our final recognition algorithm, and it achieves better performance than any single model for test cases with a mixture of known and novel objects.

3.6.3 Full System Evaluation in Amazon Robotics Challenge

To evaluate the performance of our system as a whole, we used it as part of our MIT-Princeton entry for the 2017 Amazon Robotics Challenge (ARC), where state-of-the-art pick-and-place solutions competed in the context of a warehouse automation task. Participants were tasked with designing a fully autonomous robot system to grasp and recognize a large variety of different objects from unstructured bins. The objects were characterized by a number of difficult-to-handle properties. Unlike earlier versions of the competition ([18]), half of the objects were novel to the robot in the 2017 edition by the time of the competition. The physical objects as well as related item data (i.e. product images, weight, 3D scans), were given to teams just 30 minutes before the competition. While other teams used the 30 minutes to collect training data for the new objects and re-train models, our unique system did not require any of that during those 30 minutes.

Setup. Our system setup for the competition features several differences. We incorporated weight sensors to our system, using them as a guard to signal stop for grasping primitive behaviors during execution. We also used the measured weights of objects provided by Amazon to boost recognition accuracy to near perfect performance as well as to prevent double-picking. Green screens made the background more uniform to further boost accuracy of the system in the recognition phase. For inferring affordances, Table 3.1 shows that our data-driven methods with ConvNets provide more precise affordances for both suction and grasping than the baseline al-

Table 3.3: Recognition Evaluation (% Accuracy of Top-1 Match)

Method	K vs N	Known	Novel	Mixed
Nearest Neighbor	69.2	27.2	52.6	35.0
Siamese ([58])	70.3	76.9	68.2	74.2
Two-stream	70.8	85.3	75.1	82.2
Two-stream + GE	69.2	64.3	79.8	69.0
Two-stream + GE + MP (N-net)	69.2	56.8	**82.1**	64.6
N-net + AC (K-net)	**93.2**	**99.7**	29.5	78.1
Two-stage K-net + N-net	93.2	93.6	77.5	**88.6**

gorithms. For the case of parallel-jaw grasping, however, we did not have time to develop a fully stable network architecture before the day of the competition, so we decided to avoid risks and use the baseline grasping algorithm. The ConvNet-based approach became stable with the reduction to inferring only horizontal grasps and rotating the input heightmaps.

State tracking and estimation. We also designed a state tracking and estimation algorithm for the full system in order to perform competitively in the picking task of the ARC, where the goal is to pick *target* objects out of a storage system (*e.g.* shelves, separate work-cells) and place them into specific boxes for order fulfillment.

The goal of our state tracking algorithm is to track all the objects identities, 6D poses, amodal bounding boxes, and support relationships in each bin (bin_i) of the storage system. This information is then used by the task planner during the picking task to prioritize certain pick proposals (close to, or above target objects) over others. Our state tracking algorithm is built around the assumption that: 1)The state of the objects in the storage system only changes when there is an external force (robot or human) that interacts with the storage system. 2) We have knowledge of all external interactions in terms of their action type, object category, and specific storage bin. The action types include:

- add ($object_i$, bin_i): add $object_i$ to bin_i.
- remove ($object_i$, bin_i): remove $object_i$ from bin_i.

- move ($object_i$, bin_i): update $object_i$s location in bin_i (assumes $object_i$ is already in bin_i) .

- touch (bin_i): update all object poses in bin_i.

When **adding** an object into the storage system (*e.g.* during the stowing task), we first use the recognition algorithm described in section 3.5 to identify the objects class category before placing it into a bin. Then our state tracking algorithm captures RGB-D images of the storage system at time t (before the object is placed) and at time $t + 1$ (after the object is placed). The difference between the RGB-D images captured at $t+1$ and t provides an estimate for the visible surfaces of the newly placed object (*i.e.*, near the pixel regions with the largest change). 3D models of the objects (either constructed from the same RGB-D data captured during recognition for novel objects or given by another system for known objects) are aligned to these visible surfaces via ICP-based pose estimation ([123]). To reduce the uncertainty and noise of these pose estimates, the placing primitive actions are gently executed – *i.e.*, the robot arm holding the object moves down slowly until contact between the object and storage system is detected with weight sensors, upon which then the gripper releases the object.

For the **remove** operation, we first verify the objects identity using the recognition algorithm described in section 3.5. We then remove the object ID ($object_i$) from the list of tracked objects in bin_i.

The **move** operation is called whenever the robot attempts to remove an object from a storage bin but fails due to grasping failure. When this operation is called the system will compare the depth images captured before and after the robots interaction to identify the moved objects new point cloud. The system will then re-estimate the objects pose using the ICP-based method used during the add operation.

The **touch** operation is used to detect and compensate for unintentional state changes during robot interactions. This operation is called whenever the robot at-

tempts to add, remove, or move an object in a storage bin. When this operation is called, the system will compare and compute the correspondence of the color image before and after the interaction using SIFT-flow ([62]), ignoring the region of newly added or removed objects. If the difference between the two images is larger than a threshold, we will update each objects 6D pose by aligning its 3D model to its new corresponding point cloud (obtained from the SIFT-flow) using ICP.

Combined with our affordance prediction algorithm described in section 3.4, we are able to label each grasping or suction proposal with corresponding object identities using their tracked 6D poses from the state tracker. The task planner can then prioritize certain grasp proposals (close to, or above target objects) with heuristics based on this information.

Results. During the ARC 2017 final stowing task, we had a 58.3% pick success with suction, 75% pick success with grasping, and 100% recognition accuracy during the stow task of the ARC, stowing all 20 objects within 24 suction attempts and 8 grasp attempts. Our system took 1st place in the stowing task, being the only system to have successfully stowed all known and novel objects and to have finished the task well within the allotted time frame.

Overall, the pick success rates of all teams in the ARC (62% on average reported by [79]) are generally lower than those reported in related work for grasping. We attribute this mostly to the fact that the competition uses bins full of objects that contain significantly more clutter and variety than the scenarios presented in more controlled experiments in prior work. Among the competing teams, we successfully picked the most objects in the Stow and Final Tasks, and our average picking speed was the highest [79].

Postmortem. Our system did not perform as well during the finals task of the ARC due to lack of sufficient failure recovery. On the systems side, the perception node that fetches data from all RGB-D cameras lost connection to one of our RGB-D

cameras for recognition and stalled during the middle of our stowing run for the ARC finals, which forced us to call for a hard reset during the competition. The perception node would have benefited from being able to restart and recover from disconnections. On the algorithms side, our state tracking system is particularly sensitive to drastic changes in the state (*i.e.*, when multiple objects switch locations), which causes it to lose track without recovery. In hindsight, the tracking would have benefited from some form of simultaneous object segmentation in the bin that works for novel objects and is robust to clutter. Adopting the pixel-wise deep metric learning method of the ACRV team described in [74] would be worth exploring as part of future work.

Chapter 4

Learning Visual Affordances for Sequential Manipulation

In the previous chapter, we showed that learning visual affordances enables robotic systems to acquire complex grasping skills that generalize to novel objects while using orders of magnitude less training data. However, the system is limited to picking objects that can be directly perceived and grasped by one of the primitive picking motions. Real scenarios, especially when targeting the grasp of a particular object, often require plans that deliberately sequence different primitive motions. For example, when removing an object to pick the one below, or when separating two objects before grasping one. This points to a more complex picking policy with a planning horizon that includes preparatory primitive motions like pushing whose value is difficult to reward/label in a supervised fashion.

In this chapter, we show that we can extend visual affordances with model-free deep reinforcement learning to learn policies that sequence primitive picking motions. We demonstrate that this approach makes it possible to discover and learn the synergies between non-prehensile (e.g. pushing) and prehensile (e.g. grasping) actions. Our method involves training two fully convolutional networks that map from vi-

sual observations to actions: one infers the utility of pushes for a dense pixel-wise sampling of end effector orientations and locations, while the other does the same for grasping. Both networks are trained jointly in a Q-learning framework and are entirely self-supervised by trial and error, where rewards are provided from successful grasps. In this way, our policy learns pushing motions that enable future grasps, while learning grasps that can leverage past pushes. During picking experiments in both simulation and real-world scenarios, we find that our system quickly learns complex behaviors amid challenging cases of clutter, and achieves better grasping success rates and picking efficiencies than baseline alternatives after only a few hours of training. We further demonstrate that our method is capable of generalizing to novel objects.

4.1 Learning Synergies between Pushing and Grasping

Skilled manipulation benefits from the synergies between non-prehensile (*e.g.* pushing) and prehensile (*e.g.* grasping) actions: pushing can help rearrange cluttered objects to make space for arms and fingers (see Fig. 4.1); likewise, grasping can help displace objects to make pushing movements more precise and collision-free.

Although considerable research has been devoted to both push and grasp planning, they have been predominantly studied in isolation. Combining pushing and grasping policies for sequential manipulation is a relatively unexplored problem. Pushing is traditionally studied for the task of precisely controlling the pose of an object. However, in many of the synergies between pushing and grasping, pushing plays a loosely defined role, *e.g.* separating two objects, making space in a particular area, or breaking up a cluster of objects. These goals are difficult to define or reward for model-based [70, 35, 43] or data-driven [10, 28, 48] approaches.

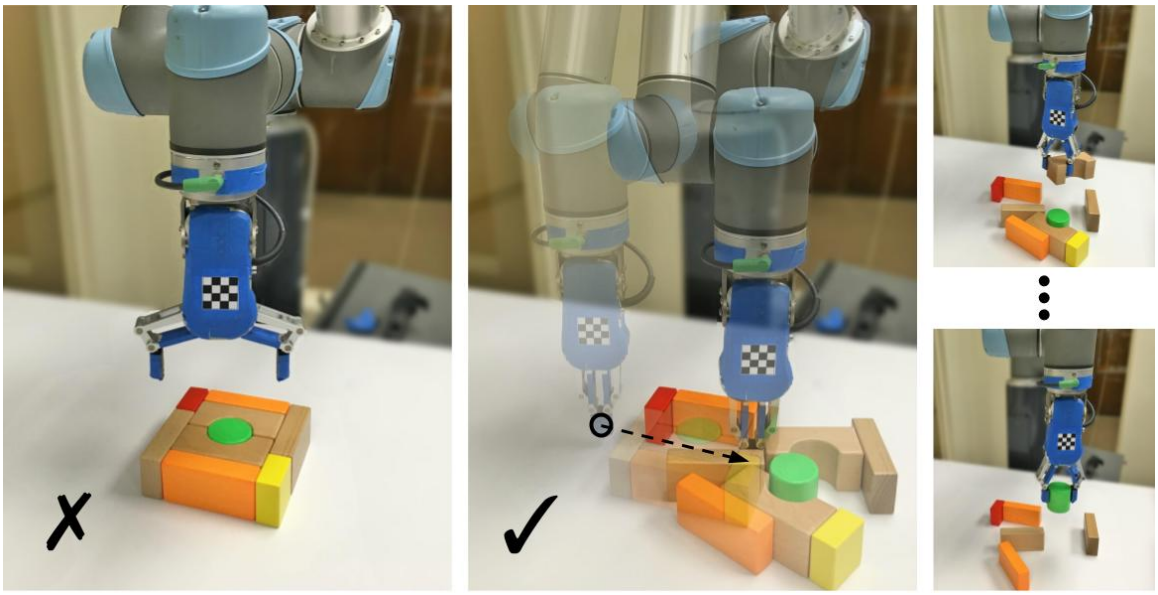


Figure 4.1: **Example configuration of tightly packed blocks** reflecting the kind of clutter that commonly appears in real-world scenarios (*e.g.* with stacks of books, boxes, etc.), which remains challenging for grasp-only manipulation policies. Our model-free system is able to plan pushing motions that can isolate these objects from each other, making them easier to grasp; improving the overall stability and efficiency of picking.

Many recent successful approaches to learning grasping policies, maximize affordance metrics learned from experience [88, 125] or induced by grasp stability metrics [36, 67]. However, it remains unclear how to plan sequences of actions that combine grasps and pushes, each learned in isolation. While hard-coded heuristics for supervising push-grasping policies have been successfully developed by exploiting domain-specific knowledge [24], they limit the types of synergistic behaviors between pushing and grasping that can be performed.

In this work, we propose to discover and learn synergies between pushing and grasping from experience through model-free deep reinforcement learning (in particular, Q-learning). The key aspects of our system are:

- We learn joint pushing and grasping policies through self-supervised trial and error. Pushing actions are useful only if, in time, enable grasping. This is in

contrast to prior approaches that define heuristics or hard-coded objectives for pushing motions.

- We train our policies end-to-end with a deep network that takes in visual observations and outputs expected return (*i.e.*, in the form of Q values) for potential pushing and grasping actions. The joint policy then chooses the action with the highest Q value – *i.e.*, , the one that maximizes the expected success of current/future grasps. This is in contrast to explicitly perceiving individual objects and planning actions on them based on hand-designed features [14].

This formulation enables our system to execute complex sequential manipulations (with pushing and grasping) of objects in unstructured picking scenarios and generalizes to novel objects (unseen in training).

Training deep end-to-end policies (*e.g.* from image pixels to joint torques) with reinforcement learning on physical systems can be expensive and time-consuming due to their prohibitively high sample complexity [60, 90, 92]. To make training tractable on a real robot, we simplify the action space to a set of end-effector-driven motion primitives. We formulate the task as a pixel-wise labeling problem: where each image pixel – and image orientation – corresponds to a specific robot motion primitive (pushing or grasping) executed on the 3D location of that pixel in the scene. For pushing, this location represents the starting position of the pushing motion; for grasping, the middle position between the two fingers during parallel-jaw grasping. We train a fully convolutional network (FCN) to take an image of the scene as input, and infer dense pixel-wise predictions of future expected reward values for all pixels – and thereby all robot motion primitives executed for all visible surfaces in the scene. This pixel-wise parameterization of robot primitive actions, which we refer to as **fully convolutional action-value functions** [125], enables us to train effective pushing and grasping policies on a single robot arm in less than a few hours of robot time.

The main contribution of this chapter is a new perspective to bridging data-driven prehensile and non-prehensile manipulation. We demonstrate that it is possible to train end-to-end deep networks to capture complementary pushing and grasping policies that benefit from each other through experience. We provide several experiments and ablation studies in both simulated and real settings to evaluate the key components of our system. Our results show that the pushing policies enlarge the set of scenarios in which grasping succeeds, and that both policies working in tandem produce complex interactions with objects (beyond our expectations) that support more efficient picking (*e.g.* pushing multiple blocks at a time, separating two objects, breaking up a cluster of objects through a chain of reactions that improves grasping). We provide additional qualitative results (video recordings of our robot in action), code, pre-trained models, and simulation environments at http://vpg.cs.princeton.edu

4.2 Related Work

Our work lies at the intersection of robotic manipulation, computer vision, and machine learning. We briefly review the related work in these domains.

Non-prehensile manipulation. Planning non-prehensile motions, such as pushing, is a fundamental problem that dates back to the early days of robotic manipulation. The literature in this area is vast, emerging early from classical solutions that explicitly model the dynamics of pushing with frictional forces [70, 35]. While inspiring, many of these methods rely on modeling assumptions that do not hold in practice [120, 10]. For example, non-uniform friction distributions across object surfaces and the variability of friction are only some of the factors that can lead to erroneous predictions of friction-modeling pushing solutions in real-world settings. While recent methods have explored data-driven algorithms for learning the dynamics of pushing [100, 73, 128], many of these works have largely focused on the execution of stable

pushes for one object at a time. Modeling the larger-scale consequences of pushing in the face of severe clutter and friction variation continues to be a complex problem; effectively using these models to discover optimal policies in real settings – even more so.

Grasping. Grasping too, has been well studied in the domain of model-based reasoning; from modeling contact forces and their resistance to external wrenches [91, 116], to characterizing grasps by their ability to constrain object mobility [96]. A common approach to deploying these methods in real systems involves pre-computing grasps from a database of known 3D object models [34], and indexing them at run-time with point cloud registration for object pose estimation [123, 122]. These methods, however, typically assume knowledge of object shapes, poses, dynamics, and contact points – information which is rarely known for novel objects in unstructured environments.

More recent data-driven methods explore the prospects of training model-agnostic deep grasping policies [94, 88, 89, 36, 67, 125] that detect grasps by exploiting learned visual features, and without explicitly using object specific knowledge (*i.e.*, shape, pose, dynamics). Pinto *et al.* [89] improve the performance of these deep policies by using models pre-trained on auxiliary tasks such as poking. Zeng *et al.* [125] demonstrate that using FCNs to efficiently model these policies with affordances can drastically improve run-times. Analogous to these methods, our data-driven framework is model-agnostic, but with the addition of improving the performance of grasping by incorporating non-prehensile actions like pushing.

