Representing Unstructured Environments for Robotic Manipulation: Toward Generalization, Dexterity and Robustness

by

Wei Gao

B.S., Tsinghua University (2017)
S.M., Massachusetts Institute of Technology (2020)

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

Doctor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author . . . . . .
Department of Electrical Engineering and Computer Science
August 27, 2021

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

Representing Unstructured Environments for Robotic Manipulation: Toward Generalization, Dexterity and Robustness

by

Wei Gao

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Science

Abstract

We would like to have highly useful robot manipulators that can handle a diversity of objects/environments, perform challenging manipulation tasks while being sufficiently robust such that deployment at scale is feasible. This thesis aims at such a generalizable, dexterous and robust manipulation pipeline. At the core of our approach is the representation of the environment. In particular, how should we represent the unstructured world such that it is useful for: 1) developing a capable manipulation pipeline; 2) performing a thorough robustness evaluation of it. To answer question 1), we propose the keypoint affordance, a novel object representation consists of 3D semantic keypoints. Existing works typically use 6 Degree-of-Freedom (DOF) poses to represent the manipulated objects. However, representing an object with a parameterized transformation defined on a fixed template cannot handle large shape mismatches among different objects. In contrast, our keypoint representation captures task-related geometric information while ignoring irrelevant details, which enables the generalization to unknown objects. We implement perception, planning and feedback control modules on top of the keypoint representation and integrate them into a fully functional perception-to-action manipulation pipeline. The second part of this thesis studies the pipeline robustness and attempts to answer the question 2). Due to the infeasibility of a parametric (pose-based) object representation, we do not have a continuous input domain for investigating how the object geometry impacts the robustness, which is a prerequisite for existing methods. To address this challenge, we model factors that affect the robustness as a structured distribution over variables (e.g. the camera pose), combined with an empirical distribution, that describes visual properties (e.g. the object geometry/texture). We then formulate the robustness evaluation as a failure rate estimation problem on this combined distribution and propose an efficient graph-based algorithm to solve it. Our formulation is applied to the developed manipulation pipeline, and it can benefit many other cyber-physical systems, such as autonomous cars.

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

Acknowledgments

There are many people that I want to thank for helping me to complete this work over the past years.

Firstly I would like to thank my advisor, Prof. Russ Tedrake. Russ directs me to think of the challenge caused by the diverse, unstructured robot working environment, which turns into this thesis. He provides me a lot of help in both academy and life, especially for the last year under the challenging situation of the pandemic. In addition, Russ deserves enormous credit for bringing together the people and integrated countless resources into the lab. Everyone lab member, including me, benefits a lot from that.

I would also like to thank the rest of my thesis committee members, Prof. Phillip Isola and Prof. Chuchu Fan, for their time, guidance and support throughout my thesis work. The collaboration is a great pleasure and the suggestions they provide significantly improve this thesis.

I have had a world-class set of labmates in the Robot Locomotion Group, who grant me such a unique and inspiring place to work. In particular, I am introduced to robotics by Twan Koolen's contribution on humanoid robot. The discussion with him at MIT is very fruitful. A special thanks to Pete Florence, Lucas Manuelli and Greg Izatt for their effort of setting up the robot, writing the drivers and maintaining the spartan software stack. My pursue of the MCMC-based risk evaluation is inspired by the contributions of Matthew O'Kelly and Aman Sinha. They offer a lot of help during my own development of these algorithms. I also want to thank Yunzhu, Tobia, Pang and many others for their helpful suggestions. Their expertise leads me to the solution of many seemingly intractable problems. And finally thanks to all the other members of the group for constantly stimulating and exciting discussions.

I have also learned a lot from many colleagues, friends and professors at the broader MIT community. I've taken/audited many interesting courses, and the collaboration with other MIT students is really a pleasure. For example, Prof. Guy Bresler's class on graphical model rebuilt my understanding of probabilistic inference and lead to an interesting contribution on point-set registration.

Finally I would like to thank my family. Thanks to my Mom and Dad for supporting me always.

Contents

Chapter 1

Introduction

In recent years, significant advancements in computer vision [23, 50, 69, 127] have brought us the hope of deploying robot manipulators "in the wild". Industrial robot manipulators have been helping us in assembly lines since 1970, but they are restricted to hand-crafted trajectories, strict separation from humans, and carefully controlled working environments. With the availability of deep neural networks, perception-to-action robot manipulation pipelines [27, 56, 61, 100] are much more reliable. We hope that in the future, robots can help us with more demanding tasks, such as cleaning up a kitchen or taking care of the elderly.

Despite these advancements, state-of-art robot manipulation pipelines today are still highly inferior to humans and cannot meet our expectations. First of all, the capability of current robot manipulators is still limited to small sets of known objects, relatively simple manipulation skills, or both. We have seen amazing demos such as OpenAI's finger gaiting [4] and BostonDynamics' kitchen manipulation [27]. However, it is unclear whether these robots can work with some new objects in another environment. Another series of works on robot grasping [45, 86, 96, 132, 151] show surprising generalization in the Amazon Picking Challenge. However, these works are restricted to picking up the object. Extending them to more dexterous tasks is not straightforward.

In addition to the capability, the robustness of manipulation pipelines is another concern, primarily due to the internal deep neural networks used for visual perception. The performance of these neural networks can be catastrophically deteriorated by adversarial

attacks [42, 72] and distributional mismatches [5, 6], which consequently leads to the failure of the entire pipeline. Thus, it is crucial to understand the robustness of a manipulation system before deployment. However compared to replaying hand-crafted trajectories in an assembly line, a perception-to-action manipulation pipeline in the wild risks from more complex factors, such as the textures of objects and backgrounds. How to characterize these complex factors and their impacts on the pipeline robustness is still an open problems.

In summary, existing pipelines are still inferior to the ideal, human-level robot manipulation system in terms of their generalization, dexterity and robustness. In this thesis, we take a step forward with respect to these perspectives. To achieve this, we need to address the challenge caused by the diverse, unstructured robot working environment, as detailed below.

1.1 The Challenge of Representation

Compared to many other types of robots such as humanoids/UAVs, perception-to-action robot manipulation in the wild has the distinct challenge of environment representation. Let us consider this challenge using the example task of robot kitchen cleanup, as shown in Fig. 1-1.

We would like a robot manipulator to come into a kitchen and clean it up. Using a classical planning and control formulation, we might consider this task as the following optimal control problem

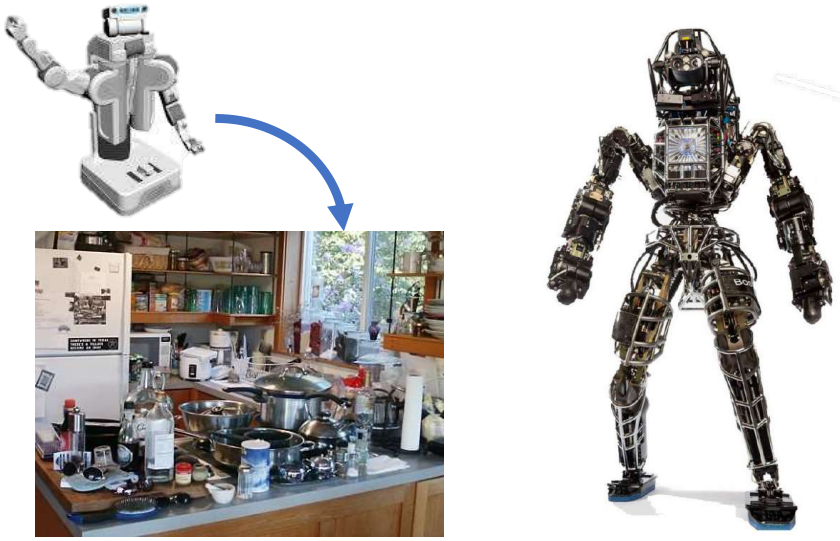
$$\min_{\pi} \ \Sigma_t c(x_t, u_t) \tag{1.1}$$

$$\text{subject to:} \tag{1.2}$$

$$x_{t+1} = f(x_t, u_t) \tag{1.3}$$

$$u_t = \pi(x_t) \tag{1.4}$$

where we would like to find a policy $\pi : X \to U$ that minimizes the sum of the cost function $c : X \times U \to R$, subject to the dynamic constraint $f : X \times U \to X$. However, one major chal-



(a) (b)

Figure 1-1: Robot manipulation in the wild such as the kitchen cleanup task in (a) has the distinct challenge of environment representation, unlike other types of robots such as the humanoid in (b). For the kitchen clean-up task, it is unclear how to define the state $x = (x_{\text{robot}}, x_{\text{world}})$ such that the environment representation x_{world} can handle kitchens with potentially unseen objects. In contrast, in classical robotic tasks such as the humanoid walking we usually have a natural state representation (as the generalized position and velocity). The lack of this state representation makes it hard to both develop a capable manipulation pipeline and understand its robustness, as described in Sec. 1.1.

lenge with this formulation is that we do not know how to define the state $x = (x_{\text{robot}}, x_{\text{world}})$. Although the robot state x_{robot} can be easily defined as a joint state (generalized position and velocity), it is unclear how should we write down the kitchen state as x_{world}, given the diversity of objects in it. In the terminology of the machine learning community, we do not know how to define the *representation* of the kitchen. Without a good state representation, we cannot write down the cost function c and dynamic function f. Consequently, many existing techniques that work well for other types of robots cannot be used for the "robot cleaning up a kitchen" problem.

As a concrete example, let us consider an extremely simplified scenario where the kitchen only contains one known object. Then, we can use the 6-DOF pose of that object as the state representation x_{world} for the kitchen, and define the cost function c and dynamic function f on top of that state. Pose-based representation has been extensively used in existing robot manipulation pipelines [56, 100, 115, 142]. However, these contri-

butions are limited to manipulating a fixed object. As detailed in Sec. 3.4, representing an object with a parameterized pose defined on a fixed geometric template cannot capture large shape variations among different objects. Thus, this pose-based representation is not suitable if the kitchen contains a diverse set of objects with potentially unknown instances.

This challenge of representation also prevails when we are considering the pipeline robustness. Intuitively, our goal is to ensure that "the manipulator is safe for *all possible kitchens*" or "the robot is very unlikely to fail in *a usual kitchen*". These descriptions are very straightforward in natural language. However, it is not easy to transform them into concrete, solvable robustness evaluation problems because we don't know how to materialize "*all possible kitchens*" or "*a usual kitchen*". Without a state representation x_{world} for the kitchen, it is hard to materialize "*all possible kitchens*" as the state space of x_{world} or materialize "*a usual kitchen*" as a prior distribution on that state space. As a result, we cannot perform a robustness evaluation for the manipulation system that we are interested in.

Let us again consider the extremely simplified example in which only one object is contained in the kitchen. For this situation, x_{world} can be the pose of that object and "all possible kitchens" is the space of the rigid transformation ($SE(3)$). Consequently, "a usual kitchen" becomes a prior distribution on the rigid transformation space, for example a uniform distribution with bounds. In this case, we have a concrete robustness evaluation formulation and many existing algorithms [11, 14, 46, 129] can be used to solve it. However, this formulation becomes invalid if we consider a diverse set of objects with different geometries (instead of a fixed instance). Thus, it is difficult to apply existing techniques to our problem due to the lack of a good state representation x_{world}.

1.2 Problem Statement and Contributions

In this thesis, we address the representation challenge described above and take steps toward an ideal manipulator that combines generalization, dexterity and robustness. In particular, we attempt to answer how to represent an environment such that it is useful for 1) developing a capable manipulation pipeline; and 2) performing a robustness evaluation for

the pipeline with respect to complex input domains (such as "all possible kitchens").

The first part of this thesis focuses on building a capable manipulation pipeline that combines *generalization* and *dexterity*. In Chapter 3, we propose a novel keypoint-based object (state) representation for generalizable, pick-and-place manipulation. In other words, we propose to use a set of 3D semantic keypoints as x_{world}. Existing methods typically use the 6-DOF pose as the underlying object representation, which can lead to infeasible placement configurations for new objects. In contrast, the proposed keypoint representation provides a unified way to specify the desired object configurations for many objects, despite the significant shape variations among them. Hardware experiments show that our approach generalizes to novel objects for pick-and-place manipulation, and this generalization is accurate enough to accomplish tasks requiring centimeter-level precision.

Chapter 4 extends the pick-and-place pipeline in Chapter 3 with physical constraints such as the collision avoidance, visibility and grasp stability. The sparse keypoint representation in Chapter 3 is not sufficient, as reasoning about physical properties requires dense geometric information. Thus, we propose a hybrid object representation consisting of sparse keypoints and dense geometry, where the dense geometry can be obtained using shape completion algorithms [92,104,155]. The integration of shape completion algorithms enables many existing planners to handle a variety of objects in a unified and precise way. Several hardware experiments demonstrate the efficacy of this integration.

Chapter 5 extends the pipeline from kinematic pick-and-place to closed-loop, contact-rich manipulation tasks. To achieve this, we first augment keypoints with local orientation information. Using oriented keypoints as the object representation, we propose a novel object-centric robot action representation in terms of regulating the linear/angular velocity or force/torque of these oriented keypoints. This formulation is surprisingly versatile – we demonstrate that it can accomplish contact-rich manipulation tasks that require precision and dexterity for many objects with different shapes, sizes and appearances. We demonstrate our method with challenging tasks such as peg-hole insertion for pegs and holes with significant shape variations and tight clearances.

The second part of this thesis focuses on the *robustness* of the manipulator, which is an autonomous system deployed in diverse, unstructured working environments. Many factors

of the environment impact the robustness. Some of them have natural representations (parameterizations), such as camera poses and illumination conditions. However, some other factors might be hard to represent, such as the object geometry discussed in Sec. 1.1. As a result, many existing works [11, 14, 46, 64, 129, 150] are not suitable for our problem, as they assume a continuous, easily-parameterizable input domain.

Initially, we plan to use the keypoints in Chapter 3 as the object geometric representation for the robustness evaluation. To be more specific, can we represent the object as a set of keypoints, and perform a robustness evaluation with the input space of these keypoints? However this approach does not work well, because the keypoint representation loses geometric information, and this missing geometric information might impact the robustness. This inspires us to use a complete geometric representation, similar to Chapter 4. In particular, we can over-parameterize each object geometry instance as a voxel grid, which generalizes well to different objects. In this case, the challenge becomes how to represent the underlying *space* or *distribution* of the object geometry. The space of "all possible mugs" is obviously a large space, but it is much smaller than the voxel space, as realistic object shapes occupy only a tiny portion of the voxel space. If we draw a random sample from the voxel space, it is very unlikely that this sample would look like a realistic object, and we should not expect the robot to handle it. Another prominent example is the textures of the object, that we might want to ensure a manipulator can handle "all textures of a mug". We can over-parameterize each object texture instance as an image (texture map), but the space of "all object textures" is again much smaller than the pixel space.

One method to address these texture/shape distributions is to use generative models [41]. These generative models learn generators that map simple distributions in the feature space to complex texture/shape distributions. With this generator, we can use existing algorithms in the continuous feature space. However, these generative models can yield unrealistic samples (see [84] for a detailed study), although many of the generated results are almost indistinguishable from authentic ones. As a result, we can discover unrealistic failure cases that would never be encountered in the real world, and this phenomenon can be exploited by the robustness evaluation algorithms.

In Part II of this thesis, we propose to directly materialize these texture/shape distribu-

(a) (b)

Figure 1-2: In Part. II of this thesis, we consider the robustness of the robot manipulator. Some environmental factors that impact the pipeline robustness might be hard to represent, such as the object geometry in (a) and texture in (b). It is hard to write down a continuous parameterization of them. As a result, we cannot perform the robustness evaluation with respect to the object geometry/texture in a continuous parameter space, which is a prerequisite of existing works [11, 14, 46, 64, 129, 150].

tions as empirical distributions (sets of offline-collected samples). Thus, we model factors that affect the system robustness as a structured distribution over variables (e.g. the object pose), combined with an empirical distribution, that describe the visual properties. We then formulate the robustness evaluation as failure search and failure rate estimation problems on this combined distribution. Compared to the generative model formulation, our method does not produce arbitrarily unrealistic failure examples. One major challenge of this representation is the lack of continuity structures among the discrete samples. To address this issue, we formulate a weighted graph over the empirical dataset using the distance in a learned latent space as the edge weights. This graph structure connects discrete samples and transforms the failure search/rare-event problems into more efficient graph-based exploration. Moreover, failure rate estimation with the proposed graph converges to the ground-truth asymptotically, despite the using of learned features in the graph representation.

In Chapter 6, we present the preliminaries about the risk-based framework. In this framework, we prioritize finding the most likely failure modes and characterizing a system's safety by its failure probability. Existing works in this risk-based framework assume continuous, easily parameterizable input domains. However as mentioned above, we need

to perform robustness evaluation with respect to complex input domains such as object textures/shapes. This is addressed in Chapter 7. Then, in Chapter 8 we apply the resulting robustness evaluation algorithm to the robot manipulation pipeline developed in Part. I. It should be emphasized that our formulation can be applied to many other cyber-physical systems, although we focus on robot manipulation in this thesis.

Chapter 2

Related Work

There is a broad array of works that are related to this thesis. Rather than listing repeated sections of related work, in this chapter we briefly comment on the overall body of work in the literature that relates to this thesis, and highlight where additional detail can be found within each subsequent chapter. We first review the existing motion planning and control techniques for robot manipulators. These methods are the basis of Part I, and they have been applied to industrial manipulators since 1970. Then, in Sec. 2.2 we review some contributions regarding perception-to-action robot manipulation pipelines, as the pipeline we propose in Part I falls into this category. These contributions are primarily driven by the success of deep neural networks. Although these networks demonstrate exceptional empirical performance, there are reliability concerns associated with using deep networks in cyber-physical systems such as manipulation pipelines, which is discussed in Sec. 2.3. In Part II of this thesis, we develop robustness evaluation algorithms to characterize and improve the pipeline reliability.

2.1 Manipulator Planning and Motion Control

There are millions of industrial robots on assembly lines today [28]. They perform tasks such as polishing, workpiece handling, painting, and welding. Despite the diversity of these application fields, the only operation of these industrial robots is to replay a pre-specified trajectory again and again. To facilitate this, the robot needs trajectory planning and motion

control algorithms. In the following text, we briefly review some of the contributions in this area, which are the basis of Part I.

Manipulation Planning: The task of manipulator planning is to automatically generate robot trajectories [74], thus reducing or eliminating the manual effort required for lead-through demonstrations. The planner takes the trajectory specification, such as the desired end-effector pose, as the input. The planned trajectory should also satisfy physical feasibility constraints, and the most prominent example is perhaps the collision avoidance [39,113,117]. Motion planners can be classified into two major categories: sampling-based planners and optimization-based algorithms.

For sampling-based planners, the motion planning problem is typically formulated as a search problem on a graph embedded in the search space. This graph can be explicitly constructed before planning, by methods such as the probabilistic road map (PRM) algorithm [40, 54]. In the PRM algorithm, we first sample a set of joint configurations that are collision-free. Then, these joint configurations are connected by edges, and collision checking is performed to eliminate infeasible edges. Finally, a graph search is performed to find a feasible trajectory. The graph can also be constructed incrementally to reduce the imposed computation and storage requirements [35,63,70,99]. There are many variants of these algorithms; please refer to [74,76] for a comprehensive study.

As suggested by the name, optimization-based planers formulate the planning problem as a constrained optimization [20,21,94,105,117,133]. A set of costs/constraints are used to encode the task specification and physical feasibility constraints. For example, we can constrain the minimum distance between the robot and the environment to avoid collisions. These optimization problems are typically non-linear and non-convex, and they are usually solved using the interior-point method or sequential convex programming [9] algorithms. Several high-quality implementations of these solvers and optimization-based planners are available [73,117,126,131], and we use the drake library [131] for our implementation in Chapter 4.

Robot Motion Control: The task of motion control algorithms is to track the robot trajectory. To enable accurate tracking, researchers have investigated friction compensation [8,13], mass/inertia identification [66,102], and online adaptive control algorithms [53,

123]. These methods are usually integrated into robot drivers because of their importance, and please refer to for [123] a detailed review. The tracking of joint-space trajectories is relatively easy since the actuation force is in the joint space. The end-effector space trajectories should be transformed into joint space actions using robot kinematics [65]. Some controllers try to achieve compliance behaviors in the joint space or end-effector space [22, 65, 112, 141]. For example, a peg-hole insertion controller should be stiff in the insertion direction (to transmit force) and compliant in tangential directions (to avoid jamming) [79, 103, 130]. Our implementation of the peg-hole insertion in Chapter 5 is based on the concept of the end-effector compliance control [130], which is detailed in Sec. 5.5.

These planning and control algorithms form the basis of this thesis. However, they are not sufficient for the task that we are interested in, such as the kitchen cleanup task in Chapter 1. The major goal of this thesis is to enable the robot to handle diverse, unstructured working environments. The robot must automatically handle different objects and task setups with visual perception, instead of replaying a pre-specified trajectory.

2.2 (Deep) Perception for Robot Manipulation

The industrial robots described in Sec. 2.1 cannot adapt to alignment errors and/or different initial object configurations. As a result, customized fixturing/gripper designs are required. For example, wafer-handling robots require wafer carriers to be localized using a three-groove kinematic coupling; thus, the initial configuration is accurate. As customized fixturing/gripper hardware can be expensive, people resort to an alternative strategy of localizing objects using visual perception. The advancement of deep learning has significantly improved the performance of visual perception algorithms, which has a huge impact on robot manipulation. In the following text, we discuss how the deep networks enable many interesting manipulation behaviors.

2.2.1 Pose Estimation for Robot Manipulation

The configuration of a known object can be defined as a rigid transformation (pose) from a geometric template model. If we know the pose of the object, the manipulator can han-

dle different initial object configurations. Pose estimation is an extensively studied topic in computer vision and robotics, and existing methods can be generally classified into geometric-based algorithms [37, 97] and learning-based approaches [115, 134, 142]. These methods take inputs as raw observations (RGBD images or point clouds) and produce the pose of the object of interest in the scene. Several contributions [56, 100] integrate pose estimators into robot manipulation pipelines to accomplish interesting pick-and-place tasks. However, as detailed in Sec. 3.4, pose estimation cannot handle the shape variations of different objects. Thus, it is not suitable for manipulation tasks such as cleaning up a kitchen. This challenge of adapting to different objects is addressed in Chapter 3.

2.2.2 Robot Grasp Planning

In recent years, there has been an explosion of literature tackling the problem of robotic grasping [45, 86, 87, 151, 152]. Given a scene observation (usually a RGBD image), these grasping algorithms compute a robot end-effector action that can pick up an object. Among the various grasping approaches, model-based methods [87, 152] typically rely on a prebuilt 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 [45, 86, 96, 151] directly evaluate the grasp quality from raw sensor inputs. Many of these approaches achieve promising robustness and generality in the Amazon Picking Challenge [120, 151, 152]. Several works also incorporate object semantic information using instance masks [120], or non-rigid registrations [114] to accomplish tasks such as picking up a specific object or transferring a grasp pose to novel instances. However, grasping algorithms cannot accomplish any manipulation tasks beyond grasping an object and then dropping it somewhere else, which we affectionately label the “pick-and-drop" task. Thus, we use the grasping algorithm more as a building block in a larger manipulation system, rather than an end in itself.

2.2.3 End-to-End Learning for Robot Manipulation

Impressive contributions [3,31,44,77,139,156] have been made in end-to-end learning with applications to robotic manipulation. These methods usually learn a visuomotor policy that maps raw observations to robot actions. Thus, they avoid the use of explicit state representations (or, they use the “image state"), which is challenging as discussed in Chapter 1. The policy is typically trained with imitation learning [32, 111, 148, 154, 156] or reinforcement learning [31, 118, 119] algorithms. In imitation learning, the algorithm requires training data in the form of state action pairs from expert demonstrations. In reinforcement learning, an efficient simulator is used for trial-and-error attempts. The policy can be trained in a purely end-to-end manner [31, 43, 111, 148, 156] or as a residual term [57, 116, 121] added to an existing policy. Some works exploit autoencoders [31,77,139,148] and domain randomization [4,15] to improve sampling efficiency. There are many variants of these two types of learning algorithms; please refer to [128] for a detailed study.

Many interesting manipulation behaviors emerge from these data-driven algorithms [3, 31,77]. However, how to efficiently generalize the trained policy to different objects, camera positions, initial object configurations and/or robot grasp poses remains an active research problem. Theoretically, this generalization can be achieved by utilizing training data with good coverage in terms of all these perspectives. However, this can be extremely expensive in practice.

2.3 The Reliability of Deep Networks

As mentioned in Sec. 2.2, deep neural networks enable many interesting manipulation behaviors. However, the performance of these neural networks can be catastrophically deteriorated by adversarial attacks [42, 72] and distributional mismatches [5, 6], which consequently leads to the failure of the entire pipeline. Thus, it is crucial to understand the robustness of neural networks and autonomous systems with networks inside.

Most works on neural network robustness analysis aim at image input and pixel-wise L_p constrained disturbance [11,14,46,129]. The practical usefulness of pixel-wise disturbance has been questioned in recent works [26, 29]. Thus, researchers have targeted more prac-

tically meaningful input domains, for example the "adversarial patch/sticker" in [12] and the object pose/illumination condition in [80]. Some works [26, 136] have also studied the robustness of systems with data-driven components inside instead of the singulated neural networks. From the perspective of methodology, robustness analyses have been formulated as optimization [80, 129], search (planning) [68, 135], and reinforcement learning [67, 108] problems. Depending on the availability of internal structures, these methods can be classified into white-box and black-box approaches; please refer to [1, 7, 18] for a more detailed review.

Existing works on robustness evaluation are typically restricted to continuous, easily parameterizable input domains. However, for neural networks and autonomous systems deployed in unstructured environments, many factors with significant impact on robustness might not have natural parameterizations. For example, we might consider how the object geometry affects the pipeline robustness, as detailed in Chapter 1. Part II of this thesis addresses this challenge and performs a robustness evaluation of the manipulation pipeline developed in Part. I.

Part I

Pipeline Development

Chapter 3

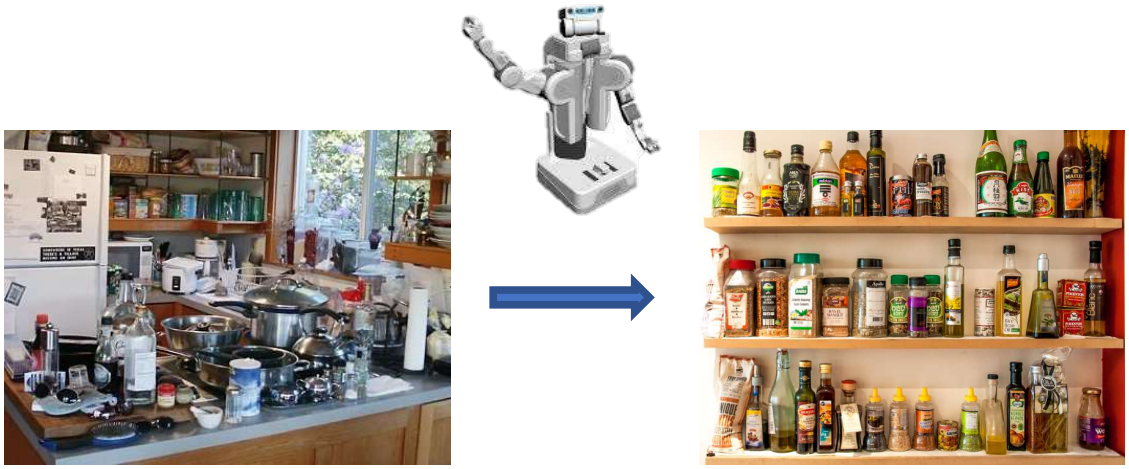
kPAM: KeyPoint Affordance for Generalizable Manipulation

3.1 Introduction

In this chapter, we consider manipulation tasks such as the kitchen cleanup in Chapter 1, under the assumption of *rigid* objects. An illustration is shown in in Fig. 3-1. If we assume the objects are rigid, then cleaning up a kitchen becomes a series of kinematic pick-and-place manipulation: the robot manipulator should inspect the environment, detect the object, pick it up and place it to the desired configuration. This type of tasks can be easily described using natural language, for example "put the mugs upright on the mug shelf," "hang the mugs on the rack by their handles" or "place the shoes onto the shoe rack."

Although this type of tasks is trivial for human, it remains challenging for existing robot manipulation pipelines. In particular, we want the robot manipulator to handle many different objects, including previously unseen ones. For instance in the "put the mugs upright on the mug shelf" task, the robot should be able to manipulate many different mugs, despite significant geometry variations among them, as shown in Fig. 3-1. This manipulation skill is obviously within the capability of the human, and it is of significant importance to both industrial applications and interactive assistant robots.

