Robot Manipulation with Learned Representations

by

Lucas Manuelli

A.B., Princeton University (2012)
S.M. Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Department of Electrical Engineering and Computer Science
August 24, 2020

Certified by .....
Russ Tedrake
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science,
Chair, Department Committee on Graduate Students

Robot Manipulation with Learned Representations

by

Lucas Manuelli

Submitted to the Department of Electrical Engineering and Computer Science
on August 24, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We would like to have robots which can perform useful manipulation tasks in real-world environments. This requires robots that can perceive the world with both precision and semantic understanding, methods for communicating desired tasks to these systems, and closed loop visual feedback controllers for robustly executing manipulation tasks. This is hard to achieve with previous methods: prior work hasn't enabled robots to densely understand the visual world with sufficient precision to perform robotic manipulation or endowed them with the semantic understanding needed to perform tasks with novel objects. This limitation arises partly from the object representations that have been used, the challenge in extracting these representations from the available sensor data in real-world settings, and the manner in which tasks have been specified. This thesis presents a family of approaches that leverage self-supervision, both in the visual domain and for learning physical dynamics, to enable robots to perform manipulation tasks. Specifically we (i) develop a pipeline to efficiently annotate visual data in cluttered and multi-object environments (ii) demonstrate the novel application of dense visual object descriptors to robotic manipulation and provide a fully self-supervised robot system to acquire them (iii) introduce the concept of category-level manipulation tasks and develop a novel object representation based on semantic 3D keypoints along with a task specification that uses these keypoints to define the task for all objects of a category, including novel instances, (iv) utilize our dense visual object descriptors to quickly learn new manipulation skills through imitation and (v) use our visual object representations to learn data-driven models that can be used to perform closed loop feedback control in manipulation tasks.

Thesis Supervisor: Russ Tedrake
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

There are many people that I would like to thank for helping me along the road to earning a PhD.

First and foremost I would like to thank my advisor Russ Tedrake. Russ took a chance on me when I was still an cconomics PhD student and gave me a wonderful opportunity to join the MIT Darpa Robotics Challenge team, and later the Robot Locomotion Group. Russ has an incredible mix of detailed technical knowledge and interest, along with the ability to layout a long-term vision and push his students to tackle important challenges in the field. Russ has been an incredibly kind and encouraging mentor to me through the PhD and I am sure we will keep in touch in the years to come.

Next I want to thank my committee members, Professors Alberto Rodriguez and Phil Isola for giving great feedback on the thesis and being very supportive throughout the process. In particular Alberto has been a great mentor to me throughout my years at MIT, even before he was part of my committee.

I also want to thank a host of other MIT faculty that I have interacted with through the years. These interactions are a big part of what made MIT such a special place to do a PhD. Specifically I would like to thank Tomás Lozano-Peréz, Leslie Kaelbling, Nicholas Roy, Neville Hogan, John Leonard, Rob Townsend and Ivan Werning.

Being on the MIT Darpa Robotics Challenge team was my first exposure to practical field robotics and served as an amazing learning experience that profoundly shaped my research interests and style throughout the PhD. I want to my fellow DRC teammates, Matt Antone, Andrés Valenzuela, Bryt Bradley, Claudia Perez D'Arpino, Gregory Izatt, Hongkai Dai, John Carter, Robin Deits , Steve Proulx, Toby Schneider, Twan Koolen, Scott Kuindersma, Pat Marion and Maurice Fallon. I especially want to thank Pat Marion and Andres Valenzuela for spending countless hours explaining the finer points of robotics to me in the cramped confines of our office in N9.

I want to thank my co-author and labmate Pete Florence. Pete has been a great co-author and friend throughout the PhD. The late nights spent in 32-380 hacking on

the robots were some of the funnest times of the PhD. I learned a lot from Pete and being able to share the PhD journey with someone that I also call a friend definitely made the experience much more enjoyable.

I also want to thank the members of the Robot Locomotion Group that I have had the pleasure of interacting with over the years for creating a stimulating research environment: Michael Posa, Frank Permenter, Ani Mujumdar, Tobia Marcucci, Jack Umenberger, Mark Petersen, Shen Shen, Terry Suh, Robin Deits, Twan Koolen, Benoit Landry, Vincent Tjeng, Yunzhu Li, Wei Gao.

I want to thank my family, Mom, Dad and Bianca for always encouraging and believing in me, even through periods of difficulty and uncertainty. I especially want to acknowledge my parents for spending so much time with me growing up and working hard to give me every opportunity to succeed. They have really shaped who I have become, both as a scholar and more importantly as a person.

Finally I want to thank my partner Katie for all of her support. You remind me that there is a life outside of research and all the great adventures we have had throughout the years have helped me keep a great balance in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the goals of robotics, and robotic manipulation specifically, is a general purpose robot, such as Baymax or C-3PO, that can perform useful tasks. Such a goal is still a work in progress, but in recent years significant advances have been made in related fields. AlphaGo [123], a reinforcement learning system trained entirely from self-play, was able to defeat Lee Sedong, the strongest human player at the time. In follow on work [122] this approach was generalized and the same algorithmic approach was used to master not only Go, but also Chess and Shogi. Meanwhile the world of computer vision was revolutionized in 2012 with the arrival of AlexNet [57]. The following years saw impressive advances in other visual tasks such as semantic segmentation [67], object detection and recognition [43] and human pose estimation [4]. Robotics has also seen impressive advances ranging from autonomous vehicles [142] to humanoid robots capable of impressively dynamic tasks [10].

For all of these advances most of the widely deployed robotic manipulation systems haven't changed much in the last 30 years. Typical robots that are deployed in auto-factories are still performing repetitive tasks such as welding and painting, where the robot is following a pre-programmed trajectory without any feedback from the environment. If we want to improve the usefulness of our robots, we need to move away from highly structured environments and robots that are performing repetitive motions with no capability for feedback and adaptation. Liberating ourselves of these constraints opens up new markets, as exemplified by the explosion of companies

(Righthand Robotics, Berkshire Grey, Covariant AI, etc.) that are competing in the logistics space.

In this thesis we aim to go further and enable our robots to perform complex manipulation tasks in real world settings. To accomplish this we use a combination of tools from both classical robotics and modern deep learning. While there are lessons to be learned from the previously mentioned advances in reinforcement learning and computer vision, we highlight the unique challenges of the robotic manipulation problem and propose new algorithms for tackling them.

1.1 Problem Statement

This thesis focuses on the question of how to enable robots to accomplish useful manipulation tasks *in the wild*. By *in the wild* we mean manipulation tasks requiring any/all of the following; (i) the use of perceptual sensors, i.e. RGBD sensors *not* external motion capture systems (ii) manipulating novel objects (iii) manipulating deformable objects, (iv) real-time closed loop visual feedback.

Robotics has shown large advances in the previous decade with the arrival of autonomous vehicles [142], humanoid robots that can do parkour [10] and drones that can navigate autonomously in cluttered GPS-denied environments [127]. These advances are impressive and one might wonder whether the techniques that led to those advances can be applied to the area of robotic manipulation. In this thesis we argue that manipulation presents a unique set of challenges from other areas of robotics that require novel solutions. A useful device to organize our thinking is to formulate the manipulation problem using the language of optimal control theory.

$$\min_{\pi} \ \mathbb{E}_{\pi}\left[\sum_{t} c(x_t, u_t)\right] \tag{1.1a}$$

$$\text{subject to} \tag{1.1b}$$

$$x_{t+1} = f(x_t, u_t) \tag{1.1c}$$

$$u_t = \pi(x_t) \tag{1.1d}$$

I argue that the manipulation problem differs from other areas of robotics in at least three dimensions. We give a brief overview here and provide further discussion in Sections 1.3.1 - 1.3.3.

1. In (1.1) the state-space is represented as x_t. For a typical robot this would often include things such as the robot's pose and joint angles as well as relevant information about the environment. In the manipulation domain however, the challenge is in representing the state of objects that must be manipulated (i.e. "state of the world") rather than the state of the robot itself. Defining and perceiving such an object representation is challenging since it must generalize to objects that the robot has never encountered before, and must be extracted from raw perceptual sensory data (e.g. RGB images).

2. The second way in which manipulation is distinct from other areas of robotics is the specification of the cost function $c(x_t, u_t)$. While for a drone a standard trajectory tracking cost of the form $c(x_t, u_t) = x_t^T Q x_t + u_t^T R u_t$ is adequate for encoding the task, such a quadratic cost cannot easily encode a complex manipulation task such as "clean up the kitchen" and struggles when dealing with novel objects.

3. A final difference is that in manipulation problems the dynamics model (1.1c) is often unknown. As above with regards to the state representation x_t, the part that is unknown or difficult to model is the dynamics of the objects/world that the robot is interacting with, rather than the dynamics of the robot itself.

Inspired by these challenges this thesis studies the following three questions.

Question 1: State Space and Object Representation What is an appropriate state-space and/or object representation for performing manipulation?

Question 2: Task Specification How can we communicate task objectives to the robot, specifically in the case where the robot must manipulate novel object instances?

Question 3: Feedback Control How can we enable robots to perform closed loop feedback control for manipulation?

1.2 Contributions

This thesis develops novel contributions in several areas related to robotic manipulation.

- Chapter 2 develops a pipeline to rapidly generate ground truth labels for RGBD data of cluttered scenes. By taking advantage of 3D reconstruction techniques we are able to vastly increase the efficiency of human labelling efforts. We used this pipeline to generate a large dataset with object pose labels (352,000 labelled images, 1,000,000+ object instances) in only a few days. This work previously appeared as [75].

- In Chapter 3 we propose dense descriptors as a useful representation for manipulation. We show how these descriptors can be quickly learned in a completely self-supervised manner (20 minutes), that they are applicable to a wide variety of objects and categories of objects, and that they enable novel manipulation tasks. In hardware experiments we demonstrate using our dense descriptor representation to grasp specific points on specific objects, including deformable objects, show that we can achieve this for a specific object in clutter, and also transfer specific grasps across objects in a class. We also contribute novel training techniques which allow us to to learn distinct multi-object dense descriptors. Additionally we show that by modifying the loss function and data sampling procedure we learn descriptors which either generalize across classes of objects or are distinct for each object instance. This work was previously published as [31].

- In Chapter 4 we introduce the concept of "category-level" manipulation tasks, examples of a *category* are mugs or shoes, and the task is defined for *all* instances of the class, rather than one particular object. To solve these types of tasks, we introduce both a novel object formulation, semantic 3D keypoints, and

a novel task specification that uses geometric costs and constraints on these keypoints to formally encode the category-level task. In addition we develop a full perception-to-action manipulation pipeline that factors the manipulation policy into instance segmentation, 3D keypoint detection, optimization based planning and local dense-geometry-based action execution. Using this pipeline we perform extensive hardware experiments demonstrating that our method can reliably accomplish tasks with never before seen objects from a category that exhibit large shape, texture and topology variations. This work was previously published as [74].

- Chapter 5 leverages self-supervised correspondence learning to perform efficient imitation learning. Specifically in hardware experiments we demonstrate four different tasks requiring real-time closed loop visual feedback. With as few as 50 demonstrations our learned policies can generalize across classes of objects, react to deformable object configurations and manipulate textureless symmetrical objects in a variety of backgrounds. This work was previously published as [32].

- Chapter 6 develops a novel formulation of dynamics model learning using self-supervised correspondence as the visual representation. We then show how to use these learned models in a model-predictive control framework to achieve closed loop feedback controllers. Using simulated manipulation experiments we demonstrate that this approach offers performance benefits over a variety of baselines, and we validate our approach in real-world robot experiments.

- We conclude by summarizing our contributions and discussing open questions and directions for future work in Chapter 7

1.3 Thesis Questions

In this Section we provide an expanded discussion of the three main questions of the thesis that were posed in Section 1.1.

1.3.1 State Space and Object Representations

The first question that we posed asks "what is an appropriate state-space and/or object representation for performing manipulation?" A standard optimal control formulation in robotics takes the form (1.1) and we must determine what is an appropriate representation of x_t. In many areas of robotics, such as flying vehicles or walking robots, x_t is the state of the robot itself. For a flying vehicle this could be it's 6-dof pose and for a walking robot it may also include the robot's joint angles. This however, is not typically the case in manipulation. As defined by Mason in [79] "manipulation refers to an agent's control of its environment through selective contact." In other words manipulation requires robots to purposefully interact with their environment to achieve tasks. Thus for robotic manipulation, the environment and the objects in it, are a fundamental part of the state representation x_t. Knowing the joint angles of your robot arm is generally not sufficient information for accomplishing meaningful manipulation tasks. In our view this means that manipulation is concerned with controlling and modifying the *environment* rather than oneself. This is in contrast to the aforementioned cases of flying vehicles or walking robots, which are typically concerned with moving **themselves** through an environment, rather that modifying the environment itself.

Consider the case of a quadrotor that is tasked from moving from point A to point B. The state of the quadrotor is well represented by it's 6-DOF pose. To perform this navigation task the quadrotor additionally requires some form of world representation. Typically for a collision free navigation task a coarse geometric occupancy map (often produced by a depth sensor or LIDAR), containing no semantic information, is a sufficient representation to accomplish the task. In this case the quadrotor can get away with a very coarse world representation containing no semantic information since it simply needs to avoid colliding with the environment. In manipulation, on the other hand, we need to reach out and interact with our environment. So performing a manipulation task such as loading the dishwasher requires a much richer geometric and semantic representation of the kitchen environment than is provided by a coarse

geometric occupancy map. Thus robot manipulation, which focuses on *interacting* with the environment, requires fundamentally different representations than those needed for moving *through* the world (e.g. flying robots, walking robots and self-driving cars).

Object Representations in Manipulation

The classical literature in manipulation has traditionally used 6-DOF pose to represent objects. If you are dealing with a known rigid object for which you have a mesh model, then 6-DOF pose completely captures the state of the object. However this representation has several limitations and drawbacks:

1. The requirement of a known mesh model. This assumption may be satisfied in some settings, such as industrial assembly, but in many other settings of interest which are unstructured (e.g. a home environment, a warehouse with a large number of constantly changing objects) is is unlikely that a mesh model will be provided for each object that must be manipulated. Although approaches such as [56] attempt to build a mesh model online in a self-supervised manner, this remains a challenging problem.

2. Because 6-DOF pose relies on matching to a rigid template model this object representation doesn't extend to deformable objects.

3. 6-DOF pose is not well defined for novel object instances. Thus it is not obvious how a manipulation system that relies on 6-DOF pose as it's object representation can interact with and manipulate novel objects. Chapters 3 and 4 propose alternative object representations to 6-DOF pose.

1.3.2 Task Specification

The second question we posed asks "how can we communicate task objectives to the robot, specifically in the case where the robot must manipulate novel object instances?". In other words how do we specify $c(x_t, u_t)$ in Equation (1.1). Traditional manipulation

approaches [68, 80] have typically specified tasks in terms of a target 6-DOF poses of specific objects. Section 1.3.1 presented several drawbacks and limitations of a pose-based approach. In particular 6-DOF pose is not well-defined for novel object instances and doesn't extend to deformable objects. Thus if we are to move beyond pose as our object representation (in order to overcome the aforementioned limitations) we will also need new approaches to specify the task. It is important to note that reinforcement learning approaches, be they model-based or model-free, still face the challenge of task specification even if the policies don't explicitly represent objects using 6-DOF pose. Reinforcement learning assumes access to a reward function, or at least requires being able to observe the numerical rewards that were accrued during an episode. Producing these reward values for tasks performed in the real world, rather than in a simulator or game environment, can be quite challenging. In [64] rewards are defined in terms of 6-DOF pose and the system uses privileged information at training time to compute these rewards. A subsequent paper [26] removes this dependence on priveleged information at training time by embedding observations in a latent space and specifying goals directly in this latent space. Although many papers [26, 2, 119, 50] have taken this approach specifying the task in a latent space is not generally guaranteed to correctly specify the goal.

As can be seen by the term $c(x_t, u_t)$ in Equation (1.1), the object representation x_t is tightly coupled with the task specification, represented by $c(x_t, u_t)$. This coupling of object representation and task specification is a major focus of Chapter 4 which shows how categories of objects (such as mugs or shoes) can be represented by semantic 3D keypoints, and many pick-and-place tasks on these categories of objects (e.g. hang the mug on the rack by the handle) can be specified via costs and constraints on these semantic 3D keypoints.

An alternative way to specify tasks is via demonstration. This is the strategy taken in the approach of Chapter 5, which completely completely bypasses an explicit definition of $c(x_t, u_t)$ and uses imitation learning to indirectly encode the task.

1.3.3 Closed-Loop Feedback Control

The final question we posed asks "how can we enable robots to perform closed loop feedback control for manipulation?" Given that vision is such an important sensing modality for manipulation this requires closing the perception-to-action loop. There are several options for achieving this in practice. The approach in Chapter 5 performs imitation learning which naturally leads to a closed-loop policy that simply attempts to imitate the demonstrators action at a particular world state. This can work fairly well but suffers from the standard limitations of imitation learning, as outlined in Chapter 6.

If the dynamics model, $f(x_t, u_t)$ in Equation (1.1), is known, then one can apply tools from optimal control theory to derive a feedback policy [47]. We must caveat this by mentioning that even in the case of rigid objects with known mesh models and perfect state estimation, synthesizing a feedback controller for manipulation tasks (e.g. peg-in-hole insertion, screwing, toppling, assembly, etc.) remains a challenging task due the hybrid nature of contact dynamics.

If the dynamics model is unknown then we are in the realm of reinforcement learning. Here there exist at least two main options, model-free RL and model-based RL. Model-free RL [3, 5, 138] attempts to directly search for a policy from observations to actions. The other alternative, often denoted as model-based RL, is to learn a model of the dynamics $f(x_t, u_t)$ and use this model to synthesize a policy. This is the approach taken in Chapter 6.

1.4 Related Work

There is a broad literature of related work to this thesis. In an attempt to provide a central and unified view of this related work I will give a brief overview of the main areas of literature that are relevant to this thesis. Throughout the discussion below I highlight chapters where additional detail can be found. The two main areas of related work for this thesis are robotic manipulation and computer vision, with most of the chapters drawing on techniques techniques from both.

The starting point for robotic manipulation are the classical model-based approaches. This covers the fundamental works [68, 80] along with more modern approaches such as [47, 91]. These approaches typically assume complete world and dynamics models, and often rely on the aid of external motion capture systems such as Vicon [139]. Although many of these works have impresseive reults, they rely on assumption that oftentimes aren't satisfied when performing manipulation tasks in more unstructured environments, specifically the assumptions of known object models and external state-estimation systems. This leads us to develop alternate approaches for perception, object representation and task specification. More details can be found in Chapters 3 and 4.

In recent years there has been an explosion of literature tackling the problem of robotic grasping [96, 101, 89, 71, 38, 156]. Some of these approaches are very general in that they work for arbitrary unknown objects, but the downside is that they cannnot accomplish any manipulation tasks beyond grasping the object and then dropping it somewhere else, we affectionately label this the "pick-and-drop" problem. Although grasping is an interesting question, we view it more as a building block in a larger manipulation system rather than an end in itself. This motivates our desire to create systems that are both "general" in that they manipulate novel objects, but also "specific" in being able to accomplish meaningful manipulation tasks beyond just "pick and drop". Further discussion can be found in Chapters 3 and 4.

There is a large literature that emerged from the Amazon Picking Challenge (APC) [156, 114, 16, 160, 44, 88]. The APC required competitors to solve a variety of pick-and-place type tasks and serves as a nice follow on from the aforementioned grasping literature since it highlights the challenges of attempting to perform a manipulation task beyond just "pick-and-drop". In particular the the APC required teams to place specific items into specific locations, and pick a specific item out of a cluttered pile. Thus being able to grasp an arbitrary object from a pile was not sufficient for accomplishing these types of tasks. The APC demonstrated that perception, and perceptual representations of objects, plays a major role in manipulation and can oftentimes be one of the most challenging components. Many APC teams arrived

at solutions that essentially combined a "grasp anything" [71, 38, 156] algorithm together with instance segmentation [43]. Although these systems were impressive and represented a significant advance in the state of the art, they were still tackling relatively simple tasks of the form "pick a specific object and drop it in a specified bin". The desire to move beyond these "pick-and-drop" style tasks was the motivation for much of the work in this thesis. Specifically we concluded that achieving this would require richer visual representations of objects, a focus of Chapters 3, 4.

On the computer vision side we make use of tools from both geometric computer vision, and modern deep learning approaches. Tools from geometric computer vision such as TSDF Fusion [17], KinectFusion [95], ElasticFusion [143], etc. are used extensively throughout this thesis as part of our data-labelling and self-supervision pipelines. A broad review of these techniques can be found in [130, 41]. These approaches allow us to take advantage of a geometric view of the world which can either be used to provide self-supervision, as in Chapter 3, or for augmenting the efficiency of human labelling, as in Chapters 2 and 4. Since one of our goals in this thesis is to be able to manipulate novel objects our perceptual systems need to be able to generalize to unseen objects. This naturally leads us to use modern deep learning based approaches, which have seen impressive advances in recent years [36]. A central theme of the thesis is in developing novel approaches to bring state-of-the-art computer vision techniques to bear on robotic manipulation problems. Chapter 3 combines the geometrical computer vision approaches with dense pixelwise descriptors [112, 15] to learn a novel object representation for manipulation. Chapter 4 uses advances in object detection [104], instance segmentation [43] and keypoint detection [129] along with geometrical computer vision techniques to define and accomplish category-level manipulation tasks. A more detailed discussion can be found in Chapters 3 and 4.

Chapter 5 combines both the self-supervised visual representation learning of Chapter 3 with Imitation Learning and Learning from Demonstration (LfD). For more detail on imitation learning methods in robotics we refer the reader to existing reviews [6, 9, 97]. Most related to our approach are methods that perform imitation learning using visual inputs [28, 118, 98]. More details can be found in Chapter 5.

A final area of related work is robot self-supervision for dynamics learning which can then be used to accomplish tasks in a model-predictive-control framework. This is in contrast to the approaches of Chapters 3 and 5 where the self-supervision was only used for the visual representation. Several approaches [24, 2] learn implicit dynamics models via a video prediction task. In contrast approaches such as [46, 92] don't solve the perception problem and instead rely on a handcrafted explicit representation on which to learn a dynamics model. Finally our work is most related to approaches such as [2, 93, 151] which still address the perception problem, but learn dynamics in a lower-dimensional latent space rather than the entire image. Chapter 6 provides more details.

Chapter 2

LabelFusion

2.1 Introduction

Advances in neural network architectures for deep learning have made significant impacts on perception for robotic manipulation tasks. State of the art networks are able to produce high quality pixelwise segmentations of RGB images, which can be used as a key component for 6DOF object pose estimation in cluttered environments [145, 160]. However for a network to be useful in practice it must be fine tuned on labeled scenes of the specific objects targeted by the manipulation task, and these networks can require tens to hundreds of thousands of labeled training examples to achieve adequate performance. To acquire sufficient data for each specific robotics application using once-per-image human labeling would be prohibitive, either in time or money. While some work has investigated closing the gap with simulated data [106, 53, 108, 42], our method can scale to these magnitudes with real data.

In this chapter we tackle this problem by developing an open-source pipeline that vastly reduces the amount of human annotation time needed to produce labeled RGBD datasets for training image segmentation neural networks. The pipeline produces ground truth segmentations and ground truth 6DOF poses for multiple objects in scenes with clutter, occlusions, and varied lighting conditions. The key components of the pipeline are: leveraging dense RGBD reconstruction to fuse together RGBD images taken from a variety of viewpoints, labeling with ICP-assisted fitting of object meshes,

and automatically rendering labels using projected object meshes. These techniques allow us to label once per scene, with each scene containing thousands of images, rather than having to annotate images individually. This reduces human annotation time by several orders of magnitude over traditional techniques. We optimize our pipeline to both collect many views of a scene and to collect many scenes with varied object arrangements. Our goal is to enable manipulation researchers and practitioners to generate customized datasets, which for example can be used to train any of the available state-of-the-art image segmentation neural network architectures. Using this method we have collected over 1,000,000 labeled object instances in multi-object scenes, with only a few days of data collection and without using any crowd sourcing platforms for human annotation.

Our primary contribution is the pipeline to rapidly generate labeled data, which researchers can use to build their own datasets, with the only hardware requirement being the RGBD sensor itself. We also have made available our own dataset, which is the largest available RGBD dataset with object-pose labels (352,000 labeled images, 1,000,000+ object instances). Additionally, we contribute a number of empirical results concerning the use of large datasets for practical deep-learning-based pixelwise segmentation of manipulation-relevant scenes in clutter – specifically, we empirically quantify the generalization value of varying aspects of the training data: (i) multi-object vs single object scenes, (ii) the number of background environments, and (iii) the number of views per scene.

2.2 Related Work

We review three areas of related work. First, we review pipelines for generating labeled RGBD data. Second, we review applications of this type of labeled data to 6DOF object pose estimation in the context of robotic manipulation tasks. Third, we review work related to our empirical evaluations, concerning questions of scale and generalization for practical learning in robotics-relevant contexts.

2.2.1 Methods for Generating Labeled RGBD Datasets

Rather than evaluate RGBD datasets based on the specific dataset they provide, we evaluate the methods used to generate them, and how well they scale. Firman [29] provides an extensive overview of over 100 available RGBD datasets. Only a few of the methods used are capable of generating labels for 6DOF object poses, and none of these associated datasets also provide per-pixel labeling of objects. One of the most related methods to ours is that used to create the T-LESS dataset [45], which contains approximately 49K RGBD images of textureless objects labeled with the 6DOF pose of each object. Compared to our approach, [45] requires highly calibrated data collection equipment. They employ fiducials for camera pose tracking which limits the ability of their method to operate in arbitrary environments. Additionally the alignment of the object models to the pointcloud is a completely manual process with no algorithmic assistance. Similarly, [145] describes a high-precision motion-capture-based approach, which does have the benefit of generating high-fidelity ground-truth pose, but its ability to scale to large scale data generation is limited by: the confines of the motion capture studio, motion capture markers on objects interfering with the data collection, and time-intensive setup for each object.

Although the approach is not capable of generating the 6 DOF poses of objects, a relevant method for per-pixel labeling is described in [160]. They employ an automated data collection pipeline in which the key idea is to use background subtraction. Two images are taken with the camera at the exact same location – in the first, no object is present, while it is in the second. Background subtraction automatically yields a pixelwise segmentation of the object. Using this approach they generate 130,000 labeled images for their 39 objects. As a pixelwise labeling method, there are a few drawbacks to this approach. The first is that in order to apply the background subtraction method, they only have a single object present in each scene. In particular there are no training images with occlusions. They could in theory extend their method to support multi-object scenes by adding objects to the scene one-by-one, but this presents practical challenges. Secondly the approach requires an accurately

calibrated robot arm to move the camera in a repeatable way. A benefit of the method, however, is that it does enable pixelwise labeling of even deformable objects.

The SceneNN [48] and ScanNet [18] data generation pipelines share some features with our method. They both use an RGBD sensor to produce a dense 3D reconstruction and then perform annotations in 3D. However, since SceneNN and ScanNet are focused on producing datasets for RGDB scene understanding tasks, the type of annotation that is needed is quite different. In particular their methods provide pixelwise segmenation into generic object classes (floor, wall, couch etc.). Neither SceneNN or ScanNet have gometric models for the specific objects in a scene and thus cannot provide 6DOF object poses. Whereas ScanNet and SceneNN focus on producing datasets for benchmarking scene understanding algorithms, we provide a pipeline to enable rapid generation labeled data for your particular application and object set.

2.2.2 Object-Specific Pose Estimation in Clutter for Robotic Manipulation

There have been a wide variety of methods to estimate object poses for manipulation. A challenge is object specificity. [145] and [160] are both state of the art pipelines for estimating object poses from RGBD images in clutter – both approaches use RGB pixelwise segmentation neural networks (trained on their datasets described in the previous section) to crop point clouds which are then fed into ICP-based algorithms to estimate object poses by registering against prior known meshes. Another approach is to directly learn pose estimation [153]. The upcoming SIXD Challenge 2017 [125] will provide a comparison of state of the art methods for 6DOF pose estimation on a common dataset. The challenge dataset contains RGBD images annotated with ground truth 6DOF object poses. This is exactly the type of data produced by our pipeline and we aim aim to submit our dataset to the 2018 challenge. There is also a trend in manipulation research to bypass object pose estimation and work directly with the raw sensor data [64, 38, 71]. Making these methods object-specific in clutter could be aided by using the pipeline presented here to train segmentation networks.

2.2.3 Empirical Evaluations of Data Requirements for Image Segmentation Generalization

While the research community is more familiar with the scale and variety of data needed for images in the style of ImageNet [110], the type of visual data that robots have available is much different than ImageNet-style images. Additionally, higher object specificity may be desired. In robotics contexts, there has been recent work in trying to identify data requirements for achieving practical performance for deep visual models trained on simulation data [106, 53, 108, 42], and specifically augmenting small datasets of real data with large datasets of simulation data [106, 53, 108, 42]. We do not know of prior studies that have performed generalization experiments with the scale of real data used here.

2.3 Data Generation Pipeline

One of the main contributions of this chapter is an efficient pipeline for generating labeled RGBD training data. The steps of the pipeline are described in the following sections: RGBD data collection, dense 3D reconstruction, object mesh generation, human assisted annotation, and rendering of labeled images.

2.3.1 RGBD Data Collection

A feature of our approach is that the RGBD sensor can either be mounted on an automated arm, as in Figure (2-1), or the the RGBD sensor can simply be hand-carried. The benefit of the former option is a reduced human workload, while the benefit of the latter option is that no sophisticated equipment (i.e. motion capture, external markers, heavy robot arm) is required, enabling data collection in a wide variety of environments. We captured 112 scenes using the handheld approach. For the remaining 26 scenes we mounted the sensor on a Kuka IIWA, as shown in Figure (2-1). The IIWA was programmed to perform a scanning pattern in both orientation and azimuth. Note that the arm-automated method does not require one to know

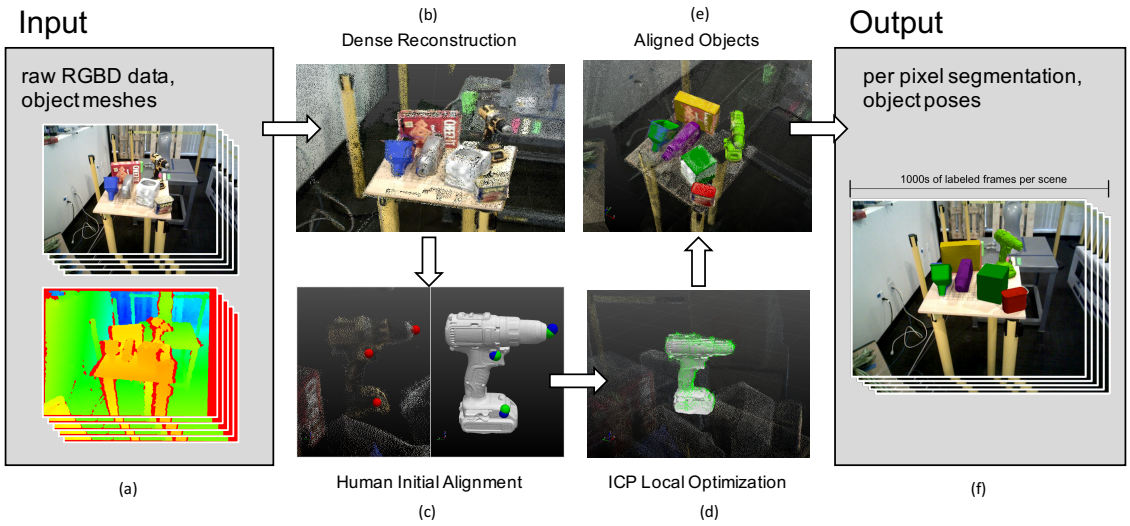


Figure 2-1: Overview of the data generation pipeline. (a) Raw data is collected using an Xtion RGBD sensor. (b) RGBD data processed by ElasticFusion into reconstructed pointcloud. (c) User annotation tool that allows for easy alignment using 3 clicks. User clicks are shown as red and blue spheres. The transform mapping the red spheres to the green spheres is then the user specified guess. (d) Cropped pointcloud coming from user specified pose estimate is shown in green. The mesh model shown in grey is then finely aligned using ICP on the cropped pointcloud. ICP is seeded with the user provided rough alignment. (e) All the aligned objects shown in reconstructed pointcloud. (f) The aligned meshes are rendered as masks in all RGB images, producing pixelwise labeled RGBD images for each view.

the transform between the robot and the camera; everything is done in camera frame. Our typical logs averaged 120 seconds in duration with data captured at 30Hz by the Asus Xtion Pro.

2.3.2 Dense 3D Reconstruction

The next step is to extract a dense 3D reconstruction of the scene, shown in Figure (2-1), from the raw RGBD data. For this step we used the open source implementation of ElasticFusion [143] with the default parameter settings, which runs in realtime on our desktop with an NVIDIA GTX 1080 GPU. ElasticFusion also provides camera pose tracking relative to the local reconstruction frame, a fact that we take advantage of when rendering labeled images. Reconstruction performance can be affected by the amount of geometric features and RGB texture in the scene. Most natural indoor

scenes provide sufficient texture, but large, flat surfaces with no RGB or depth texture can occasionally incur failure modes. Our pipeline is designed in a modular fashion so that any 3D reconstruction method that provides camera pose tracking can be used in place of ElasticFusion.

2.3.3 Object Mesh Generation

A pre-processing step for the pipeline is to obtain meshes for each object. Once obtained, meshes speed annotation by enabling alignment of the mesh model rather than manually intensive pixelwise segmentation of the 3D reconstruction. Using meshes necessitates rigid objects, but imposes no other restrictions on the objects themselves. We tested several different mesh construction techniques when building our dataset. In total there are twelve objects. Four object meshes were generated using an Artec Space Spider handheld scanner. One object was scanned using Next Engine turntable scanner. For the four objects which are part of the YCB dataset [12] we used the provided meshes. One of our objects, a tissue box, was modeled using primitive box geometry. In addition our pipeline provides a volumetric meshing method using the VTK implementation of [17] that operates directly on the data already produced by ElasticFusion. Finally, there exist several relatively low cost all-in-one solutions [49], [116], [126] which use RGBD sensors such as the Asus Xtion, Intel RealSense R300 and Occipital Structure Sensor, to generate object meshes. The only requirement is that the mesh be sufficiently high quality to enable the ICP based alignment (see section 2.3.4). RGB textures of meshes are not necessary.

2.3.4 Human Assisted Annotation

One of the key contributions of the chapter is in reducing the amount of human annotation time needed to generate labeled per-pixel and pose data of objects in clutter. We evaluated several global registration methods [164, 149] to try to automatically align our known objects to the 3D reconstruction but none of them came close to providing satisfactory results. This is due to a variety of reasons, but a principle one

is that many scene points didn't belong to any of the objects.

To circumvent this problem we developed a novel user interface that utilizes human input to assist traditional registration techniques. The user interface was developed using Director [76], a robotics interface and visualization framework. Typically the objects of interest are on a table or another flat surface – if so, a single click from the user segments out the table. The user identifies each object in the scene and then selects the corresponding mesh from the mesh library. They then perform a 3-click-based initialization of the object pose. Our insight for the alignment stage was that if the user provides a rough initial pose for the object, then traditional ICP-based techniques can successfully provide the fine alignment. The human provides the rough initial alignment by clicking three points on the object in the reconstructed pointcloud, and then clicking roughly the same three points in the object mesh, see Figure 2-1 (c). The transform that best aligns the 3 model points, shown in red, with the three scene points, shown in blue, in a least squares sense is found using the vtkLandmarkTransform function. The resulting transform then specifies an initial alignment of the object mesh to the scene, and a cropped pointcloud is taken from the points within 1cm of the roughly aligned model, as shown in green in Figure 2-1 (d). Finally, we perform ICP to align this cropped pointcloud to the model, using the rough aligment of the model as the initial seed. In practice this results in very good alignments even for cluttered scenes such as Figure 2-1 (e).

