공학박사학위논문

# 비전 기반 물체 조작 학습:
# 형상 인식 기반 방법론

## Learning for Vision-Based Object Manipulation:
## A Shape Recognition-Based Approach

2024 년 2 월

서울대학교 대학원

기계공학부

김 승 연

# Learning for Vision-Based Object Manipulation: A Shape Recognition-Based Approach

SEUNGYEON KIM

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in the
DEPARTMENT OF MECHANICAL ENGINEERING

at
SEOUL NATIONAL UNIVERSITY

February 2024

# 비전 기반 물체 조작 학습:
# 형상 인식 기반 방법론

## Learning for Vision-Based Object Manipulation:
## A Shape Recognition-Based Approach

지도교수 박 종 우

이 논문을 공학박사 학위논문으로 제출함

2023 년 10 월


서울대학교 대학원

기계공학부

김 승 연


김승연의 공학박사 학위논문을 인준함

2023 년 12 월

위 원 장 : 이 동 준 (인)

부위원장 : 박 종 우 (인)

위 원 : 임 종 우 (인)

위 원 : 김 아 영 (인)

위 원 : 김 순 겸 (인)

# ABSTRACT

## Learning for Vision-Based Object Manipulation: A Shape Recognition-Based Approach

by

Seungyeon Kim

Department of Mechanical and Aerospace Engineering
Seoul National University

Vision-based object manipulation has emerged as a fundamental aspect of robotics, encompassing tasks like grasping, pushing, and rearranging objects within complex environments. Recent technological advancements have led robots to increasingly rely on vision sensors as their primary means of interacting with the surrounding objects or environments. However, these interactions pose numerous challenges. From the inherent data inefficiencies found in end-to-end deep learning approaches, requiring extensive data collection and elaborate network training, to the various constraints that must be

navigated for practical application in real-world environments, addressing these challenges is pivotal to expanding the application scope of robotic object manipulation.

In this thesis, we demonstrate the effectiveness of mitigating many challenges associated with vision-based object manipulation through the integration of *shape recognition* methods. This approach involves recognizing an object's shape by aligning raw vision data with a 3D model, subsequently generating robot actions (e.g., grasping and pushing) based on the object's geometry. These methods not only address data inefficiencies but also reduce the need for extensive data collection without compromising accuracy and efficiency. We have discovered that 3D shape recognition of the scene effectively resolves numerous challenges encountered in object manipulation, especially in practical application in real-world environments, significantly simplifying and enhancing the handling of various challenging vision-based object manipulation problems. Our demonstration highlights that integrating shape recognition techniques into vision-based object manipulation substantially enhances a robot's interaction capabilities within its environment. Consequently, our primary contribution lies in introducing *shape recognition-based object manipulation*.

Building on this concept, we introduce a pioneering approach to grasping based on shape recognition. This method integrates a diverse set of shape templates, particularly deformable superquadrics, with a deep learning network referred to as the *Deformable SuperQuadric Network (DSQNet)*. DSQNet is designed to identify complete object shapes by inferring deformable superquadrics from partial point cloud data. Through supervised learning, DSQNet generates the eight parameters and the pose of the deformable superquadric, accurately aligning with the entire object shape, even considering occluded sections. Subsequent strategies for grasping account for the gripper's kinematic and structural attributes, leveraging the closed-form equations associated with deformable superquadrics. Comparative analyses demonstrate DSQNet's superiority over

existing shape recognition baselines in both accuracy and speed. Notably, our shape recognition-based method sets new standards in grasping success rates, surpassing prevailing techniques, thanks to its precise shape recognition capabilities.

We then leverage the advantages of shape recognition to facilitate the learning of pushing dynamics models. Initially, we also employ superquadrics to recognize object shapes placed on tabletops. An inherent advantage of integrating shape recognition in this context is its natural ability to define a $\mathrm{SE}(2)$-equivariant pushing dynamics model. This sets the stage for developing the *SuperQuadric Pushing Dynamics Network (SQPD-Net)*, a neural network-based $\mathrm{SE}(2)$-equivariant pushing dynamics model. This approach inherently acknowledges the symmetry within physical systems, resulting in substantial improvements in generalization performance. Comparative assessments demonstrate that our shape recognition-based model surpasses existing vision-based pushing dynamics models, particularly due to the reinforced $\mathrm{SE}(2)$-equivariance. Moreover, the effectiveness of our model is further validated through its application in model-based optimal controls across various pushing manipulation tasks, corroborated by both simulation and real-world experiments.

Finally, we tackle the intricate task of mechanical search on cluttered shelves employing shape recognition methods. This task involves locating and grasping a specific target object situated within a cluttered shelf environment, even when the target object is occluded by other objects, eluding initial detection by vision sensors. In such scenarios, the robot's task is to strategically rearrange nearby objects to reveal the target's position while avoiding collisions with the shelf and surrounding objects. To address this challenge, we also leverage a superquadric shape recognition model. These models enable the use of shape recognition-based object manipulation techniques developed in this thesis, and moreover, facilitates quick and efficient reasoning about potential poses of the occluded target object. They achieve this by expediting computations for various

tasks like depth image rendering and collision checking. This empowers the robot to effectively and safely find and grasp the target object. Our method demonstrates success in finding and grasping the target object using a conventional robot gripper in both simulated and real-world settings.

**Keywords:** Vision-based object manipulation, shape recognition, robotic grasping, pushing manipulation, pushing dynamics learning, mechanical search, object rearrangement.

**Student Number:** 2019-39029

# Contents

# List of Tables

# List of Figures

# 1 Introduction

## 1.1 Learning for Vision-Based Object Manipulation

Interacting with and manipulating objects is a fundamental skill in robotics, crucial for both human-robot collaboration and the automation of a wide array of tasks. Object manipulation can generally be categorized into two primary methods: (i) prehensile manipulation, where a robot directly holds or grasps an object, maintaining continuous contact, and (ii) non-prehensile manipulation, where the robot influences an object's pose or movement without continuously grasping onto it. For instance, prehensile manipulation involves picking up an object, while non-prehensile manipulation might entail pushing objects on a table or throwing them into a bucket. When objects are identifiable and their CAD models or physical properties (e.g., mass, inertia, and friction coefficient) are known, the problem of object manipulation becomes substantially more straightforward. Such scenarios benefit from well-understood, reliable, and efficient model-based solutions that have found prominence in structured manufacturing environments (see [1, 2] for grasping and [3, 4, 5, 6] for pushing). However, in practice, especially in many of

today's warehouse logistics and automation environments, a far more common scenario is the need to manipulate, in real-time, unknown objects that the robot encounters for the first time. These objects might not have available CAD models or known physical properties, and they may be only partially visible due to, for instance, occlusions. In this thesis, we address the *vision-based object manipulation* problem of interacting with objects using solely camera vision data.

Spurred in part by the tremendous success of deep learning networks in recognizing and detecting objects within images, there has recently been a surge in interest in employing these networks to manipulate partially visible unknown objects. One of the prevailing methods involves using deep networks that directly process raw vision data (e.g., RGB images or depth images) and output the corresponding robot actions. For prehensile manipulation, several studies have focused on generating a grasp pose for an object directly from raw vision data using trained deep neural networks [7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. For non-prehensile manipulation, especially pushing manipulation, there are chiefly two types of approaches: (i) model-free methods which train a policy that directly maps raw vision data to a sequence of pushing actions, maximizing a task-specific reward function [17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33] and (ii) model-based methods. The latter first construct a pushing dynamics model that predicts subsequent vision data after a robot executes a pushing action. This is followed by identifying an optimal sequence of pushing actions that meet a predefined task criteria [34, 35, 36, 37, 38, 39, 40, 41, 42, 43]. It's worth noting that this thesis places its emphasis on model-based methods for non-prehensile manipulation. These methodologies have proven effective in rapidly and precisely executing object manipulation tasks through a straightforward pass of the neural network.

Despite the widely reported successes of deep learning-based methods, these impressive results often come with significant costs and effort. Deep learning methods


Figure 1.1: The box object and pushing vector in *Scene 1* are transformed by some same planar rigid-body transformation as those in *Scene 2*. An ideal pushing dynamics model should be $\mathrm{SE}(2)$-equivariant, i.e., the resulting motion in *Scene 2* is a transformation of that in *Scene 1*.

necessitate very large datasets, typically comprising hundreds of thousands to millions of vision sensor data-robot action pairs. For learning prehensile manipulation, physics-based simulations are frequently employed to supplement the real data obtained experimentally. However, as is widely recognized, the accuracy of any physics-based simulator—especially when multiple contacts with friction are involved—is often questionable at best. As expected, training the network with such extensive datasets can be both time-consuming and computationally demanding. A particularly critical issue in prehensile manipulation is that the trained network tends to be reliable only for the gripper used during data collection or those of a very similar design. Different gripper designs inevitably necessitate some degree of re-training using newly collected data.

Moreover, beyond the data-driven methods for learning non-prehensile manipulation, this thesis argues that their generalization performances, even with large-scale datasets, are still less than satisfactory. One of the significant reasons for this shortfall is that the neural network models employed in existing approaches do not adequately account for

the symmetry of the physical systems—more precisely, they lack *equivariance*. Consider a pushing dynamics model designed for model-based pushing manipulation. Suppose this model is trained using an experience where a robot pushes a box in the direction indicated by a red arrow, as illustrated in Figure 1.1 (*Scene 1*). Now, envision a new scenario in which the same box is positioned differently, yet the robot pushes it in the same relative direction, as depicted in Figure 1.1 (*Scene 2*). Intuitively, an effective model should be able to generalize effortlessly to such scenarios, where tabletop objects undergo translation or rotation along the $z$-axis. In more technical language, the pushing dynamics model must exhibit equivariance to the $\mathrm{SE}(2)$ transformation. Unfortunately, many state-of-the-art data-driven methods lack this equivariance property, leading to suboptimal learning performances.

For these and other reasons, learning both prehensile and non-prehensile manipulations using only camera vision data remains a practical challenge.

## 1.2 A Shape Recognition-based Approach

In this thesis, we propose *shape recognition-based object manipulation*, a method that involves recognizing the object shape by matching raw vision data to a 3D shape and subsequently generating robot actions (such as grasping and pushing) based on the object's geometry. Shape recognition methodologies bypass many inherent limitations, particularly those related to data inefficiencies. The extensive requirements of data collection and network training, typical of end-to-end strategies, are circumvented through the object shape recognition phase. Some of these shape recognition methods are independent of training datasets [44, 45, 46, 47, 48, 49, 50], while others require considerably less data for model training [51, 52, 53, 54, 55, 56, 57, 58, 59, 60]. Our aim is to devise methods that effectively conduct object manipulation, capitalizing on shape recognition's

data efficiency.