Pushing with grasping. Combining both non-prehensile and prehensile manipulation policies is interesting, albeit an area of research that has been much less explored. The seminal work of Dogar *et al.* [24] presents a robust planning framework for push-grasping (non-prehensile motions baked within grasping primitives) to reduce grasp uncertainty as well as an additional motion primitive – sweeping – to

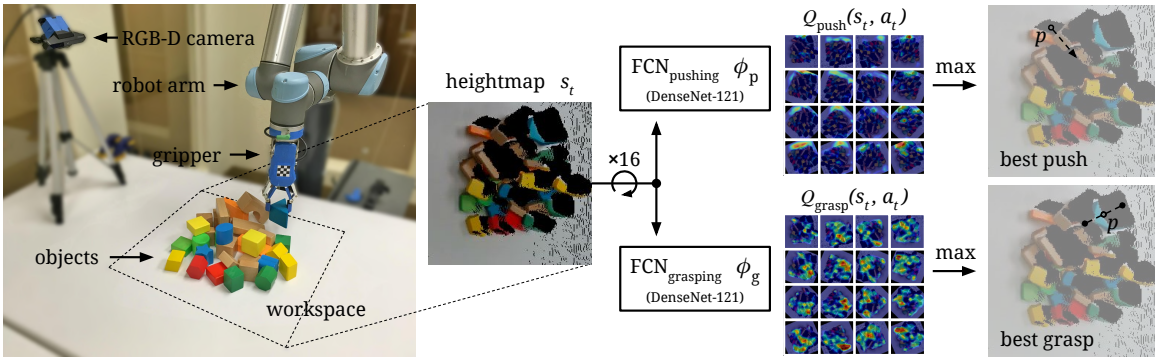


Figure 4.2: **Overview** of our system and Q-learning formulation. Our robot arm operates over a workspace observed by a statically mounted RGB-D camera. Visual 3D data is re-projected onto an orthographic RGB-D heightmap, which serves as a representation of the current state s_t. The heightmaps are then fed into two FCNs - one ϕ_p inferring pixel-wise Q values (visualized with heat maps) for pushing to the right of the heightmap and another ϕ_g for horizontal grasping over the heightmap. Each pixel represents a different location on which to execute the primitive. This is repeated for 16 different rotations of the heightmap to account for various pushing and grasping angles. These FCNs jointly define our deep Q function and are trained simultaneously.

move around obstacles in clutter. The policies in their framework, however, remain largely handcrafted. In contrast, our method is data-driven and learned online by self-supervision.

Other methods [84, 48] explore the model-free planning of pushing motions to move objects to target positions that are more favorable for pre-designed grasping algorithms – the behaviors of which are typically handcrafted, fixed, and well-known in advance. This knowledge is primarily used to define concrete goals (*e.g.* target positions) that can aid in the design or training of pushing policies. However, trying to define similar goals for data-driven model-agnostic grasping policies (where optimal behaviors emerge from experience) become less clear, as these policies are constantly learning, changing, and adapting behaviors over time with more data.

More closely related to our work is that of Boularias *et al.* [14], which explores the use of reinforcement learning for training control policies to select among push and grasp proposals represented by hand-crafted features. They propose a pipeline that first segments images into objects, proposes pushing and grasping actions, extracts

hand-tuned features for each action, then executes the action with highest expected reward. While inspiring, their method models perception and control policies separately (not end-to-end); it relies on model-based simulation to predict the motion of pushed objects and to infer its benefits for future grasping (those predictions are the two "features" provided to the pushing policy); it is tuned to work mainly for convex objects, and demonstrated on only one scenario with only two objects (a cylinder next to a box). In contrast, we train perception and control policies with end-to-end deep networks; we make no assumptions about the shapes or dynamics of objects (model-free), and we demonstrate that our formulation works not only for a variety of test cases with numerous objects (up to 30+), but also that it is capable of quickly generalizing to novel objects and scenarios. To the best of our knowledge, our work is the first model-free system to perform reinforcement learning of complementary pushing and grasping policies with deep networks that operate end-to-end from visual observations to actions.

4.3 Problem Formulation

We formulate the task of pushing-for-grasping as a Markov decision process: at any given state s_t at time t, the agent (*i.e.*, robot) chooses and executes an action a_t according to a policy $\pi(s_t)$, then transitions to a new state s_{t+1} and receives an immediate corresponding reward $R_{a_t}(s_t, s_{t+1})$. The goal of our robotic reinforcement learning problem is to find an optimal policy π^* that maximizes the expected sum of future rewards, given by $R_t = \sum_{i=t}^{\infty} \gamma R_{a_i}(s_i, s_{i+1})$, *i.e.*, γ-discounted sum over an infinite-horizon of future returns from time t to ∞.

In this work, we investigate the use of off-policy Q-learning to train a greedy deterministic policy $\pi(s_t)$ that chooses actions by maximizing the action-value function (*i.e.*, Q-function) $Q_\pi(s_t, a_t)$, which measures the expected reward of taking action a_t

in state s_t at time t. Formally, our learning objective is to iteratively minimize the temporal difference error δ_t of $Q_\pi(s_t, a_t)$ to a fixed target value y_t:

$$\delta_t = |Q(s_t, a_t) - y_t|$$

$$y_t = R_{a_t}(s_t, s_{t+1}) + \gamma\, Q(s_{t+1}, \underset{a'}{\operatorname{argmax}}(Q(s_{t+1}, a')))$$

where a' is the set of all available actions.

4.4 Method

This section provides details of our Q-learning formulation, network architectures, and training protocols.

4.4.1 State Representations

We model each state s_t as an RGB-D heightmap image representation of the scene at time t. To compute this heightmap, we capture RGB-D images from a fixed-mount camera, project the data onto a 3D point cloud, and orthographically back-project upwards in the gravity direction to construct a heightmap image representation with both color (RGB) and height-from-bottom (D) channels (see Fig. 4.2). The edges of the heightmaps are predefined with respect to the boundaries of the agent's workspace for picking. In our experiments, this area covers a 0.448^2m tabletop surface. Since our heightmaps have a pixel resolution of 224×224, each pixel spatially represents a 2^2mm vertical column of 3D space in the agent's workspace.

4.4.2 Primitive Actions

We parameterize each action a_t as a motion primitive behavior ψ (*e.g.* pushing or grasping) executed at the 3D location q projected from a pixel p of the heightmap

image representation of the state s_t:

$$a = (\psi, q) \mid \psi \in \{\text{push}, \text{grasp}\}, qp \in s_t$$

Our motion primitive behaviors are defined as follows:

Pushing: q denotes the starting position of a 10cm push in one of $k = 16$ directions. The trajectory of the push is straight. It is physically executed in our experiments using the tip of a closed two-finger gripper.

Grasping: q denotes the middle position of a top-down parallel-jaw grasp in one of $k = 16$ orientations. During a grasp attempt, both fingers attempt to move 3cm below q (in the gravity direction) before closing the fingers. In both primitives, robot arm motion planning is automatically executed with stable, collision-free IK solves [22].

4.4.3 Learning Fully Convolutional Action-Value Functions

We extend vanilla deep Q-networks (DQN) [76] by modeling our Q-function as two feed-forward fully convolutional networks (FCNs) [64] ϕ_{p} and ϕ_{g}; one for each motion primitive behavior (pushing and grasping respectively). Each individual FCN ϕ_ψ takes as input the heightmap image representation of the state s_t and outputs a dense pixel-wise map of Q values with the same image size and resolution as that of s_t, where each individual Q value prediction at a pixel p represents the future expected reward of executing primitive ψ at 3D location q where $qp \in s_t$. Note that this formulation is a direct amalgamation of Q-learning with visual affordance-based manipulation [125].

Both FCNs ϕ_{p} and ϕ_{g} share the same network architecture: two parallel 121-layer DenseNet [46] pre-trained on ImageNet [21], followed by channel-wise concatenation and 2 additional 1×1 convolutional layers interleaved with nonlinear activation func-

tions (ReLU) [81] and spatial batch normalization [49], then bilinearly upsampled. One DenseNet tower takes as input the color channels (RGB) of the heightmap, while the other takes as input the channel-wise cloned depth channel (DDD) (normalized by subtracting mean and dividing standard deviation) of the heightmap.

To simplify learning oriented motion primitives for pushing and grasping, we account for different orientations by rotating the input heightmap s_t into $k = 16$ orientations (different multiples of 22.5°) and then consider only horizontal pushes (to the right) and grasps in the rotated heightmaps. Thus, the input to each FCN ϕ_ψ is $k = 16$ rotated heightmaps, and the total output is 32 pixel-wise maps of Q values (16 for pushes in different directions, and 16 for grasps at different orientations). The action that maximizes the Q-function is the primitive and pixel with the highest Q value across all 32 pixel-wise maps: $\text{argmax}_{a'_t}(Q(s_t, a'_t)) = \text{argmax}_{(\psi,p)}(\phi_\text{p}(s_t), \phi_\text{g}(s_t))$.

Our pixel-wise parameterization of both state and action spaces enables the use of FCNs as Q-function approximators, which provides several advantages. First, the Q value prediction for each action now has an explicit notion of spatial locality with respect to other actions, as well as to the input observation of the state (*e.g.* with receptive fields). Second, FCNs are efficient for pixel-wise computations. Each forward pass of our network architecture ϕ_ψ takes on average 75ms to execute, which enables computing Q values for all 1,605,632 (*i.e.*, $224 \times 224 \times 32$) possible actions within 2.5 seconds. Finally, our FCN models can converge with less training data since the parameterization of end effector locations (pixel-wise sampling) and orientations (by rotating s_t) enables convolutional features to be shared across locations and orientations (*i.e.*, equivariance to translation and rotation).

Additional extensions to deep networks for Q-function estimation such as double Q-learning [114], and duelling networks [115], have the potential to improve performance but are not the focus of this work.

4.4.4 Rewards

Our reward scheme for reinforcement learning is simple. We assign $R_{\text{g}}(s_t, s_{t+1}) = 1$ if a grasp is successful (computed by thresholding on the antipodal distances between gripper fingers after a grasp attempt) and $R_{\text{p}}(s_t, s_{t+1}) = 0.5$ for pushes that make detectable changes to the environment (where changes are detected if the sum of differences between heightmaps exceeds some threshold τ, *i.e.*, $\sum(s_{t+1} - s_t) > \tau$). Note that the intrinsic reward $R_{\text{p}}(s_t, s_{t+1})$ does not explicitly consider whether a push enables future grasps. Rather, it simply encourages the system to make pushes that cause change. The synergy between pushing and grasping is learned mainly through reinforcement (see experiments in Sec. 4.5.3).

4.4.5 Training details.

Our Q-learning FCNs are trained at each iteration i using the Huber loss function:

$$\mathcal{L}_i = \begin{cases} \frac{1}{2}(Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-})^2, \text{for } |Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}| < 1, \\ |Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}| - \frac{1}{2}, \text{ otherwise.} \end{cases}$$

where θ_i are the parameters of the neural network at iteration i, and the target network parameters θ_i^- are held fixed between individual updates. We pass gradients only through the single pixel p and network ϕ_ψ from which the value predictions of the executed action a_i was computed. All other pixels at iteration i backpropagate with 0 loss.

We train our FCNs ϕ_ψ by stochastic gradient descent with momentum, using fixed learning rates of 10^{-4}, momentum of 0.9, and weight decay 2^{-5}. Our models are trained in PyTorch with an NVIDIA Titan X on an Intel Xeon CPU E5-2699 v3 clocked at 2.30GHz. We train with prioritized experience replay [103] using stochastic rank-based prioritization, approximated with a power-law distribution. Our explo-

ration strategy is ϵ-greedy, with ϵ initialized at 0.5 then annealed over training to 0.1. Our future discount γ is constant at 0.5.

In our experiments (Sec. 4.5), we train all of our models by self-supervision with the same procedure: n objects (*i.e.*, toy blocks) are randomly selected and dropped into the 0.448^2m workspace in front of the robot. The robot then automatically performs data collection by trial and error, until the workspace is void of objects, at which point n objects are again randomly dropped into the workspace. In simulation $n = 10$, while in real-world settings $n = 30$.

4.4.6 Testing details.

Since our policy is greedy deterministic during test time, it is possible for it to get stuck repeatedly executing the same action while the state representation (and thus value estimates) remain the same as no change is made to the environment. Naively weighting actions based on visit counts can also be inefficient due our pixel-wise parameterization of the action space. Hence to alleviate this issue, during testing we prescribe a small learning rate to the network at 10^{-5} and continue backpropagating gradients through the network after each executed action. For the purposes of evaluation, network weights are reset to their original state (after training and before testing) prior to each new experiment test run – indicated by when all objects in the workspace have been successfully grasped (*i.e.*, completion) or when the number of consecutively executed actions for which there is no change to the environment exceeds 10.

4.5 Experiments

We executed a series of experiments to test the proposed approach, which we call **Visual Pushing for Grasping (VPG)**. The goals of the experiments are three-

fold: 1) to investigate whether the addition of pushing as a motion primitive can enlarge the set of scenarios in which objects can successfully be grasped (*i.e.*, does pushing help grasping), 2) to test whether it is feasible to train pushing policies with supervision mainly from the future expected success of another grasping policy trained simultaneously, and 3) to demonstrate that our formulation is capable of training effective, non-trivial pushing-for-grasping policies directly from visual observations on a real system.

4.5.1 Baseline Methods

To address these goals, we compare the picking performance of VPG to the following baseline approaches:

Reactive Grasping-only Policy (Grasping-only) is a grasping policy that uses the same pixel-wise state and action space formulation as our proposed method described in Section 5.3, but uses a single FCN supervised with binary classification (from trial and error) to infer pixel-wise affordance values between 0 and 1 for grasping only. This baseline is a greedy deterministic policy that follows the action which maximizes the immediate grasping affordance value at every time step t. This baseline is analogous to a self-supervised version of a state-of-the-art top-down parallel-jaw grasping algorithm [125]. For a fair comparison, we extend that method using DenseNet [46] pre-trained on ImageNet [21].

Reactive Pushing and Grasping Policy (P+G Reactive) is an augmented version of the previous baseline, but with an additional FCN to infer pixel-wise affordance values between 0 and 1 for pushing. Both networks are trained with binary classification from self-supervised trial and error, where pushing is explicitly supervised with a binary value from change detection (as described in Section 4.4.4). Change detection is the simplest form of direct supervision for pushing, but requires higher values of ϵ for the exploration strategy to maintain stable training. This policy follows the

action which maximizes the immediate affordance value (which can come from either the pushing or grasping FCNs).

Both aforementioned baselines are reactive as they do not plan long-horizon strategies, but instead greedily choose actions based on affordances computed from the current state s_t. Our training optimization parameters for these baselines are kept the same as that of VPG.

4.5.2 Evaluation Metrics

We test the methods by executing a series of tests in which the system must pick and remove objects from a table with novel arrangements of objects (as described in Sec. 4.4.6).

For each test, we execute n runs ($n \in \sim 10, 30$) and then evaluate performance with 3 metrics: 1) the average % completion rate over the n test runs, which measures the ability of the policy to finish the task by picking up all objects without failing consecutively for more than 10 attempts, 2) the average % grasp success rate per completion, and 3) the % action efficiency (defined as $\frac{\text{\# objects in test}}{\text{\# actions before completion}}$), which describes how succinctly the policy is capable of finishing the task. Note that grasp success rate is equivalent to action efficiency for grasping-only policies. For all of these metrics, higher is better.

We run experiments on both simulated and real-world platforms. While our main objective is to demonstrate effective VPG policies on a real robot, we also run experiments in simulation to provide controlled environments for fair evaluation between methods and for ablation studies. In experiments on both platforms, we run tests with objects placed in both random and challenging arrangements.

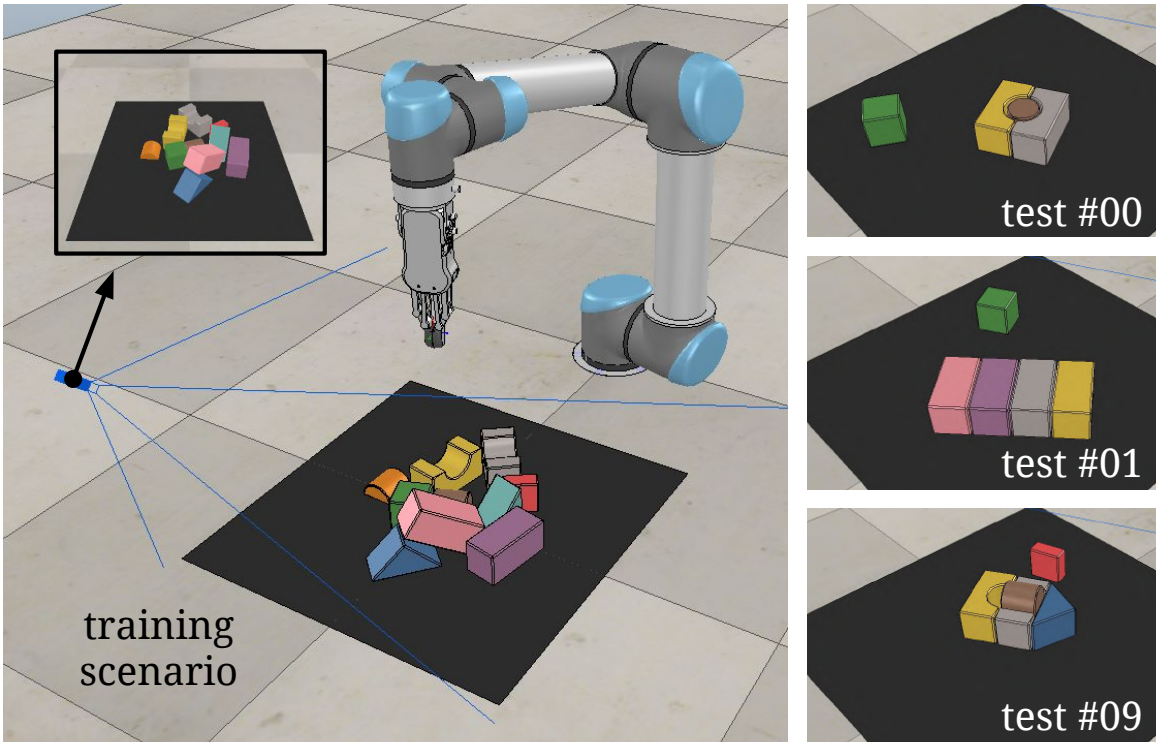


Figure 4.3: **Simulation environment.** Policies are trained in scenarios with random arrangements of 10 objects (left), then evaluated in scenarios with varying degrees of clutter (10 objects, 30 objects, or challenging object arrangements). In the most challenging scenarios, adversarial clutter was manually engineered to reflect challenging real-world picking scenarios (*e.g.* tightly packed boxes, as shown on the right).