The entire human annotation process takes approximately 30 seconds per object. This is much faster than aligning the full object meshes by hand without using the 3-click technique which can take several minutes per object and results in less accurate object poses. We also compared our method with human labeling (polygon-drawing) each image, and found intersection over union (IoU) above 80%, with approximately four orders of magnitude less human effort per image (supplementary figures on our website http://labelfusion.csail.mit.edu/).

2.3.5 Rendering of Labeled Images and Object Poses

After the human annotation step of Section 2.3.4, the rest of the pipeline is automated. Given the previous steps it is easy to generate per-pixel object labels by projecting the 3D object poses back into the 2D RGB images. Since our reconstruction method, ElasticFusion, provides camera poses relative to the local reconstruction frame, and we have already aligned our object models to the reconstructed pointcloud, we also have object poses in each camera frame, for each image frame in the log. Given object poses in camera frame it is easy to get the pixelwise labels by projecting the object meshes into the rendered images. An RGB image with projected object meshes is displayed in Figure 2-1 (f).

2.3.6 Discussion

As compared to existing methods such as [145, 45, 105] our method requires no sophisticated calibration, works for arbitrary rigid objects in general environments, and requires only 30 seconds of human annotation time per object per scene. Since the human annotation is done on the full 3D reconstruction, one labeling effort automatically labels thousands of RGBD images of the same scene from different viewpoints.

2.4 Results

We first analyze the effectiveness of the LabelFusion data generation pipeline (Section 2.4.1). We then use data generated from our pipeline to perform practical empirical experiments to quantify the generalization value of different aspects of training data (Section 2.4.2).

2.4.1 Evaluation of Data Generation Pipeline

LabelFusion has the capability to rapidly produce large amounts of labeled data, with minimal human annotation time. In total we generated over 352,000 labeled

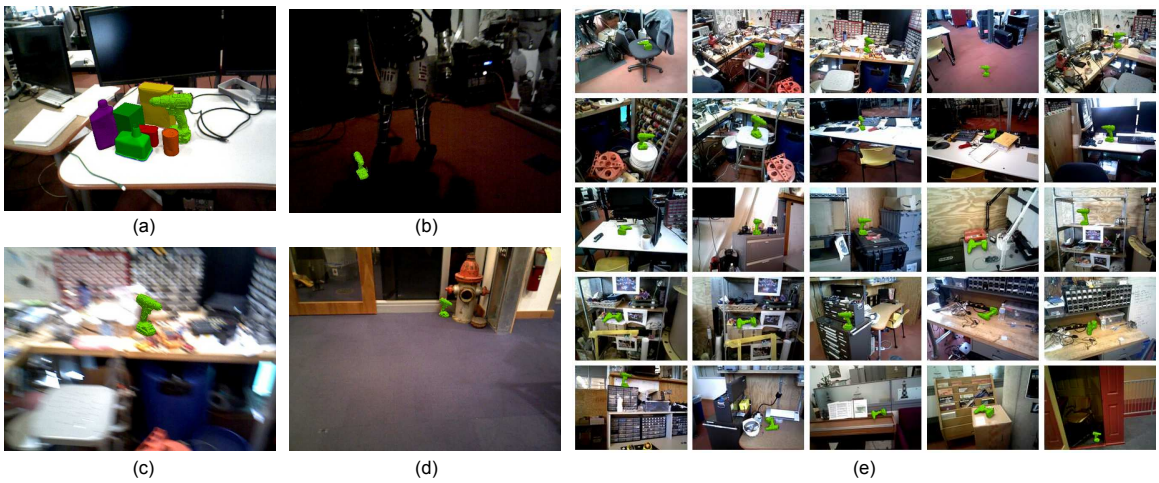


Figure 2-2: Examples of labeled data generated by our pipeline: (a) heavily cluttered multi-object, (b) low light conditions, (c) motion blur, (d) distance from object, (e) 25 different environments. All of these scenes were collected by hand-carrying the RGBD sensor.

# objects	12
# distinct scenes	105 single/double object 33 with 6+ objects
# unique object instances aligned	339
avg duration of single scene	120 seconds, 3600 RGBD frames
# labeled RGBD frames	352,000
# labeled object instances	1,000,000+

Table 2.1: Dataset Description

RGBD images, of which over 200,000 were generated in approximately one day by two people. Because many of our images are multi- object, this amounts to over 1,000,000 labeled object instances. Detailed statistics are provided in Table 2.1. The pipeline is open-source and intended for use. We were able to create training data in a wide variety of scenarios; examples are provided in Figure 2-2. In particular, we highlight the wide diversity of environments enabled by hand-carried data collection, the wide variety of lighting conditions, and the heavy clutter both of backgrounds and of multi-labeled object scenes.

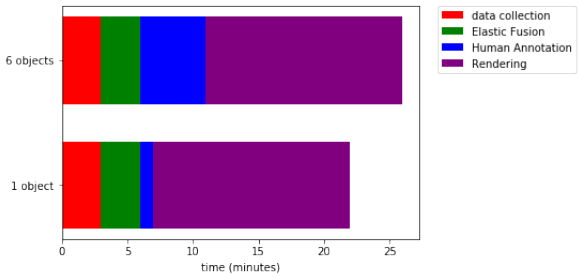


Figure 2-3: Time required for each step of pipeline.

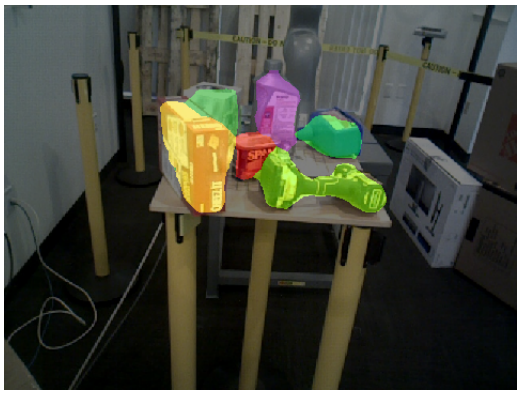


Figure 2-4: Example segmentation performance (alpha-blended with RGB image) of network (e) on a multi-object test scene.

For scaling to large scale data collection, the time required to generate data is critical. Our pipeline is highly automated and most components run at approximately real-time, as shown in Figure 2-3. The amount of human time required is approximately 30 seconds per object per scene, which for a typical single-object scene is less than real-time. Post-processing runtime is several times greater than real-time, but is easily parallelizable – in practice, a small cluster of 2-4 modern desktop machines (quad-core Intel i7 and Nvidia GTX 900 series or higher) can be made to post-process the data from a single sensor at real-time rates. With a reasonable amount of resources (one to two people and a handful of computers), it would be possible to keep up with the real-time rate of the sensor (generating labeled data at 30 Hz).

2.4.2 Empirical Evaluations: How Much Data Is Needed For Practical Object-Specific Segmentation?

With the capability to rapidly generate a vast sum of labeled real RGBD data, questions of "how much data is needed?" and "which types of data are most valuable?" are accessible. We explore practical generalization performance while varying three axes of the training data: (i) whether the training set includes multi-object scenes with occlusions or only single-object scenes, (ii) the number of background environments, and (iii) the number of views used per scene. For each, we train a state-of-the-art ResNet segmentation network [14] with different subsets of training data, and evaluate each network's generalization performance on common test sets. Further experimental details are provided in our supplementary material; due to space constraints we can only summarize results here.

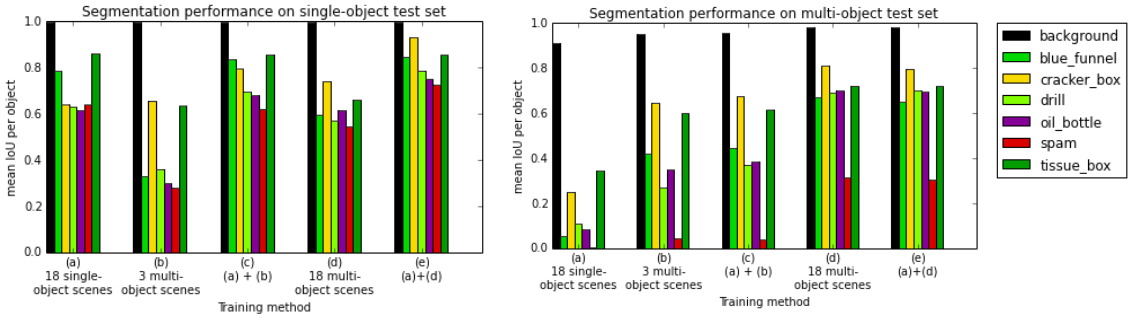


Figure 2-5: Comparisons of training on single-object vs. multi-object scenes and testing on single-object (left) and multi-object (right) scenes.

First, we investigate whether there is a benefit of using training data with heavily occluded and cluttered multi-object scenes, compared to training with only single-object scenes. Although they encounter difficulties with heavy occlusions in multi-object scenes, [160] uses purely single-object scenes for training. We trained five different networks to enable comparison of segmentation performance on novel scenes (different placements of the objects) for a single background environment. Results of segmentation performance on novel scenes (measured using the mean IoU, intersection over union, per object) show an advantage given multi-object occluded scenes (b) compared to single-object scenes (a) (Figure 2-5, right). In particular, the average IoU

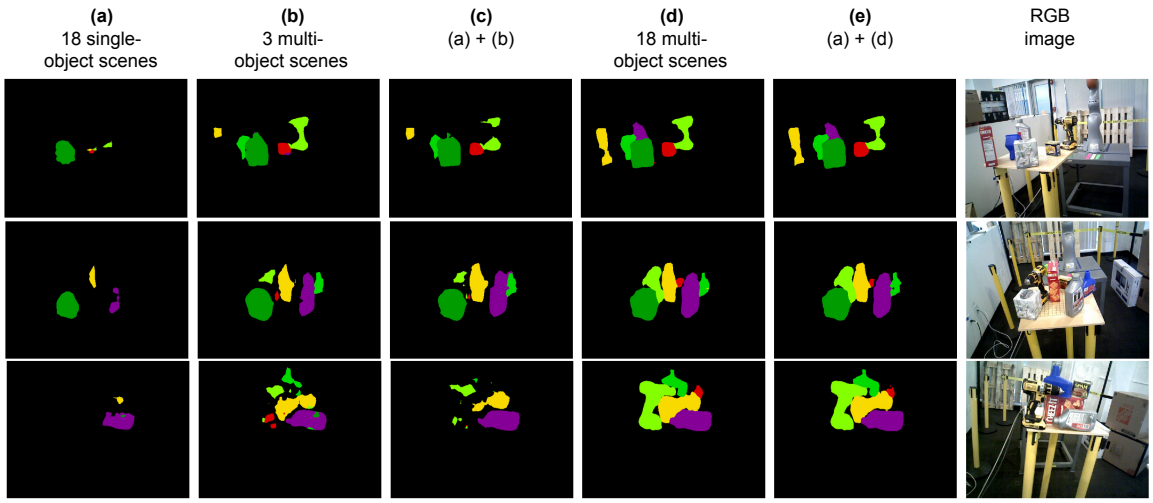


Figure 2-6: Comparison of segmentation performance on novel multi-object test scenes. Networks are either trained on (a) single object scenes only, (b,d), multi-object test scenes only, or a mixture (c,e).

per object increases 190% given training set (b) instead of (a) in Figure 2-5, right, even though (b) has strictly less labeled pixels than (a), due to occlusions. This implies that the value of the multi-object training data is more valuable per pixel than the single-object training data. When the same amount of scenes for the single-object scenes are used to train a network with multi-object scenes (d), the increase in IoU performance averaged across objects is 369%. Once the network has been trained on 18 multi-object scenes (d), an additional 18 single-object training scenes have no noticeable effect on multi-object generalization (e). For generalization performance on single-object scenes (Figure 2-5, left), this effect is not observed; single-object training scenes are sufficient for IoU performance above 60%.

Second, we ask: how does the performance curve grow as more and more training data is added from different background environments? To test this, we train different networks respectively on 1, 2, 5, 10, 25, and 50 scenes each labeled with a single drill object. The smaller datasets are subsets of the larger datasets; this directly allows us to measure the value of providing more data. The test set is comprised of 11 background environments which none of the networks have seen. We observe a steady increase in segmentation performance that is approximately logarithmic with the number of training scene backgrounds used (Figure 2-7, left). We also took

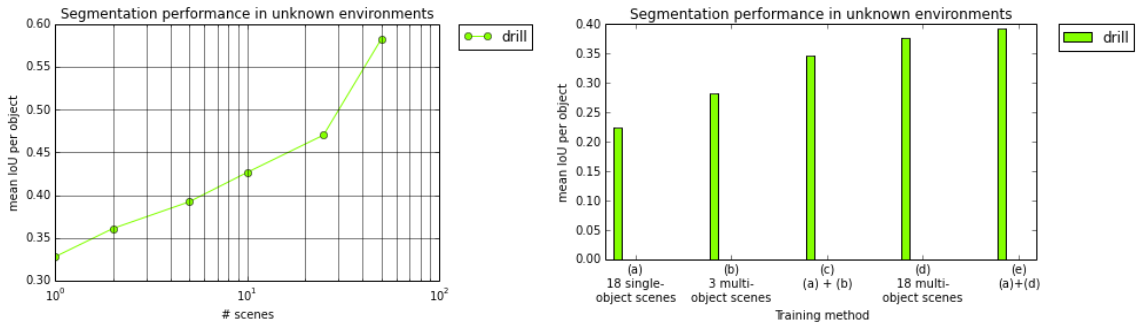


Figure 2-7: (left) Generalization performance as a function of the number of environments provided at training time, for a set of six networks trained on 50 different scenes or some subset ({1, 2, 5, 10, 25}) of those scenes. (right) Performance on the same test set of unknown scenes, but measured for the 5 training configurations for the multi-object, single-environment-only setup described previously.

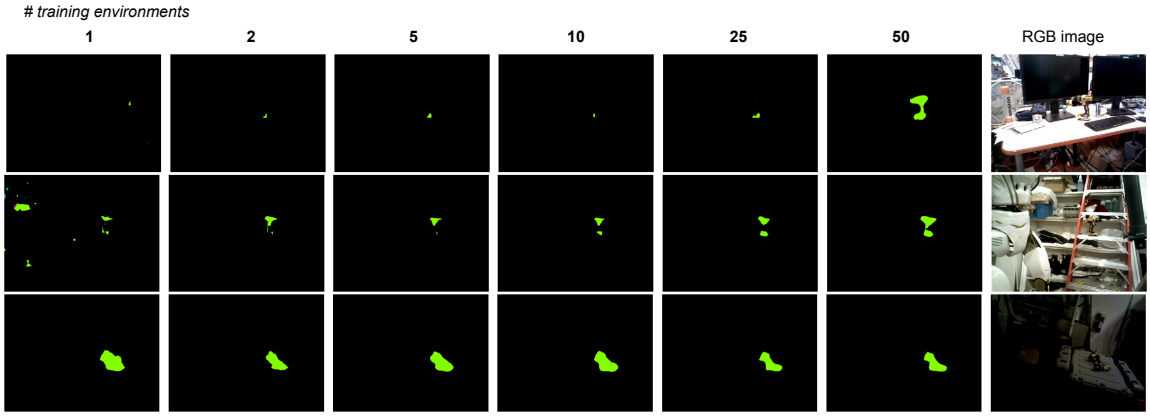


Figure 2-8: Comparison of segmentation performance on novel background environments. Networks were trained on {1, 2, 5, 10, 25, 50} background environments.

our multi-object networks trained on a single background and tested them on the 11 novel environments with the drill. We observe an advantage of the multi-object training data with occlusions over the single-object training data in generalizing to novel background environments (Figure 2-7, right).

Third, we investigate whether 30 Hz data is necessary, or whether significantly less data suffices (Figure 2-9). We perform experiments with downsampling the effective sensor rate both for robot-arm-mounted multi-object single-background training set (e), and the hand-carried many-environments dataset with either 10 or 50 scenes. For each, we train four different networks, where one has all data available and the others have downsampled data at respectively 0.03, 0.3, and 3 Hz. We observe a monotonic

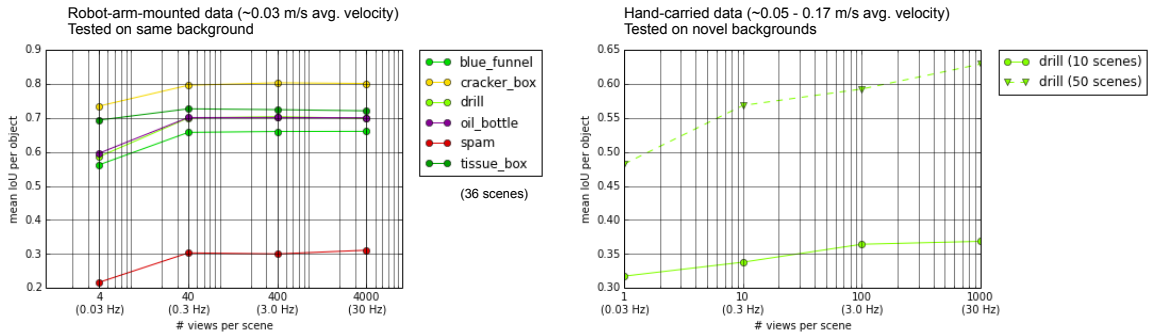


Figure 2-9: Pixelwise segmentation performance as a function of the number of views per scene, reduced by downsampling the native 30 Hz sensor to {0.03, 0.3, 3.0.} Hz.

increase in segmentation performance as the effective sensor rate is increased, but with heavily diminished returns after 0.3 Hz for the slower robot-arm-mounted data (∼0.03 m/s camera motion velocity). The hand-carried data (∼0.05 - 0.17 m/s) shows more gains with higher rates.

2.5 Conclusion

This chapter introduced LabelFusion, our pipeline for efficiently generating RGBD data annotated with per-pixel labels and ground truth object poses. Specifically only a few minutes of human time are required for labeling a scene containing thousands of RGBD images. LabelFusion is open source and available for community use, and we also supply an example dataset generated by our pipeline [61].

The capability to produce a large, labeled dataset enabled us to answer several questions related to the type and quantity of training data needed for practical deep learning segmentation networks in a robotic manipulation context. Specifically we found that networks trained on multi-object scenes performed significantly better than those trained on single object scenes, both on novel multi-object scenes with the same background, and on single-object scenes with new backgrounds. Increasing the variety of backgrounds in the training data for single-object scenes also improved generalization performance for new backgrounds, with approximately 50 different backgrounds breaking into above-50% IoU on entirely novel scenes. Our recommendation is to

focus on multi-object data collection in a variety of backgrounds for the most gains in generalization performance.

We hope that our pipeline lowers the barrier to entry for using deep learning approaches for perception in support of robotic manipulation tasks by reducing the amount of human time needed to generate vast quantities of labeled data for *your* specific environment and set of objects. It is also our hope that our analysis of segmentation network performance provides guidance on the type and quantity of data that needs to be collected to achieve desired levels of generalization performance.

Acknowledgement

The authors thank Matthew O'Kelly for his guidance with segmentation networks and manuscript feedback. We also thank Allison Fastman and Sammy Creasey of Toyota Research Institute for their help with hardware, including object scanning and robot arm automation. David Johnson of Draper Laboratory and Shuran Song of Princeton University provided valuable input on training. We are grateful we were able to use the robot arm testing facility from Toyota Research Insitute. This work was supported by the Air Force/Lincoln Laboratory award no. 7000374874, by the Defense Advanced Research Projects Agency via Air Force Research Laboratory award FA8750-12-1-0321, and by NSF Contract IIS-1427050. The views expressed are not endorsed by the sponsors.

Chapter 3

DenseObjectNets

3.1 Introduction

What is the right object representation for manipulation? While task-specific reinforcement learning can achieve impressively dexterous skills for a given specific task [64], it is unclear which is the best route to efficiently achieving many different tasks. Other recent work [71, 38] can provide very general grasping functionality but does not address specificity. Achieving specificity, the ability to accomplish specific tasks with specific objects, may require solving the data association problem. At a coarse level the task of identifying individual objects to manipulate can be solved by instance segmentation, as demonstrated in the Amazon Robotics Challenge (ARC) [114, 84] or [52]. Whole-object-level segmentation, however, does not provide any information on the rich structure of the objects themselves, and hence may not be an appropriate representation for solving more complex tasks. While not previously applied to the robotic manipulation domain, recent work has demonstrated advances in learning dense pixel level data association [15, 112], including self-supervision from raw RGBD data [112], which inspired our present work.

In this chapter we propose and demonstrate using dense visual description as a representation for robotic manipulation. We demonstrate the first autonomous system that can entirely self-supervise to learn consistent dense visual representations of

Code, data, and video available: github.com/RobotLocomotion/pytorch-dense-correspondence

objects, and the first system we know of that is capable of performing the manipulation demonstrations we provide. Specifically, with no human supervision during training, our system can grasp specific locations on deformable objects, grasp semantically corresponding locations on instances in a class, and grasp specific locations on specific instances in clutter. Towards this goal, we also provide practical contributions to dense visual descriptor learning with general computer vision applications outside of robotic manipulation. We call our visual representations Dense Object Nets, which are deep neural networks trained to provide dense (pixelwise) description of objects.

Contributions. We believe our largest contribution is that we introduce dense descriptors as a representation useful for robotic manipulation. We've also shown that self-supervised dense visual descriptor learning can be applied to a wide variety of potentially non-rigid objects and classes (47 objects so far, including 3 distinct classes), can be learned quickly (approximately 20 minutes), and can enable new manipulation tasks. In example tasks we grasp specific points on objects across potentially deformed configurations, do so with object instance-specificity in clutter, or transfer specific grasps across objects in a class. We also contribute novel techniques to enable multi-object distinct dense descriptors, and show that by modifying the loss function and sampling procedure, we can either acquire descriptors which generalize across classes of objects, or descriptors that are distinct for each object instance. Finally, we contribute general training techniques for dense descriptors which we found to be critical to achieving good performance in practice.

Chapter Organization. In Section 3.2 we describe related work. As preliminary in Section 3.3.1 we describe the general technique for self-supervising dense visual descriptor learning, which is from [112] but reproduced here for clarity. We then describe additional techniques we've developed for object-centric visual descriptors in Section 3.3.2, and Section 3.3.3 describes techniques for distinct multi-object descriptors. Section 3.4 describes our experimental setup for our autonomous system, and Section 5 describes our results: our learned visual descriptors for a wide variety of objects (Section 3.5.1) multi-object descriptors and selective class generalization (Sections 3.5.2 and 3.5.3), and robotic manipulation demonstrations (Section 3.5.4).

3.2 Related Work

We review three main areas of related work: learned descriptors, self-supervised visual learning for robots, and robot learning for specific tasks. The task of correspondence estimation from multiple views of the same scene is fundamental in computer vision, whereas dense semantic correspondence across *different* scenes was popularized by [66]. Recent advances have been made by introducing a pixel-wise variant of contrastive loss [39] combined with deep convolutional networks, as in Choy et al. [15] and Schmidt et al. [112]. For cross-instance semantic correspondence, [15] relies on human annotations, while [112] learns these unsupervised, as we do here. Other work [133] uses image warping to learn descriptors, and most require manually annotated labels [131, 120, 11]. Zeng et al. [158] also uses dense 3D reconstruction to provide automated labeling, but for descriptors of 3D volume patches. Some of these works [112, 133, 11], like ours, learn descriptors for specific object instances or classes, while others [158] learn descriptors for establishing correspondence of arbitrary data. None of these prior works in dense visual learning involve robots.

In the area of self-supervised visual robot learning, while some recent work has sought to understand 'how will the world change given the robot's action?" [24, 93] in this work we instead ask "what is the current visual state of the robot's world?". We address this question with a dense description that is consistent across viewpoints, object configurations and (if desired) object classes. At the coarse level of semantic segmentation several works from the Amazon Robotics Challenge used robots to automate the data collection and annotation process through image-level background subtraction [156, 84, 114]. In contrast this work uses 3D reconstruction-based change detection and dense pixelwise correspondences, which provides a much richer supervisory signal for use during training.

In the area of robot learning for a specific task there have been impressive works on end-to-end reinforcement learning [64, 159]. In these papers the goal is to learn a specific task, encoded with a reward function, whereas we learn a general task agnostic visual representation. There have also been several works focusing on grasping from

RGB or depth images [38, 156, 71, 101]. These papers focus on successfully grasping any item out of a pile, and are effectively looking for graspable features. They have no consistent object representation or specific location on that object, and thus the robotic manipulation tasks we demonstrate in Section 3.5.4, e.g. grasping specific points on an object across potentially deformed object configurations, are out of scope for these works.

3.3 Methodology

3.3.1 Preliminary: Self-Supervised Pixelwise Contrastive Loss

We use self-supervised pixelwise contrastive loss, as developed in [15, 112]. This learns a dense visual descriptor mapping which maps a full-resolution RGB image, $\mathbb{R}^{W \times H \times 3}$ to a dense descriptor space, $\mathbb{R}^{W \times H \times D}$, where for each pixel we have a D-dimensional descriptor vector. Training is performed in a Siamese fashion, where a pair of RGB images, I_a and I_b are sampled from one RGBD video, and many pixel matches and non-matches are generated from the pair of images. A pixel $u_a \in \mathbb{R}^2$ from image I_a is a match with pixel u_b from image I_b if they correspond to the same vertex of the dense 3D reconstruction (Figure 3-1 (c-f)). The dense descriptor mapping is trained via pixelwise contrastive loss. The loss function aims to minimize the distance between descriptors corresponding to a match, while descriptors corresponding to a non-match should be at least a distance M apart, where M is a margin parameter. The dense descriptor mapping $f(\cdot)$ is used to map an image $I \in \mathbb{R}^{W \times H \times 3}$ to descriptor space $f(I) \in \mathbb{R}^{W \times H \times D}$. Given a pixel u we use $f(I)(u)$ to denote the descriptor corresponding to pixel u in image I. We simply round the real-valued pixel $u \in \mathbb{R}^2$ to the closest discrete pixel value $u \in \mathbb{N}^2$, but any continuously-differentiable interpolation can be used for sub-pixel resolution. We denote $D(\cdot)$ as the L_2 distance between a pair of pixel descriptors: $D(I_a, u_a, I_b, u_b) \triangleq ||f(I_a)(u_a) - f(I_b)(u_b)||_2$. At each iteration of training, a large number (on the order of 1 million total) of matches N_{matches} and non-matches $N_{\text{non-matches}}$ are generated between images I_a and I_b. The images are

mapped to corresponding descriptor images via $f(\cdot)$ and the loss function is

$$\mathcal{L}_{\text{matches}}(I_a, I_b) = \frac{1}{N_{\text{matches}}} \sum_{N_{\text{matches}}} D(I_a, u_a, I_b, u_b)^2 \tag{3.1}$$

$$\mathcal{L}_{\text{non-matches}}(I_a, I_b) = \frac{1}{N_{\text{non-matches}}} \sum_{N_{\text{non-matches}}} \max(0, M - D(I_a, u_a, I_b, u_b))^2 \tag{3.2}$$

$$\mathcal{L}(I_a, I_b) = \mathcal{L}_{\text{matches}}(I_a, I_b) + \mathcal{L}_{\text{non-matches}}(I_a, I_b) \tag{3.3}$$

3.3.2 Training Procedures for Object-Centric Descriptors

Prior work [112] has used dynamic reconstruction [94] of raw RGBD data for only within-scene data association and remarkably showed that even without cross-scene data association, descriptors could be learned that were consistent across many dynamic scenes of the upper body of a human subject. While dynamic reconstruction is powerful, the challenges of topology changes [34] and difficulties of occlusion make it difficult to reliably deploy for an autonomous system. Schmidt et al. [112] also used data associations from static scene reconstructions for the task of relocalization in the same static environment. In contrast we sought to use only static reconstruction but seek consistency for dynamic objects. Other work [133] obtains dense descriptor consistency for a curated dataset of celebrity faces using only image warping for data association.

Using our robot mounted camera we are able to reliably collect high quality dense reconstructions for static scenes. Initially we applied only static-scene reconstruction to learn descriptors for specific objects, but we found that the learned object descriptors were not naturally consistent for challenging datasets with objects in significantly different configurations. Subsequently we developed techniques that leverage 3D reconstruction change detection, data augmentation, and loss function balancing to reliably produce consistent object representations with only static-scene data association for the wide variety of objects we have tested. These techniques also improve the precision of correspondences, as is discussed in Section 3.5.1. While we have tried many other ideas (see Appendix A.4.2), these are the techniques that were

empirically found to significantly improve performance.

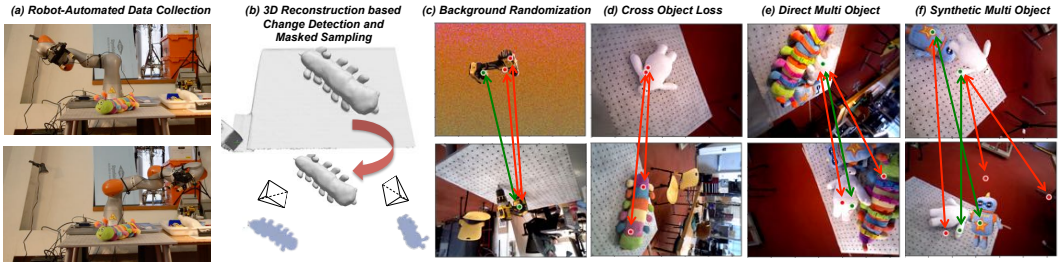


Figure 3-1: Overview of the data collection and training procedure. (a) automated collection with a robot arm. (b) change detection using the dense 3D reconstruction. (c)-(f) matches depicted in green, non-matches depicted in red.

Object masking via 3D change detection. Since we are trying to learn descriptors of objects that take up only a fraction of a full image, we observe significant improvements if the representational power of the models are focused on the objects rather than the backgrounds. A 640×480 image contains $307,200$ pixels but an image in our dataset may have as few as 1,000 to 10,000 of those pixels, or .3%-3%, that correspond to the object of interest. Initial testing with human-labeled object masks [78] showed that if matches for data associations were sampled only on the object (while non-matches were sampled from the full image) then correspondence performance was significantly improved. In order to provide autonomous object masking without any human input, we leverage our 3D reconstructions and results from the literature on 3D change detection [22] to recover the object-only part of the reconstruction (Figure 3-1b). Projecting this geometry into each camera frame yields object masks for each image. We want to emphasize that automatic object masking enables many other techniques in this chapter, including: background domain randomization, cross-object loss, and synthetic multi-object scenes.

Background domain randomization. A strategy to encourage cross-scene consistency is to enforce that the learned descriptors are not reliant on the background. Since we have autonomously acquired object masks, we can domain randomize [134] the background (Figure 3-1c top) to encourage consistency – rather than memorizing the background (i.e. by describing the object by where it is relative to a table edge), the descriptors are forced to be representative of only the object.

Hard-negative scaling. Although as in [112] we originally normalized $\mathcal{L}_{\text{matches}}$ and $\mathcal{L}_{\text{non-matches}}$ by N_{matches} and $N_{\text{non-matches}}$, respectively, we found that what we call the "hard-negative rate", i.e. the percentage of sampled non-matches for which $M - D(I_a, u_a, I_b, u_b) > 0$ would quickly drop well below 1% during training. While not precisely hard-negative mining [21], we empirically measure improved performance if rather than scaling $\mathcal{L}_{\text{non-matches}}$ by $N_{\text{non-matches}}$, we adaptively scale by the number of hard negatives in the non-match sampling, $N_{\text{hard-negatives}}$, where $\mathbb{1}$ is the indicator function:

$$N_{\text{hard-negatives}} = \sum_{N_{\text{non-matches}}} \mathbb{1}(M - D(I_a, u_a, I_b, u_b) > 0) \tag{3.4}$$

$$\mathcal{L}_{\text{non-matches}}(I_a, I_b) = \frac{1}{N_{\text{hard-negatives}}} \sum_{N_{\text{non-matches}}} max(0, M - D(I_a, u_a, I_b, u_b))^2 \tag{3.5}$$

Data augmentation and normalization. While we collect only a modest number of scenes (4-10) per object or class, we ensure they are diverse in orientations, crops, and lighting conditions. We also applied synthetic 180-degree rotations randomly to our images. Additionally we find gains in performance by projecting all features to the unit sphere, i.e. $f(I)(u) \leftarrow \frac{f(I)(u)}{||f(I)(u)||}$ when using high-dimensional descriptors spaces (i.e., more than $D = 4$). This is explained further in Appendix A.4.1.

3.3.3 Multi-Object Dense Descriptors

We of course would like robots to have dense visual models of more than just one object. When we began this work it wasn't obvious to us what scale of changes to our training procedure or model architecture would be required in order to simultaneously (a) achieve individual single-object performance comparable to a single-object-only model, while also (b) learn dense visual descriptors for objects that are *globally distinct* – i.e., the bill of a hat would occupy a different place in descriptor space than the handle of a mug. To achieve distinctness, we introduce three strategies:

i. Cross-object loss. The most direct way to ensure that different objects occupy different subsets of descriptor space is to directly impose *cross-object loss* (Figure

3-1d). Between two different objects, we know that each and every pair of pixels between them is a non-match. Accordingly we randomly select two images of two different objects, randomly sample many pixels from each object (enabled by object masking), and apply non-match loss (with hard-negative scaling) to all of these pixel pairs.

ii. Direct training on multi-object scenes. A nice property of pixelwise contrastive loss, with data associations provided by 3D geometry, is that we can directly train on multi-object, cluttered scenes *without any individual object masks* (Figure 3-1e). This is in contrast with training pixelwise semantic segmentation, which requires labels for each individual object in clutter that may be difficult to attain, i.e. through human labeling. With pixel-level data associations provided instead by 3D geometry, the sampling of matches and the loss function still makes sense, even in clutter.

iii. Synthetic multi-object scenes. We can also synthetically create multi-object scenes by layering object masks [114]. To use dense data associations through synthetic image merging, we prune matches that become occluded during layering (Figure 3-1f). A benefit of this procedure is that we can create a combinatorial number of "multi-object" scenes from only single object-scenes, and can cover a wide range of occlusion types without collecting physical scenes for each.

3.4 Experimental

Data Collection and Pre-Processing. The minimum requirement for raw data is to collect an RGBD video of an object or objects. Figure 3-1 shows our experimental setup; we utilize a 7-DOF robot arm (Kuka IIWA LBR) with an RGBD sensor (Primesense Carmine 1.09) mounted at the end-effector. With the robot arm, data collection can be highly automated, and we can achieve reliable camera poses by using forward kinematics along with knowledge of the camera extrinsic calibration. For dense reconstruction we use TSDF fusion [17] of the depth images with camera poses provided by forward kinematics. An alternative route to collecting data which

does not require a calibrated robot is to use a dense SLAM method (for example, [95, 143]). In between collecting RGBD videos, the object of interest should be moved to a variety of configurations, and the lighting can be changed if desired. While for many of our data collections a human moved the object between configurations, we have also implemented and demonstrated (see our video) the robot autonomously rearranging the objects, which highly automates the object learning process. We employ a Schunk two-finger gripper and plan grasps directly on the object point cloud (Appendix A.3). If multiple different objects are used, currently the human must still switch the objects for the robot and indicate which scenes correspond to which object, but even this information could be automated by the robot picking objects from an auxiliary bin.

Training Dense Descriptors. For training, at each iteration we randomly sample between some subset of specified image comparison types (Single Object Within Scene, Different Object Across Scene, Multi Object Within Scene, Synthetic Multi Object), and then sample some set of matches and non-matches for each. In this work, we use only static-scene reconstructions, so pixel matches between images can be easily found by raycasting and reprojecting against the dense 3D reconstruction model, and appropriately checking for occlusions and field-of-view constraints. For the dense descriptor mapping we train a 34-layer, stride-8 ResNet pretrained on ImageNet, but we expect any fully-convolutional network (FCN) that has shown effectiveness on semantic segmentation tasks to work well. Additional training details are contained in Appendix A.4.