Moreover, this thesis demonstrates that numerous challenges encountered in object manipulation can effectively be addressed using shape recognition methods. The key insight is that 3D reasoning of a scene from visual observations, particularly through 3D shape recognition, significantly simplifies and enhances the handling of various challenging vision-based object manipulation problems. Building on this perspective, we have developed shape recognition-based object manipulation algorithms for grasping unknown objects, learning pushing dynamics models for tabletop object manipulation, and conducting mechanical searches in cluttered shelf environments. The main contributions of the thesis are outlined in the following chapters.

## 1.3 Contribution

### 1.3.1 A Novel Shape Recognition-based Object Grasping Method

For prehensile manipulation, various shape recognition-based grasping techniques have been introduced. In these methodologies, recognition and grasp pose generation are decoupled, allowing for the application of traditional grasp pose generation algorithms (e.g., [1, 2]) to the identified 3D shape. Thus, these methods can accommodate a broader range of grippers by simply modifying the grasp pose generation module. Nonetheless, metrics such as computational time or grasp success rate hinge on the precision and efficiency of object recognition, making the choice of object representation crucial. Object representation spans from basic shape primitives like boxes, spheres, cylinders, and other standard shapes [44, 45, 46, 47, 48, 49, 54, 59], to advanced 3D representations like voxel-valued or implicit function representations [51, 58, 60]. While the latter can detail various and intricate shapes but often demand prolonged planning stages to produce feasible grasp poses, so they are not suitable for scenarios where real-time manipulation is

required. Conversely, the former quickly deduces the grasp pose, but it may fall short in representing intricacies of complex objects, potentially reducing the grasp success rate. Thus, the challenge remains in selecting an object representation that optimally balances these trade-offs.

Building on this discussion, we introduce a novel shape recognition-based grasping method that integrates a more extensive set of shape templates, namely the deformable superquadrics, with a deep learning network. This network, termed *Deformable Super-Quadric Network (DSQNet)*, is trained to identify complete object shapes using a set of deformable superquadrics inferred from partially observed point cloud data input. These deformable superquadrics are characterized by eight continuous parameters, enabling them to represent a diverse array of shapes. Additionally, closed-form surface equations are available [61, 62, 63], facilitating the efficient computation of point-to-surface distances for fitting. We devise a suite of deformable superquadric primitives tailored for grasping applications. This suite offers a harmonious blend of expressiveness and efficiency. Through supervised learning, the DSQNet produces the eight parameters along with the pose of the deformable superquadric, ensuring alignment with the complete shape of the object, inclusive of occluded sections. Our findings indicate that DSQNet can recognize object shapes both accurately and swiftly when compared with existing shape recognition-based baseline methods. Leveraging precise shape recognition, our approach also tops the charts in grasping success rates among existing techniques. Notably, our recognition-driven method demonstrates a grasping performance on par with many prevalent end-to-end strategies. This is achieved while enjoying the benefits of minimal training data requirements and adaptability to a broad spectrum of grippers, in contrast to the constraints of end-to-end methods.

### 1.3.2 Adopting Shape Recognition for Pushing Dynamics Learning

Taking the same perspective, we extend the advantages of shape recognition to the problem of learning non-prehensile manipulation, with a particular focus on learning pushing dynamics. While shape recognition has conferred numerous benefits in grasping problems, its application to pushing manipulation remains unexplored. Mirroring the grasping method described earlier, we first recognize the shapes of the objects on the tabletop using superquadrics. One significant advantage of introducing shape recognition to this domain is the natural ability to define the $\mathrm{SE}(2)$-equivariant pushing dynamics model, allowing us to purposefully design a neural network architecture inherently possessing the equivariance property. To this end, we introduce a neural network-based $\mathrm{SE}(2)$-equivariant pushing dynamics model named *SuperQuadric Pushing Dynamics Network (SQPD-Net)*. Central to ensuring the model's equivariance is the appropriate transformation of the pushing action coordinates and the objects' poses. This method inherently accounts for the symmetry of the physical systems, leading to substantial enhancements in generalization performance. Our results demonstrate that our shape recognition-based model outperforms existing vision-based pushing dynamics models, especially with the aid of $\mathrm{SE}(2)$-equivariance. Further validation of our model's effectiveness is provided through its application in model-based optimal controls for various pushing manipulation tasks, corroborated by both simulation and real-world experiments.

### 1.3.3 Mechanical Search on Shelves using Shape Recognition

The final contribution of this thesis addresses the practical yet challenging task of *mechanical search on cluttered shelves*. This entails finding and grasping a desired target object on a cluttered shelf, even when the target is occluded by unknown objects and initially remains undetected by a vision sensor. In such scenarios, the robot has the

responsibility to rearrange the surrounding objects in order to determine the target's pose and subsequently grasp it, ensuring all the while that it avoids collisions with the shelf and adjacent objects. The geometric configuration of the shelf, which permits visual observations exclusively from the front and restricts the manipulator's operational space, introduces further challenges. We employ shape recognition models to tackle this problem, specifically the superquadric recognition model mentioned previously. By integrating the prehensile and non-prehensile manipulation techniques we developed above, the superquadric object representation further facilitates fast depth image rendering and collision assessments. This empowers the robot to effectively and safely (i.e., without causing collision) find the target object. We have corroborated the efficacy of our approach, demonstrating its capability to find and grasp target objects using a conventional two-finger robot gripper, both in simulations and real-world scenarios. Importantly, our method remains robust even when faced with noise originating from vision sensor data in real-world environments.

## 1.4 Organization

In Chapter 2, we review the superquadrics and their deformable counterparts. We begin by offering a succinct description of the formulas associated with superquadrics, encompassing both their implicit surface functions and surface normal vectors, which are essential for grasp pose generation. Subsequently, we introduce a more expressive set of shapes termed deformable superquadrics. These are derived by applying global tapering and bending deformations to the vanilla superquadrics. We conclude the chapter by deriving the formulas needed to determine the surface normal vector of these deformable superquadrics.

In Chapter 3, we introduce a novel shape recognition-based grasping method that

advances beyond existing techniques. We begin by detailing a shape recognition model called the *Deformable Superquadric Network (DSQNet)*. This model takes partially observed point cloud data as its input and produces the eight parameters and the pose of the deformable superquadric, ensuring it aligns with the complete shape of the partially observed object, even accounting for occluded sections. We then elaborate on how our comprehensive framework depicts the object using multiple deformable superquadrics, incorporating an additional segmentation step when the object consists of several parts. Additionally, we explain the training processes for both the segmentation network and the DSQNet. Through comparative analysis, we demonstrate that our method outperforms its counterparts in terms of accuracy and computational speed. Furthermore, we validate that our shape recognition-based grasping method consistently achieves the highest success rate when compared to existing methods. This chapter is based on [64].

In Chapter 4, we introduce a shape recognition-based pushing dynamics model that exhibits the $\mathrm{SE}(2)$-equivariance property, setting it apart from existing approaches. We begin with a detailed definition of the $\mathrm{SE}(2)$-equivariant pushing dynamics model and purposefully design a neural network architecture which, by its construction, possesses the equivariance property. Importantly, we elucidate the fundamental approach to ensure model equivariance by appropriately transforming the coordinates associated with the pushing action and the objects' poses. Subsequently, we detail the neural network-based $\mathrm{SE}(2)$-equivariant pushing dynamics model, termed *SuperQuadric Pushing Dynamics Network (SQPD-Net)*, which draws from the superquadric recognition model. In the latter part of this chapter, we formulate a model-based pushing manipulation problem, designed to manipulate objects utilizing the acquired pushing dynamics model aligned with predefined task criteria. Our evaluations show that our dynamics model consistently outperforms existing state-of-the-art methods in predicting post-push object

trajectories. Furthermore, we validate the effectiveness of our model through its application in model-based optimal control scenarios across a range of pushing manipulation tasks, with results validated in both simulated and real-world settings. This chapter is based on [65].

Chapter 5 addresses the challenge of mechanical search on cluttered shelves, leveraging the merits of the shape recognition method. We begin by outlining a pioneering framework to find and grasp a target object using a standard gripper, incorporating both prehensile and non-prehensile manipulations. Specifically, we introduce two key indicator functions: (i) an existence function, which discerns the potential presence of the target, and (ii) a graspability function, assessing the viability of grasping the detected target. Subsequently, we formulate a model-based optimal control problem, termed *Search-for-Grasp*. We further detail how the proposed indicator functions and their corresponding dynamics models can be adeptly estimated by harnessing a shape recognition model. Our evaluations reveal that our approach reliably finds and grasps the target object using a standard robotic gripper in both simulated and real-world environments. Notably, we demonstrate the adaptability and robustness of our method in the face of noise from real-world vision sensors. This chapter is based on [66].

# 2

# Preliminaries: Deformable Superquadrics

## 2.1 Introduction

In this chapter, we review the use of *deformable superquadrics* as a set of shape primitives. The superquadrics constitute an extended set of quadric surfaces [61, 62], and they are further categorized into superellipsoids, superparaboloids, and supertoroids. Chapter 2.2 briefly describes the formulas for these superquadrics, including their implicit surface functions and surface normal vectors (needed for grasp pose generation). *Throughout this dissertation, we use the term "superquadrics" to exclusively refer to superellipsoids, except for Chapter 3.7; this excluded section introduces algorithms that utilize both superellipsoids and superparaboloids.*

Although more expressive than quadrics, superquadrics are still limited by their inability to capture tapered and bent objects. Chapter 2.3 describes more expressive shape primitives named *deformable superquadrics*, obtained by applying global tapering and

bending deformations to superquadrics [67, 63]. Formulas for determining the surface normal vector of deformable superquadrics are also derived.

## 2.2 Superquadrics

### 2.2.1 Superellipsoids

The superellipsoids are an extended set of ellipsoid surfaces that can be used to represent diverse shapes ranging from boxes, cylinders, and ellipsoids to bi-cones, octahedra, and other complex symmetric shapes, even those with rounded corners and edges. The corresponding implicit equations for a superellipsoid surface are of the form

$$f(x,y,z) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}} = 1, \tag{2.2.1}$$

where $a_1, a_2, a_3$ are size parameters and $e_1, e_2$ are shape parameters. Some examples of possible shapes (with fixed size parameters) are shown in the upper row of Figure 2.1.