4.5.3 Simulation Experiments

Our simulation setup uses a UR5 robot arm with an RG2 gripper in V-REP [25] (illustrated in Fig. 4.3) with Bullet Physics 2.83 for dynamics and V-REP's internal inverse kinematics module for robot motion planning. Each test run in simulation was run $n = 30$ times. The objects used in these simulations include 9 different 3D toy blocks, the shapes and colors of which are randomly chosen during experiments. Most dynamics parameters are kept default except friction coefficients, which have been modified to achieve synthetic object interaction behaviors as similar as possible to that of the real-world. We did not perform any tuning of random seeds for the simulated physics in our experiments. We also simulate a statically mounted perspective 3D camera in the environment, from which perception data is captured. RGB-D images

Table 4.1: Simulation Results on Random Arrangements (Mean %)

Method	Completion	Grasp Success	Action Efficiency
Grasping-only [125]	90.9	55.8	55.8
P+G Reactive	54.5	59.4	47.7
VPG	**100.0**	**67.7**	**60.9**

of resolution 640 × 480 are rendered with OpenGL from the camera, without any noise models for depth or color.

Comparisons to Baselines. Our first experiment compares VPG to the two baseline methods in a simulation where 30 objects are randomly dropped onto a table. This scenario is similar to the training scenario, except it has 30 objects rather than 10, thus testing the generalization of policies to more cluttered scenarios. Results are shown in Table 4.1. We see that VPG outperforms both baseline methods across all metrics. It is interesting to note that P+G reactive performs poorly in terms of completion rates and action efficiency. This likely due to its tendency (in the face of clutter) to continually push objects around until they are forced out of the workspace as grasping affordances remain low.

Challenging Arrangements. We also compare VPG in simulation to the baseline methods on 11 challenging test cases with adversarial clutter. Each test case consists of a configuration of 3 - 6 objects placed in the workspace in front of the robot, 3 configurations of which are shown in Fig. 4.3. These configurations are manually engineered to reflect challenging picking scenarios, and remain exclusive from the training procedure (described in Sec. 4.4.5). Across many of these test cases, objects are laid closely side by side, in positions and orientations that even an optimal grasping policy would have trouble successfully picking up any of the objects without de-cluttering first. As a sanity check, a single isolated object is additionally placed in the workspace separate from the configuration. This is just to ensure that all policies have been sufficiently trained prior to the benchmark (*i.e.*, a policy is not ready if fails to grasp the isolated object).

Table 4.2: Simulation Results on Challenging Arrangements (Mean %)

Method	Completion	Grasp Success	Action Efficiency
Grasping-only [125]	40.6	51.7	51.7
P+G Reactive	48.2	59.0	46.4
VPG	**82.7**	**77.2**	**60.1**

Results are shown in Table 4.2. From the completion results, we observe that the addition of pushing enlarges the set of the scenarios for which successful grasping can be performed. Across the collection of test cases, the grasping-only policy frequently struggles to complete the picking task (with a 0% completion rate for 5 out of the 11 test cases). We observe this be particularly true in scenarios where large cuboids are laid closely side-by-side (Fig. 4.3). Even when the policy does successfully complete the task, the average grasp success rates remain relatively low at 50-60%.

Upon the addition of pushing as an additional action primitive in the P+G reactive policy, we immediately see an increase in picking completion rates and there are no longer cases in which the policy completely fails with a 0% completion rate. While the P+G reactive policy achieves higher completion and grasp success rates than grasping-only, the average action efficiency is lower. This suggests that the policy executes a large number of pushes, many of which are not succinct and may not actually help grasping. This is expected, since P+G reactive uses binary supervision from change detection for pushing – pushing motions are not directly supervised by how well they help grasping.

By enabling joint planning of pushing and grasping with VPG, we observe substantially higher completion and grasp success rates (with a 100% completion rate for 5 of the 11 test cases). The higher action efficiency also indicates that the pushes are now more succinct in how they help grasping.

No Pushing Rewards? We next investigate whether our method can learn synergistic pushing and grasping actions even without any intrinsic rewards for pushing ($R_{\text{p}}(s_t, s_{t+1}) = 0$). We call this variant of our algorithm "VPG-noreward". In this

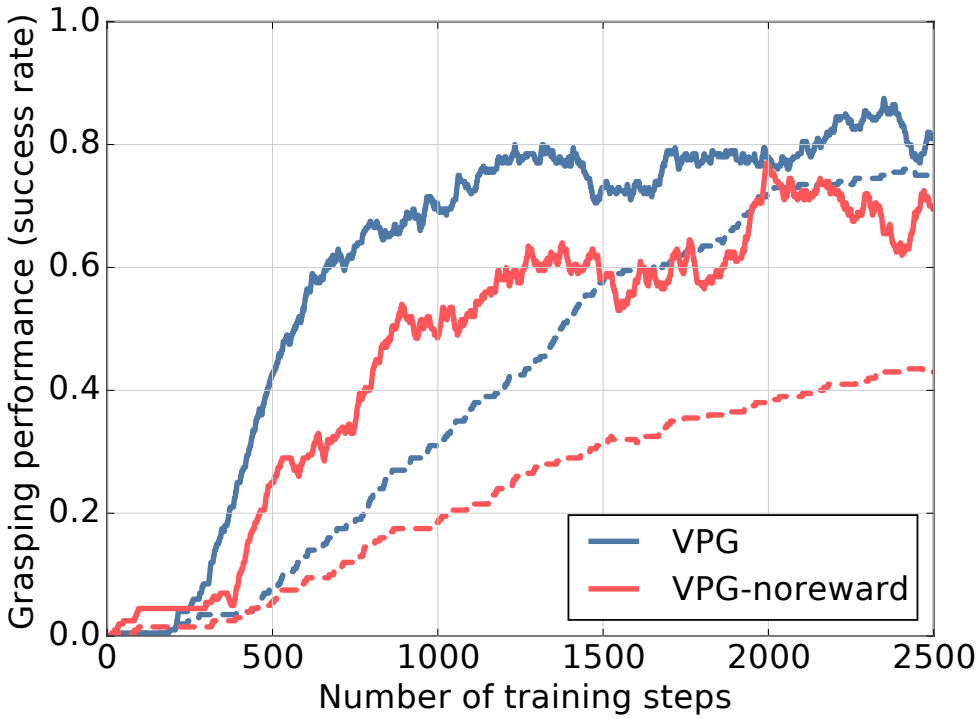


Figure 4.4: Comparing performance of VPG policies trained with and without rewards for pushing. Solid lines indicate % grasp success rates (primary metric of performance) and dotted lines indicate % push-then-grasp success rates (secondary metric to measure quality of pushes) over training steps.

more difficult setting, the pushing policy learns to effect change only through the reward provided by future grasps.

For this study, we run tests in simulation with 10 randomly placed objects. We report results with plots of grasping performance versus training steps. Grasping performance is measured by the % grasp success rate over the last $j = 200$ grasp attempts, indicated by solid lines in Fig. 4.4. We also report the % push-then-grasp success rates (*i.e.*, pushes followed immediately by a grasp – considered successful if the grasp was successful), indicated by dotted lines. Since there is no defacto way to measure the quality of the pushing motions for how well they benefit a model-free grasping policy, this secondary metric serves as a good approximation. The numbers reported at earlier training steps (*i.e.*, iteration $i < j$) in Fig. 4.4 are weighted by $\frac{i}{j}$. Each training step consists of capturing data, computing a forward pass, executing

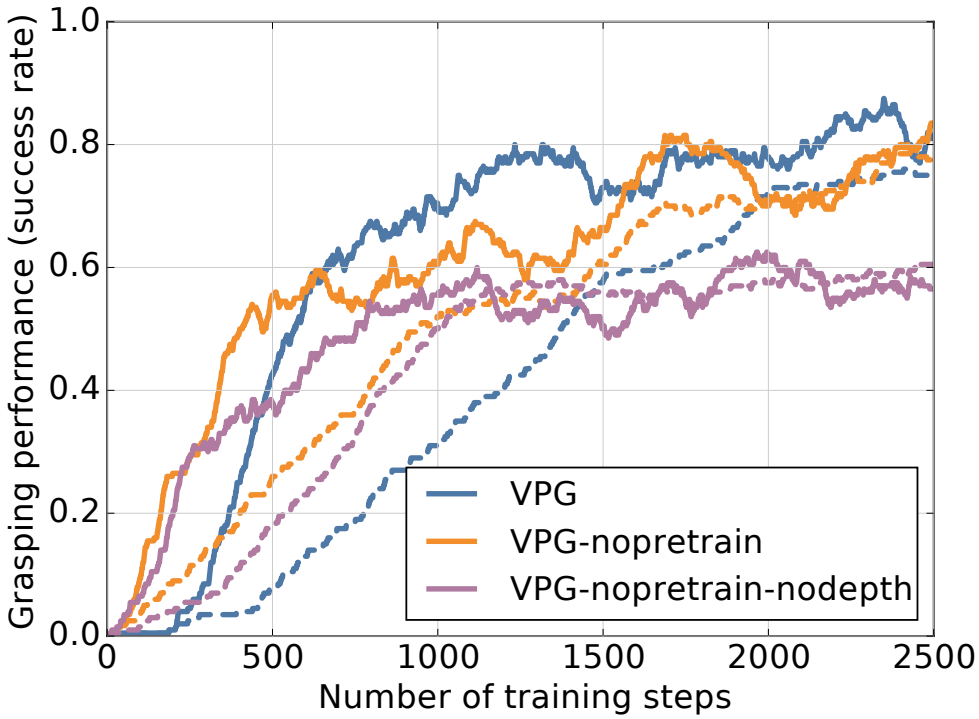


Figure 4.5: Comparing performance of VPG policies initialized without weights pre-trained on ImageNet and without the depth channels of the RGB-D heightmap (*i.e.*, no height-from-bottom, only color information). Solid lines indicate % grasp success rates (primary metric of performance) and dotted lines indicate % push-then-grasp success rates (secondary metric to measure quality of pushes) over training steps.

an action, backpropagating, and running a single iteration of experience replay (with another forward pass and backpropagation on a sample from the replay buffer).

From these results, we see that VPG-noreward is capable of learning effective pushing and grasping policies – achieving grasping success rates at 70-80%. We also see that it learns a pushing policy that increasingly helps grasping (note the positive slope of the dotted red line, which suggests pushes are helping future grasps more and more as the system trains). This rate of improvement is not as good as VPG, but the final performance is only slightly lower.

No ImageNet Pre-training? We trained a version of VPG ("VPG-nopretrain") without ImageNet pre-training of FCN weights (*i.e.*, with only random initialization) and report its performance versus training steps in Fig. 4.5. Interestingly, the results

suggest that ImageNet pre-training is not a major contributor to the sample efficiency of VPG nor to the final performance of the model in simulation. This could be due to the fact that the statistics of pixel patterns found in ImageNet images are different compared to that of re-projected heightmap images. The slight delay before the upward slope of the training curve could also be an artifact due to the FCNs spending early training steps to escape the ImageNet local optimum.

No Height-from-bottom Information? We trained another version of VPG ("VPG-nopretrain-nodepth") without ImageNet pre-training and without the depth channels of the RGB-D heightmap images (*i.e.*, no height-from-bottom, only color information) and report its performance in Fig. 4.5. This modification meant that each FCN ϕ_{p} and ϕ_{g} no longer has a second DenseNet tower to compute features from the channel-wise cloned depth channels (DDD) of the heightmaps. The results show that sample complexity remains similar, but the average final grasping performance is lower by about 15%. This suggests that geometric cues from the depth (height-from-bottom) channels are important for achieving reasonable grasping performance with VPG.

Shortsighted Policies? We also investigate the importance of long-term lookahead. Our Q-learning formulation in theory enables our policies to plan long-term strategies (*e.g.* chaining multiple pushes to enable grasping, grasping to enable pushing, grasping to enable other grasps, etc.). To test the value of these strategies, we trained a shortsighted version of VPG ("VPG-myopic") where the discount factor on future rewards is smaller at $\gamma = 0.2$ (trained in simulation with 10 randomly placed objects). We evaluate this policy over the 11 hard test cases in simulation and report comparisons to our method in Table 4.3. Interestingly, we see that VPG-myopic improves its grasping performance at a faster pace early in training (presumably optimizing for short-term grasping rewards), but ultimately achieves lower average performance (*i.e.*, grasp success, action efficiency) across most hard test cases. This suggests that

Table 4.3: Comparison with Myopic Policies (Mean %)

Method	Completion	Grasp Success	Action Efficiency
VPG-myopic	79.1	74.3	53.7
VPG	**82.7**	**77.2**	**60.1**

the ability to plan long-term strategies for sequential manipulation could benefit the overall stability and efficiency of pick-and-place.

4.5.4 Real-World Experiments

In this section, we evaluate the best performing variant of VPG (with rewards and long-term planning) on a real robot. Our real-world setup consists of a UR5 robot arm with an RG2 gripper, overlooking a tabletop scenario. Objects vary across different experiments, including a collection of 30+ different toy blocks for training and testing, as well as a collection of other random office objects to test generalization to novel objects (see Fig. 4.7). For perception data, RGB-D images of resolution 640×480 are captured from an Intel RealSense SR300, statically mounted on a fixed tripod overlooking the tabletop setting. The camera is localized with respect to the robot base by an automatic calibration procedure, during which the camera tracks the location of a checkerboard pattern taped onto the gripper. The calibration optimizes for extrinsics as the robot moves the gripper over a grid of 3D locations (predefined with respect to robot coordinates) within the camera field of view.

Random Arrangements. We first tested VPG on the real robot in cluttered environments with 30 randomly placed objects. Fig. 4.6 shows its performance versus training time in comparison to the grasping-only policy (baseline method) – where curves show the % grasp success rate over the last $m = 200$ grasp attempts (solid lines) and % push-then-grasp success rates (dotted lines) for both methods.

Interestingly, the improvement of performance early in training is similar between VPG and grasping-only. This is surprising, as one would expect VPG to require more

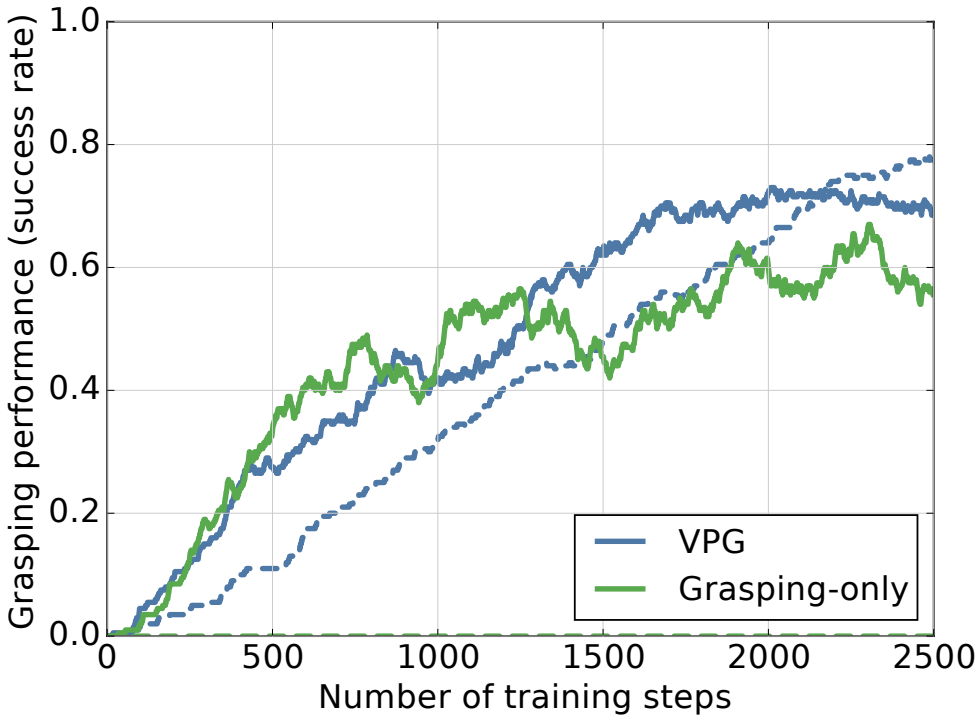


Figure 4.6: Evaluating VPG in real-world tests with random 30+ object arrangements. Solid lines indicate % grasp success rates (primary metric of performance) and dotted lines indicate % push-then-grasp success rates (secondary metric to measure quality of pushes) over training steps.

training samples (and thus more training time) to achieve comparable performance, since only one action (either a grasp or a push) can be executed per training step. This similarity in growth of performance can likely be attributed to our method optimizing the pushing policies to make grasping easier even at a very early stage of training. While the grasping-only policy is busy fine-tuning itself to detect harder grasps, VPG spends time learning pushes that can make grasping easier.

As expected, the grasping performance of the VPG policy surpasses that of the grasping-only policy in later training steps. Not only is the performance better, it is also less erratic. This is likely because it avoids long sequences of failed grasps, which happens occasionally for grasping-only when faced with highly cluttered configurations of objects.

This experiment also suggests that VPG is quite sample efficient – we are able to train effective pushing and grasping policies in less than 2000 transitions. At 10 seconds per action execution on a real robot, this amounts to about 5.5 hours of wall-clock training time. This is a substantial advantage over prior work on deep reinforcement learning for manipulation (*e.g.* 10 million sample transitions (10 hours of interaction time on 16 robots) for block stacking [90]).