3.5 Results

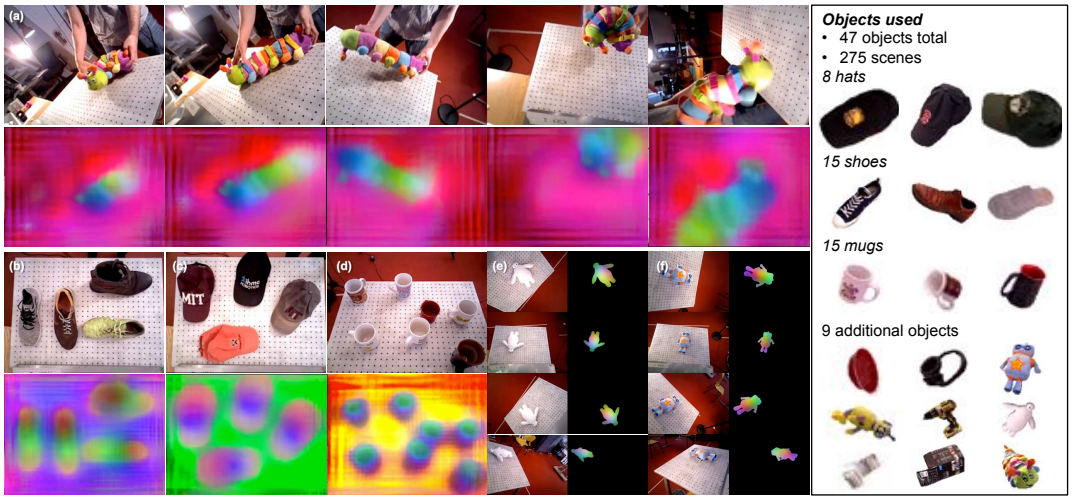


Figure 3-2: Learned object descriptors can be consistent across significant deformation (a) and, if desired, across object classes (b-d). Shown for each (a) and (b-d) are RGB frames (top) and corresponding descriptor images (bottom) that are the direct output of a feed-forward pass through a trained network. (e)-(f) shows that we can learn descriptors for low texture objects, with the descriptors masked for clear visualization. Our object set is also summarized (right).

3.5.1 Single-Object Dense Descriptors

We observe that with our training procedures described in Section 3.3.2, for a wide variety of objects we can acquire dense descriptors that are invariant to viewpoint, configuration, and deformation. The variety of objects includes moderately deformable objects such as soft plush toys, shoes, mugs, and hats, and can include very low-texture objects (Figure 3-2). Many of these objects were just grabbed from around the lab (including the authors' and labmates' shoes and hats), and dense visual models can be reliably trained with the same network architecture and training parameters. The techniques in Section 3.3.2 provide significant improvement in both (a) qualitative consistency over a wide variety of viewpoints, and (b) quantitative precision in correspondences. As with other works that learn pairwise mappings to some descriptor space [113], in practice performance can widely vary based on specific sampling of data associations and non-associations used during training. One way to quantitatively evaluate correspondence precision is with human-labeled (used only

for evaluation; never for training) correspondences across two images of an object in *different* configurations. Given two images I_a, I_b containing the same object and pixel locations $u_a^* \in I_a, u_b^* \in I_b$ corresponding to the same physical point on the object, we can use our dense descriptors to estimate u_b^* as $\hat{u}_b$:

$$\hat{u}_b \triangleq \underset{u_b \in I_b}{\arg\min} \; D(I_a, u_a^*, I_b, u_b) \tag{3.6}$$

Figure 3-3 (b-c) shows a quantitative comparison of ablative experiments, for four different training procedures described in Figure 3-3a. Our new standard single-object training procedure (**standard-SO**) performs significantly better than our implementation of prior work's training procedures (**Schmidt**), and we isolate and measure significant improvement in correspondence precision for both object-masking and hard-negative scaling. We also find that for some low-texture objects, orientation randomization and background domain randomization are critical for attaining consistent object descriptors. Otherwise the model may learn to memorize which side of the object is closest the table, rather than a consistent object model (Figure 3-4b). Background domain randomization is most beneficial for smaller datasets, where it can significantly reduce overfitting and encourage consistency (Figure 3-4a); it is less critical for high-texture objects and larger datasets.

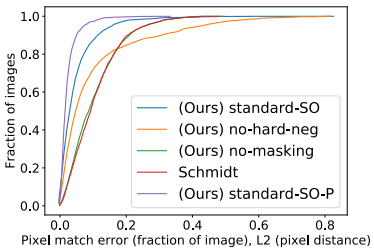
3.5.2 Multi-Object Dense Descriptors

An early observation during experimentation was that *overlap* in descriptor space naturally occurs if the same model is trained simultaneously on different singulated objects, where sampling of matches and non-matches was only performed *within scene*. Since there is no component of the loss function that requires different objects to occupy different subsets of descriptor space, the model maps them to an overlapping subset of descriptor space, distinct from the background but not each other (Figure 3-5a). Accordingly we sought to answer the question of whether or not we could *separate* these objects into unique parts of descriptor space.

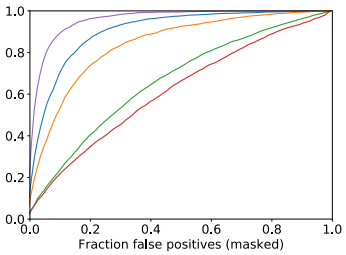
By applying cross-object loss (Section 3.3.3.i, training mode **specific** in Figure 3-

	single or multi object dataset	masked match sampling	scale by hard negatives	cross-object loss
standard-SO	single	✓	✓	
no-masking	single		✓	
no-hard-neg	single	✓		
Schmidt	single			
consistent	multi	✓	✓	
specific	multi	✓	✓	✓

(a)

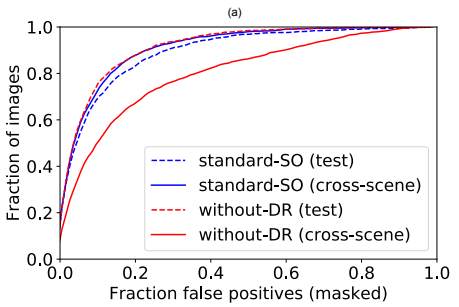


(b)



(c)

Figure 3-3: (a) table describing the network training procedures referenced in experiments. (**standard-SO** = "standard single object". **standard-SO-P** is detailed in Appendix A.4.1). (b) Plots the cdf of the L2 pixel distance (normalized by image diagonal, 800 for a 640 x 480 image) between the best match $\hat{u}_b$ and the true match u_b^*, e.g. for **standard-SO** in 93% of image pairs the normalized pixel distance between u_b^* and $\hat{u}_b$ is less than 13%. All networks were trained on the same dataset. (c) Plots the cdf of the fraction of pixels u_b of the object pixels with $D(I_a, u_a^*, I_b, u_b) < D(I_a, u_a^*, I_b, u_b^*)$, i.e. they are closer in descriptor space to u_a^* than the true match u_b^*.



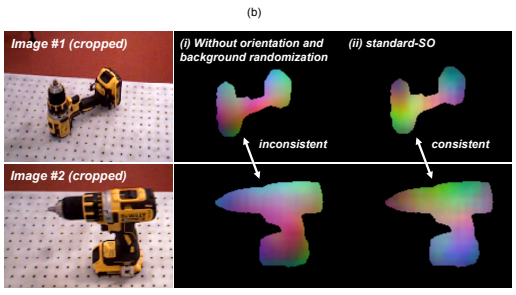


Figure 3-4: (a), with same axes as Figure 3-3b, compares **standard-SO** with **without-DR**, for which the only difference is that **without-DR** used no background domain randomization during training. The dataset used for (a) is of three objects, 4 scenes each. (b) shows that for a dataset containing 10 scenes of a drill, learned descriptors are inconsistent without background and orientation randomization during training (middle), but consistent with them (right).

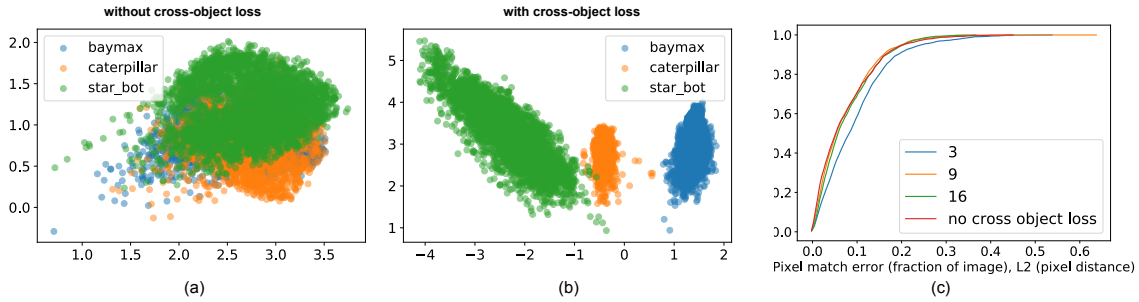


Figure 3-5: Comparison of training without any distinct object loss (a) vs. using cross-object loss (b). In (b), 50% of training iterations applied cross-object loss and 50% applied single-object within-scene loss, whereas (a) is 100% single-object within-scene loss. The plots show a scatter of the descriptors for 10,000 randomly-selected pixels for each of three distinct objects. Networks were trained with $D = 2$ to allow direct cluster visualization. (c) Same axes as Figure 3-3 (a). All networks were trained on the same 3 object dataset. Networks with a number label were trained with cross object loss and the number denotes the descriptor dimension. no-cross-object is a network trained without cross object loss.

3a), we can convincingly separate multiple objects such that they each occupy distinct subsets of descriptor space (Figure 3-5b). Note that cross-object loss is an extension of sampling *across scene* as opposed to only *within scene*. Given that we can separate objects in descriptor space, we next investigate: does the introduction of object distinctness significantly limit the ability of the models to achieve correspondence precision for each individual object? For multi-object datasets, we observe that there is a measurable decrease in correspondence precision for small-dimensional descriptor spaces when the cross-object loss is introduced, but we can recover correspondence precision by training slightly larger-dimensional descriptor spaces (Figure 3-5c). For the most part, 3-dimensional descriptor spaces were sufficient to achieve saturated (did not improve with higher-dimension) correspondence precision for single objects, yet this is often not the case for distinct multi-object networks.

3.5.3 Selective Class Generalization or Instance Specificity

Surprisingly we find that when trained simultaneously on similar items of a class using training mode **consistent**, the learned descriptors naturally generalize well across sufficiently similar instances of the class. This result of converging descriptors

across a class is similar to the surprising generalization observed for human datasets in [112, 133]. Here we show that we can obtain class consistent dense descriptors for 3 different classes of objects (hats, shoes, and mugs) trained with only static-scene data association. We observe that the descriptors are consistent despite considerable differences in color, texture, deformation, and even to some extent underlying shape. The training requirements are reasonably modest – only 6 instances of hats were used for training yet the descriptors generalize well to unseen hats, including a blue hat, a color never observed during training. The generalization extends to instances that a priori we thought would be failure modes: we expected the boot (Figure 3-6h) to be a failure mode but there is still reasonable consistency with other shoes. Sufficiently different objects are not well generalized, however – for example Baymax and Starbot (Figure 3-2e,f) are both anthropomorphic toys but we do not attain general descriptors for them. While initially we proposed research into further encouraging consistency within classes, for example by training a Dense Object Net to fool an instance-discriminator, the level of consistency that naturally emerges is remarkable and was sufficient for our desired levels of precision and applications.

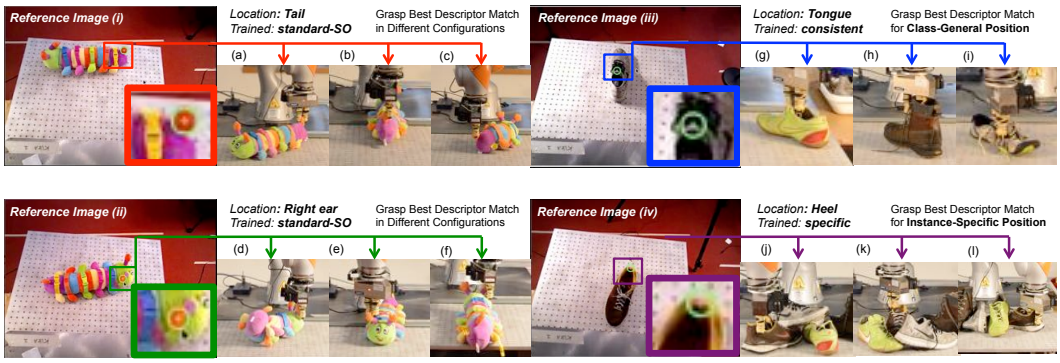
For other applications, however, instance-specificity is desired. For example, what if you would like your robot to recognize a certain point on hat A as distinct from the comparable point on hat B? Although we could separate very distinct objects in multi-object settings as discussed in the previous section, it wasn't obvious to us if we could satisfactorily separate objects of the same class. We observe, however, that by applying the multi-object techniques (**specific** in Figure 3-3) previously discussed, we can indeed learn distinct descriptors even for very similar objects in a class (Figure 3-6iv).

3.5.4 Example Applications to Robotic Manipulation: Grasping Specific Points

Here we demonstrate a variety of manipulation applications in grasping specific points on objects, where the point of interest is specified in a reference image. We emphasize

there could be many other applications, as mentioned in the Conclusion. In our demonstrations, a user clicks on just one pixel u_a^* in one reference image. Now the robot has the ability to autonomously identify the corresponding point in new scenes via Equation 3.6. Akin to other works with similarity learning in metric spaces [113], we set a simple threshold to determine whether a valid match exists. If a match is identified in the new scene we can instruct the robot to autonomously grasp this point by looking up the corresponding location in the point cloud and using simple geometric grasping techniques (details in Appendix A.3).

The particular novel components of these manipulation demonstrations are in grasping the visual corresponding points for arbitrary pixels that are either in different (potentially deformed) configurations (Fig. 3-6i-ii), general across instances of classes (Fig. 3-6iii), or instance-specific in clutter (Fig. 3-6iv). Our video[1] best displays these tasks. Note that only a dense (as opposed to sparse) method can easily accommodate the arbitrary selection of interaction points, and class-generalization is out of scope for hand-designed descriptors such as SIFT. This is also out of scope for general grasp planners like [38, 156, 71, 101] which lack any visual object representation, and for segmentation based methods [156, 84, 114] since the visual representation provided by segmentation doesn't capture any information beyond the object mask.



[1]See video (`https://youtu.be/L5UW1VapKNE`) for extensive videos of the different types of robot picking.

Figure 3-6: Depiction of "grasp specific point" demonstrations. For each the user specifies a pixel in a single reference image, and the robot automatically grasps the best match in test configurations. For single-object demonstrations, two different points for the caterpillar object are shown: tail (i) and right ear (ii). Note that the "right-ear" demonstration is an example of the ability to break symmetry on reasonably symmetrical objects. For class generalization (iii), trained with **consistent**, the robot grasps the class-general point on a variety of instances. This was trained on only 4 shoes and extends to unseen instances of the shoe class, for example (iii-i). For instance-specificity (iv) trained with **specific** and augmented with synthetic multi object scenes (3.3.3.iii), the robot grasps this point on the specific instance even in clutter.

3.6 Conclusion

This work introduces Dense Object Nets as visual object representations which are useful for robotic manipulation and can be acquired with only robot self-supervision. Building on prior work on learning pixel-level data associations we develop new techniques for object-centricness, multi-object distinct descriptors, and learning dense descriptors *by and for* robotic manipulation. Without these object centric techniques we found that data associations from static-scene reconstructions were not sufficient to achieve consistent object descriptors. Our approach has enabled automated and reliable descriptor learning at scale for a wide variety of objects (47 objects, and 3 classes). We also show how learned dense descriptors can be extended to the multi object setting. With new contrastive techniques we are able to train Dense Object Nets that map different objects to different parts of descriptor space. Quantitative experiments show we can train these multi object networks while still retaining the performance of networks that do not distinguish objects. We also can learn class-general descriptors which generalize across different object instances, and demonstrated this result for three classes: shoes, hats, and mugs. Using class-general descriptors we demonstrate a robot transferring grasps across different instances of a class. Finally we demonstrate that our distinct-object techniques work even for objects which belong to the same class. This is demonstrated by the robot grasping a specific point on a target shoe in a cluttered pile of shoes. We believe Dense Object Nets can enable

many new approaches to robotic manipulation, and are a novel object representation that addresses goals (i-iv) stated in the abstract. In future work we are interested to explore new approaches to solving manipulation problems that exploit the dense visual information that learned dense descriptors provide, and how these dense descriptors can benefit other types of robot learning, e.g. learning how to grasp, manipulate and place a set of objects of interest.

Acknowledgments

The authors thank Duy-Nguyen Ta (calibration), Alex Alspach (hardware), Yunzhu Li (training techniques), Greg Izatt (robot software), and Pat Marion (perception and visualization) for their help. We also thank Tanner Schmidt for helpful comments in preparing the paper. This work was supported by: Army Research Office, Sponsor Award No. W911NF-15-1-0166; Draper Laboratory Incorporated, Sponsor Award No. SC001-0000001002; Lincoln Laboratory/Air Force, Sponsor Award No. 7000374874; Amazon Research Award, 2D-01029900; Lockheed Martin Corporation, Sponsor Award No. RPP2016-002. Views expressed in the chapter are not endorsed by the sponsors.

Chapter 4

kPAM: Keypoint Affordances for Robotic Manipulation

4.1 Introduction

This chapter focuses on pose-aware robotic pick and place at a category level. Contrary to single-instance pick and place, the manipulation policy should generalize to potentially unknown instances in the category with different shape, size, appearance, and topology. These tasks can be easily described using natural language, for example "put the mugs upright on the shelf," "hang the mugs on the rack by their handle" or "place the shoes onto the shoe rack." However, converting these intuitive descriptions into concrete robot actions remains a significant challenge. Accomplishing these types of tasks is of significant importance to both industrial applications and interactive assistant robots.

While a large body of work addresses robotic picking for arbitrary objects [38, 157, 54], existing methods have not demonstrated pick *and* place with an interpretable and generalizable approach. One way to achieve generalization at the object category level, and perhaps the most straightforward approach is to attempt to extend existing instance-level pick and place pipelines with category-level pose estimators [111, 140]. However, as detailed in Sec. 4.4, representing an object with a parameterized pose defined on a fixed geometric template, as these works do, may not adequately capture

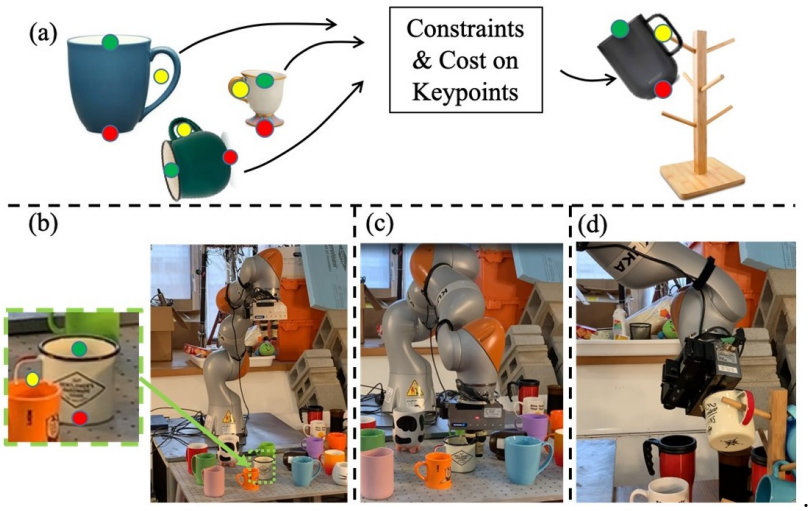


Figure 4-1: kPAM is a framework for defining and accomplishing category level manipulation tasks. The key distinction of kPAM is the use of semantic 3D keypoints as the object representation (a), which enables flexible specification of manipulation targets as geometric costs/constraints on keypoints. Using this framework we can handle wide intra-class shape variation (a) and reliably accomplish category-level manipulation tasks such as perceiving (b), grasping (c), and (d) placing any mug on a rack by its handle. A video demo for this task is available on our project page.

large intra-class shape or topology variations, and can lead to physically infeasible target pose for certain instances in the category. Other recent work has developed dense correspondence visual models, including at a category level, as a general representation for robot manipulation [31], but did not formulate how to specify and solve the task of manipulating objects into specific configurations. As a different route to address category-level pick and place, without an explicit object representation, [37] trains end-to-end policies in simulation to generalize across the object category. It is unclear, however, how to measure the reward function for this type of approach in a fully general way without an object representation that can adequately capture the human's intention for the task.

Contributions. Our main contribution is a novel formulation of the category-level pick and place task which uses semantic 3D keypoints as the object representation. This keypoint representation enables a simple and interpretable specification of the manipulation target as geometric costs and constraints on the keypoints, which flexibly generalizes existing pose-based manipulation targets. Using this formulation,

we contribute a manipulation pipeline that factors the problem into 1) instance segmentation, 2) 3D keypoint detection, 3) optimization-based robot action planning 4) geometric grasping and action execution. This factorization allows us to leverage well-established solutions for these submodules and combine them into a general and effective manipulation pipeline. The keypoint representation ignores task-irrelevant geometric details of the object, making our method robust to large intra-category shape and topology variations. We experimentally demonstrate the use of this keypoint representation with our manipulation pipeline on several category-level pick and place tasks implemented on real hardware. We show that our approach generalizes to novel objects in the category, and that this generalization is accurate enough to accomplish tasks requiring centimeter level precision.

Chapter Organization. In Sec. 4.2 we review related work. Sec. 4.3 describes our formulation of category-level manipulation tasks. The formulation is introduced in Sec. 4.3.1 using a concrete example, while Sec. 4.3.2 describes the general formulation. Sec. 4.4 compares our formulation with pose-based pick and place pipelines, highlighting the flexibility and generality of our method. Sec. 4.5 describes the results of hardware experiments on 3 different category-level manipulation tasks, specifically showing generalization to novel object instances. Sec. 4.6 discusses limitations and future work and Sec. 4.7 concludes.

4.2 Related Work

4.2.1 Object Representations and Perception for Manipulation

There exist a number of object representations, and methods for perceiving these representations, that have been demonstrated to be useful for robot manipulation. For a pick and place task involving a known object the standard solution starts by estimating the object's 6DOF pose. This allows the robot to then move the object from it's estimated pose to the specified target pose. Pose estimation is an extensively

studied topic in computer vision and robotics, and existing methods can be generally classified into geometry-based algorithms [90, 33] and learning-based approaches [135, 140, 111]. There exist several datasets [140, 146] annotated with aligned geometric templates, and pose estimators [111, 140] trained on these datasets can produce a category-level pose estimation. Consequently, a straightforward approach to category-level pose-aware manipulation is to combine single object pick and place pipelines with these perception systems. However, pose estimation can be ambiguous under large intra-category shape variations, and moving the object to the specified target pose for the geometric template can lead to incorrect or physically infeasible states for different instances within a category of objects. For example knowing the pose and size of a coffee mug relative to some canonical mug is not sufficient to successfully hang it on a rack by its handle. A more technical discussion is presented in Sec. 4.4.

Other work has developed and used representations that may be more generalizable than object-specific pose estimation. Recent work has demonstrated dense visual descriptors [112] as a fully self-supervised object representation for manipulation that can generalize at the category level [31]. In comparison with our present work based on 3D keypoints: (i) it is unclear how to extend dense visual descriptors to represent the full object configuration due to self-occlusions which would require N layers of occluded descriptors, (ii) the sparse keypoint representation may in practice be more effective at establishing task-relevant correspondence across significant topology variation, and (iii) correspondence alone may not fully define a class-general configuration-change manipulation task, but the addition of human-specified geometric costs and constraints on 3D keypoints may. Keypoints have also been used in prior works as components of manipulation pipelines. Several prior works demonstrate the manipulation of deformable objects, and keypoint detection plays a role in their respective perception pipelines. The detected keypoints are typically used as grasp points [73, 115] or building blocks for other shape parameterizations, e.g. the polygons in [136, 86, 85] on which the manipulation policy is defined. These approaches tackled various challenging manipulation tasks such as bed making and towel folding. In contrast, we propose a novel category-level manipulation target specification using costs and constraints

defined on 3D keypoints. Additionally the state-machine approach in [73, 115] and manipulation primitives of [136, 85] are specific to cloth and hence our manipulation task is out of scope for these approaches.

4.2.2 Grasping Algorithms

In recent years there have been significant advances in grasping algorithms that allow robots to reliably pick up a wide range of objects, including potentially unseen objects. Among various approaches for grasping, model-based methods [160, 70] typically rely on a pre-built grasp database of common 3D object models labeled with sets of feasible grasps. During execution, these methods associate the sensor input with an object entry in the database for grasp planning. In contrast, model-free methods [157, 38, 72] directly evaluate the grasp quality from raw sensor inputs. Many of these approaches achieved promising robustness and generality in the Amazon Picking Challenge [160, 114, 157]. Several works also incorporate object semantic information using instance masks [114], or non-rigid registrations [107] to accomplish tasks such as picking up a specific object or transferring a grasp pose to novel instances.

In this work we focus on category-level manipulation tasks which require placing novel instances of a category into desired goal states. Although the ability to reliably grasp an object is an important part of our manipulation pipeline, it doesn't help with the problem of deciding what to do with the object after it has been grasped. Thus the tasks that we consider are out of scope for the aforementioned grasping works.

4.2.3 End-to-End Reinforcement Learning

There have been impressive contributions [37, 5] in end-to-end reinforcement learning with applications to robotic manipulation. In particular, [37] has demonstrated robotic pick and place across different instances and is the most related to our work. These end-to-end methods encode a manipulation task into a reward function and train the policy using trial-and-error.

However, in order to accomplish the category level pose-aware manipulation task,

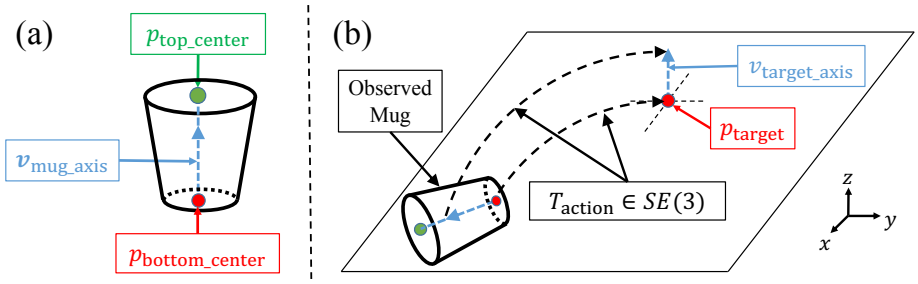


Figure 4-2: An overview of our manipulation formulation using the "put mugs upright on the table" task as an example: (a) we train a category level keypoint detector that produces two keypoints: $p_{\text{bottom\_center}}$ and $p_{\text{top\_center}}$. The axis of the mug $v_{\text{mug\_axis}}$ is a unit vector from $p_{\text{bottom\_center}}$ to $p_{\text{top\_center}}$. (b) Given an observed mug, its two keypoints on bottom center and top center are detected. The rigid transform T_{action}, which represents the robotic pick-and-place action, is solved to move the bottom center of the mug to the target location p_{target} and align the mug axis with the target direction $v_{\text{target\_axis}}$.

these end-to-end methods lack a general, flexible, and interpretable way to specify the desired configuration, which is required for the reward function. In [37], the target configuration is implemented specific to the demonstrated task and object category. Extending it to other desired configurations, object categories and tasks is not obvious. In this way, using end-to-end reinforcement learning allows the policy to be learned from experience without worrying about the details of shape variation, but only transfers the burden of shape variation to the choice and implementation of the reward function. We believe that our proposed object representation of 3D keypoints could be used as a solution to this problem.

4.3 Manipulation Formulation

In this section, we describe our formulation of the category level manipulation problem. Sec. 4.3.1 describes the approach using a concrete example while Sec. 4.3.2 presents the general formulation.

4.3.1 Concrete Motivating Example

Consider the task of "put the mug upright on the table". We want to come up with a manipulation policy that will accomplish this task for mugs with different size, shape, texture and topology.

To accomplish this task, we pick 2 semantic keypoints on the mugs: the bottom center $p_{\text{bottom\_center}}$ and the top center $p_{\text{top\_center}}$, as shown in Fig. 4-2 (a). Additionally, we assume we have a keypoint detector, discussed in Section 4.3.2, that takes as input raw observations (typically RGBD images or point clouds) and outputs the 3D locations of the specified keypoints. Note that there is no restriction that the keypoints be on the object surface, as evidenced by keypoint $p_{\text{top\_center}}$ in Fig. 4-2 (a). The 3D keypoints are usually expressed in the camera frame, but they can be transformed to an arbitrary frame using the known camera extrinsics. In the following text, we use $\mathbf{p} = \{p_i\}_{i=1}^{N} \in R^{3\times N}$ to denote the detected keypoint positions in world frame, where p_i is the i^{th} detected keypoint, and N is the total number of keypoints. In this example $N = 2$.

For robotic pick-and-place of mostly rigid objects, we represent the robot action as a rigid transform T_{action} on the manipulated object. Thus, the keypoints associated with the manipulated object will be transformed as $T_{\text{action}}\mathbf{p} \in R^{3\times N}$ using the robot action. In practice, this action T_{action} is implemented by first grasping the object using the algorithm detailed in Sec. 4.3.2 and then planning and executing a trajectory which ends with the object in the desired target location. This trajectory may require approaching the target from a specific direction, for example in the "mug upright on the table" task the mug must approach the table from above.

Given the above analysis, the manipulation task we want to accomplish can be formulated as finding a rigid transformation T_{action} such that

1. The transformed mug bottom center keypoint should be placed at some target location:

$$||T_{\text{action}} p_{\text{bottom\_center}} - p_{\text{target}}|| = 0 \tag{4.1}$$

2. The transformed direction from the mug bottom center to the top center should

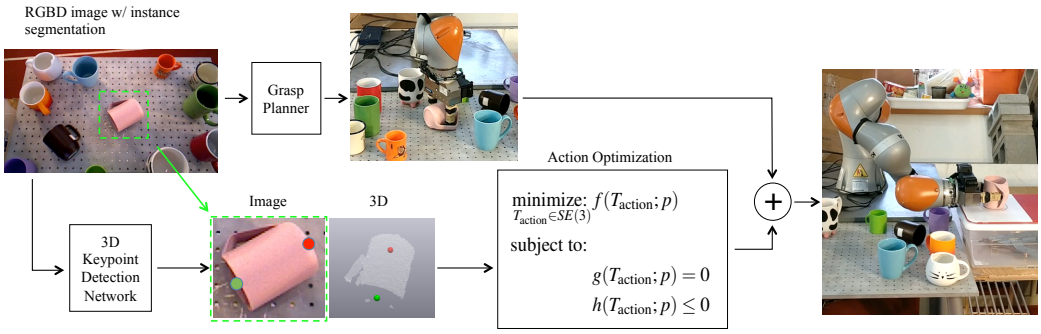


Figure 4-3: An overview of the category level pick and place pipeline using our manipulation formulation. Given a RGBD image with instance segmentation, the semantic 3D keypoints of the object in question are detected. We then feed these 3D keypoints into an optimization based planning algorithm to compute the robot pick and place actions, which is represented by a rigid transformation T_{action}. Finally, we use an object-agnostic grasp planner to pick up the object and apply the computed robot action.

be aligned with the upright direction. This is encoded by adding a cost to the objective function

$$||1 - \langle v_{\text{target\_axis}},\ \text{rot}(T_{\text{action}}) v_{\text{mug\_axis}} \rangle||^2 \tag{4.2}$$

where $\text{rot}(T)$ is the rotational component of the rigid transformation T, the target orientation $v_{\text{target\_axis}} = [0, 0, 1]^T$, and

$$v_{\text{mug\_axis}} = \frac{p_{\text{top\_center}} - p_{\text{bottom\_center}}}{||p_{\text{top\_center}} - p_{\text{bottom\_center}}||} \tag{4.3}$$

An illustration is presented in Fig. 4-2 (b). The above problem is an inverse kinematics problem with T_{action} as the decision variable, a constraint given by Eq. (4.1) and cost given by Eq. (4.2). This inverse kinematics problem can be reliably solved using off-the-shelf optimization solvers such as [132]. We then pick up the object using robotic grasping algorithms [72, 38] and execute a robot trajectory which applies the manipulation action T_{action} to the grasped object.

4.3.2 General Formulation

Given an arbitrary category-level manipulation task we propose to solve it in the following manner. First the modeler must specify a set of semantic 3D keypoints $\mathbf{p} = \{p_i\}_{i=1}^N$ for the category, together with a set of geometric costs and constraints on these keypoints which fully specify the task, e.g. Eq. (4.1)-(4.2). It is up to the modeler to choose the keypoints, costs and constraints that encode the task. This step can be seen as analogous to choosing costs and constraints in a trajectory optimization or planning method, or specifying a reward function in a reinforcement learning approach. The only restriction on the choice of the 3D keypoints is that they must be well defined for all instances of the category that might be encountered at test time. In particular the 3D keypoints need not lie on the object surface, as demonstrated by $p_{\text{top\_center}}$ in Fig. 4-6. The keypoints can also lie in regions of the object that may be occluded at test time, as exemplified by the $p_{\text{bottom\_center}}$ keypoint in Fig. 4-6 which is subject to self-occlusion when the mug is viewed from the side.

Once we have chosen keypoints together with geometric costs and constraints as the problem specification there exist natural formulations for each remaining piece of the manipulation pipeline. This enables us to factor the manipulation policy into 4 subproblems: 1) object instance segmentation 2) category level 3D keypoint detection, 3) a kinematic optimization problem to determine the manipulation action T_{action} and 4) grasping the object and executing the desired manipulation action T_{action}. An illustration of our complete manipulation pipeline is shown in Fig. 4-3. In the following sections, we describe each component of our manipulation pipeline in detail.

Instance Segmentation and Keypoint Detection As discussed in Section 4.3.1 the kPAM pipeline requires being able to detect category-level 3D keypoints from RGBD images of specific object instances. Here we present a specific approach we used to the keypoint detection problem, but note that any technique that can detect these 3D keypoints could be used instead.

We use the state-of-the-art integral network [129] for 3D keypoint detection. For each keypoint, the network produces a probability heatmap and a depth prediction

map as the raw outputs. The 2-D image coordinates and depth value are extracted using the integral operation [129]. The 3-D keypoints are recovered using the calibrated camera intrinsic parameters. These keypoints are then transformed into world frame using the camera extrinsics.

We collect the training data for keypoint detection using a pipeline similar to LabelFusion [75]. Given a scene containing the object of interest we first perform a 3D reconstruction. Then we manually label the keypoints on the 3D reconstruction. We note that this does not require pre-built object meshes. Keypoint locations in image space can be recovered by projecting the 3D keypoint annotations into the camera image using the known camera calibration. Training dataset statistics are provided in Fig. 4-7 (c). In total labeling our 117 training scenes took less than four hours of manual annotation time and resulted in over 100,000 labeled images. Even with this relatively small amount of human labeling time we were able to achieve centimeter accurate keypoint detections, enabling us to accomplish challenging tasks requiring high precision, see Section 4.5. More details on the keypoint detection network are contained in the supplementary material.

The keypoint detection network [129] requires object instance segmentation as the input, and we integrate Mask R-CNN [43] into our manipulation pipeline to accomplish this step. The training data mentioned above for the keypoint detector [129] can also be used to train the instance segmentation network [43]. Please refer to the supplemental material for more detail.

kPAM Optimization The optimization used to find the desired robot action T^*_{action} can in general be written as

$$\begin{aligned} &\underset{T_{\text{action}} \in SE(3)}{\text{minimize:}} \; f(T_{\text{action}}; \mathbf{p}) \\ &\text{subject to:} \\ &\qquad g(T_{\text{action}}; \mathbf{p}) = 0 \\ &\qquad h(T_{\text{action}}; \mathbf{p}) \leq 0 \end{aligned} \tag{4.4}$$

where f is a scalar cost function, g and h are the equality and inequality constraints,

respectively. The robot action T_{action} is the decision variable of the optimization problem, and the detected keypoint locations enter the optimization parametrically.