The implicit function of Equation (2.2.1) can be explicitly parametrized using polar coordinates $(\theta, \phi)$ as follows:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \cos^{e_1}\theta \cos^{e_2}\phi \\ a_2 \cos^{e_1}\theta \sin^{e_2}\phi \\ a_3 \sin^{e_1}\theta \end{bmatrix}, \tag{2.2.2}$$

where $-\pi/2 \le \theta \le \pi/2$ and $-\pi \le \phi \le \pi$. Note that for exponent $e$, $\cos^e\theta := \mathrm{sgn}(\cos\theta)|\cos\theta|^e$, and $\sin^e\theta$ is defined similarly.

The outward pointing surface normal vector $\mathbf{n}$ of a superellipsoid surface at a point

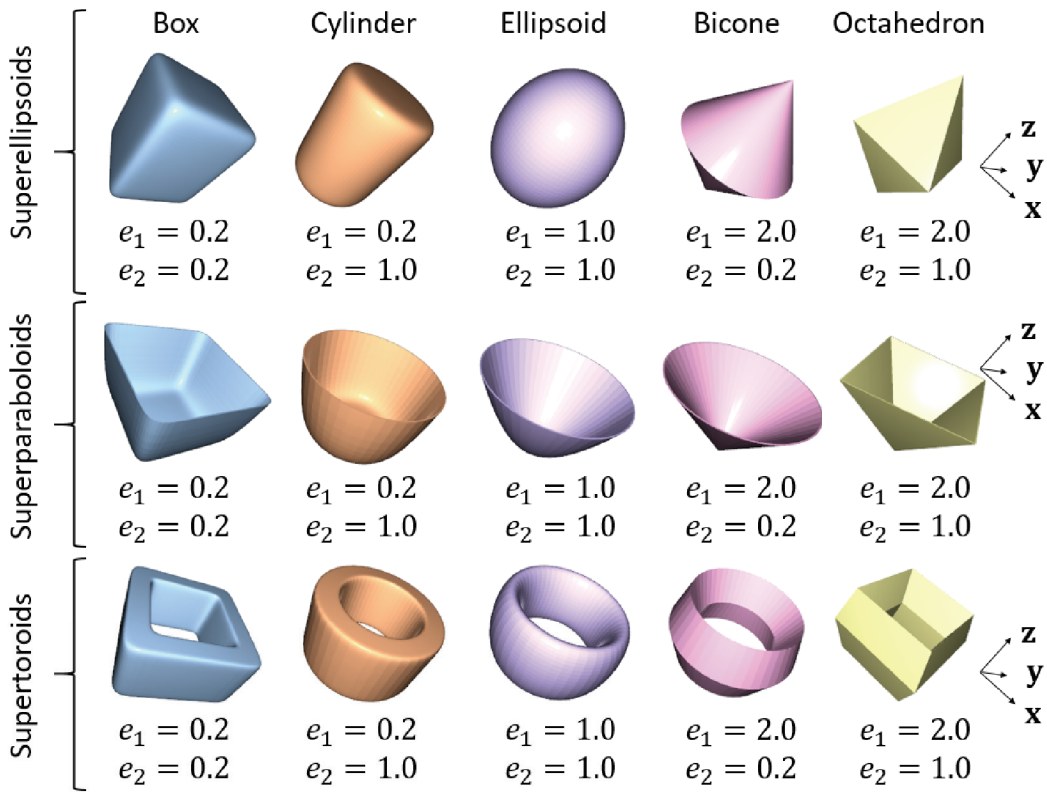


Figure 2.1: Examples of superellipsoids (upper row, Chapter 2.2.1), superparaboloids (middle row, Chapter 2.2.2), and supertoroids (lower row, Chapter 2.2.3).

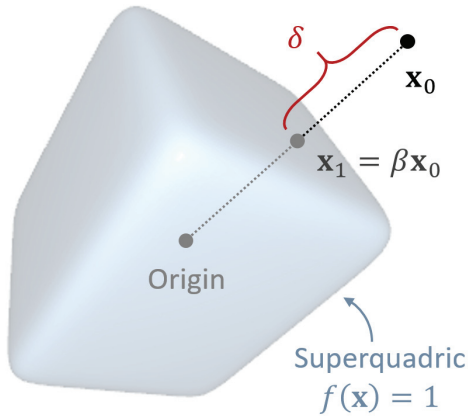


Figure 2.2: Radial distance $\delta$ between a point $\mathbf{x}_0$ and a superquadric $f(\mathbf{x}) = 1$.

$\mathbf{x}(\theta, \phi)$ can be calculated as the vector product of the two tangent vectors $\frac{\partial \mathbf{x}}{\partial \phi}$ and $\frac{\partial \mathbf{x}}{\partial \theta}$:

$$\mathbf{n}(\mathbf{x}) = \frac{\partial \mathbf{x}}{\partial \phi} \times \frac{\partial \mathbf{x}}{\partial \theta} = c(\theta, \phi) \begin{bmatrix} \frac{1}{a_1} \cos^{2-e_1}\theta \cos^{2-e_2}\phi \\ \frac{1}{a_2} \cos^{2-e_1}\theta \sin^{2-e_2}\phi \\ \frac{1}{a_3} \sin^{2-e_1}\theta \end{bmatrix}, \tag{2.2.3}$$

for some scalar function $c(\theta, \phi)$ (for our purposes $c(\theta, \phi)$ can be ignored since we only require the surface normal direction).

**Distance between a point and a superellipsoid $\delta$.** There is no closed-form solution for the shortest distance between a point $\mathbf{x}_0$ and a superquadric surface $f(\mathbf{x}) = 1$. Instead, we obtain another distance form that has a closed form, named the *radial distance*, and we primarily use this distance form when fitting data points (or a data point cloud) to a superquadric. The radial distance $\delta(\mathbf{x}_0, f)$ between a point $\mathbf{x}_0$ and a superquadric $f(\mathbf{x}) = 1$ is defined by the Euclidean distance between $\mathbf{x}_0$ and $\mathbf{x}_1$ – $\mathbf{x}_1$ is the intersection point of the surface $f(\mathbf{x}) = 1$ and the straight line connecting $\mathbf{x}_0$ and the origin of the superquadric – as described in Figure 2.2.

The intersection point $\mathbf{x}_1$ can be written as $\mathbf{x}_1 = \beta \mathbf{x}_0$, where $\beta$ is a positive scalar value. Since the point $\mathbf{x}_1$ lies on the surface of the superquadric, the equation $f(\mathbf{x}_1) = 1$ must be satisfied. Since $f(\mathbf{x}_1)$ can be rewritten as

$$f(\mathbf{x}_1) = \left( \left| \frac{\beta x_0}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{\beta y_0}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{\beta z_0}{a_3} \right|^{\frac{2}{e_1}} = \beta^{\frac{2}{e_1}} f(\mathbf{x}_0), \tag{2.2.4}$$

we can obtain $\beta$ in a closed form as follows:

$$\beta = f^{-\frac{e_1}{2}}(\mathbf{x}_0). \tag{2.2.5}$$

Thus, the radial distance $\delta$ between the point $\mathbf{x}_0$ and the superquadric $f(\mathbf{x}) = 1$ can be calculated with a closed-form solution:

$$\delta = ||(1-\beta)\mathbf{x}_0|| = ||\mathbf{x}_0|| |1 - f^{-\frac{e_1}{2}}(\mathbf{x}_0)|. \tag{2.2.6}$$

### 2.2.2 Superparaboloids

The superparaboloids are the geometric shapes that resemble paraboloids, and the implicit function for a superparaboloid surface has the following form

$$f(x,y,z) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} - \left( \frac{z}{a_3} \right) = 1, \tag{2.2.7}$$

where $a_1, a_2, a_3$ controls the sizes and $e_1, e_2$ controls the geometric shapes. Some examples of the superparaboloids are shown in the middle row of Figure 2.1. Similarly, the implicit function of Equation (2.2.7) can be explicitly parametrized using coordinates $(\theta, u)$ as follows:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 u \cos^{e_2}\theta \\ a_2 u \sin^{e_2}\theta \\ a_3 (u^{\frac{2}{e_1}} - 1) \end{bmatrix}, \tag{2.2.8}$$

where $-\pi \leq \theta \leq \pi$ and $0 \leq u \leq 1$. The surface normal vector $\mathbf{n}$ of a superparaboloid can be calculated in the same way as in the case of superquadrics using this explicit representation.

### 2.2.3 Supertoroids

The supertoroids are the geometric shapes that resemble toroids, and the implicit function for a supertoroid surface has the following form

$$f(x,y,z) = \left( \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{2}} - a_4 \right)^{\frac{2}{e_1}} + \left( \frac{z}{a_3} \right)^{\frac{2}{e_1}} = 1, \tag{2.2.9}$$

where $a_1, a_2, a_3, a_4$ controls the sizes and $e_1, e_2$ controls the geometric shapes. Especially, $a_4$ controls the size of the hole, while $a_1$ and $a_2$ control the thickness of the shape. Some examples of the supertoroids are shown in the lower row of Figure 2.1. Similarly, the implicit function (2.2.9) can be explicitly parametrized using coordinates $(\theta, \phi)$ as follows:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1(a_4 + \cos^{e_1}\theta)\cos^{e_2}\phi \\ a_2(a_4 + \cos^{e_1}\theta)\sin^{e_2}\phi \\ a_3 \sin^{e_1}\theta \end{bmatrix}, \tag{2.2.10}$$

where $-\pi \leq \theta \leq \pi$ and $-\pi \leq \phi \leq \pi$. The surface normal vector $\mathbf{n}$ of a supertoroid can be calculated in the same way as in the case of superellipsoids using this explicit representation.

## 2.3 Deformable Superquadrics

### 2.3.1 Tapering Deformation

In the case of tapering, a shape is gradually thinned or expanded along some direction. A parametric tapering deformation $D_t$ along the $z$-axis is defined to be

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \longmapsto \mathbf{X} = \begin{bmatrix} t_k(z)x \\ t_k(z)y \\ z \end{bmatrix}, \tag{2.3.11}$$

where $t_k(z)$ is called the tapering function with tapering parameter $k$. In our later examples we will make frequent use of the linear tapering function

$$t_k(z) = \frac{k}{a_3}z + 1, \tag{2.3.12}$$

where $-1 \leq k \leq 1$. Observe that the tapered superquadrics correspond to the original superquadrics when $k = 0$. Examples of tapered superquadrics are shown in the upper row of Figure 2.3.