Challenging Arrangements. We also ran experiments in the real-world comparing VPG with grasping-only on 7 challenging test cases with adversarial clutter (see examples in top row of Fig. 4.7). The results appear in Table 4.4. Note that the differences between VPG and Grasping-only are quite large in these challenging real-world cases.

Table 4.4: Real-world Results on Challenging Arrangements (Mean %)

Method	Completion	Grasp Success	Action Efficiency
Grasping-only [125]	42.9	43.5	43.5
VPG	**71.4**	**83.3**	**69.0**

Video recordings of these experiments are provided on our project webpage [4]. They show that the VPG pushing and grasping policies perform interesting synergistic behaviors, and are more capable of efficiently completing picking tasks in cluttered scenarios in tandem than grasping-only policies.

Novel Objects. Finally, we tested our VPG models (trained on toy blocks) on a collection of real-world scenes with novel objects (examples of which are shown in the bottom row of Fig. 4.7). Overall, the system is capable of generalizing to sets of objects that fall within a similar shape distribution as that of the training objects, but struggles when completely new shapes or anomalies (reflective objects with no depth data) are introduced. The robot is capable of planning complex pushing motions that can de-clutter scenarios with these novel objects. We also show several video recordings of these test runs on our project webpage [4].

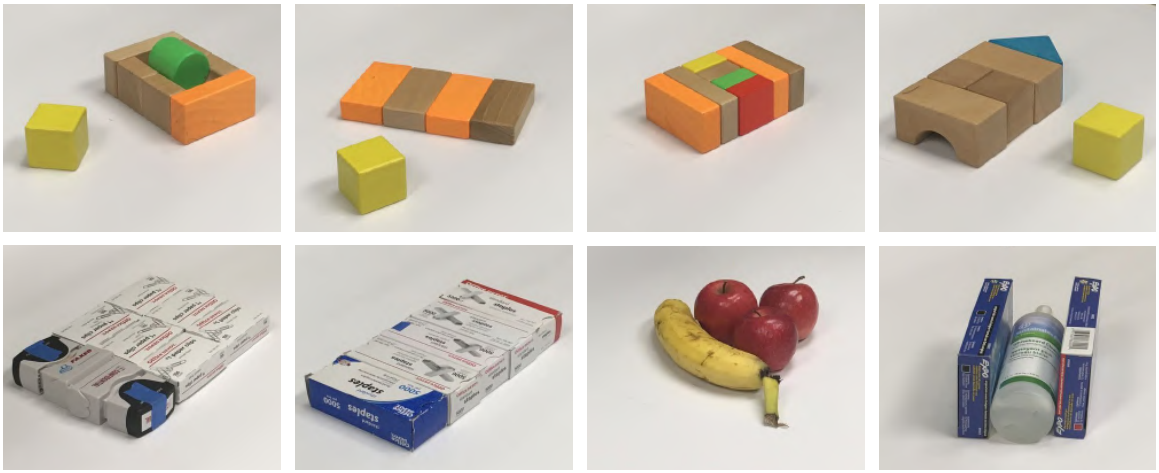


Figure 4.7: **Examples of challenging arrangements** in real-world settings with toy blocks (top row) and novel objects (bottom row).

Chapter 5

Learning Visual Affordances with Residual Physics

In the previous two chapters, we showed that we can learn the visual affordances of pushing and grasping motion primitives, both of which take in a spatial location as input (*e.g.* pushing starting points, or grasping points). The spatial location is typically inferred from the 3D location of the pixel with the highest predicted affordance value. However, if we have manipulation motion primitives that take in non-spatial parameters (*e.g.* robot arm velocity for throwing), how do we learn the visual affordances for such primitives in a sample efficient manner?

In this chapter, we show that we can combine visual affordances with residual physics (learning to predict residual values on top of control parameter estimates from an initial analytical controller) to efficiently learn how to grasp and throw arbitrary objects for pick-and-place. This formulation enables learning end-to-end visuomotor policies that leverage the benefits of analytical models (e.g. generalization) while still maintaining the capacity (via data-driven residuals) to account for real-world dynamics that are not explicitly modeled. Within this formulation, we investigate the synergies between grasping and throwing (*i.e.*, learning grasps that enable more

accurate throws) and between simulation and deep learning (*i.e.*, using deep networks to predict residuals on top of control parameters predicted by a physics simulator). The resulting system, ***TossingBot***, is able to grasp and successfully throw arbitrary objects into boxes located outside its maximum reach range at 500+ mean picks per hour (600+ grasps per hour with 85% throwing accuracy); and generalizes to new objects and target locations.

5.1 Learning to Throw Arbitrary Objects

Throwing is a means to increase the capabilities of a manipulator by exploiting dynamics, a form of dynamic extrinsic dexterity [15]. In the case of pick-and-place, throwing enables a robot arm to place objects rapidly into boxes located outside its maximum kinematic range, which not only reduces the total physical space used by the robot, but also maximizes its picking efficiency. Rather than having to transport objects to their destination before executing the next pick, objects are instead immediately "passed to Newton" (see Fig. 5.1).

However, precisely throwing *arbitrary objects* in unstructured settings is challenging because it depends on many factors: from pre-throw conditions (*e.g.* initial grasp of the object) to varying object-centric properties (*e.g.* mass distribution, friction, shape) and dynamics (*e.g.* aerodynamics). For example, grasping a screwdriver near the tip before throwing it can cause centripetal accelerations to swing it forward with significantly higher release velocities – resulting in drastically different projectile trajectories than if it were grasped closer to its center of mass (see Fig. 5.2). Yet regardless of how it is grasped, its aerial trajectory would differ from that of a thrown ping pong ball, which can significantly decelerate after release due to air resistance. Many of these factors are notoriously difficult to model or measure analytically [71] – hence prior studies are often confined to assuming homogeneous pre-throw conditions

Figure 5.1: **TossingBot** learns to grasp arbitrary objects from an unstructured bin and to throw them into target boxes located outside its maximum kinematic reach range. The aerial trajectory of different objects are controlled by jointly optimizing grasping policies and throwing release velocities.

(*e.g.* object fixtured in gripper or manually reset after each throw) with predetermined, homogeneous objects (*e.g.* balls or darts). Such assumptions rarely hold in real unstructured settings, where a throwing system needs to acquire its own pre-throw conditions (via grasping) and adapt its throws to account for varying properties and dynamics of arbitrary objects.

In this work, we present ***TossingBot***, an end-to-end formulation that uses trial and error to learn how to plan control parameters for grasping and throwing from visual observations. The formulation learns grasping and throwing jointly – discovering grasps that enable accurate throws, while learning throws that compensate for the dynamics of arbitrary objects. There are two key aspects to our system:

- **Joint learning of grasping and throwing policies** with a deep neural network that maps from visual observations (of objects in a bin) to control grasping

and throwing parameters: the likelihood of grasping success for a dense pixel-wise sampling of end effector orientations and locations [124], and the throwing release velocities for each sampled grasp. Grasping is directly supervised by the accuracy of throws (grasp success = accurate throw), while throws are directly conditioned on specific grasps (via dense predictions). As a result, the end-to-end policy learns to execute stable grasps that lead to predictable throws, as well as throwing velocities that account for the variations in object-centric properties and dynamics that can be inferred from visual information.

- **Residual learning of throw release velocities** δ on top of velocities $\hat{v}$ predicted by a physics controller based on an ideal ballistic motion. The complete controller uses the superposition of the two predictions to obtain a final throwing release velocity $v = \hat{v} + \delta$. The physics-based controller uses ballistics to provide consistent estimates of $\hat{v}$ that generalize well to different landing locations, while the data-driven residuals learn to compensate for object-centric properties and dynamics. Our experiments show that this hybrid data-driven method, *Residual Physics*, leads to significantly more accurate throws than baseline alternatives.

This formulation enables our system to grasp and throw arbitrary objects reliably into target boxes located outside its maximum reach range at 500+ mean picks per hour (MPPH), and generalizes to new objects and target landing locations.

The primary contribution of this chapter is to provide new perspectives on throwing: in particular – its relationship to grasping, its efficient learning by combining physics with trial and error, and its potential to improve practical real-world picking systems. We provide several experiments and ablation studies in both simulated and real settings to evaluate the key components of our system. We observe that throwing performance strongly correlates with the quality of grasps, and experimental results show that our formulation is capable of learning synergistic grasping and throwing policies for arbitrary objects in real settings. By visualizing deep features, we find

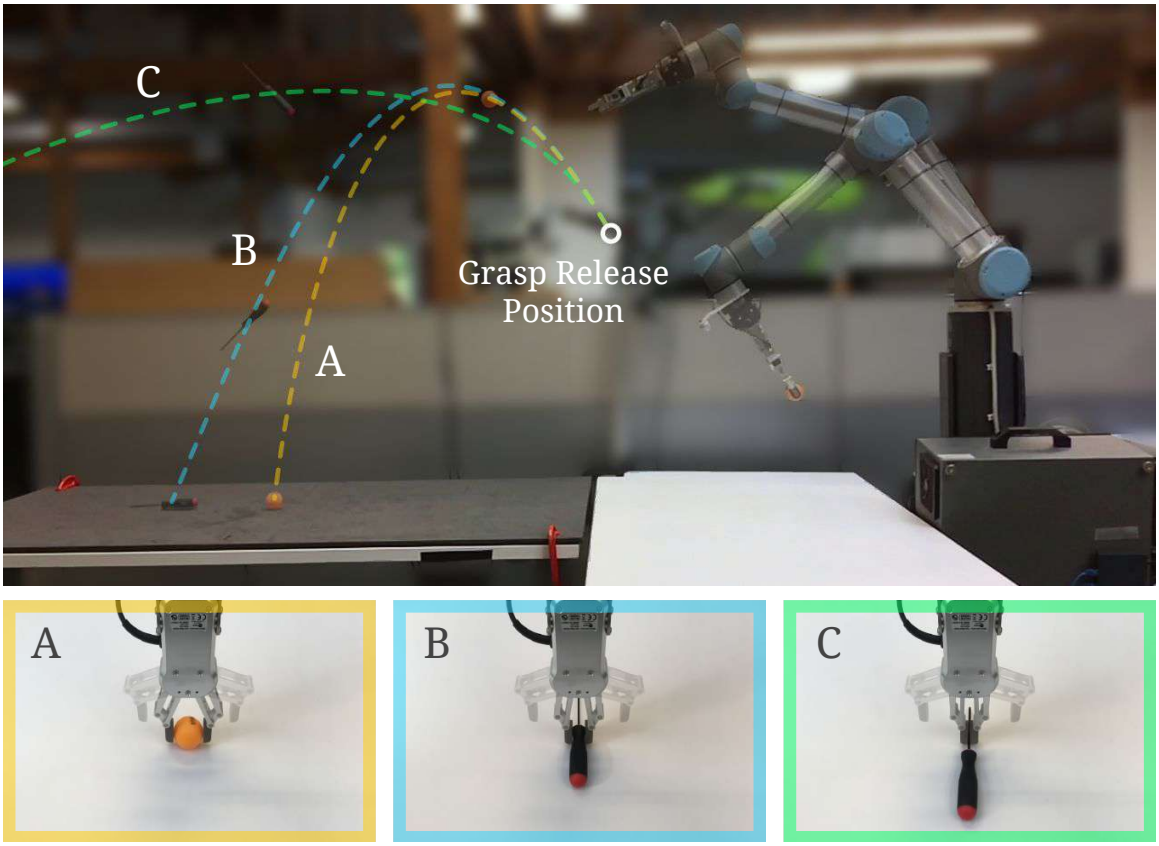


Figure 5.2: **Projectile trajectories** of a thrown ping pong ball (a), screwdriver grasped and thrown by its handle (b), and the same screwdriver grasped and thrown by its shaft (c). The difference between (a) and (b) is largely due to aerodynamics, while the difference between (b) and (c) is largely due to grasping at different offsets from the object's center of mass (near the handle). Our goal is to learn joint grasping and throwing policies that can compensate for these differences to achieve accurate targeted throws.

that TossingBot learns priors that enable it to distinguish visually between different objects based on their geometric and physical attributes – without any explicit supervision. Qualitative results (videos) are available at http://tossingbot.cs.princeton.edu [3]

5.2 Related Work

Analytical models for throwing. Many previous systems built for throwing [71, 29, 78, 105, 112] rely on handcrafting or approximating dynamics based on fric-

tional rigid body mechanics, and then optimizing control parameters to execute a throw such that the projectile (typically a ball) lands at a target location. However, as highlighted in Mason and Lynch [71], accurately modeling throwing dynamics is challenging. It requires knowledge of physical properties that are difficult to estimate (*e.g.* aerodynamics, inertia, coefficients of restitution, friction, shape, mass distribution etc.) for both objects and manipulators. As a result, these model-based systems often observe limited throwing accuracy (*e.g.* 40% success rate in [105]), and have difficulty generalizing to changing dynamics over time (*e.g.* deteriorating friction on gripper finger contact surfaces from repeated throwing). In our work, we leverage deep learning and self-supervision to compensate for the dynamics that are not explicitly accounted for in contact/ballistic models, and we train our policies online via trial and error so that they can adapt to new situations on the fly (*e.g.* new object and manipulator dynamics).

Learning models for throwing. More recently, learning-based systems for robotic throwing [6, 45, 57, 30] have also been proposed, which ignore low-level dynamics and directly optimize for task-level success signals (*e.g.* did the projectile land on the target?). These methods have demonstrated better accuracy than those which solely rely only on analytical models, but have two primary drawbacks: 1) limited generalization to new object types (beyond balls, blocks, or darts), and 2) limited pre-throw conditions (*e.g.* human operators are required to manually reset objects and manipulators to match a prescribed initial state before every throw), which makes training from trial and error costly. Both drawbacks prevent their use in real unstructured settings.

In contrast to prior work, we make no assumptions on the physical properties of thrown objects, nor do we assume that the objects are at a fixed pose in the gripper before each throw. Instead, we propose an object-agnostic pick-and-throw formulation that jointly learns to acquire its own pre-throw conditions (via grasping) while

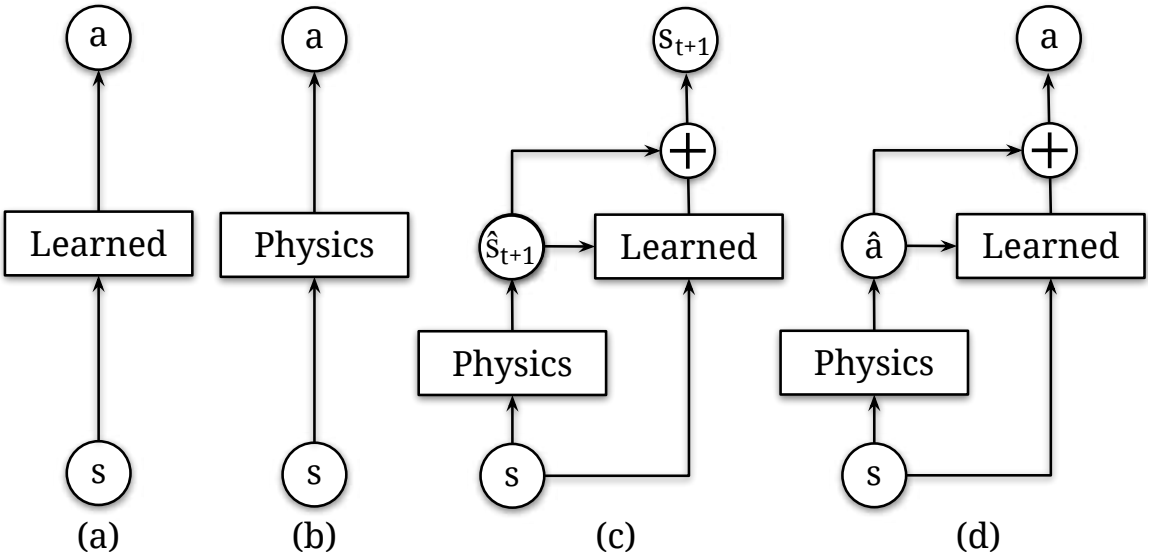


Figure 5.3: Learning residual models and policies: (a) analytical solutions that determine action a from state s; (b) data-driven policies that learn the direct mapping from states to actions; (c) hybrid models that combine analytical models with learning to predict future states s_{t+1}; (d) hybrid policies (like ours) that combine analytical solutions with learning to determine action a.

learning throwing control parameters that compensate for varying object properties and dynamics. The system learns from scratch through self-supervised trial and error, and resets it own training so that human intervention is kept at a minimum.

Learning residual models and policies. Our approach to data-efficient learning, *Residual Physics*, falls under a broader category of hybrid controllers [5, 85] that leverage both 1) analytical models to provide initial estimates of control parameters, and 2) learned residuals on top of those estimates to compensate for unknown dynamics (see Fig. 5.3d). In contrast to prior work on learning residuals on predictions of future states for model-based control [7, 56] or data-augmented models [27, 53], we instead directly learn the residuals on control parameters (*i.e.*, action space) with deep networks. This approach provides a wider range of data-driven corrections that can compensate for noisy observations as well as dynamics that are not explicitly modeled. These benefits are also observed in concurrent work on residual reinforcement learning [54, 108] in block-assembly and object manipulation tasks.