In addition to the constraints used in Sec. 4.3.1, a wide variety of costs and constraints can be used in the optimization (4.4). This allows the user to flexibly specify a large variety of manipulation tasks. Below we present several costs/constraints used in our experiments:

1. L2 distance cost between the transformed keypoint with its nominal target location:

$$||T_{\text{action}} p_i - p_{\text{target\_}i}||^2 \tag{4.5}$$

   This is a relaxation of the target position constraint presented in Sec. 4.3.1.

2. Half space constraint on the keypoint:

$$\langle n_{\text{plane}}, T_{\text{action}} p_i \rangle \leq b_{\text{plane}} \tag{4.6}$$

   where $n_{\text{plane}} \in R^3$ and $b_{\text{plane}} \in R$ defines the separating plane of the half space. Using the mug in Sec. 4.3.1 as an example, this constraint can be used to ensure all the keypoints are above the table to avoid penetration.

3. The point-to-plane distance cost of the keypoint

$$||\langle n_{\text{plane}}, T_{\text{action}} p_i \rangle - b_{\text{plane}}||^2 \tag{4.7}$$

   where $n_{\text{plane}} \in R^3$ and $b_{\text{plane}} \in R$ defines the plane that the keypoint p_i should be in contact with. By using this cost with keypoints that should be placed on the contact surface, for instance the $p_{\text{bottom\_center}}$ of the mug in Sec. 4.3.1, the optimization (4.4) can prevent the object from floating in the air.

4. The robot action T_{action} should be within the robot's workspace and avoid collisions.

Robot Grasping Robotic grasping algorithms, such as [38, 72], can be used to apply the abstracted robot action $T_{\text{action}} \in SE(3)$ produced by the kPAM optimization (4.4)

to the manipulated object. If the object is rigid and the grasp is tight (no relative motion between the gripper and object), applying a rigid transformation to the robot gripper will apply the same transformation to the manipulated object. These grasping algorithms [38, 70] are object-agnostic and can robustly generalize to novel instances within a given category.

For the purposes of this work we developed a grasp planner which uses the detected keypoints, together with local dense geometric information from a pointcloud, to find high quality grasps. This local geometric information is incorporated with an algorithm similar to the baseline method of [157]. In general the keypoints used to specify the manipulation task aren't sufficient to determine a good grasp on the object. Thus incorporating local dense geometric information from a depth image or pointcloud can be advantageous. This geometric information is readily available from the RGBD image used for keypoint detection, and doesn't require object meshes. Our grasp planner leverages the detected keypoints to reduce the search space of grasps, allowing us to focus our search on, for example, the heel of a shoe or the rim of a mug. Once we know which aspect of the local geometry to focus on, a high quality grasp can be found by any variety of geometric or learning-based grasping algorithms [38, 72].

We stress that keypoints are a sparse representation of the object sufficient for describing the manipulation task. However grasping, which depends on the detailed local geoemetry, can benefit from denser RGBD and pointcloud data. This doesn't detract from keypoints as an object representation for manipulation, but rather shows the benefits of different representations for different components of the manipulation pipeline.

4.4 Comparison and Discussions

In this section we compare our approach, as outlined in Sec. 4.3, to existing robotic pick and place methods that use pose as the object representation.

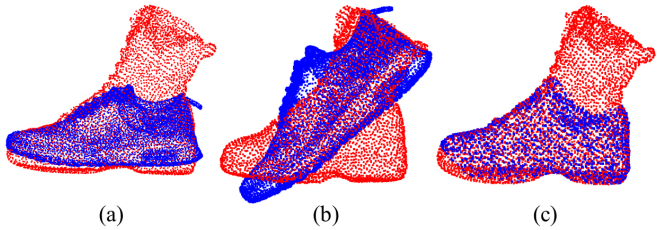


Figure 4-4: A pose representation cannot capture large intra-category variations. Here we show different alignment results from a shoe template (blue) to a boot observation (red). (a) and (b) are produced by [33] with variation on the random seed, and the estimated transformation consists of a rigid pose and a global scale. In (c), the estimated transformation is a fully non-rigid deformation field in [90]. In these examples, the shoe template and transformations can not capture the geometry of the boot observation. Additionally, there may exist multiple suboptimal alignments which make the pose estimator ambiguous. The subsequent robotic pick and place action from these estimations are different, despite these alignments being reasonable geometrically.

4.4.1 Keypoint Representation vs Pose Representation

At the foundation of existing pose-estimation methods is the assumption that the geometry of the object can be represented as a parameterized transformation defined on a fixed template. Commonly used parameterized pose families include rigid, affine, articulated or general deformable. For a given observation (typically an RGBD image or pointcloud), these pose estimators produce a parameterized transformation that aligns the geometric template to the observation.

However, the pose representation is not able to capture large intra-category shape variation. An illustration is presented in Fig. 4-4, where we try to align a shoe template (blue) to a boot observation (red). The alignments in Fig. 4-4 (a) and (b) are produced by [33] where the estimated transformation consists of a rigid pose and a global scale. Fig. 4-4 (c) is produced by [90] and the estimated transformation is a fully non-rigid deformation field. In these examples, the shoe template and transformations cannot capture the geometry of the boot observation. Additionally, there may exist multiple suboptimal alignments which make the pose estimator ambiguous, as shown in Fig. 4-4. Feeding these ambiguous estimations into a pose-based manipulation pipeline will produce different pick and place actions and final configurations of the manipulated

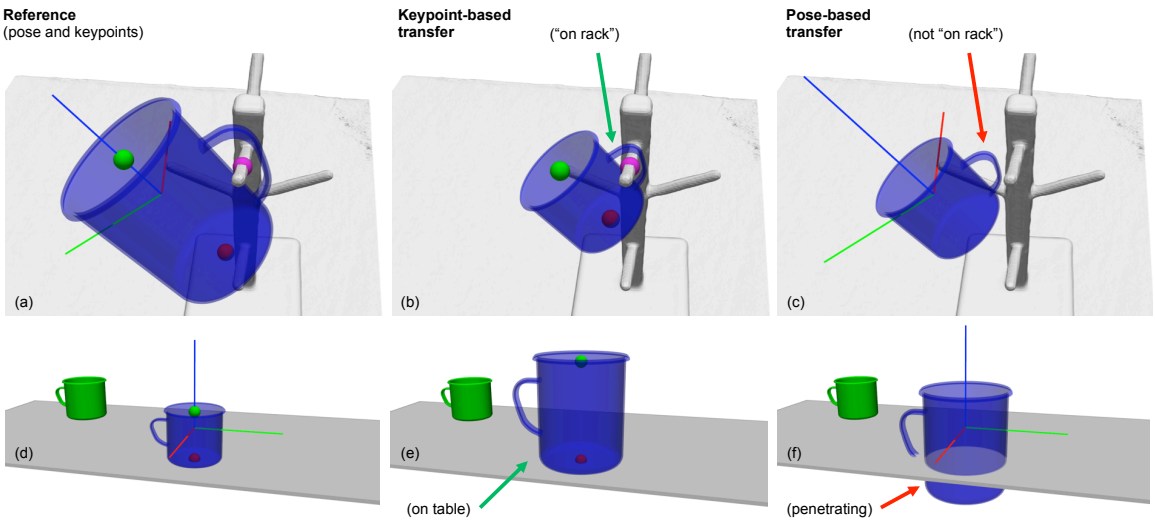


Figure 4-5: A comparison of the keypoint based manipulation with pose based manipulation for two different tasks involving mugs. The first row considers the mug on rack task, where a mug must be hung on a rack by its handle. (a) Shows a reference mug in the goal state, (b) and (c) show a scaled down mug instance that could be encountered at test time. (b) uses keypoint based optimization with a constraint on the handle keypoint to find the target state for the mug. The optimized goal state successfully achieves the task of hanging the mug on the rack. In contrast (c) shows the scaled mug instance at the pose defined by (a), which leads to the handle of the mug completely missing the rack, a failure of the task. The second row shows the task of putting a mug on a table. Again (a) shows a reference mug in a goal state, (b) - (c) show a scaled up mug that could be encountered at test time. (b) uses keypoint based optimization with costs/constraints on the bottom and top keypoints to place the mug in a valid goal state. (c) directly uses the pose from (a) on the new mug instance which leads to an invalid goal state where the mug is penetrating the table.

object.

In contrast, we use semantic 3D keypoints as a sparse but task-specific object representation for the manipulation task. Many existing works demonstrate accurate 3D keypoint detection that generalizes to novel instances within the category. We leverage these contributions to build a robust and flexible manipulation pipeline.

Conceptually, a pose representation can also be transformed into keypoint representation given keypoint annotations on the template. However, in practice the transformed keypoints can be inaccurate as the template and the pose cannot fully capture the geometry of new instances. Using the shoe keypoint annotation in Fig. 4-6 as an example, transforming the keypoints p_5 and p_6 to a boot using the shoe to boot

alignment in Fig 4-4 would result in erroneous keypoint detections. A general non-rigid kinematic model (and the associated estimator) that can handle large variations of shape and topology, such as in the example of Fig. 4-4, remains an open problem. Our method avoids this problem by sidestepping the geometric alignment phase and directly detecting the 3D keypoint locations.

4.4.2 Keypoint Target vs Pose Target

For existing pose-based pick and place pipelines, the manipulation task is defined as a target pose of the objects. For a given scene where the pose of each object has been estimated, these pipelines grasp the object in question and use the robot to move the objects from their current pose to the target pose.

The proposed method can be regarded as a generalization of the pose-based pick and place algorithms. If we detect 3 or more keypoints and assign their target positions as the manipulation goal, then this is equivalent to pose-based manipulation. In addition, our method can specify more flexible manipulation problems with explicit geometric constraints, such as the bottom of the cup must be on the table and its orientation must be aligned with the upright direction, see Sec. 4.3.1. The proposed method also naturally generalizes to other objects within the given category, as the keypoint representation ignores many task-irrelevant geometric details.

On the contrary a pose target is object-specific and defining a target pose at the category level can lead to manipulation actions that are physically infeasible. Consider the mug on table task from Section 4.3.1. Fig. 4-5 (d) shows the target pose for the reference mug model. Directly applying this pose to the scaled mug instance in Fig. 4-5 (f) leads to physically infeasible state where the mug is penetrating the table. In contrast, using the optimization formulation of Section 4.3 results in the mug resting stably on the table, shown in Fig. 4-5 (e).

In addition to leading to states which are physically infeasible, pose-based targets at a category level can also lead to poses which are physically feasible but fail to accomplish the manipulation task. Figures 4-5 (a) - (c) show the mug on rack task. In this task the goal is to hang a mug on a rack by its handle. Fig. 4-5 (a) shows

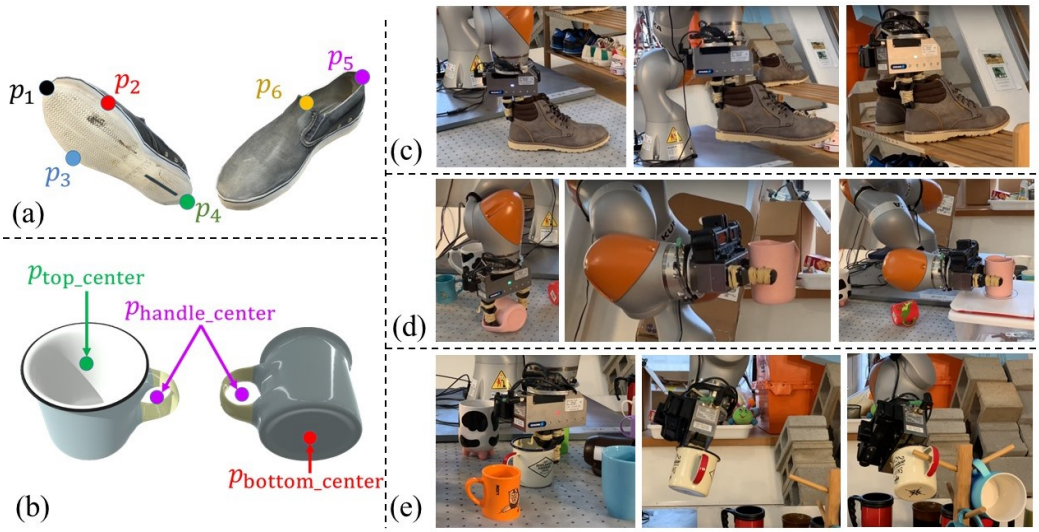


Figure 4-6: An overview of our experiments. (a) and (b) are the semantic keypoints we used for the manipulation of shoes and mugs. We use three manipulation tasks to evaluate our pipeline: (c) put shoes on a shelf; (d) put mugs on a mug shelf; (e) hang mugs on a rack by the mug handles. The video of these experiments are available on our project page.

the reference model in the goal state. Fig. 4-5 (c) shows the result of applying the pose based target to the scaled down mug instance. As can be seen even though the pose unambiguously matches the target pose exactly, this state doesn't accomplish the manipulation task since the mug handle completely misses the rack. Fig. 4-5 (b) shows the result of our kPAM approach. Simply by adding a constraint that handle center keypoint should be on the rack, a valid goal state is returned by the kPAM optimization.

4.5 Results

In this section, we demonstrate a variety of pose-aware pick and place tasks using our keypoint-based manipulation pipeline. The particular novelty of these demonstrations is that our method is able to handle large intra-category variations without any instance-wise tuning or specification. We utilize a 7-DOF robot arm (Kuka IIWA LBR) mounted with a Schunk WSG 50 parallel jaw gripper. An RGBD sensor (Primesense Carmine 1.09) is also mounted on the end effector. The video demo on

(a) Test Shoes

(b) Test Mugs

Object Type	# train objects	# scenes	# images
Shoe	10	43	39,403
Mug	21	74	70,094

(c) Training dataset statistics

# test objects	# Trials	Placed on shelf	Heel Error (cm)	Toe Error (cm)
20	100	98%	1.09 ± (1.29)	4.34 ± (3.05)

(d) Shoes on Rack

Initial Orientation	# test objects	# Trials	Placed upright on shelf	< 3cm error	< 5cm error
Upright	40	80	100%	97.5%	100%
Horizontal	19	38	97.3%	89.4%	94.7%

(e) Mugs on Shelf

Mug Size	# test objects	# Trials	Success Rate
Regular	25	100	100%
Small	5	20	50%

(f) Mugs on Rack

Figure 4-7: Quantitative results from the 3 hardware experiments. (a) and (b) show some of the test objects for the experiments. (c) statistics of the training data (d) We report the average heel and toe errors (along the horizontal direction) from their desired locations as well as the standard deviation. (e) The reported errors for the mug on shelf task are the distance from the bottom center keypoint to the target location of that keypoint in the optimization program. (f) reports success rates for the mug on rack task for different sized mugs. Mugs with handles having either height or width less than 2cm are classified as "small" (more details in supplementary material). A trial was deemed successful if the mug ended up hanging on the rack by the mug handle. Videos of the experiments are available on our project page.

our project page best demonstrates our solution to these tasks. More details about the experimental setup are included in the supplemental material.

4.5.1 Put shoes on a shoe rack

Task Description

Task Description Our first manipulation task is to put shoes on a shoe rack, as shown in Fig. 4-6 (c). We use shoes with different appearance and geometry to evaluate the generality and robustness of our manipulation policy. The six keypoints used in this manipulation task are illustrated in Fig 4-6 (a), and the costs and constraints in the optimization (4.4) are

1. The L2 distance cost (4.5) between keypoints p_1, p_2, p_3 and p_4 to their nominal target locations.

2. The sole of the shoe should be in contact with the rack surface. In particular, the point-to-plane cost (4.7) is used to penalize the deviation of keypoints p_2, p_3 and p_4 from the supporting surface.

3. All the keypoints should be above the supporting surface to avoid penetration. A half-space constraint (4.6) is used to enforce this condition.

For our experiments we place the shoe rack in a known position, but this constraint could be easily relaxed by adding a pose-estimation module for the shoe-rack.

Experimental Results The shoe keypoint detection network was trained on a labeled dataset of 10 shoes, detailed in Figure 4-7 (c). Experiments were conducted with a held out test set of 20 shoes with large variations in shape, size and visual appearance (more details in the video and supplemental material). For each shoe we ran 5 trials of the manipulation task. Each trial consisted of a single shoe being placed on the table in front of the robot. Using the kPAM pipeline the robot would pick up the shoe and place it on a shoe rack. The shoe rack was marked so that the horizontal deviation of the shoe's toe and heel bottom keypoints (p_1 and p_4 respectively in Fig. 4-6) from their nominal target locations could be determined. Quantitative results are given in Fig. 4-7 (d). Out of 100 trials only twice did the pipeline fail to place the shoe on the rack. Both failures were due to inaccurate keypoint detections. One led to a failed grasp and another to an incorrect T_{action}. For trials which ended up with the shoe on the rack average errors for the heel and toe keypoint locations are given in Fig. 4-7 (d). During the course of our experiments we noticed that the majority of these errors come from the fact that when the robot grasps the shoe by the heel the closing of the gripper often results in the object shifting from the position it was in when the RGBD image used for keypoint detection was captured. This accounts for the majority of the errors observed in the final heel and toe keypoint locations. The keypoint detections and resulting T_{action} would have almost always results in heel and toe errors of less than 1 cm if we were able to exactly apply T_{action}

to the object. Since our experimental setup relies on a wrist mounted camera we are not able to re-perceive the object after grasping it. We believe that these errors could be further reduced by adding an external camera that would allow us to re-run our keypoint detection after grasping the object to account for any object movement during the grasp. Overall kPAM approach was very successful at the shoes on rack task with a greater than 97% success rate.

4.5.2 Put mugs upright on a shelf

Task Description We also perform a real-world demonstration of the "put mugs upright on a shelf" task described in Sec. 4.3.1, as shown in Fig. 4-6 (d). The keypoints used in this task are illustrated in Fig. 4-6 (b). The costs and constraints for this task include the target position constraint (4.1) and the axis alignment constraint (4.2). This task is very similar to the mugs task in [37]. For this set of experiments we didn't place any costs or constraints on the yaw degree of freedom of the mug, but if a specific yaw orientation was desired this could be incorporated by adding an L2 cost (4.5) between the $p_{\text{handle\_center}}$ keypoint with its target location.

Experimental Results The mug keypoint detection network was trained on a dataset of 21 standard sized mugs, detailed in Fig. 4-7 (c). Experiments for the mug on shelf task were conducted using a held out test set of 40 mugs with large variations in shape, size and visual appearance (more details in the video and supplemental material). All mugs could be grasped when in the upright orientation, but due to the limited stroke of our gripper (7.5cm when fully open) only 19 of these mugs could be grasped when lying horizontally. For mugs in that could be grasped horizontally we ran two trials with the mug starting from a horizontal orientation, and two trials with the mug in a vertical orientation. For the remaining mugs we ran two trials for each mug with the mug starting in an upright orientation. Quantitative performance was evaluated by recording whether the mug ended up upright on the shelf, and the distance of the mug's bottom center keypoint to the target location. Results are shown in Fig. 4-7 (e). Overall our system was very reliable, managing to place the mug on the shelf within 5cm of the target location in all but 2 trials. In one of these failures

the mug was placed upside down. In this case the mug was laying horizontally on the table and the RGB image used in keypoint detection (see Fig. B-7 in the Appendix) was taken from a side-on profile where the handle is occluded and it is very difficult to distinguish the top from the bottom of the mug. This led our keypoint detector to mix up the top and bottom of the mug, causing it to be placed upside down. The keypoint detection error is understandable in this case since it is very difficult to distinguish the top from the bottom of this mug in the single RGBD image. In addition this particular instance was a small kids sized mug, whereas all the training data for mugs contained only regular sized mugs. See Section 4.6 for more discussion on this failure.

Overall the accuracy in the mug on shelf task was very high, with 97% of upright trials, and 88% of horizontal trials resulting in bottom keypoint final location errors of less than 3cm. Qualitatively the majority of this error arose from the object moving slightly during the grasping process with the rest attributed to the keypoint detection.

4.5.3 Hang the mugs on the rack by their handles

Task Description To demonstrate the accuracy and robustness of our method we tasked the robot with autonomously hanging mugs on a rack by their handle. An illustration of this task is provided in Fig. 4-6 (e). The relatively small mug handles (2-3 centimeters) challenge the accuracy of our manipulation pipeline. The costs and constraints in this task are

1. The target location constraint (4.1) between $p_{\text{handle\_center}}$ to its target location on the rack axis.

2. The keypoint L2 distance cost (4.5) from $p_{\text{top\_center}}$ and $p_{\text{bottom\_center}}$ to their nominal target locations.

In order to avoid collisions between the mug and an intermediate goal for the mug was specified. Using the notation of (4.4) this intermediate goal T_{approach} was gotten by shifting T_{action} away from the rack by 10cm along the direction of the target peg. We then executed the final placement by moving the end effector in a straight line

connecting T_{approach} to T_{action}. During these experiments the mug-rack was placed in a fixed known position, but this constraints be easily relaxed by adding a pose-estimation module for the mug-rack.

Experimental Results For the mug on rack experiments we used the same keypoint detection network as for the mug on shelf experiments. Experiments were conducted using a held out test set of 30 mugs with large variation in shape, texture and topology. Of these 5 were very small mugs whose handles had a minimum dimension (either height or width) of less than 2cm (see the supplementary material for more details). We note that the training data did not contain any such "small" mugs. Each trial consisted of placing a single mug on the table in front of the robot. Then the kPAM pipeline was run and a trial was recorded as successful if the mug ended up hanging on the rack by its handle. Five trials were run for each mug and quantitative results are reported in Fig. 4-7 (e). For regular sized mugs we were able to hang them on the rack with a 100% success rate. The small mugs were much more challenging but we still achieved a 50% success rate. The small mugs have very tiny handles, which stresses the accuracy of the entire system. In particular the total error of the keypoint detection, grasping and execution needed to successfully complete the task for the small mugs was on the order of 1-1.5 cm. Two main factors contributed to failures in the mug on rack task. The first, similar to the case of shoe on rack task, is that during grasping the closing of the gripper often moves the object from the location at which it was perceived. Even a small disturbance (i.e. $<$ 1cm) can lead to a failure in the mug on rack task since the required tolerances are very small. The second contributing factor to failures is inaccurate keypoint detections. Again an inaccurate detection of even 0.5-1cm can be sufficient for the mug handle to miss the rack entirely. As discussed previously, the movement of the object during grasping could be alleviated by the addition of an external camera that would allow us to re-perceive the object after grasping.

4.6 Limitations and Future Work

Our current data collection pipeline in Sec. 4.3.2 requires human annotation, although the use of 3D reconstruction somewhat alleviates this manual labor. An interesting direction for future work is to train our keypoint detector using synthetic data, as demonstrated in [135, 140].

Representing the robot action with a rigid transformation T_{action} is valid for robotic pick-and-place. However, this abstraction does not work for deformable objects or more dexterous manipulation actions on rigid objects, such as the in-hand manipulation in [5]. Combining these learning-based or model-based approaches with the keypoint representation to build a manipulation policy that generalizes to categories of objects would is a promising direction for future work. In addition to the usage in our pipeline, we believe that the keypoint representation can potentially contribute to various learning-based manipulation approaches as 1) a reward function to flexibly specify the manipulation target or 2) an alternative input to the policy/value neural network, which is more robust to shape variation and large deformation than the widely-used pose representation.

4.7 Conclusion

In this chapter we contribute a novel formulation of category-level manipulation which uses semantic 3D keypoints as the object representation. Using keypoints to represent the object enables us to simply and interpretably specify the manipulation target as geometric costs and constraints on the keypoints, which flexibly generalizes existing pose-based manipulation methods. This formulation naturally allows us to factor the manipulation policy into the 3D keypoint detection, optimization-based robot action planning and grasping based action execution. By factoring the problem we are able to leverage advances in these sub-problems and combine them into a general and effective perception-to-action manipulation pipeline. Through extensive hardware experiments, we demonstrate that our pipeline is robust to large intra-category shape

variation and can accomplish manipulation tasks requiring centimeter level precision.

Acknowledgements

The authors thank Ethan Weber (instance segmentation training data generation) and Pat Marion (visualization) for their help. This work was supported by: National Science Foundation, Award No. IIS-1427050; Draper Laboratory Incorporated, Award No. SC001-0000001002; Lockheed Martin Corporation, Award No. RPP2016-002; Amazon Research Award.

Chapter 5

Self-Supervised Correspondence in Visuomotor Policy Learning

5.1 Introduction

To achieve general-purpose manipulation skills, robots will need to use vision-based policies and learn new tasks in a scalable fashion with limited human supervision. For visual training, prior work has often used methods such as end-to-end training [64], autoencoding [27], and pose-based losses [64, 162]. These methods, however, have not benefitted from the rich sources of self-supervision that may be provided by dense three-dimensional computer vision techniques [69, 100, 112], for example correspondence learning which robots can automate without human input [31].

Correspondence is fundamental in computer vision, and we believe it has fundamental usefulness for robots learning complex tasks requiring visual feedback. In this chapter we introduce using self-supervised correspondence for visuomotor policies, and our results suggest this enables policy learning that is surprisingly capable. Our evaluations pair correspondence training with a simple imitation learning objective, and extensive hardware validation shows that learned policies can address challenging scenarios: manipulating deformable objects, generalizing across a class of objects, and visual challenges such as textureless objects, clutter, moderate occlusion, and lighting variation (Fig 5-1). Additionally our simulation-based comparisons empirically suggest

that our method offers significant generalization and sample complexity advantages compared to existing methods for training visuomotor policies, while requiring no additional human supervision. To bound our method's scope: while spatial correspondence alone cannot suffice for all tasks (for example, it cannot discriminate when to be finished cooking eggs), there is a wide set of tasks for which dense spatial correspondence may be useful: essentially any spatial manipulation task.

5.1.1 Contributions

Our primary contribution is (i) a novel formulation of visuomotor policy learning using self-supervised correspondence. Through simulation experiments (ii) we measure that this approach offers sample complexity and generalization benefits compared to a variety of baselines, and (iii) we validate our method in real world experiments. We believe that compared to the existing state of the art in robotic manipulation, the abilities of our learned policies represent exciting levels of performance, especially the generalization across challenging scenarios (category-level manipulation, deformable objects, visually challenging scenes) and with limited data (between 50 and 150 demonstrations). We also (iv) introduce a novel data augmentation technique for behavior cloning, and (v) demonstrate a new technique for multi-camera time-synchronized dense spatial correspondence learning.

5.2 Related Work

We focus our related work review around two topics: visual training methods for visuomotor policies, and approaches for providing the policy learning signal. An overview of the broader topic of robot learning in manipulation is provided in [58]. For more related work in self-supervised robotic visual learning, including correspondence learning, we refer to the reader to [31].

Figure 5-1: Examples of autonomous policies, including a variety of non-prehensile, class-general, and deformable manipulation. Table 5.3 details hardware results.

5.2.1 Visual Training Methods for Visuomotor Policies

There have been three primary methods used in the robot learning literature to train the visual portion of visuomotor policies. Often these methods are used together – for example [64, 162] use pose-based losses together with end-to-end training. **(1) End-to-End training.** This approach can be applied to any learning signal that is formed as a consequence of a robot's actions, for example through imitation learning or reinforcement learning. While often end-to-end training is complemented with other learning signals, other works use purely end-to-end training. **(2) Autoencoders.** Autoencoding can be applied to any data with no supervision and is commonly used to aid visuomotor policy learning [27, 150, 25, 35, 137, 103]. Sometimes polices are learned with a frozen encoder [27, 150, 25], other times in conjunction with end-to-end training [103]. **(3) Pose-based losses.** In [64], for example, a separate dataset is collected of the robot holding objects, and assuming that the objects are rigid and graspable, then using the robot's encoders and forward kinematics the visual model can be trained to predict the object pose. In [162], pose-based auxiliary losses are used regardless of whether or not objects are held – we wouldn't expect this to learn how to predict object configurations unless they are also rigid and grasped. Simulation-based works [51] have also used auxiliary losses for object and gripper positions.

In our comparison experimentation, we include end-to-end training and autoencoding, but not pose-based losses, since they are not applicable to deformable or un-graspable objects. While the above are three of the most popular, other visual training methods include: training observation dynamics models [2, 20], using time-contrastive learning [119], or using no visual training and instead using only generic pre-trained visual features [117]. Relevant concurrent works include [59] which proposes autoencoder-style visual training but with a reference image and novel architecture, and [121] which proposes a graph-based reward function using a fixed set of correspondences.

5.2.2 Methods for Learning Vision-Based Closed-Loop Policies

While the previous section discussed visual training methods, to acquire policies they must be paired with a policy learning signal. We are particularly interested in approaches that can (i) scalably address a wide variety of tasks with potentially deformable and unknown objects, (ii) use a small incremental amount of human effort (on the order of 1 human-hour) per each new object or task, and (iii) produce real-time vision-based closed-loop policies.

One source of policy learning signal may be from reinforcement learning, which has demonstrated many compelling results. A primary challenge, however, is the difficulty of measuring rewards in the real world. Some tasks such as grasping can be self-supervised [101], and other tasks can leverage assumptions that objects are grasped and rigid in order to compute rewards [64], but this only applies to a subset of tasks. A more generalizable direction may be offered by unsupervised methods of obtaining reward signals [27, 117, 119]. Another direction which has shown promising results is using sim-to-real transfer [51, 165, 82, 5], but our interest in a small amount of incremental human effort per new task is challenging for these methods, since they currently require significant engineering effort for each new simulation scenario.

Another powerful source of signal may come from imitation learning from demonstration, which several recent works have shown promise in using to produce real-time vision-based closed-loop policies [150, 25, 117, 28, 103, 162, 154, 119]. We point the reader to a number of existing reviews of learning from demonstration [6, 8]. Another direction may be to learn models from observations and specify goals via observations [27, 2, 20], but these may be limited to tasks for which autonomous exploration has a reasonable chance of success. In terms of limitations of these prior works, one primary challenge relates to reliability and sample complexity – it is not clear how much data and training would be required in order to achieve any given level of reliability. Relatedly, a second limitation is that little work has characterized the distributions over which these methods should be trained and subsequently expected to generalize.

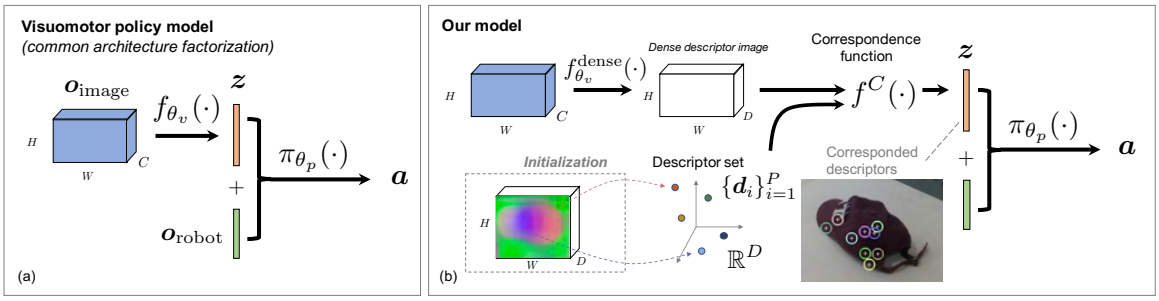


Figure 5-2: Diagram of common visuomotor policy factorization (a), and our proposed model (b) using visual models trained on correspondence.

Third, like in many areas of robotics it is difficult to reproduce results and compare approaches on a common set of metrics. While we believe hardware validation is critical, we also believe that increased effort should be put into simulation-based results that compare methods and can be reproduced.

5.3 Visuomotor Formulation

First as preliminary we identify some primary attributes of existing approaches in visuomotor policy learning (Sec. 5.3.1). We then present our approach based on self-supervised correspondence (Sec. 5.3.2). The discussion of visuomotor policy learning in this section is agnostic to the specific learning algorithm, i.e. reinforcement learning, imitation learning, etc., and focuses on the model structure and sets of trainable parameters. Sec. 5.4 discusses the application of our approach to a specific case of imitation learning.

5.3.1 Preliminary: Visuomotor Policies

We would like to have a policy $\boldsymbol{a}_t = \pi_\theta(\boldsymbol{o}_{0:t})$, where $\boldsymbol{o}_{0:t} = (\boldsymbol{o}_0, \boldsymbol{o}_1, \ldots \boldsymbol{o}_t)$ is the full sequence of the robot's observations during some episode up until time t, with each $\boldsymbol{o}_i \in \mathcal{O}$, the robot's observation space. This sequence of observations is mapped by $\pi_\theta(\cdot)$, the robot's policy parameterized by θ, to the robot's actions $\boldsymbol{a}$, each $\in \mathcal{A}$. In particular, we are interested in visuomotor policies in which the observation space contains high-dimensional images $\mathcal{O}_{\text{image}} \subset \mathcal{O}$, for example $\mathcal{O}_{\text{image}} = \mathbb{R}^{W \times H \times C}$ for a

C-channel, width W, and height H image. The visual data is perhaps complemented with additional lower-dimensional measurements $\mathcal{O}_{\text{robot}}$, such as produced from sensors like the robot's encoders, such that $\mathcal{O}_{\text{robot}} \times \mathcal{O}_{\text{image}} = \mathcal{O}$.

It is common for a visuomotor policy to have an architecture that can factored as displayed in Fig. 5-2(a),

$$\boldsymbol{z} = f_{\theta_v}(\boldsymbol{o}_{\text{image}}) : \boldsymbol{o}_{\text{image}} \in \mathcal{O}_{\text{image}},\ \boldsymbol{z} \in \mathbb{R}^Z \tag{5.1}$$

$$\boldsymbol{a} = \pi_{\theta_p}(\boldsymbol{z}, \boldsymbol{o}_{\text{robot}}) : \boldsymbol{o}_{\text{robot}} \in \mathcal{O}_{\text{robot}},\ \boldsymbol{z} \in \mathbb{R}^Z,\ \boldsymbol{a} \in \mathcal{A} \tag{5.2}$$

in which a visual model $f_{\theta_v}(\cdot)$, parameterized by θ_v, processes the high-dimensional $\boldsymbol{o}_{\text{image}}$ into a much smaller Z-dimensional representation $\boldsymbol{z}$. The policy model $\pi_{\theta_p}(\cdot)$ then combines the output of the visual model with other observations $\boldsymbol{o}_{\text{robot}}$. This is a practical modeling choice – images are extremely high dimensional, i.e. in this work we use images in $\mathbb{R}^{640\times480\times3} = \mathbb{R}^{921{,}600}$, whereas our $\mathcal{O}_{\text{robot}}$ is at most $\mathbb{R}^{13}$. A wide variety of works have employed a similar architecture to [64], consisting of convolutional networks extracting features from raw images into an approximately $Z = 32$ to 100 bottleneck representation of features, e.g. [27, 103, 28, 147, 35, 162]

5.3.2 Visual Correspondence Models for Visuomotor Policy Learning

The objective of the visual model is to produce a feature vector $\boldsymbol{z}$ which serves as a suitable input for policy learning. In particular, we are interested in deploying policies that can operate directly on RGB images. Given the role that pose estimation has played in traditional manipulation pipelines it seems valuable to encode the configuration of objects of interest in the vector $\boldsymbol{z}$. Pose estimation, however, doesn't extend to the cases of deformable or unknown objects. Some of the prior works discussed in Sec. 5.2.1, for example [64, 27], have interpreted their learned feature points $\boldsymbol{z}$ as encoding useful spatial information for the objects and task. These feature points are learned via the supervisory signals of end-to-end, pose-based, or

autoencoding losses, and don't explicitly train for spatial correspondence. In contrast our approach is to directly employ visual correspondence training, building off the approach of [31] which can in a self-supervised manner, learn pixel descriptors of objects that are effective in finding correspondences between RGB images.