Robot pick-and-place manipulation has been pursued for decades [56, 100, 134]. However, existing methods typically restrict to a small set of known objects with offline-captured



(a) (b)

Figure 3-1: In this chapter, we consider manipulation tasks such as kitchen cleanup under the assumption of *rigid* objects. Then, cleaning up a kitchen becomes a series of kinematic pick-and-place manipulation. Existing pick-and-place pipelines typically restrict to a small set of known objects with offline-captured geometric templates. In this work, we would like our robot to handle a variety of objects with significant shape variations and unknown instances. For example, we might want the robot to organize bottles in (a) into the cabinet in (b). The manipulation pipeline should handle many different bottles here.

high-quality geometric templates. On the other hand, a lot of works [45, 86, 96, 132, 151] address robotic grasping for arbitrary objects, and they have shown surprising generalization to different objects. However, existing methods have not demonstrated pick *and place* manipulation: most of these robot grasping pipelines would only drop the grasped object into a "target bin". This is not suitable for our tasks such as "put the mugs upright on the mug shelf".

In this chapter, we focus on pose-aware robotic pick and place at a category level. We want the robot to not only pick up many different mugs, but also place them onto a mug shelf. Since picking up the object has been addressed by many existing grasping algorithms [45, 86, 96, 132, 151], the challenge becomes how to perform object placement. In particular, we need to define the target configuration for many different objects, under the significant shape variation among them. For instance, we need to specify the target configuration for mugs in the "put the mugs upright on the mug shelf", which should be compatible with many different mug instances.

This is exactly the challenge discussed in Chapter 1.1. In particular, we need a cost

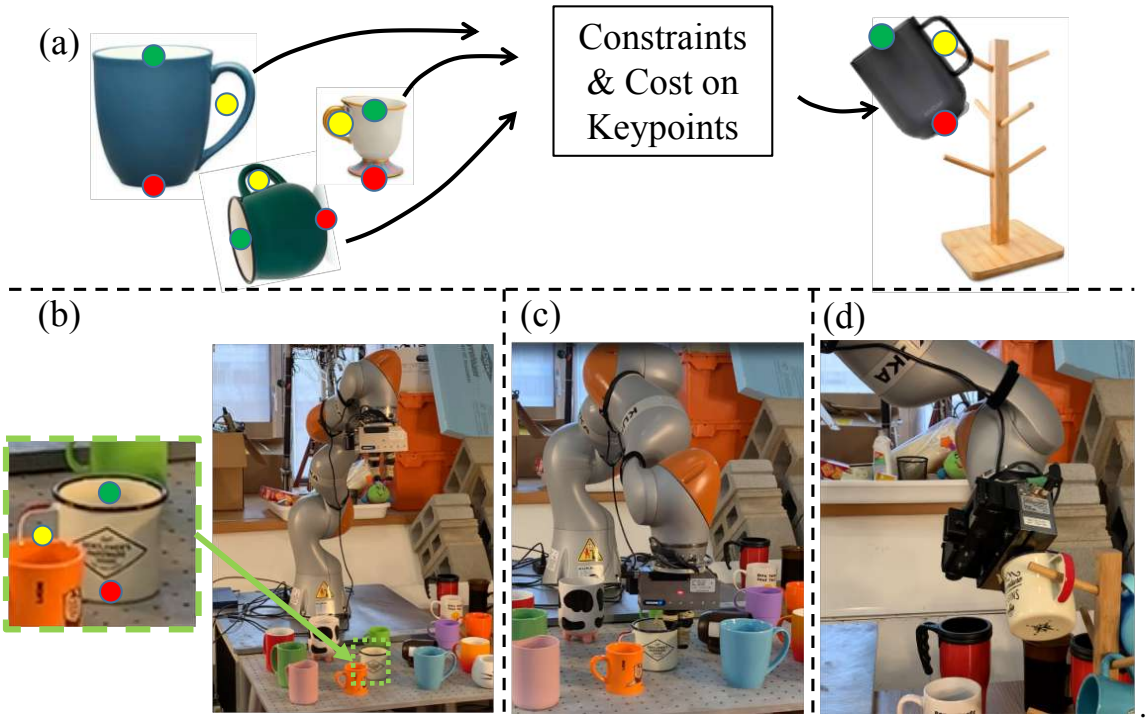


Figure 3-2: We propose kPAM, a framework for defining and accomplishing category level manipulation tasks. The key distinction of kPAM is the use of semantic 3D keypoints as the object representation (a), which enables flexible specification of manipulation targets as geometric costs/constraints on keypoints. 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 this link.

function c built upon the state representation x that encodes the desired configuration of the objects. Perhaps a straightforward solution would be using 6-DOF pose as the state x, and this state can be estimated using many existing pose estimation algorithms. With this pose-based state representation, we can define the desired object configuration as a target pose, and the cost c is the difference between the current pose and target pose of the object. However, as detailed in Sec 3.4, representing an object with a parameterized pose defined on a fixed geometric template may not adequately capture large intra-class shape or topology variations, and can lead to physical infeasibility for certain instances in the category.

In this chapter we propose kPAM[1]: a novel formulation of the category-level pick and

[1]kPAM stands for **K**ey**P**oint **A**ffordances **M**anipulation

place manipulation which uses semantic 3D keypoints as the object (state) representation. An illustrate is provided in Fig. 3-2. 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. We demonstrate our method on several hardware robot manipulation tasks. We show its generalization to a variety of objects and this generalization can achieve accuracy at the level of 1 centimeter.

This chapter is organized as follows. Sec. 3.2 discusses the related works. Sec. 3.3 describes our manipulation formulation. Sec. 3.3.1 introduces the formulation using a concrete example, while Sec. 3.3.2 describes the general formulation. Sec. 3.4 compares our formulation with pose-based pick and place pipelines to highlight the flexibility and generality of our method. Sec. 3.5 demonstrates our methods on several pose-aware manipulation tasks and shows generalization to novel instances. Sec. 3.6 concludes.

3.2 Related Works

3.2.1 6-DOF Pose Representations for Pick-and-Place Manipulation

The default solution to the pick and place of a known object is to estimate its 6 DOF pose. The robot then moves the object from its estimated pose to the target pose. Pose estimation is extensively studied in the computer vision community, as reviewed in Sec. 2.2.1. Several datasets [142, 147] are annotated with pre-aligned geometric templates, and pose estimators [115, 142] 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 varia-

tions, 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. 3.4.

3.2.2 Grasping Algorithms

Grasping algorithms enable finding stable grasp poses that allow robots to reliably pick up objects. As reviewed in Sec. 2.2.2, robot grasping planning is extensively studied and many contributions [45, 86, 96, 151] demonstrate surprising generalization in the Amazon Picking Challenge. However, in this chapter we would like the robot to achieve purposeful manipulation by moving the object within a category to some target configuration. This is a task that requires much more than just being able to find a grasp on the object, and is out of scope for the robot grasping algorithms.

3.2.3 End-to-End Reinforcement Learning

As reviewed in Sec. 2.2.3, there have been impressive contributions [3, 44] in end-to-end reinforcement learning with applications to robotic manipulation. In particular, [44] 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, 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 [44], 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. Our proposed object

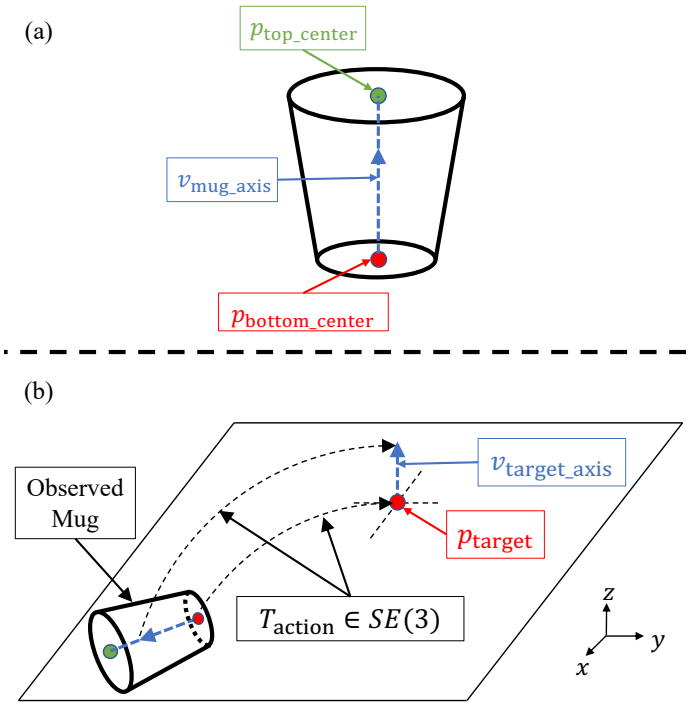


Figure 3-3: An overview of our manipulation formulation using the "put mugs upright on the table" task as an example: (a) we train a category level keypoint detector that produces two keypoints: $p_{\text{bottom\_center}}$ and $p_{\text{top\_center}}$. The axis of the mug $v_{\text{mug\_axis}}$ is a 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 $v_{\text{target\_axis}}$.

representation of 3D keypoints could be used as a solution to this problem.

3.3 Manipulation Formulation

In this section, we describe our formulation of the category level manipulation problem. Sec. 3.3.1 describes the approach using a concrete example and Sec. 3.3.2 presents the general formulation.

3.3.1 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. 3-3 (a). Additionally, we assume we have a keypoint detector, discussed in Section 3.3.2, that takes 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. 3-3 (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 $p \in R^{3\times N}$ 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 $T_{\text{action}}p \in R^{3\times N}$ using the robot action. In practice, this action T_{action} is implemented by first grasping the object using the algorithm detailed in Sec. 3.3.2 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: find a rigid transformation T_{action} such that

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

$$||T_{\text{action}}p_{\text{bottom\_center}} - p_{\text{target}}|| = 0 \tag{3.1}$$

2. The transformed direction from the mug bottom center to the top center should be aligned with the upright direction. This is encoded by adding a cost to the objective function

$$||1 - \langle v_{\text{target\_axis}},\ \text{rot}(T_{\text{action}})v_{\text{mug\_axis}}\rangle||^2 \tag{3.2}$$

where $\text{rot}(T)$ is the rotational component of the rigid transformation T, the target

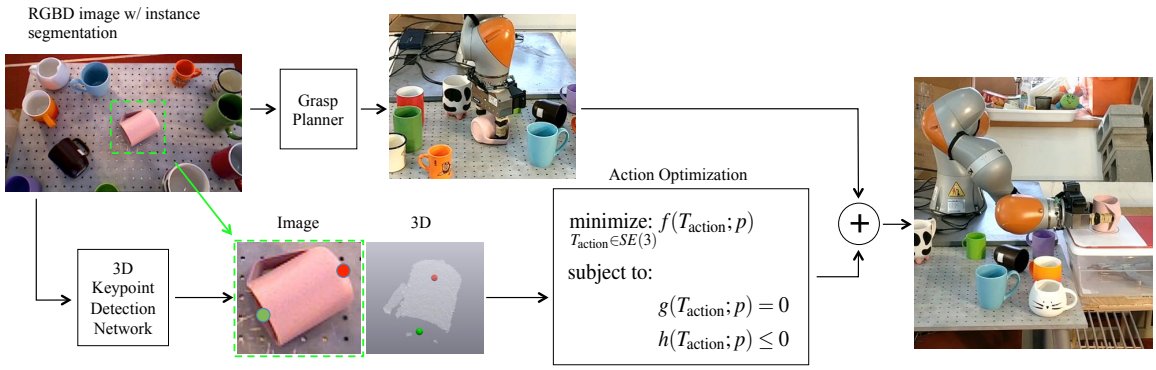


Figure 3-4: An overview of the category level pick and place pipeline using our manipulation formulation. Given a RGBD image with instance segmentation, the semantic 3D keypoints of the object in question are detected. We then feed these 3D keypoints into an optimization based planning algorithm to compute 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.

orientation $v_{\text{target\_axis}} = [0,0,1]^T$, and

$$v_{\text{mug\_axis}} = \frac{p_{\text{top\_center}} - p_{\text{bottom\_center}}}{||p_{\text{top\_center}} - p_{\text{bottom\_center}}||} \tag{3.3}$$

An illustration is presented in Fig. 3-3 (b). The above problem is an inverse kinematics problem with T_{action} as the decision variable, a constraint given by Eq. (3.1) and cost given by Eq. (3.2). This inverse kinematics problem can be reliably solved using off-the-shelf optimization solvers such as [131]. We then pick up the object using robotic grasping algorithms [45,87,96] and execute a robot trajectory which applies the manipulation action T_{action} to the grasped object.

3.3.2 General Formulation

For an arbitrary category level manipulation task we can represent an object using task-relevant semantic 3D keypoints. The task is then specified via geometric costs and constraints on these keypoints, which affords a flexible way of formulating the manipulation problem. The user selects keypoints, e.g. $p_{\text{top\_center}}$ and $p_{\text{bottom\_center}}$ in the example of Sec. 3.3.1, together with costs and constraints, e.g. (3.1) and (3.2), which fully specify the task. Once we have chosen this as the problem specification, there exist natural formula-

tions for each remaining piece of the manipulation pipeline. This allows 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. 3-4. In the following sections, we describe each component of our manipulation pipeline in detail.

Instance Segmentation and Keypoint Detection As discussed in Section 3.3.1 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 [127] 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 [127]. 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 [90]. 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. 3-8 (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 3.5.

The keypoint detection network [127] requires object instance segmentation as the input, and we integrate Mask R-CNN [49] into our manipulation pipeline to accomplish this step. The training data mentioned above for the keypoint detector [127] can also be used to

train the instance segmentation network [49]. 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

$$\begin{aligned} &\underset{T_{\text{action}} \in SE(3)}{\text{minimize:}} \; f(T_{\text{action}}; p) \\ &\text{subject to:} \\ &\qquad g(T_{\text{action}}; p) = 0 \\ &\qquad h(T_{\text{action}}; p) \leq 0 \end{aligned} \tag{3.4}$$

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. 3.3.1, a wide variety of costs and constraints can be used in the optimization (3.4). This allows the user to flexibly specify a wide variety of manipulation tasks. In practice we found that this specification was rich enough to cover all of our desired use cases. Although an exhaustive list is infeasible, we present several costs/constraints used in our experiments:

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

$$||T_{\text{action}} p_i - p_{\text{target\_}i}||^2 \tag{3.5}$$

   This is a relaxation of the target position constraint presented in Sec. 3.3.1.

2. Half space constraint on the keypoint:

$$\langle \text{plane}, T_{\text{action}} p_i \rangle \leq b_{\text{plane}} \tag{3.6}$$

   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. 3.3.1 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

$$||\langle n_{\text{plane}}, T_{\text{action}} p_i \rangle - b_{\text{plane}}||^2 \tag{3.7}$$

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. 3.3.1, the optimization (3.4) 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 [45, 87, 96] can be used to apply the abstracted robot action $T_{\text{action}} \in SE(3)$ produced by the kPAM optimization (3.4) to the manipulated object. If the object is rigid and the grasping 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 [45, 87, 96] 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 [151]. 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 [45, 86, 96].

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

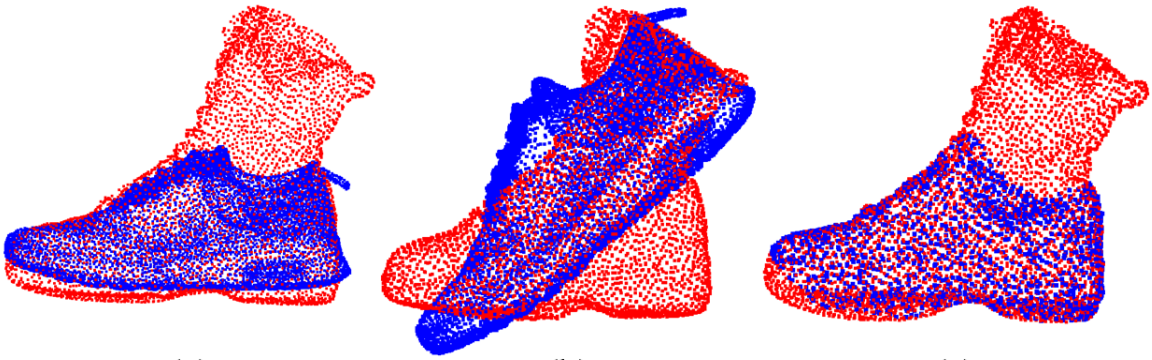


Figure 3-5: A pose representation cannot capture large intra-category variations. Here we show different alignment results from a shoe template (blue) to a boot observation (red). (a) and (b) are produced by [37] with variation on the random seed, and the estimated transformation consists of a rigid pose and a global scale. In (c), the estimated transformation is a fully non-rigid deformation field in [97]. 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.

keypoints as an object representation for manipulation, but rather shows the benefits of different representations for different pieces of the manipulation pipeline.

3.4 Comparison and Discussions

In this section we compare our approach, as outlined in Sec. 3.3, to existing robotic pick and place methods that use pose as the object representation.

3.4.1 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.

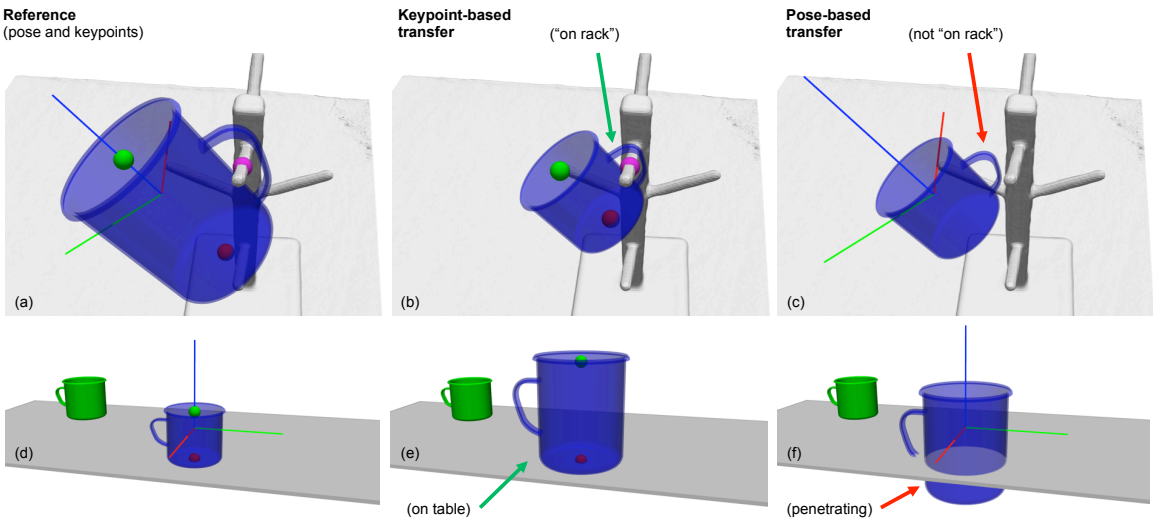


Figure 3-6: A comparison of the keypoint based manipulation with pose based manipulation for two different tasks involving mugs. The first row considers the mug on rack task, where a mug must be hung on a rack by its handle. (a) Shows a reference mug in the goal state, (b) and (c) show a scaled down 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.

However, the pose representation is not able to capture large intra-category shape variation. An illustration is presented in Fig. 3-5, where we try to align a shoe template (blue) to a boot observation (red). Fig. 3-5 (a) and (b) are produced by [37] where the estimated transformation consists of a rigid pose and a global scale. Fig. 3-5 (c) is produced by [97] 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. 3-5. Feeding these ambiguous estimations into a pose-based manipulation pipeline will produce different pick and place actions and final configurations of the manipulated 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. 3-7 as an example, transforming the keypoints p_5 and p_6 to a boot using the shoe to boot alignment in Fig 3-5 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. 3-5, remains an open problem. Our method avoids this problem by sidestepping the geometric alignment phase and directly detecting the 3D keypoint locations.

3.4.2 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. 3.3.1. 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

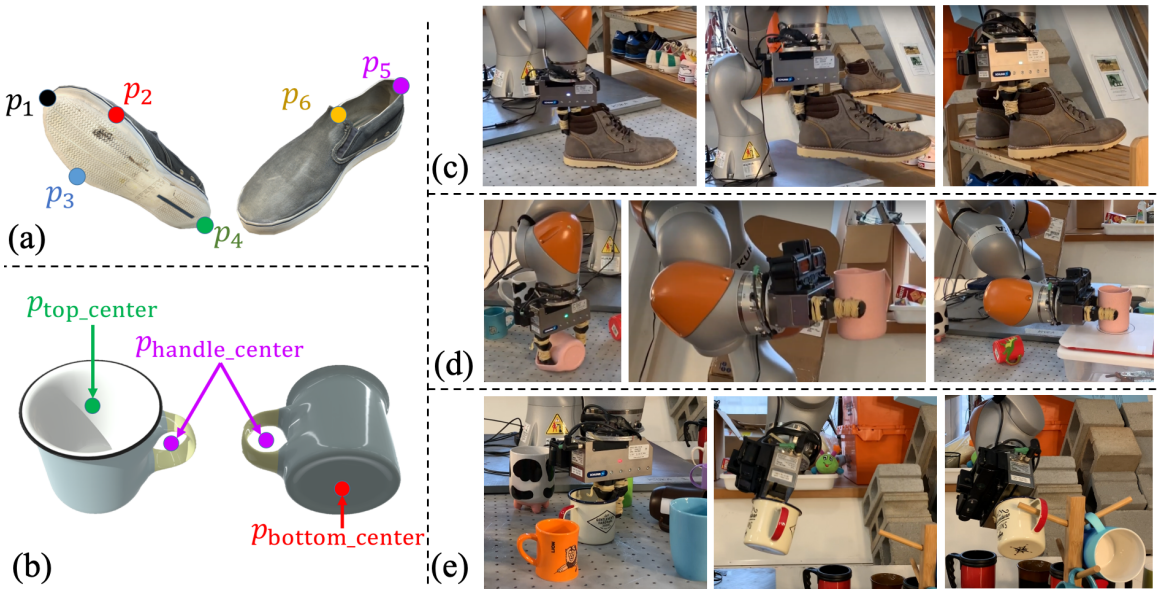


Figure 3-7: An overview of our experiments. (a) and (b) are the semantic keypoints we used for the manipulation of shoes and mugs. We use three manipulation tasks to evaluate our pipeline: (c) put shoes on a shelf; (d) put mugs on a mug shelf; (e) hang mugs on a rack by the mug handles. The video of these experiments are available on this link.

table task from Section 3.3.1. Fig. 3-6 (d) shows the target pose for the reference mug model. Directly applying this pose to the scaled mug instance in Fig. 3-6 (f) leads to physically infeasible state where the mug is penetrating the table. In contrast, using the optimization formulation of Section 3.3 results in the mug resting stably on the table, shown in Fig. 3-6 (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 3-6 (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. 3-6 (a) shows the reference model in the goal state. Fig. 3-6 (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. 3-6 (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.

(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 3-8: 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 this link.

3.5 Experiments

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 this link best demonstrates our solution to these tasks. More details about the experimental setup are in the Appendix A.

3.5.1 Put Shoes on a Shoe Rack

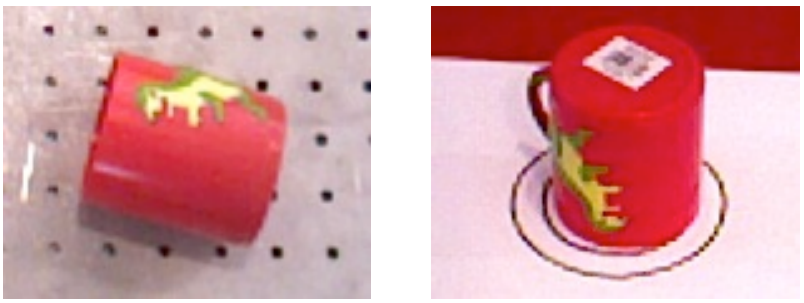
Task Description Our first manipulation task is to put shoes on a shoe rack, as shown in Fig. 3-7 (c). We use shoes with different appearance and geometry to evaluate the generality and robustness of our manipulation policy. The six keypoints used in this manipulation task are illustrated in Fig 3-7 (a), and the costs and constraints in the optimization (3.4) are

1. The L2 distance cost (3.5) 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 (3.7) 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 (3.6) is used to enforce this condition.

Experimental Results The shoe keypoint detection network was trained on a labeled dataset of 10 shoes, detailed in Figure 3-8 (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. 3-7) from their nominal target locations could be determined. Quantitative results are given in Fig. 3-8 (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 grasping 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. 3-8 (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



(a) (b)

Figure 3-9: An interesting failure case of our pipeline performing the “put mugs on a shelf task”. The input RGBD image is shown in (a), and the keypoint detector confuses the mug top with bottom. As a result, the pipline places the mug to the target configuration upside down.

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.

3.5.2 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. 3.3.1, as shown in Fig. 3-7 (d). The keypoints used in this task are illustrated in Fig. 3-7 (b). The costs and constraints for this task include the target position constraint (3.1) and the axis alignment constraint (3.2). If a target orientation is also specified w.r.t the yaw axis of the mug, we also add an L2 cost (3.5) between the $p_{\text{handle\_center}}$ keypoint with its target location. This task is similar to the mugs task in [44].

Experimental Results The mug keypoint detection network was trained on a dataset of 21 standard sized mugs, detailed in Fig. 3-8 (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. Quantative 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. 3-8 (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 was taken from a side-on profile where the handle is occluded and it is very difficult to tell 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 as shown in Fig. 3-9. 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.

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.

3.5.3 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. 3-7 (f). 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 (3.1) between $p_{\text{handle\_center}}$ to its target location on the rack axis.

2. The keypoint L2 distance cost (3.5) 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 (3.4) 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}.

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. 5 trials were run for each mug. Quantitative results are given in Fig. 3-8 (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 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 dur-

ing grasping could be alleviated by the addition of an external camera that would allow us to re-perceive the object after grasping.

3.6 Conclusion

In this chapter we propose 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.

Chapter 4

kPAM-SC: Generalizable Manipulation Planning using Shape Completion

4.1 Introduction

In Chapter 3, we propose a keypoint-based object (state) representation and demonstrate it on several challenging manipulation tasks. Chapter 3 mainly focuses on the reasoning of desired object configurations which generalizes to new objects. In this chapter, we study the generalizable manipulation planning problem: the pipeline should compute robot trajectories that move a set of objects to their target configuration while satisfying *physical feasibility* constraints. In contrast to existing works that assume known object templates, we are interested in manipulation planning for a category of objects with potentially unknown instances and large intra-category shape variations. To achieve it, we need an object representation with which the manipulation planner can reason about both the physical feasibility and desired object configuration, while being generalizable to novel instances.

Existing manipulation planners [39, 58, 117, 133] are built upon the 6-DOF pose representation of the manipulated objects. However as detailed in Chapter 3, 6-DOF pose cannot handle large intra-category shape variations. Consequently, in Chapter 3 we propose an object representation consists of semantic 3D keypoints, which provides a concise way to specify the target configuration for many objects. Although this approach has successfully accomplished several challenging manipulation tasks, it lacks the complete and

dense geometric information of the object. Thus, the kPAM pipeline in Chapter 3 cannot reason about physical properties such as the collision, static equilibrium, visibility, and grasp stability of the planned robot action, despite their practical importance in robotic applications. As a result, we need to manually specify various intermediate robot configurations to ensure the robot action is feasible, which can be labor-intensive and sub-optimal. Furthermore, the pipeline can be overly confident in physically infeasible robot trajectories and send them for execution, which is rather dangerous.

Building on Chapter 3, in this chapter we resolve this limitation with a new hybrid object representation which combines both (i) semantic 3D keypoints and (ii) full dense geometry (a point cloud or mesh). The dense geometry is obtained by leveraging well-established shape completion algorithms [92, 104, 155], which generalize well to novel object instances. With the combined dense geometry and keypoints as the object representation, we formulate the manipulation task as a motion planning problem that can encode both the object target configuration and physical feasibility for a category of objects. This motion planning problem can be solved by a variety of existing planners and the resulting robot trajectories can move the objects to their target configuration in a physically feasible way. The entire perception-to-action manipulation pipeline is robust to large intra-category shape variation. Extensive hardware experiments demonstrate our method can reliably accomplish manipulation tasks with never-before-seen objects in a category.

The contribution of this chapter is twofold. Conceptually, we introduce a hybrid object representation consists of dense geometry and keypoints as the interface between the perception module and planner. This representation has similar functionalities with existing 6-DOF pose representation with templates, while the generalizability to novel instances makes it a promising alternative. On the perspective of implementation, we contribute a novel integration of shape completion with the keypoint detection and manipulation planning. This integration enables many existing manipulation planners, either optimization-based methods [117, 133] or sampling-based approaches [39, 125], to handle a category of objects in a unified and precise way.