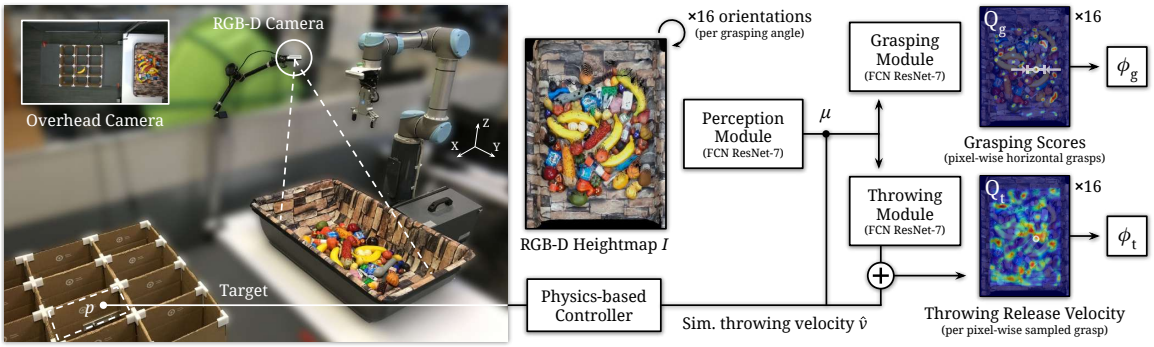


Figure 5.4: **Overview.** An RGB-D heightmap of the scene is fed into a perception module to compute spatial features μ. In parallel, target location p is fed into a physics-based controller to provide an initial estimate of throwing release velocity $\hat{v}$, which is concatenated with μ then fed into grasping and throwing modules. Grasping module predicts probability of grasp success for a dense pixel-wise sampling of horizontal grasps, while throwing module outputs dense prediction of residuals (per sampled grasp), which are added to $\hat{v}$ to get final predictions of throwing release velocities. We rotate input heightmaps by 16 orientations to account for 16 grasping angles. Robot executes the grasp with the highest score, followed by a throw using its corresponding predicted velocity.

5.3 Method Overview

TossingBot consists of a neural network $f(I, p)$ that takes as input a visual observation I of objects in a bin and the 3D position of a target landing location p, and outputs a prediction of parameters ϕ_g and ϕ_t used by two motion primitives for grasping and throwing respectively (see Fig. 5.4). The learning objective is to optimize the predictions of parameters ϕ_g and ϕ_t such that executing the grasping primitive using ϕ_g followed by the throwing primitive using ϕ_t results in an object (observed in I) landing on p at each time-step.

The network f consists of three parts: 1) a perception module that accepts visual input I and outputs a spatial feature representation μ; this is shared as input into 2) a grasping module that predicts ϕ_g and 3) a throwing module that predicts ϕ_t. f is trained end-to-end through self-supervision from trial and error by tracking the ground truth landing positions of thrown objects. The following subsections provide

an overview of these three modules, while the next two sections delve into details of the most novel aspects of the system.

5.3.1 Perception Module: Learning Visual Representations

We represent the visual input I as an RGB-D heightmap image of the workspace (*i.e.*, a bin of objects). To compute this heightmap, we capture RGB-D images from a fixed-mount camera, project the data onto a 3D point cloud, and orthographically back-project upwards in the gravity direction to construct a heightmap image representation with both color (RGB) and height-from-bottom (D) channels. The RGB and D channels are normalized (mean-subtracted and divided by standard deviation) so that learned convolutional filters can be shared across the two modalities.

The edges of the heightmaps are defined with respect to the boundaries of the robot's picking workspace. In our experiments, this area covers a 0.9×0.7m tabletop surface, on top of which we place a bin of objects. Our heightmaps have a pixel resolution of 180×140, hence each pixel $i \in I$ represents a 5×5mm vertical column of 3D space in the robot's workspace. Using its height-from-bottom value, each pixel thereby corresponds to a unique 3D location in the robot's workspace. The input I is fed into the perception network, a 7-layer fully convolutional residual network [8, 39, 64] (interleaved with 2 layers of spatial 2×2 max-pooling), which outputs a spatial feature representation μ of size $45 \times 35 \times 512$ that is then fed into the grasping and throwing modules.

5.3.2 Grasping Module: Learning Parallel-jaw Grasps

The grasping module consists of a grasping network that predicts the probability of grasping success for a predefined grasping primitive across a dense pixel-wise sampling of end effector locations and orientations in I.

Grasping primitive. The grasping primitive takes as input parameters $\phi_g = (x, \theta)$ and executes a top-down parallel-jaw grasp centered at a 3D location $x = (x_x, x_y, x_z)$ oriented θ° around the gravity direction. During execution, the open gripper approaches x along the gravity direction until the 3D position of the middle point between the gripper fingertips meets x, at which point the gripper closes, and lifts upwards 10cm. This primitive is open-loop, with robot arm motion planning executed using a stable, collision-free IK solver [22].

Grasping network. The grasping network is a 7-layer fully convolutional residual network [8, 39, 64] (interleaved with 2 layers of spatial bilinear $2\times$ upsampling). This accepts the visual feature representation μ as input, and outputs a probability map Q_g with the same image size and resolution as that of the input heightmap I. Each value of a pixel $q_i \in Q_g$ represents the predicted probability of grasping success (*i.e.*, grasping affordance) when executing a top-down parallel-jaw grasp centered at the 3D location of $i \in I$ with the gripper oriented horizontally with respect to the heightmap I.

As in [125, 124], we account for different grasping angles by rotating the input heightmap by 16 orientations (multiples of 22.5°) before feeding into the network. The pixel with the highest predicted probability among all 16 maps determines the parameters $\phi_g = (x, \theta)$ for the grasping primitive to be executed: the 3D location of a pixel determines the grasping position x, and the orientation of the heightmap determines grasping angle θ. This visual state and action representation has been shown to provide sample efficiency when used in conjunction with fully-convolutional action-value functions for grasping and pushing [124, 125]. Each pixel-wise prediction shares convolutional features for all grasping locations and orientations (*i.e.*, translation and rotation equivariance).

5.3.3 Throwing Module: Learning Throwing Velocities

The goal of the throwing module is to predict the release position and velocity of a predefined throwing primitive for each possible grasp (over the dense pixel-wise sampling of end effector locations and orientations in I).

Throwing primitive. The throwing primitive takes as input parameters $\phi_t = (r, v)$ and executes an end effector trajectory such that the mid-point between the gripper fingertips reaches a desired release position $r = (r_x, r_y, r_z)$ and velocity $v = (v_x, v_y, v_z)$, at which point the gripper opens and releases the object. During execution, the robot arm curls inwards while grasping onto an object, then uncurls outward at high speed, releasing the object at the desired position and velocity. Throughout this motion, the gripper is oriented such that the axis between the fingertips is orthogonal to the plane of the intended aerial trajectory. In our system, the direction of curling/uncurling aligns with (v_x, v_y). Fig. 5.2 visualizes this motion primitive and its end effector trajectory. The throwing primitive is executed after each successful grasp attempt (checked by thresholding the distance between fingertips).

Planning the release position. In most real-world settings, only a handful of release positions are accessible by the robot for throwing. So for simplicity in our system, we directly derive the release position r from the given target landing location p using two assumptions: 1) the aerial trajectory of a projectile is linear on the xy-horizontal-plane and in the same direction as $v_{x,y} = (v_x, v_y)$. In other words, we assume that the forces of aerodynamic drag *orthogonal* to $v_{x,y}$ are negligible. This is not to be confused with the primary forces of drag that exist in *parallel* to $v_{x,y}$, which our system will learn to compensate. We also assume 2) that $\sqrt{r_x^2 + r_y^2}$ is at a fixed distance c_d from the robot base origin, and that r_z is at a fixed constant height c_h. Formally, these constraints can be written as: $(r_{x,y} - p_{t_{x,y}}) \times v_{x,y} = 0$ and $\sqrt{r_x^2 + r_y^2} = c_d$ and $r_z = c_h$. In our experiments, we select constant values of c_h and c_d

such that all release positions are accessible by the robot: $c_h = 0.04$m and $c_d = 0.7$m in simulation, and $c_h = 0.02$m and $c_d = 0.76$m in real settings.

Planning the release velocity. Given a target landing location p and release position r, there could be multiple solutions of the release velocity v for which the object lands on p. To remove this ambiguity, we further constrain the direction of v to be angled 45° upwards in the direction of p. Formally, this constraint can be defined as $\|v_{x,y}\| = v_z$. Under all the aforementioned constraints, the only unknown variable for throwing is $\|v_{x,y}\|$, which represents the magnitude of the final release velocity. Specifically, assuming a fixed throwing release height r_z, fixed release distance c_d from robot base origin, and release velocity direction angled 45° upwards: for any given target landing location $p = (p_x, p_y, p_z)$, we can derive the release position r and release velocity magnitude $\|v\|$ that achieves the target landing location p assuming equations of linear projectile motion:

$$\begin{aligned} \theta &= \arctan\left(\frac{p_y}{p_x}\right) \\ r_x &= c_d \sin(\theta) \\ r_y &= c_d \cos(\theta) \end{aligned} \tag{5.1}$$

$$\|v\| = \sqrt{\frac{a(p_x^2 + p_y^2)}{(r_z - p_z - \sqrt{p_x^2 + p_y^2})}} \tag{5.2}$$

where a is acceleration from gravity.

These equations are valid for any given target landing location p, as long as both $\|v\|$ and r are within robot physical limits. Hence assuming no aerial obstacles, varying only the velocity magnitude $\|v\|$ is sufficient to cover the space of all possible projectile landing locations. In the following section, we describe how the throwing module predicts $\|v_{x,y}\|$.

5.4 Learning Residual Physics for Throwing

A key aspect of TossingBot's throwing module is that it learns to predict a residual δ on top of the estimated release velocity $\|\hat{v}_{x,y}\|$ from a physics-based controller (*i.e.*, ballistic equations of projectile motion), then uses the superposition of the two predictions to compute a final release velocity $\|v_{x,y}\| = \|\hat{v}_{x,y}\| + \delta$ for the throwing primitive. Conceptually, this enables our models to leverage the advantages of physics-based controllers (*e.g.* generalization via analytical models), while still maintaining the capacity (via data-driven residual δ) to account for aerodynamic drag and offsets to the real-world projectile velocity (conditioned on the grasp), which are otherwise not analytically modeled. Our experiments in Sec. 5.6 show that this approach, a.k.a. *Residual Physics*, yields significant improvements in both accuracy and generalization of throwing arbitrary objects compared to baseline alternatives: *e.g.* using only the physics-based controller (Fig. 5.3a), or directly training f to regress $\|v_{x,y}\|$ (Fig. 5.3b).

Physics-based controller. The physics-based controller uses the standard equations of linear projectile motion, by assuming a grasp on the center of mass of the object, to analytically solve back for the release velocity $\hat{v}$ given the target landing location p and release position r of the throwing primitive: $p = r + \hat{v}t + \frac{1}{2}at^2$. This controller assumes that the aerial trajectory of the projectile moves along a ballistic path affected only by gravity, which imparts a downward acceleration $a_z = -9.8\text{m/s}^2$.

We also provide the estimated physics-based release velocity $\hat{v}$ as input into both the grasping and throwing networks by concatenating the visual feature representation μ with a k-channel image ($k = 128$) where each pixel holds the value of $\hat{v}$, repeated across channels. Providing $\hat{v}$ as input enables our grasping and throwing predictions to be conditioned on $\hat{v}$ – *i.e.*, larger values of $\hat{v}$ for farther target locations can lead to a different set of effective grasps.

This physics-based controller has several advantages in that it provides a closed-form solution, generalizes well to new landing locations p, and serves as a consistent approximation for v. However, it also relies on several assumptions that do not generally hold. First, it assumes that the effects of aerodynamic drag are negligible. However, as we show in our experiments in Fig. 5.2, the aerial trajectory for lightweight objects like ping pong balls can be substantially influenced by drag. Second, it assumes that the gripper release velocity v directly determines the velocity of the projectile. This is not true since the object is often not grasped at the center of mass, nor is the object completely immobilized by the grasp prior to release. For example, as illustrated in Fig. 5.2, a screwdriver picked up by the shaft can be flung forward with a significantly higher velocity than the gripper release velocity due to centripetal forces, resulting in a farther aerial trajectory.

Estimating residual release velocity. To compensate for the shortcomings of the physics-based controller, the throwing module includes a throwing network that predicts a residual δ on top of the estimated release velocity $\|\hat{v}_{x,y}\|$ for each possible grasp. The throwing network is a 7-layer fully convolutional residual network [39] interleaved with 2 layers of spatial bilinear $2\times$ upsampling that accepts the visual feature representation μ as input, and outputs an image Q_t with the same size and resolution as that of the input heightmap I. Q_t has a pixel-wise one-to-one spatial correspondence with I, thus each pixel in Q_t also corresponds one-to-one with the pixel-wise probability predictions of grasping success $q_i \in Q_g$ (for all possible grasps using rotating input I). Each pixel in Q_t holds a prediction of the residual value δ_i added on top of the estimated release velocity $\|\hat{v}_{x,y}\|$ from a physics-based controller, to compute the final release velocity v_i of the throwing primitive after executing the grasp at pixel i. The better the prediction of δ_i, the more likely the grasped and thrown object will land on the target location p.

5.5 Jointly Learning Grasping and Throwing

Our full network f (including the perception, grasping, and residual throwing modules) is trained end-to-end using the following loss function: $\mathcal{L} = \mathcal{L}_g + y_i \mathcal{L}_t$, where $\mathcal{L}_g$ is the binary cross-entropy error from predictions of grasping success:

$$\mathcal{L}_g = -(y_i \log q_i + (1 - y_i) \log(1 - q_i))$$

and $\mathcal{L}_t$ is the Huber loss from its regression of δ_i for throwing:

$$\mathcal{L}_t = \begin{cases} \frac{1}{2}(\delta_i - \bar{\delta}_i)^2, \text{for } |\delta_i - \bar{\delta}_i| < 1, \\ |\delta_i - \bar{\delta}_i| - \frac{1}{2}, \text{ otherwise.} \end{cases}$$

where y_i is the binary ground truth grasp success label and $\bar{\delta}_i$ is the ground truth residual label. We use an Huber loss [32] instead of an MSE loss for regression since we find that it is less sensitive to inaccurate outlier labels. We pass gradients only through the single pixel i on which the grasping primitive was executed. All other pixels backpropagate with 0 loss.

We train our network f by stochastic gradient descent with momentum, using fixed learning rates of 10^{-4}, momentum of 0.9, and weight decay 2^{-5}. Our models are trained from scratch (*i.e.*, random initialization) in PyTorch with an NVIDIA Titan X on an Intel Xeon CPU E5-2699 v3 clocked at 2.30GHz. We train with prioritized experience replay [103] using stochastic rank-based prioritization, approximated with a power-law distribution. Our exploration strategy is ϵ-greedy, with ϵ initialized at 0.5 then annealed over training to 0.1. Specifically, when executing a grasp, the robot has an ϵ chance to sample a random grasp within the robot's workspace for picking; likewise when executing a throw, the robot has an ϵ chance to explore a random positive release velocity.

Training via self-supervision. We obtain our ground truth training labels y_i and $\bar{\delta}_i$ through trial and error. At each training step, the robot captures RGB-D images to construct visual input I, performs a forward pass of $f(I, p)$ to make a prediction of primitive parameters ϕ_g and ϕ_t, executes the grasping primitive using ϕ_g, then executes the throwing primitive using ϕ_t. We obtain ground truth grasp success labels y_i by one of two ways:

1. Success after grasping, by checking the distance between gripper fingertips after the grasping primitive.
2. Success after throwing, by checking the binary signal of whether or not a throw lands in the correct box.

As we show in Sec. 5.6.5, supervising grasps by the accuracy of throws eventually leads to more stable grasps and better overall throwing performance, since the grasping policy learns to favor grasps that lead to successful throws. After each throw, we measure the object's actual landing location $\bar{p}$ using a calibrated overhead RGB-D camera to detect changes in the landing zone before and after the throw. Regardless of where the object lands, its actual landing location $\bar{p}$ and the executed release velocity v is recorded and saved to the experience replay buffer as a training sample, with which we obtain the ground truth residual label $\bar{\delta}_i = \|v_{x,y}\| - \|\hat{v}_{x,y}\|_{\bar{p}}$.

In experiments in Sec. 5.6, we train our models by self-supervision with the same procedure: n objects are randomly dropped into the 0.9×0.7m workspace in front of the robot. The robot performs data collection until the workspace is void of objects, at which point n objects are again randomly dropped into the workspace. In simulation $n = 12$, while in real-world experiments $n = 80+$. In our real-world setup, the landing zone (on which target boxes are positioned) is slightly titled at a 15° angle adjacent to the bin. When the workspace is void of objects, the robot lifts the bottomless boxes such that the objects slide back into the bin. In this way, human intervention is kept at a minimum during the training process.

5.6 Evaluation

We execute a series of experiments in simulated and real settings to evaluate the learned grasping and throwing policies. The goal of the experiments are four-fold: 1) to evaluate the overall accuracy and efficiency of our pick-and-throw system on arbitrary objects, 2) to test its generalization to new objects and target locations unseen during training, 3) to investigate how learned grasps can improve the accuracy of subsequent throws, and 4) to compare our proposed method based on *Residual Physics* to other baseline alternatives.

Evaluation metrics are 1) grasping success: the % rate which an object remains in the gripper after executing the grasping primitive (by measuring distance between fingertips), and 2) throwing success: the % rate which a thrown object lands in the intended target box (tracked by an overhead camera).

5.6.1 Experimental Setup

We evaluate each policy on its ability to grasp and throw various objects into 12 boxes located outside a UR5 robot arm's maximum reach range (as shown in Fig. 5.1). Specifically, the task is to pick objects from a cluttered bin and stow them uniformly into the boxes such that all boxes have the same number of objects, regardless of object type. Since boxes are located *outside* the robot's reach range, throwing is necessary to succeed in the task. Each box is 20cm tall with a 25×15cm opening. The middle of the top opening of each box is used as the input target landing position p to the formulation $f(I, p)$.