We introduce four different methods for how to employ dense correspondence models as the visual basis of visuomotor policy learning. The first three are based on the idea of a set of points on the object(s) that are localized either in image-space or 3D space. We represent these points as a set $\{\boldsymbol{d}_i\}_{i=1}^{P}$ of P descriptors, with each $\boldsymbol{d}_i \in \mathbb{R}^D$ representing some vector in the D-dimensional descriptor space produced by a dense descriptor model $f_{\theta_v}^{\text{dense}}(\cdot)$. This $f_{\theta_v}^{\text{dense}}(\cdot)$, a deep CNN, maps a full-resolution RGB image, $\mathbb{R}^{W \times H \times 3}$, to a full-resolution descriptor image, $\mathbb{R}^{W \times H \times D}$. Let us term $f^C(\cdot)$ to be the non-parametric correspondence function that, given one or more descriptors and a dense descriptor image $f_{\theta_v}^{\text{dense}}(\boldsymbol{o}_{\text{image}})$, provides the predicted location of the descriptor(s):

$$\boldsymbol{z} = f^C\big(f_{\theta_v}^{\text{dense}}(\boldsymbol{o}_{\text{image}}), \{\boldsymbol{d}_i\}_{i=1}^{P}\big) \tag{5.3}$$

Specifically $f^C : \mathbb{R}^{W \times H \times D} \times \mathbb{R}^{P \times D} \rightarrow \mathbb{R}^{P \times K}$, where $K = 2$ corresponds to: $\boldsymbol{z}$ is the predicted corresponding (u, v) pixel coordinates of each descriptor in the image, while $K = 3$ is their predicted 3D coordinates. All four methods optimize a generic policy-based loss function, shown in Eq. (5.4), and vary only in the set of learnable parameters Θ and how $\boldsymbol{z}$ is acquired (the first three use Eq. 5.3). This loss function $\mathcal{L}$ is generic and could represent any approach for learning the parameters of a visuomotor policy.

$$\min_{\Theta} \; \mathcal{L}\Big(\pi_{\theta_p}\big(\boldsymbol{z}, \boldsymbol{o}_{\text{robot}}\big)\Big) \tag{5.4}$$

Fixed Descriptor Set. This method only optimizes the policy parameters,

The specific form of $f^C(\cdot)$ is defined by how the correspondence model was trained. In our preferred model we compute a spatial-expectation using a correspondence kernel, either in image-space or 3D. See [30], Chapter 4, for details.

$\Theta = \{\theta_p\}$. In this case both the set of descriptors $\{\boldsymbol{d}_i\}_{i=1}^P$ and visual model $f_{\theta_v}^{\text{dense}}(\cdot)$ are fixed. We use a simple initialization scheme of sampling $\{\boldsymbol{d}_i\}$ from a single masked reference descriptor image. While we have found this method to be surprisingly effective, it is unsatisfying that the visual model's representation is not optimized after the random initialization process.

Descriptor Set Optimization. This method optimizes the descriptor set $\{\boldsymbol{d}_i\}_{i=1}^P$ along with the policy parameters θ_p while keeping the dense descriptor mapping $f_{\theta_v}^{dense}$ fixed. Intuitively $f_{\theta_v}^{dense}$ has already been trained to perform correspondence, and we are simply allowing the policy optimization to choose *what to correspond*. We have observed that Descriptor Set Optimization can improve validation error in some cases over a Fixed Descriptor Set, and adds minimal computational cost and parameters.

End-to-End Dense Optimization. The third option is to train the full model architecture end-to-end by including θ_v in the optimization. While we may have expected this approach to allow the visual model to more precisely focus its modeling ability on task-critical parts of images, we so far have not observed a performance advantage of this approach over Descriptor Set Optimization.

End-to-End with Correspondence Pretraining. The fourth option is to directly apply a differentiable operation to a model which was previously trained on dense correspondence. We can apply any differentiable operation $g(\cdot)$ on top of $f_{\theta_v}^{\text{dense}}$ directly to produce a representation $\boldsymbol{z} = g\big(f_{\theta_v}^{dense}(\boldsymbol{o}_{\text{image}})\big)$. For example, we can apply non-parametric channel-wise spatial expectations to each of the D channels of the dense descriptor images. The optimization variables in this case are $\Theta = \{\theta_p, \theta_v\}$.

For our $f_{\theta_v}^{\text{dense}}$ we use a 34-layer ResNet, as in [31], which is a powerful vision backbone. Accordingly, using either a fixed- or optimized- descriptor set will significantly increase policy training speed, since it does not require forward-backward optimizing through a very deep convolutional network in each step of policy training, which in our case is 1 to 2 orders of magnitude faster.

5.4 Visual Imitation Formulation

We now propose how to use the general approach of Sec. 5.3.2 for a specific type of imitation learning for robot manipulation.

5.4.1 Robot Observation and Action Spaces

At the lowest level our controller sends joint velocity commands to the robot. For ease of providing demonstrations via teleoperation, the operator commands relative-to-current desired end-effector poses $T_{\Delta,\text{cmd}}$. A low-level Jacobian based controller then tracks these end-effector pose setpoints. Our learned policies also output $T_{\Delta,\text{cmd}}$. The teleoperator also commands a gripper width setpoint which again is tracked by a low-level controller. Thus the action space is $\boldsymbol{a} = (T_{\Delta,\text{cmd}}, w_{\text{gripper}}) \in \mathcal{A} = SE(3) \times \mathbb{R}^+$.

Our $\boldsymbol{o}_{\text{robot}} \in \mathbb{R}^{13}$ is (i) three 3D points on the hand as in [64], (ii) an axis-angle rotation relative to the task's starting pose, and (iii) the gripper width. As noted previously, $\boldsymbol{o}_{\text{image}} \in \mathbb{R}^{921,600}$.

5.4.2 Imitation Learning Visuomotor Policies

To evaluate visual learning strategies for enabling visuomotor policy learning, we use imitation learning via a simple behavioral cloning [102] strategy, which a few recent works have demonstrated to be viable for learning visuomotor manipulation policies [103, 162]. Optimizing a policy with parameters Θ on the behavioral cloning objective, given a dataset of N_{train} trajectories of observation-sequence-to-action pairs $\{(\boldsymbol{o}_t, \boldsymbol{a}_t^*)\}_{t=0}^{T_i}$ can be written as:

$$\min_{\Theta} \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \sum_{t=0}^{T_i} \mathcal{L}_{\text{BC}}\big(\boldsymbol{a}_t^*, \pi_\Theta(\boldsymbol{o}_t)\big) \tag{5.5}$$

For our loss function we use a simple weighted sum of l_1 and l_2 loss, $\mathcal{L}_{\text{BC}}(\cdot) = ||\boldsymbol{a}^* - \pi(\cdot)||_2^2 + \lambda||\boldsymbol{a}^* - \pi(\cdot)||_1$ where we use $\lambda = 0.1$. We scale $\boldsymbol{a}^*$ to equalize 1.0m end-effector translation, 0.1 radians end-effector rotation, and 1.0m gripper translation.

5.4.3 Training for Feedback through Data Augmentation

We introduce a simple technique which we have found to be effective in at least partially addressing a primary issue in imitation learning: the issue of cascading errors [109]. While other works have shown that injecting noise into the dynamics either during imitation learning [63] or sim-to-real transfer [99] can alleviate cascading errors, we provide a simple method based only on data augmentation. This method does not address recovering from discrete changes in the environment, but can address local feedback stabilization.

Consider the output of our policy in a global frame, $\boldsymbol{a} = (T_{\text{cmd}}, w_{\text{gripper}})$, which we can acquire from $T_{\Delta,\text{cmd}}$ since we know the end-effector pose. As previously mentioned a low-level controller tracks these setpoints, thus our learned policies can stabilize a trajectory by commanding the same global-frame setpoint $\boldsymbol{a}$ in the face of small disturbances to the robot state. If we want our policy to command the same setpoint in the face of a slightly perturbed robot state $\tilde{\boldsymbol{o}}_{\text{robot}}$ we can simply use $((\boldsymbol{o}_{\text{image}}, \tilde{\boldsymbol{o}}_{\text{robot}}), \boldsymbol{a})$ as an observation-action pair. These noise-augmented observation-action pairs are generated on-the-fly during training. A remaining question, of course, is what scale of noise is appropriate. In practice given our robot's scale and typical speeds we find $\tilde{\boldsymbol{o}}_{\text{robot}} \sim \mathcal{N}(\boldsymbol{o}_{\text{robot}}, I\boldsymbol{\sigma})$ with σ_i of 1mm, 1 degree, and 1cm works well respectively for translational, rotational, and gripper components.

5.4.4 Multi-View Time-Synchronized Correspondence Training

Unlike in previous work which trained robotic-supervised correspondence models only for static environments [31], we now would like to train correspondence models with dynamic environments. Other prior work [112] has used dynamic non-rigid reconstruction [94] to address dynamic scenes. The approach we demonstrate here instead is to correspond pixels between two camera views with images that are approximately synchronized in time, similar to the full-image-embedding training in [119], but here for pixel-to-pixel correspondence.

For training, like the static-scene case, finding pixel correspondences between images requires only depth images, camera poses, and camera intrinsics. Autonomous object masking can, similar to [31], be performed using 3D-based background subtraction, using only the live depth sensors' point clouds. Since both (a) the time-synchronized technique can only correspond between time-synchronized images rather than many different static-scene views ([31] used approximately 400]), and (b) the time-synchronized technique does not have access to highly accurate many-view-fused 3D geometry as used in [31], it was unclear that our time-synchronized training would provide as compelling results as shown in Sec. 5.5.4. To encourage generalization despite having using only two static views, we add rotation, scale, and shear image augmentations, and to help alleviate incorrect correspondences due to noisy depth images, we add photometric-error-based rejection of correspondences.

5.4.5 Policy Models

We use two standard classes of policy models, Multi-Layer Perceptrons (MLP) and Long Short-Term Memory (LSTM) recurrent networks, which are familiar model classes to many different types of machine learning problems and in particular have been demonstrated to be viable for real-world visuomotor control [64, 162, 103]. In our evaluations the MLPs are only provided current observations, $\pi(\boldsymbol{o}_t)$, whereas through recurrence the LSTMs use the full observation sequence. The Appendix provides more model details.

5.5 Results

Our experimentation sought to answer these primary questions: (1) Is it possible to use self-supervised descriptors as successful input to learned visuomotor policies? (2) How does visual correspondence learning compare to the benchmarked methods in terms of enabling effective policy learning, as measured by generalization performance and sample complexity? We also evaluate (3) the effect of noise augmentation, and (4) whether our dynamic-scene visual training technique is capable of effective

correspondence learning. Simulation setup and results are detailed in Sec. 5.5.1, 5.5.2, hardware setup and results are in Sec. 5.5.3, 5.5.4.

5.5.1 Simulation Experimental Setup

We use simulated imitation learning tasks (Fig. 5-3) to compare the generalization performance of behavior-cloned policies where the only difference is how the "visual representation" $\boldsymbol{z}$ is acquired. The first two tasks involve reaching to an object whose configuration varies between trials either in translation only, or rotation as well. The additional two tasks are both pushing tasks, which require feedback due to simulated external disturbances. Expert demonstrations use simple hand-designed policies using ground truth object state information. The compared methods are:

1. *Ground truth 3D points (GT-3D):* $\boldsymbol{z}$ is ground truth world-frame 3D locations of points on the object.

2. *Ground truth 2D image coordinates (GT-2D):* $\boldsymbol{z}$ is similar to the previous baseline, but the points are projected into the camera using the ground truth camera parameters.

3. *Autoencoder (AE):* $\boldsymbol{z}$ is the encoding of a pre-trained autoencoder, similar to the visual training in [27, 35].

4. *End-to-End (E2E):* $\boldsymbol{z}$ is the intermediate representation from end-to-end training. This closely resembles the visual training and models in [64, 162], but we do not also add pose-based losses, in order to investigate end-to-end learning without these auxiliary losses.

5. *Ours, Dense descriptors (DD):* $\boldsymbol{z}$ is the expected image-space locations (DD-2D) or 3D-space locations (DD-3D) of the descriptor set $\{\boldsymbol{d}\}_i$, where the visual model was trained on dense correspondence.

Note that the two vision-based baselines AE and E2E share an identical model architecture for producing $\boldsymbol{z}$, and differ only in the method used to train the parameters.

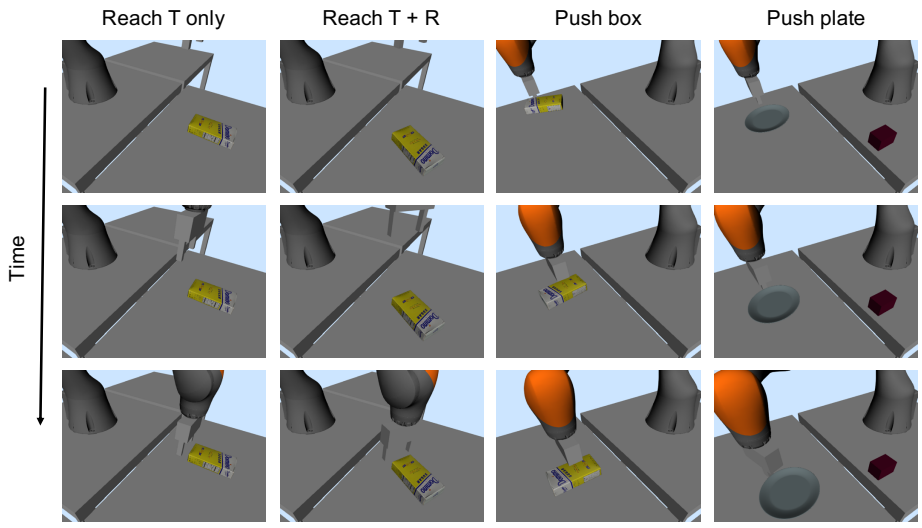


Figure 5-3: RGB images used for visuomotor control in each of the simulation tasks. T=translation, R=rotation, see Appendix for task descriptions.

The model is close to [64, 27, 162] with the key architectural traits of having a few convolutional layers followed by a channel-wise spatial expectation operation, which has been widely used [28, 25, 13, 147, 155, 124, 35]. Most methods we compare (AE, E2E, DD-2D) use only one RGB camera stream as input to learned policies; DD-3D additionally uses the depth image. DD methods use descriptor set optimization (Sec. 5.3.2) and use both views for the correspondence training, before policy training.

See the Appendix for additional model and task details.

5.5.2 Simulation Results

Table 5.1 contains the results of the simulation experiments. Interestingly we find that our method's visual representation is capable of enabling policy learning that is remarkably close in performance to what can be achieved if the policy has access to ground truth world state information. In contrast the performances of the end-to-end (E2E) and autoencoder (AE) methods vary much more across the different tasks. Since our method benefits from object mask information during visual training, we also experimented with letting the autoencoder use this information by applying the reconstruction loss on only the masked image. Additionally we tried training the autoencoder end-to-end during behavior cloning. Both of these yield mixed results,

Method / Task	Reach T only	Reach T + R	Push box	Push plate
Ground truth 3D points	100.0	100.0	100.0	90.5
Ground truth 2D image coord.	94.1	95.6	100.0	92.0
RGB policy input				
Autoencoder, frozen	8.1	61.1	31.0	53.0
Autoencoder w/ mask, frozen	16.3	10.0	73.0	67.0
Autoencoder, then End-to-End	40.7	38.9	–	16.0
End-to-End	43.0	32.2	**100.0**	5.5
End-to-End (34-layer ResNet)	–	3.3	–	–
DD 2D image coord. (ours)	**94.1**	**97.8**	**100.0**	**87.0**
RGBD policy input				
DD 3D coord. (ours)	**100.0**	**100.0**	–	**98.0**

Table 5.1: Summary of simulation results (success rate, as %). DD = Dense Descriptor. See Appendix for task success criteria and additional details.

depending on the task.

Since the vision network in our method is a 34-layer ResNet, we wanted to see if the end-to-end method would benefit from using the same, deeper vision backbone. The deeper network did not improve closed-loop performance (Table 5.1) although it did reduce behavior-cloning validation error. This suggests the advantage of our method comes from the correspondence training rather than the model capacity.

The binary success metrics of Table 5.1, however, do not fully convey the methods' performances. We also experiment with varying the number of demonstrations, and characterize the performance distributions. By plotting the performance for the *"Reach, T + R"* task over a projection of the sampled object configurations (Figure 5-4), we learn that the few failures of our method occur when the box position lies outside the convex hull of the training data. Interestingly the GT-2D baseline also struggles with similar failure modes, while the GT-3D method succeeds in more cases outside the convex hull. This suggests that policies that consume 3D information are better able to extrapolate outside the training distribution; our DD-3D method also provides better generalization than DD-2D. The baseline vision-based methods do not generalize as well; for example, the E2E performance distribution is shown in Figure 5-4. On this task we find that with just 30 demonstrations our method outperforms

Noise / Method	*30 demonstrations* GT-2D	DD-2D	*200 demonstrations* GT-2D	DD-2D
No noise	5.6	1.1	5.6	3.4
With noise	**73.3**	**73.3**	**95.6**	**97.8**

Table 5.2: Comparison of using our feedback-training noise augmentation technique or not (success rate, as %) on the *"Reach, T + R"* task. *No noise* uses $\sigma_i = 0.0$, whereas *With noise* uses $\sigma_{\text{translation}} = 1\text{mm}$, $\sigma_{\text{rotation}} = 1$ degree. See Section 5.5.1 for descriptions of GT-2D (ground truth) and DD-2D (our method).

both AE and E2E with 200 demonstrations.

The pushing tasks are of particular interest since they demand closed-loop visual feedback. Disturbances are applied to the object both while collecting demonstrations and deploying the learned policies. Since the *"Push box"* task used a dynamic state feedback controller to provide demonstrations, we find that we need the sequence model (LSTM) for the policy network to achieve the task, even when the policy has access to ground truth object state. On the other hand, the *"Push plate"* task employed a static feedback controller to provide demonstrations, and so MLP models that consume only the current observation, $\pi_{\theta_p}(\boldsymbol{o}_t)$, are sufficient.

Interestingly a variety of methods performed well on the *"Push box"* task while large differences were evident in the *"Push plate"* task. We speculate that this is because higher precision is required to accurately push the plate as compared to the box. Since the rectangular robot finger experiences a patch contact with the box, while only a point contact with the plate, there is more open loop stability in pushing the box. On the harder *"Push plate"* task we found that our DD-2D method performed almost as well as the GT-2D baseline and significantly outperformed both the AE and E2E approaches, and that DD-3D improved performance even further.

Additionally we find (Table 5.2) our noise augmentation technique (Sec. 5.4.3) has a marked effect on task success for behavior-cloned policies. This applies to ground truth methods and our method, with as few as 30 demonstrations or as many as 200.

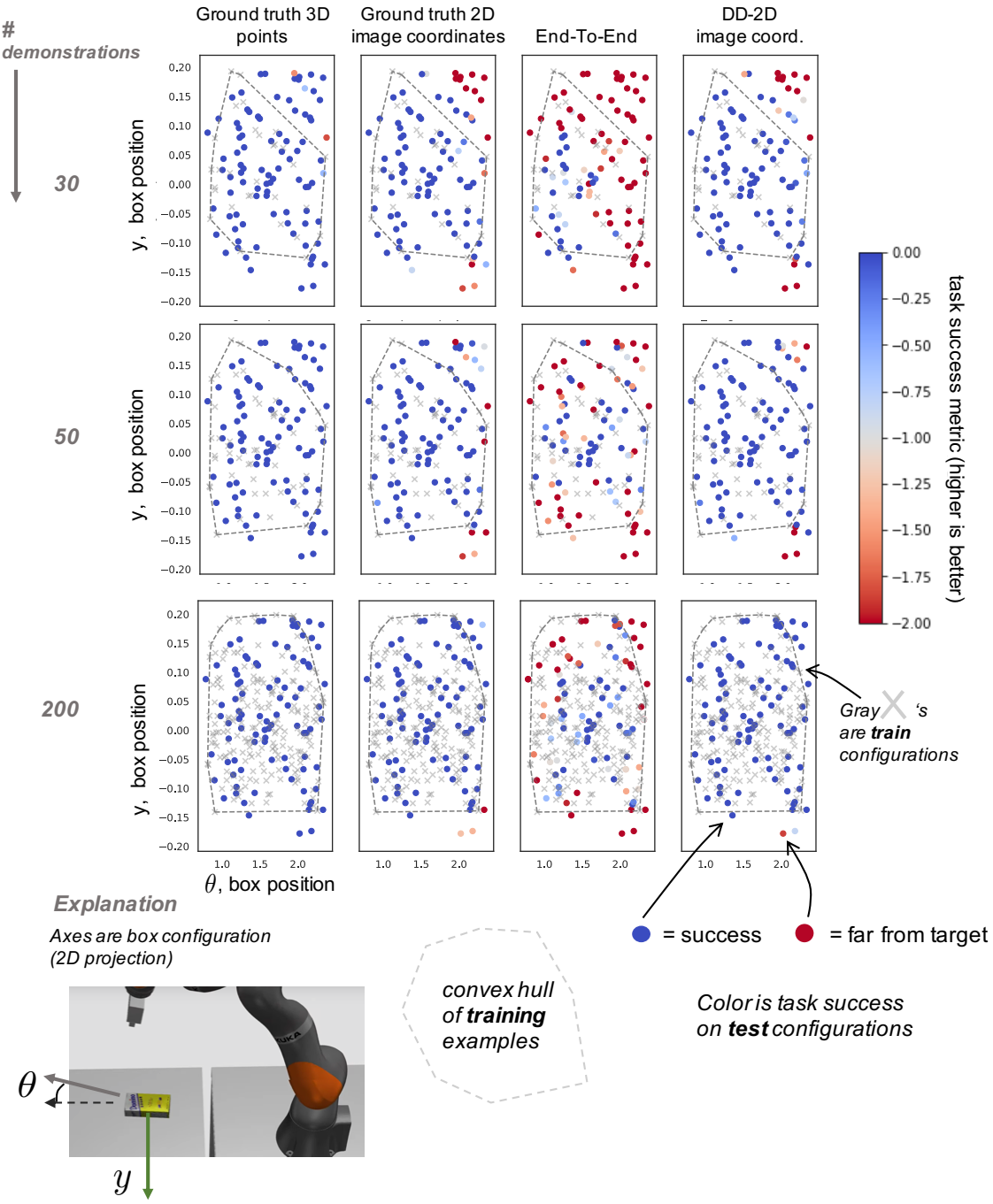


Figure 5-4: Task success distribution plotted over the 2D projection of the varied box configurations for the *"Reach, $T + R$"* task. The color of each point represents the result of deploying the learned policy with the object at that θ, y. Specifically the color encodes the distance to target threshold: $\min(0, -(\Delta\text{translation} + \Delta\text{rotation}) + \epsilon)$, where ϵ is the success threshold. The x coordinate is not shown in order to plot in 2D. Dark blue corresponds to perfect performance on the task with the object in that configuration, red is poor performance. Note that the color scale cuts off at -2 in order to highlight differences in the range [-2,0]. Each gray "x" in each subplot represents the configurations of the box in the training set, for either (from top to bottom) 30, 50, or 200 demonstrations. The dashed gray line shows the convex hull of the respective training sets.

Task	Success criterion	Trained with manual disturbances	*Without disturbances* # attempts	# success	%	*With disturbances* # attempts	# success	%	*Demonstration data* # total	time (min.)
Push sugar box	box is < 3 cm from finish line	yes	6	6	100.0	70	68	97.1	51	13.9
Flip shoe, single instance	shoe is upright	no	43	42	97.7	40	35	87.5	50	6.5
Flip shoe, class-general										
previously seen shoes (14)	shoe is upright	no	43	38	88.4	–	–	–	146	17.5
novel shoes (12)	shoe is upright	no	22	17	77.3	–	–	–	146	17.5
Pick-then-hang hat on rack	hat is on the rack	yes	50	42	84.0	41	28	68.3	52	11.5
Push-then-grab plate	plate is off the table	yes	22	21	95.5	27	22	81.5	94	27.4
Total			186			178				

Table 5.3: Summary of task attempts and success rates for hardware validation experiments. Autonomous re-tries are counted as successes.

5.5.3 Hardware Experimental Setup

We used a Kuka IIWA LBR robot with a Schunk WSG 50 parallel jaw gripper to perform imitation learning for the five tasks detailed in Figure 5-1. RGBD sensing was provided by RealSense D415 cameras rigidly mounted offboard the robot and calibrated to the robot's coordinate frame. Note that for effective correspondence learning between views, it is ideal to have views with *some* overlap such that correspondences exist, but still maintain different-enough views. All shown hardware results use only RGB input for the trained policies (DD-2D, Sec. 5.5.1) and use descriptor set optimization (Sec. 5.3.2). Human demonstrations were provided by teleoperating the robot with a mouse and keyboard.

5.5.4 Hardware Results

We validate both our visual learning method and its use in imitation learning in the real world. As in simulation, we *only use demonstration data* for both visual training and policy learning; no additional data collection is needed. While the simulation results provide a controlled environment for comparisons, there are a number of additional challenges in our real world experiments: (i) visual complexity (textures, lighting, backgrounds, clutter), (ii) use of human demonstrations rather than expert

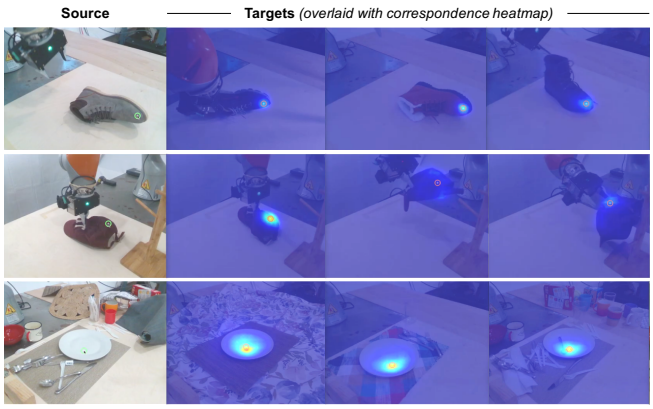


Figure 5-5: Learned correspondences from demonstration data, depicted as correspondence heatmaps between a source pixel (left, with the green reticle) and target scenes (right, with red reticle as best predicted correspondence).

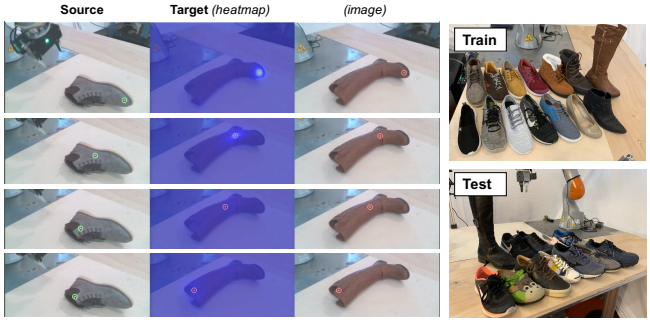


Figure 5-6: Learned correspondences (left) between a standard-sized shoe and an extra-tall boot. A small amount of movement near the top of the ankle on the shoe (far left) corresponds to a "stretched-out" movement on the boot (right). Images cropped for visualization. Also shown are shoe train and test instances (far right).

simulation controllers, (iii) real physical contact, and (iv) imperfect correspondence learning due to noisy depth sensors and calibration. Our hardware experiments test all of these aspects. All real hardware experiments use LSTM policy networks, since we suspect our human demonstrators use dynamic internal state.

Learned Correspondences from Dynamic Scenes

Fig. 5-5 displays visualizations of learned correspondences from demonstration data. The results show that the learned visual models, despite imperfect depth sensor noise, calibration, and only time-synchronized image pairs, are capable of identifying correspondences across a class of objects, for an object in different deformable configu-

rations, and for objects in a diversity of backgrounds. Figure 5-6 displays class-general correspondences for a particularly challenging instance with large shape variation.

Real-World Visuomotor Policies

Figure 5-1 displays examples of autonomous hardware results, and Table 5.3 provides a quantitative overview. To highlight a few results, several of the tasks achieve over 95% reliability, including the *"Push sugar box"* task with and without disturbances, and the *"Flip shoe, single instance"* and *"Push-then-grab plate"* tasks without disturbances. Each of the different tasks present significant challenges, best appreciated in our video. Several of the tasks include non-prehensile manipulation, including pushing the box and plate, and flipping the shoes. In the *"Pick-then-hang hat on rack"* task, the robot autonomously reacts to the deformable configuration of the hat after disturbances. The *"Push-then-grab plate"* task as well is highly challenging given the visual clutter, the symmetry and lack of visual texture for the object, and requires using "extrinsic dexterity" [91] via the wood block to enable sliding the gripper into position to grasp the plate.

5.6 Conclusion

Our experiments have shown self-supervised correspondence training to enable efficient policy learning in the real world, and our simulated imitation learning comparisons empirically suggest that our method outperforms two vision-based baselines in terms of generalization and sample complexity. While different hyperparameters, model architectures, and other changes to the baselines may increase their performance, our method is already near the upper bound of what can be expected in the used experimental setting: it achieves results comparable to baselines using ground truth information. One reason our approach may outperform the vision-based baselines is that it additionally uses a fundamentally different source of supervision, provided by visual correspondence training. Since our approach is self-supervised, it does not entail additional human supervision.

Dense descriptor learning has shown to be an exciting route for improving visuomotor policy learning. While this has enabled the variety of tasks shown, there are many that are out of scope. One current limitation is that our visual representation does not explicitly address simultaneously viewing multiple object instances of the same class. Future work could, similar to the visual pipeline in [74], combine both instance-level segmentation with intra-instance visual representations. Additionally, returning to the cooking eggs example in the Introduction, it is interesting to consider using spatial correspondence as part, but not the entirety, of the visual representation of the world.

Acknowledgements

This work was supported by National Science Foundation Award No. IIS-1427050, Lockheed Martin Corporation Award No. RPP2016-002, and an Amazon Research Award grant. The views expressed are not endorsed by our funding sponsors.

Chapter 6

Keypoints into the Future: Self-Supervised Correspondence with Model-Based Reinforcement Learning

6.1 Introduction

It has been argued that one of the hallmarks of human-level learning is the ability to construct and leverage causal models of the world [62]. In the area of manipulation, this manifests itself in the ability to approximately predict how an object will move if we grasp or push it. Traditional model-based robotics has successfully leveraged such predictive models, oftentimes derived from first principles, to solve challenging planning and control problems [87, 83]. In the area of practical vision-based robotic manipulation, however, it has been particularly hard to leverage such predictive models, due to varied and novel objects and the high-dimensional observation spaces involved (e.g., RGB or RGBD images). Alternative approaches, such as imitation learning or model-free reinforcement learning, sidestep the task of building a predictive model and directly learn a policy. Although this can be appealing, model-based techniques offer several benefits. They can be sample efficient compared to model-free methods and, in contrast to behavior cloning techniques, can leverage off-policy non-expert data. Once

a model has been acquired, it can be used together with a planner to achieve a wide variety of tasks and goals. One of the main challenges for model learning applied to robotic manipulation is determining the state representation on which the dynamics model should be learned. Prior work has used approaches ranging from full image space dynamics [23, 19, 152, 128] to a variety of autoencoder formulations [1, 40].

In this paper we propose to use object keypoints, which are tracked over time, as the latent state in which to learn the dynamics. These keypoints anchor our model-based predictions, and provide various advantages over alternatives such as abstract latent states: (i) the output is interpretable, which enables the ability to analyze the performance of the visual model separately from the predictive model. (ii) The representation is 3D and hence can naturally handle changing general off-axis 3D camera positions. (iii) As demonstrated in [31, 32] the visual models we use, *Dense Object Nets*, have demonstrated reliable performance in a variety of real-world settings and are able to generalize at the category level. We found that autoencoder approaches particularly struggled with category-level generalization. And as discussed in prior works [31, 74], keypoints and dense correspondences provide advantages over using 6D object poses: they can apply to deformable objects and represent category-level generalization.

We show that this formulation enables reliable, sample-efficient learning capable of precise visual-feedback-based manipulation in the real world – and is trained with nothing other than a small amount of interaction data (10 minutes) and a single demonstration for goal specification. In our approach to acquiring keypoints, we extract them as descriptors which are tracked from a dense descriptor model – while multiple approaches could be used to acquire keypoints, this route can be entirely self-supervised. As opposed to [32] which uses keypoints from a dense-correspondence model in an imitation learning framework, the use of keypoints as input to a model-based RL system presents several unique challenges. In particular the keypoints need to be both informative for the task at hand and be able to be tracked accurately. In this paper we explore these challenges and propose solutions.

Contributions: Our primary contribution is (i) a novel formulation of predictive

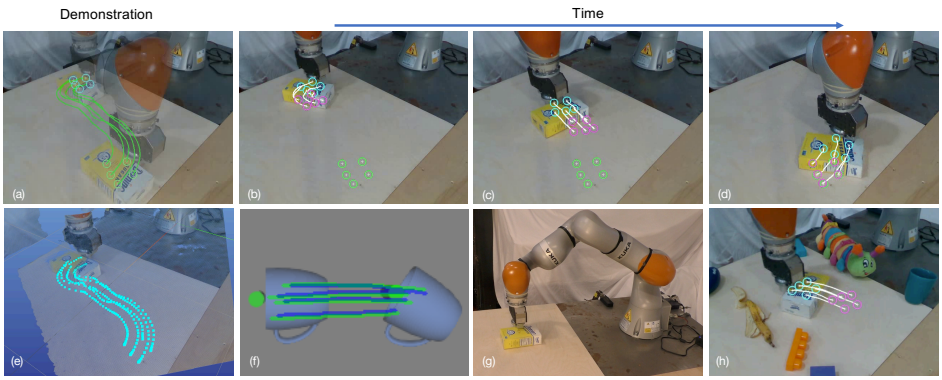


Figure 6-1: (a) Shows the initial pose (blue keypoints) and goal pose (green keypoints) along with the demonstration trajectory. (b) - (d) show the MPC at different points along the episode. Current keypoints are in blue, white lines and purple keypoints show the optimized trajectory from the MPC algorithm, using the learned dynamics model. Goal keypoints are still shown in green. (e) shows the demonstration trajectory in a 3D visualizer. (f) Illustrates a dynamics model on a category level task. The actual keypoint trajectory is shown in green; the predicted trajectory using the learned model shown in blue. (g) Overview of our hardware setup. (h) Example of visual clutter.

model-learning using learned dense visual descriptors as the state representation. We use this approach to perform closed-loop visual feedback control via model-predictive-control (MPC). (ii) Using simulated manipulation experiments we demonstrate that this approach offers performance benefits over a variety of baselines, and (iii) we validate our approach in real-world robot experiments.

6.2 Related Work

We focus our related work on methods that target robotic manipulation with learned predictive models. The Introduction discussed alternative approaches to synthesizing closed-loop feedback controllers without predictive models, via imitation learning or model-free reinforcement learning.

Model-based RL in Robotic Manipulation. These methods can be classified by whether they use first-principles-based or data-driven models, and whether they consume raw visual inputs (such as RGBD images) or they consume ground truth state

information (from a simulator or an external perception system). First-principles-based models, e.g., [47, 163], rely on known object models and thus don't generalize to novel or unknown objects, and also rely on external vision systems. Given this, the tasks we consider are out of scope for these approaches. In the area of methods that use external vision systems but learn data-driven models, [46] learns a dynamics model for a closed-loop-controlled planar pushing task, however, the approach is tailored to the specifics of the pusher-slider task and doesn't readily generalize to other tasks, or to novel objects within the same task. [92] learns deep dynamics models for a variety of different simulated dexterous manipulation tasks, using ground truth object states, and one task on real hardware, using an external camera-based 3D object position tracker. Although [46, 92] achieve impressive results, their reliance on ground truth object state and/or specialized visual trackers limits their general applicability in more diversified real-world manipulation tasks.

There also exists a large literature on model-based RL for robotic manipulation that operates in the more challenging problem class of directly consuming image observations. Methods can be broadly categorized into whether they predict the image-space dynamics of the entire image [23, 20, 19, 152, 128], or predict the dynamics of a low-dimensional latent state [148, 141, 1, 2, 40]. Although image-space dynamics approaches are general, they require large training datasets. For latent-space dynamics approaches, to avoid a trivial solution where all observations get mapped to a constant vector, a regularization strategy is needed for the latent state $\boldsymbol{z}$. [141, 40] use an autoencoder architecture and regularize the latent state $\boldsymbol{z}$ using a reconstruction loss. [1, 2] regularize the latent space by simultaneously training both forward and inverse dynamics models while [148] uses a contrastive loss on the latent state. In contrast to these approaches we use visual-correspondence pretraining to produce a latent state which is physically grounded and interpretable as the 3D locations of keypoints on the object(s).