To calculate the surface normal vector of deformable superquadrics, the formulas for inverse transformations and Jacobian matrices of the deformation functions are required. The inverse transformation $D_t^{-1}$ of the tapering deformation $D_t$ is simply calculated by:

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \longmapsto \mathbf{x} = \begin{bmatrix} \frac{1}{t_k(Z)}X \\ \frac{1}{t_k(Z)}Y \\ Z \end{bmatrix}. \tag{2.3.13}$$

The Jacobian matrix $\partial D_t/\partial \mathbf{x}$ of the tapering deformation $D_t$ is obtained by:

$$\frac{\partial D_t}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} t_k(z) & 0 & \frac{\partial t_k}{\partial z}(z)x \\ 0 & t_k(z) & \frac{\partial t_k}{\partial z}(z)y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3.14}$$

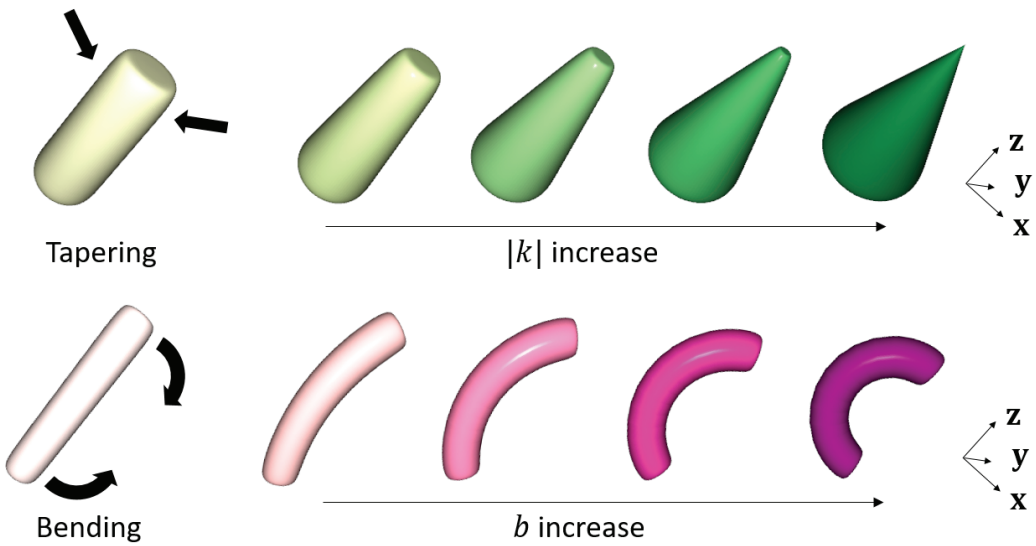


Figure 2.3: Examples of tapered superquadrics (upper row, Chapter 2.3.1), and bent superquadrics (lower row, Chapter 2.3.2).

### 2.3.2 Bending Deformation

Among the many possible bending deformations (e.g., parabolic, V-shaped), we will mostly rely on the parametric bending deformation $D_b$ that bends the $z$-axis into a circular section via the deformation

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \longmapsto \mathbf{X} = \begin{bmatrix} x + (R - r)\cos\alpha \\ y + (R - r)\sin\alpha \\ (b^{-1} - r)\sin\gamma \end{bmatrix}, \tag{2.3.15}$$

where

$$\begin{aligned} \gamma &= zb, \\ r &= \cos(\alpha - \operatorname{atan2}(y, x))\sqrt{x^2 + y^2}, \\ R &= b^{-1} - (b^{-1} - r)\cos\gamma. \end{aligned} \tag{2.3.16}$$

Here $b > 0$ and $\alpha$ respectively represent the degree and direction of bending in the *x-y* plane. The deformable superquadric converges to the original superquadric as $b$ approaches zero. Examples for bent superquadrics are shown in the lower row of Figure 2.3.

The inverse transformation $D_b^{-1}$ of the bending deformation $D_b$ is calculated by:

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \longmapsto \mathbf{x} = \begin{bmatrix} X - (R' - r')\cos\alpha \\ Y - (R' - r')\sin\alpha \\ \frac{\gamma'}{b} \end{bmatrix}, \tag{2.3.17}$$

where

$$\begin{aligned} R' &= \cos(\alpha - \operatorname{atan2}(Y, X))\sqrt{X^2 + Y^2}, \\ r' &= b^{-1} - \sqrt{(Z^2 + (b^{-1} - R')^2}, \\ \gamma' &= \operatorname{atan2}(Z, b^{-1} - R'). \end{aligned} \tag{2.3.18}$$

We note that the values $(r, R, \gamma)$ for the transformation $D_b$ and the $(r', R', \gamma')$ for the inverse transformation $D_b^{-1}$ have the same meaning, i.e., they are the same values, but expressed in different coordinates. The Jacobian matrix $\partial D_b / \partial \mathbf{x}$ is obtained by:

$$\frac{\partial D_b}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} 1 + r_x(\cos\gamma - 1)\cos\alpha & r_y(\cos\gamma - 1)\cos\alpha & (1 - rb)\sin\gamma\cos\alpha \\ r_x(\cos\gamma - 1)\sin\alpha & 1 + r_y(\cos\gamma - 1)\sin\alpha & (1 - rb)\sin\gamma\sin\alpha \\ -r_x\sin\gamma & -r_y\sin\gamma & (1 - rb)\cos\gamma \end{bmatrix}, \tag{2.3.19}$$

where

$$\begin{aligned} r_x &= \frac{-y\sin(\alpha - \operatorname{atan2}(y, x)) + x\cos(\alpha - \operatorname{atan2}(y, x))}{\sqrt{x^2 + y^2}}, \\ r_y &= \frac{x\sin(\alpha - \operatorname{atan2}(y, x)) + y\cos(\alpha - \operatorname{atan2}(y, x))}{\sqrt{x^2 + y^2}}. \end{aligned} \tag{2.3.20}$$

### 2.3.3 Combined Deformation

Tapering and bending can be concatenated in the obvious way, i.e., $D = D_b \circ D_t$ (note that the two deformations are not commutative, i.e., $D_b \circ D_t \neq D_t \circ D_b$). Applying $D$ to a superquadric surface $f(\mathbf{x}) = 1$, the implicit equations for a deformable superquadric surface are of the form $f \circ D^{-1}(\mathbf{X}) = 1$; we denote this implicit surface equation by $f_D \equiv f \circ D^{-1}$.

Finally, the surface normal vector $\mathbf{N}$ of the deformable superquadric surface $f \circ D^{-1}(\mathbf{X}) = 1$ at a point $\mathbf{X} = D(\mathbf{x})$ can be calculated as follows:

$$\mathbf{N}(\mathbf{X}) = \det\left(\frac{\partial D}{\partial \mathbf{x}}\right) \frac{\partial D}{\partial \mathbf{x}}^{-T} \mathbf{n}(\mathbf{x}), \tag{2.3.21}$$

where $\mathbf{n}(\mathbf{x})$ is the surface normal vector of the superquadric surface $f(\mathbf{x}) = 1$ at the point $\mathbf{x}$. The Jacobian matrix of the combined deformation $\partial D/\partial \mathbf{x}$ can be calculated by:

$$\frac{\partial D}{\partial \mathbf{x}}(\mathbf{x}) = \frac{\partial D_b}{\partial \mathbf{x}}(D_t(\mathbf{x})) \frac{\partial D_t}{\partial \mathbf{x}}(\mathbf{x}). \tag{2.3.22}$$

# 3

# DSQNet: Deformable Superquadric Network

## 3.1 Introduction

The problem of grasping known objects for which prior CAD models are available is by now well-understood, with several reliable and efficient model-based solutions deployed in structured manufacturing settings (see, e.g., [1, 2]). In practice, however, especially in many of today's warehouse logistics and automation environments, a far more prevalent scenario is the need to grasp, in real-time, unknown objects that are seen for the first time by the robot (or more specifically, for which prior CAD models of the objects are not available) that may be only partially visible because of, e.g., occlusions.

Spurred in part by the great success of deep learning networks in the recognition and detection of objects in images, recently there has been considerable interest in applying deep learning networks to the problem of grasping partially visible unknown objects. Existing approaches can be roughly divided into those that are *end-to-end*, in the sense

of training a deep neural network to generate a grasp pose for an object directly from the raw vision data input, [7, 8, 9, 10, 11, 12, 13, 14, 15], and *two-step methods* (or alternatively, *recognition-based methods*) that first attempt to recognize the object shape by matching the raw vision data to a set of predefined shape primitives [45, 46, 47, 54, 48, 59, 49, 68, 50] or general 3D representation such as voxel-valued or implicit functions representations [51, 58, 60, 69, 70, 71], and then to generate a grasp pose based on the object geometry. In particular, this chapter focuses on shape primitive-based two-step grasping methods.

Despite the widely reported successes of end-to-end methods, these impressive results are usually obtained at significant cost and effort. End-to-end methods typically require very large data sets, on the order of hundreds of thousands to millions of vision sensor data-grasp pose pairs. Physics-based simulations are widely used to augment the experimentally obtained real data, but as is well-known, the accuracy of any physics-based simulator, particularly when multiple contacts with friction are involved, remains questionable at best. Not surprisingly, network training also can be very time-consuming and computationally expensive for such large data sets. Finally, and perhaps most critically, the trained the network will only work reliably for the gripper used to collect the training data, or those that are very similar in design; different gripper designs will inevitably require some level of re-training with newly collected data. We refer to the survey paper for comprehensive reviews for the end-to-end grasping methods [16].

Recognition-based methods overcome many of the limitations of end-to-end methods, at the cost of relying on object shape models that may lack sufficient generality. Large data collection and expensive network training of end-to-end methods are now replaced by an object shape recognition step, which requires analysis and additional

real-time computation based on certain assumptions made about the objects. Some approaches require no training data sets at all [44, 45, 46, 47, 48, 49, 50] or considerably less data [54, 59, 68]. Since recognition and grasp pose generation are decoupled, recognition-based methods can be used for a wider range of grippers simply by modifying the grasp pose generation module.