Simulation setup. The simulation environment (show in Fig. 5.6) is built using PyBullet [19]. We use 8 different objects: 4 seen during training and 4 unseen for testing. Training objects are chosen in order of increasing difficulty: 4cm-diameter ball, $4 \times 4 \times 4$cm cube, 3cm-diameter 16cm-long rod, and a 16cm-long hammer (union

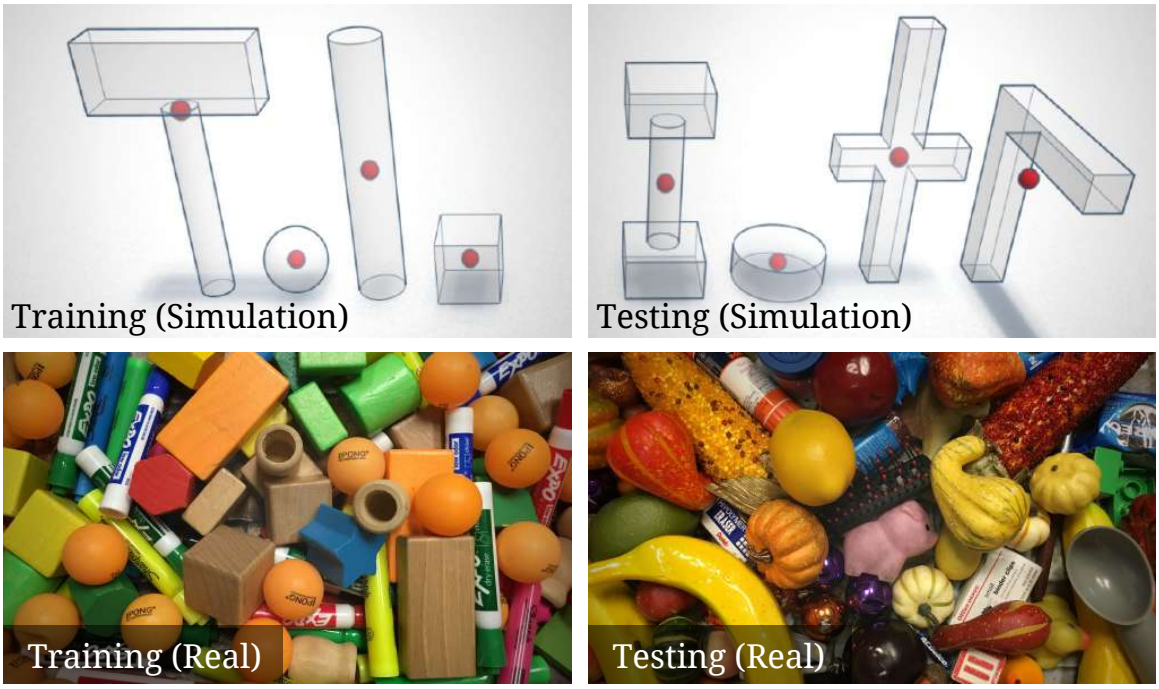


Figure 5.5: **Objects** used in simulated (top) and real (bottom) experiments, split by training objects (left) and unseen testing objects (right). The center of mass of each simulated object is indicated with a red sphere (for illustration).

of 2cm-diameter 12cm-long rod with $10 \times 4 \times 2.5$cm block). Throwing difficulty is determined by how much an object's projectile trajectory changes depending on its initial grasp and center of mass (CoM). For example, the trajectory of the ball is mostly agnostic to grasp location and orientation, while both rod (CoM in middle) and hammer objects (CoM between handle and shaft) can have drastically different projectile trajectories depending on the grasping point. Objects are illustrated in Fig. 5.5 – their CoMs indicated with a red sphere. Multiple copies of each object (12 in total) are randomly colored and dropped into the bin during training and testing.

Although simulation provides a consistent and controlled environment for fair ablative analyses, the simulated environment does not account for aerodynamics, and as a result, performance in simulation does not necessarily reflect the performance in the real world. Therefore we also provide quantitative experiments on a real system.

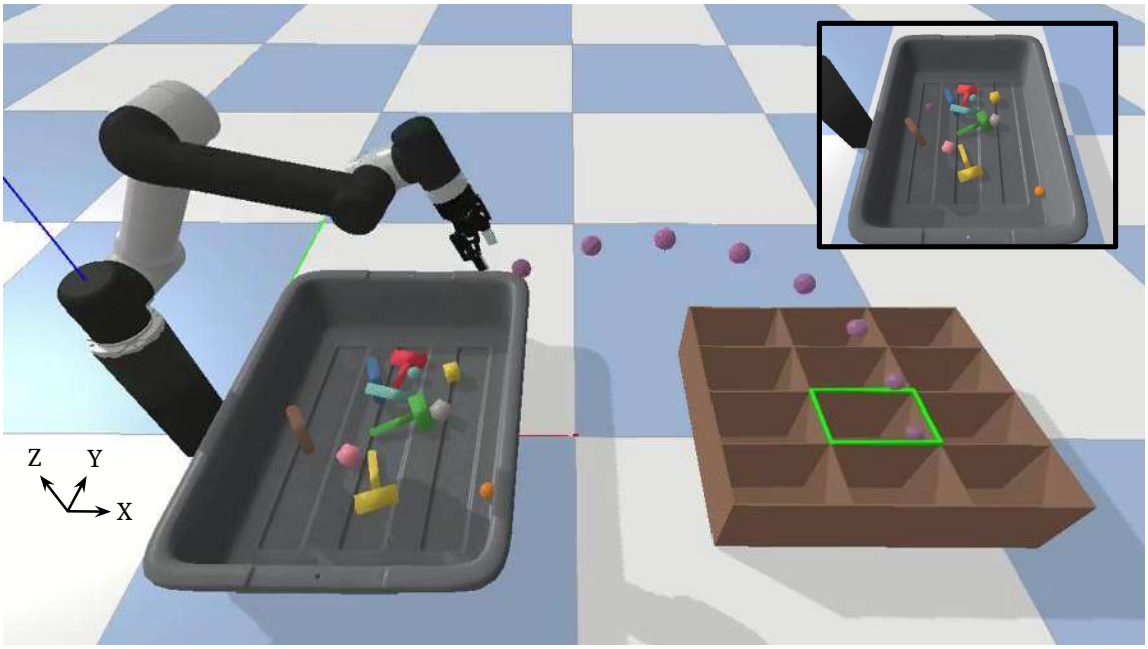


Figure 5.6: **Simulation environment** in PyBullet [19]. This snapshot illustrates the aerial motion trajectory of a purple ball being thrown into the target landing box highlighted in green. The top right image depicts the view captured from the simulated RGB-D camera before the ball was grasped and thrown.

Real-world setup. We use a UR5 arm with an RG2 gripper to pick and throw a collection of 80+ different toy blocks, fake fruit, decorative items, and office objects (see Fig. 5.5). For perception data, we capture 640×480 RGB-D images using a calibrated Intel RealSense D415 statically mounted overlooking the bin of objects from the side. The camera is localized with respect to the robot base using an automatic calibration procedure from [124]. A second RealSense D415 is mounted above the boxes looking downwards to track landing locations of thrown objects by measuring changes between images captured before and after executed throws.

5.6.2 Baseline Methods

Residual-physics denotes our approach described in Sec. 5.3. Since there are no comparable available algorithms that can learn joint grasping and throwing policies,

we compare our approach to three baselines based on variations of the proposed method:

Regression is a variant of our approach where the throwing network is trained to directly regress the final release velocity v, instead of the residual δ. Specifically, each pixel in the output Q_t of the throwing network holds a prediction of the final release velocity $\|v_{x,y}\|$ for the throwing primitive. The physics-based controller is removed from this baseline, but in order to ensure a fair comparison, we concatenate the visual features μ with the xy-distance d between the target landing location and release position (*i.e.*, $d = \|r_{x,y} - p_{t_{x,y}}\|$) before feeding into the grasping and throwing networks. Conceptually, this variant of our approach is forced to learn physics from scratch instead of bootstrapping on physics-based control.

Physics-only is also a variant of our approach where the throwing network is removed and completely replaced by velocity predictions made by the physics-based controller. In other words, this variant only learns grasping and uses physics for throwing (without learning a residual).

Regression-pretrained-on-physics is a version of ***Regression*** that is pre-trained on release velocity predictions $\hat{v}$ made by the physics-based controller described in Sec. 5.3.3. The shorthand name for this method is ***Regression-PoP***.

5.6.3 Baseline Comparisons

In simulated and real settings, we train our models via trial and error for 15,000 steps, then test each model for 1,000 steps and report their average grasping and throwing success rates.

Simulation results are reported in Table 5.1 and 5.2. Each column of the table represents a different set of test objects e.g., "Hammers" is a set of n hammers, "Seen" is a mixed set of objects seen during training, "Unseen" is a mixed set of objects not seen during training.

Table 5.1: Throwing Performance in Simulation (Mean %)

Method	Balls	Cubes	Rods	Hammers	Seen	Unseen
Regression	70.9	48.8	37.5	32.8	41.8	28.4
Regression-PoP	96.1	73.5	52.8	47.8	56.2	35.0
Physics-only	98.6	83.5	77.2	70.4	82.6	50.0
Residual-physics	**99.6**	**86.3**	**86.4**	**81.2**	**88.6**	**66.5**

Table 5.2: Grasping Performance in Simulation (Mean %)

Method	Balls	Cubes	Rods	Hammers	Seen	Unseen
Regression	99.4	99.2	89.0	87.8	95.6	69.4
Regression-PoP	99.2	98.0	89.8	87.0	96.4	70.6
Physics-only	99.4	99.2	87.6	85.2	96.6	64.0
Residual-physics	98.8	99.2	89.2	84.8	96.0	74.6

The throwing results in Table 5.1 indicate that learning residuals (Residual-physics) on top of a physics-based controller provides the most accurate throws. Physics-only performs competitively in simulation, where the environment is void of aerodynamics and unstable contact dynamics, but falls short of performance in comparison to Residual-physics – particularly for difficult objects like rods or hammers of which the grasping offsets from CoM can significantly change projectile trajectories. We also observe that regression pre-trained on physics (Regression-PoP) always consistently outperforms regression alone. On the other hand, the results in Table 5.2 show that grasping performance remains roughly the same across all methods. All policies experience moderately lower grasping and throwing success rates for unseen testing objects.

Fig. 5.7 plots the average throwing performance of all baseline methods over training steps on the hardest object set: hammers. Throwing performance is measured by throwing success rates over the last j = 1,000 attempts. Numbers reported at earlier training steps (*i.e.*, iteration $i < j$) in Fig. 5.7 are weighted by $\frac{i}{j}$. The plot

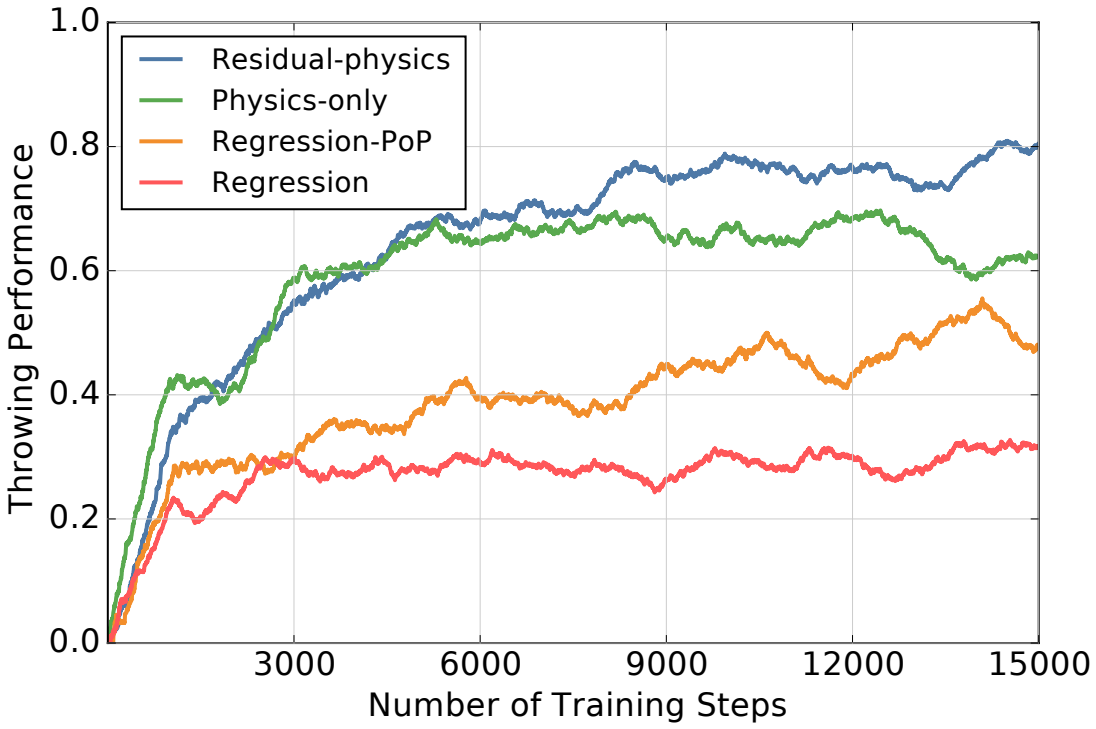


Figure 5.7: Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

shows that as soon as the performance of Physics-only begins to asymptote, Residual-physics starts to outperform Physics-only by learning residual throwing velocities that compensate for grasping offsets from the object CoM.

Real-world results are reported in Table 5.3 on seen and unseen object sets. The results show that Residual-physics continues to provide more accurate throws than the baseline methods. Most notably, in contrast to simulation, Physics-only does not perform as competitively to Residual-physics in the real-world. This is likely because the ballistic model used by Physics-only does not account for the unmodelled and uncertain contact- and aero-dynamics in the real world. Residual-physics can compensate for them in one of two ways: Either improving the model, or avoiding

regions of the model that are not predictable. This allows to maintain a throwing accuracy above 80% for both seen and unseen objects.

Interestingly, our system also seems to exceed the average performance of an untrained human. To measure human throwing performance, we asked 15 willing participants (average height: 174.0±8.3cm) to stand in place of the robot in the real-world setup and then grasp and throw 80 objects from the bin into the target boxes round-robin. Objects came from the collection of unseen test objects used in the robot experiments, and were kept consistent across runs. Participants were asked to pick-and-throw at whichever speed felt most comfortable (*i.e.*, we did not compare picking speeds).

Surprisingly, human performance was lower than we expected. The largest contributor to poor performance was fatigue – the accuracy of throws deteriorates over time, particularly after around the 20th object regardless of picking speed. The second largest contributor to performance was the physical height of the participant (taller performed better) – this may be due to differences in throwing distance (measured from grasp release to object landing locations, which is smaller for taller participants with longer arms) and the throwing strategies (taller participants more often preferred overhand throws to underhand ones). Other common throwing strategies included: 1) largely using tactile feedback to grasp objects in the bin so that visual field of view remains in focus on target boxes, 2) grasping objects with one hand and throwing with the other so that the throwing arm can make more repeatable movements, 3) and grouping objects by weight, then correspondingly changing to different grasping and throwing strategies. These additional strategies were interesting, but did not seem to strongly correlate with better performance.

Table 5.3: Grasping and Throwing Performance in Real (Mean %)

	Grasping		Throwing	
Method	Seen	Unseen	Seen	Unseen
Human-baseline	–	–	–	80.1±10.8
Regression-PoP	83.4	75.6	54.2	52.0
Physics-only	85.7	76.4	61.3	58.5
Residual-physics	86.9	73.2	**84.7**	**82.3**

Table 5.4: Picking Speed vs State-of-the-Art Systems

System	Mean Picks Per Hour (MPPH)
Cartman [80]	120
Dex-Net 2.0 [67]	250
FC-GQ-CNN [101]	296
Dex-Net 4.0 [69]	312
TossingBot (w/ Placing)	432
TossingBot (w/ Throwing)	**514**

5.6.4 Pick-and-Place Efficiency

Throwing enables our system (TossingBot) to achieve picking speeds of 514 mean picks per hour (MPPH), where 1 pick = successful grasp and accurate throw. Specifically, the system performs 608 grasps per hour, and achieves 84.7% throwing accuracy, yielding 514 MPPH. In Table 5.4, we compare against other state-of-the-art picking systems found in literature: Cartman [80], Dex-Net 2.0 [67], FC-GQ-CNN [101], Dex-Net 4.0 [69], and a variant of TossingBot that places objects into a box 0.8m away from the bin without throwing. This is not a like-for-like comparison, since throwing is only practical for certain types of objects (*e.g.* not eggs), and placing is only practical for limited distance ranges. Yet, the results suggest that throwing may be useful to improve the overall MPPH in some applications.