Visual Representation Learning For more related-work in self-supervised visual learning for robotics, we refer the reader to [31]. Approaches that use autoencoders [141, 40] or full image-space dynamics [23, 20, 19, 152] rely on image reconstruction

as their source of visual supervision. [59] is perhaps most related to ours, in that they first learn a visual model which is then used for a downstream task, and they show that freezing the visual model and using a keypoint-type representation as an input to model-free RL algorithms improves performance on Atari ALE [7]. Our approach is distinct in that (i) we use a predictive model-based method rather than a model-free method, (ii) we use a correspondence-based training loss, while [59] uses an image-reconstruction-based loss with a specialized pixel-space transport mechanism, and (iii) we demonstrate results with real-world hardware. As a baseline, we try using their vision model as an input to the same model-based RL algorithm used by our own method.

6.3 Formulation: Self-Supervised Correspondence in Model-Based RL

This section describes our approach to model-based RL using visual observations. The goal is to learn a dynamics model that can then be used to perform online planning for closed-loop control.

6.3.1 Model-Based Reinforcement Learning

Our setting consists of an environment with states $\boldsymbol{x} \in \mathcal{X}$, observations $\boldsymbol{o} \in \mathcal{O}$, actions $\boldsymbol{a} \in \mathcal{A}$ and transition dynamics $\boldsymbol{x}' = f_{\text{state}}(\boldsymbol{x}, \boldsymbol{a})$. The task is specified by a reward function $r(x, a)$ and the goal is to choose actions to maximize the expected reward over a trajectory. We approach this problem by first learning an approximate dynamics model $\hat{f}_{\theta_{\text{dyn}}}$, which is trained to minize the dynamics prediction error on the observed data $\mathcal{D}$. The learned model is then used to perform online planning to obtain a feedback controller.

Typical example environments are depicted in Figures 6-1, 6-3. The state $\boldsymbol{x}$ contains information about the underlying pose and physical properties of the object, but we only have access to the observation $\boldsymbol{o}$. The observation typically consists of

both the robot's proprioceptive information $\mathcal{O}_{\text{robot}}$ (such as joint angles, end-effector poses, etc.) and high-dimensional images $\mathcal{O}_{\text{image}} \in \mathbb{R}^{W \times H \times C}$ for a C-channel image of height H and width W. Hence the full observation space is $\mathcal{O} = \mathcal{O}_{\text{robot}} \times \mathcal{O}_{\text{image}}$. For the purposes of our approach we will assume that $\boldsymbol{x}$ is fully observable from $\boldsymbol{o}$ (or a short history of $\boldsymbol{o}$ in order to infer velocity information). Note that this still allows for the object to undergo significant partial occlusion as long as it is not completely occluded, see Figure 6-3 (c) for an example. While our work focuses on addressing partial occlusion, future work may address full occlusion via models with longer time horizons or higher-level planning.

Rather than learn the dynamics directly in the observation space, as in [23, 20], we instead learn a mapping $g : \mathcal{O} \to \mathcal{Z}$ from the high-dimensional observation space $\mathcal{O}$ to a low-dimensional latent space $\mathcal{Z}$ together with a dynamics model $\hat{\boldsymbol{z}}_{t+1} = \hat{f}_{\theta_{dyn}}(\boldsymbol{z}_{t-l:t}, \boldsymbol{a}_t)$ in this latent space, where $\boldsymbol{z}_{t-l:t} = (\boldsymbol{z}_t, \boldsymbol{z}_{t-1}, \ldots, \boldsymbol{z}_{t-l})$ encodes a short history (we use $l = 1$ in all experiments). This latent state should capture sufficient information about the true state $\boldsymbol{x}$ such that driving $\boldsymbol{z} \to \boldsymbol{z}*$ sufficiently well achieves the goal of driving $\boldsymbol{x}$ to $\boldsymbol{x}^*$ (where $\boldsymbol{z}^*$ is the latent state corresponding to $\boldsymbol{x}^*$). Given the current latent state and a sequence of actions $\{\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots\}$ we can predict future latent states $\boldsymbol{z}$ by repeatedly applying our learned dynamics model. The forward model is then trained to minimize the dynamics prediction error (also know as *simulation error*) over a horizon H

$$\mathcal{L}_{\text{dynamics}} = \sum_{h=1}^{H} ||\hat{\boldsymbol{z}}_{t+h} - \boldsymbol{z}_{t+h}||_2^2, \quad \hat{\boldsymbol{z}}_{t+h+1} = \hat{f}_{\theta_{\text{dyn}}}(\hat{\boldsymbol{z}}_{t-l:t}, \boldsymbol{a}_{t+h}), \quad \hat{\boldsymbol{z}}_t = \boldsymbol{z}_t \tag{6.1}$$

6.3.2 Learning a Visual Representation

The objective of the visual model $g : \mathcal{O} \to \mathcal{Z}$ is to produce a low-dimensional feature vector $\boldsymbol{z}$ which serves as a suitable latent state in which to learn the dynamics. For the types of tasks and environments that we are interested in, spatial information about object locations is a critical piece of information. Pose estimation has played a critical role in classical manipulation pipelines, and was also used in dynamics learning approaches such as [46, 92]. In general producing pose information from

high-dimensional observations (such as RGBD images) requires a dedicated perception system. Although pose can be a powerful state representation when dealing with a single known object, as noted in [31, 74, 32] it has several drawbacks that limit its usefulness in more general manipulation scenarios. In particular it doesn't readily (i) extend to the case of deformable objects, (ii) generalize to novel objects or (iii) extend to category-level tasks.

Our strategy is to leverage visual-correspondence pre-training to track points on the object of interest. The locations of these tracked points can then serve as the latent state on which we learn the dynamics. Similar to the approach taken in [31, 32], we use visual-correspondence learning, which is trained in a completely self-supervised fashion, to train a visual model which that can be used to find correspondences across RGB images. We then propose several approaches to produce a low-dimensional latent $\boldsymbol{z}$ using the pre-trained dense-correspondence model.

First we give a bit of background on dense correspondence models (see [30, 31] for more details). Given an image observation $\boldsymbol{o}_{\text{image}} \in \mathbb{R}^{W \times H \times C}$ (where C denotes the number of channels), the dense-correspondence model $g_{\theta_{\text{dc}}}$ outputs a full-resolution descriptor image $\mathcal{I}_D \in \mathbb{R}^{W \times H \times D}$. Since we want to learn a dynamics model on a low-dimensional state, we need a way to construct $\boldsymbol{z}$ from the descriptor image $\mathcal{I}_D$. The idea, similar to [32], is for $\boldsymbol{z}$ to be a set of points on the object(s) that are localized in either image-space or 3D space. These points are represented as a set $\{d_i\}_{i=1}^K$ of K descriptors, where each $d_i \in \mathbb{R}^D$ is a vector in the underlying descriptor space. A parameterless *correspondence function* $g_c(\mathcal{I}_D, d_i)$ [1] extracts the location of the keypoint $y_i \in \mathbb{R}^B$ from the current observation. Combining our learned correspondences together with the reference descriptors, we have a function that maps image observations $\boldsymbol{o}_{\text{image}}$ to keypoint locations $\boldsymbol{y} = \{y_i\}_{i=1}^K$. We propose four methods for constructing the latent state $\boldsymbol{z} = g_{\theta_z}(\boldsymbol{y})$ from $\boldsymbol{y}$, where θ_z denotes the (potentially empty) set of trainable parameters in this mapping.

Descriptor Set (DS): In our simplest variant the latent state $\boldsymbol{z}$ is simply made up of keypoint locations y_i for a set of descriptor keypoints randomly sampled from the

[1] see Appendix D.1.2 for more details on the visual-correspondence model

object. Specifically, we sample K (we use $K = 50$ in all experiments) descriptors $\{d_i\}_{i=1}^K$ corresponding to pixels from a masked reference descriptor image in our training set. Thus

$$\boldsymbol{z} = (\boldsymbol{z}_{\text{object}}, \boldsymbol{o}_{\text{robot}}) = (\boldsymbol{y}, \boldsymbol{o}_{\text{robot}}) = (\{y_i\}_{i=1}^K, \boldsymbol{o}_{\text{robot}}) \tag{6.2}$$

Spatial Descriptor Set (SDS): Rather than randomly sampling descriptors, as in **(DS)**, this method attempts to choose descriptors $\boldsymbol{d} = \{d_i\}_{i=1}^K$ having specific properties. In particular we would like the descriptors $\{d_i\}_{i=1}^K$ to be (i) *reliable*, and (ii) *spatially separated*. By *reliable* we mean that they can be localized with high accuracy and don't become occluded during the typical operating conditions, see Figure 6-2 for an example. *Spatially separated* means that the chosen descriptors aren't all clustered around the same physical location on the object(s) of interest, but rather are sufficiently spread out (either in 3D space or pixel space) to provide meaningful information about both object position and orientation. Our dense descriptor model can provide a confidence score associated with descriptors and their associated correspondences.[1] Figure 6-2 shows a clear example of high confidence for a valid match and low-confidence when no valid correspondence exists due to an occlusion. We use this feature of our visual model to compute a confidence score c_i for each each descriptor d_i according to what fraction of images in the training set $\mathcal{D}$ contain a high probability correspondence for d_i. The intuition is that descriptors d_i corresponding to points on the object that are easy to localize and remain unoccluded will have a high confidence score. As in the **DS** method we initially select a large number of descriptors ($K = 100$) corresponding to points on the object. We then select the K^* descriptors with the highest average confidence on the training set and which additionally satisfy a threshold on minimum separation distance (typically 25 pixels in a 640×480 image). In the experiments we use $K^* = 4$ or $K^* = 5$.

Weighted Descriptor Set (WDS): One problem that can arise when learning the dynamics of a latent state $\boldsymbol{z}$ is that some component of $\boldsymbol{z}$ may be noisy or unreliable,

[1] see Appendix for more details

Figure 6-2: Visualization of the learned visual-correspondence model on a reference image (left) and target image (right). Colored numbers in the target image represent the probability that the detected correspondence is valid. Green reticle shows a valid correspondence with a high confidence score, red reticle shows a case where no correspondence exists in the target image due to occlusion, hence the low confidence probability. Confidence heatmap shown in bottom right.

which can make it difficult or impossible to learn a dynamics model $\boldsymbol{z}' = f(\boldsymbol{z}, \boldsymbol{a})$. This problem doesn't arise in the imitation learning setting of [32] which performs supervised learning from $\boldsymbol{z}_t \rightarrow \boldsymbol{a}_t^{\text{expert}}$, and thus can learn to ignore components of the latent state $\boldsymbol{z}_t$ that aren't useful for predicting the action $\boldsymbol{a}_t^{\text{expert}}$. The fundamental difference of our dynamics learning formulation compared to the imitation learning setup of [32] is that the latent $\boldsymbol{z}_t$ appears directly in the cost function (6.1). There are a variety of reasons that one or more of the tracked descriptors could be unreliable (e.g. occlusions, regions of the object where the correspondence model has less accurate) and thus we would like our dynamics learning framework to be robust to this. We achieve this by defining a learnable mapping $\phi : \boldsymbol{y} \rightarrow \boldsymbol{z}$ which maps the keypoint locations $\boldsymbol{y}$ to the latent state $\boldsymbol{z}$, where the keypoints $\boldsymbol{y}$ are as in our **DS** method. A regularization strategy is needed to ensure that ϕ doesn't collapse to the trivial solution $\phi \equiv 0$. We introduce learnable weights $\alpha \in \mathbb{R}^{K \times K}$ and define $w_{k,i} = \frac{\exp(\alpha_{k,i})}{\sum_{j=1}^{K} \exp(\alpha_{k,j})}$. Let $W \in \mathbb{R}^{K \times K}$ be the matrix with entries $w_{k,i}$, where the parameterization guarantees

that $w_{k,i} \geq 0$ and $\sum_{i=1}^{K} w_{k,i} = 1$. ϕ is defined as

$$\tilde{y}_k = \sum_{i=1}^{K} w_{k,i} y_i, \quad \phi(\boldsymbol{y}; \alpha) = \tilde{\boldsymbol{y}} = \{\tilde{y}_k\}_{k=1}^{K} \tag{6.3}$$

Note that each y_i is a keypoint location in $\mathbb{R}^B$. Thus $\tilde{\boldsymbol{y}}$ is simply a convex combination of the keypoints in $\boldsymbol{y}$. The latent state $\boldsymbol{z}$ is then defined as

$$\boldsymbol{z} = (\boldsymbol{z}_{\text{object}}, \boldsymbol{o}_{\text{robot}}) = (\tilde{\boldsymbol{y}}, \boldsymbol{o}_{\text{robot}}) = (\phi(\boldsymbol{y}, \alpha), \boldsymbol{o}_{\text{robot}}) \tag{6.4}$$

The learnable weights α are trained jointly with the parameters θ_{dyn} of the dynamics model, and are fixed at test time. The fact that $\boldsymbol{z}_{\text{object}}$ is gotten from $\boldsymbol{y}$ by taking a weighted linear combination preserves the interpretation of $\boldsymbol{z}$ as tracking keypoints on the object, while allowing some flexibility to ignore unreliable keypoints.

Weighted Spatial Descriptor Set (WSDS): This method is simply the combination of **(SDS)** and **(WDS)**.

6.3.3 Learning the Dynamics

We adopt a standard dynamics learning framework where we aim to predict the evolution of the latent state $\boldsymbol{z}$. We train our dynamics model to minimize the prediction error in Equation (6.1) where $\boldsymbol{z}_t = g_{\theta_z}(\boldsymbol{o}_t)$. Our proposed methods differ in the structure of the mapping $g : \mathcal{O} \to \mathcal{Z}$ and the set of trainable parameters Θ.[1] For all of our methods we keep the weights of the visual-correspondence network, θ_{dc}, fixed.

6.3.4 Online Planning for Closed-Loop Control

Once we have learned a dynamics model $\boldsymbol{z}' = f(\boldsymbol{z}, \boldsymbol{a})$, we use online planning with MPC to select an action. Given a goal latent state $\boldsymbol{z}^*$, we want to find an action sequence $\{\boldsymbol{a}_{t'}\}_{t'=t}^{t+H-1}$ that maximizes the reward $R = \sum_{t'=t}^{t+H-1} r(\boldsymbol{z}_{t'}, \boldsymbol{a}_{t'})$. Our dynamics learning approach is agnostic to the type of optimizer used to solve the MPC problem

[1] see Appendix D.2.2 for details

and the focus of our work is on the visual and dynamics learning, rather than the specifics of the MPC. Many model-based RL approaches [152, 19, 92, 40, 23] use a random-sampling based planner (e.g. cross-entropy method) to solve the underlying MPC problem. We experimented with random shooting, gradient based shooting, cross-entropy and model-predictive path integral (MPPI) planners and found that MPPI worked best in our scenarios.[1]

6.4 Results

We perform experiments aimed at answering the following questions: (1) Is it possible to successfully use self-supervised descriptors as the latent state for a model-based RL system? (2) What is the effect of various design decisions in our algorithm? (3) How does visual-correspondence learning compare to several benchmark methods in terms of enabling effective model-based RL policies? (4) Can we apply the method on real hardware?

Our main contribution is on the formulation of the visual model and dynamics learning problem rather than the specifics of the MPC. However, to accurately compare our approach to various baselines, we need to perform experiments in which the dynamics model is used in closed-loop to solve a manipulation task. Our formulation of dynamics learning is very general, and in principle can handle a wide variety of manipulation scenarios. However, even with an accurate model (whether it comes from first principles or is learned), using this model to perform closed-loop feedback control remains a challenging problem. Thus, for our closed-loop experiments, we limit ourselves to pushing tasks that can adequately be solved by the planners outlined in Section 6.3.4.

Tasks: Extended experimental details are provided in the Appendix but we provide a brief overview of the tasks here. We perform experiments with four simulated tasks (depicted in Figure 6-3) and one hardware task. All tasks involve pushing an object to a desired goal state. The first three simulation tasks, denoted as *top-down, angled,*

[1] See Appendix D.3 for more details.

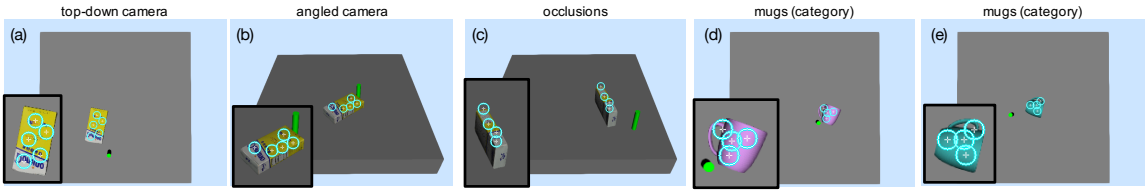


Figure 6-3: This figure shows reference descriptors $\{d_i\}_{i=1}^{K}$ of the **Spatial Descriptor Set** variant for our different simulation environments. In particular in (c) the sides of the object undergo occlusions as the box rotates about the vertical axis. (d)-(e) show two different mugs from the category-level *mugs (category)* task.

occlusions involve pushing a single object. *top-down* has cameras in a top-down orientation while they are angled at 45 degrees in *angled*. Task *occlusions* keeps the angled camera positions but the box is resting on a different face, resulting in significant self-occlusions. Task *mugs* involves pushing many different objects from a category, in this case mugs with different size, shape and textures. The hardware task is essentially identical to the *angled* sim task.

6.4.1 Visual-correspondence Performance

Figures 6-1, 6-2 show the performance of our dense visual-correspondence model. In particular Figure 6-1 shows the localization performance on real data along a trajectory while Figure 6-2 shows an example of the confidence scores used as in the **SDS** method. Figure 6-3 shows the descriptors used in the **SDS** method for each of our simulation tasks. In particular Figure 6-3 (d)-(e) shows the ability of the descriptors to accurately find correspondences across different object instances within a category, despite differences in color and shape.

6.4.2 Ablations on visual-correspondence for dynamics learning

Ablation studies show that the choice of *what to track* can have a substantial impact on model performance, especially in the case of partial occlusions. Quantitative results are detailed in Table 6.1. On tasks *top-down* and *angled camera* all methods perform reasonably well, almost matching the performance of the **GT 3D** baseline

Task	*top-down camera*		*angled camera*		*occlusions*		*mugs (category)*	
Method / task data	pos, cm	angle, °	pos, cm	angle, °	pos, cm	angle, °	pos, cm	angle, °
Avg. trajectory	11.32	32.05	11.32	32.05	10.36	31.81	11.28	56.25
GT 3D	1.14	4.01	1.14	4.01	1.29	5.45	–	–
SDS	1.19	3.88	1.10	3.97	1.46	6.77	1.20	15.03
WDS	1.16	4.43	1.30	5.12	1.49	8.64	1.00	13.29
DS	1.12	4.07	1.45	5.50	3.66	12.27	1.19	11.73
WSDS	1.19	4.00	1.18	4.86	1.28	5.59	1.39	18.18

Table 6.1: *Ablations and comparison with ground-truth* – quantitative results for various ablations of our method on four simulated tasks. Each method was evaluated on the same set of 200 different initial and goal states. The *pos* and *angle* columns denote the translational (in cm) and rotational (in degrees) deviations of the object from the goal position, averaged across all trials for a specific method and task. *Avg. trajectory* denotes the average translation and rotation between the initial and goal poses for each task.

that uses ground truth state information. Intuitively this is because there are minimal occlusions in these settings, and so almost all keypoints can be tracked reliably using dense-visual-correspondence. In contrast the *occlusions* introduces the potential for significant occlusions. Given the camera angle as shown in Figure 6-3 (c) and the fact that the object rotates through the full 360 degrees of yaw during the task, only the top face of the box remains unoccluded while the 4 side faces are alternately occluded and visible. This task exposes significant differences in performance among our various ablations. In particular **SDS**, **WDS** and **WSDS** perform significantly better than **DS**. We believe that this is due to the fact that some of the descriptors $\{d_i\}_{i=1}^K$ that are tracked in **DS** correspond to locations on the object that become occluded during an episode. The dense visual-correspondence model is not able to track keypoints through occlusions, and when trying to localize an occluded point our dense-correspondence model maps it to the closest point in descriptor space, which is not the location of the true correpondence. Hence the keypoint locations that makeup the latent state $\boldsymbol{z}_{\text{object}}$ for **DS** suffer reduced accuracy, leading to a less accurate dynamics model and ultimately lower performance when used for closed-loop MPC.

On the category-level task *mugs* the camera is in a top-down position and thus occlusions are no longer an issue. However because the task involves different objects from a category there is shape variation among the different objects. Thus the methods

Task	*top-down camera*		*angled camera*		*occlusions*		*mugs (category)*	
Method	pos, cm	angle, °	pos, cm	angle, °	pos, cm	angle, °	pos, cm	angle, °
SDS (ours)	1.19	**3.88**	**1.10**	**3.97**	**1.46**	**6.77**	1.20	15.03
WDS (ours)	**1.16**	4.43	1.30	5.12	1.49	8.64	**1.00**	**13.29**
Transporter 3D	2.01	15.95	4.36	25.14	3.72	20.65	2.81	61.22
Transporter 2D	2.08	13.59	3.60	23.60	2.74	18.86	2.33	60.18
Autoencoder	2.18	14.79	2.85	13.79	3.08	14.20	9.05	56.84

Table 6.2: *Comparisons with baselines* – quantitative results of our method compared to various baselines on our four simulated tasks. Each method was evaluated on the same set of 200 different initial and goal states. *pos* and *angle* denote the translational (in cm) and rotational (in degrees) deviations of the object from the goal position, averaged across all trials for a specific method and task.

that use $K = 50$ keypoints, namely **DS** and **WDS**, perform better than the sparser variants **SDS**, **WSDS** that use only $K = 4, 5$ keypoints. We believe that this is due to the fact that having a larger number of keypoints better captures the shape variation across object instances and allows for a more accurate dynamics model.

6.4.3 Comparison of visual-correspondence pretraining with baselines

In our comparisons against baselines, our model outperforms alternatives on all experimental tasks. The largest differences are apparent in tasks *occlusions* and *mugs* which involve partial occlusions and category-level generalization, respectively. The **transporter** baseline uses the keypoint locations from [60] as the latent state z , while the **autoencoder** baseline jointly trains an autoencoder with a forward dynamics model. For a detailed discussion of the baselines see Appendix D.4.3. Quantitative results are detailed in Table 6.2.

On task *top-down camera*, the **transporter**[59] model was able to achieve performance that was only slightly worse than **WDS** and **SDS**, while the **autoencoder** performed significantly worse. The top-down setting is ideally suited for the feature transport approach of the transporter model.

On tasks *angled camera* and *occlusions*, which have angled camera positions as opposed to the top-down task, the performance of our methods remained consistent while **transporter** suffered. This is potentially due to the fact that the feature

transport mechanism of **transporter** is not well-suited to off-axis camera positions in 3D worlds. The performance of the **autoencoder** baseline remainded consistent, but worse than our approach, across tasks without category-level generalization.

Task *mugs* contains a variety of different mug shapes with varied visual appearances and hence tests category-level generalization of both the perception and dynamics models. As discussed in Section 6.4.1, our dense-correspondence model is able to find correspondences across these variations in appearance using only self-supervision, which allows us to learn a dynamics model that is effective for completing the task. This task is significantly harder than the other tasks not only because of the presence of novel objects, but because the goal states involve much larger rotations, as detailed in the first row of Table 6.1. Both the **transporter** and **autoencoder** baselines perform poorly in this task. We hypothesize that this is due to the fact that there is much more variance in the visual appearance of the objects as compared to the other tasks and thus the latent state $\boldsymbol{z}$ produced by these baselines is not amenable to dynamics learning.

6.4.4 Hardware

Experimental Setup: We used a Kuka IIWA LBR robot with a custom cylindrical pusher attached to the end-effector to perform our hardware experiments, see Figure 6-1. RGBD sensing was provided by two RealSense D415 cameras rigidly mounted offboard the robot and calibrated to the robot's coordinate frame. To enable effective correspondence learning between views, it is ideal to have views with *some* overlap such that correspondences exist, but still maintain different-enough views. At test time only a single camera is used to localize the dense-descriptor keypoints. The robot is controlled by commanding end-effector velocity in the xy plane at 5Hz.

Hardware Results: For visual learning we collected a small dataset of the object in different positions to provide a diverse set of views for training the dense-correspondence model. For dynamics learning, we collected a dataset of the robot randomly pushing the object around. This amounted to approximately 10 minutes of interaction time and was used to train the dynamics model. All hardware experiments used the **SDS**

method. To enable our MPC controller to accomplish long-horizon tasks, we supplied the controller with a reference trajectory for the object keypoints that came from a single demonstration, see Figure 6-1 (a),(d). The MPC controller then tracked this reference trajectory using a 2 second MPC horizon, which corresponds to $H = 10$ since we are commanding actions at 5 Hz. We collected 4 different reference trajectories[1] and ran multiple rollouts for each trajectory, varying the initial condition of the object pose during each run to test the region of attraction of our controller. In all cases our controller showed the ability to stabilize the system to the reference trajectory in spite of perturbations to the initial condition. Quantitative results are detailed in Figure 6-4. In particular, we see that the ability of the controller to stabilize the trajectory in the face of disturbances in the initial condition depends on the trajectory. For trajectory (1) the controller is able to stabilize disturbances of up to 60 degrees, while trajectory (4) has a much smaller region of attraction. As can be seen in Appendix D.5, trajectory (1) is a relatively simple trajectory with minimal orientation change between start and goal, while trajectory (4) involves a challenging 180-degree orientation change and requires the robot to operate at the edge of its kinematic workspace, reducing its control authority. Overall, our system exhibits impressive feedback and is able to track a trajectory in the keypoint latent space, enabling one-shot imitation learning. We encourage the reader to watch the videos on our project page to see the system in action.

6.5 Conclusion

We presented a method for using self-supervised visual-correspondence learning as input to a predictive dynamics model. Our approach produces interpretable latent states that outperform competing baselines on a variety of simulated manipulation tasks. Additionally, we demonstrated how the category-level generalization of our visual-correspondence model enables learning of a category-level dynamics model, resulting in large performance gains over baselines. Finally, we demonstrated our

[1] see Appendix D.5 for details on the reference trajectories

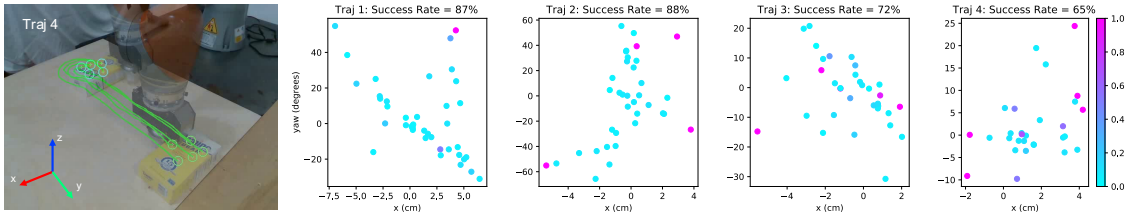


Figure 6-4: Left image shows demonstration for trajectory 4. Scatter plots show results of our approach on the four different reference trajectory tracking tasks. The axes of the plots show the deviation of the object starting pose from the initial pose of the demonstration. Color indicates the distance between final and goal poses, lower cost is better. The various reference trajectories are of different difficulties, as reflected by the different regions of attraction of the MPC controller. More details can be found in Appendix D.5 and videos are on our project page

.

approach on a real hardware system, and showed its ability to stabilize complex long-horizon plans by tracking the latent state trajectory from a single demonstration.

Acknowledgements

This work was supported by Amazon.com Services LLC Award No. CC MISC 00272683 2020 TR and an Amazon Research Award. The views expressed are not endorsed by our funding sponsors.

Chapter 7

Conclusion

This thesis has studied various questions surrounding robotic manipulation, from how to develop and perceive relevant object representations to how to synthesize feedback controllers. In this chapter we summarize what we have learned and explain how the work in the preceding chapters fits together to address the overarching questions of the thesis. Finally we close by proposing some directions for future work.

7.1 Summary of Contributions

Our first step into the world of robotic manipulation was LabelFusion (Chapter 2), which proposed a pipeline for efficiently collecting the type of labeled data needed to train learning based perception systems on tasks relevant to manipulation (e.g. instance segmentation, pose estimation, etc.). This pipeline allowed us to very efficiently collect a large dataset of RGBD images with high-quality object-pose annotations. Although this approach represented a significant advance in efficiency over existing data-labelling approaches it suffers some drawbacks almost by construction. LabelFusion has three primary limitations. (1) We can only handle known objects for which we have complete mesh models, thus we cannot generalize to novel objects. (2) The method doesn't extend to deformable objects. (3) It requires human initialization of 3D object poses. The first two limitations arise directly from the fact that we have chosen to use 6DOF pose as our object representation. This is a major factor that led to us moving away

from 6DOF pose as our object representation in subsequent chapters.

The aforementioned limitations of LabelFusion directly inspired the work of Chapter 3, DenseObjectNets. In particular, at the start of the DenseObjectNets project we set ourselves three requirements for the object representation: (i) The representation should be able to handle both rigid and deformable objects, (ii) it shouldn't rely on having pre-built 3D mesh models, (iii) should be learned in an entirely self-supervised manner. This led to our development of dense-descriptors as an object representation for manipulation, which can be viewed as the primary contribution of this work. This representation satisfies requirements (i) - (iii) outlined above and took advantage of multi-view consistency as the source of supervision. In terms of how this work fits into the broader theme of the thesis, it offers an answer to the first question posed in 1.1, namely "what is an appropriate state-space and/or object representation for performing manipulation."

We believe that DenseObjectNets was particularly successful because it combines the supervision from multi-view consistency with a *dense* pixelwise representation. This dense pixelwise representation implies that there are *thousands* of matches between image pairs which provides a copious amount of supervision for the learning process. We were also pleasantly surprised that the dense-descriptor representation shows generalization across a category of objects, i.e. we can use the dense-descriptors to find correspondences across different shoes. We showed the usefulness of dense-descriptor in simple manipulation tasks that required grasping an object at a specific location, e.g. "grab the shoe by the tongue" and showed that we could accomplish this across a category of objects (e.g. many different shoes). Circling back to the broader questions of the thesis posed in Section 1.1 DenseObjectNets tackles question 1 concerning how to represent objects.

An important fact to note is that while LabelFusion uses an *explicit* object representation, namely 6-DOF pose, DenseObjectNets is an *implicit* object representation. In this case *implicit* means that the individual values of the descriptors don't have meaning by themselves, but rather are useful when used to solve the correspondence problem. In general when one moves from supervised learning to self-supervised or

unsupervised learning you generally tradeoff an explicit representation for an implicit representation.

The success of DenseObjectNets in generalization at the category level directly inspired kPAM (Chapter 4). In particular we were motivated by the challenge of performing a non-trivial manipulation task, e.g. hang a mug on a rack by the handle, for many different objects in a category. This type of task is a step more difficult than the tasks considered in DenseObjectNets, e.g. grasping an object by a specific point. The main contribution of this Chapter is a novel formulation of the category-level pick and place problem that uses semantic 3D keypoints as the object representation. We also contribute a manipulation pipeline that factors the problem into 1) instance segmentation, 2) 3D keypoint detection, 3) optimization-based robot action planning 4) geometric grasping and action execution. The kPAM pipline addresses questions 1 and 2 in Section 1.1, namely how to represent objects and how to specify tasks.

At the start of the kPAM project we planned to use dense-descriptors as our object representation however we ran into a few challenges that eventually led us to represent objects using semantic 3D keypoints instead. Because the task was more involved we needed to integrate with our object representation with a downstream task planner. Since the physical world is 3D moving objects around in space also fundamentally requires a 3D object representation. Although it is very powerful DenseObjectNets is a fundamentally 2D representation, since it lives in pixel space. It can be lifted into 3D using a depth image, but this doesn't alleviate the problem of occlusions, particularly self-occlusion. The second challenge is that because the category-level generalization in DenseObjectNets emerges naturally without any supervision it is difficult to ensure that the representation generalizes sufficiently well across the set of objects you are interested in. In particular we noticed that for the shoe category, the descriptors generalized well across different sneakers. However, when faced with a high heel, which visually looks very different even thought it is part of the shoe category, the descriptors didn't generalize as well. For these reasons we chose to return to a more *explicit* object representation, namely semantic 3D keypoints. This representation solved the aforementioned problems of being in 3D and generalizing

across objects in the category, but comes at the cost of requiring human supervision. One advantage of having human supervision is that if you are failing to accurately detect the keypoints on an object of interest there is a straightforward procedure for improving the performance of the vision system. One can simply collect and label more data for that object (or similar objects, e.g. high heels) and add it to the training set. One final point to note is that kPAM is a *sparse* representation rather than the *dense* representation of DenseObjectNets. This is an important distinction since downstream modules (e.g. planners, controllers) that consume visual representations typically consume a sparse representation (e.g. keypoints, 6DOF pose) rather than dense representations (e.g. descriptor images, segmentation maps, etc.). The move to a sparse representation is also a theme of Chapters 5 and 6.

So far Chapters 3 and 4 have tackled questions 1 and 2 posed in Section 1.1 but have not considered the third question, namely "how to enable robots to perform closed-loop feedback control for manipulation." Achieving closed-loop visual feedback for manipulation was the goal of the work in Chapter 5. The main contribution was a novel formulation of visuomotor policy learning using self-supervised correspondence. We used this novel formulation in an imitation learning framework to perform closed-loop feedback for manipulation tasks. By taking advantage of visual-correspondence learning to pre-train the visual system we were able to perform very sample efficient imitation learning, achieving generalization across challenging scenarios (category-level manipulation, deformable objects, visually challenging scenes) and with limited data (between 50 and 150 demonstrations).

Chapter 6 also focuses on the question of "how to enable robots to perform closed-loop feedback control for manipulation?", but takes a different approach to the one of Chapter 5. In particular Chapter 5 relied on imitation learning to get a feedback controller, thus reducing the challenge of closed-loop control to a supervised learning problem. Although this approach can be very powerful and has several desirable properties, it still suffers from all the limitations imitation learning. In particular it requires expert demonstrations and cannot be used to achieve goals other than the ones it was trained on. Instead Chapter 6 uses self-supervised correspondence

as visual input to a learned predictive model. By learning a predictive model of the environment dynamics we are able to leverage online model-predictive-control to synthesize a feedback controller. We demonstrate the effectiveness of this approach against competing baselines in a variety of simulated experiments and show that the learned predictive models also extend to real-world manipulation tasks.

7.2 Future Directions

Although the work in this thesis has made progress towards answering the questions posed at the outset in Section 1.1, much more remains to be done. The endeavor of robotic manipulation has the goal of building robots that can accomplish useful manipulation tasks *in the wild*. Getting there will require advances in several areas.

Scaling up visual learning: If we want to deploy robotic manipulation systems in the wild we will need perception systems that can handle all the variability and challenge that this entails. Self-supervised approaches such as DenseObjectNets are promising in this regard since they don't require human annotations. On the other hand human understandable representations (e.g. semantic keypoints of kPAM, 6DOF object pose) are useful in communicating tasks to the robot. I believe that finding ways to combine self-supervision with small amounts of explicit annotations provides a promising way forward.

Communicating tasks to robots: A challenge that is often overlooked in manipulation is how to specify the task the robot. Whether using classical model based approaches, such as task and motion planning (TAMP), or reinforcement learning approaches we must still specify the task to the robot. For a TAMP it could be a set of costs and constraints on geometric primitives and for reinforcement learning it is encapsulated in the reward function. In general specifying a complex long-horizon task such as "make dinner and clean up the kitchen" remains a challenge, independent of which approach (model-based or RL) is taken. How a task is specified is intimately related to how one represents the world and the objects in it. As shown in Chapter 5 human demonstrations can be an effective way to demonstrate a task, but more

work is needed to scale up such approaches to longer horizon tasks in a flexible and composable manner.