The main limitations of existing recognition-based methods are, not surprisingly, in the accuracy and overall performance of the object recognition. Most recognition methods rely on the use of shape primitives, e.g., boxes, spheres, cylinders, and other standard shapes, with which even simple everyday objects such as bottles and mugs often cannot be easily represented. For example, even the most expressive among currently used shape primitives, the superquadric [44, 49], has difficulty in capturing the mug of Figure 3.1(a).

A more critical limitation of existing recognition-based methods is that they typically involve a computation-intensive optimization as an intermediate step, in the form of optimally fitting a given shape primitive to a set of partially observed point cloud data. The optimization usually cannot be performed in real-time. Worse, the local optima obtained sometimes bear no relation to the actual object: because only partial point cloud data are fitted, the fitted primitives can often miss entirely the occluded parts of the objects (see Figure 3.1(b) for the case of a box). Techniques like point cloud mirroring [49] have had some limited success in overcoming some of these deficiencies, but mostly for simple symmetric objects like boxes, cylinders, and spheres.

In this paper, we directly address the two fundamental limitations of current recognition based grasp generation methods. First, we employ a richer class of shape primitives, the *deformable superquadrics* [63]. The deformable superquadrics are parametrized through eight continuous parameters and can express a more varied range of shapes (see Figure 3.1(a) and 2.3). Closed-form surface equations are also available [61, 62, 63],

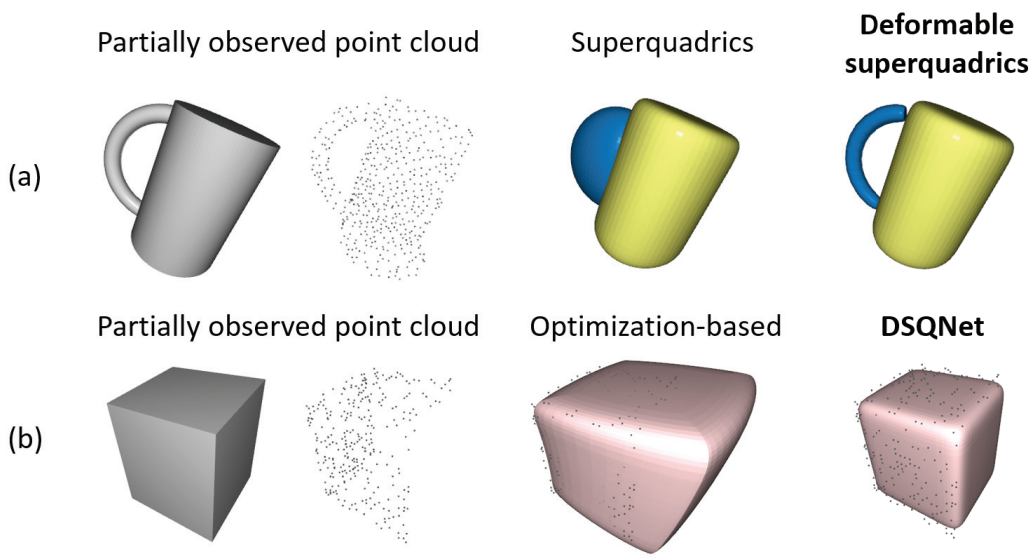


Figure 3.1: (a) Shape recognition results for superquadric versus deformable superquadric primitives: the latter can capture the handle of the mug, whereas the former cannot. (b) Shape recognition results using optimization-based fitting versus DSQNet: the latter is able to capture the occluded parts of the box that are missed by conventional methods.

which can, in turn, be used to efficiently calculate point-to-surface distances for fitting. We construct a set of deformable superquadric primitives targeted for grasping applications that can be easily concatenated and balances expressiveness with efficiency.

The second, and more significant, contribution is that rather than attempt to optimally fit a deformable superquadric surface to a set of partially observed point cloud data, we instead develop a supervised learning algorithm for this purpose. Specifically, we design a neural network architecture – we refer to the network as the *Deformable Superquadric Network (DSQNet)* – that takes as input the partially observed point cloud data, and outputs the eight parameters and the pose of the deformable superquadric so that it matches the full shape of the object, including the occluded parts (see Figure 3.1(b)). To train DSQNet, a dataset consisting of pairs of a ground-truth point cloud and a corresponding partially observed point cloud is constructed: (i) a set of synthetic primitive object shapes are generated, and (ii) for each primitive shape, a ground-truth point cloud is uniformly sampled, and a partially observed point cloud is rendered using a simulated depth camera. The network is then trained to minimize fitting errors between the ground-truth point cloud and the predicted deformable superquadric.

Our recognition-based method proceeds in three steps as shown in Figure 3.2: (i) a trained segmentation network – for our purposes we use the DGCNN [72] network – is used to segment a partially observed point cloud into a set of simpler point clouds; (ii) The trained DSQNet converts each point cloud into a deformable superquadric primitive, with its collective union representing the full object shape; (iii) grasp poses are generated in a gripper-dependent manner from the recognized full shapes. For a two-finger gripper, a sampling-based grasping algorithm that exploits the closed-form surface equations for deformable superquadrics to find antipodal grasp points is used.

Our approach retains the many advantages of recognition-based methods, and at

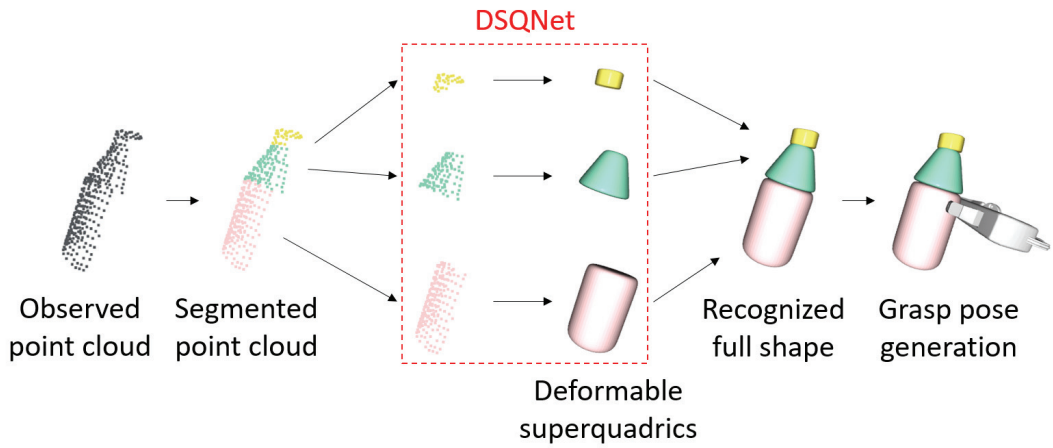


Figure 3.2: Pipeline for proposed recognition-based grasping algorithm.

the same time exploits the learning framework used in end-to-end approaches to precisely address the two critical limitations of existing recognition-based methods (real-time recognition, and identifying object shapes from partial point cloud data). Recognition is performed quickly and accurately with a simple forward pass of the neural network. Using a range of everyday objects, extensive experiments and benchmark comparisons against existing methods highlight both the strengths, and also potential areas of improvement, of our approach. On recognizing household objects, our method achieves the highest accuracy (with respect to volumetric IoU) and fastest computation speeds among existing recognition-based methods. Conducting over 150 trials of physical grasping experiments involving 15 standard household objects, our grasping method shows a 93% success rate, outperforming current state-of-the-art methods by over 7%.

## 3.2 Related Works

### 3.2.1 Optimization-based Recognition Methods for Grasping

Optimization-based recognition methods predict the full shape of the object, including occluded parts, under the assumption that most everyday objects are combinations of simple shapes (e.g., box). Another underlying reason for this assumption is that grasp poses can be quickly and easily generated for such objects.

**Optimization-based recognition using shape primitives.** A common feature of optimization based methods is that they attempt to find a set of shape primitives that best fit the partially observed point cloud data. The set of pre-defined geometric shape primitives can be bounding boxes [45], shapes consisting of cylinders, boxes and spheres [46, 47], or some customized shape primitive sets [48]. It is important to note that these simple shape primitives cover only a limited subset of possible shapes; for example, ellipsoids and cones cannot be easily represented using these existing shape primitives.

**Using superquadrics beyeond using pre-defined shape primitives.** Several studies have used superquadric shape primitives as one possible remedy. Using five continuous parameters, superquadrics can represent a wider range of shapes, and also have closed-form parametric equations that can be used to efficiently fit the primitive to the given point cloud data [61, 62]. Previous works that use superquadrics to fit target objects for grasp planning include [44, 73]. More recently, several works have focused on superquadric fitting of unknown objects based on depth sensor data [74, 49, 75, 76]. [50] predicts a superquadric representation of a local part of the target object. It bears repeating that superquadrics are still not expressive enough to represent common shapes of everyday objects, e.g., cones, or handles of cups and mugs. One of the contributions of this paper is the application of a richer set of shape primitives, the deformable superquadrics [63].

**Heuristics for optimization-based recognition.** Optimization-based methods for occluded object recognition are almost always accompanied by a set of heuristics designed to detect, e.g., symmetries in the partially observed point cloud, for example extrusion detection [74], symmetric plane detection [77], or point cloud mirroring [49]. These existing methods are only applicable to box-, cylinder-, or sphere-like objects with simple symmetry; extension to more complex object shapes is not straightforward. There have been attempts to capture shape uncertainty of more complex shapes using Gaussian processes [78], but these methods also have difficulties when large parts of the object are occluded.

### 3.2.2 Learning-based Recognition Methods for Grasping

To address the limitations of shape optimization methods, learning-based recognition methods that rely on deep neural networks have been proposed. Unlike optimization-based methods, learning-based methods do not rely on heuristics or simplifying assumptions; rather, they rely on experiential training data to predict the occluded parts of the objects. Another important advantage of learning-based methods is that real-time recognition is possible with just a simple forward pass of the neural network.

**Learning object shapes using general 3D representation.** One class of learning-based methods predicts the full shape of the object by using general 3D representation such as voxel-valued or implicit function representations. [51] trains a 3D convolutional neural network to predict the occupancy grids (a type of voxel-valued representation). [58] learns a parametric object implicit function (more specifically, a neural network that takes a 3D query point as input and outputs signed distance from the object surface). [60] proposes a network that predicts a voxel-valued representation of a local part of the target object. [69] developes a method for generating a 3D voxel grid directly from

RGB-D images. [70] integrates uncertainty into their shape completion network in which the network predicts the likelihood of accuracy for each point in the generated model. [71] presents a technique for predicting the depth image of an object's 'back' side using a masked depth image. This process allows for the rapid combination of the front and back sides to create a complete object mesh. These methods all share the disadvantage of a time-consuming planning stage to generate feasible grasp poses. For example, the recognized object mesh is fed into the computationally expensive GraspIt! or used to perform a time-consuming grasp quality optimization for these high-dimensional object representations.