This chapter is organized as follows. Sec. 4.2 reviews the related works. Sec. 4.3 discusses the overall manipulation pipeline and its components. Sec. 4.4 presents experi-

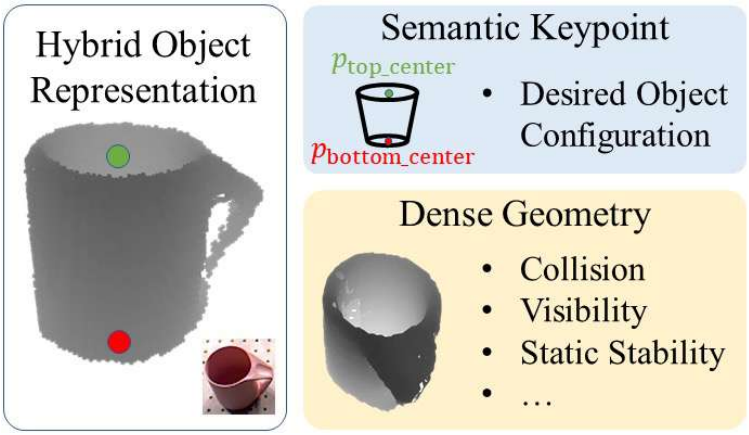


Figure 4-1: An overview of the proposed hybrid object representation for category-level manipulation planning. We exploit a hybrid object representation consists of semantic keypoints and dense geometry. The semantic keypoint are used to specify the desired object target configuration, while the dense geometry is used to ensure the physical feasibility of the planned robot action. Benefit from advances in learning based keypoint detection and shape completion, the proposed object representation can be obtained from raw images, and the resulting perception-to-action pipeline generalizes to novel instances within the given category.

mental results on a hardware robot and show our method is able to reason about physical feasibility for a category of objects despite large intra-category shape variation. Sec. 4.5 concludes this chapter.

4.2 Related Work

4.2.1 Pose-based Manipulation Planning

For most manipulation planner, the object pose is the default interface between the from the perception module: the perception module estimates the pose from raw sensor inputs; the planner takes the estimated pose as input and plans robot actions. it is typically assumed that the planner has access to the geometric template of the object. Researchers have made various contributions on the manipulation planning algorithms, as reviewed in Sec. 2.1. System contributions [16, 61] integrate state-of-the-art pose estimators with trajectory planners to build fully functional manipulation pipelines and solve real-world tasks such as packing and assembly.

However as mentioned in Sec. 3.4, pose estimation can be ambiguous under large intra-category shape variations, and using one geometric template for motion planning and object target specification can lead to physically infeasiblity for other instances. Thus, pose-based object representation is not suitable for manipulation planning of many different objects.

4.2.2 Grasping and Manipulation with Shape Completion

Robot grasp planning is the task of computing a stable grasp pose that allows the robot to reliably pick up the object. A detailed discussion of grasp planning algorithms is presented in Sec. 2.2.2. As the geometry obtained from typical RGBD sensors are noisy and incomplete, several works on grasp planning [85, 106, 138, 140, 143] combine shape completion with grasping planning for improved performance and robustness. Some of these methods [85, 140] estimate the grasp quality based on geometric information such as antipodal points or surface normal, using the completed geometry instead of raw sensory observation. [106] also shows shape completion can improve the performance of robot object searching.

In this work, we are interested in category-level robotic manipulation planning. This task requires reasoning about both the desired object configuration and physical feasibility, and is out of the scope for the above-mentioned methods.

4.3 Manipulation Pipeline

As illustrated in Fig. 4-2, we use the hybrid object representation consists of dense geometry and keypoints as the interface between the perception and planning modules. The semantic keypoints are designated manually and used to specify the object target configuration, while the dense geometry is used to ensure the physical feasibility of the planned robot action. The perception part includes shape completion and keypoint detection, which is detailed in Sec. 4.3.1. The manipulation planning and grasp planning that use the perceived results are presented in Sec. 4.3.2 and Sec. 4.3.3, respectively.

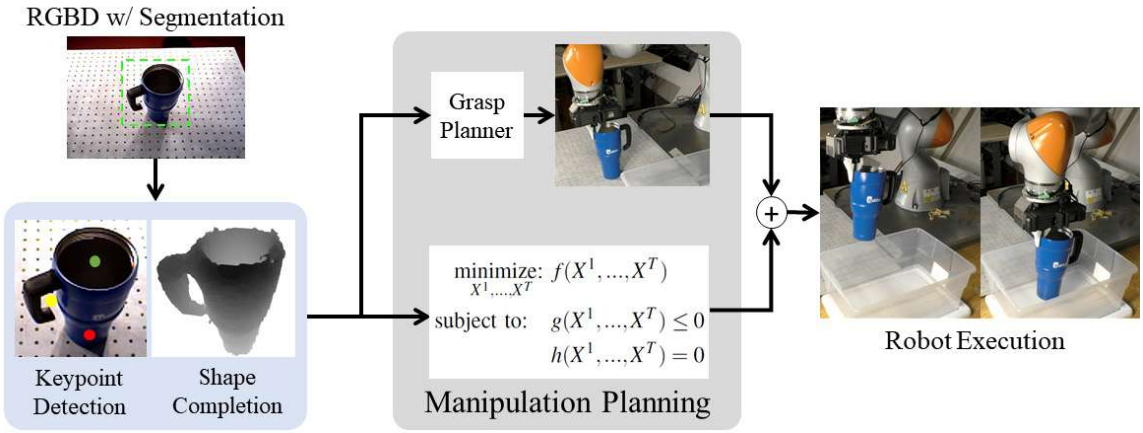


Figure 4-2: An overview of the manipulation pipeline. The hybrid object representation consists of dense geometry and keypoint is used as the interface between the perception module and manipulation planner. Given a RGBD image with instance segmentation, we perform shape completion and keypoint detection to obtain dense geometry and 3D keypoints, respectively. Then, the perception result is used to plan robot trajectories that move the objects to their desired configurations while satisfying physical constraints. Semantic keypoints are used to specify the object target configuration, while dense geometry is used to ensure the physical feasibility of the planned robot action.

4.3.1 Perception

The task of perception is to produce the proposed hybrid object representation from raw sensor inputs, which consists of 3D keypoints and dense geometry (point cloud or mesh). For 3D keypoints we adopt the method in our previous work kPAM [88], and this subsection would mainly focus on the perception of dense geometry. For dense geometry, we leverage recent advances in shape completion to obtain a complete point cloud or mesh of the object. Note that although we present specific approaches used in our pipeline, any technique for keypoint detection and shape completion can be used instead.

We use the state-of-the-art ShapeHD network [155] for 3D shape completion. ShapeHD is a fully convolutional network that takes RGBD images as input and predicts 3D volumetric occupancy. Then the completed point cloud can be extracted by taking the occupied voxel. If the object mesh is required, triangulation algorithms such as marching cubes can be used. The completed geometry are aligned with the observed object (viewer-centered in [155]) and expressed in the camera frame, we can further transform it into the world frame using the calibrated camera extrinsic parameters.

The shape completion network requires training data consists of RGBD images and corresponding ground-truth 3D occupancy volumes. We collect training data using a self-supervised method similar to LabelFusion [90]. Given a scene containing one object of interest we first perform 3D reconstruction of that scene. Then, we perform background subtraction to obtain the reconstructed mesh of the object. Finally, we can get the occupancy volume by transforming the reconstructed mesh into camera frame and voxelization. Note that the data generation procedure does not require pre-built object template or human annotation. In our experiment, we scan 117 training scenes and collect over 100,000 pairs of RGBD images and ground-truth 3D occupancy volumes within four hours. Even with small amount of data we were able to achieve reliable and generalizable shape completion, some qualitative results are shown in Sec. 4.4.2.

4.3.2 Manipulation Planning

The manipulation planning module produces the robot trajectories given the perception result. As mentioned in Sec. 4.3, we represent an object o_j by its 3D semantic keypoints $p_j \in R^{3\times N}$ and dense geometry (point cloud or mesh), where $1 \leq j \leq M$ and M is the number of objects.

Following many existing works [39, 47, 125], we assume that the robot can change the state of an object only if the object is grasped by the robot. Furthermore, we assume the object is rigid and the grasp is tight. In other words, there is no relative motion between the gripper and the grasped object. Both the semantic keypoints and dense geometry would move with the robot end-effector during grasping. To achieve this, we need a grasp planner which is discussed in Sec. 4.3.3.

Given the object representation, the concatenated configuration for robot and objects at time t is defined as $X^t = [o_1^t, ..., o_M^t, q^t]$, where $1 \leq t \leq T$, T is the number of time knots

and q^t is the robot joint configuration. The general planning problem can be written as

$$\underset{X^1,...,X^T}{\text{minimize:}} \; f(X^1,...,X^T) \tag{4.1}$$

$$\text{subject to:} \quad g(X^1,...,X^T) \leq 0 \tag{4.2}$$

$$h(X^1,...,X^T) = 0 \tag{4.3}$$

where f is the objective function, g and h are the concentrated inequality and equality constraints. If optimization-based planning algorithms [117, 131] are used to solve the problem (4.1), f, g and h should be differentiable. On the other hand, many sampling-based planners [62, 75] or TAMP algorithms [39, 60] only need a binary predicate on whether the constraint is satisfied.

Using the proposed object representation consist of semantic 3D keypoints and dense geometry, the key benefit is that the motion planner can handle a category of objects without instance-wise tuning. In the following text, we discuss several important costs and constraints that are related to the object representation.

Object Target Configuration Let p_j^t be the keypoints of the object o_j at the time t, where $1 \leq t \leq T$. The target configuration of an object o_j can be represented as a set of costs and constraints on its semantic keypoint p_j^T, where T is the terminal time knot. For instance, to place the mug at some target location p_{target} as illustrated in Fig. 4-1, we need an equality constraint

$$||p^T_{\text{bottom\_center}} - p_{\text{target}}|| = 0 \tag{4.4}$$

where $p^T_{\text{bottom\_center}}$ is the mug bottom-center keypoint expressed at time T. Note that this constraint can handle mugs with different size, shape and topology. Many other costs and constraints can be used to specify the object target configuration. Please refer to kPAM [88] for more details.

Collision Avoidance The dense geometry information from shape completion can be used to ensure the planned trajectory is collision-free. Specifically, let B_r denote the set of rigid bodies of the objects, robot and environment, where the geometry of objects are obtained

using shape completion. We need to ensure

$$\text{signed\_distance}(X^t; b_i, b_j) \geq \delta_{\text{safe}} \tag{4.5}$$

$$\text{for } b_i \in B_r,\ b_j \in B_r,\ i \neq j,\ 1 \leq t \leq T \tag{4.6}$$

where δ_{safe} is a threshold, signed_distance$(X^t; b_i, b_j)$ is the signed distance [117] between the pair of rigid body (b_i, b_j) at the configuration X^t. Practically, it is usually unnecessary to check the collision of every rigid body pairs, as most rigid bodies except the grasped object and robot end-effector have limited movement.

Geometric Predicates In many planning algorithms [48, 133], geometric predicates are used to model the geometric relationship between the objects and the environment. Although these predicates are typically proposed in the context of known objects with geometric templates, they can benefit from shape completion and naturally generalize to a category of objects. Here we summarize several examples used in these manipulation planners.

- The static stability constraint enforces that the object placement surfaces are aligned with one of the environment placement regions. To use this predicate, it is required to extract the surfaces on the object that afford placing from the object dense geometry. Please refer to [48] for more details.
- The visibility constraint requires the line segments from the sensor to the object are not blocked by other objects or the robot. In other words, the manipulate object should not block or be blocked by other objects.
- The containment constraint enforces the convex hull of an object is included in a container. This constraint needs the convex hull of the object, which can be computed using the dense geometry from shape completion.

4.3.3 Grasping

The grasp planning module is responsible to compute a grasp pose that allows the robot to stably pick up and transfer the object. Various algorithms [45, 87, 96] have been proposed

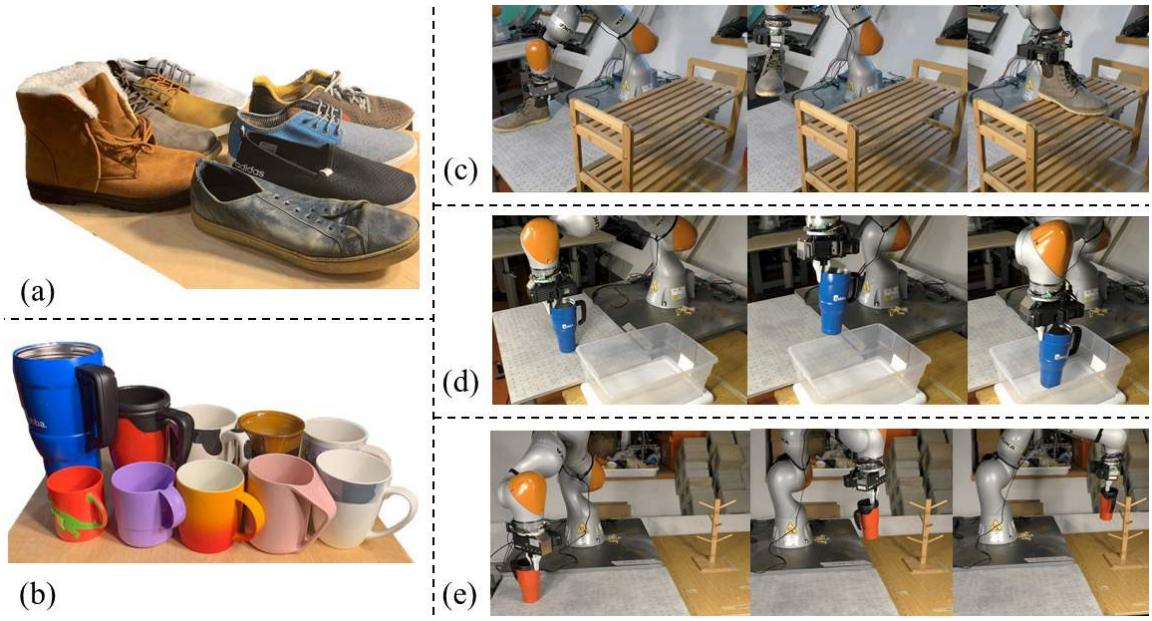


Figure 4-3: An overview of our experiments. (a) and (b) are the shoes and mugs used to test the manipulation pipeline. Note that both the shoes and mugs contain substantial intra-category shape variation. We use three manipulation tasks to evaluate our method: (c) put shoes on a shelf; (d) put mugs in a container; (e) hang mugs on a rack by the mug handles. Readers are recommended to visit our this link to watch the video demo.

for grasp planning. Some of them [85,140,143] are built upon shape completion and can be easily integrated into our pipeline. These algorithms are object-agnostic and can robustly generalize to novel instances within a given category.

4.4 Results

In this section, we demonstrate a variety of manipulation tasks using our pipeline. The particular novelty of these demonstrations is that our method can automatically plan robot trajectories that handle large intra-category variations without any instance-wise tuning or specification. The video demo and source code are available on our this link.

4.4.1 Experiment Setup and Implementation Details

We use 8 shoes and 10 mugs to test the manipulation pipeline, as illustrated in Fig. 4-3. Note that both shoes and mugs have substantial intra-category shape variation. The

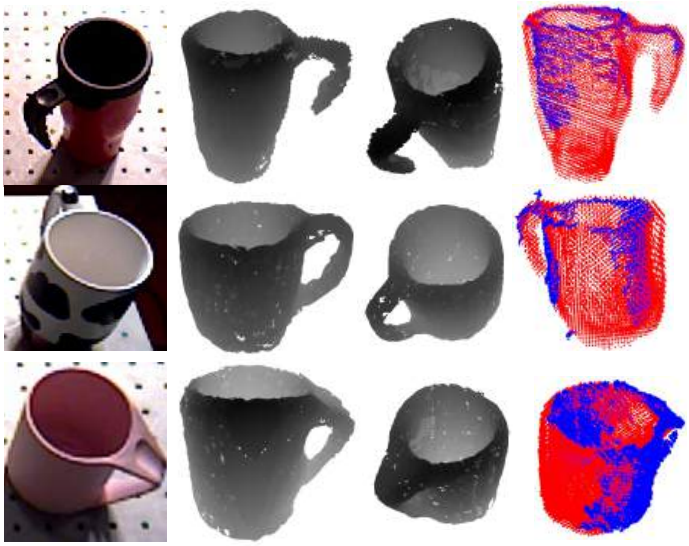


Figure 4-4: The qualitative results for shape completion. (a) is the input RGB image. (b) and (c) are the dense geometry from shape completion in two viewing directions. (d) compares the point cloud from depth image and shape completion: the depth image point cloud is in blue while shape completion is in red. Although the completed geometry contains small holes and defects, the accuracy is sufficient for many manipulation tasks.

Table 4.1: Training Data Statistics

Object Type	**# Train Objects**	**# Scenes**	**# Images**
Shoe	12	51	41056
Mug	21	74	70094

keypoints same as Chapter 3 are used to define the target configuration of the shoe and mug. The statistics of the training objects is shown in Table 4.1. The robot setup is identical to Chapter 3.

The drake library [131], which implements optimization based motion planning, is used for manipulation planning. The costs and constraints include object target configuration and collision avoidance. For the purpose of this work, we use a fixed contact-mode sequence consists of picking, transferring and placing of the object, as the scheduling of contact-mode sequences is the main focus of task-level planning. We emphasize that the proposed object representation and manipulation pipeline are agnostic to the concrete planning algorithm that solves (4.1). Many motion planners, either optimization-based methods [117, 133] or sampling-based approaches [39, 125], can be plugged in and used.

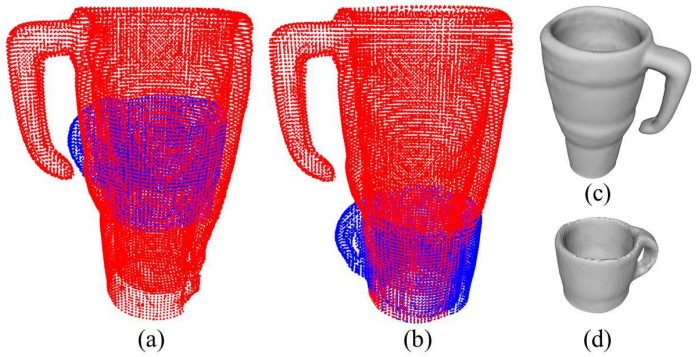


Figure 4-5: Pose representation cannot capture large intra-category variations. (a) and (b) show two alignment results by [37] (variation on the random seed), where we attempt to register the mug template (d) into the observation (c). Using these pose estimation results for manipulation planning can lead to physically infeasibility, as the rigid transformation defined on the mug template (d) cannot capture the geometry of mug (c). Additionally, there may exist multiple suboptimal alignments which make the pose estimator ambiguous.

4.4.2 Perception and Comparison with Pose Estimation

In this subsection, we provide results of shape completion and compare it with the widely-used 6-DOF pose representation. Fig. 4-4 shows several completed dense geometry for representative mugs. The network takes input from images in Fig. 4-4 (a) and produces the dense geometries in Fig. 4-4 (b) and (c). Fig. 4-4 (d) compares the point cloud from depth images and shape completion. Although the completed geometry contains small holes and defects, the accuracy is sufficient for many manipulation tasks. Note that the network generalizes to instances with substantial variations on geometry and topology.

In contrast, pose estimation can be ambiguous and fail to capture large shape variations. An illustration is provided in Fig. 4-5. where the pose estimator in [37] is used to align two mugs. Using dense geometry from pose estimation can lead to undetected physically infeasibility, as shown in Sec. 4.4.5.

4.4.3 Manipulation Task Specifications

We use the following three manipulation tasks to test our pipeline:

1) Put shoes on a shelf: The first manipulation task is to put shoes on a shoe shelf, as shown in Fig. 4-3 (c). The final shoe configuration should be in alignment with the shelf.

Table 4.2: Robot Experiment Statistics

Task	#Trails	#Failure	#Planning Failure	#Grasp Failure	#Execution Failure
shoe2shelf	50	14	12	2	0
mug2container	50	10	9	0	1
mug2rack	50	15	10	0	5

Table 4.3: False-Negative (Overly Confident, see Sec. 4.4.5) Statistics

Manipulation Pipeline	#Trails	#False Negative (Overly Confident)	False Negative Rate
6-DoF Pose	50	19	38%
kPAM	50	9	18%
Ours	50	1	2%

The manipulation pipeline has the pre-built template of the robot and the shoe rack, but need to deal with shoes with different appearance and geometry.

2) Place mugs into a container: The second manipulation task is to place mugs into a box without colliding with it, as shown in Fig. 4-3 (d). The mug should be upright and its handle should be aligned with the container.

3) Hang mugs on a rack: The last manipulation task is to hang mugs onto a mug rack by the mug handle, as shown in Fig. 4-3 (e). The geometry and position of the mug rack are available to the pipeline.

Note that although task 1) and 3) have been performed in Chapter 3, it uses various manually-specified robot trajectories. This manual specification has two major limitations: 1) it is labor-intensive and can hardly scale to more complex environments and manipulation tasks; 2) the pipeline tends to be overly confident, as it cannot detect physical infeasibility without dense geometry. A comparison is made in Sec. 4.4.5.

4.4.4 Result and Failure Mode

The video demo is on this link. The statistics about three different tasks is summarized in Table. 4.2. Most failure cases result from the failure of the motion planner. Since the

motion planner used in this work uses non-convex optimization internally, it can be trapped in bad local minima without a good initialization. This problem can be resolved by using sampling-based motion planners such as RRT or RRT*. These planners are globally optimal, although they need longer running time.

The grasp failure in Table. 4.2 means (1) the robot fails to grasp the object; or (2) the relative motion between the gripper and the grasped object is too large. The relative motion may occur during the grasping, or when the object is not rigid. This problem could be alleviated by the addition of an external camera that would allow us to re-perceive the object after grasping.

The execution failure in Table. 4.2 refers to the situations such that (1) the robot makes collision (despite the planning is successful); or (2) the object is not placed into the target configuration. This problem necessitates the execution monitoring described in [24].

4.4.5 Comparison with Alternative Pipelines

In this subsection, we compare our method with two alternative manipulation pipelines: 1) a manipulation pipeline based on 6-DOF pose with a geometric template; 2) the original kPAM pipeline. For the 6-DOF pose based manipulation pipeline, we use the same pose estimator as the baseline (Fig. 4) in kPAM: first initialize the alignment with detected keypoints, then perform ICP fitting between the observed point cloud and geometric template to get the 6-DOF pose.

Without an accurate characterization of the dense geometry, these alternatives suffer from false-negative (overly-confidence): the resulted robot trajectory can be physically infeasible even if the pipeline claims both the perception and planning succeed. Note that this false-negative is much more adversarial than the planning failure in Table. 4.2 (Sec. 4.4.4). When the planner fails the pipeline would be stopped and it is still safe. In contrast, when false-negative happens the unsafe trajectory would be sent to the robot for execution, which is rather dangerous.

Table. 4.3 summarizes the false-negative rate of all three methods on the "mug2container" task. We mark a trail as "false-negative" if the pipeline claims perception and planning suc-

ceeds, but the resulted trajectory leads to a collision. The false-negative rates of the two alternatives are much higher than our method, which implies our method is much safer. This highlights the benefit of accurate characterization of dense geometry and the integration of shape completion into the manipulation pipeline.

4.5 Conclusions

In this chapter, we focus on manipulation planning of a category of objects, where the robot should move a set of objects to their target configuration while satisfying physical feasibility. This is challenging for existing manipulation planners as they assume known object templates and 6-DOF pose estimation, which doesn't generalize of novel instances within the category. Thus, we propose a new hybrid object representation consists of semantic keypoints and dense geometry as the interface between the perception module and planning module. On the perspective of implementation, we contribute a novel integration of shape completion with keypoint detector and manipulation planner. In this way, both the perception and planning module generalizes to novel instance. Extensive hardware experiments demonstrate our manipulation pipeline is robust to large intra-category shape variation.

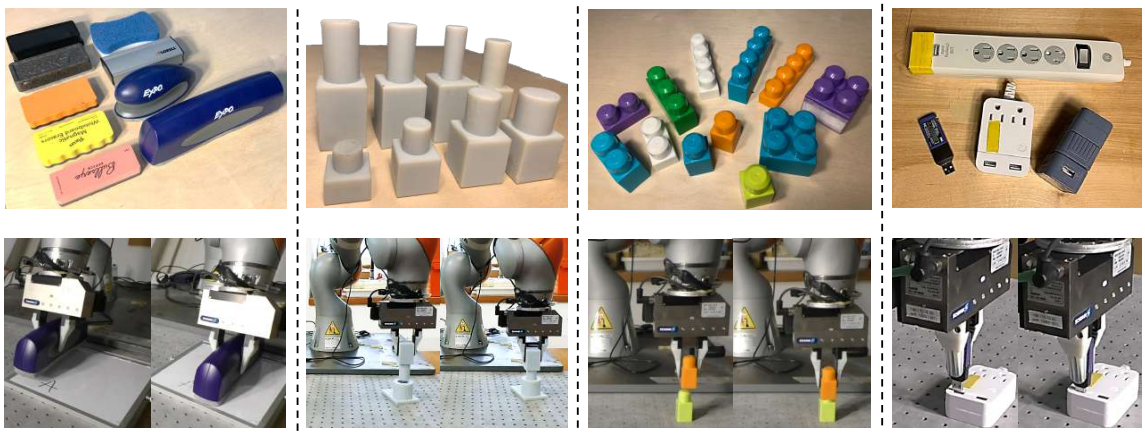
Chapter 5

kPAM 2.0: Feedback Control for Generalizable Manipulation

5.1 Introduction

In this chapter we focus on generalizable, closed-loop manipulation for contact-rich tasks. We have seen pick-and-place manipulation that can handle a variety of objects in our previous works (Chapter 3 and 4). However, these prior works are *open-loop*: once the action has been planned, the robot would close the eye and simply follow the action. Open-loop manipulation policies are not suitable for tasks such as using a screwdriver or peg-hole insertion, where the control action must be regulated online with both visual and proprioceptive measurements. In this chapter, we aims at a manipulation framework that is capable of performing these types of contact-rich tasks, while being generalizable to a category of objects with potentially unknown instances with different shapes, sizes, and appearances. Furthermore, the framework should be able to handle different initial object configurations and robot grasp poses for practical applicability.

Contributions on visuomotor policy learning exploit neural network policies trained with data-driven algorithms [31,77,139,156], and many interesting manipulation behaviours emerge from them. However, how to efficiently generalize the trained policy to unseen objects, different initial object configurations, camera positions and/or robot grasp poses remains an active research problem. On the other hand, several vision-based closed-loop



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

Figure 5-1: In this chapter we explore generalizable, perception-to-action robotic manipulation for contact-rich tasks that can automatically handle a category of objects with large intra-category shape variation. Here we demonstrate: (a) wiping a whiteboard using erasers with different shape and size; (b)-(d) Peg-hole insertion for (b) 0.2 [mm] tight tolerance pegs and holes, (c) LEGO blocks and (d) USB ports. The particular novelty of these demonstrations is that our method automatically handle objects under significant intra-category shape variation (top row) without any instance-wise tuning, for each task (a)-(d). The video demo is on https://sites.google.com/view/generalizable-feedback/home.

manipulation pipelines [56, 61, 100] use 6-DOF pose as the object representation. They build a feedback loop on top of a real-time pose estimator. However, as detailed in Sec. 4 of Chapter 3, representing an object with a parameterized pose defined on a fixed geometric template, as these works do, may not adequately capture large intra-class shape or topology variations. Thus, in Chapter 3 (kPAM) we use semantic 3D keypoints as the object representation instead of 6-DOF pose. kPAM assumes an arbitrary rigid transformation can be applied to the object. This assumption is not true for contact-rich tasks in this work: although peg-hole insertion is eventually a rigid transformation of the peg, it is not easy to "apply" this rigid transformation.

In this chapter, we contribute a novel manipulation framework that is capable of precise, contact-rich manipulation for a category of objects, despite large intra-category variations in shapes, sizes and appearances. To achieve this, we adopt and extend the keypoint-based object representation in Chapter 3 for pick-and-place. We first augment keypoints with local orientation information. Using the oriented keypoint, we propose a novel object-centric action representation as the linear/angular velocity or force/torque of an oriented

keypoint. This action representation enables closed-loop policies and contact-rich tasks, despite intra-category shape and size variations of manipulated objects. Using the object and action representation, we further introduce the processing of force/torque measurement for a category of objects, the re-targeting to different initial object configuration, and significant simplification of the kPAM pick-and-place pipeline. Moreover, our framework is also agnostic to robot grasp poses, enabling flexible integration with grasp planners.