In addition to throwing, there are 3 other aspects that enable our system's picking speeds: 1) fast algorithmic run-time speeds (220ms for inference), 2) real-time

TSDF fusion [20, 82, 122] of RGB-D data, which enables us to capture and aggregate observed 3D data of the scene simultaneously as the robot moves around within the field-of-view, and 3) online training and inference in parallel to robot actions:

Algorithm 1 System Pipeline

Initialize *robot*.
Initialize policy with model f.
Initialize replay *buffer*.
while step $i < N$ and not *terminate* **do**
    I^i = *robot*.CaptureState()
    p^i = *robot*.SelectTarget()
    ϕ_g^i, ϕ_t^i = f.Inference(I^i,p^i)
    while *robot*.is_grasping **do**
        f.ExperienceReplay(*buffer*)
    y^{i-1} = *robot*.CheckGraspSuccess()
    robot.ExecuteThrow(ϕ_t^{i-1},p^{i-1}) ▷ asynchronous
    while *robot*.is_throwing **do**
        f.ExperienceReplay(*buffer*)
    robot.ExecuteGrasp(ϕ_g^i) ▷ asynchronous
    $\bar{p}^{i-1}$ = *robot*.TrackLanding()
    buffer.SaveData(I^{i-1},p^{i-1},ϕ_g^{i-1},ϕ_t^{i-1},y^{i-1},$\bar{p}^{i-1}$)
    $i = i + 1$

5.6.5 Learning Stable Grasps for Throwing

We next investigate the importance of supervising grasps with the accuracy of throws. To this end, we train two variants of Residual-physics: 1) grasping network supervised by accuracy of throws (*i.e.*, grasp success = object landed on target), and 2) grasping network supervised by checking grasp width after grasping primitive (*i.e.*, grasp success = object in gripper). We plot their grasping and throwing success rates over training steps in Fig. 5.8 on the hammer object set.

The results indicate that throwing performance significantly improves when grasping is supervised by the accuracy of throws. This not only suggests that the grasping policies are capable of learning to execute the subset of grasps that lead to more predictable throws, but also indirectly that throwing accuracy is strongly influenced by

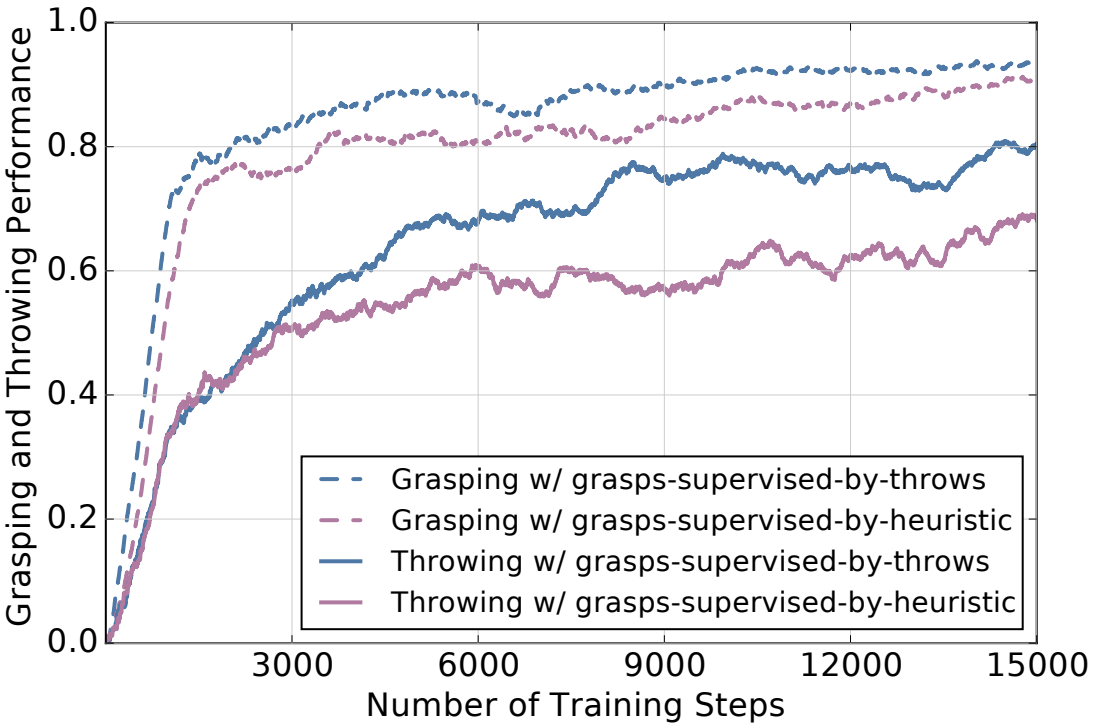


Figure 5.8: Both grasping and throwing success rates of Residual-physics policies improve when grasps are supervised by the accuracy of throws (blue), versus when grasps are supervised by a heuristic that checks grasp width (purple).

the quality of grasps. Interestingly, the results also show that grasping performance slightly increases when supervised by the accuracy of throws.

We also investigate the quality of learned grasps by visualizing 2D histograms of successful grasps, mapped directly on the hammer object in Fig. 5.9. To create this visualization from simulation, we record each grasping position by saving the 3D location (with respect to the hammer) of the middle point between gripper fingertips after each successful grasp. We then project the grasping positions recorded over 15,000 training steps onto a 2D histogram, where darker regions indicate more grasps. The silhouette of the hammer is outlined in black, with a green dot indicating its CoM. We illustrate the grasp histograms of three policies: Residual-physics with

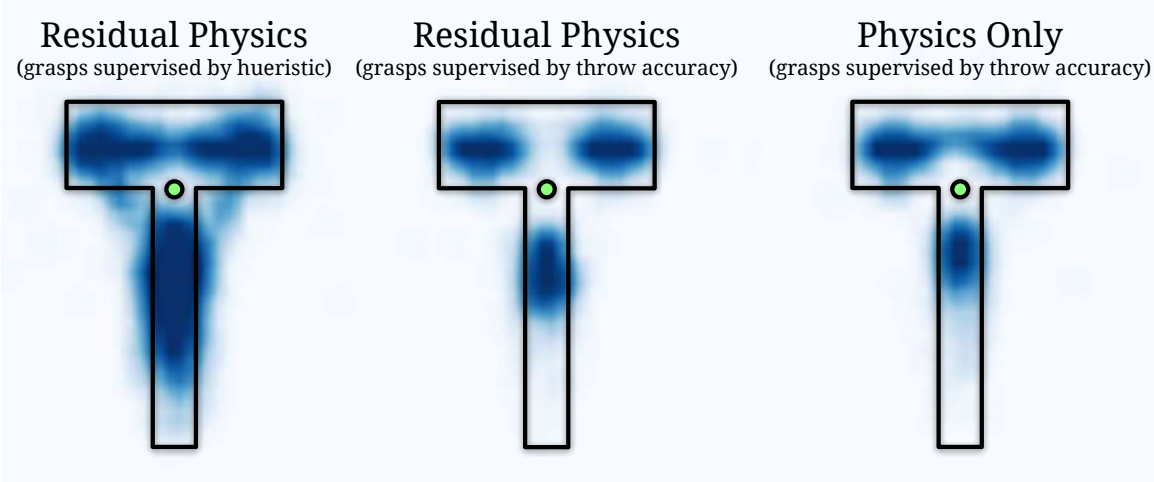


Figure 5.9: Projected 2D histograms of successful grasping positions on hammers in simulation: show that 1) leveraging accuracy of throws as supervision enables the grasping policy to learn a more restricted but stable set of grasps, while 2) learning throwing in general helps to relax this constraint.

grasping supervised by heuristic that checks grasp width after grasping primitive (left), Residual-physics with grasping supervised by accuracy of throws (middle), and Physics-only with grasping supervised by accuracy of throws (right).

The differences between left and middle histograms indicate that leveraging accurate throws as a supervisory signal encourages the grasping policy to learn a more restricted but stable set of grasps: slightly further from the CoM to avoid unintentional collisions between the fingers and rest of the object at the moment of release, but also further from the ends of the handle to avoid less predictable throws. The differences between middle and right histograms show that when using only ballistics for the throwing module (*i.e.*, without learning throwing), the grasping policy seems to further optimize for grasps that closer to the CoM. This leads to a more restricted set of grasps in contrast to Residual-physics, where the throwing can learn to compensate respectively.

We also provide additional 2D grasp histogram visualizations in Fig. 5.10 for all simulation objects. Across all policies, the histograms visualizing grasps which lead to successful throws (columns 2, 5, 8) share large overlaps with the grasps that lead

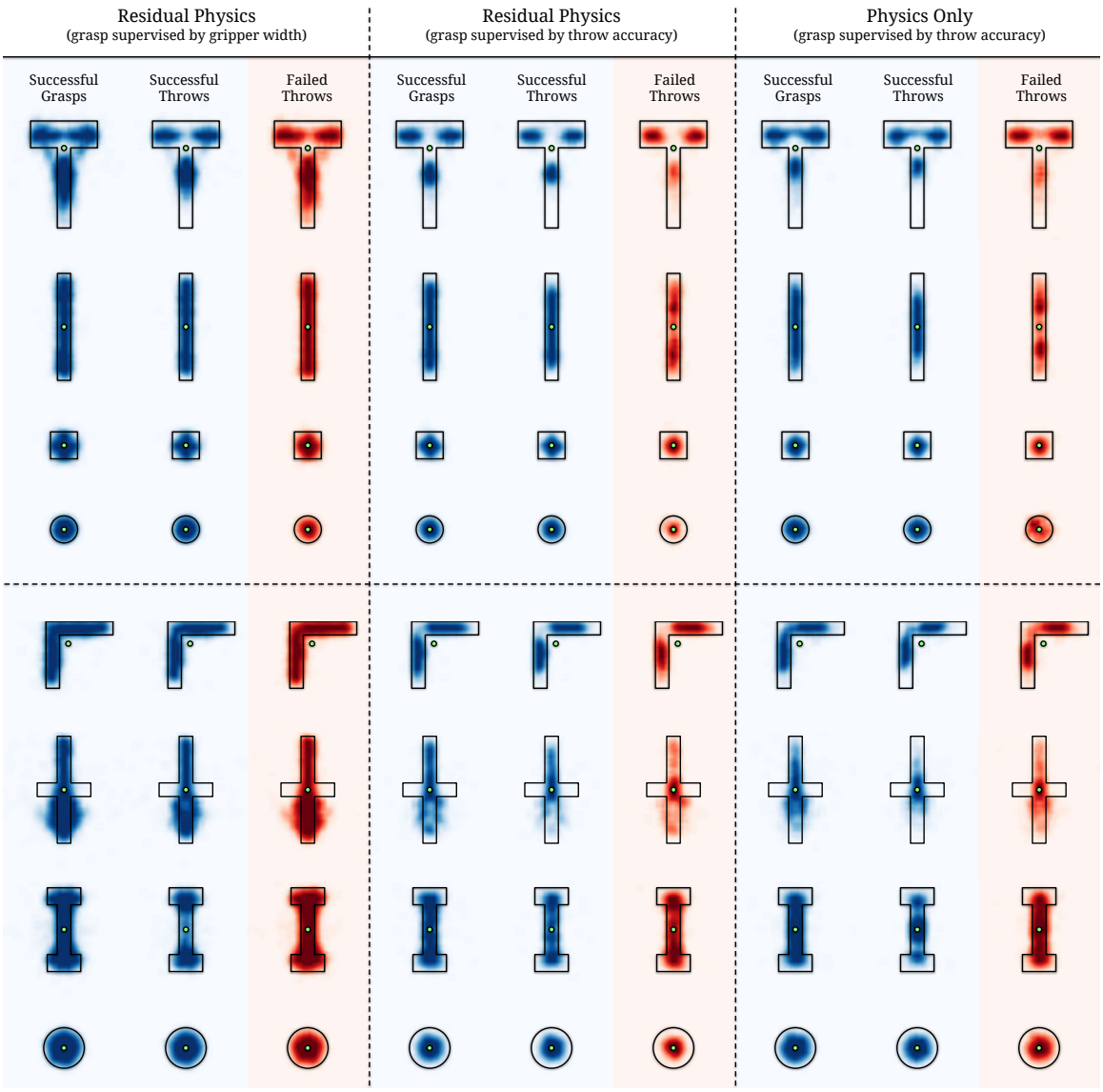


Figure 5.10: **Additional grasping histograms** of all simulation objects. Histograms are generated for successful grasps, grasps that lead to successful throws, and grasps that lead to failed throws – recorded over 15,000 training steps. Darker regions indicate more grasps. The silhouette of each object is outlined in black, with a green dot indicating its CoM.

to failed throws (red columns 3, 6, 9). This suggests grasping and throwing might have been learned simultaneously, rather than one after the other – likely because the way the robot throws is not trivially conditioned on how it grasps.

5.6.6 Generalizing to New Target Locations

To explore how the trained policies generalize to new target locations, we displace the locations of the boxes in both the horizontal plane from where they were during training, such that there is no overlap between training and testing locations. For this experiment, we set in simulation 12 training boxes and 12 testing boxes; while in real settings, we set 4 training and 4 testing boxes (limited by physical setup). We record each model's throwing performance on seen objects over these new box locations across 1,000 testing steps in Table 5.5.

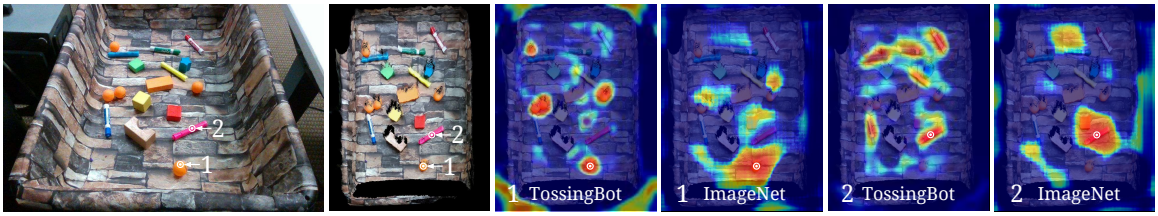
Table 5.5: Throwing to Unseen Locations (Mean %)

Method	Simulation	Real
Regression-PoP	26.5	32.7
Physics-only	79.6	62.2
Residual-physics	**87.2**	**83.9**

We observe that in both simulated and real experiments, Residual-physics significantly outperforms the regression baseline. The performance margin in this scenario illustrates how Residual-physics leverages the generalization of the ballistic equations to adapt to new target locations.

5.6.7 Emerging Object Semantics from Interaction

In this section, we explore the deep features being learned by the neural network f – i.e., "What does TossingBot learn from grasping and throwing arbitrary objects?" and "Do they convey any meaningful structure or representation?" To this end, we place several training objects in the bin (well-isolated from each other for visualization purposes), capture RGB-D images to construct heightmap I, and feed it through the network f (already trained for 15,000 steps from the real experiments). Training objects include marker pens, ping pong balls, and wooden toy blocks of different shapes (see Fig. 5.11). We then extract the intermediate spatial feature representation of



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

Figure 5.11: **Emerging semantics from interaction.** Visualizing pixel-wise deep features μ learned by TossingBot (c,e) overlaid on the input heightmap image (b) generated from an RGB-D side-view (a) of a bin of objects. (c) shows a heatmap of pixel-wise feature distances (hotter = smaller distance) from the feature vector of a query pixel on a ping pong ball (labeled 1). Likewise, (e) shows a heatmap of pixel-wise feature distances from the feature vector of a query pixel on a pink marker pen (labeled 2). These visualizations show that TossingBot learns features that distinguish object categories from each other without explicit supervision (*i.e.*, only task-level grasping and throwing). For reference, the same visualization technique is used on deep features generated by a ResNet-18 pre-trained on ImageNet (d,f).

the network μ (described in Sec. 5.3.1), which essentially holds a 512-dimensional feature vector for each pixel of the heightmap I (after $4\times$ upsampling to the same resolution). We then extract the feature vector from a query pixel belonging to one of the ping pong balls in the bin, and visualize its distance to all other pixel-wise features as a heatmap in Fig. 5.11a (where hotter regions indicate smaller distances), overlaid on the original input heightmap. More specifically, we rank each pixel based on its ℓ_2 feature distance to the query pixel feature, then colorize it based on its rank (*i.e.*, higher rank = closer feature distance = hotter color).

Interestingly, the visualization immediately localizes all other ping pong balls in the scene – presumably because they share similar deep features. It is also interesting to note that the orange wooden block, despite sharing a similar color, does not get picked up by the query. Similarly, Fig. 5.11b illustrates the feature distances between a query pixel on a pink marker pen to all other pixels of the scene. The visualization immediately localizes all other marker pens, which share similar shape and mass, but do not necessarily share color textures.

These interesting results suggest that the deep network is learning to bias the features (*i.e.*, learning a prior) based on the objects' shapes more so than their visual textures. The network likely learns that geometric cues are more useful for learning grasping and throwing policies – *i.e.*, provides more information related to grasping interactions and projectile behaviors. In addition to shape, one could also argue that the learned deep features reflect the second-order (beyond visual or geometric) physical attributes of objects which influence their aerial behaviors when thrown. This perspective is also plausible, since the throwing policies are effectively learning to compensate for these physical attributes respectively. For comparison, these visualizations generated by features from TossingBot are more informative in this setting than those generated using deep features from a 18-layer ResNet pre-trained on ImageNet (also shown in Fig. 5.11).

These emerging features were learned implicitly from scratch without any explicit supervision beyond task-level grasping and throwing. Yet, they seem to be sufficient for enabling the system to distinguish between ping pong balls and markers. As such, this experiment speaks out to a broader concept related to machine vision: how should robots learn the semantics of the visual world? From the perspective of classic computer vision, semantics are often pre-defined using human-fabricated image datasets and manually constructed class categories (*i.e.*, this is a "hammer", and this is a "pen"). However, our experiment suggests that it is possible to implicitly learn such object-level semantics from physical interactions alone (as long as they matter for the task at hand). The more complex these interactions, the higher the resolution of the semantics. Towards more generally intelligent robots – perhaps it is sufficient for them to develop their own notion of semantics through interaction, without requiring any human intervention.