**Jointly learning 3D recognition and grasp planner.** Recently, various approaches integrate a grasp generation network with a shape completion module to expedite the process of grasp pose generation. [79] uses a dual-network strategy, involving one network for shape completion and another for predicting grasp outcomes. They observe enhanced performance in the grasp prediction network when it utilizes the feature space created by the shape completion network. [80] and [81] implement a shape completion network that processes voxelized point cloud inputs. This network's outputs are then employed to concurrently train a network focused on refining grasp poses. [82] emphasizes the interconnection between 3-D reconstruction and grasping and accordingly adopt a self-supervised method to reconstruct an object and determine a suitable grasp. [83] focuses on simultaneously regressing a grasp pose while also reconstructing an object's point cloud.

**Learning object shapes via 3D shape primitives.** Other works have attempted to use neural networks to recognize object shapes by predicting a set of shape primitives rather than their general 3D representations [54, 59, 68]. Like optimization-based methods, these methods can efficiently generate grasp poses. At the same time, they can recognize objects quickly and directly without the use of heuristics. [54] uses a deep

neural network to output shape primitives consisting of cylinders, boxes, or spheres to predict the full object shapes. The more recent [59] and [68] use pre-defined finite shape templates as primitives: the point cloud is first segmented into simpler point clouds using a trained deep network, after which each segment is recognized using the best-fitting shape template.

Our work is also in the spirit of [54, 59, 68] in merging learning methods with shape primitives. Because the shape primitives used in existing methods are not expressive enough to capture a large class of common everyday objects, we propose a deep learning based recognition framework that uses the more expressive deformable superquadrics as shape primitives.

## 3.3 Deformable Superquadric Network

We now describe the Deformable Superquadric Network (DSQNet), a deep neural network that takes a *partially observed point cloud* $\mathcal{P} := \{\mathbf{x}_i \in \mathbb{R}^3\}_{i=1}^N$ as input, and outputs the deformable superquadric primitive that best represents the full object shape. $\mathcal{P}$ is obtained by observing an object from a (synthetic or real-world) depth camera. Specifically, DSQNet predicts the following parameters (see Chapter 2.2): size $\mathbf{a} = (a_1, a_2, a_3) \in \mathbb{R}^3$, shape $\mathbf{e} = (e_1, e_2) \in \mathbb{R}^2$, tapering coefficient $k \in \mathbb{R}$, bending coefficients $b \in \mathbb{R}$ and $\alpha \in \mathbb{S}^1$, and a pose $\mathbf{T} \in \mathrm{SE}(3)$ that arbitrarily translates and rotates the canonical surface $f_D(\mathbf{x}) = 1$ in three-dimensional space. Using the predicted parameters, the surface equation for the deformable superquadric primitive at the pose $\mathbf{T}$ can be written as follows:

$$f_D(\mathbf{T}^{-1}\mathbf{x}) = 1. \tag{3.3.1}$$

Note that DSQNet is not trained to merely fit the partially observed point cloud, but rather the *ground-truth point cloud* $\mathcal{P}_g := \{\mathbf{x}_{g,i} \in \mathbb{R}^3\}_{i=1}^{N_g}$, which is obtained by uniformly sampling the surface of the object. In this regard, the objective of the DSQNet is to achieve the approximate equality

$$f_D(\mathbf{T}^{-1}\mathbf{x}_{g,i}) \approx 1 \quad \text{for all} \quad i = 1, ..., N_g. \tag{3.3.2}$$

Below we discuss further requirements on the network architecture and loss function for training.

### 3.3.1 DSQNet Architecture

The neural network architecture for DSQNet is shown in Figure 3.3. The first requirement of DSQNet is that it should be permutation-invariant, i.e., the output should not depend on how the input point cloud data are ordered. Among existing permutation-invariant networks, we adopt the EdgeConv layers from the Dynamic Graph Convolution Neural Network (DGCNN) [72]. The input point cloud $\mathcal{P}$ passes through five EdgeConv layers with point-wise latent space dimensions (64, 64, 128, 256) and a max pooling layer, producing a 1024-dimensional global feature vector that captures semantic information about $\mathcal{P}$ in a permutation-invariant manner.

The global feature vector is then passed through additional networks (described below) before producing the output deformable superquadric parameters: size $\mathbf{a}$, shape $\mathbf{e}$, tapering $k$, bending $(b, \alpha)$, and pose $\mathbf{T}$.

- A lower bound of 0.2 is imposed on each of the shape parameters $\mathbf{e} = (e_1, e_2) \in \mathbb{R}^2$ to prevent Equation (2.2.1) from diverging as $e_1$ or $e_2$ goes to zero. Also, to prevent the shapes from becoming overly complex, e.g., non-convex shapes that are less likely to occur in practice, an upper bound of 1.7 is imposed.

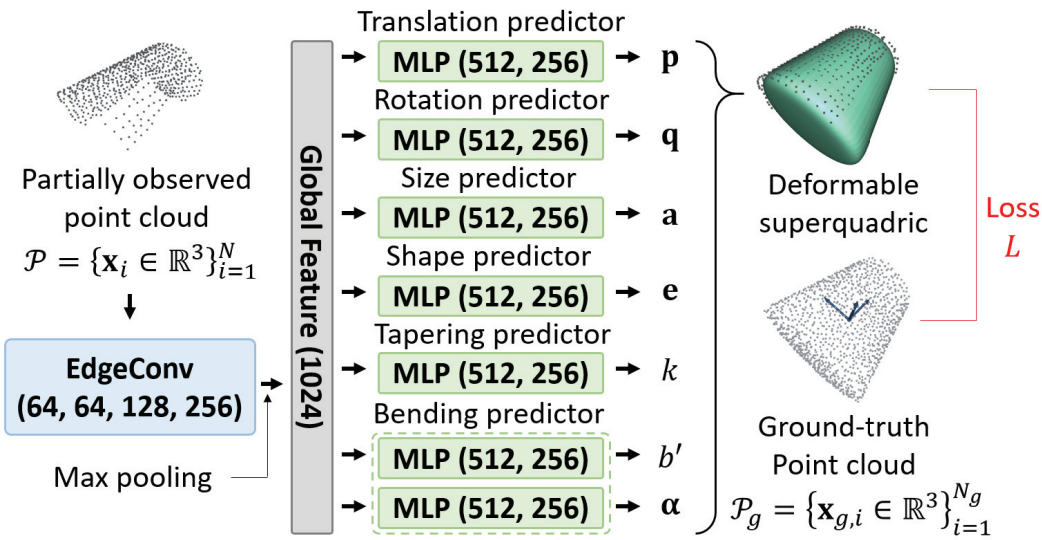


Figure 3.3: DSQNet architecture.

Table 3.1: Network configuration for each predictor

| **PREDICTOR** | Individual layers | Output | Constraints |
|---|---|---|---|
| Translation | (512, 256, 3) | $\mathbf{p}$ | - |
| Rotation | (512, 256, 4, normalize) | $\mathbf{q}$ | $\mathbb{S}^3$ |
| Size | (512, 256, 3, sigmoid) | $\mathbf{a}$ | [0.03,0.53] |
| Shape | (512, 256, 2, sigmoid) | $\mathbf{e}$ | [0.2,1.7] |
| Tapering | (512, 256, 1, sigmoid) | $k$ | [-0.9,0.9] |
| Bending | (512, 256, 1, sigmoid) | $b'$ | [0.01,0.75] |
| | (512, 256, 2, normalize) | $\boldsymbol{\alpha}$ | $\mathbb{S}^1$ |

- For the bending parameter $b \in \mathbb{R}$, instead of directly predicting $b$, we predict the value $b' := \max(a_1, a_2)b$, with $b'$ bounded to the interval $[0.01, 0.75]$ to prevent overfitting of irregular shapes.

- For the direction of bending $\alpha \in \mathbb{S}^1$, we predict the two-dimensional vector $\boldsymbol{\alpha} = (\cos\alpha, \sin\alpha)$ which is restricted to a unit circle $\mathbb{S}^1$ in $\mathbb{R}^2$.

- For the pose $\mathbf{T} = [\mathbf{R}; \mathbf{p}]$ consisting of the translation vector $\mathbf{p} = (p_x, p_y, p_z)$ and rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$, we use the unit quaternion representation $\mathbf{q} = (q_x, q_y, q_z, q_w)$ for $\mathbf{R}$, where $\mathbf{q}$ is restricted to be of unit norm.

The network architecture consists of seven fully-connected multilayer perceptrons (512, 256) followed by leaky ReLU nonlinearities. The result of each layer (a 256-dimensional feature vector) is then passed to a final linear layer and an additional non-linearity that outputs $\mathbf{p}, \mathbf{q}, \mathbf{a}, \mathbf{e}, k, b'$, and $\boldsymbol{\alpha}$, all while satisfying the constraints and bounds in Table 3.1.

**Point Cloud Preprocessing** Partially observed point cloud data of an object can have many different numerical representations depending on the reference frames used. We introduce a preprocessing technique that standardizes the point cloud numerical representation, reducing the complexity of the data statistics and thereby accelerating neural network training. For each partially observed point cloud data, we first apply principal component analysis and obtain orthogonal eigenvectors $v_1, v_2, v_3 \in \mathbb{R}^3$ with corresponding eigenvalues $\lambda_1 < \lambda_2 < \lambda_3$. The points are represented with respect to a frame whose origin coincides with the point cloud center of mass, with axes aligned along the principal axes $V = [v_2, v_1, v_3] \in \mathbb{R}^{3\times3}$. The length scale for the partially observed point cloud is then scaled so that the distance between the two furthest points is one. In the case of the primitive dataset, the same preprocessing procedure is also applied to the ground-truth point cloud.