Another desirable property of our framework is the extendibility. As shown in Sec. 5.3.2, our framework includes a perception module and a feedback agent, establishes their interfaces but leaves the room for their actual implementation. Thus, various existing model-based or data-driven algorithms for perception and control can potentially be plugged into our framework and automatically generalize to new objects and task setups, as long as the proposed object and action representation are used as their input/output.

Our framework is instantiated and implemented on a hardware robot. We demonstrate several contact-rich manipulation tasks that requires precision and dexterity for a category of objects, such as peg-hole insertion for pegs and holes with significant shape variations and tight clearance.

This chapter is organized as follows. Sec. 5.2 review related works. Sec. 5.3 describes our formulation of the closed-loop manipulation framework. Sec. 5.4 show the orientation information can significantly simplify the kPAM pipeline for category-level pick-and-place manipulation. Sec. 5.5 discusses our implementation of the perception module. Sec. 5.6 presents hardware experiments on several challenging tasks, specifically showing generalization of our method. Sec. 5.7 concludes.

5.2 Related Work

5.2.1 Object Representation for Closed-Loop Manipulation

Several teams [56, 61, 100] incorporate real-time pose estimators into closed-loop manipulation pipelines and show impressive manipulation demos. To generalize these pipelines to a category of objects, a straightforward approach would be combining them with a

category-level pose estimator such as [115, 142]. However, as detailed in the Chapter 3, pose estimation can be ambiguous under large intra-category shape variations; directly feeding an estimated pose into a manipulation pipeline can lead to physical infeasibility for other instances in the category.

On the other hand, many contributions propose to train a visuomotor policy using data-driven algorithms [4,31,32,77,116,139,154,156]. The state representation in these methods is usually an internal state of the policy neural network (or the “image state"). As reviewed in Sec. 2.2.3, many interesting manipulation behaviours emerge in these approaches. Compared with these methods, the key advantage of our framework is the automatic generalization to new object instance, camera position, initial object configuration and robot grasp pose. On the other hand, many of these data-driven algorithms can be integrated into our framework and that would be a promising future direction.

5.2.2 Robotic Manipulation with Proprioceptive Feedback

There have been impressive works [2, 112, 130] on robot control with applications to several industrially important tasks, as reviewed in Sec. 2.1. By using joint torque sensors and/or 6-DOF force/torque sensors along with other proprioceptive sensors, the robot can perform many impressive tasks, for instance peg-hole insertion with very tight tolerance (H7h7) [130] or polishing an non-flat surface with smooth motion and force trajectories [2]. However, these approaches typically assume perfect knowledge of the geometry and object location. In this way, these methods eliminate the inaccuracy caused by perception and grasping. For many tasks these pre-requisites can be hard to satisfy.

5.2.3 Pick-and-Place Manipulation at a Category Level

Several works [109, 149] focus on pick-and-place manipulation for a category of objects. The kPAM pipeline in Chapter 3 proposed to use semantic 3D keypoint as the object representation. KETO [109] extends kPAM [88] with self-supervised keypoint learning, It demonstrates robotic tool manipulations such as hammering and pushing. Form2Fit [149] uses shape descriptors to perform object placement in robot assembly.

In this chapter, we focus on generalizable manipulation with closed-loop feedback for contact-rich scenarios. Using an open-loop policy, as the aforementioned works [109, 149] do, typically cannot accomplish these tasks. A comparison is made in the Sec. 5.6.2 of our experiment.

5.3 Manipulation Framework

In this section, we discuss our formulation of generalizable manipulation framework. Sec. 5.3.1 describes the approach using a concrete example, and Sec. 5.3.2 presents the general formulation. The subsequent sections discuss the details and extensions of the general formulation.

5.3.1 Concrete Motivating Example

Consider the task of peg-hole insertion, as illustrated in Fig. 5-2 (a). We want to come up with a manipulation policy that automatically generalizes to a different peg in Fig. 5-2 (b), and a different robot grasp pose in Fig. 5-2 (c).

In Chapter 3, we propose to represent the object by a set of semantic 3D keypoints. The motivation is: keypoint is well defined within a category while 6-DOF pose cannot capture large shape variation (see Sec. 4 of Chapter 3 for details). We adopt this idea and choose two keypoints: the $p_{\text{peg\_end}}$ that is attached to the peg and the $p_{\text{hole\_top}}$ that is attached to the hole, as shown in Fig. 5-2 (d). Similar to kPAM, we assume that we have a keypoint detector, for instance a deep network trained with supervised data, that can produce these specified keypoints in real-time.

These two keypoints provide the location information. However, the peg-hole insertion task also depends on the relative orientation of the peg and hole. Thus, we augment the keypoint with orientation information, as if a rigid coordinate is attached to each keypoint, as shown in Fig. 5-2 (e). For the peg-hole insertion task, we let the z axis of the $p_{\text{peg\_end}}$, $p_{\text{hole\_top}}$ be the axis of the peg and hole, respectively. The x axis of $p_{\text{peg\_end}}$, $p_{\text{hole\_top}}$ can be chosen arbitrarily, but when the x axes of $p_{\text{peg\_end}}$ and $p_{\text{hole\_top}}$ are aligned the peg should be able to insert into the hole.

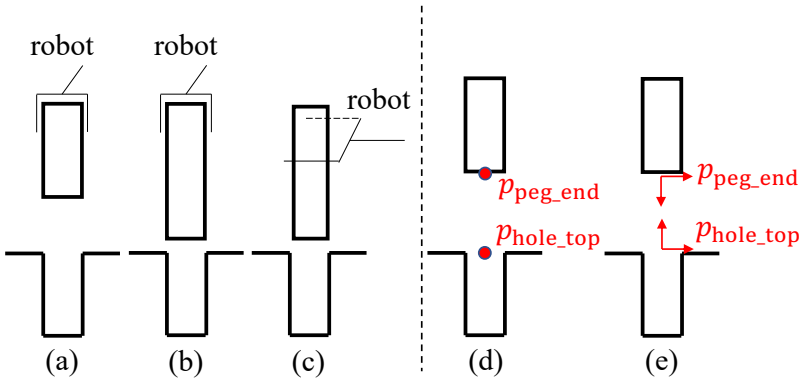


Figure 5-2: **The object representation using peg-hole insertion as an example**. We would like the manipulation pipeline generalize to (b) a different peg; and (c) a different robot grasp pose. We adopt the semantic 3D keypoint proposed in kPAM [88] as a local but task-specific object representation, as shown in (d). Since the task depends on the local relative orientation between the peg and hole, we augment keypoint with orientation information, as if a rigid coordinate is attached to each keypoint shown in (e).

The coordinate in Fig. 5-2 (e) is also used to illustrate 6-DOF pose in the literature. The key difference between the oriented keypoint and 6-DOF pose is: oriented keypoint is a *local* but task-specific characterization of the object geometry, while pose with geometric template is global. The choice of a local object representation is inspired by the observation that in many manipulation tasks, only a local object part interacts with the environment and is important for the task. For instance, the $p_{\text{peg\_end}}$ keypoint only characterizes a local object part that will be engaged with the hole, and it does not imply task-irrelevant geometric details such as the handle grasped by the robot. This locality enables generalization to novel objects as the unrelated geometric details are ignored.

As illustrated in Fig. 5-3, with the oriented keypoint we propose to represent the robot action as either 1) the desired linear and angular velocity of the $p_{\text{peg\_end}}$ keypoint, as shown in Fig. 5-3 (a); or 2) the desired force and torque at the $p_{\text{peg\_end}}$ keypoint, as shown in Fig. 5-3 (b). Note that these two object-centric action representations are agnostic to the robot grasp pose, since these actions are defined only w.r.t the object (not the robot). Under the assumption of no relative motion between the peg and the robot gripper (the grasp is tight), these actions can be mapped to joint space commands, as described in Sec. 5.3.5.

Suppose we have implemented an agent (which can be a hand-written controller or a neural network policy) using the object and action representations mentioned above as the

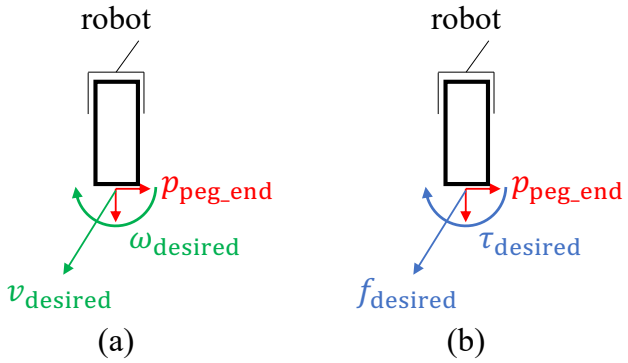


Figure 5-3: **Overview of the object-centric action space**. The action can be: (a) the desired linear/angular velocity of an oriented keypoint; (b) the desired force/torque of an oriented keypoint. Note that the action space is agnostic to the robot grasp pose.

input and output, together with a perception module that produces the required keypoints in real-time and a joint-level controller that maps the agent output to joint command, then the resulting manipulation policy would automatically generalize to different objects and robot grasp poses, for instance the ones in Fig. 5-2 (a), (b) and (c). Even if the policy doesn't directly transfer due to unmodeled factors, it would be a good initialization for many data-driven or model-based algorithms [71, 78, 119].

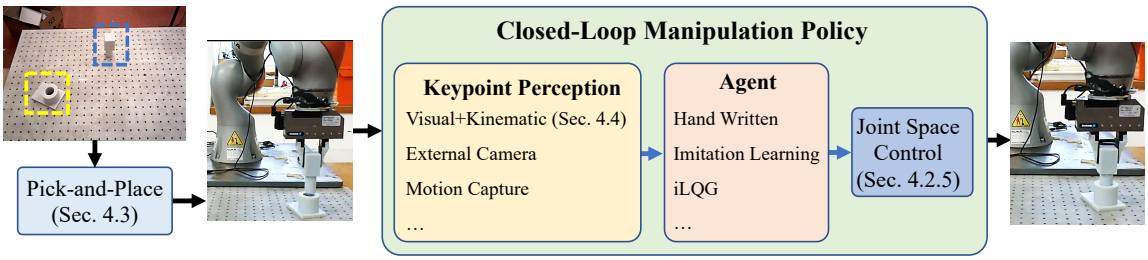


Figure 5-4: **Overview of the manipulation framework**. The closed-loop policy consists of 1) a perception module that produces oriented keypoint in real time; 2) an agent with the state and action space shown in Fig. 5-2 and Fig. 5-3, respectively; 3) a joint-space controller that maps agent outputs to joint-space commands. Note that many different implementations of the perception module and agent can be used within our framework and the resulting pipeline automatically generalize to new objects and task setups. For many applications the objects are randomly placed initially. In this scenario, we perform a kinematic pick-and-place to move the object to some desired initial condition (for instance moving the peg right above the hole), from where the closed-loop policy starts operating, as detailed in Sec. 5.4.

5.3.2 General Formulation

We can think of a robot as a programmable force/motion generator [52]. We propose to represent the task-specific motion profile as the motion of a set of oriented keypoints, and the force profile as the spatial force/torque w.r.t some keypoints.

Thus, given a category-level manipulation problem we propose to solve it in the following manner. First the modeler selects a set of 3D oriented keypoints that capture the task-specific force/motion profile. Once we have chosen keypoints, the manipulation framework can be factored into: 1) the perception module that outputs the oriented keypoint from sensory inputs; 2) the agent that takes the perceived keypoint as input and produces the desired linear/angular velocity or force/torque of an oriented keypoint as output; 3) the low-level controller that maps the agent output to joint-space command. An illustration is shown in Fig. 5-4. The framework can be extended with force/torque measurements and generalization to different initial object configurations, as shown in Sec. 5.3.3 and Sec. 5.3.4, respectively. For many applications, objects are randomly placed initially. In this case, we perform a kinematic pick-and-place to move the object to some desired initial configurations, from where the closed-loop policy starts, as shown in Sec. 5.4. To make the overall manipulation operation generalizable, all these extensions must work for a category of objects.

It should be emphasized that our framework establishes the interfaces of the perception module or closed-loop agent, but leaves the room for their instantiation. The only requirement is that for the perception module it should output oriented keypoints in real time, and for the agent it should use the state and action space mentioned above. There are many solutions for both of them. For instance, in our experiment a wrist-mounted camera is used for visual keypoint detection and the robotic kinematics is used for realtime keypoint tracking (see Sec. 5.5). Alternatively, external cameras or motion capture markers can also be used for keypoint perception. Similarly one might explore various model-based or data-driven controllers as the feedback agent according to the task in hand. As long as these perception and control module use the proposed object and action representation as the input/output, they can be plugged into our framework and the resulting policy automatically generalize

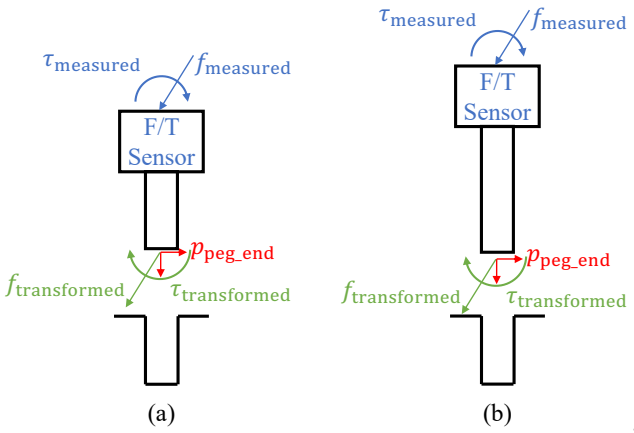


Figure 5-5: **The processing of force/torque measurement in our formulation**. For a wrist-mounted force/torque sensor, its raw measurements f_{measured} and τ_{measured} vary with the object geometry and grasp pose. Thus, we propose to transform them to an oriented keypoint ($p_{\text{peg\_end}}$ here), as the transformed measurement captures the task-specific force/torque profile. The closed-loop agent takes the transformed force/torque measurement as input would generalize w.r.t object geometry.

to novel objects and task setups.

5.3.3 Force/Torque Measurement

Some robots are equipped with wrist-mounted force/torque sensors or joint torque sensors. For contact-rich manipulation tasks, it's beneficial use this information as the input of the agent. However, the raw output from these sensors varies with the object geometry and robot grasp pose, as shown in Fig. 5-5. As a result, directly feeding these measurements into the agent does not generalize automatically.

The solution to this problem is to transform the measured force/torque to the kinematic frame of an oriented keypoint, as shown in Fig 5-5. Using the peg-hole insertion as an example, we can transform the force/torque measurement from the robot wrist to the coordinate of $p_{\text{peg\_end}}$, as if a "virtual sensor" is mounted at $p_{\text{peg\_end}}$. In this way, we can expect similar force/torque profiles across different objects and robot grasp pose, as shown in Fig 5-5.

If the robot is equipped with a joint torque sensor, we can also estimate the force/torque on $p_{\text{peg\_end}}$ using robot kinematics by assuming that the robot has no other contact. Let $J_{\text{peg\_end}}$ be the Jacobian that maps robot joint velocity to the linear/angular velocity of

$p_{\text{peg\_end}}$, the force/torque $f_{\text{estimated}} \in R^6$ can be estimated as

$$f_{\text{estimated}} = \text{argmin}_f |J^T_{\text{peg\_end}} f - \tau_{\text{external}}|^2 \tag{5.1}$$

where τ_{external} is the measured external joint torque. Here we assume the gravity has already been compensated.

5.3.4 Generalization w.r.t Global Rigid Transformation

Suppose we want to re-target the peg-hole insertion policy to the hole at a different location. Intuitively, this re-targeting is essentially a rigid transformation of the manipulation policy. Can we somehow “apply" this transformation directly?

In our framework, both the agent input (oriented keypoints and force/torque w.r.t keypoints) and output (linear/angular velocity or force/torque of an oriented keypoint) are expressed in 3D space. In other words, we can apply a rigid transformation to both the agent input and output. This property provides generalization w.r.t the global rigid transformation. Before feeding the input to the agent, we can transform the input from the world frame to some “nominal frame”. After agent computation, we can transform its output back to the world frame. The “nominal frame” can be chosen arbitrary, for instance in the peg-hole insertion task we can align it with the initial configuration of $p_{\text{hole\_top}}$. Thus, the global rigid transformation is transparent to the agent.

In contrast, many existing contributions [31, 33, 77, 154] work with input/output in joint space and/or image space (raw image or 2D keypoint), on which a rigid transformation cannot be applied. Thus, re-targeting agents in these methods to new initial object configurations and camera positions might need re-training.

5.3.5 Joint Space Control

The agent output is the desired linear/angular velocity or force/torque of an oriented keypoint p. An important observation is: if we assume the object is rigid and grasp is tight (grasped object is static w.r.t the gripper), then the object can be regarded as part of the

robot, thus standard joint-space controllers can be used map these agent outputs to joint-space commands. This generalizes the “object attachment" in the context of collision-free manipulation planning [24].

With this observation, we discuss several possible implementations of joint-space controller according to the robot interface. Let J_p be the Jacobian the maps the joint velocity $\dot{q}$ to the linear/angular velocity of p. One straightforward method to transform the commanded velocity v_p into joint space velocity command $\dot{q}_{\text{desired}}$ is

$$\dot{q}_{\text{desired}} = \text{argmin}_{\dot{q}} |J_p \dot{q} - v_p|^2 + \text{reg}(q) \tag{5.2}$$

where $\text{reg}(\dot{q})$ is a regularizer term. If the robot driver accepts joint velocity commands, we can send $\dot{q}_{\text{desired}}$ to the robot directly. Similarly, the desired force/torque F_p can also be transformed into joint space using J_p by

$$\tau_{\text{desired}} = J_p^T F_p + g \tag{5.3}$$

where g is the gravitational force in joint space. Here we ignore the inertia and Coriolis force of the robot.

Since standard joint-space controllers can map the agent output to joint commands, some more sophisticated controllers can also be used and might provide better tracking performance, for instance the task-space impedance controller described in [98]. The detailed discussion is omitted as they are out of the scope of this paper.

5.4 Pick-and-Place Manipulation

In many applications, the objects are randomly placed initially, potentially in a clutter. For this scenario, we use a two step manipulation scheme: the robot first perform a kinematic pick-and-place to move the object to some desired initial configuration (for instance moving the peg right above the hole), then the closed-loop policy discussed in Sec. 5.3 starts from that initial configuration. To make the entire manipulation operation generalizable, this pick-and-place step should also works for a category of objects. kPAM in Chapter 3 is

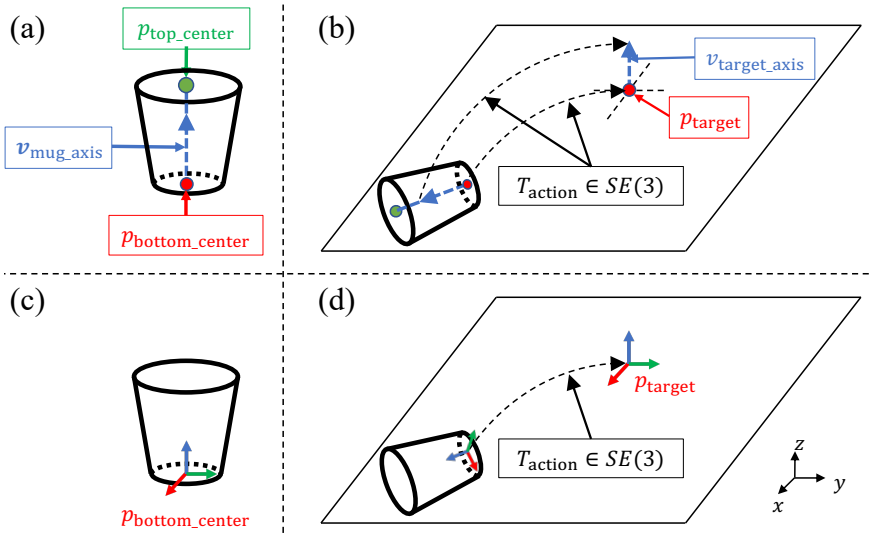


Figure 5-6: **Overview of kPAM [88] for category-level pick-and-place and its simplification with oriented keypoint**. kPAM is used as a pre-step for the closed-loop policy in our framework. Using the original mug demo of kPAM as an example: (a) In kPAM the object is represented by a set of semantic 3D keypoints. (b) The rigid transformation T_{action}, which represents the robot pick-and-place action, is solved to move $p_{\text{bottom\_center}}$ to the target location p_{target} and align the mug axis with the target direction $v_{\text{target\_axis}}$. (c) In this paper we propose to add orientation information to the $p_{\text{bottom\_center}}$ keypoint. (d) The desired configuration of the mug can be encoded as the target configuration of $p_{\text{bottom\_center}}$, and T_{action} is the relative transform between $p_{\text{bottom\_center}}$ and its target configuration. Note that this formulation also generalizes to mugs with different shape, size and topology. Please refer to Sec. 5.4 for more details.

proposed for this task and used as a sub-step. Here we briefly review it and show how the orientation information of the keypoint can be used to significantly simplify the kPAM framework, as illustrated in Fig. 5-6.

In kPAM, each object is represented by a set of semantic 3D keypoints $p \in R^{3\times N}$. For instance, we can represent the mug by two keypoint $p_{\text{top\_center}}$ and $p_{\text{bottom\_center}}$, as shown in Fig. 5-6 (a). To place the mug upright at some target location p_{target} , we need to plan a robot action T_{action} such that

$$||T_{\text{action}} p_{\text{bottom\_center}} - p_{\text{target}}|| = 0 \tag{5.4}$$

$$||\text{rotation}(T_{\text{action}}) v_{\text{mug\_axis}} - (0,0,1)^T|| = 0 \tag{5.5}$$

$$\text{where: } v_{\text{mug\_axis}} = \text{normalized}(p_{\text{top\_center}} - p_{\text{bottom\_center}}) \tag{5.6}$$

Alternatively, we can add orientation information to the $p_{\text{bottom\_center}}$ keypoint as shown in Fig. 5-6 (c): the z axis of $p_{\text{bottom\_center}}$ is aligned with $v_{\text{mug\_axis}}$ in Eq. (4.6), while the x axis of $p_{\text{bottom\_center}}$ is chosen randomly since the mug is symmetric. Then, the target configuration of the mug can be represented as a target configuration of $p_{\text{bottom\_center}}$, as shown in Fig. 5-6 (d). The robot pick-and-place action T_{action} is the relative transformation between $p_{\text{bottom\_center}}$ and its target configuration. Note that this formulation also generalizes to mugs with different shape, size and topology.

By adding the orientation information to keypoint, in many applications (for example all the demos in kPAM) we can avoid setting up costs/constraints and solving an optimization problem to find T_{action}, although for complex object it might be beneficial to represent it with many keypoints (with or without orientation) and define object target configuration using costs/constraints similar to Eq. (4.4) and (4.5).

5.5 Perception Implementation

In this section, we discuss the implementation details of keypoint perception used in our experiment. We stress that our framework is not restricted to this perception implementation, and we will discuss the trade off with an alternative.

We use robot kinematics to track the oriented keypoint in real-time and feed it as the input to the closed-loop agent. Suppose we know the oriented keypoint relative to the robot gripper, then when robot moves we can compute the oriented keypoint in the world frame using the forward kinematics. Here we assume the object is rigid and the grasp is tight (no relative motion between the object and the gripper), which is well satisfied in our experiment.

We use a robot wrist-mounted camera to perform the "single-shot" visual perception: After moving the camera to a suitable location, we capture an input image and perform object detection, keypoint detection and grasp planning using the method and neural networks described in Chapter 3 (see Appendix for more details). The raw output of the keypoint network we used is 3D keypoints (not oriented), and the orientation (the axis of the coordinate) is computed with the relative direction of two 3D keypoints and heuristics. Given visual

perception results, we execute robot grasping and compute the keypoints expressed in the robot gripper frame, using the keypoints in the world frame (from camera perception) and the robot gripper pose (from robot kinematics). After grasping, we use robot kinematics for real-time keypoint tracking and feed the result into the closed-loop agent, as mentioned above. Other objects (and keypoints on these objects) are assumed static. Benefit from the automatically keypoint annotation pipeline using multiview consistency in kPAM, it only takes four hours to annotate the training data for all experiments.

An alternative perception implementation would be using external cameras for realtime keypoint tracking. Compared with it, our method is more robust to occlusion, since our kinematics-based perception can tolerate complete occlusion while no vision-based tracker can. Besides, it is also more robust to symmetric objects. Consider a round peg rotating relative to its axis, it is very hard for visual perception to detect this motion while the perception with robot kinematics still works.

On the other hand, the object might move relative to the gripper when the external force on the object is large (although we haven't observe it in our experiment), and external cameras can resolve this issue. Furthermore, using external cameras in our framework can potentially benefit from end-to-end learning, if a neural network is used as the agent similar to [33]. Since our main focus is the overall manipulation formulation and its generalization, the exploration of external cameras and these trade-offs are out of our scope.

5.6 Results

We prototype our framework on a hardware robot and demonstrate a variety of contact-rich manipulation tasks. The particular novelty of these demonstrations is that our method is able to handle objects with large intra-category shape variations without any instance-wise tuning. The overview of the experiment is shown in Fig. 5-1. The video demo is available on this link.

We use a 7-DOF Kuka IIWA arm mounted with a Schunk WSG 50 parallel jaw gripper and a RGBD sensor (Primesense Carmine 1.09). More details regarding hardware setup is provided in Appendix. The torque measured by the Kuka IIWA arm is used to compute the

force/torque measurement as the input to the agent, as shown in Sec. 5.3.3.

5.6.1 Task Description

Whiteboard wiping: The robot must detect the whiteboard eraser, pick it up and use it to erase a small whiteboard, as shown in Fig. 5-1 (a). We use two oriented keypoints for this task: p_{front} and p_{center} as shown in Fig. 5-7 (a). For a successful wiping, the x-y plane trajectory of p_{front} should be aligned with the edge of the whiteboard, while the z axis force on p_{center} must be regulated to ensure the eraser is in contact with the whiteboard. We set the nominal z axis force to be 10 [N] and implement the agent as the hybrid force/motion controller [112] to control the force on z axis and position on other dimensions. The robot needs to deal with whiteboard erasers with significant shapes and sizes variation.

Peg-hole insertion: The robot must detect the peg and hole, pick up the peg and insert it into the hole. We use three groups of objects: 1) 3D-printed pegs and holes with 0.2 [mm] clearance, as shown in Fig. 5-1 (b); 2) LEGO blocks as shown in Fig. 5-1 (c); 3) The USB drive and ports in Fig. 5-1 (d). The same code is used for all three object groups, and we trained three networks for the detection of printed peg-holes, LEGO blocks and USB ports, respectively. The impedance controller in [95] is used as the agent.

Due to the graspability limitation of the USB drive, we pre-fix it to the robot gripper, while the USB port as the “hole" is detected from the visual perception.

5.6.2 Experimental Result

The failure rates of our method are summarized in Table 5.1. For the wiping task, we mark a trial as failure if 1) the discrepancy between p_{front} with the whiteboard edge is larger than 2 [cm]; or 2) the z axis force on p_{center} is less than 5 [N] (the nominal value is 10 [N]). For the peg-hole insertion task, we mark a trial as a failure if the peg is not inserted into the hole. Our method is first compared with an open-loop baseline: for wiping task the open-loop policy simply replays the trajectory of p_{front}, for the peg-hole insertion the open-loop policy always commands a downward motion. This baseline is similar to the policy in kPAM [88], Form2Fit [149] and KETO [109]. Our method has a much lower failure rate,

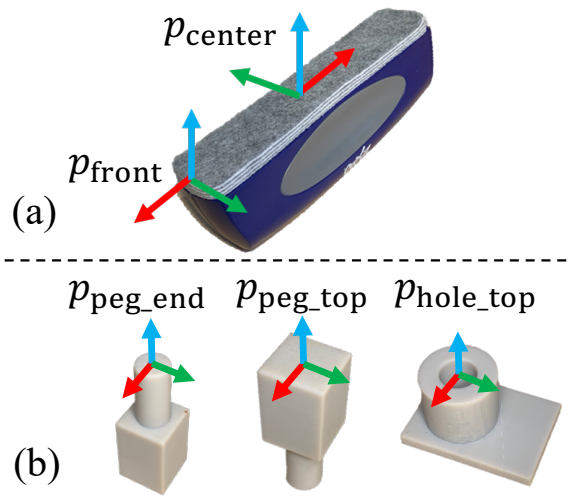


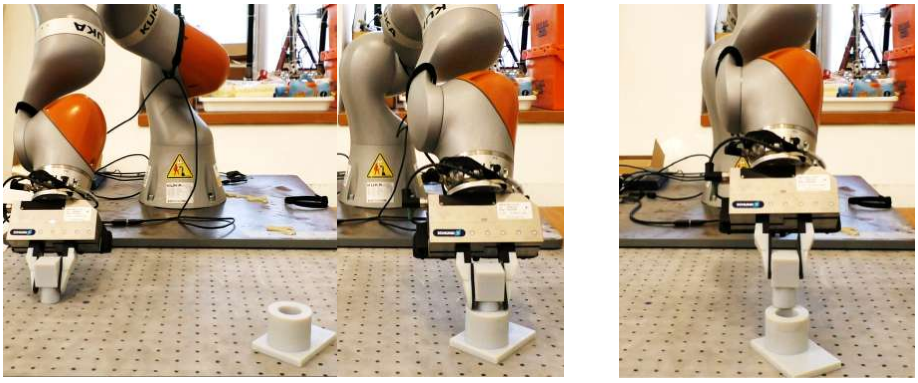
Figure 5-7: **The specified keypoints used in our experiment**. (a) Two keypoints are detected for the whiteboard wiping task. (b) Two keypoints are detected for the peg, one keypoint is detected for the hole. For the manipulation of LEGO blocks in Fig. 5-1 (c), the $p_{\text{peg\_top}}$ for another LEGO block is used as $p_{\text{hole\_top}}$. Please refer to Sec. 5.6.1 for more details.

as shown in Table. 5.1.