Longer horizon tasks: In order to accomplish longer horizon tasks such as "make dinner and clean up the kitchen" I believe that we will need to combine the learning based approaches developed in this thesis with planners, such as task and motion planners, that can reason over longer horizons. The kPAM approach of Chapter 4 is a small step in this direction, and shows how to connect learning-based and classical approaches, but more work is needed in this area.

Appendices

Appendix A

Dense Object Nets

A.1 Experimental Hardware

All of our data was collected using an RGBD camera mounted on the end effector of a 7 DOF robot arm (see Figure A-1). We used a Kuka IIWA LBR robot with a Schunk WSG 50 parallel jaw gripper. A Primesense Carmine 1.09 RGBD sensor was mounted to the Schunk gripper and precisely calibrated for both intrinsics and extrinsics.

A.2 Experimental Setup: Data Collection and Pre-Processing

As discussed in the paper all of our data consists of 3D dense reconstructions of a static scene. To collect data for a single scene we place an object (or set of objects) on a table in front of the robot. We then perform a scanning pattern with the robot which allows the camera to capture many different viewpoints of the static scene. This procedure takes about 70 seconds during which approximately 2100 RGBD frames are captured (the sensor outputs at 30Hz). Using the forward kinematics of the robot, together with our camera extrinsic calibration, we also record the precise camera pose corresponding to each image. Since we have the camera poses corresponding to each depth image we use TSDF fusion [17] to obtain a dense 3D reconstruction. Although we use our robot's

(a) (b)

Figure A-1: (a) Kuka IIWA LRB robot arm. (b) Schunk WSG 50 gripper with Primesense Carmine 1.09 attached

forward kinematics to produce the dense 3D reconstruction together with camera pose tracking, any dense SLAM method (such as [95]) could be used instead. In practice we found that using the robot's forward kinematics to obtain camera pose estimates produces very reliable 3D reconstructions which are robust to lack of geometric or RGB texture, varying lighting conditions, etc. Next we obtain new depth images for each recorded camera frame by rendering against the 3D reconstruction using our camera pose estimates. This step produces depth images which are globally consistent and free of artifacts (i.e. missing depth values, noise etc.). This keeps the process of finding correspondences across RGB images as a simple operation between poses and depth images. To enable the specific training techniques discussed in the paper we also need to know which parts of the scene correspond to the objects of interest. To do this we implemented the change detection technique of [22]. In practice since all of our data was collected on a tabletop, and our reconstructions can be easily globally aligned (due to the fact that we know the global camera poses from the robot's forward kinematics) we can simply crop the reconstruction to the area above the table. Once we have isolated the part of the reconstruction corresponding to the objects of interest, we can easily render binary masks via the same procedure as was used to generate the depth images.

Our RGBD sensor captures images at 30Hz, so we downsample the images to avoid having images which are too visually similar. Specifically we downsample images so

that the camera poses are sufficiently different (at least 5cm of translation, or 10 degrees of rotation). After downsampling we are left with approximately 315 images per scene.

In between collecting RGBD videos, the object of interest should be moved to a variety of configurations, and the lighting can be changed if desired. While for many of our data collections a human moved the object between configurations, we have also implemented and demonstrated (see our video) the robot autonomously rearranging the objects, which highly automates the object learning process. We employ a Schunk two-finger gripper and plan grasps directly on the object point cloud (see Appendix A.3). If multiple different objects are used, currently the human must still switch the objects for the robot and indicate which scenes correspond to which object, but even this information could be automated by the robot picking objects from an auxiliary bin and use continuity to simply identify which logs are of the same object.

A.3 Grasping Pipeline

While our learned visual descriptors can help us determine *where* to grasp, they can't be used during the bootstrapping phase before visual learning has occurred, and they don't constrain the 6DOF orientation of the gripper. Accordingly to choose grasps our pipeline employs simple geometric point cloud based techniques. There are two types of grasping that are performed in the paper. The first is performed while grasping the object to automatically reorient it during data collection, during the visual learning process. To achieve this we first use the depth images and camera poses to fuse a point cloud of the scene. We then randomly sample many grasps on the point cloud and prune those that are in collision. The remaining collision free grasps are then scored by a simple geometric criterion that evaluates the grasps for antipodality. The highest scoring grasp is then chosen for execution. The second type of grasping is, as in Section 5.4 of the paper, when we are attempting to grasp a specific point. In this case the robot moves to a handful of pre-specified poses and records those RGBD images. The RGB images are used to look up the best descriptor match and determine

a pixel space location to grasp in one image, and the corresponding depth image and camera pose are used to determine the 3D location of this point. The handful of depth images are also fused into a point cloud, and the grasping procedure is almost the same as the first type, with the slight modification that all the randomly sampled grasps are centered around the target grasp point. Although there are a variety of learning based grasping algorithms [38, 71] that could have been used, we found that our simple geometric based grasp planning was sufficient for the tasks at hand.

A.4 Network Architecture and Training Details

For our network we use 34-layer, stride-8 ResNet (pretrained on ImageNet), and then bilinearly upsample to produce a full resolution $640x480$ image.

For training, at each iteration we randomly sample between some specified subset of specified image comparison types (Single Object Within Scene, Different Object Across Scene, Multi Object Within Scene, Synthetic Multi Object). The weighting between how often each type is chosen is done via specifying their probabilities of being selected. Once the type has been sampled we then sample some set of matches and non-matches for each (around 1 million in total). Each step of the optimizer uses a single pair of images.

All the networks were trained with the same optimizer settings. Networks were trained for 3500 steps using an Adam optimizer with a weight decay of $1e-4$. Training was performed on a single Nvidia 1080 Ti or Titan Xp, a single step takes approximately 0.22 seconds, i.e. approximately 13 minutes, and so together with collecting a handful of scenes the entire training for a new object can take 20 minutes. The learning rate was set to $1e-4$ and dropped by 0.9 every 250 iterations. The networks trained with procedure **specific** used a 50-50 split of within scene image pairs and across scene image pairs 50% of the time. For the network used to grasp the heel of the red/brown shoe in Section 5.4 we sampled equally the three data types (Single Object Within Scene, Different Object Across Scene, Synthetic Multi Object).

A.4.1 Descriptor Projection to Unit Sphere

There is one additional small feature we discovered prior to camera-ready submission that gives substantial quantitative performance gains, although it was not used nor needed for most of the experiments including all hardware experiments. As is standard in many metric learning works, for example [113], we can add a simple parameterless normalization layer in which we project all features to the unit sphere, $f(I)(u) \leftarrow \frac{f(I)(u)}{||f(I)(u)||}$. This is contrast to channel-wise normalization mentioned in [15]. Given that there is a projection to the unit-sphere manifold, higher dimensions are needed in order to see improvements. While without unit-sphere projection, we see saturation of performance at around $D = 3$ for single-object descriptors, with unit-sphere projection we see significant gains in going from $D = 4$ to $D = 8$ even for single-object descriptors. Further work could investigate the effect of unit-sphere projection on descriptor consistency across classes. The network marked as **standard-SO-P** in Figures 3b and 3c was trained identically to **standard-SO**, but with the unit sphere projection, and with $D = 16$.

A.4.2 Additional Approaches Which Did not Improve Performance

During the course of our experimentation we found that the network architecture and training procedure outlined in Section 3.3 gave the best performance. However we also tried a variety of other network architectures and loss functions which did not improve performance. We discuss a few of them here.

Triplet loss [113] uses a triplet loss instead of the contrastive loss. We implemented a pixelwise version of this triplet loss, but found that it actually reduced performance on our dataset.

Scaling loss by pixel distance Our loss function tries to ensure that non-matches have descriptors that are at least a margin M apart. It can be hard, however, for the network to take two pixels that are next to each other, and assign them significantly different descriptors. In an effort to try to less heavily penalize non-matches that are

close to the true match, we introduced an additional scaling term using the L2 pixel distance. Following the notation from Section 3.3 let $u_a \in I_a, u_b \in I_b$ correspond to a non-match. Additionally, using the notation of Equation 3.6 let $u_b^* \in I_b$ correspond to the true match for u_a. Define $\Delta(u_b, u_b^*) \in \mathbb{R}$ to be the L2 distance, measured in pixels, between u_b and u_b^*. The scaled non-match loss is now defined as

$$\mathcal{L}_{\text{non-matches}}(I_a, I_b) = \frac{1}{N_{\text{non-matches}}} \sum_{N_{\text{non-matches}}} \frac{1}{M_p} \min(\Delta(u_b, u_b^*), M_p) \cdot \max(0, M - D(I_a, u_a, I_b, u_b))^2 \tag{A.1}$$

where M_p is the pixel distance at which this additional loss component saturates. Parameter sweeps on M_p, where $M_p = 1$ is the original un-scaled loss function, did not show significant differences in performance. A potentially useful extension for future work would be to try scaling by the geodesic distance in the 3D reconstructios.

Convolutional spatial transformer: We implemented a convolutional spatial transformer, as described in [15]. The convolutional spatial transformer is meant to help the network achieve scale and rotation invariance for each feature. We did not find any significant performance gains with our implementation of such a convolutional spatial transformer. A hypothesis for why we did not see performance gains is that our used network architecture (34-layer ResNet) was significantly deeper than the architecture used in [15], and our data collection and augmentation provided significant variety in scale and rotation – accordingly, our network must approximate scale and rotation invariance in order to fit the training data.

Ratios for sampling non-matches: Given a pixel u_a corresponding to a point on an object, non-matches u_b can either be on the object, or on the background. Let $I_{b,object}$ denote object pixels in image I_b, and $I_{b,background}$ denote background pixels. During training we experimented with varying the fraction of non-matches u_b which lie in $I_{b,object}$ vs. $I_{b,background}$. In general we found performance was insensitive to this ratio as long as the fraction of non-matches in $I_{b,object}$ was between 25% to 75%.

Appendix B

kPAM

B.1 Robot Hardware

Our experimental setup consists of a robot arm, an end-effector mounted RGBD camera and a parallel jaw gripper. Our robot is a 7-DOF Kuka IIWA LBR. Mounted on the end-effector is a Schunk WSG 50 parallel jaw gripper. Additionally we mount a Primesense Carmine 1.09 RGBD sensor to the gripper body.

B.2 Dataset Generation and Annotation

In order to reduce the human annotation time required for neural network training we use a data collection pipeline similar to that used in [31]. The main idea is to collect many RGBD images of a static scene and perform a dense 3D reconstruction. Then, similarly to [75], we can label the 3D reconstruction and propagate these labels back to the individual RGBD frames. This 3D to 2D labelling approach allows us to generate over 100,000 labelled images with only a few hours of human annotation time.

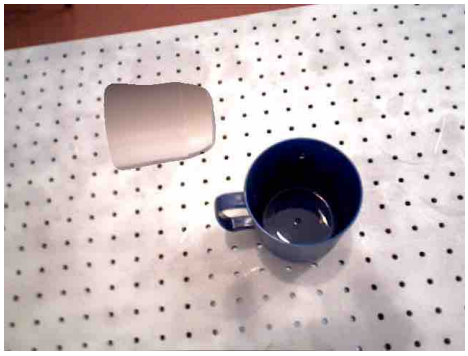
B.2.1 3D Reconstruction and Masking

Here we give a brief overview of the approach used to generate the 3D reconstruction, more details can be found in [31]. Our data is made up of 3D reconstructions of a static scene containing a single object of interest. Using our the wrist mounted camera on the robot, we move the robot's end-effector to capture a variety of RGBD images of the static scene. From the robot's forward kinematics, we know the camera pose corresponding to each image which allows us to use TSDF fusion [17] to obtain a dense 3D reconstruction. After discarding images that were taken from very similar poses, we are left with approximately 400 RGBD images per scene.

The next step is to detect which parts of the 3D reconstruction correspond to the object of interest. This is done using the change detection method described in [22]. In our particular setup all the reconstructions were of a tabletop scene in front of the robot. Since our reconstructions are globally aligned (due to the fact that we use the robot's forward kinematics to compute camera poses), we can simply crop the 3D reconstruction to the area above the table. At this point we have the portion of the 3D reconstruction that corresponds to the object of interest. This, together with the fact that we have camera poses, allows us to easily render binary masks (which segments the object from the background) for each RGBD image.

B.2.2 Instance Segmentation

The instance segmentation network requires training images with pixelwise semantic labels. Using the background subtraction technique detailed in Section B.2.1, we have pixelwise labels for all the images in our 3D reconstructions. However, these images contain only a single object, while we need the instance segmentation network to handle multiple instances at the test time. Thus, we augment the training data by creating multi-object composite images from our single object annotated images using a method similar to [114]. We crop the object from one image (using the binary mask described in Section B.2.1) and paste this cropped section on top of an existing background. This process can be repeated to generate composite images with arbitrary



(a) Mugs composite image

(b) Shoes composite image

Figure B-1: Multi object composite images used in instance segmentation training

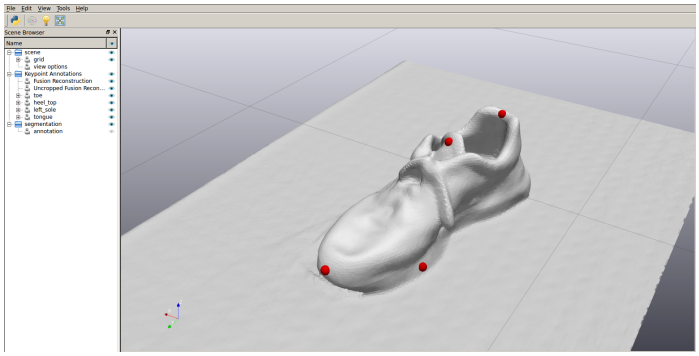
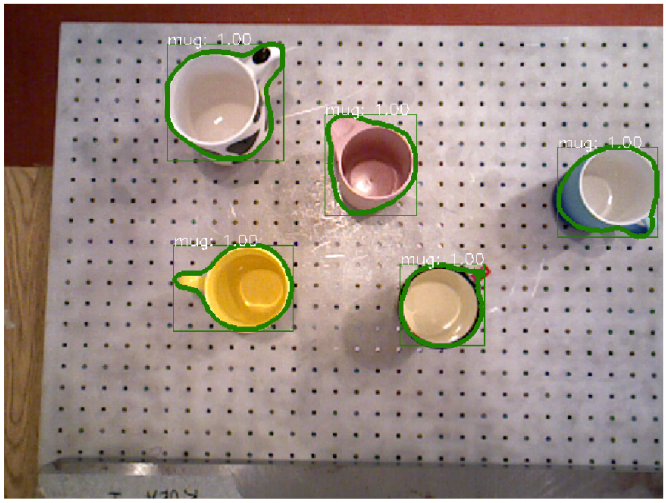


Figure B-2: A screenshot from our custom keypoint annotation tool.

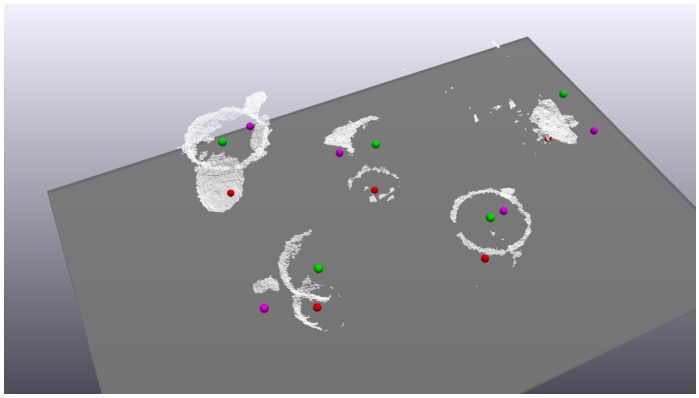
numbers of object. Examples of such images are shown in Figure B-1.

B.2.3 Keypoint Detection

The keypoint detection network requires training images annotated with pixel coordinates and depth for each keypoint. As mentioned in Section 4.3.2, we annotate 3D keypoints on the reconstructed mesh, transform the keypoints into the camera frame and project the keypoints into each image. To annotate the 3D keypoints on the reconstructed mesh, we developed a custom labelling tool based on the Director [77] user interface, shown in Figure B-2. We labelled a total of 117 scenes, 43 of which were shoes and 74 of which were mugs. Annotating these scenes took only a few hours and resulted in over 100,000 labelled images for keypoint network training.



(a) RGB image used for keypoint detections with Mask R-CNN annotations overlaid



(b) Keypoint detections

Figure B-3: 3D visualization of pointcloud and keypoint detections for the image from (a). The keypoints are colored as in Figure 4-6. The *top center* keypoint is green, the *bottom center* keypoint is red, and the *handle center* keypoint is purple.

B.3 Neural Network Architecture and Training

B.3.1 Instance Segmentation

For the instance segmentation, we used an open source Mask R-CNN implementation [81]. We used a R-101-FPN backbone that was pretrained on the COCO dataset [65]. We then fine-tuned on a dataset of 10,000 images generated using the procedure outlined in Section B.2.2. The network was trained for 40,000 iterations using the default training schedule of [81].

B.3.2 Keypoint Detection

We modify the integral network [129] for 3D keypoint detection. The network takes images cropped by the bounding box from MaskRCNN as the input. The network produces the probability distribution map $g_i(u, v)$ that represents how likely keypoint i is to occur at pixel (u, v), with $\sum_{u,v} g_i(u, v) = 1$. We then compute the expected values of these spatial distributions to recover a pixel coordinate of the keypoint i:

$$[u_i, v_i]^T = \sum_{u,v} [u \cdot g_i(u, v), v \cdot g_i(u, v)]^T \tag{B.1}$$

For the z coordinates (depth) of the keypoint, we also predict a depth value at every pixel denoted as $d_i(u, v)$. The depth of the keypoint i can be computed as

$$z_i = \sum_{u,v} d_i(u, v) \cdot g_i(u, v) \tag{B.2}$$

Given the training images with annotated pixel coordinate and depth for each keypoint, we use the integral loss and heatmap regression loss (see Section 2 of [129] for details) to train the network. We use a network with a 34 layers Resnet as the backbone. The network is trained on a dataset generated using the procedure described in Section B.2.3.

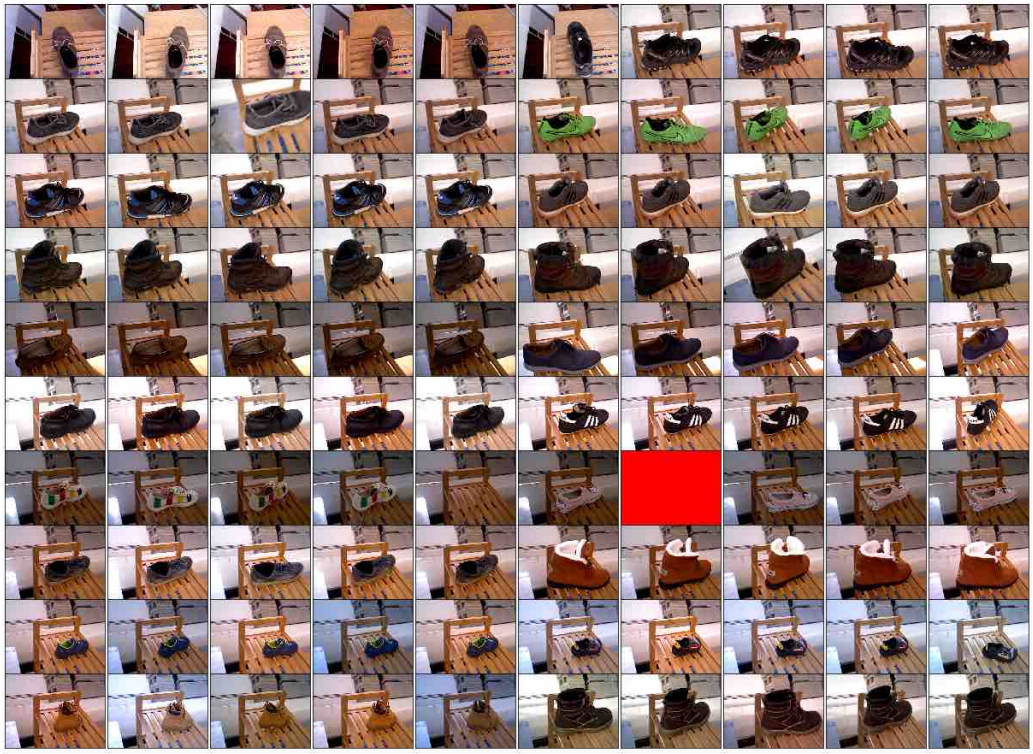
B.4 Experiments

Figures B-4, B-5, B-6 illustrate the results of experiments. These figures containing tiled images showing thee initial RGB image used for keypoint detection, along with an image of the object after running the kPAM pipeline. In the following sections we discuss more details related to the mug on shelf and mug on rack experiments.

B.4.1 Mugs Upright on Shelf

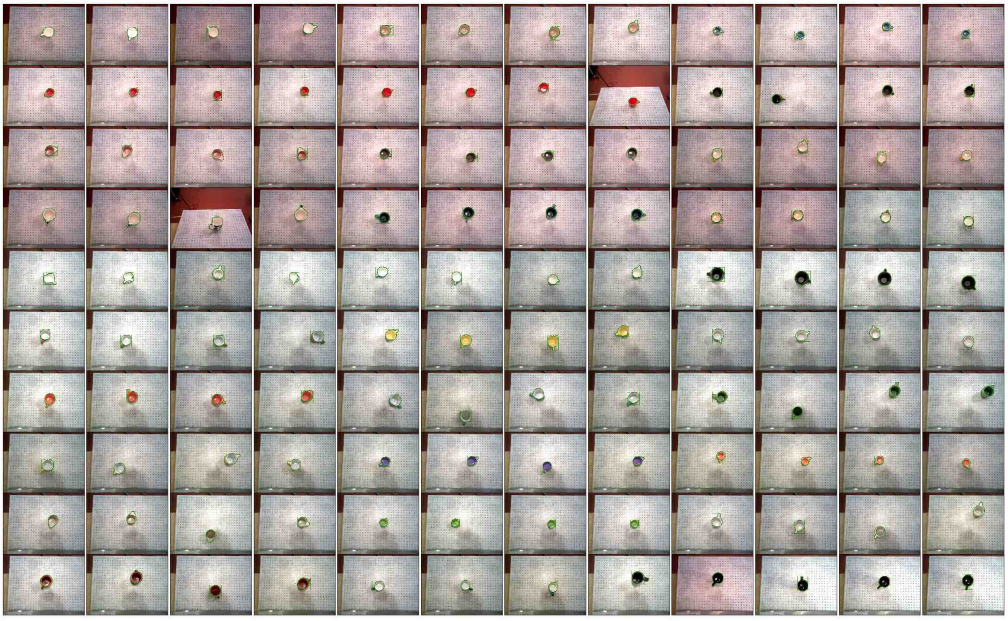
Results for the mug on shelf experiment are detailed in Figure 4-7. A trial was classified as a sucess if the mug ended up upright on the shelf with it's bottom center

(a) RGB images used for keypoint detection in shoe on rack experiments



(b) Image of the shoe rack after running the kPAM pipeline. Red images indicate trials where no image of the final placement was captured due to an upstream failure of the pipeline causing the trial to be aborted.

Figure B-4: Before and after images of the shoe on rack experiment for all 100 trials.



(a) RGB images used for keypoint detection in mug on rack experiments

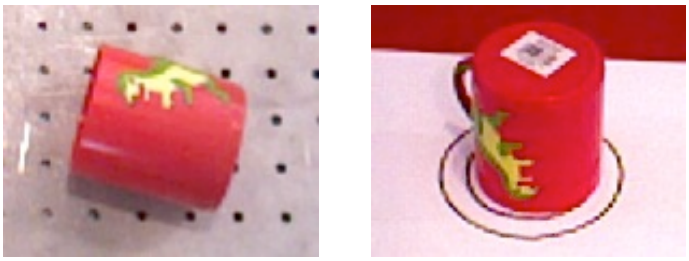
(b) Image of the mug rack after running the kPAM pipeline. Red images indicate trials where no image of the final placement was captured due to an upstream failure of the pipeline causing the trial to be aborted.

Figure B-5: Before and after images of the mug on rack experiments for all 120 trials.

(a) RGB images used for keypoint detection in mug on shelf experiments

(b) Image of the mug shelf after running the kPAM pipeline. Red images indicate trials where no image of the final placement was captured due to an upstream failure of the pipeline causing the trial to be aborted.

Figure B-6: Before and after images of the mug on shelf experiments for all 118 trials.



(a) (b)

Figure B-7: (a) The RGB image for the single failure trial of the mug on shelf task that led to the mug being put in an incorrect orientation. In this case the keypoint detection confused the top and bottom of the mug and it was placed upside down. (b) The resulting upside down placement of the mug.

keypoint within 5cm of the target location. Out of 118 trials we experienced 2 failures. One failure was due to a combination of inaccurate keypoint detections together with the mug being torqued as it was grasped. Since we only have a wrist mounted camera we cannot re-perceive the object to compensate for the fact that the object moves during the grasping process. As discussed in Section 4.6 this could be alleviated by adding an externally mounted camera.

The other failure was resulted from the mug being placed upside down. Figure B-7 shows the RGB image used for keypoint detection, along with the final position of the mug. As discussed in Section 4.5.2 this failure occurred because the keypoint detection confused the top and bottom of the mug. Given that the image was taken from a side view where the handle is occluded and it is difficult to distinguish top from bottom is understandable that the keypoint detection failed in this case. There are several ways to deal with this type of issue in the future. One approach would be to additionally predict a confidence value for each keypoint detection. This would allow us to detect that we were uncertain about the keypoint detections in Figure B-7 (a). We could then move the robot and collect another image that would allow us to unambiguously detect the keypoints.

Figure B-8: The 5 mugs on the left are the test mugs used in experiment that were characterized as *small*. For comparison the four mugs on the right are part of the *regular* category.

B.4.2 Hang mug on rack by its handle

As discussed in Section 4.5.3 mugs were divided into two groups, *regular* and *small*, based on their size. A mug was characterized as *small* if the handle had a minimum dimension (either height or width) of less than 2cm. Examples of mugs from each category are shown in Figure B-8. Mugs with such small handle sizes presented a challenge for our manipulation pipeline since hanging them on the rack requires increased precision.

Appendix C

Self-Supervised Correspondence in Visuomotor Policy Learning

C.1 Simulation Tasks

Our simulation environment was configured to closely match our real hardware experimentation. Using Drake [132], we simulate the 7-DOF robot arm, gripper, objects, and multi-view RGBD sensing. ***"Reach T only"***: goal is to move the end-effector to a target position relative to the sugar box object; success is within 1.2cm of target. The box pose only varies in translation, not rotation; training positions drawn from a truncated Gaussian (σ_x = 5cm,σ_y = 10cm), centered on the table, truncated to a 40cm×10cm region. Test distribution drawn from uniform over same region. ***"Reach T + R"***: same as *"Reach T only"* but now the box pose varies in rotation as well, drawn from a uniform [-30,30] degrees; success is within 1.2cm and 2 degrees. ***"Push box"***: goal is to push the box object across the table, and the box is subject to random external disturbances; success if translated across table and final box orientation is within 2 degrees of target. ***"Push plate"***: goal is to push a plate across a table to a specific goal location, and the plate is subject to external disturbances; success if plate center is within 1cm of target position.

C.2 Policy Networks

All experiments using an ***"MLP"*** had a two-layer network with 128 hidden units, 20% dropout, in each layer and ReLU nonlinearities. Training was 75,000 steps with RMSProp, $\alpha = 0.9$, with a batch size of 16, and lr starting at $1e-4$, and decaying by a factor of 0.5 every 10,000 steps. All experiments using an ***"LSTM"*** had a single LSTM layer with 100 units preprocessed by two MLP layers of 100 units, 10% dropout, and layer-normalized prior to the LSTM layer. Training was 200,000 steps with RMSProp, $\alpha = 0.9$, with lr starting at $2e-3$, decaying 0.75 every 40,000 steps, with truncated backpropagation of maximum 50 steps, and gradient clipping of maximum magnitude 1.0. As recommended in [103] we train LSTMs on downsampled trajectories, we use 5 Hz.

C.3 Vision Networks

Both ***"AE"*** and ***"E2E"*** methods used an identical architecture, with the only difference being the additional decoder used for the AE method during autoencoding. The network is almost exactly as in [64] and [27], but we provided a full-width image, 320×240. We used 16 2D feature points. ***"DD"*** architecture is identical to [31]. DD-2D computes image-space spatial expectation, DD-3D computes 3D-space spatial expectation using the depth image, see [30] for details; both used 16 descriptors. The ***"E2E (34-layer)"*** network is exactly the DD architecture but with $D = 16$ and channel-wise 2D spatial softmax to obtain $\boldsymbol{z}$.

Appendix D

Keypoints into the Future: Self-Supervised Correspondence with Model-Based Reinforcement Learning

D.1 Dense Correspondence

Here we give a brief overview of the dense correspondence model formulation [112, 31] with spatial distribution losses [32, 30]. We briefly explain the loss functions and how the descriptors $\{d_i\}$ are localized.

D.1.1 Network Architecture

We use an architecture that produces a full resolution descriptor image. Namely it maps

$$W \times H \times 3 \rightarrow W \times H \times D \tag{D.1}$$

We use a FCN (fully-convolutional network architecture) [67] with a ResNet-50 or ResNet-101 with the number of classes set to the descriptor dimension. Note that the FCN used in this work does not use striding and upsampling as in the architecture originally used in [31].

D.1.2 Loss Function

For all shown experiments we use the spatially-distributed loss formulation with a combination of heatmap and 3D spatial expectation losses, as described in [32], Chapter 4.

Heatmap Loss

Let p^* be the pixel space location of a ground truth match. Then we can define the ground-truth heatmap as

$$H^*_{p^*}(p) = \exp\left(-\frac{||p - p^*||_2^2}{\sigma^2}\right) \tag{D.2}$$

p represents a pixel location. A predicted heatmap can be obtained from the descriptor image $\mathcal{I}_D$ together with a reference descriptor d^*. Then the predicted heatmap is gotten by

$$\hat{H}(p; d^*, \mathcal{I}_D, \eta) = \exp\left(-\frac{||\mathcal{I}_D(p) - d^*||_2^2}{\eta^2}\right) \tag{D.3}$$

The heatmap can also be normalized to sum to one, in which case it represents a probability distribution over the image.

$$\tilde{H}(p) = \frac{\hat{H}(p)}{\sum_{p' \in \Omega} \hat{H}(p')} \tag{D.4}$$

The heatmap loss is simply the MSE between H^* and $\hat{H}$ with mean reduction.

$$L_{\text{heatmap}} = \frac{1}{|\Omega|} \sum_{p \in \Omega} ||\hat{H}(p) - H^*(p)||_2^2 \tag{D.5}$$

Spatial Expectation Loss

Given a descriptor d^* together with a descriptor image $\mathcal{I}_D$ we can compute the 2D spatial expectation as

$$J_{\text{pixel}}(d^*, I_D, \eta) = \sum_{p \in \Omega} p \cdot \tilde{H}(p; d^*, I_D, \eta) \tag{D.6}$$

If we also have a depth image $\mathbf{Z}$ then we can define the spatial expectation over the depth channel as

$$J_z(d^*, I_D, \mathcal{Z}, \eta) = \sum_{p \in \Omega} \mathbf{Z}(p) \cdot \tilde{H}(p; d^*, I_D, \eta) \tag{D.7}$$

The spatial expectation loss is simply the L1 loss between the ground truth and estimated correspondence using

$$L_{\text{spatial pixel}} = ||p^* - J_{\text{pixel}}(d^*)||_1 \tag{D.8}$$

We can also use our 3D spatial expectation J_z to compute a 3D spatial expectation loss. In particular given a depth image $\mathbf{Z}$ let the depth value corresponding to pixel p be denoted by $\mathbf{Z}(p)$. The spatial expectation loss is simply

$$L_{\text{spatial z}} = ||\mathcal{Z}(p^*) - J_z(d^*, I_D, \mathcal{Z}, \eta)||_1 \tag{D.9}$$

being careful to only take the expectation over pixels with valid depth values $\mathcal{Z}(p)$.

Total Loss

The total loss is a combination of the heatmap loss and the spatial loss

$$L = w_{\text{heatmap}} L_{\text{heatmap}} + w_{\text{spatial}}(L_{\text{spatial pixel}} + L_{\text{spatial z}}) \tag{D.10}$$

where the w are weights.

D.1.3 Correspondence Function

The correspondence function $g_c(\mathcal{I}_D, d_i)$ in Section 6.3.2 is defined using the spatial expectations J_{pixel}, J_z defined above to localize the descriptor d_i in either pixel space or 3D space. If in 3D we additionally use the known camera extrinsics to express the localized point in world frame.

D.2 Training Details

This section provides details on the simulation and hardware experiments.

D.2.1 Trajectory Data Augmentation

Many physical systems exhibit invariances in their dynamics. For example, in the environments we consider the dynamics are invariant to translation in the xy plane and rotation about the z axis. In other words, if we translate or rotate our frame of reference the dynamics don't change. Encoding this invariance into our dynamics model has the potential to greatly simplify the learning problem. [46] achieves this by parameterizing the dynamics relative to the object frame, however this approach requires having access to the ground truth object frame and assumes you are dealing with a single rigid object. Another approach, taken by [159, 161, 128], is to rotate the observation into a frame defined by the action. While this can work well in the setting of simple manipulation primitives (e.g. push along a straight line for $5cm$) it doesn't naturally extend to the realtime feedback setting where you are commanding actions continuously at $5 - 10$ Hz without returning the robot to a reference position. Since in our approach the latent-state $\boldsymbol{z}$ is a physically grounded 3D quantity we are able to encode some of this invariance by using an alternative approach based on data augmentation. Given a latent-state action trajectory $\{(\boldsymbol{y}_t, \boldsymbol{a}_t)\}$ then transforming this trajectory using a homogeneous transform T, which consists of xy translation and z rotation, yields another valid trajectory $\{(\tilde{\boldsymbol{y}}_t, \tilde{\boldsymbol{a}}_t)\} = \{(T \cdot \boldsymbol{y}_t, T \cdot \boldsymbol{a}_t)\}$. At training time we augment the training trajectories by sampling such random homogeneous transforms T.

D.2.2 Training Details

All methods used the same architecture for the dynamics model $\hat{f}_{\theta_{dyn}}$, an MLP with two hidden layers of 500 units. All of our variants, along with the **transporter** baselines, use visual pre-training. The visual models are trained for 100 epochs and the model with the best test error is used. The dense-correspondence model uses both

Method	Learnable Parameters Θ
DS	$\{\theta_{dyn}\}$
SDS	$\{\theta_{dyn}\}$
WDS	$\{\theta_{dyn}, \alpha\}$
WSDS	$\{\theta_{dyn}, \alpha\}$
Transporter	$\{\theta_{dyn}\}$
Autoencoder	$\{\theta_{dyn}, \theta_{\text{autoencoder}}\}$

Table D.1: The set of learnable parameters for the different methods during the dynamics learning phase. For our methods and transporter, these parameters don't include the weights of the visual model which remain fixed during the dynamics learning phase.

camera views for visual pretraining, while the transporter model uses only the images from the camera used at test time. For the dynamics learning all methods are trained for 1000 epochs using an Adam optimizer [55] with a learning rate of 10^{-4}. For each method the model with the best test error was used for evaluation. Table D.1 details the set of learnable parameters for each method.