### 3.3.2 Loss Function for Training

The loss function used to train DSQNet measures the fit between the ground truth point cloud $\mathcal{P}_g$ and the surface corresponding to the deformable superquadric parameters $\{a_1, a_2, a_3, e_1, e_2, k, b, \alpha\}$ and pose $\mathbf{T}$. Making use of Equation (3.3.2) for the distance between a point and a deformable superquadric surface, one obvious choice of loss function is

$$\frac{1}{N_g}\sum_{i=1}^{N_g} \|f_D(\mathbf{T}^{-1}\mathbf{x}_{g,i}) - 1\|^2, \tag{3.3.3}$$

where $f_D$ denotes the canonical surface equation for the deformable superquadric. In [84] it is shown that the above loss function can lead to deformable superquadric surfaces with large shape parameter $e_1$ or large volume $\sqrt{a_1 a_2 a_3}$. [84] remedies this by proposing the following distance between a point $\mathbf{x}_0 \in \mathbb{R}^3$ and a deformable superquadric surface $f_D(\mathbf{x}) = f \circ D^{-1}(\mathbf{x}) = 1$:

$$\delta(\mathbf{x}_0, f_D) = ||\mathbf{x}_0|| \left|1 - f^{-\frac{e_1}{2}} \circ D^{-1}(\mathbf{x}_0)\right|, \tag{3.3.4}$$

where $\|\cdot\|$ denotes the Euclidean norm. The loss function corresponding to this distance metric is

$$L = \frac{1}{N_g}\sum_{i=1}^{N_g} \delta^2(\mathbf{T}^{-1}\mathbf{x}_{g,i}, f_D). \tag{3.3.5}$$

To prevent overfitting and also to improve stability and convergence of the optimization, a regularization term can also be added. Since bending and tapering occur along the $z$-axis of the superquadric, motivated by the heuristic proposed in [63], we use the following loss function:

$$L = \frac{1}{N_g}\sum_{i=1}^{N_g} (\delta^2(\mathbf{T}^{-1}\mathbf{x}_{g,i}, f_D)) + w||\mathbf{z} \times \mathbf{z}_g||^2, \tag{3.3.6}$$

where $\mathbf{z}$ (i.e., the third column of the rotation $\mathbf{R}$) and $\mathbf{z}_g$ are respectively the predicted and ground-truth $z$-axes of the pose of deformable superquadric, and $w$ is a weighting

parameter (for our later experiments we set $w$ to 0.01).

## 3.4 Grasp Pose Generation Using Deformable Superquadric Primitives

We now describe how to generate feasible grasp poses given the recognized shape expressed as a set of deformable superquadric primitives. Once the shape of an object is recognized, for a given gripper we can apply conventional grasp pose generation techniques [1, 2]. Our focus in this paper will be on parallel jaw grippers; consequently we adopt an antipodal points sampling-based grasp pose generation method. We first describe the antipodal points sampling algorithm for deformable superquadric primitives, then describe our grasp pose generation algorithm using the sampled pairs of antipodal points.

### 3.4.1 Antipodal Points Sampling for Deformable Superquadric Primitives

The problem of finding antipodal points is defined as follows [85]: for a given surface $S$, find two points $\mathbf{x}_1, \mathbf{x}_2 \in S$ such that

$$\mathbf{n}(\mathbf{x}_1) + \mathbf{n}(\mathbf{x}_2) = \mathbf{0}, \tag{3.4.7}$$

$$(\mathbf{x}_1 - \mathbf{x}_2) \times \mathbf{n}(\mathbf{x}_1) = \mathbf{0}, \tag{3.4.8}$$

where $\mathbf{n}(\mathbf{x})$ is the normal vector to the surface at $\mathbf{x} \in S$. The antipodal sampling method is well-suited to deformable superquadric primitives, since surface normal vectors can be efficiently calculated using the closed-form Equation (2.3.21).

The pairs of antipodal points are sampled via the following steps. First, candidate points are uniformly sampled on the surfaces of the recognized deformable superquadric

primitives $\{f_{Dj}, \mathbf{T}_j\}_{j=1}^{n_p}$. Then, for each sampled point $\mathbf{x}_1$, solutions $\mathbf{x}^*$ to Equation (3.4.8) are obtained by finding the intersection points between a line $l(t) = \mathbf{x}_1 + \mathbf{n}(\mathbf{x}_1) \cdot t$, where $t \in \mathbb{R}$, and all deformable superquadric primitives. Finally, for every two points $\mathbf{x}_1, \mathbf{x}^*$, we verify a relaxed version of equation (3.4.7): $\mathbf{n}(\mathbf{x}_1) \cdot \mathbf{n}(\mathbf{x}^*) < -0.9||\mathbf{n}(\mathbf{x}_1)|| \cdot ||\mathbf{n}(\mathbf{x}^*)||$. Among all intersection points $\mathbf{x}^*$ that satisfy the above requirements, the one that is furthest from $\mathbf{x}_1$ – we denote this point $\mathbf{x}_2$ – is chosen to be the antipodal point to $\mathbf{x}_1$.

Finding the intersections between a line $l$ and a deformable superquadric $f_D(\mathbf{T}^{-1}\mathbf{x}) = 1$ is not trivial, since the equation $g(t) = f_D(\mathbf{T}^{-1} \cdot l(t)) - 1 = 0$ has no closed-form solution for $t \in \mathbb{R}$. Therefore, for each deformable superquadric primitive, we use a Newton-Raphson method to find the furthest intersection $\mathbf{x}^* = l(t^*)$ from $\mathbf{x}_1 = l(0)$ among the solutions of $g(t) = 0$ and $t > 0$. For initial guesses we use the intersections between the line $l(t)$ and a sphere centered at the deformable superquadric origin with radius is $\max(a_1, a_2, a_3)$, which can be obtained via closed-form solutions.

### 3.4.2 Grasp Pose Generation Algorithm

After the antipodal point pairs are sampled using the above algorithm, six-dof grasp poses are generated heuristically for each sampled antipodal point pair. For each pair, the gripper's approach vector is sampled at intervals of 30 degrees (i.e., twelve grasp poses for each pair). At the final stage, the algorithm checks for collisions and kinematic feasibility of the grasp pose. Among all grasp poses that satisfy the requirements, we choose a grasp pose whose approach vector is closest to the direction of gravity, to favor top-down grasping so as to ease the problem of finding a collision-free path. As a result we calculate the grasp score $c$ as the negative inner product between the normal vector of the table $\mathbf{z}_t$ and the grasp approach vector $\mathbf{z}$, i.e., $c = -\mathbf{z}^T\mathbf{z}_t$; the grasp pose with

the maximum $c$ is chosen. When there are multiple grasp poses with the maximum $c$ (e.g., multiple top-down grasp poses), we randomly select one grasp pose among these. Our grasp pose generation algorithm is summarized in Algorithm 1.

## 3.5 Experimental Results

We now describe experiments comparing (i) the baseline performance of DSQNet against other state-of-the-art methods for a range of shape recognition tasks, and (ii) the grasp success rates of our recognition-based grasping method against existing methods for grasping of real-world objects.

### 3.5.1 Fitting Using Only Partially Observed Point Clouds

In the first set of experiments, we confirm that optimization-based shape recognition methods – fitting only the partially observed point cloud to a deformable superquadric – do not fare well in predicting the full object shape, and verify that a supervised learning framework can greatly enhance the recognition performance.

We compare shape recognition results between DSQNet and the Deformable Superquadric Optimization (DSQOpt) method for a range of test sets of the primitive dataset. DSQOpt minimizes the Gross and Boult point-to-surface error metric between the partially observed point cloud and the deformable superquadric, i.e., equation (3.3.4), using the gradient descent method. For a fair comparison, DSQOpt uses the same ranges of deformable superquadric parameters as those for DSQNet.

As illustrated in Figure 3.4, both qualitative and quantitative experimental results confirm that DSQNet is more successful than DSQOpt in determining complete shapes of objects close to the ground-truth. For a quantitative analysis, we use the average Gross and Boult point-to-surface error (lower-the-better) and the average volumetric

**Algorithm 1** Grasp Pose Generation Using Deformable Superquadric Primitives

**Input:** Deformable superquadric primitives
$\{f_{Dj}, \mathbf{T}_j\}_{j=1}^{n_p}$.
**Output:** Grasp pose $\mathbf{g}_f \in \mathrm{SE}(3)$.
Uniformly sample points $\mathcal{P}_s := \{\mathbf{x}_{s,i} \in \mathbb{R}^3\}_{i=1}^{n_s}$
$\mathbf{g}_f = I_4,\ c_f = 0$
**for** $i = 1 : n_s$ **do**
    $E = \varnothing,\ \mathbf{x}_1 = \mathbf{x}_{s,i}$
    Compute $\mathbf{n}(\mathbf{x}_1)$ using (2.3.21)
    **for** $j = 1 : n_p$ **do**
        Find the solutions $t > 0$ of $f_{Dj}(\mathbf{T}_j^{-1} \cdot l(t)) = 1$
        $t^*$ = the maximum value among the solutions
        $\mathbf{x}^* = l(t^*)$
        Compute $\mathbf{n}(\mathbf{x}^*)$ using (2.3.21)
        **if** $\mathbf{n}(\mathbf{x}_1) \cdot \mathbf{n}(\mathbf{x}^*) < -0.9||\mathbf{n}(\mathbf{x}_1)|| \cdot ||\mathbf{n}(\mathbf{x}^*)||$ **then**
            $E \leftarrow E \cup \{\mathbf{x}^*\}$
        **end if**
    **end for**
    **if** $E \neq \varnothing$ **then**
        $\mathbf{x}_2$ = the furthest point in $E$ from $\mathbf{x}_1$
        Generate grasp poses $\{g_k\}_{k=1}^{12}$ for $(\mathbf{x}_1, \mathbf{x}_2)$
        **for** $k = 1 : 12$ **do**
            **if** kinematic feasible and not collide **then**
                $c = -\mathbf{z}^T \mathbf{z}_t$
                **if** $c > c_f$ **then**
                $c_f \leftarrow c,\ \mathbf{g}_f \leftarrow g_k$
                **end if**
            **end if**
        **end for**
    **end if**
**end for**