For the wiping task, it is crucial to measure and regulate the contact force in a closed-loop manner else the eraser would not touch the whiteboard. Thus, an open-loop policy typically cannot successfully erase texts on the whiteboard.

For the peg-hole insertion task, the typical accuracy of visual keypoint detection is about 5 [mm] when the distance between the object and camera is about 80 [cm]. The perception error is much larger than the clearance (0.2 [mm] for printed pegs and holes, almost zero for LEGO blocks and USB ports), which requires the agent to correct itself using closed-loop feedback with the measured keypoints and force/torque. For the printed pegs and holes, the agent can tolerate about 5-10 [mm] visual perception error, thus the failure rate is decent and much lower than the open-loop kinematic policy. However, if the perception error is too large the feedback agent wouldn't be able to correct it, as shown in Fig. 5-8 (b). The LEGO blocks have large chamfers, which makes the insertion much easier.

On the other hand, the USB port is much more demanding on the perception accuracy (roughly 3 [mm] error on the shorter side of USB would result in failure). We use a two-



(a) (b)

Figure 5-8: **Typical failure modes**. (a) Grasp failure. (b) The keypoint detection error is too large such that the closed-loop agent can't correct it with feedback.

Table 5.1: #Failure/#Trial (Failure Rate) Comparison

Task	Our Method	Open-Loop Baseline
Whiteboard wiping	1/25 (4%)	16/20 (80%)
Printed peg-hole	12/45 (26%)	19/20 (95%)
LEGO block	2/25 (8%)	7/20 (35%)
USB port	9/20 (45%)	20/20 (100%)

step perception scheme: the first coarse step roughly locates the object; then we move the wrist-mount camera closer to the object and perform the second, more accurate perception. In this way, we can reduce the perception error to 2 [mm].

To demonstrate the superiority of our keypoint formulation over pose-based methods, we implement a pose estimator and test it on our setups. The pose estimator is the same as the baseline (Fig. 4) in kPAM [38]: first initialize the alignment with detected keypoints, then perform ICP fitting between the observed point cloud and geometric template to get the 6-DOF pose. As demonstrated in kPAM (Fig. 5 of kPAM), a valid 6-DOF pose trajectory for one object can lead to physical infeasibility for another instance. For this reason, many trials would fail kinematically and usually require manual safety stop, as summarized in Table 5.2. The failure rate is much higher than our approach.

Table 5.2: Summary for Pose-Based Baseline

Task	Kinematic Failure Rate	Note on Failure
Whiteboard wiping	10/25 (40%)	Eraser collide into table; Eraser not aligned with whiteboard edge
Printed peg-hole	25/45 (55%)	Peg collide into hole; No overlapping between peg and hole (alignment error too large)
LEGO block	9/25 (36%)	
USB port	17/20 (85%)	

5.7 Conclusion

In this chapter, we explore generalizable, perception-to-action robotic manipulation with closed-loop feedback for contact-rich tasks. We adopt the 3D keypoint representation proposed for pick-and-place, and extend it to closed-loop policies with contact-rich tasks. We first augment keypoints with local orientation information, and propose a novel object-centric action space as the linear/angular velocity or force/torque of an oriented keypoint. Using this object and action representation, we present a new manipulation formulation that incorporates of the force/torque measurement, keypoint perception using visual and proprioceptive sensors, and the generalizable pick-and-place manipulation with oriented keypoint. Moreover, the pipeline is agnostic to the robot grasp pose and initial object configuration, which makes it flexible for integration and deployment. Extensive hardware experiments demonstrate our method can accomplish contact-rich manipulation tasks that require precision and dexterity for a category of objects with large intra-category shape variation.

Part II

Robustness Characterization

The manipulation pipelines described in Part I are based on the breakthrough of deep neural networks, which make human-level performance possible on narrow perception tasks (such as object recognition and keypoint detection). While these networks enable interesting perception-to-action manipulation behaviors, their vulnerability to adversarial attacks and distributional mismatches can lead to the failure of the entire manipulation pipeline. As the robot interacts with the world physically, failures can be dangerous and expensive. For example, we have broken the robot gripper finger twice during our development of the kPAM pipeline, despite the significant effort on software and hardware safety monitoring. Thus, we need to understand the risk of the system before deployment at scale (for instance the Amazon scale). This is our first motivation to study the robustness evaluation problem in Part II.

Our second motivation is that we also need to address the challenge of representation in robustness evaluation, similar to the pipeline development in Part I. The robot will be deployed in unstructured environments (such as kitchens with a diverse set of objects). Many factors of the environment would impact the robustness of the manipulator. Some of them have natural parameterizations, such as the camera pose and illumination condition. However, some other factors might be hard to represent. For instance, in Chapter 3 we have learned that it is surprisingly hard to define a parametric model of the object geometry that can capture the shape of "all possible mugs". Without this parametric model, we do not have a continuous parameter space as the input domain for robustness evaluation, which is a prerequisite for existing algorithms. In other words, we do not have a continuous input domain when investigating how does the mug geometry impact the robustness of the manipulation pipeline.

Initially, we plan to use the keypoints in Chapter 3 as the object geometric representation for robustness evaluation, as shown in Fig. 5-9. However this approach does not work well, because the keypoint representation loses geometric information, and the missing geometric information might impact the robustness. This inspires us to use a complete geometric representation, similar to Chapter 4. In particular, we can over-parameterize each object geometry instance as a voxel grid, which generalizes well to different objects. In this case, the challenge becomes how to represent the underlying *space* or *distribution* of

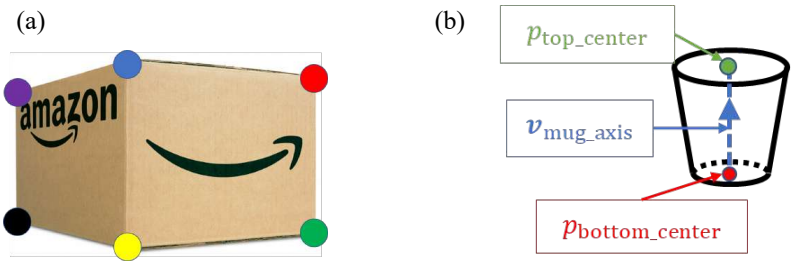


Figure 5-9: (a) Initially, we plan to use the keypoints in Chapter 3 as the object geometric representation for robustness evaluation. For example, we might use 8 corner keypoints to represent the geometry of a box. (b) However, one challenge with the keypoint representation is that it cannot capture the complete geometric information for complex the objects, such as the mug here. Furthermore, the missing geometric information might impact the robustness.

the object geometry. The space of "all possible mugs" is obviously a large space, but it is much smaller than the voxel space, as realistic object shapes only occupy a tiny portion of the voxel space. If we draw a random sample from the voxel space, it is very unlikely that this sample would look like a realistic object, and we should not expect the robot to handle it. Another prominent example is the textures of the object, that we might want to ensure a manipulator can handle "all textures of a mug". We can over-parameterize each object texture instance as an image (texture map), but the space of "all object textures" is again much smaller than the pixel space.

In Part II of this thesis, we aim to address this challenge and perform a robustness evaluation of the pipeline we developed. In Chapter 6, we present the preliminaries about the risk-based framework. In this framework, we prioritize finding the most likely failure modes and characterizing a system's safety by its probability of failure. Existing works in this risk-based framework assume continuous, easily parameterizable input domains. However as mentioned above, for our problems the robustness might depend on complex factors such as the object textures/shapes. To resolve this limitation, in Chapter 7 we formulate the robustness evaluation as failure search and failure rate estimation problems on a semi-empirical input distribution. To efficiently solve those problems, we proposed an auxiliary graph over those samples, which transforms the failure search/rare-event into search problems on a graph. In Chapter 8, we apply the resulting algorithm to the robustness evaluation of the robot manipulation that we develop. It is emphasized that although we use

the manipulation pipeline as an example, our formulation and algorithm can be applied to a variety of other autonomous systems.

Chapter 6

Preliminary: The Risk Based Framework

6.1 Introduction

In this chapter, we review the risk-based framework to evaluate the robustness of safety-critical autonomous systems. The review in this chapter assumes that a continuous, easily-parametrizable input domain is available for robustness evaluation. In the next chapters, we will discuss how to address more complex input domains (such as object textures and shape), which is necessary for the systems that we're interested in.

The failures of these safety-critical systems can be extremely costly, for instance the collision of autonomous cars. This necessitates moving beyond the standard metric of measuring the average performance, which is ubiquitous in evaluating "less critical" applications such as the face recognition. As such, several communities have adapted existing tools for testing safety-critical softwares to deep neural networks and autonomous systems built upon these networks.

One of these tools is the formal verification [51,64,83,110,145]. These methods attempt to find a proof of the "correctness" of an autonomous system. Let $g : X \to \{0,1\}$ be the binary specification function such that $g(x) = 0$ means the system behaves incorrectly. The goal of formal verification is to prove that the set $G = \{x : g(x) = 0\}$ is empty, in other words the system behaviors are always correct. However, there is a scalability limitation for these

formal verification algorithms. It is extremely difficult, if not impossible, to formally verify a modern deep network due to its size and complexity. Verifying a robot manipulator or autonomous driving system built upon deep networks can only be more challenging, since they involve other components. Moreover, formal verification algorithms usually require re-implementing the system in a formal language (such as Coq), which can be very difficult for complex systems.

As it is hard to guarantee the system behaviors are always correct, researchers [101, 122, 137, 144] have considered *probabilistic robustness evaluation*, which is a relaxation of the "always correct" specification. The underlying idea is that system is allowed to fail, but only at an extremely low probability. If the failure rate of a robot manipulator or autonomous vehicle is lower than the probability of an earthquake, then we should be confident in their safety. In this situation, we're using the failure probability as the *risk* of an autonomous system.

Real-world experiments are the most straightforward method for characterizing the system failure rate. However as argued in [101], real-world experiments have two major limitations: 1) testing these systems in the real world can be very dangerous; 2) it requires prohibitive amounts of time due to the rare nature of serious accidents. For example, a recent study [29] argues that autonomous cars need to drive "hundreds of millions of miles and, under some scenarios, hundreds of billions of miles to create enough data to demonstrate their safety." The challenge of real-world experiments motivates the use of simulation in robustness evaluation. Before deploying the system into the real world, we should first understand its failure mode and risk with simulated environments. The simulator avoids the danger of physical failure while allowing parallelized, faster-than-real-time evaluations. Furthermore, active control of the simulation environment allows us to prioritize the situation that the system is more likely to fail. On the other hand, a good simulator is required to ensure the simulated result transfers to the real world. As we're interested in systems built upon deep networks for visual perception, the simulator should contain a good renderer to produce input images to the systems' perception modules.

In this chapter, we discuss several existing contributions [101,122,137,144] about probabilistic robustness evaluation in simulated environments. In Sec. 6.2, we would consider

the detailed mathematical formulation of the risk framework. Due to the rare nature of failures, the robustness evaluation is formulated as *rare-event simulation* problems. In Sec. 6.3, we review the multi-level splitting algorithm which is used to solve rare-event simulation problems.

6.2 Mathematical Formulation

Mathematically, it is assumed a risk function $r : X \to R$ that measures the safety of an input $x \in X$, so that high value of $r(x)$ corresponds to dangerous situations. The input domain X collects all factors that affect the risk of the agent. Existing works usually assume X is a subset of R^n with rather simple geometry, such as the pixel-space disturbance for an image classifier [64] or the car initial position for a driving policy [101]. Given this risk function, the problems that we would like to solve are:

Failure Search (Falsification): Find a collection of $x_1, ...x_k$ such that

$$r(x_i) \geq r_{\text{threshold}}, \text{ where } 1 \leq i \leq k \tag{6.1}$$

where $r_{\text{threshold}}$ is a threshold for the risk.

Failure Rate Estimation: Compute the failure rate

$$p_{\text{failure}} = P_{p_x}[r(\mathscr{X}) \geq r_{\text{threshold}}] \tag{6.2}$$

where $\mathscr{X}$ is a random variable in X with $p_x(\cdot)$ as the prior distribution.

The failure examples found in Problem. 6.1 would be used to improve the system robustness. If the system is a neural network, we can add the failure examples into the training set and re-train the network. The failure rate estimated in Problem. 6.2 is a characterization of system safety. We should ensure the failure rate is extremely low before the actual deployment. Solving Problem. 6.1 and 6.2 requires the following three components:

Simulator: Because evaluating a safety-critical algorithm in the real world is slow and dangerous, we require the ability to simulate in a virtual world. To evaluate the risk func-

tion $f(x)$, we need to perform a simulation of the system under input condition x and check whether the system behaves correctly. In other words, the evaluation of f requires simulation of the system, which can be computationally expensive.

Input space and prior distribution: We need to collect factors that affect the system robustness and use it as the input domain X. We also need a prior distribution p_x with which we can sample from and evaluate the density. This is easy for factors with natural parameterizations, for instance [101] evaluates an autonomous driving pipeline with respect to car initial position/velocity. However this can be challenging for other factors, for instance we might want to evaluate an autonomous car with respect to the space/distribution of "all possible human clothing". We contribute a novel formulation to address it in Chapter 7.

Search algorithm: With the simulator and prior distribution, one straightforward method to solve Problem. 6.1 and 6.2 is brute-force simulation. We can simulate many times and collect the failure simulation from them for the failure search Problem. 6.1. Similarly, we can estimate the failure rate by the Naive Monte Carlo algorithm: perform many simulations and count how many of them exceed the risk threshold. However when the system is robust (the failure rate is low), finding failure cases must be regarded as a *rare-event simulation* and naive algorithms can be extremely inefficient. Thus, existing works propose to use the multi-level splitting algorithm, which is discussed in Sec. 6.3.

6.3 Multi-Level Splitting Algorithm

In this section, we review the multi-level splitting algorithm for the failure rate estimation in Problem. (6.2). We start with the simplest form, which requires manually specifying the levels $r_0, r_1, ..., r_k$, where $r_0 = -\infty$, $r_k = r_{\text{threshold}}$ and $r_{i+1} > r_i$. The failure rate in Eq. (6.2) can be factored as

$$p_{\text{failure}} = P_{p_x}[r(\mathscr{X}) \geq r_k] \tag{6.3}$$

$$= (\Pi_{i=0}^{k-1} P_{p_x}[r(\mathscr{X}) \geq r_{i+1} | r(\mathscr{X}) \geq r_i]) P_{p_x}[r(\mathscr{X}) \geq r_0] \tag{6.4}$$

$$= \Pi_{i=0}^{k-1} P_{p_x}[r(\mathscr{X}) \geq r_{i+1} | r(\mathscr{X}) \geq r_i] \tag{6.5}$$

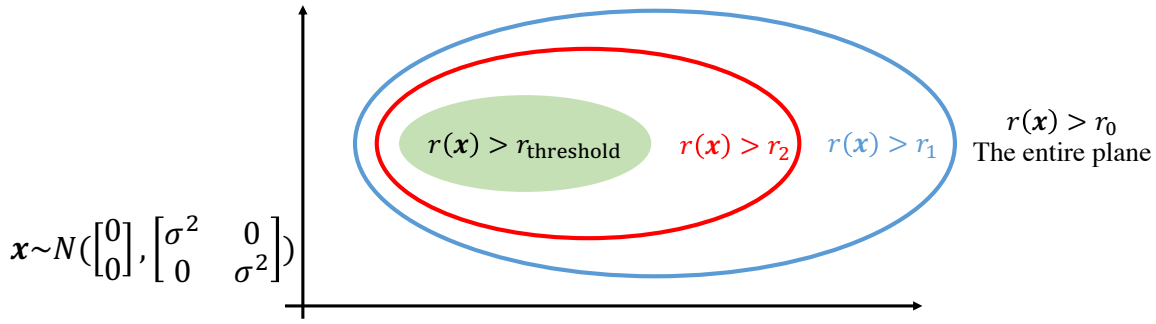


Figure 6-1: A 2-dimensional illustration of the multi-level splitting algorithm. The goal is to compute the probability of the green region with a 2D prior distribution, for instance a Gaussian. This can be hard for the naive Monte-Carlo algorithm. Thus, the multi-level splitting algorithm constructs a list of larger regions that eventually converges to the target region, for instance the red and blue ones. In this way, we can factor the probability of the green region as the product of a series of conditional probability in Eq. (6.3), which is the probability to reach the next smaller region conditioned on drawing a sample in the current region. These conditional probability terms are easier to estimate.

where levels should be selected such that the probability $P_{p_x}[r(\mathscr{X}) \geq r_{i+1} | r(\mathscr{X}) \geq r_i]$ is sufficient large. A 2-dimensional illustration is provided in Fig. 6-1. In this figure, the goal is to compute the probability of the green region, which can be hard for the naive Monte-Carlo algorithm. Thus, the algorithm constructs a list of larger regions that eventually converges to the target region. The $P_{p_x}[r(\mathscr{X}) \geq r_{i+1} | r(\mathscr{X}) \geq r_i]$ term in Eq. (6.3) is the probability to reach the next region $i+1$ conditioned on drawing a sample in region i.

If we can draw i.i.d samples from each of the distribution $p_i(x) \propto p_x(x)\mathbf{1}(r(x) \geq r_i)$, then we can estimate the conditional probability $P_{p_x}[r(\mathscr{X}) \geq r_{i+1} | r(\mathscr{X}) \geq r_i]$ using naive Monte-Carlo estimation (counting how many of the i.i.d samples have a risk larger than r_{i+1}). Thus, we need algorithms that can efficiently draw samples from the un-normalized distribution $p_i(x) \propto p_x(x)\mathbf{1}(r(x) \geq r_i)$. There are two main categories of algorithms for sampling from $p_i(x) \propto p_x(x)\mathbf{1}(r(x) \geq r_i)$: the cross-entropy (CE) algorithm and Markov Chain Monte Carlo (MCMC). Both of them have been applied to the robustness evaluation of neural networks or autonomous agents [101, 137]. Here we would focus on the MCMC algorithm, while the discussion about CE can be founded in [101].

The MCMC algorithm is summarized in Algorithm 1. The underlying idea is to construct a Markov chain whose stationary distribution is the one that we would like to sample

Algorithm 1: MCMC algorithm to sample from $p_x(x)\mathbf{1}(r(x) \geq r_i)$

Result: A sample from $p_x(x)\mathbf{1}(r(x) \geq r_i)$
Input: initial point x_0; proposal distribution $g(\cdot|\cdot)$;
Set $t = 0$;
while $t < iteration\_limit$ **do**
  Generate a random candidate x' according to $g(\cdot|x_t)$;
  Compute the acceptance rate $A(x', x_t) = \min(1, \frac{p_x(x')\mathbf{1}(r(x')\geq r_i)g(x'|x_t)}{p_x(x_t)\mathbf{1}(r(x_t)\geq r_i)g(x_t|x')})$;
  Generate a uniform random number $u \in [0, 1]$;
  if $u \leq A(x', x_t)$ **then**
    Set $x_{t+1} = x'$;
  else
    Set $x_{t+1} = x_t$;
  end
  Set $t = t + 1$;
end
Return x_t as the result;

from. Given an initial point x_0, we iteratively simulate the Markov chain. In each iteration, we first generate a candidate x'. Then, we would decide whether we should accept the transition to the candidate or reject it. This is achieved by computing an accept rate A which needs to satisfy the detailed balance condition [137]. Once this condition is satisfied, the stationary distribution of this Markov chain would be $p_i(x) \propto p_x(x)\mathbf{1}(r(x) \geq r_i)$. As a result, the current state x_t of the Markov chain can be regarded as a sample from $p_i(x) \propto p_x(x)\mathbf{1}(r(x) \geq r_i)$, if we simulate this Markov chain for an infinity number of iterations. This cannot be achieved in practice, thus we simulate the Markov chain until mixing.

The multi-level splitting algorithm built upon MCMC is summarized in Algorithm 2. We would maintain a set of samples $X_t = \{x_1^t, ...x_{N_p}^t\}$ for each iteration t, where particles in X_t have a risk larger than r_t. The initial sample set X_0 is drawn from the prior distribution $p_x(\cdot)$. In each iteration t, we first compute $P_{p_x}[r(\mathscr{X}) \geq r_t | r(\mathscr{X}) \geq r_{t-1}]$ by counting how many samples in X_{t-1} have a risk larger than r_t. Then we would perform many MCMC simulates to obtain the set X_t. For each MCMC simulation, we use an initial point sampled from X_t', which is the subset of X_{t-1} with risk larger than r_t.

In practice, however, it is better to choose the levels online instead of specifying them

Algorithm 2: Multi-level splitting algorithm to estimate $P_{p_x}[r(\mathscr{X}) \geq r_{\text{threshold}}]$

Result: An estimate of $P_{p_x}[r(\mathscr{X}) \geq r_{\text{threshold}}]$
Input: a prior distribution $p_x(\cdot)$; levels $r_1, .., r_K$; particle number N_p;
Generate N_p samples $X_0 = \{x_1^0, ...x_{N_p}^0\}$ from $p_x(\cdot)$;
Set $t = 1, p = 1$;
while $t \leq K$ **do**
  Select particles from X_{t-1} with risk larger than r_t, let them be $X_t' = \{x_1^{t-1}, ...x_{N_t}^{t-1}\}$;
  $N_t \leftarrow \text{sizeof}(X_t')$;
  $p \leftarrow p \times N_t / N_p$;
  $X_t \leftarrow \{\}$;
  for *i in range(N_p)* **do**
    Sample a random point x_i' from X_t';
    Run MCMC simulation with risk level r_t and initial point x_i', obtain a sample x_i;
    Append x_i into X_t.
  end
  Set $t = t + 1$;
end
Return p as the result;

manually, because the manual specification of levels might lead to no particles in X_{t-1} has a risk larger than r_t ("early kill"). Thus, we choose the level r_t online from the particle set X_{t-1} as the α-quantile of the particle risks in X_{t-1}. In this way, there are always particles that remain in X_t' with risks larger than the threshold r_t.

To sample from $p_x(x)\mathbf{1}(r(x) \geq r_i)$ in the MCMC algorithm (Algorithm. 1), we need a proposal distribution $g(\cdot|\cdot)$ which provides hints about where to search with the current x_k. Usually $g(\cdot|\cdot)$ is a Gaussian distribution with x_k as the center (mean), which intuitively exploits the locality (continuity): searching in the vicinity of the high-risk particle x_k would be more likely to yield another high-risk x'. Some works [122] shape the covariance of the Gaussian or exploit the gradient of the risk r, while they still exploited the locality. This is the reason why MCMC is more efficient than the naive Monte-Carlo. If we use a uniform distribution in X as the proposal distribution $g(\cdot|\cdot)$, the algorithm falls back to naive Monte-Carlo. An 2D illustration of MCMC algorithm with Gaussian proposal distribution is shown in Fig 6-2. Intuitively, we perform Gaussian random walk initialized at x_0 and constrained to the region defined by $r(x) \geq r_i$. If the random walk is performed for a long

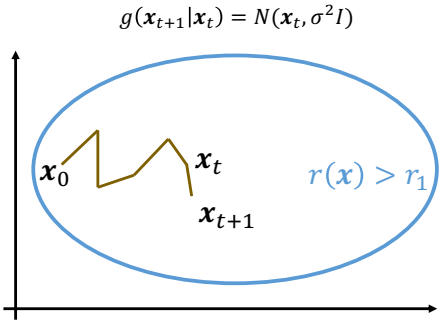


Figure 6-2: 2-dimensional illustration of the MCMC simulation in Algorithm. 1 with Gaussian proposal distribution. Intuitively, we perform Gaussian random walk initialized at x_0 and constrained to the region defined by $r(x) \geq r_i$. If the random walk is performed for a long time, then the distribution of x_t would be the target distribution (thus not dependent on the initial point x_0).

time, then the distribution of x_t would be the target distribution (thus not dependent on the initial point x_0).

In the following text, we will explore the robustness evaluation with challenging input domains (such as object textures and shapes). This locality idea plays an important role in our problem formulation and algorithm development.

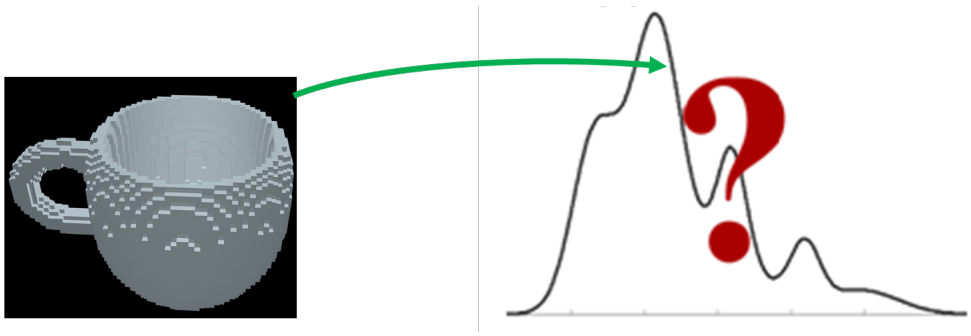
Chapter 7

Robustness Evaluation using Semi-Empirical Distributions

7.1 Introduction

In Chapter 6, we review existing contributions [101, 122, 137, 144] on the risk-based framework. These works assume a continuous, easily-parameterizable input domain as the search space for robustness evaluation. For example, [137] evaluates the robustness of image classification networks with respect to the pixel-wise disturbance. However, this assumption can be invalid when evaluating a robot manipulator, which is an autonomous system deployed in unstructured working environments. Many factors of the environment would impact the robustness. Some of them have natural representations (parameterizations), such as the camera pose and illumination condition. However, some other factors are more challenging, such as the geometry of the object. As mentioned in Sec. 3.4, it is surprisingly hard to define a parametric model of the object geometry that can capture the shape of "all possible mugs". As a result, we do not have a continuous input domain when investigating how does the object geometry impact the robustness. In this chapter, we address this challenge and answer how to represent the object geometry for robustness evaluation.

In Chapter 3, we use keypoints as the object geometry representation for developing the manipulation pipeline. However, keypoints are not suitable for robustness evaluation: the keypoint representation loses geometric information, and the missing geometric informa-



(a) Object Shape Instance by Voxel Representation

(b) Probability Density Function in Voxel Space

Figure 7-1: To keep all the geometric information for robustness evaluation, we propose to over-parameterize each object shape instance as a voxel grid, as shown in (a). However, with this voxel representation the challenge becomes how to represent the object geometry *distribution*, since realistic object shapes only occupy a very small portion of the voxel space. Thus, we might need a probabilistic density function in the voxel space which gives high probability to realistic objects, as shown in (b).

tion might impact the robustness. This inspires us to over-parameterize each object shape instance as a voxel grid to keep all the information, as shown in Fig. 7-1 (a). However, with this voxel representation, the challenge becomes how to represent the object geometry *distribution*. Realistic object shapes only occupy a very small portion of the voxel space. If we draw a random sample from the voxel space, it is very unlikely that this sample would look like a realistic object, and we should not expect the robot manipulator to handle it. Thus, we might need a probabilistic density function in the voxel space which gives high probability to realistic objects, as shown in Fig. 7-1 (b). Similarly, we can over-parameterize each object texture instance as an image (texture map), but it is challenging to represent the object texture distribution.

One method to address these hard distributions is to use generative models (such as GANs [41]). These generative models learn generators that map simple distributions in the feature space to those texture/shape distributions. With this generative model, we can apply existing algorithms in Chapter 6 to the continuous feature space. However, these generative models can yield unrealistic samples (see [84] for a detailed study), although many of the generated results are almost indistinguishable from authentic ones. As a result, we can discover unrealistic failure cases that would never be encountered in the real world, and this phenomenon can be exploited by the attacking algorithms. An experimental comparison is

made in Sec. 7.5.