Chapter 6

Future Work

Interest in robust and versatile robotic manipulation is almost as old as robotics. Robot grasping and object recognition have been two of the main drivers of robotic research. Yet, the reality in industry is that most automated picking systems are restricted to known objects, in controlled configurations, with specialized hardware. These systems typically use classic manipulation algorithms which typically assume full knowledge of the objects beforehand (*e.g.* class categories, stable grasps, 3D object CAD models). In Chapter 2, we show that these algorithms can be improved with deep learning, however, they remain limited because they suffer from propagating errors, and do not have the ability to generalize to novel objects unseen in training. This makes them difficult to scale up for application like warehouse logistics (in which thousands of new products can be added to the warehouse each day) or clearing debris in disaster response scenarios (where the shape of debris can be unpredictable).

To address these limitations, in this thesis we introduce a formulation for learning visual affordances with convolutional networks. The idea is to use classic controllers to design motion primitives, then use FCNs to map from visual observations (*e.g.* images) to the perceived affordances (*e.g.* confidence scores or action-values) of the primitives for every pixel in the input. Visual affordances is a form of end-to-end

learning that leverages spatial equivariance as a means to be sample efficient. In Chapters 3 - 5, we show that visual affordances can enable robotic systems to learn a variety of complex vision-based manipulation skills (*e.g.* pushing, grasping, throwing) that generalize to novel objects while using orders of magnitude less data.

Learning visual affordances for robotic manipulation is a formulation that is still far from mature, with limitations that suggest directions for future work. Here we discuss some of these directions:

More Complex Manipulations. In Chapter 3, we present a system that uses visual affordances to infer grasps for four different primitive actions (with suction and parallel-jaw grasping) directly in the RGB-D image. This proved faster and more reliable than doing object segmentation and state estimation prior to grasp planning in Chapter 2. We observed that the model learns the visual features that make a region of an image graspable or suctionable. It also seems to learn more complex rules, *e.g.* that tags are often easier to suction that the object itself, or that the center of a long object is preferable than its ends. It would be interesting to explore the limits of the approach. For example learning visual affordances for more complex behaviors, e.g., scooping an object against a wall, sliding an object across a surface, or twisting a doorknob and pushing forward to open a door – skills which require a more global understanding of the geometry and physical properties of the environment. In the same vein, motion primitives are defined with parameters specified on a regular discrete grid (pixel maps), which provides learning efficiency with deep convolutional networks, but inherently limits expressiveness – it would be interesting to explore other (possibly continuous) action parameterizations that allow more expressive motions (without excessively inducing sample complexity), including more dynamic pushes, parallel rather than sequential combinations of pushing and grasping, and the use of more varied contact surfaces of the robot.

Multitask Learning. In this thesis, we show that visual affordances can be formulated for pushing, suction, parallel-jaw grasping, and tossing. By expanding the number of manipulations that can be defined under this unified formulation and network architecture, it would be interesting to explore the question: how much shared knowledge between tasks and manipulation skills can be transferred to each other? In the advent of deep learning for computer vision, a common practice for improving the sample efficiency and generalization of vision models is by initializing them with the pre-trained weights and parameters of other vision models (with the same network architecture) trained on different tasks. It is possible that the same could be done for manipulation skills learned with our formulation of visual affordances to transfer pre-trained latent feature representations that may accelerate the learning process or converge to better performance for new manipulation tasks.

Closed-loop Manipulation. Most existing manipulation approaches, whether model-based or data-driven are for the most part, based on open-loop executions of planned primitives (*e.g.* for grasping, pushing, etc.). Our formulation is no different. The system decides what to do from visual data, and executes it almost blindly, except with simple feedback to enable guarded moves like move until contact. Indeed, the most common failure modes are when small errors in the estimated affordances lead to fingers landing on top of an object rather than on the sides, or lead to a deficient suction latch, or lead to a grasp that is only marginally stable and likely to fail when the robot lifts the object. It is unlikely that manipulation error rates can be trimmed to industrial grade without the explicit use of closed-loop feedback during the operation. Exploring how visual affordances can be extended to be closed-loop is an interesting topic for future work.

Multi-modal Manipulation. In this thesis, we present a formulation for learning visual affordances from visual data (RGB-D images), which is just one member of a larger family of perceptual modalities, *e.g.* force-torque, tactile, radar/sonar, lidar,

sound, etc. Exploring additional sensing channels and making effective use of them while leveraging spatial equivariance may enable the system to better react to new scenarios and better adapt its learned manipulation strategies.

Bibliography

[1] Project webpage: Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. URL `http://apc.cs.princeton.edu/`.

[2] Project webpage: Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. URL `http://arc.cs.princeton.edu/`.

[3] Project webpage: Tossingbot: Learning to throw arbitrary objects with residual physics. URL `https://tossingbot.cs.princeton.edu/`.

[4] Project webpage: Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. URL `http://vpg.cs.princeton.edu`.

[5] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2006.

[6] Eric W Aboaf, Christopher G Atkeson, and David J Reinkensmeyer. Task-level robot learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 1988.

[7] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[9] Ruzena Bajcsy and Mario Campos. Active and exploratory perception. *CVGIP: Image Understanding*, 56(1):31–40, 1992.

[10] Maria Bauza and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[11] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 2015.

[12] Paul J Besl and Neil D McKay. A method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, 1992.

[13] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact. *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[14] Abdeslam Boularias, James Andrew Bagnell, and Anthony Stentz. Learning to manipulate unknown objects in clutter by reinforcement. In *AAAI Conference on Artificial Intelligence*, 2015.

[15] Nikhil Chaval-Dafle, Alberto Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[16] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. Active vision in robotic systems: A survey of recent developments. *The International Journal of Robotics Research (IJRR)*, 2011.

[17] Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research (IJRR)*, 2011.

[18] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano, and Peter R Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, 2016.

[19] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2018.

[20] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Special Interest Group on Computer GRAPHics and Interactive Techniques (SIGGRAPH)*, 1996.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[22] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University Robotics Institute, 2010.

[23] A. Dias, C. Brites, J. Ascenso, and F. Pereira. Sift-based homographies for efficient multiview distributed visual sensing. *IEEE Sensors Journal*, 15(5): 2643–2656, May 2015. ISSN 1530437X. doi: 10.1109/JSEN.2014.2355914.

[24] Mehmet R Dogar and Siddhartha S Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Autonomous Robots*, 2012.

[25] M. Freese E. Rohmer, S. P. N. Singh. V-rep: a versatile and scalable robot simulation framework. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[26] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martın-Martın, Arne Sieverling, Vincent Wall, and Oliver Brock. Lessons from the amazon picking challenge: Four aspects of building robotic systems. In *Robotics: Science and Systems (RSS)*, 2016.

[27] Nima Fazeli, Samuel Zapolsky, Evan Drumwright, and Alberto Rodriguez. Learning data-efficient rigid-body contact models: Case study of planar impact. *Conference on Robot Learning (CoRL)*, 2017.

[28] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[29] Yizhi Gai, Yukinori Kobayashi, Yohei Hoshino, and Takanori Emaru. Motion control of a ball throwing robot with a flexible robotic arm. *World Academy of Science, Engineering and Technology (WASET)*, 2013.

[30] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[31] James J Gibson. *The ecological approach to visual perception: classic edition*. Psychology Press, 2014.

[32] Ross Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[34] Corey Goldfeder, Matei Ciocarlie, Hao Dang, and Peter K Allen. The columbia grasp database. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[35] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. Planar sliding with dry friction part 1. limit surface and moment function. *Wear*, 1991.

[36] Marcus Gualtieri, Andreas ten Pas, Kate Saenko, and Robert Platt. High precision grasp pose detection in dense clutter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[37] Marcus Gualtieri, Andreas ten Pas, Kate Saenko, and Robert Platt. High precision grasp pose detection in dense clutter. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[38] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision (ECCV)*, 2014.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[40] Carlos Hernandez, Mukunda Bharatheesha, Wilson Ko, GaInternational Society for Engineers, Hans Researchers, Jethro Tan, Kanter van Deurzen, Maarten de Vries, Bas Van Mil, Jeff Van Egmond, Ruben Burger, Mihai Morariu, Jihong Ju, Xander Gerrmann, Ronald Ensing, Jan Van Frankenhuyzen, and Martijn Wisse. Team delft's robot winner of the amazon picking challenge 2016. *Robot World Cup*, 2016.

[41] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[42] Elad Hoffer, Itay Hubara, and Nir Ailon. Deep unsupervised learning through spatial contrasting. *arXiv preprint arXiv:1610.00243*, 2016.

[43] François Robert Hogan and Alberto Rodriguez. Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[44] Thomas E Horton, Arpan Chakraborty, and Robert St Amant. Affordances for robots: a brief survey. *AVANT. Pismo Awangardy Filozoficzno-Naukowej*, 2012.

[45] Jwu-Sheng Hu, Ming-Chih Chien, Yung-Jung Chang, Shyh-Haur Su, and Chen-Yu Kai. A ball-throwing robot with visual feedback. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[46] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[47] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[48] Clavera Ignasi, David Held, and P Abbeel. Policy transfer via modularity. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[49] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[50] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[51] Dinesh Jayaraman and Kristen Grauman. Look-ahead before you leap: End-to-end active recognition by forecasting the effect of motion. *European Conference on Computer Vision (ECCV)*, 2016.

[52] Duofan Jiang, Hesheng Wang, Weidong Chen, and Ruimin Wu. A novel occlusion-free active recognition algorithm for objects in clutter. *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2016.

[53] Yifeng Jiang, Jiazheng Sun, and C Karen Liu. Data-augmented contact model for rigid body simulation. *arXiv preprint arXiv:1803.04019*, 2018.

[54] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[55] Rico Jonschkowski, Clemens Eppner, Sebastian Höfer, Roberto Martín-Martín, and Oliver Brock. Probabilistic multi-class segmentation for the amazon picking challenge. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[56] Alina Kloss, Stefan Schaal, and Jeannette Bohg. Combining learned and analytical models for predicting action effects. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[57] Jens Kober, Erhan Öztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2011.

[58] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. *International Conference on Machine Learning Workshop (ICML)*, 2015.

[59] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research (IJRR)*, 2015.

[60] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, 2016.

[61] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. *International Society for Engineers and Researchers (ISER)*, 2016.

[62] Ce Liu, Jenny Yuen, and Antonio Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2011.

[63] Ming-Yu Liu, Oncel Tuzel, Ashok Veeraraghavan, Yuichi Taguchi, Tim K Marks, and Rama Chellappa. Fast object localization and pose estimation in heavy clutter for robotic bin picking. *The International Journal of Robotics Research (IJRR)*, 2012.

[64] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[65] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

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

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

[68] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-net 3.0: Computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[69] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 2019.

[70] Matthew T Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research (IJRR)*, 1986.

[71] Matthew T Mason and Kevin M Lynch. Dynamic manipulation. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1993.

[72] Eiichi Matsumoto, Masaki Saito, Ayaka Kume, and Jethro Tan. End-to-end learning of object grasp poses in the amazon robotics challenge. *Warehouse Picking Automation Workshop (WPAW)*, 2016.

[73] Tekin Meriçli, Manuela Veloso, and H Levent Akın. Push-manipulation of complex passive mobile objects using experimentally acquired motion models. *Autonomous Robots*, 2015.

[74] Anton Milan, Trung Pham, K Vijay, Douglas Morrison, Adam W Tow, L Liu, J Erskine, R Grinover, A Gurman, T Hunn, et al. Semantic segmentation from limited training data. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[75] A.T. Miller, Steffen Knoop, H.I. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. *IEEE International Conference on Robotics and Automation*, 2003.

[76] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[77] Antonio Morales, Eris Chinellato, Andrew H Fagg, and Angel P Del Pobil. Using experience for assessing grasp reliability. *International Conference on Humanoid Robots (Humanoids)*, 2004.

[78] Wataru Mori, Jun Ueda, and Tsukasa Ogasawara. 1-dof dynamic pitching robot that independently controls velocity, angular velocity, and direction of a ball: Contact models and motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[79] D Morrison, AW Tow, M McTaggart, R Smith, N Kelly-Boxall, S Wade-McCue, J Erskine, R Grinover, A Gurman, T Hunn, et al. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[80] Douglas Morrison, Adam W Tow, M McTaggart, R Smith, N Kelly-Boxall, S Wade-McCue, J Erskine, R Grinover, A Gurman, T Hunn, et al. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[81] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

[82] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[83] Matthias Nieuwenhuisen, David Droeschel, Dirk Holz, Jörg Stückler, Alexander Berner, Jun Li, Reinhard Klein, and Sven Behnke. Mobile bin picking with an anthropomorphic service robot. *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[84] Damir Omrčen, Christian Böge, Tamim Asfour, Aleš Ude, and Rüdiger Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2009.

[85] Peter Pastor, Mrinal Kalakrishnan, Jonathan Binney, Jonathan Kelly, Ludovic Righetti, Gaurav Sukhatme, and Stefan Schaal. Learning task error models for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[86] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[87] L. Pinto, J. Davidson, and A. Gupta. Supervision via competition: Robot adversaries for learning tasks. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[88] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[89] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[90] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, et al. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[91] Domenico Prattichizzo and Jeffrey C Trinkle. Grasping. In *Springer Handbook of Robotics*. 2008.

[92] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[93] Joseph Redmon and Anelia Angelova. Real-time grasp detection using convolutional neural networks. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[94] Joseph Redmon and Anelia Angelova. Real-time grasp detection using convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[95] Colin Rennie, Rahul Shome, Kostas E Bekris, and Alberto F De Souza. A dataset for improved rgbd-based object detection and pose estimation for warehouse pick-and-place. *Robotics and Automation Letters (RA-L)*, 2016.

[96] Alberto Rodriguez, Matthew T Mason, and Steve Ferry. From caging to grasping. *The International Journal of Robotics Research (IJRR)*, 2012.

[97] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3dim*, volume 1, pages 145–152, 2001.

[98] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.

[99] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. *European Conference on Computer Vision (ECCV)*, 2010.

[100] Marcos Salganicoff, Giorgio Metta, Andrea Oddera, and Giulio Sandini. *A vision-based learning method for pushing manipulation*. 1993.

[101] Vishal Satish, Jeffrey Mahler, and Ken Goldberg. On-policy dataset synthesis for learning deep robot grasping policies based on fully-convolutional grasp quality neural networks. *IEEE Robotics and Automation Letters (RA-L)*, 2018.

[102] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research (IJRR)*, 2008.

[103] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations (ICLR)*, 2016.

[104] Max Schwarz, Anton Milan, Christian Lenz, Aura Munoz, Arul Selvam Periyasamy, Michael Schreiber, Sebastian Schüller, and Sven Behnke. Nimbro picking: Versatile part handling for warehouse automation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[105] Taku Senoo, Akio Namiki, and Masatoshi Ishikawa. High-speed throwing motion based on kinetic chain approach. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[106] Abhinav Shrivastava, Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Data-driven visual similarity for cross-domain image matching. *ACM Transactions on Graphics (TOG)*, 2011.

[107] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. *European Conference on Computer Vision (ECCV)*, 2012.

[108] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[109] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[110] Arjun Singh, James Sha, Karthik S Narayan, Tudor Achim, and Pieter Abbeel. Bigbird: A large-scale 3d database of object instances. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[111] Martin Styner, Christian Brechbuhler, G Szckely, and Guido Gerig. Parametric estimate of intensity inhomogeneities applied to mri. *IEEE Transactions on Medical Imaging*, 2000.

[112] Orion Taylor and Alberto Rodriguez. Optimal shape and motion planning for dynamic planar manipulation. *Autonomous Robots*, 2019.

[113] Andreas ten Pas and Robert Platt. Using geometry to detect grasp poses in 3d point clouds. *International Symposium on Robotics Research (ISRR)*, 2015.

[114] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

[115] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning (ICML)*, 2015.

[116] Jonathan Weisz and Peter K Allen. Pose error robust grasping from contact wrench space metrics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[117] Jay M Wong, Vincent Kee, Tiffany Le, Syler Wagner, Gian-Luca Mariottini, Abraham Schneider, Lei Hamilton, Rahul Chipalkatty, Mitchell Hebert, David M.S. Johnson, Jimmy Wu, Bolei Zhou, and Antonio Torralba. Segicp: Integrated deep semantic segmentation and pose estimation. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[118] Kanzhi Wu, Ravindra Ranasinghe, and Gamini Dissanayake. Active recognition and pose estimation of household objects in clutter. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[119] Jianxiong Xiao, Shuran Song, Daniel Suo, and Fisher Yu. Marvin: A minimalist GPU-only N-dimensional ConvNet framework. URL `http://marvin.is`.

[120] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[121] Kuan-Ting Yu, Nima Fazeli, Nikhil Chavan-Dafle, Orion Taylor, Elliott Donlon, Guillermo Diaz Lankenau, and Alberto Rodriguez. A summary of team mit's approach to the amazon picking challenge 2015. *arXiv preprint arXiv:1604.03639*, 2016.

[122] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[123] 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 International Conference on Robotics and Automation (ICRA)*, 2017.

[124] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[125] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois Robert Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, Nima Fazeli, Ferran Alet, Nikhil Chavan Dafle, Rachel Holladay, Isabella Morona, Prem Qu Nair, Druck Green, Ian Taylor, Weber Liu, Thomas Funkhouser, and Alberto Rodriguez. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research (IJRR)*, 2018.

[126] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *Robotics: Science and Systems (RSS)*, 2019.

[127] Hao Zhang, Pinxin Long, Dandan Zhou, Zhongfeng Qian, Zheng Wang, Weiwei Wan, Dinesh Manocha, Chonhyon Park, Tommy Hu, Chao Cao, et al. Dorapicker: An autonomous picking system for general objects. In *IEEE International Conference on Automation Science and Engineering (CASE)*, 2016.

[128] Jiaji Zhou, Robert Paolini, J Andrew Bagnell, and Matthew T Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.