D.3 Online Model-Predictive Control

Following [92] we use the model-predictive path integral (MPPI) approach derived in [144]. Here we provide a brief overview but refer the reader to [144] for more details. MPPI is a gradient-free optimizer that considers coordination between timesteps when sampling action trajectories. The algorithm proceeds by sampling N trajectories, rolling them out using the learned model, computing the reward/cost for each trajectory, and then re-weighting the trajectories in order to sample a new set of trajectories. Let H be look-ahead horizon of the MPC, then a single trajectory consists of state-action pairs $\{(x_t^{(k)}, a_t^{(k)}\}_{t=0}^{H-1}$. Let $R_k = \sum_{t'=t}^{t+H-1} r(x_t^{(k)}, a_t^{(k)})$ be the reward of the k-th trajectory. Define

$$\mu_t = \frac{\sum_{k=0}^{N} \left(e^{\gamma R_k}\right) a_t^{(k)}}{\sum_{k=0}^{N} e^{\gamma R_k}}$$

A filtering technique is then used to sample new trajectories from the previously computed mean μ_t. Specifically

$$a_t^i = n_t^i + \mu_t \tag{D.11}$$

where the noise n_t^i is sampled via

$$u_t^i \sim \mathcal{N}(0, \Sigma), \;\; \forall i \in 0, \ldots, N-1, \;\; \forall t \in 0, \ldots, H-1 \tag{D.12}$$

$$n_t^i = \beta u_t^i + (1-\beta) n_{t-1}^i \quad \text{where } n_{t<0} = 0 \tag{D.13}$$

This procedure is repeated for M iterations at which point the best action sequence is selected. All of our experiments we used $N = 1000, M = 3, H = 10, \beta = 0.7$. The cost/reward in the MPC objective varied slightly between the hardware and simulation experiments, more details are provided below.

D.4 Simulation Experiments

To evaluate our method we consider four manipulation tasks in simulation. We use the Drake simulation environment [132] which provides both the underlying physics simulation and rendering of RGBD images at VGA resolution $640 \times 480 \times 3$. Figure 6-3 shows image from the four simulation tasks that we consider.

- **top down camera:** This environment, depicted in Figure 6-3 (a), consists of the sugar-box object from the YCB dataset [12] laying flat on a table. The robot is represented as cylindrical pusher (shown in green) and the action $\boldsymbol{a}$ is the $x-y$ velocity of the pusher in the plane. The environment timestep is $dt = 0.1$, so the agent must command actions at 10Hz. Two cameras are placed directly above the table, facing downwards. The camera positions are offset by 90 degrees about their z-axis. Our methods use both camera feeds for training the visual correspondence model, but only one camera feed at test time. An image from this camera is shown in Figure 6-3 (a). All other methods use only a single camera feed at both train and test time.

- **angled camera:** This environment is identical to *top down camera* but has different camera positions. Instead of being top down the two cameras are located on adjacent sides of the table and angled at 45 degrees, see figure 6-3 (b). The setting of angled cameras is more similar to our hardware experimental setup and is useful for comparing approaches that use pixel space vs. 3D representations.

- **occlusions:** This environment uses the same setup of task *angled camera* the only difference being that the object is now laying on its side, see Figure 6-3 (c). This, together with the angled camera position, means that occlusions become a significant factor. In particular as the box rotates through the full 360 degrees in yaw, the sides of the box become alternately occluded or visible. The top face of the box is the only one that remains unoccluded for all poses of the object.

- **mugs (category):** This environment has the same top-down camera placements and cylindrical pusher as task *angled camera*. Instead of a single object however, we use a collection of 10 different mug models and vary the color and texture on each episode. This environment tests category-level vision and dynamics generalization. Two mug instances are shown in Figures 6-3 (d) and (e).

D.4.1 Data Collection

For each environment we collect a static dataset that is then used to learn the visual dynamics model. All methods have access to exactly the same dataset and the visual pretraining for our method and the **transporter** baseline is done using this same dataset. For each task the dataset is generated by collecting 500 trajectories of length 40 using a scripted random policy. The simulator timestep is 0.1 seconds so a trajectory of length 40 equates to 4 seconds.

D.4.2 Evaluating closed-Loop MPC performance

For each environment we evaluate the different methods by planning to a desired goal-state image and computing the pose error (both translation and rotation) using

the ground truth simulator state. Goal states are generated sampling a random control input and applying it to the environment for 15 time steps. We further require that goal states are sufficiently far from initial states (in both translation and rotation). This generates a diverse set of initial and goal state pairs for evaluation. The simulator state is then reset to the initial state and we use closed-loop MPC to control the system to the goal state. The MPC cost function is simply the L2 distance between the latent state and the goal state. Ground truth state information is used to compute the error between the final and goal poses for the object.

D.4.3 Baselines

To demonstrate the benefits of our approach over prior methods we compare against several baselines.

- **Ground Truth 3D points (GT_3D):** This baseline is used for tasks *top down camera, angled camera, occlusions* since those environment use just a single object. $\boldsymbol{z}_{\text{object}}$ contains ground truth world-frame 3D locations of 4 points on the object. Knowing the location of 4 points is equivalent to knowing the object pose for a rigid object. We believe that this is a strong baseline that provides an upper bound on what is achievable with our descriptor-based methods that attempt to track points on the object.

- **Transporter:** We use the *Transporter* autoencoder formulation from [59] to pre-train a visual model. Following the original paper we use 6 keypoints and freeze the visual model while training the dynamics model. We investigate two variants using the transporter approach. In **transporter 2D** $\boldsymbol{z}_{\text{object}}$ are the pixel-space locations of the keypoints. In **transporter 3D** $\boldsymbol{z}_{\text{object}}$ are the 3D world frame locations of the keypoints, computed from the pixel space by using the depth image together with the camera intrinsics and extrinsics.

- **Autoencoder:** This method jointly learns the visual model and the dynamics model. Specifically we jointly train a convolutional autoencoder together with

a forward dynamics model. The loss is a combination of the dynamics loss, Equation (6.1), and an image reconstruction loss, which penalizes the L2 distance between the reconstructed and actual images. Note that this is exactly the autoencoder baseline from [148]. Following [148] images are downsampled to 64×64 before being passed into the network. The encoder architecture contains 6 2D convolutions with kernel sizes $[3, 4, 3, 4, 4, 4]$, strides $[1, 2, 1, 2, 2, 2]$ and filter sizes $[64, 64, 64, 128, 256, 256]$. Leaky ReLU activations are added between the convolutional layers. The final output is flattened and passed through a fully-connected layer to form the latent-state $\boldsymbol{z}_{\text{object}}$. We experimented with different dimensions for $\boldsymbol{z}$ from 16 to 64 and found that 64 worked best. Hence a 64 dimensional latent state is used for all experiments. The decoder follows the one in [40] and consists of a dense layer followed by 4 transposed convolutions with kernels size 4 and stride 2 which upscales the output image to 64×64.

D.5 Hardware Experiments

D.5.1 Hardware Setup

We used a Kuka IIWA LBR robot with a custom cylindrical pusher attached to the end-effector to perform our hardware experiments, see Figure D-1. RGBD sensing was provided by two RealSense D415 cameras rigidly mounted offboard the robot and calibrated to the robot's coordinate frame. To enable effective correspondence learning between views, it is ideal to have views with *some* overlap such that correspondences exist, but still maintain different-enough viewpoints from each camera. At test time only a single camera is used to localize the dense-descriptor keypoints. The robot is controlled by commanding end-effector velocity in the xy plane at 5Hz. A high-rate Jacobian space controller consumes these 5Hz end-effector velocity commands and closes the loop to command the robot's joint positions at 200Hz.

Figure D-1: Overview of our experimental setup, including the two external Realsense D415 cameras. Images from both cameras are used to train the dense-descriptor model, while only the right camera is used at runtime to localize the keypoints on the object.

D.5.2 One-Shot Imitation Learning

Although our learned dynamics model together with online MPC is able to plan over short to medium horizons, we can track much longer horizon plans by providing a single demonstration and using a trajectory tracking cost in our MPC formulation. This demonstration can in principle come from any source, in our case we used a human teleoperating the robot. We capture observations throughout the trajectory at $5Hz$ resulting in a trajectory of observations $\{\boldsymbol{o}_t^*\}_{t=0}^T$. Using our visual model we convert these observations into keypoints $\{\boldsymbol{y}_t^*\}_{t=0}^T$ and latent state $\{\boldsymbol{z}_t^*\}_{t=0}^T$ trajectories. These trajectories are used to guide the MPC. In particular the cost/reward function in the MPC is

$$r(z_t, a_t) = -||z_t - z_t^*||_2^2 \tag{D.14}$$

This trajectory cost allows us to accurately track long horizon plans (where the demonstrations are as long as 15 seconds) using an MPC horizon of 2 seconds. The four demonstrations trajectories used in the hardware experiments are illustrated in Figure D-2.

D.5.3 Results

In this section we provide more details on the hardware experiments from Section 6.4.4. Figure D-3 expands on Figure 6-4 showing the region of attraction of our MPC controller when attempting to stabilize the four different trajectories shown in Figure D-2. We define a trajectory as a success if the final object position is within 3 cm and 30 degrees of the goal position. Given that during trials we explicitly chose initial conditions to test the region of attraction of the MPC controller, success rates are not particularly meaningful, as the success rate depends on the initial condition. Table D.2 shows the average translational and angular errors among successful trajectories.

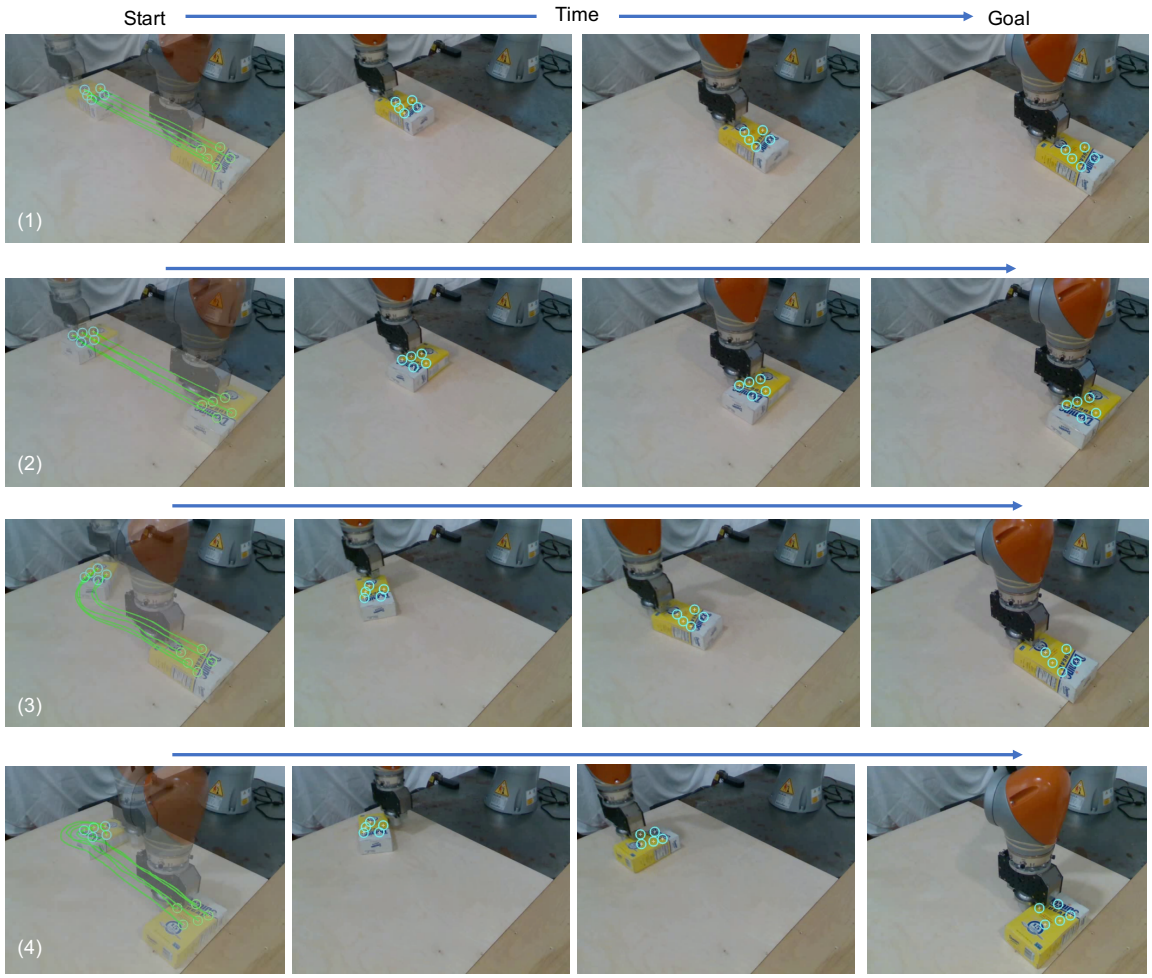


Figure D-2: The 4 demonstration trajectories used for the hardware experiments. The left image of each row shows the starting position blended with the goal position. The **SDS** keypoints are shown in teal for each frame. The green lines show the paths followed by the keypoints moving from the starting position to the final position. The right image of each row shows the final/goal position.

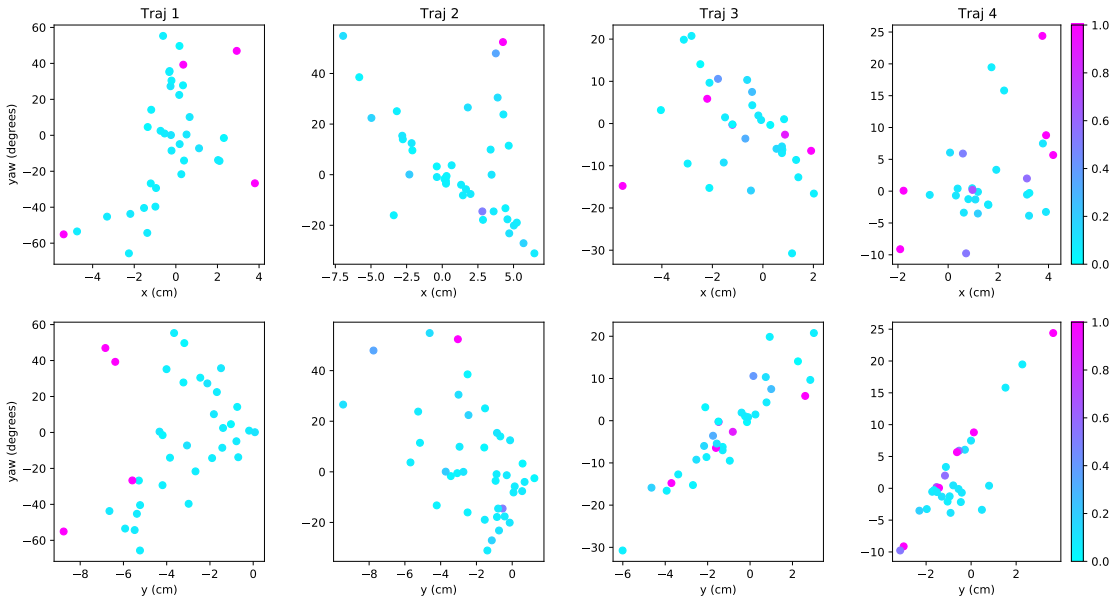


Figure D-3: Scatter plots show results of our approach on the four different reference trajectory tracking tasks. The axes of the plots show the deviation of the object starting pose from the initial pose of the demonstration. The top row shows the deviation in the x and yaw axes, while the bottom shows the deviation in the y and yaw axes. Axes are illustrated in Figure 6-4. The color indicates the distance between final and goal poses, lower cost is better. The numerical value is computed as cost $= \frac{\Delta\text{pos}}{3} + \frac{\Delta\text{angle}}{30}$ where $\Delta\text{pos}, \Delta\text{angle}$ are the translational (in centimeters) and angular (in degrees) errors between the object's final position and the goal position. The costs are rescaled and plotted in the range $[0, 1]$. The various reference trajectories, shown in Figure D-2, are of different difficulties, as reflected by the different regions of attraction of the MPC controller. Videos of the closed-loop rollouts can be found at project page.

Trajectory	pos (cm)	angle °	success rate	num trials
1	1.23 ± 0.55	5.25 ± 3.99	87.5%	40
2	1.10 ± 0.319	7.32 ± 4.08	89%	36
3	1.16 ± 0.58	3.80 ± 2.54	73%	33
4	1.44 ± 0.30	9.73 ± 5.48	65%	29

Table D.2: Quantitative results of hardware experiments. A trial is considered a success rate if the final object position was within 3 cm and 30 degrees of the goal pose. Note that, as shown in Figure D-3 the initial conditions were intentionally chosen to test the region of attraction of our controller, thus the success rates are not meaningful in and of themselves and are included only for completeness. The pos (cm) and angle columns show the deviation of the final object position from the target. Note that the mean and standard deviation are only calculated over the successful trials. This serves to give a sense of the accuracy that can be achieved by using our closed-loop MPC controller.

Bibliography

[1] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. "Learning to see by moving". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 37–45.

[2] Pulkit Agrawal et al. "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances in Neural Information Processing Systems*. 2016, pp. 5074–5082.

[3] Ilge Akkaya et al. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint arXiv:1910.07113* (2019).

[4] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. "Densepose: Dense human pose estimation in the wild". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7297–7306.

[5] Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *arXiv preprint arXiv:1808.00177* (2018).

[6] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[7] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[8] Aude Billard et al. "Robot programming by demonstration". In: *Springer handbook of robotics* (2008), pp. 1371–1394.

[9] Aude Billard et al. "Survey: Robot programming by demonstration". In: *Handbook of robotics* 59.BOOK_CHAP (2008).

[10] *Boston Dynamics Atlas*. URL: `https://www.youtube.com/watch?v=_sBBaNYex3E`.

[11] Eric Brachmann et al. "Learning 6d object pose estimation using 3d object coordinates". In: *European conference on computer vision*. Springer. 2014, pp. 536–551.

[12] Berk Calli et al. "Yale-CMU-Berkeley dataset for robotic manipulation research". In: *The International Journal of Robotics Research* 36.3 (2017), pp. 261–268. DOI: `10.1177/0278364917700714`. URL: `https://doi.org/10.1177/0278364917700714`.

[13] Yevgen Chebotar et al. "Path integral guided policy search". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3381–3388.

[14] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *arXiv:1606.00915* (2016).

[15] Christopher B Choy et al. "Universal correspondence network". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2414–2422.

[16] Nikolaus Correll et al. "Analysis and observations from the first amazon picking challenge". In: *IEEE Transactions on Automation Science and Engineering* 15.1 (2016), pp. 172–188.

[17] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: (1996).

[18] Angela Dai et al. "Scannet: Richly-annotated 3d reconstructions of indoor scenes". In: *arXiv preprint arXiv:1702.04405* (2017).

[19] Frederik Ebert et al. "Self-supervised visual planning with temporal skip connections". In: *arXiv preprint arXiv:1710.05268* (2017).

[20] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv preprint arXiv:1812.00568* (2018).

[21] Pedro F Felzenszwalb et al. "Object detection with discriminatively trained part-based models". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2010), pp. 1627–1645.

[22] Ross Finman et al. "Toward lifelong object segmentation from change detection in dense rgb-d maps". In: *Mobile Robots (ECMR), 2013 European Conference on*. IEEE. 2013, pp. 178–185.

[23] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in neural information processing systems*. 2016, pp. 64–72.

[24] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[25] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided cost learning: Deep inverse optimal control via policy optimization". In: *International Conference on Machine Learning*. 2016, pp. 49–58.

[26] Chelsea Finn et al. "Deep spatial autoencoders for visuomotor learning". In: *arXiv preprint arXiv:1509.06113* (2015).

[27] Chelsea Finn et al. "Deep spatial autoencoders for visuomotor learning". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 512–519.

[28] Chelsea Finn et al. “One-shot visual imitation learning via meta-learning”. In: *arXiv preprint arXiv:1709.04905* (2017).

[29] Michael Firman. “RGBD Datasets: Past, Present and Future”. In: *CVPR Workshop on Large Scale 3D Data: Acquisition, Modelling and Analysis*. 2016.

[30] Peter R. Florence. “Dense Visual Learning for Robot Manipulation”. PhD thesis. Massachusetts Institute of Technology, 2019.

[31] Peter R Florence, Lucas Manuelli, and Russ Tedrake. “Dense object nets: Learning dense visual object descriptors by and for robotic manipulation”. In: *Conference on Robot Learning (CoRL)* (2018).

[32] Peter Florence, Lucas Manuelli, and Russ Tedrake. “Self-Supervised Correspondence in Visuomotor Policy Learning”. In: *IEEE Robotics and Automation Letters* (2019).

[33] Wei Gao and Russ Tedrake. “FilterReg: Robust and Efficient Probabilistic Point-Set Registration using Gaussian Filter and Twist Parameterization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11095–11104.

[34] Wei Gao and Russ Tedrake. “SurfelWarp: Efficient Non-Volumetric Single View Dynamic Reconstruction”. In: Robotics: Science and Systems. 2018.

[35] Ali Ghadirzadeh et al. “Deep predictive policy training using reinforcement learning”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 2351–2358.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016.

[37] Marcus Gualtieri, Andreas ten Pas, and Robert Platt. “Pick and place without geometric object models”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7433–7440.

[38] Marcus Gualtieri et al. “High precision grasp pose detection in dense clutter”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 598–605.

[39] Raia Hadsell, Sumit Chopra, and Yann LeCun. “Dimensionality reduction by learning an invariant mapping”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2006, pp. 1735–1742.

[40] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *arXiv preprint arXiv:1811.04551* (2018).

[41] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[42] Hironori Hattori et al. “Learning scene-specific pedestrian detectors without real data”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3819–3827.

[43] Kaiming He et al. “Mask r-cnn”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2980–2988.

[44] Carlos Hernandez et al. "Team delft's robot winner of the amazon picking challenge 2016". In: *Robot World Cup*. Springer. 2016, pp. 613–624.

[45] Tomas Hodan et al. "T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects". In: *arXiv preprint arXiv:1701.05498* (2017).

[46] Francois R Hogan, Maria Bauza, and Alberto Rodriguez. "A Data-Efficient Approach to Precise and Controlled Pushing". In: *arXiv preprint arXiv:1807.09904* (2018).

[47] François Robert Hogan and Alberto Rodriguez. "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics". In: *arXiv preprint arXiv:1611.08268* (2016).

[48] Binh-Son Hua et al. "Scenenn: A scene meshes dataset with annotations". In: *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE. 2016, pp. 92–101.

[49] *itSeez3D*. URL: `https://itseez3d.com`.

[50] Stephen James, Michael Bloesch, and Andrew J Davison. "Task-Embedded Control Networks for Few-Shot Imitation Learning". In: *arXiv preprint arXiv:1810.03237* (2018).

[51] Stephen James, Andrew J Davison, and Edward Johns. "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task". In: *Conference on Robot Learning (CoRL)* (2017).

[52] Eric Jang et al. "End-to-End Learning of Semantic Grasping". In: *arXiv preprint arXiv:1707.01932* (2017).

[53] M. Johnson-Roberson et al. "Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?" In: *IEEE International Conference on Robotics and Automation*. 2017, pp. 1–8.

[54] Dmitry Kalashnikov et al. "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard et al. Vol. 87. Proceedings of Machine Learning Research. PMLR, 29–31 Oct 2018, pp. 651–673. URL: `http://proceedings.mlr.press/v87/kalashnikov18a.html`.

[55] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[56] Michael Krainin, Brian Curless, and Dieter Fox. "Autonomous generation of complete 3D object models using next best view manipulation planning". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 5031–5037.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *NIPS*. 2012, pp. 1097–1105.

[58] Oliver Kroemer, Scott Niekum, and George Konidaris. "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms". In: *arXiv preprint arXiv:1907.03146* (2019).

[59] Tejas Kulkarni et al. "Unsupervised Learning of Object Keypoints for Perception and Control". In: *arXiv preprint arXiv:1906.11883* (2019).

[60] Tejas Kulkarni et al. "Unsupervised learning of object keypoints for perception and control". In: *arXiv preprint arXiv:1906.11883* (2019).

[61] *LabelFusion*. `http://labelfusion.csail.mit.edu`.

[62] Brenden M Lake et al. "Building machines that learn and think like people". In: *Behavioral and brain sciences* 40 (2017).

[63] Michael Laskey et al. "Dart: Noise injection for robust imitation learning". In: *arXiv preprint arXiv:1703.09327* (2017).

[64] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[65] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[66] Ce Liu, Jenny Yuen, and Antonio Torralba. "Sift flow: Dense correspondence across scenes and its applications". In: *IEEE transactions on pattern analysis and machine intelligence* 33.5 (2011), pp. 978–994.

[67] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[68] Tomas Lozano-Perez, Matthew T Mason, and Russell H Taylor. "Automatic synthesis of fine-motion strategies for robots". In: *The International Journal of Robotics Research* 3.1 (1984), pp. 3–24.

[69] Reza Mahjourian, Martin Wicke, and Anelia Angelova. "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5667–5675.

[70] Jeffrey Mahler et al. "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1957–1964.

[71] Jeffrey Mahler et al. "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *arXiv preprint arXiv:1703.09312* (2017).

[72] Jeffrey Mahler et al. "Learning ambidextrous robot grasping policies". In: *Science Robotics* 4.26 (2019), eaau4984.

[73] Jeremy Maitin-Shepard et al. "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2308–2315.

[74] Lucas Manuelli et al. “kpam: Keypoint affordances for category-level robotic manipulation”. In: *arXiv preprint arXiv:1903.06684* (2019).

[75] P. Marion et al. “Label Fusion: A Pipeline for Generating Ground Truth Labels for Real RGBD Data of Cluttered Scenes”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 3235–3242. DOI: 10.1109/ICRA.2018.8460950.

[76] Pat Marion. *Director: A robotics interface and visualization framework*. 2015. URL: http://github.com/RobotLocomotion/director.

[77] Pat Marion. *Director: A robotics interface and visualization framework*. 2015. URL: http://github.com/RobotLocomotion/director.

[78] Pat Marion et al. “LabelFusion: A pipeline for generating ground truth labels for real rgbd data of cluttered scenes”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[79] Matthew T Mason. “Toward Robotic Manipulation”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 1–28.

[80] Matthew T Mason and J Kenneth Salisbury Jr. “Robot hands and the mechanics of manipulation”. In: (1985).

[81] Francisco Massa and Ross Girshick. *maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch*. https://github.com/facebookresearch/maskrcnn-benchmark. Accessed: [Insert date here]. 2018.

[82] Jan Matas, Stephen James, and Andrew J Davison. “Sim-to-real reinforcement learning for deformable object manipulation”. In: *Conference on Robot Learning (CoRL)* (2018).

[83] Daniel Mellinger, Nathan Michael, and Vijay Kumar. “Trajectory generation and control for precise aggressive maneuvers with quadrotors”. In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[84] A Milan et al. “Semantic segmentation from limited training data”. In: *arXiv preprint arXiv:1709.07665* (2017).

[85] Stephen Miller et al. “A geometric approach to robotic laundry folding”. In: *The International Journal of Robotics Research* 31.2 (2012), pp. 249–267.

[86] Stephen Miller et al. “Parametrized shape models for clothing”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4861–4868.

[87] Joseph Moore, Rick Cory, and Russ Tedrake. “Robust post-stall perching with a simple fixed-wing glider using LQR-Trees”. In: *Bioinspiration & biomimetics* 9.2 (2014), p. 025013.

[88] Douglas Morrison et al. “Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7757–7764.

[89] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach". In: *arXiv preprint arXiv:1804.05172* (2018).

[90] Andriy Myronenko and Xubo Song. "Point set registration: Coherent point drift". In: *IEEE transactions on pattern analysis and machine intelligence* 32.12 (2010), pp. 2262–2275.

[91] R. Holladay N. Chavan-Dafle and A. Rodriguez. "Planar In-Hand Manipulation via Motion Cones". In: *IJRR* (2019).

[92] Anusha Nagabandi et al. "Deep Dynamics Models for Learning Dexterous Manipulation". In: *arXiv preprint arXiv:1909.11652* (2019).

[93] Ashvin Nair et al. "Combining self-supervised learning and imitation for vision-based rope manipulation". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2146–2153.

[94] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. "DynamicFusion: Reconstruction and Tracking of Non-Rigid Scenes in Real-Time". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 343–352.

[95] Richard A Newcombe et al. "Kinectfusion: Real-time dense surface mapping and tracking." In: *ISMAR*. Vol. 11. 2011. IEEE. 2011, pp. 127–136.

[96] Van-Duc Nguyen. "Constructing force-closure grasps". In: *The International Journal of Robotics Research* 7.3 (1988), pp. 3–16.

[97] Takayuki Osa et al. "An algorithmic perspective on imitation learning". In: *Foundations and Trends® in Robotics* 7.1-2 (2018), pp. 1–179.

[98] Deepak Pathak et al. "Zero-shot visual imitation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2050–2053.

[99] Xue Bin Peng et al. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 1–8.

[100] Sudeep Pillai, Rareş Ambruş, and Adrien Gaidon. "Superdepth: Self-supervised, super-resolved monocular depth estimation". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9250–9256.

[101] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3406–3413.

[102] Dean A Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems*. 1989, pp. 305–313.

[103] Rouhollah Rahmatizadeh et al. "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 3758–3765.

[104] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[105] Colin Rennie et al. "A Dataset for Improved RGBD-based Object Detection and Pose Estimation for Warehouse Pick-and-Place". In: *CoRR* abs/1509.01277 (2015). URL: http://arxiv.org/abs/1509.01277.

[106] Stephan R Richter et al. "Playing for data: Ground truth from computer games". In: *European Conference on Computer Vision*. Springer. 2016, pp. 102–118.

[107] Diego Rodriguez et al. "Transferring grasping skills to novel instances by latent space non-rigid registration". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[108] German Ros et al. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: 2016.

[109] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[110] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[111] Caner Sahin and Tae-Kyun Kim. "Category-level 6D Object Pose Recovery in Depth Images". In: *arXiv preprint arXiv:1808.00255* (2018).

[112] Tanner Schmidt, Richard Newcombe, and Dieter Fox. "Self-supervised visual descriptor learning for dense correspondence". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 420–427.

[113] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[114] Max Schwarz et al. "Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 3347–3354.

[115] Daniel Seita et al. "Robot bed-making: Deep transfer learning using depth sensing of deformable fabric". In: *arXiv preprint arXiv:1809.09810* (2018).

[116] *Sense for RealSense*. URL: https://www.3dsystems.com/shop/realsense/sense.

[117] Pierre Sermanet, Kelvin Xu, and Sergey Levine. "Unsupervised perceptual rewards for imitation learning". In: *Robotics: Science and Systems (RSS)* (2017).

[118] Pierre Sermanet et al. "Time-Contrastive Networks: Self-Supervised Learning from Video". In: *arXiv preprint arXiv:1704.06888* (2017).

[119] Pierre Sermanet et al. "Time-contrastive networks: Self-supervised learning from video". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1134–1141.

[120] Jamie Shotton et al. "Scene coordinate regression forests for camera relocalization in RGB-D images". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 2930–2937.

[121] Maximilian Sieb et al. "Graph-Structured Visual Imitation". In: *arXiv preprint arXiv:1907.05518* (2019).

[122] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[123] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[124] Avi Singh, Larry Yang, and Sergey Levine. "GPLAC: Generalizing Vision-Based Robotic Skills using Weakly Labeled Images". In: *ICCV*. 2017.

[125] *SIXD Challenge*. URL: `http://cmp.felk.cvut.cz/sixd/challenge_2017/`.

[126] *Skanect*. URL: `http://skanect.occipital.com/`.

[127] *Skydio*. URL: `https://www.skydio.com/`.

[128] HJ Suh and Russ Tedrake. "The Surprising Effectiveness of Linear Models for Visual Foresight in Object Pile Manipulation". In: *arXiv preprint arXiv:2002.09093* (2020).

[129] Xiao Sun et al. "Integral human pose regression". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 529–545.

[130] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[131] Jonathan Taylor et al. "The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 103–110.

[132] Russ Tedrake and the Drake Development Team. *Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems*. 2016. URL: `https://drake.mit.edu`.

[133] James Thewlis, Hakan Bilen, and Andrea Vedaldi. "Unsupervised learning of object landmarks by factorized spatial embeddings". In: *Proc. ICCV*. Vol. 1. 2. 2017, p. 5.

[134] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 23–30.

[135] Jonathan Tremblay et al. "Deep object pose estimation for semantic robotic grasping of household objects". In: *Conference on Robot Learning (CoRL)* (2018).

[136] Jur Van Den Berg et al. "Gravity-based robotic cloth folding". In: *Algorithmic Foundations of Robotics IX*. Springer, 2010, pp. 409–424.

[137] Herke Van Hoof et al. "Stable reinforcement learning with autoencoders for tactile and visual data". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 3928–3934.

[138] Mel Vecerik et al. "A practical approach to insertion with variable socket position using deep reinforcement learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 754–760.

[139] *Vicon Motion Capture System*. https://www.vicon.com/.

[140] He Wang et al. "Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation". In: *arXiv preprint arXiv:1901.02970* (2019).

[141] Manuel Watter et al. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in neural information processing systems*. 2015, pp. 2746–2754.

[142] *Waymo Driver*. URL: `https://waymo.com/`.

[143] Thomas Whelan et al. "ElasticFusion: Dense SLAM without a pose graph". In: Robotics: Science and Systems. 2015.

[144] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. "Model predictive path integral control using covariance variable importance sampling". In: *arXiv preprint arXiv:1509.01149* (2015).

[145] J. M. Wong et al. "SegICP: Integrated Deep Semantic Segmentation and Pose Estimation". In: *ArXiv e-prints* (Mar. 2017). arXiv: `1703.01661 [cs.RO]`.

[146] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. "Beyond pascal: A benchmark for 3d object detection in the wild". In: *IEEE Winter Conference on Applications of Computer Vision*. IEEE. 2014, pp. 75–82.

[147] Ali Yahya et al. "Collective robot reinforcement learning with distributed asynchronous guided policy search". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 79–86.

[148] Wilson Yan et al. "Learning Predictive Representations for Deformable Objects Using Contrastive Estimation". In: *arXiv preprint arXiv:2003.05436* (2020).

[149] Jiaolong Yang et al. "Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 38.11 (Nov. 2016), pp. 2241–2254. ISSN: 0162-8828. DOI: `10.1109/TPAMI.2015.2513405`. URL: `https://doi.org/10.1109/TPAMI.2015.2513405`.

[150] Pin-Chu Yang et al. "Repeatable folding task by humanoid robot worker using deep learning". In: *IEEE Robotics and Automation Letters* 2.2 (2016), pp. 397–403.

[151] Yufei Ye et al. "Object-centric Forward Modeling for Model Predictive Control". In: *arXiv preprint arXiv:1910.03568* (2019).

[152] Lin Yen-Chen, Maria Bauza, and Phillip Isola. "Experience-Embedded Visual Foresight". In: *arXiv preprint arXiv:1911.05071* (2019).

[153] Jincheng Yu et al. "A vision-based robotic grasping system using deep learning for 3D object recognition and pose estimation". In: *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1175–1180.

[154] Tianhe Yu et al. "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning". In: *arXiv preprint arXiv:1802.01557* (2018).

[155] Tianhe Yu et al. "Unsupervised Visuomotor Control through Distributional Planning Networks". In: *arXiv preprint arXiv:1902.05542* (2019).

[156] Andy Zeng et al. "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching". In: *arXiv preprint arXiv:1710.01330* (2017).

[157] Andy Zeng et al. "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[158] Andy Zeng et al. "3dmatch: Learning local geometric descriptors from rgb-d reconstructions". In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 199–208.

[159] Andy Zeng et al. "Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning". In: *arXiv preprint arXiv:1803.09956* (2018).

[160] Andy Zeng et al. "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1386–1383.

[161] Andy Zeng et al. "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics". In: *arXiv preprint arXiv:1903.11239* (2019).

[162] Tianhao Zhang et al. "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[163] Jiaji Zhou, Yifan Hou, and Matthew T Mason. "Pushing revisited: Differential flatness, trajectory planning, and stabilization". In: *The International Journal of Robotics Research* 38.12-13 (2019), pp. 1477–1489.

[164] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Fast Global Registration". In: *European Conference on Computer Vision*. Springer. 2016, pp. 766–782.

[165] Yuke Zhu et al. "Reinforcement and imitation learning for diverse visuomotor skills". In: *arXiv preprint arXiv:1802.09564* (2018).