In this chapter, we propose to directly materialize these texture/shape distributions as empirical distributions (sets of offline-collected samples). Thus, we model factors that affect the system robustness as a structured distribution over variables (e.g. the camera pose), combined with an empirical distribution, that describe the visual properties. We then formulate robustness evaluation as failure search and failure rate estimation problems on this combined distribution. Compared to the generative model formulation, our method will not produce arbitrarily unrealistic failure examples. One major challenge of this representation is the lack of continuity structures among the discrete samples. To address this, we formulate a weighted graph over the empirical dataset using the distance in a learned latent space as the edge weights. This graph structure connects discrete samples and transforms the search/rare-event problems into more efficient graph-based exploration. Moreover, failure rate estimation with the proposed graph converges to the ground-truth asymptotically, despite the using of learned features in the graph representation. We've performed a proof-of-concept experiment on an MNIST image classifier. In the next chapter, we will apply it to the robustness evaluation of the manipulation pipeline.

7.2 Related Work

7.2.1 Adversarial Examples

The investigation about neural network robustness is generally motivated by the fact that the neural network might fail on particular adversarial inputs, usually in surprising ways [129]. Thus, many contributions aim to discover and characterize those adversarial examples, as reviewed in Sec. 2.3.

Existing works on adversarial examples are typically restricted to continuous, easily parameterizable input domains. However, many factors with a significant impact on robustness might not have natural parameterizations. Some works [124] address this problem by learning a generative model and performing attacks in the learned feature distribution, which is usually a Gaussian. However, this might lead to discovering un-realistic input that

might never be encountered in the real world. Consequently, if we compute the failure rate with those learned distributions the estimation can be incorrect.

7.2.2 Formal Methods

Formal methods consider whether a neural network or autonomous system can ever violate a safety specification. They either return a proof that there is no such violation or a counterexample. Researchers have formulated the formal verification of neural networks as a satisfiability [64, 150] problem. To improve the efficiency, many works proposed various approximate solutions [55, 107, 146] in this framework. These problems are usually NP-hard [64] and many of those approaches require special network structure [64, 83]. As a result, scaling them to common image-based neural networks is not easy. Furthermore, when these verification algorithms yield a counterexample, it doesn't produce how likely will this counterexample occurs. Thus, it is hard to assess the robustness unless a proof of no violation is given.

7.2.3 Failure Rate Estimation by Rare-Event Simulation

An alternative strategy of robustness evaluation is to characterize the risk of neural networks (or systems with networks inside) as the probability of failure, under some prior distributions of the environments. For a well-implemented neural network the failure rate can be very small, thus we can view it as a rare-event simulation problem. Rare-event simulation has a long history and has made a major impact on various domains, please refer to [59] for a detailed study. In the past years, researchers also have applied rare-event algorithms, such as the cross-entropy method and MCMC simulation, to the robustness evaluation of neural networks and autonomous systems [101, 122, 137, 144]. These works typically assume a continuous, easily parameterizable input domain. In this work, we aim at more challenging inputs such as "the space/distribution of all possible human clothing", where a natural parameterization is not available.

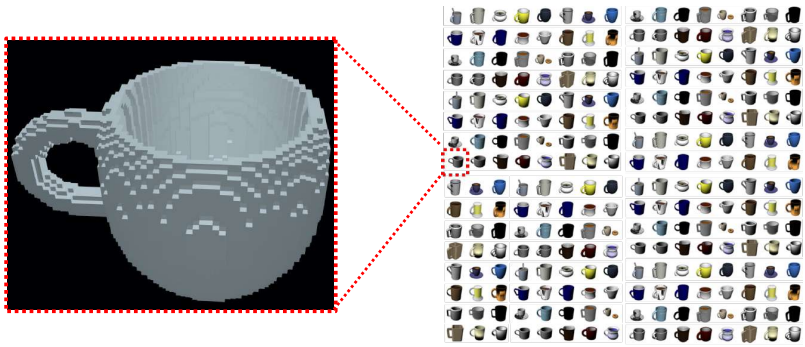


Figure 7-2: We propose to over-parameterize each object shape instance as a voxel grid and approximate the object shape distribution as an empirical distribution. For example, we would use a set of offline-collected mug shapes to represent the distribution of "all possible mugs that a manipulator can encounter".

7.2.4 Robust Training

Many contributions [17, 81, 145] aim to train an agent that is robust to particular types of adversarial attacks. The improvement of robustness is either observed empirically [81, 82] or guaranteed theoretically [17, 145]. These robust training approaches focus on the training phase, typically requiring alternating the training protocol such as loss function or data augmentation techniques. On the other hand, this paper aims at evaluating a well-trained network (agent). These two types of works complement each other.

7.3 Problem Formulation

To incorporate more complex factors such as the object texture/shape distributions into the robustness evaluation, we adopt the risk-based framework in Chapter 6. We assume a risk function $r : X \times Y \to R$ that measures the safety of an input (x, y). The input is divided into two parts. $x \in X$ concatenates all inputs with natural, continuous parameterizations. We can think of X as a subset of R^n with simple geometry, and it is easy to define a structured prior distribution on X. For instance, X might include the camera pose when evaluating a robot manipulator.

On the other hand, y collects complex factors such as object texture and shape. As mentioned in Sec. 7.1, it is challenging to define prior distributions of those factors. For example, we can over-parameterize each object shape instance as a voxel grid, but the

distribution of realistic object shapes is not easy as it only occupies a small portion of the voxel space. If we draw a random sample from the voxel space, it is very unlikely that this sample would look like a realistic object, and we should not expect the robot manipulator to handle it. Similarly, it is challenging to define the object texture distribution, which only occupies a small portion of the pixel space.

To address this challenge, we propose to approximate prior distributions in the Y space by empirical distributions. Practically, we would collect a set of M samples $\{y_1, ..., y_M\}$ as a discrete approximation of the Y space. Then, the prior distribution in Y space is approximated by an empirical distribution. For instance, we would use a set of offline-collected mug shapes to represent the distribution of "all possible mugs that a manipulator can encounter", as shown in Fig. 7-2. Due to the complexity of the object shape/texture distributions, our sample-based approximation might have a very high cardinality. Without loss of generality, we focus on the uniform distribution $p_y(y = y_i) = 1/M$. Using the sample-based representation, the problems that we would like to study become:

Failure Search: Find a collection of $(x_1, y_1), ...(x_k, y_k)$ pairs such that

$$r(x_i, y_i) \geq r_{\text{threshold}}, \text{ where } x_i \in X \text{ and } y_i \in \{y_1, ..., y_M\} \tag{7.1}$$

Failure Rate Estimation: Compute the failure rate

$$p_{\text{failure}} = P_{p_{xy}}[r(\mathscr{X}, \mathscr{Y}) \geq r_{\text{threshold}}] \tag{7.2}$$

$$\text{where: } p_{xy}(\cdot) = p_x(\cdot)p_y(\cdot) \tag{7.3}$$

$$p_y(y = y_i) = 1/M \text{ for } y_i \in \{y_1, ..., y_M\} \tag{7.4}$$

The main advantage of this formulation is the interpretability: we know that $\{y_1, ..., y_M\}$ are all authentic, thus our method will not produce arbitrarily unrealistic failure examples. On the other hand, using samples to represent the distribution preserves all the available information, as we can only access those distributions by sampling.

This sample-based representation is very similar to the usual test set, thus it is natural to ask "can we iterate through $\{y_1, ..., y_M\}$ like a test set?" For example, we can compute

(a) (b) (c)

Figure 7-3: The authenticity of the generated sample from a GAN generator is not guaranteed and they can be unrealistic. If we perform feature-space interpolation between two realistic samples from a GAN (such as (a) and (c) in the figure), then we might get a unrealistic sample in the middle (such as (b)). This can be exploited by the robustness evaluation algorithm, as detailed in Sec. 7.3.1.

$P_{p_{xy}}[r(\mathscr{X},\mathscr{Y})]$ in Eq. (7.2) as $\Sigma_i P[r(\mathscr{X},y_i) \geq r_{\text{threshold}}]/M$. The problem is that for each particular y_i, we must search over the X space or compute $P[r(\mathscr{X},y_i) \geq r_{\text{threshold}}]$. When the evaluated agent is robust, both of them can be hard and expensive. For example, in our experiments we need about 10^8 network forward evaluation for a y_i. This can lead to a substantial computational burden when M is large.

7.3.1 Comparison with Generative Model Representation

An alternative approach, which we compare our method against, would be to materialize the distribution in the Y space using generative models, such as GAN [41] or VAE [25]. Mathematically, we would train a generator $G : R^{n_z} \to Y$ that maps from the feature space R^{n_z} to Y.The training requires a collection of $y_1,...,y_k$ from the space Y, which can be regarded as samples from a prior distribution $p_y(\cdot)$ over the space Y. These generative models are trained to map a simple distribution in the feature space $p_z(\cdot)$ to the prior distribution $p_y(\cdot)$. For instance, many works on image/shape synthesis [10, 153] train GANs that map feature space Gaussian distributions to image/shape distributions. If we have trained such a generative model, our problems can be transformed into

Failure Search: Find a collection of $(x_1,z_1),...(x_k,z_k)$ with $(x_i,z_i) \in (X \times R^{n_z})$, such that

$$r_{\text{GAN}}(x_i,z_i) = r(x_i,G(z_i)) \geq r_{\text{threshold}}, \text{ where } 1 \leq i \leq k \tag{7.5}$$

where $r_{\text{threshold}}$ is a threshold for the risk.

Failure Rate Estimation: Compute the failure rate

$$p_{\text{failure}} = P_{p_{xz}}[r_{\text{GAN}}(\mathscr{X}, \mathscr{Z}) \geq r_{\text{threshold}}] \tag{7.6}$$

where $\mathscr{X}, \mathscr{Z}$ are random variables in $X \times R^{n_z}$ with $p_{xz}(\cdot) = p_x(\cdot)p_z(\cdot)$ as the prior distribution. $p_z(\cdot)$ is usually the feature space Guassian distribution.

The above problems are defined in continuous input domains, thus we can exploit algorithms in Chapter 6 to solve them. However, the authenticity of the generated sample $y = G(z)$ is not guaranteed and they can be unrealistic, even for well-trained models such as [10]. If we perform feature-space interpolation between two realistic samples from a GAN (such as Fig. 7-3 (a) and (c)), then we might get a unrealistic sample in the middle (such as Fig. 7-3 (b)).

This is actually consistent with the incredible success of generative models [10, 153] yielding images that are almost authentic. When we use a generative model in a robustness evaluation algorithm, we are actively searching this model for adversarial cases; to be useful, the generative model should almost never produce a bad example. We compare our method against the generative model approach in Sec. 7.5.1.

7.4 Graph Structure in the Discrete Samples

In this section, we discuss the algorithm to solve the failure search and failure rate estimation in Eq. (7.1) and (7.2). We will start with the discussion of the failure search problem for clarity.

In Sec. 7.3 we propose sample-based representation as $\{y_1, ..., y_M\}$. One issue associated with this representation is that we lose all the structure. To be more specific, the risk evaluated using y_1 provides no hint about the risk we expect for y_2, since they are just two samples without explicit correlation. This is in contrast to the continuous-space search in Eq. (6.1). In these continuous setups, the risk of a particular $x_1 \in X$ provides a hint about the risk in the vicinity of x_1. If we sample another x_2 in the vicinity of a high-risk x_1, then

Algorithm 3: Failure search based on graph representation

Input: initial y_0; continuous space search algorithm; neighbourhood $N(\cdot)$ for each $y \in Y$;
Set $t = 0$; $visited = \emptyset$; $result = \emptyset$;
while $t <$ *iteration_limit* **do**
  if $visited = \emptyset$ **then**
    Set $y_{\text{next}} = y_0$;
  else
    Select a y' from *visited* for expansion according to the risk;
    Select a y_{next} from $N(y')$;
  end
  Invoke the continuous space algorithm on y_{next}, which produces:
    1) $x_1, ..., x_m$ with $r(x_j, y_{\text{next}}) \geq r_{\text{threshold}}$;
    2) the risk $r_{y_{\text{next}}}$ of y_{next}, for example $r_{y_{\text{next}}} = \max_x r(x, y_{\text{next}})$;
  Append $(x_1, y_{\text{next}}), ..., (x_m, y_{\text{next}})$ to *result*;
  Append $(y_{\text{next}}, r_{y_{\text{next}}})$ to *visited*;
  Set $t = t + 1$;
end

it would be more likely that x_2 also has a high risk. This heuristic is also critical to the continuous space rare-event simulation algorithm in Sec. 6.3 with the MCMC simulation in Algorithm 1. In MCMC, the proposal distribution $g(\cdot|\cdot)$ is typically selected as a Gaussian distribution with the current high risk x_k as the center, which encourages the search in the vicinity of x_k.

Inspired by this heuristic, we proposed to exploit the locality in the large sample set $\{y_1, ..., y_M\}$. Suppose we have a distance function $D : Y \times Y \to R$ that measures the "similarity" of two points in Y. Then we can exploit locality in the failure search problem; if we have found y_1 with a high risk, then points that are "similar" to y_1 are also likely to have high risk values, where "similar" means small distance according to D. The risk is only defined on (x, y) pairs, and we can define the risk for y from that. For instance, we might define a "maximum risk" as $r_{\max}(y) = \max_x r(x, y)$.

We propose to represent D using a simple Euclidean distance in a learned latent "feature-space". In particular, we would train an encoder $E : Y \to R^{n_z}$ that maps the Y space into a n_z-dimensional feature space. Let $z_1, ..., z_M$ be the encoded feature vector of $y_1, ..., y_M$. Then, the distance D between two y_i and y_j is the L_2 distance between z_i and z_j. Using

Algorithm 4: Multi-level splitting algorithm for $P_{p_{xy}(\cdot)}[r(\mathscr{X},\mathscr{Y}) \geq r_{\text{threshold}}]$

Result: An estimate of $P_{p_{xy}(\cdot)}[r(\mathscr{X},\mathscr{Y}) \geq r_{\text{threshold}}]$
Input: a prior distribution $p_{xy}(\cdot)$; levels $r_1,..,r_K$; particle number N_p;
Generate N_p samples $X_0 = \{(x_1^0,y_1^0),...(x_{N_p}^0,y_{N_p}^0)\}$ from $p_{xy}(\cdot)$;
Set $t = 1, p = 1$;
while $t \leq K$ **do**
  Select particles from X_{t-1} with risk larger than r_t, let them be $X_t' = \{(x_1^{t-1},y_1^{t-1}),...(x_{N_t}^{t-1},y_{N_t}^{t-1})\}$;
  $p \leftarrow p \times N_t/N_p$;
  $X_t \leftarrow \{\}$;
  for *i in range(N_p)* **do**
    Sample a random point (x_i',y_i') from X_t';
    Run MCMC simulation with risk level r_t and initial point (x_i',y_i'), obtain a sample (x_i,y_i);
    Append (x_i,y_i) into X_t.
  end
  Set $t = t+1$;
end
Return p as the result;

this feature space distance, we would build a nearest neighbour graph over Y using existing nearest-neighbour algorithms. In this graph, each y_i would maintain a neighbourhood set $N(y_i) \subset \{y_1,...,y_M\}$, which collects points that are "similiar" to y_i.

With this graph, the failure search problem is transformed into a graph search problem, as summarized in Algorithm (3). We need a vertex risk function to characterize the risk of y_i, which will be used to determine whether this y_i should be expanded for searching in its vicinity. We can use the maximum risk $r_{\max}(y) = \max_x r(x,y)$ or mean risk $r_{\max}(y) = \mathbf{E}_{\mathscr{X}\sim p_x} r(\mathscr{X},y)$. A continuous space algorithm is required to find $x_1,...,x_m$ and compute the vertex risk function. This requires searching over the X space and can be very expensive, especially when the evaluated agent is robust.

7.4.1 Graph-based Rare-Event Simulation

The graph structure mentioned above can also be used for rare-event simulation in Problem (7.2). We will use the multi-level splitting algorithm described in Chapter 6. However, for the problem that we are interested in, the search space is the product space

$X \times \{y_1, ..., y_M\}$. Thus, each particle in the multi-level splitting will be a (x, y) pair, as shown in Algorithm 4.

Algorithm 5: MCMC to sample from $p_x(x)p_y(y)\mathbf{1}(r(x,y) \geq r_i)$

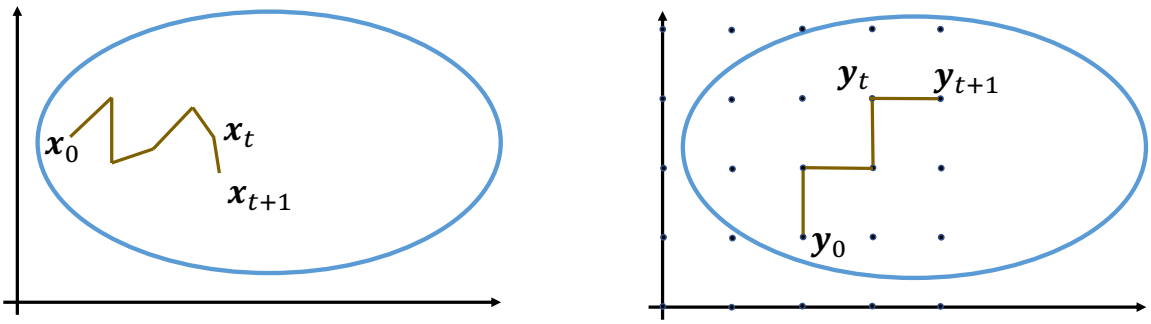
Result: A sample from $p_x(x)p_y(y)\mathbf{1}(r(x,y) \geq r_i)$
Input: initial point x_0, y_0; proposal distribution $g_{xy}(\cdot|\cdot) = g_x(\cdot|\cdot)g_y(\cdot|\cdot)$;
Set $t = 0$;
while $t < iteration\_limit$ **do**
  Generate a random candidate x', y' according to $g(\cdot|x_t, y_t)$;
  Compute the acceptance rate $A = \min(1, \frac{p_x(x')p_y(y')\mathbf{1}(r(x',y') \geq r_i)g_x(x'|x_t)g_y(y'|y_t)}{p_x(x_t)p_y(y_t)\mathbf{1}(r(x_t,y_t) \geq r_i)g_x(x_t|x')g_y(y_t|y')})$;
  Generate a uniform random number $u \in [0, 1]$;
  if $u \leq A$ **then**
    Set $x_{t+1} = x', y_{t+t} = y'$;
  else
    Set $x_{t+1} = x_t, y_{t+1} = y_t$;
  end
  Set $t = t + 1$;
end
Return x_t, y_t as the result;

The multi-level splitting in Algorithm 4 requires MCMC simulation internally, and we need a proposal distribution $g_{xy}(\cdot|\cdot)$ for MCMC in order to draw samples from $p_x(x)p_y(y)\mathbf{1}(r(x,y) \geq r_i)$. This graph can also be used to construct this proposal distribution $g_{xy}(\cdot|\cdot)$. The MCMC algorithm for (X, Y) space with the graph-based proposal distribution is summarized in Algorithm 5. We need a $g_y(y_j|y_i)$ which is the transition probability in space Y from y_i to y_j. We can define it as

$$g_y(y_j|y_i) = \begin{cases} 1/N(y_i), & \text{if} \quad y_i \in N(y_i) \\ 0, & \text{otherwise} \end{cases} \tag{7.7}$$

In our experiment, we use a proposal distribution that is a random switching between exploring on X and Y. This is similar to coordinate descent in an optimization algorithm.

This proposal distribution $g_y(\cdot|y_i)$ means we would search the neighbors of y_i with a uniform probability, which exploits the locality in the vicinity of y_i. We can also use other variants, for instance prioritizing points with smaller distances or blending Eq. (7.7) with a uniform distribution over Y. Moreover, it can be shown [59] that the MCMC simulation



(a) (b)

Figure 7-4: A 2-dimensional illustration of the graph-based MCMC simulation and its comparison with continuous MCMC in Chapter 6. As shown in (a), the continuous MCMC with a Gaussian proposal distribution is equivalent to Gaussian random walk contained in the region $r(x) > r_i$. Alternatively, we can discretize the 2d place as a grid, as shown in (b). This grid is a graph where each point has 4 neighbours (up, down, left right). We can perform MCMC on this graph with a proposal distribution that choose a neighbour point randomly, as described in Eq. (7.7).

converges to the target distribution given sufficient iterations for mixing, thus each level in the multi-level splitting algorithm converges asymptotically to the ground-truth. As a result, the failure rate estimation also converges asymptotically to the ground truth, despite the fact that we exploit learned features to provide hints about the locality.

A 2D illustration of the graph-based MCMC is shown in Fig. 7-4, with comparison to the continuous MCMC in Chapter 6. As shown in Fig. 7-4 (a), continuous MCMC with a Gaussian proposal distribution is equivalent to Gaussian random walk contained in the region $r(x) > r_i$. Alternatively, we can discretize the 2d place as a grid. This grid is a graph where each point has 4 neighbours (up, down, left right). We can perform MCMC on this grid, which might look like Fig. 7-4 (b) with a proposal distribution that choose a neighbour point randomly, as described in Eq. (7.7). In practice, we would perform MCMC on graphs embedded in high-dimensional feature spaces, which can be hard to visualize.

7.5 Proof-of-Concept Experiment on MNIST

We first evaluate our method on the MNIST image classification task. In particular, we train an image classifier and evaluate its robustness. The Y space consists of 10000 MNIST

images. The X space is the pixel-space disturbance with L_∞-norm constrained to be less than ρ. All MNIST images in the Y space are correctly classified without disturbance. We apply pixel-space disturbance as a data augmentation technique in the training of the neural network classifier, to make the network more robust to the attack and reduce the failure rate. The risk function $r : X \times Y \to R$ is the maximum score of incorrect labels minus the score of the correct label.

We evaluate both the failure search and failure rate estimation on this problem. For failure search, we use the continuous-space multi-level splitting described in Algorithm. 1 as the internal continuous space search in Algorithm 3. When using this continuous-space multi-level splitting on a particular $y_i \in Y$, it produces a set of $x_1, ..., x_m$ with risk larger than the threshold and a failure rate estimation $P[r(\mathscr{X}, y_i) > r_{\text{threshold}}]$. The failure search in Algorithm (3) requires a risk value for y_i, and we use the maximum risk over all the encountered x during the continuous-space multi-level splitting algorithm.

For both rare-event simulation and failure search, we need to build a graph for the Y space. We train a separate VAE [25] to extract the feature and build a nearest-neighbor graph on top of that. Our failure search and rare-event simulation are compared with baselines which use the random search in the Y space instead of the graph-based search. We use the same amount of computational resources for both our method and the baseline, measured in terms of the network forward evaluation counts. Note that our method is "black-box": we do not need to access the internal structure of the neural network or evaluate its gradient.

The experimental result is shown in Fig. 7-5. Each data point is computed from 10 independent runs. We tune the disturbance L_∞-norm radius ρ to control the failure rates, and compare the performance of the proposed method and baseline at different levels of failure rates. Fig. 7-5 (a) show the number of discovered high-risk images (y) in failure search with at least one x such that $r(x, y) > r_{\text{threshold}}$. From the figure, our method can discover more failure cases than the baseline. Fig. 7-5 (b) shows the relative variance comparison between our method and baseline on the failure rate estimation problem. Our method has a consistently lower variance. Some high-risk MNIST images discovered by our method are shown in Fig. 7-6 (a).

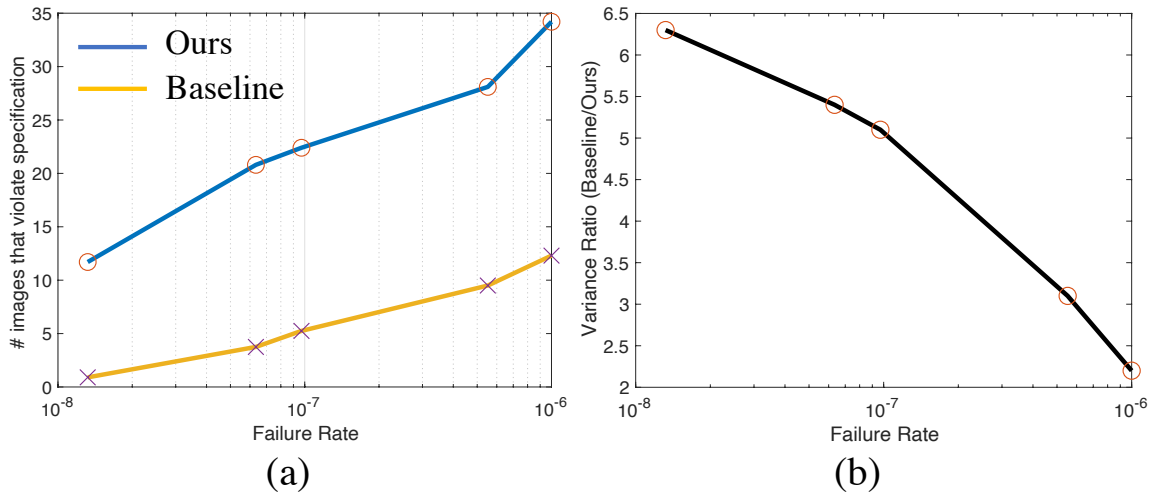


Figure 7-5: The failure search and failure rate estimation result on the MNIST experiment in Sec. 7.5. (a) Our method can discover more failure cases (the image y with at least one x such that $r(x,y) > r_{\text{threshold}}$) than the baseline. (b) Our method achieves a much smaller variance on the failure rate estimation problem than the baseline.

7.5.1 Comparison with Generative Model based Formulation

Our sample-based representation of the Y space is compared with the representation using generative models. In particular, our method is compared with the following two baselines:

GAN_1: We train a GAN [41] generator that maps the feature space R^{n_z} to image space, and use the continuous MCMC algorithm on $X \times R^{n_z}$. To evaluate the risk function we need the ground-truth label of the generated images. Thus, our GAN will also take the label as the inputs (in addition to the Gaussian noise). This is known as conditional GAN in existing work [93].

GAN_2: We use the same network as GAN_1. However, we weight each generated image by the normalized score from the GAN discriminator. In other words, we use the GAN discriminator score as an un-normalized prior distribution which encourages the search algorithm to find authentic MNIST samples. On the other hand, this leads to a roughly doubled computational burden due to the evaluation of the discriminator.

Almost all failure examples found by GAN_1 are not realistic. Examples of the discovered high-risk images is shown in Fig. 7-6. Fig. 7-6 (a) is the high-risk samples discovered by the proposed method, while (b) the result of the GAN_1 baseline. These images are not cherry-picked. The GAN_1 baseline tends to produce unrealistic failure cases, which is not

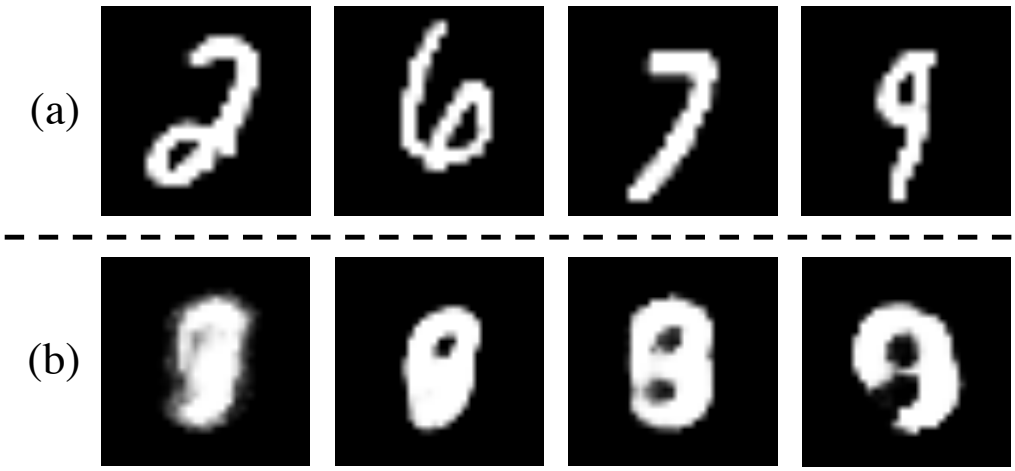


Figure 7-6: Examples of discovered failure cases for (a) the proposed method; and (b) the GAN_1 baseline in Sec. 7.5.1. The generative model baseline tends to produce un-realistic failure cases.

meaningful in practice.

The GAN_2 baseline is much stronger than the GAN_1 and some of the failure cases found by GAN_2 are realistic. To quantify this, we randomly select 20 failure images found by GAN_2 and ask 34 human subjects to label them as "Real", "Unknown" or "Fake". The result is shown in Table. 7.1. From the table, about 1/3 to 1/2 of the generated images are designated as authentic. Examples of the "Real", "Unknown" or "Fake" images are shown in Fig. 7-7. On the other hand, the GAN_2 baseline cannot guarantee the generated failure cases are realistic. Moreover, the failure rate estimation from GAN_2 is much larger than the result from our formulation, as shown in Fig. 7-8. This is because many failure images are not realistic, and these unrealistic failures can be designated with high risk. As a result, the failure rate estimation from the GAN baseline might not corresponds to the actual risk of the image classifier.

(a) (b) (c)

Figure 7-7: Examples of the high-risk images discovered by the GAN_2 baseline in Sec. 7.5.1. (a), (b) and (c) are examples of "Real", "Unkown" or "Fake" images. The quantitative result is in Table. 7.1.

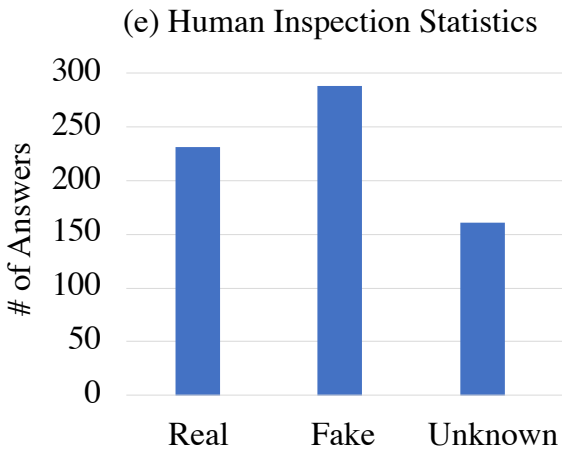


Table 7.1: Human inspection statistics of the failure images found by the GAN_2 baseline (34 human subjects, 20 failure images, 680 answers in total).

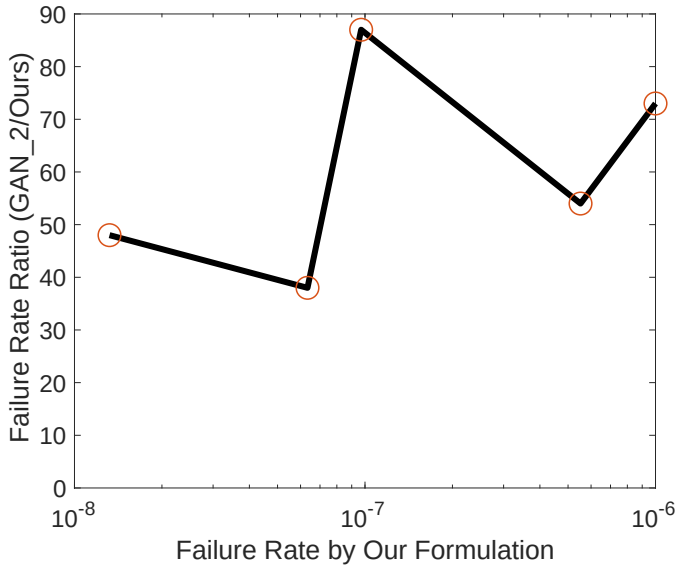


Figure 7-8: The GAN_2 baseline produce failure rates that is much larger than our formulation, this is because many failure images are not realistic and designated with high risk.

7.6 Conclusion

In this chapter, we study the robustness evaluation of complex systems with respect to challenging input domain such as the object shapes/textures. We propose to directly represent these spaces by samples, and formulate the robustness evaluation as failure search or rare-event simulation problems with respect to semi-empirical distributions. We further propose to build a graph over the samples, which transforms the failure search and failure rate estimation into graph search problems. Importantly, failure rate estimation by rare-event simulation with the proposed graph structure converges to ground-truth failure rate asymptotically, despite the graph exploits learned features.

Chapter 8

Application to a Robot Manipulation Pipeline

In this chapter, we perform a robustness evaluation of the kPAM manipulation pipeline that is presented in Chapter 3. Sec 8.1 focuses on evaluating the keypoint perception module, and Sec. 8.2 studies the failure rate of the entire pipeline. The robustness of the keypoint perception module depends on complex factors such as the object texture, which is addressed using the algorithm discussed in Chapter 7. The entire manipulation pipeline has some binary failure modes, which cannot be addressed in the risk-based framework. To resolve it, we propose a factorized verification scheme in Sec. 8.2.

8.1 Robustness Evaluation of Keypoint Perception

In this section, we conduct a probabilistic robustness evaluation of the keypoint detection network, which is the perception module of the kPAM robot manipulator pipeline in Chapter 3. The network takes input as raw RGBD images and produces a list of 3D keypoints expressed in the camera frame. After keypoint perception, the pipeline would perform kPAM action planning, grasp planning and action execution. These remaining modules do not involve deep neural networks, thus the robotics community has a better understanding of their failure modes. The robustness of the keypoint detector depends on the geometry and appearance (texture) of the object, which will be addressed using the failure search and

rare-event simulation algorithms discussed in Chapter 7.

The risk-based formulation requires a larger collection of samples as the underlying representation of the object texture/shape distributions. For example, if Amazon would like to understand the robustness of their box manipulation pipeline, then they should collect all possible boxes that they have ever encountered in production. This data collection procedure, however, is not aligned with our main focus of the algorithmic contribution, and the required engineering effort is out of our capability. Thus, we simplify the data collection by only investigating the robustness with respect to different object textures. In other words, we keep the geometry model fixed during the robustness evaluation. Furthermore, we use an open-source image dataset as the texture maps of the object, instead of collecting a large-scale object texture dataset. On the other hand, it is emphasized that the actual deployment of this algorithm requires the data collection as a modeling procedure. The failure samples and failure rate are meaningful only if the input distribution matches the real-world situations.

In Sec. 8.1.1 we discuss the detailed formulation and experimental setup. The evaluation pipeline requires an efficient render customized for texture switching, which is discussed Sec. 8.1.2. The evaluation result and comparison with baseline is presented in Sec. 8.1.3.

8.1.1 Experimental Setup

We formulate the robustness evaluation as the risk framework in Chapter 7. As mentioned above, we focus on the textures as our space Y. For the geometry, we use a simple mug model with only 80 vertices to ensure the efficiency of rendering, as shown in Fig. 8-1 (a). We use the 1000 MNIST images to represent the object textures distribution, an example is shown in Fig. 8-1 (b). The continuous X space collects a variety of physical parameters: the object pose, camera pose, object scale and illumination parameters. We assume the mug always lies on the table, and its pose has 4 dimensions (2 planar translations, the rotation about z axis and the mug axis). The camera pose is a 6-DoF rigid transformation, but it is constrained such that the object is within the camera field of view. To ensure this, we

(a) (b) (c)

Figure 8-1: The OpenGL based renderer used to evaluate the keypoint detection neural network. (c) shows the rendered mug image using the mug mesh model in (a) and texture image in (b). This renderer is optimized for texture switching, as detailed in Sec. 8.1.2.

would first sample to object pose, then sample to the camera position. To determine the camera orientation, we would sample a point on the table in the vicinity of the mug, and let the camera look at that point. Finally, the camera is allowed to rotate with respect to its own axis.

The risk function $r : X \times Y \to R$ is the L2 distance between the predicted keypoint and the ground-truth. To evaluate this risk function for a given input (x, y), we need to first render the camera image with the texture y and physical parameters x. Then, we would feed the rendered image into the keypoint detection network to obtain the prediction, and compare it with the ground truth. The ground-truth keypoint can be computed using the physical parameter x.

8.1.2 Implementation Details

The robustness evaluation pipeline requires a render that produces an RGBD image input given the texture $y \in Y$ and physical parameter $x \in X$. Many off-the-shelf renderers such as Unreal/Blender/Unity are available, and they can render high-quality photo-realistic images. However, our algorithm in Chapter 7 requires the random walk in the Y space, which means the object texture is changed at each iteration. These off-the-shelf renderers are not optimized for fast switching of textures (and object geometry). In contrast, they assume one texture is used for a lot of rendering and they perform many pre-computation steps (for example, the lighting map) for a given texture. As a result, these renderers are very inefficient for our application.

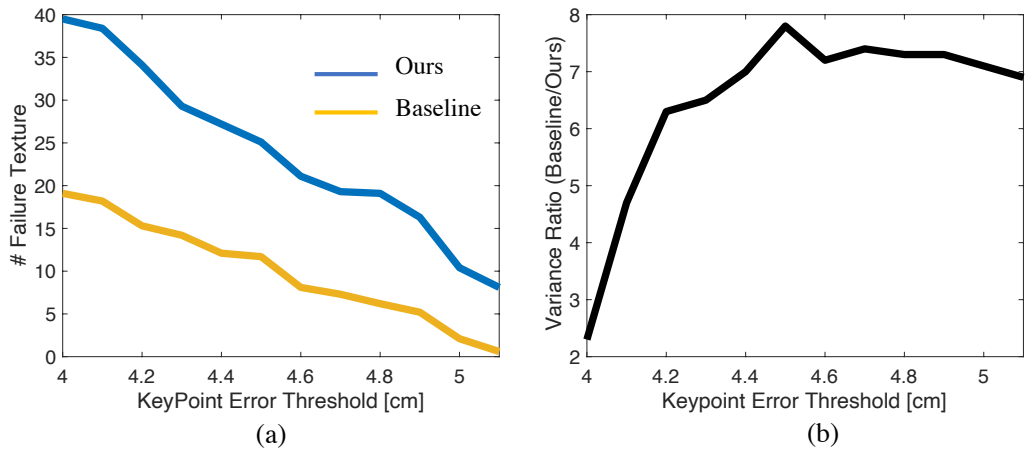


Figure 8-2: The failure search and failure rate estimation results on the keypoint perception experiment in Sec. 8.1.3. (a) Our method can discover more failure textures (the image y with at least one x such that $r(x,y) > r_{\text{threshold}}$) than the baseline. (b) Our method achieves a much smaller variance on the failure rate estimation than the baseline.

To resolve this issue, we use an OpenGL-based renderer customized to our problem. An example of the rendered images is shown in Fig. 8-1 (c). This renderer uses a simple phong-shading, and all the texture maps used to represent the Y space are cached in memory. As a result, we can achieve more than 600 [FPS] rendering on a Nvidia 2080 Ti GPU, despite the switching of object textures. On the other hand, the image produced by this simple renderer is not as realistic as off-the-shelf renderers such as Unreal/Blender. Implementing a high-quality renderer which supports fast texture switching is not aligned with our focus of the algorithmic contribution, and is left for future work.

The keypoint detection network we evaluated is adapted from the one on the robot (described in Sec. 3.5). To improve the efficiency, we reduce the input/output dimension, replace resnet34 backbone with a hand-written CNN, and use direct keypoint regression instead of the confidence map regression. As a result, the average accuracy decreases from 0.4 [cm] to 1.3 [cm], while the evaluation speed increases to 800 [FPS] from about 50 [FPS].

8.1.3 Results

We evaluate both failure search and rare-event estimation on the keypoint perception network. Our method is compared with a baseline that replaces the graph-based search with a

Table 8.1: Failure Rate Estimation Statistics

Threshold [cm]	3.5	4.0	4.5	5.0
Naive	$2.62e^{-4}$	$6.56e^{-5}$	N. A.	N. A.
Proposed	$(2.54 \pm 0.23)e^{-4}$	$(6.71 \pm 0.93)e^{-5}$	$(1.21 \pm 0.23)e^{-5}$	$(5.45 \pm 1.58)e^{-6}$
Baseline	$(2.68 \pm 0.19)e^{-4}$	$(6.84 \pm 1.34)e^{-5}$	$(1.58 \pm 0.60)e^{-5}$	$(6.59 \pm 4.17)e^{-6}$

random search. We compare our method with the baseline on different thresholds of keypoint errors. The algorithms for both failure search and rare-event estimation are the same as ones in Sec. 7.5.

Fig. 8-2 (a) shows the number of high-risk textures found by both algorithms, where a texture y is designated as high-risk if at least one $x \in X$ is found such that $r(x,y) \geq r_{\text{threshold}}$. From the figure, our method can found more failure textures than the baseline. At a large failure threshold of keypoint error, our method can found about 10x more failure textures than the baseline. Fig. 8-2 (b) shows the relative variance comparison between our method and baseline on failure rate estimation. Table. 8.1 shows the statistics of the failure rate estimation. For the "Naive" method in the table, we perform naive Monte-Carlo simulation until the number of discovered failure causes reach 100. For the proposed method and the baseline, we perform 10 individual runs and compute the empirical variance. From the figure and table, our method has a consistently lower variance than the baseline.

8.2 Robustness Evaluation of Manipulation Pipeline

In addition to evaluating the visual perception module, it is desirable to evaluate the robustness of the *whole system*, as argued in [101]. In particular, we want to estimate the failure rate of the whole system (instead of its sub-components) under the prior distribution in the joint $X \times Y$ space. The underlying motivation is that sub-components might interact with each other in a rather complex way. Verifying each component independently might not be able to capture the failure caused by this interaction.

The algorithms described in Chapter 6 and Chapter 7 are black-box. They do not require information about system internal structures other than a (black-box) risk function $r : X \times Y \to R$. Thus, theoretically we only need a new risk function to evaluate the robustness of

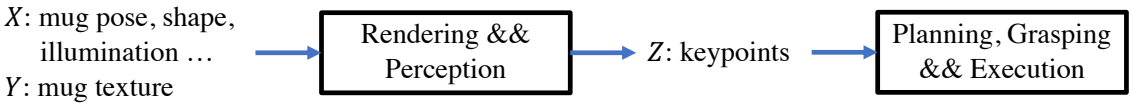


Figure 8-3: The robustness evaluation pipeline for the manipulation system. Given an input $(x, y) \in X \times Y$, we would render the image, perform object detection which produces keypoint $z \in Z$. Then, the kPAM action planning and execution are only dependent on the keypoint.

the entire manipulation pipeline. This is exactly what [101] proposed. They use the time-to-collision as the risk of an autonomous driving pipeline while treating the driving policy (which contains perception, planning and control modules) as a black box.

However, we cannot directly apply this technique to the manipulation pipeline, because some failure modes of a robot manipulator are intrinsically *binary*. Using the motion planners in Chapter 3 and 4 as an example. These planners would either find a valid robot trajectory or fail. We can easily obtain a binary success flag, but it is very hard to write down a continuous risk, which is necessary for the algorithm in Chapter 7 and 6.

To resolve this limitation, we exploit the staged structure of the manipulator pipeline and use a factorized, component-wise robustness evaluation pipeline. The visual perception is evaluated by the Multi-level splitting algorithm in Chapter 6, while other components are evaluated using naive Monte-Carlo simulation. On the other hand, we still want to get the failure rate of the *whole system*. This requires careful interfacing between the robustness evaluation of sub-components, as detailed below.

8.2.1 Component-Wise Verification with Whole-System Failure Rate

Consider the probabilistic robustness evaluation of the kPAM manipulation pipeline depicted in Fig. 8-3. The parameter space (X, Y) is the same as the one in Sec. 8.1.1. Given an input $(x, y) \in X \times Y$, we render a scene image, perform keypoint detection, run grasping and kPAM action planning. Then, we compare the placed keypoint location with the desired keypoint location. A manipulation attempt is designated as a failure if either 1) any sub-component fails, or 2) the placed keypoint location is too far away from the desired location.

The robot manipulation is a staged pipeline. We separate the pipeline into two parts, as shown in Fig 8-3. One of them is the visual perception, which will be evaluated using the multi-level splitting algorithm in Sec. 8.1. The other part includes all other components. As mentioned in Sec. 8.2, this part contains binary failure modes (such as the failure of the motion planer) and we would evaluate it using naive Monte-Carlo simulation. However, we still want to compute the failure rate of the entire pipeline under the prior distribution $p_{xy}(\cdot)$, which can be factored as

$$P_{p_{xy}(\cdot)}[\text{pipeline fails}] = P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception works}]P_{p_{xy}(\cdot)}[\text{perception works}] \tag{8.1}$$

$$+ P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception fails}]P_{p_{xy}(\cdot)}[\text{perception fails}] \tag{8.2}$$

here we need an auxiliary definition of "perception failure". We can define it as the L2 distance between perceived keypoints and ground-truth is greater than 4 [cm]. Among terms in Eq. 8.1, we have

$$P_{p_{xy}(\cdot)}[\text{perception works}] \approx 1 \text{ and } P_{p_{xy}(\cdot)}[\text{perception works}] \leq 1 \tag{8.3}$$

$$P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception fails}] \approx 1 \tag{8.4}$$

$$\text{and } P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception fails}] \leq 1 \tag{8.5}$$

Thus, we can approximate (and upper bound) $P_{p_{xy}(\cdot)}[\text{pipeline fails}]$ as

$$P_{p_{xy}(\cdot)}[\text{pipeline fails}] \leq P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception works}] \tag{8.6}$$

$$+ P_{p_{xy}(\cdot)}[\text{perception fails}] \tag{8.7}$$

where the term $P_{p_{xy}(\cdot)}[\text{perception fails}]$ has already been computed in Sec. 8.1. Thus, we only need to evaluate $P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception works}]$.

As shown in Fig. 8-3, after keypoint perception the remaining components are only dependent on the perceived keypoints. In other words, if we can sample from the unnormalized "correct" keypoint distribution $Q(z) = E_{p_{xy}(\cdot)}[p(z|x,y)\mathbf{1}(\text{keypoint perception works})]$, then the evaluation is independent of the rendering and neural network evaluation.

Because the keypoint perception is very likely to succeed, it is very easy to sample from this distribution. However, to avoid performing the rendering and neural network evaluation during the robustness evaluation of the planning && execution module, we would use a generative model to approximate this distribution. In particular, we would marginalize the y terms in $Q(z) = E_{p_{xy}(\cdot)}[p(z|x,y)\mathbf{1}(\text{keypoint perception works})]$ and learn a conditional distribution $P_{Z|X}(\cdot|X=x)$ using a generative model ($Q(z) = E_{p_x(\cdot)}[P_{Z|X}(\cdot|X=x)]$). This generative model only works with low-dimensional data instead of images, thus can be much more efficient. To compute $P_{p_{xy}(\cdot)}[\text{pipeline fails}|\text{perception works}]$, we would draw many samples from this generative model and test whether the pipeline works.

8.2.2 Experimental Results

We train a GAN to approximate the conditional keypoint distribution $P_{Z|X}(\cdot|X=x)$. The GAN network's inputs include the pose/shape of the mug and Gaussian white noise. The network produces the position of the keypoint relative to its ground-truth. In other words, the GAN network learns the error distribution of keypoints conditioned on the input x. This network requires training data in the form of (x,z) tuples. From the discussion above, we only need to approximate the keypoint distribution when the error is less than a threshold (4 [cm]). Thus, the generation of the training data can be performed very easily. Once the GAN network is trained, we can sample a (x,z) pair by first drawing a x sample from the prior distribution $p_x(\cdot)$, then sample z from the $P_{Z|X}(\cdot|X=x)$ distribution approximated by the GAN.

With the method to sample the (x,z) pairs, we can evaluate the planning && execution component of the pipeline, which only needs keypoints as the input. This component consists of: 1) planning a trajectory to reach the object; 2) computing a grasp pose; and 3) planning a trajectory to the desired configuration. The failure is defined as either 1) the

	Generative Model (Planning && Execution)	Baseline (Planning && Execution)	Perception
Failure Rate	0.00742	0.00678	6.71e-5

Table 8.2: The failure rate of the planning && execution module with/without a learned input keypoint distribution ("Generative"/"Baseline"). The incorporation of keypoint distribution leads to a higher failure rate estimation than the baseline. On the other hand, the failure rate is much larger than the failure rate of keypoint perception. Thus, the failure of the system is dominated by the failure of the planning && execution module.

	GAN	Non-Isotropic Gaussian	Isotropic Gaussain
Failure Rate	0.00742	0.00729	0.00717

Table 8.3: The failure rate of the planning && execution module with different generative models.

planner doesn't find a trajectory, or 2) the placement keypoint location is too far away from its goal (the threshold is 4 [cm]). We run the naive Monte-Carlo evaluation until 100 failure cases have been discovered.

The experimental result is summarized in Table 8.2. We compare it with a baseline that doesn't account for the keypoint distribution (in other words, the keypoint z is the ground-truth location directly computed from physical parameter x). From the table, the incorporation of keypoint distribution leads to a higher failure rate estimation than the baseline. This implies the baseline might underestimate the failure rate. On the other hand, we can see this failure rate is much larger than the failure rate of the keypoint perception. Thus, the failure of the whole system is dominated by the failure of the planning && execution module, which is consistent with our observation on the robot.

We also compare the GAN modeling of the keypoint error distribution with some other much simpler generative models. The result is shown in Table 8.3. For the non-isotropic Gaussian model, we assume the conditional keypoint error distribution does not depends on x and approximates the keypoint error distribution as a 3-dimensional Gaussian distribution. For the isotropic Gaussian model, we use three 1-dimensional Gaussian distributions to approximate the keypoint error distribution. From the table, we observe the difference is not noticeable. This might imply a very simple generative model can capture the keypoint error distribution.

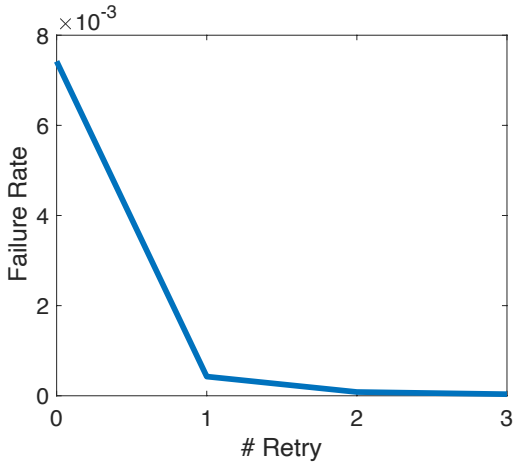


Figure 8-4: The failure rate of the planning && execution module with different numbers of retry.

To improve the robustness and reduce the failure rate, a simple method is to retry once the planning && execution module fails. Fig. 8-4 shows the failure rate with the increased number of retry. From the figure, with only one additional retry the failure rate is 17x smaller. With three retry attempts, the failure rate of the planning && execution module is $3.52e^{-5}$, which is comparable to the failure rate of the perception module. This implies the whole-system failure rate is about $1e^{-4}$ from Eq. (8.1).

Chapter 9

Discussion

This thesis tackles the challenge caused by the diverse, unstructured environments, in which the robot manipulator would be deployed. In this chapter, we first summarize the contributions and explain how the works in the preceding chapters fit together to address the overall question of this thesis. Then, we discuss the trade-off between various state representations proposed in this thesis and existing works. Finally, we close by proposing some directions for future work.

9.1 Summary of Contributions

This thesis approaches the challenge of the state representation and take steps toward a generalizable, dexterous and robust manipulation pipeline. We start by considering the example problem of cleaning up a kitchen, as mentioned in Chapter 1. The kitchen might contain a variety of different objects with potentially unknown instances, and the manipulator should automatically *generalize* to them. This is out of the capability of existing robot manipulators, as they can only handle known objects with detailed template models.

Initially, we plan to accomplish this kitchen cleanup task by pose estimation. The author is very familiar with this topic and has made several contributions [36, 37] on pose estimation algorithms. However, we run into several challenges such as the topological inconsistency and shape mismatches among different objects. This inspires us that the pose might not be an ideal object representation. In other words, it is surprisingly hard to

define a parametric model of the object geometry that can capture the shape of "all possible mugs". Due to the lack of this parametric model, we cannot perform parameter (pose) estimation.

To address this challenge, in kPAM (Chapter 3) we propose to use semantic 3D keypoints as an object representation. Using this object representation, we contribute a novel formulation of the generalizable pick-and-place manipulation that factors the problem into 1) 3D keypoint detection, 2) optimization-based robot action planning, and 3) 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. kPAM-SC (Chapter 4) extends kPAM with the reasoning of physical feasibility such as collision, static equilibrium, visibility and grasp stability. The keypoint representation is not sufficient for this task, as it lacks the dense and complete geometric information of the object. Thus, we propose to integrate 3D shape completion algorithms into the pipeline.

kPAM 2.0 (Chapter 5) improves the pipeline in terms of the *dexterity*. In particular, we aim at a manipulation framework that is capable of performing contact-rich tasks, while being generalizable to objects with different shapes, sizes, and appearances. To achieve this, we first augment the keypoint representation in Chapter 3 with local orientation information. Using this oriented keypoint, we propose a novel object-centric action representation as the linear/angular velocity or force/torque of an oriented keypoint. This action representation enables closed-loop policies and contact-rich tasks, despite intra-category shape and size variations of manipulated objects. We experimentally demonstrate our framework on several challenging contact-rich manipulation tasks.

Part II of this thesis focuses on the *robustness* of the manipulator. Our motivation to investigate the pipeline robustness is twofold: 1) cyber-physical systems such as manipulators should be highly reliable to avoid causing damage to humans and properties; 2) we also need to address the challenge of representation when considering the robustness, similar to the pipeline development. As mentioned above, the lack of a parametric model of the object geometries (such as "all possible mugs") motivates us to exploit keypoints to develop a capable manipulation pipeline. In the context of the robustness evaluation, the

lack of the parametric model implies we do not have a continuous parameter space as the input domain, which is a prerequisite for existing robustness evaluation algorithms. Given this challenge, a natural question to ask is "can we use the keypoint representation for robustness evaluation?" To be more specific, can we represent the mugs as a set of keypoints, and perform a robustness evaluation with respect to the input space of keypoints? However, this approach does not work, because the information contained in keypoints is not sufficient for robustness evaluation. Keypoints only characterize several local object parts, but the geometry information missed by keypoints might also affect the robustness.

It is hard to define concise, natural parameterizations for the object geometry, but it can be *over-parameterized*. For example, we can use voxel grids to represent each object shape instance, although the space of voxel grids is much larger than "all possible mugs". Similarly, we can use an image (texture map) to represent each object texture instance, although the pixel space is again much larger than "all possible mug textures". To evaluate the robustness of the manipulator with respect to "all possible mug textures", we need to use a subspace of the pixel space as the input domain. Thus, the question becomes how to represent the "all possible mug textures" as a sub-space of the pixel space?

In the Part II of this thesis, we proposed to approximate these complex spaces (distributions) as samples, as we can only access them by sampling. For example, we would represent "all possible mug textures" as a large collection of mug textures, which is a subset of the pixel space. Then, we model factors that affect the system robustness as a structured distribution over variables (e.g. the object pose), combined with an empirical distribution, that describe the visual properties. We then formulate the robustness evaluation as failure search and failure rate estimation problems on this combined distribution. One major challenge of this representation is the lack of continuity structures among the discrete samples. To address this, we formulate a weighted graph over the empirical dataset using the distance in a learned latent space as the edge weights. This graph structure connects discrete samples and transforms the search/rare-event problems into more efficient graph-based exploration. Our method is applied to the manipulation pipeline we developed, and it can potentially benefit many other cyber-physical systems such as the self-driving car

The contribution of this thesis is twofold. Conceptually, we address the challenge

caused by the unstructured environments and propose several object representations in the development and robustness evaluation of the manipulation pipeline. Systemically, we take steps toward a generalizable, dexterous and robust manipulator by building a capable manipulation pipeline and evaluate its robustness.

9.2 Environment Representation for Manipulation

The most important environment for a robot manipulator is obviously the objects. However, it is surprisingly hard to define a parametric model of the object geometry that can capture "all possible mugs". 6-DOF pose (or deformable pose) with geometric templates might be able to capture many mug instances, but it is not hard to find an "adversarial mug" which is hard to represent. Furthermore, a continuous parameterization of the object geometry cannot handle topology inconsistency, as detailed in Sec. 3.4.

This challenge prevails in both the development of the manipulation pipeline and its robustness evaluation. For pipeline development, the lack of a precise, complete parametric model implies we cannot perform parameter (pose) estimation. Regarding robustness evaluation, the lack the parametric model means we don not have a continuous input domain as the search space. As a result, researchers have proposed a variety of state (object) representations to address this challenge. In the following text, we would discuss some of them and make comparisons from several perspectives.

9.2.1 Information Contained in the Representation

The state representation can be regarded as an information extraction from the raw sensor data. The raw sensor input (such as the image or point cloud) contains all the information that is available, but it is not easy to directly use them. If we extract a state representation from the raw sensor input, for instance estimating the pose from the image input, then the information contained in the state (pose) might be easier to interpret and use. On the other hand, the information contained in the state might be incomplete. For example, if the object pose is used as the state, then we lose the appearance information of the object, although it is available from the image input.

Some researchers [77, 78] believe it is better to exploit all the information contained in the raw sensory input, as more information does not hurt. Thus, it is argued that the loss of information would potentially lead to performance deterioration. On the other hand, during our development of the manipulation pipeline we found that lots of information is not useful and *should be ignored*. Using the kPAM pipeline in Chapter 3 as an example: once the keypoint is detected, we won't need to know the appearance of the object anymore. In other words, *conditioned on* the keypoint information, the manipulation pipeline should be independent of the object texture. Thus, we ignore the object texture after keypoint detection, although this texture information is available in the RGBD image input. Actually, ignoring the task un-related information is the underlying reason why our pipeline can generalize to different objects and task setups.

The required information depends on the task. For example, we can typically ignore the object appearance during pipeline development. However, this is not true for robustness evaluation. Let's consider the keypoint perception module in our pipeline. Although theoretically the keypoint location should not depend on the object appearance, in practice the appearance would impact the robustness of neural networks. Thus, to evaluate the robustness of the manipulator we must consider how to represent the object appearance (texture) instead of simply ignoring it.

9.2.2 Learned Representation vs. Hand-Crafted Representation

The state representation in many data-driven algorithms [3,15,77,78] is not explicit defined. They use an implicit state instead, which is usually an internal feature of the neural network. These neural networks are typically trained using imitation learning [78] or reinforcement learning [118, 119] algorithms. The learning procedure requires training data in the form of human demonstrations and/or trial-and-error attempts.

One major advantage of these data-driven algorithms is that almost no prior knowledge is required. As a result, these algorithms can be applied to a variety of tasks in many different fields. As long as an efficient simulator or a large human-demonstration dataset is available, many data-driven algorithms [118, 119] can be easily used in a black-box way.

A lot of interesting manipulation behaviors automatically emerges from these data-driven algorithms [4, 130].

On the other hand, it is hard to interpret the implicit representation (a neural network internal feature). This is exactly why generative models (GANs) are not ideal in our investigation of robustness evaluation. Performing search in the learned feature space would lead to unrealistic failure cases and un-interpretable failure rate estimation. From the perspective of pipeline development, the learned feature makes it hard to incorporate prior knowledge. For example, in Chapter 5 we use keypoints to tell the policy which part of the object is relevant to the manipulation task, which part is not (for instance, the peg handle shape is not relevant to the peg-hole insertion task). It is hard to incorporate this prior into implicit state representations such as [4, 77].

9.2.3 Dense Representation vs. Sparse Representation

In this section, we review the dense descriptor proposed in Dense Object Net [34], which is a representative dense object representation. This representation will be compared with the sparse keypoint representation proposed in this thesis to highlight the trade-off.

In [33, 34], each pixel is assigned a descriptor by a neural network. Thus, the object representation contains *dense, global* visual information of the object. In contrast, the keypoint (oriented or not) proposed in this thesis only characterizes a set of *sparse, local* object parts that are relevant to the manipulation task. The pixel-wise descriptor in [34] is trained such that the L_2-distance in the feature spaces implies correspondence. [34] also develops a pipeline that trains the dense descriptor in a self-supervised way.

Dense object net [34] uses the learned descriptor to accomplish the "grasp a specific point of an object" manipulation task (such as the "pick-up a shoe at its heel" in [34]). They would first record a descriptor value for a particular shoe heel. Then given an input image, they would extract the dense descriptor, find the pixel (descriptor) that is closest to the recorded descriptor, and perform robot grasping at that pixel. We can think of the object representation in this task is actually the 2D keypoint (pixel) instead of the dense descriptor. The dense descriptor is an intermediate object representation which is transformed

into 2D keypoints eventually. A similar transformation is also exploited in a following-up work [33]. The neural network policy in [33] takes 2D keypoints as the input, where 2D keypoints are obtained from the dense descriptor.

The discussion above highlights two advantages of the sparse keypoint representation: 1) the dimension is much lower; 2) only task-relevant information (such as the "shoe heel" location) is contained in the keypoint representation. This motivates us to use keypoints directly for both pick-and-place and contact-rich manipulation. On the other hand, our method requires manual labeling of the training data (ground-truth keypoint locations, please refer to the Appendix for more details), while the network in [34] can be trained in a self-supervised way.

Dense object representation, such as the dense descriptor and complete object geometry, usually contains more information. However as mentioned in Sec. 9.2.1, many manipulation tasks are not dependent on this additional information. Moreover, the perception algorithm for dense object representation might be more challenging. On the other hand, this additional information can be helpful for some tasks. For example, in Chapter 4 we use the complete object geometry for the reasoning of physical feasibility.

9.3 Future Directions

Although this thesis has made progress toward a generalizable, dexterous and robust robot manipulation pipeline, much more remains to be done. Among them, the most important one is scaling up the system implementation. Both the robot manipulation and robustness evaluation experiments are still research prototypes. The manipulation pipeline only handles tens of the objects at a failure rate of 1%. This is not sufficient for deployment at scale (for instance the Amazon scale). The robustness evaluation experiment is limited to 1) an unrealistic renderer; 2) only considering the variation of object textures, and 3) using an open-source dataset to emulate the object textures. For the actual deployment, these limitations should be addressed.

Appendix A

Implementation Details of kPAM

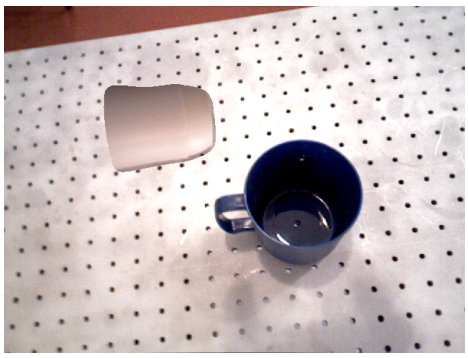
A.1 Dataset Generation and Annotation

In order to reduce the human annotation time required for dataset labelling we use a data collection pipeline similar to that used in [34]. The main idea is to collect many RGBD images of a static scene and perform a dense 3D reconstruction. Then, similarly to [90], 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.

A.1.1 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 [34]. Our data is made up of 3D reconstructions of a static scene. 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. Using the robot's forward kinematics we know the camera pose corresponding to each image which allows us to use TSDF fusion [19] 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 of [30]. In our particular case



(a) Mugs composite image

(b) Shoes composite image

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

all of 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.

A.1.2 Instance Segmentation

In order to train our instance segmentation network we require training images with pixelwise semantic labels. Using the background subtraction technique detailed in Section A.1.1 we have pixelwise labels for all the images in our 3D reconstructions. However, these images contain only a single object. We need our instance segmentation network to be able to handle multiple objects at test time. Similar to [120] we augment our training data by creating multi-object composite images from our single object annotated images. This is done by simply pasting the cropped part of the image corresponding to the object in question on top of an existing background. This process can be repeated to generate composite images with arbitrary numbers of object. Four our experiments we generated images with between 1 and 7 objects. Examples of such images are shown in Figure A-1.

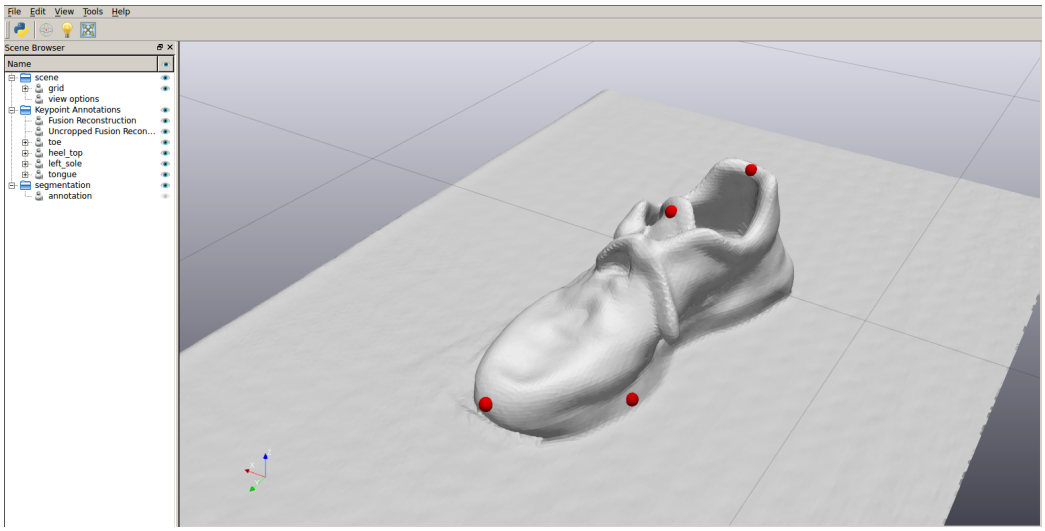


Figure A-2: A screenshot from our custom keypoint annotation tool based on [89] while annotating a shoe scene.

A.1.3 Keypoint Detection

We developed a custom labelling tool based on the Director [89] visualizer for annotating keypoint locations in the 3D reconstruction. Using this tool annotating a 3D reconstruction takes on the order of 1-2 minutes depending on the number of keypoints to be labelled (3 for mugs, 6 for the shoes). Once the 3D reconstruction is annotated we can backproject these labels into each of the 400 individual RGBD frames corresponding to that scene. The fact that we only have to annotate the 3D reconstruction vastly improves the efficiency of our data generation and labelling approach. As detailed in Figure 3-8 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.

A.2 Instance Segmentation Network

For the instance segmentation we used the official Mask R-CNN implementation [91]. We used a R-101-FPN backbone that was pretrained on imagenet. We then fine-tuned on a dataset of 10,000 images generated using the procedure outlined in Section A.1.2. The network was trained for 40,000 iterations using the default training schedule from the official implementatation [91].

A.3 Keypoint Detection Network

We modify the integral network [127] 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:

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

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

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

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 [127] 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 A.1.3.

Appendix B

Implementation Details of kPAM 2.0

B.1 Experiment Setup

We use a 7-DOF Kuka IIWA arm mounted with a 1-DOF Schunk WSG gripper and a Primesense RGBD sensor. Agents in our experiment use the desired keypoint linear/angular velocity as the action representation. The joint torque measured by the Kuka IIWA arm is used to compute the force/torque measurement as the input to the agent.

We use the two-step manipulation scheme for all the experiments: the robot first perform a kinematic pick-and-place to singulate the object and move it to some desired initial configuration (for instance move the peg right above the hole), then the closed-loop policy starts from that initial configuration. For the pick-and-place manipulation, we hard-code an intermediate robot pose similar to kPAM [88].

B.2 Description of Objects

In this section, we provide a detailed description of the objects in our experiment.

Whiteboard wiping: For this experiment we have 11 erasers available on Amazon, where 4 of them have exactly the same shape but different colors. The erasers height roughly ranges from 1.5 [cm] to 7 [cm], length ranges from 15 [cm] to 30 [cm]. All the erasers are handled by the same policy.

We perform 3 trials for each eraser, except those 4 identical shape ones. For those 4

erasers, we perform 1 trial for each of them.

Printed peg-hole: For this experiment we have 9 peg-hole pairs made by 3D printing. The diameters ranges from 1.5 [cm] to 2.5 [cm]. The peg height ranges from 2.5 [cm] to 15 [cm]. All peg-hole pairs have a tolerance of 0.2 [mm] (which is roughly the best accuracy achieved by our 3D printer).

LEGO block: We have 5 types of LEGO blocks in this experiment: 1x1, 1x2, 1x3, 1x4, 2x2. For each type there are many instances with different colors and slightly different tolerance (some are a little bit tighter than the others).

USB drive-port: We have 4 USB drive-port pairs in this experiment. We have 1 use drive and 3 plates with USB port, where one of those 3 plates has 2 USB ports on it. Those plates have a significant shape variation, where a pose estimator use plate as the template is very likely to fail on another.

We perform 5 trials for each peg-hole pairs in the printed peg-hole, LEGO blocks and USB. Those 9+5+4=18 pairs are evaluated with the same manipulation policy.

For the open-loop baseline, we reduce the number of trials to 20 per task for safety reason: the open-loop policy is much more likely to generate a large force in short period. As a result, the force monitor might be too late to shut down the policy. We've once broken the finger of the parallel gripper while evaluating this baseline. For this baseline, we also test each object with a roughly the same repetitions.

B.3 Visual Perception Implementation

As mentioned in the paper, we use robot kinematics for real-time keypoint tracking. To initialize the keypoint tracking, we need to run object detection, keypoint detection and grasping planning. We use a wrist-mounted camera for those perception tasks.

For object detection and keypoint detection, we use exactly the same neural network structure, data collection pipeline and annotation tools as kPAM [88], and their implementation details are provided in the appendix of kPAM. The keypoint neural network produces 3D keypoints (not oriented). Thus, we compute the axis of oriented keypoints by the relative direction of two 3D keypoints. We also use heuristics to reduce the number of required

3D keypoints annotations. For instance we assume the p_{front} keypoint has an opposite x axis as the p_{center} keypoint in the whiteboard experiment. In this way, we only need one "auxiliary" 3D keypoint (instead of two) for the x axis of the p_{front} and p_{center} keypoints.

For grasp planning, we follow kPAM [88] to use 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 [151].

B.4 Controller Implementation

The agent produces keypoint velocity command as the output. For the whiteboard experiment where we would like to regulate the z-axis force, we would first convert the z-axis desired force into a command velocity as

$$v_{\text{command\_}z} = K_z(f_{\text{measured\_}z} - f_{\text{desired\_}z}) \tag{B.1}$$

where $f_{\text{measured\_}z}$ is a low-pass filtered force measurement. After this conversion we only have keypoint velocity command.

The keypoint velocity command is further converted into joint velocity command, using Equ. (2) in the paper. The joint velocity commands is then sent to the robot driver. The agent runs at 50 Hz, however there exists some jitter as the agent is a python process in a docker container. The converter that maps keypoint velocity into joint velocity runs at 200 Hz, and the converted velocity is filtered to avoid excessively large velocity. The robot driver runs at 1000 Hz and it might also perform some filtering internally (the detailed information about the Kuka robot driver is not available to us).

B.5 Visual Perception Accuracy Statistics

In this section, w evaluate the accuracy of keypoint detection and its correlation with the success of trials. To do this, we annotated keypoints in our printed peg-hole experiment.

We divided the keypoints into two groups: if a peg-hole insertion succeeds, we place all keypoints detected in this trail to the "success" group. The "fail" group is defined similarly. The statistics is shown in the following table.

Table B.1: Accuracy Statistics for Keypoint Perception.

	overall	success keypoints	fail keypoints
mean error [cm]	0.51	0.43	0.73
max error [cm]	1.14	0.62	1.14
min error [cm]	0.27	0.27	0.51

From the table, we can see the keypoint detection is generally rather accurate with a mean error of 0.51 [cm] and maximum error 1.14 [cm]. The "success" group has a mean accuracy of 0.43 [cm], which is smaller than the 0.73 [cm] mean error of the "fail" group. This implies the keypoint accuracy has a strong correlation with the success of trials. From the table, the maximum keypoint error that can be tolerated in the printed peg-hole experiment is about 0.62 [cm]. However, this does not guarantee the success of insertion as we have observed failure trials with a smaller keypoint error.

Bibliography

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[2] Walid Amanhoud, Mahdi Khoramshahi, and Aude Billard. A dynamical system approach to motion and force generation in contact tasks. Robotics: Science and Systems (RSS), 2019.

[3] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[4] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[5] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

[7] Siddhant Bhambri, Sumanyu Muku, Avinash Tulasi, and A Balaji Buduru. A survey of black-box adversarial attacks on computer vision models. *arXiv preprint arXiv:1912.01667*, 2019.

[8] Basilio Bona and Marina Indri. Friction compensation in robotics: an overview. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4360–4367. IEEE, 2005.

[9] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[10] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[11] Tom B Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. Unrestricted adversarial examples. *arXiv preprint arXiv:1809.08352*, 2018.

[12] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[13] C Canudas de Wit, P Noel, A Aubin, and Bernard Brogliato. Adaptive friction compensation in robot manipulators: Low velocities. *The International journal of robotics research*, 10(3):189–199, 1991.

[14] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2017.

[15] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[16] Sachin Chitta, E Gil Jones, Matei Ciocarlie, and Kaijen Hsiao. Perception, planning, and execution for mobile manipulation in unstructured environments. *IEEE Robotics and Automation Magazine, Special Issue on Mobile Manipulation*, 19(2):58–71, 2012.

[17] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[18] Anthony Corso, Robert J Moss, Mark Koren, Ritchie Lee, and Mykel J Kochenderfer. A survey of algorithms for black-box safety validation. *arXiv preprint arXiv:2005.02979*, 2020.

[19] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

[20] Hongkai Dai, Gregory Izatt, and Russ Tedrake. Global inverse kinematics via mixed-integer convex optimization. *The International Journal of Robotics Research*, 38(12-13):1420–1441, 2019.

[21] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302. IEEE, 2014.

[22] Alessandro De Luca and Raffaella Mattone. Sensorless robot collision detection and hybrid force/motion control. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 999–1004. IEEE, 2005.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[24] Rosen Diankov. Automated construction of robotic manipulation programs. 2010.

[25] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[26] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63(4):1031–1053, 2019.

[27] Boston Dynamic. Introducing spot, 2019.

[28] Joseph F Engelberger. *Robotics in practice: management and applications of industrial robots*. Springer Science & Business Media, 2012.

[29] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. 2018.

[30] Ross Finman, Thomas Whelan, Michael Kaess, and John J Leonard. Toward lifelong object segmentation from change detection in dense rgb-d maps. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 178–185. IEEE, 2013.

[31] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[32] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[33] Peter Florence, Lucas Manuelli, and Russ Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 2019.

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

[35] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[36] Wei Gao and Russ Tedrake. Surfelwarp: Efficient non-volumetric single view dynamic reconstruction. *Robotics Science and Systems (RSS)*, 2018.

[37] Wei Gao and Russ Tedrake. Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization. *CVPR*, 2019.

[38] Wei Gao and Russ Tedrake. kpam-sc: Generalizable manipulation planning using keypoint affordance and shape completion. *arXiv preprint arXiv:1909.06980*, 2019.

[39] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research*, 37(13-14):1796–1825, 2018.

[40] Roland Geraerts and Mark H Overmars. A comparative study of probabilistic roadmap planners. In *Algorithmic foundations of robotics V*, pages 43–57. Springer, 2004.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680, 2014.

[42] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[43] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[44] 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)*, pages 7433–7440. IEEE, 2018.

[45] Marcus Gualtieri, Andreas Ten Pas, Kate Saenko, and Robert Platt. High precision grasp pose detection in dense clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 598–605. IEEE, 2016.

[46] Han Xu Yao Ma Hao-Chen, Liu Debayan Deb, Hui Liu Ji-Liang Tang Anil, and K Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178, 2020.

[47] Kris Hauser. Task planning with continuous actions and nondeterministic motion planning queries. 2010.

[48] Joshua A Haustein, Kaiyu Hang, Johannes Stork, and Danica Kragic. Object placement planning and optimization for robot manipulators. *arXiv preprint arXiv:1907.02555*, 2019.

[49] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[51] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in neural information processing systems*, pages 2266–2276, 2017.

[52] Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. Force-and-motion constrained planning for tool use.

[53] T Hsia. Adaptive control of robot manipulators-a review. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 183–189. IEEE, 1986.

[54] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, 25(7):627–643, 2006.

[55] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[56] Toyota Research Institute. Introducing tri manipulation for human-assist robots, 2019.

[57] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[58] Joseph L Jones, Emmanuel Mazer, and Patrick A O'Donnell. Task-level planning of pick-and-place robot motions. In *Artificial intelligence at MIT*, pages 40–59. MIT Press, 1991.

[59] Sandeep Juneja and Perwez Shahabuddin. Rare-event simulation techniques: an introduction and recent advances. *Handbooks in operations research and management science*, 13:291–350, 2006.

[60] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[61] Daniel Kappler, Franziska Meier, Jan Issac, Jim Mainprice, Cristina Garcia Cifuentes, Manuel Wüthrich, Vincent Berenz, Stefan Schaal, Nathan Ratliff, and Jeannette Bohg. Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, 3(3):1864–1871, 2018.

[62] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[63] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483. IEEE, 2011.

[64] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[65] Oussama Khatib and Joel Burdick. Motion and force control of robot manipulators. In *Proceedings. 1986 IEEE international conference on robotics and automation*, volume 3, pages 1381–1386. IEEE, 1986.

[66] Pradeep K Khosla and Takeo Kanade. Parameter identification of robot dynamics. In *1985 24th IEEE conference on decision and control*, pages 1754–1760. IEEE, 1985.

[67] Mark Koren and Mykel J Kochenderfer. Adaptive stress testing without domain heuristics using go-explore. *arXiv preprint arXiv:2004.04292*, 2020.

[68] Markus Koschi, Christian Pek, Sebastian Maierhofer, and Matthias Althoff. Computationally efficient safety falsification of adaptive cruise control systems. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2879–2886. IEEE, 2019.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[70] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[71] Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.

[72] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[73] Florent Lamiraux and J Mirabel. Humanoid path planner, 2014.

[74] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[75] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[76] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[77] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[78] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *arXiv preprint arXiv:1501.05611*, 2016.

[79] Yangmin Li. Hybrid control approach to the peg-in hole problem. *IEEE Robotics & Automation Magazine*, 4(2):52–60, 1997.

[80] Hsueh-Ti Derek Liu, Michael Tao, Chun-Liang Li, Derek Nowrouzezahrai, and Alec Jacobson. Beyond pixel norm-balls: Parametric adversaries using an analytically differentiable renderer. *arXiv preprint arXiv:1808.02651*, 2018.

[81] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018.

[82] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-bnn: Improved adversarial defense through robust bayesian neural network. *arXiv preprint arXiv:1810.01279*, 2018.

[83] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[84] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. *arXiv preprint arXiv:1711.10337*, 2017.

[85] Jens Lundell, Francesco Verdoja, and Ville Kyrki. Robust grasp planning over uncertain shape completions. *arXiv preprint arXiv:1903.00645*, 2019.

[86] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019.

[87] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. 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)*, pages 1957–1964. IEEE, 2016.

[88] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *The International Symposium on Robotic Research (ISRR)*, 2019.

[89] Pat Marion. Director: A robotics interface and visualization framework, 2015.

[90] Pat Marion, Peter R Florence, Lucas Manuelli, and Russ Tedrake. Labelfusion: A pipeline for generating ground truth labels for real rgbd data of cluttered scenes. *arXiv preprint arXiv:1707.04796*, 2017.

[91] 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`, 2018. Accessed: [Insert date here].

[92] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[93] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[94] Igor Mordatch, Jack M Wang, Emanuel Todorov, and Vladlen Koltun. Animating human lower limbs using contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 32(6):1–8, 2013.

[95] Guillaume Morel, Ezio Malis, and Sylvie Boudet. Impedance based combination of visual and force control. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 2, pages 1743–1748. IEEE, 1998.

[96] Douglas Morrison, Peter Corke, and Jürgen Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *arXiv preprint arXiv:1804.05172*, 2018.

[97] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[98] Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757, 2008.

[99] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(10):20, 2016.

[100] Nvidia. Celebrating robotics in seattle, 2019.

[101] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, Russ Tedrake, and John C Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *Advances in Neural Information Processing Systems*, 31:9827–9838, 2018.

[102] Howard Olsen and George Bekey. Identification of robot dynamics. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1004–1010. IEEE, 1986.

[103] Hyeonjun Park, Jaeheung Park, Dong-Hyuk Lee, Jae-Han Park, Moon-Hong Baeg, and Ji-Hun Bae. Compliance-based robotic peg-in-hole assembly strategy without force feedback. *IEEE Transactions on Industrial Electronics*, 64(8):6299–6309, 2017.

[104] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019.

[105] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[106] Andrew Price, Linyi Jin, and Dmitry Berenson. Inferring occluded geometry improves performance when retrieving an object from dense clutter. *arXiv preprint arXiv:1907.08770*, 2019.

[107] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[108] Xin Qin, Nikos Aréchiga, Andrew Best, and Jyotirmoy Deshmukh. Automatic testing and falsification with dynamically constrained reinforcement learning. *arXiv preprint arXiv:1910.13645*, 2019.

[109] Zengyi Qin, Kuan Fang, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Keto: Learning keypoint representations for tool manipulation. *arXiv preprint arXiv:1910.11977*, 2019.

[110] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[111] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. 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)*, pages 3758–3765. IEEE, 2018.

[112] Marc H Raibert, John J Craig, et al. Hybrid position/force control of manipulators. 1981.

[113] Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3, pages 1936–1941. IEEE, 2002.

[114] Diego Rodriguez, Corbin Cogswell, Seongyong Koo, and Sven Behnke. Transferring grasping skills to novel instances by latent space non-rigid registration. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[115] Caner Sahin and Tae-Kyun Kim. Category-level 6d object pose recovery in depth images. In *European Conference on Computer Vision*, pages 665–681. Springer, 2018.

[116] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. *arXiv preprint arXiv:1906.05841*, 2019.

[117] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. Citeseer.

[118] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[119] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[120] Max Schwarz, Christian Lenz, Germán Martín García, Seongyong Koo, Arul Selvam Periyasamy, Michael Schreiber, and Sven Behnke. Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3347–3354. IEEE, 2018.

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

[122] Aman Sinha, Matthew O'Kelly, Russ Tedrake, and John C Duchi. Neural bridge sampling for evaluating safety-critical autonomous systems. *Advances in Neural Information Processing Systems*, 33, 2020.

[123] Jean-Jacques E Slotine and Weiping Li. On the adaptive control of robot manipulators. *The international journal of robotics research*, 6(3):49–59, 1987.

[124] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing unrestricted adversarial examples with generative models. *Advances in Neural Information Processing Systems*, 31:8312–8323, 2018.

[125] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[126] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[127] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral human pose regression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 529–545, 2018.

[128] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[129] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[130] Te Tang, Hsien-Chung Lin, and Masayoshi Tomizuka. A learning-based framework for robot peg-hole-insertion. In *ASME 2015 Dynamic Systems and Control Conference*, pages V002T27A002–V002T27A002. American Society of Mechanical Engineers, 2015.

[131] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016.

[132] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473, 2017.

[133] Marc Toussaint, Kelsey Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning.

[134] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. *Conference on Robot Learning (CoRL)*, 2018.

[135] Cumhur Erkan Tuncali and Georgios Fainekos. Rapidly-exploring random trees for testing automated vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 661–666. IEEE, 2019.

[136] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1555–1562. IEEE, 2018.

[137] Jonathan Uesato, Ananya Kumar, Csaba Szepesvari, Tom Erez, Avraham Ruderman, Keith Anderson, Nicolas Heess, Pushmeet Kohli, et al. Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. *arXiv preprint arXiv:1812.01647*, 2018.

[138] Mark Van der Merwe, Qingkai Lu, Balakumar Sundaralingam, Martin Matak, and Tucker Hermans. Learning continuous 3d reconstructions for geometrically aware grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11516–11522. IEEE, 2020.

[139] Herke Van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934. IEEE, 2016.

[140] Jacob Varley, Chad DeChant, Adam Richardson, Joaquín Ruales, and Peter Allen. Shape completion enabled robotic grasping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2442–2447. IEEE, 2017.

[141] Brian J Waibel and Homayoon Kazerooni. Theory and experiments on the stability of robot compliance control. *IEEE Transactions on Robotics and Automation*, 7(1):95–104, 1991.

[142] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. *arXiv preprint arXiv:1901.02970*, 2019.

[143] David Watkins-Valls, Jacob Varley, and Peter Allen. Multi-modal geometric learning for grasping and manipulation. *arXiv preprint arXiv:1803.07671*, 2018.

[144] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M Pawan Kumar. A statistical approach to assessing neural network robustness. *arXiv preprint arXiv:1811.07209*, 2018.

[145] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[146] Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5777–5783, 2018.

[147] 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*, pages 75–82. IEEE, 2014.

[148] Pin-Chu Yang, Kazuma Sasaki, Kanata Suzuki, Kei Kase, Shigeki Sugano, and Tetsuya Ogata. Repeatable folding task by humanoid robot worker using deep learning. *IEEE Robotics and Automation Letters*, 2(2):397–403, 2016.

[149] Kevin Zakka, Andy Zeng, Johnny Lee, and Shuran Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. *arXiv preprint arXiv:1910.13675*, 2019.

[150] Radosiaw R Zakrzewski. Verification of a trained neural network accuracy. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1657–1662. IEEE, 2001.

[151] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, 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)*, pages 1–8. IEEE, 2018.

[152] Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1386–1383. IEEE, 2017.

[153] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR, 2019.

[154] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[155] Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Josh Tenenbaum, Bill Freeman, and Jiajun Wu. Learning to reconstruct shapes from unseen classes. In *Advances in Neural Information Processing Systems*, pages 2257–2268, 2018.

[156] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.