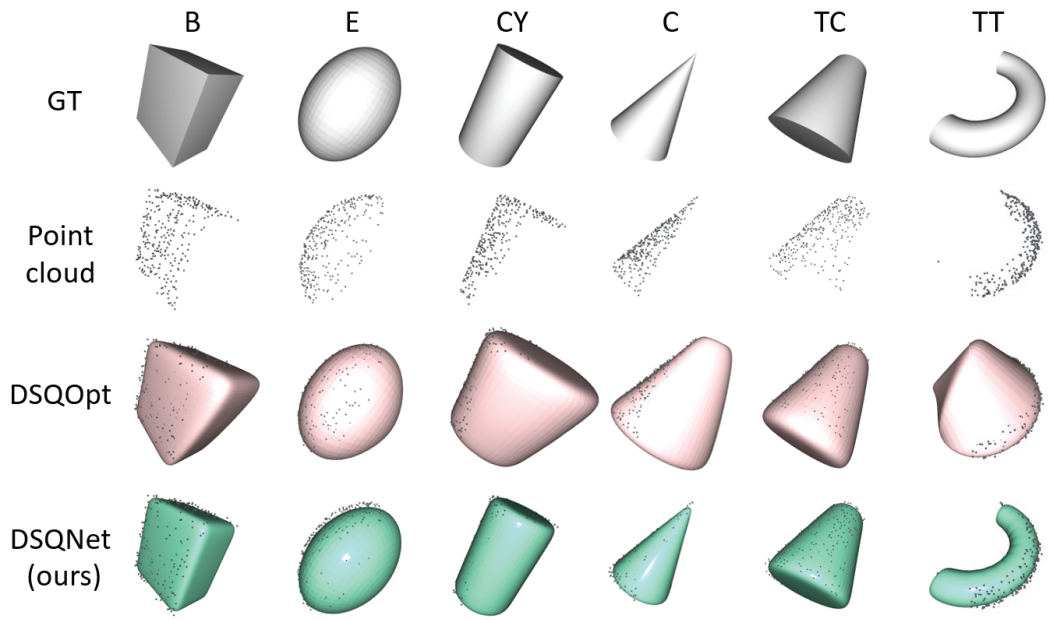


Figure 3.4: Shape recognition results for DSQNet and DSQOpt for six types of primitive data sets.

Table 3.2: Fitting error and IoU comparison between DSQNet and DSQOpt

| **METHOD** | B | E | CY | C | TC | TT |
|---|---|---|---|---|---|---|
| DSQOpt, error | **1.7e-4** | **1.8e-5** | **2.6e-4** | **2.5e-4** | **2.2e-4** | **9.1e-4** |
| DSQNet, error | 2.5e-3 | 7.1e-3 | 4.1e-3 | 3.5e-3 | 3.1e-3 | 1.3e-2 |
| DSQOpt, IoU | .4194 | **.8640** | .3924 | .2637 | .4116 | .1378 |
| DSQNet, IoU | **.8288** | .8423 | **.8575** | **.8394** | **.8292** | **.7584** |

intersection over union (IoU) measure between the predicted and ground-truth mesh (higher-the-better) as shown in Table 3.2. The partially observed point clouds fit well to both deformable superquadric surfaces predicted from DSQOpt and DSQNet. Although the point-to-surface error is much lower for DSQOpt than DSQNet for all primitive types, DSQNet achieves higher volumetric IoU, with an average of over 0.75. For DSQOpt, since the occluded parts of the objects are not fitted, the predicted shapes differ significantly over these regions, resulting in a poor IoU. In contrast, DSQNet predicts the complete shape close to the ground-truth, even over occluded regions, with the aid of ground-truth shape supervision, showing higher overall recognition performance. Somewhat unusually, the optimization-based method also successfully predicts the complete shape for the case of the ellipsoid.

### 3.5.2 Shape Recognition for Synthetic Objects

In the second set of experiments, we compare the performance of DSQNet against existing methods for shape recognition tasks. For the baseline methods, we use the minimum volume bounding box approach (MVBB) and Primitive Shape CNN (PS-CNN), which is the recognition method of the state-of-the-art recognition-based grasping methods. We also compared the performance with the regular *Superquadric Network (SQNet)* which is identical to DSQNet but uses regular (i.e., non-deformable) superquadrics.

The two baselines, SQNet, and DSQNet all predict full shapes of the segmented partially observed point clouds, which are obtained using our segmentation network. The minimum volume bounding box (MVBB) methods finds a minimum volume bounding box for each segmented point cloud using principal component analysis [86]. The bounding box approach has been successfully applied to certain recognition tasks [45], thus we compare the performance of MVBB as a baseline.

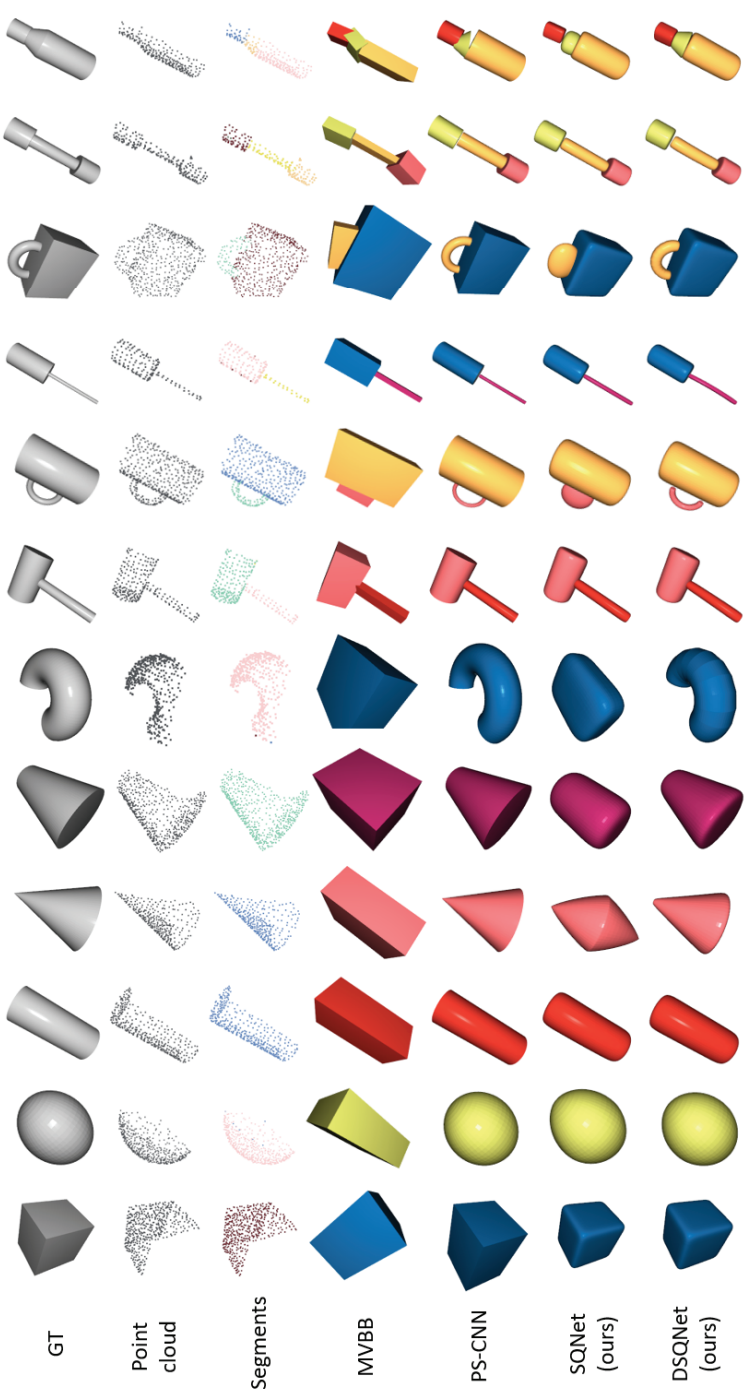


Figure 3.5: Shape recognition results for MVBB, PS-CNN, SQNet, and DSQNet for twelve types of object data sets.

Table 3.3: IoU comparison between MVBB, PS-CNN, SQNet, and DSQNet for object dataset

| **OBJECT** | MVBB | PS-CNN | SQNet (ours) | DSQNet (ours) |
|---|---|---|---|---|
| B | .3795 | .6442 | .8517 | **.8759** |
| E | .3026 | .7429 | .8483 | **.8666** |
| CY | .5283 | .7988 | .8903 | **.8939** |
| C | .3065 | .5946 | .5421 | **.8039** |
| TC | .4448 | .7504 | .7340 | **.8264** |
| TT | .3546 | .6141 | .3691 | **.6759** |
| Hammer | .5293 | .8101 | **.8358** | .8208 |
| Mug | .4666 | .8282 | .7786 | **.8483** |
| Screwdriver | .5535 | .8346 | .8631 | **.8655** |
| Padlock | .4343 | .6751 | .8182 | **.8312** |
| Dumbbell | .4367 | **.7976** | .7589 | .7017 |
| Bottle | .4045 | .7610 | .8120 | **.8189** |
| **Average** | .4284 | .7376 | .7588 | **.8191** |

PS-CNN is our customized implementation of [59], in which objects are recognized using predefined shape templates consisting of finite shape primitives. Object shapes are recognized by first segmenting the partially observed point cloud into simple point clouds. Each segmented point cloud is then fitted to one of the shape templates using the lowest fitting score for the Iterative Closest Point (ICP) algorithm. For a fair comparison, we use a shape template consisted of 100 uniformly sampled shape parameters for each primitive type (i.e., a total of 600 shapes). We modify our segmentation network to provide primitive type information, since the original work predicts the shape type at the segmentation stage.

We compare all methods both qualitatively and quantitatively for the test sets of the object dataset. DSQNet shows the best shape recognition performance (volumetric IoU) as shown in Figure 3.5 and Table 3.3. MVBB fails to recognize full shapes even for boxes (B), and shows significantly lower IoU values compared to other methods. Recognition performance of PS-CNN is moderately better but limited by the fact that the shape template contains only a finite number of shapes. SQNet successfully recognizes most of the shapes, but fails to recognize shapes that involve deformations (i.e., C, TC, and TT). Among the four algorithms, DSQNet performs the best for most objects, with an average volumetric IoU of 0.8191. For the dumbbell, DSQNet is outperformed by SQNet and PS-CNN; this can be attributed to both ends of the dumbbell being recognized as a truncated torus, since the segmented point clouds at both ends have holes.

We also confirm that DSQNet shows the best performance in terms of both volumetric IoU and recognition speed. Figure 3.6 shows a graph of average volumetric IoU versus average recognition time for all objects. For PS-CNN, ten different experiments with a range of shape templates are plotted; from left to right, the number of shape templates of each primitive type is increased from 10 to 100 in increments of 10.

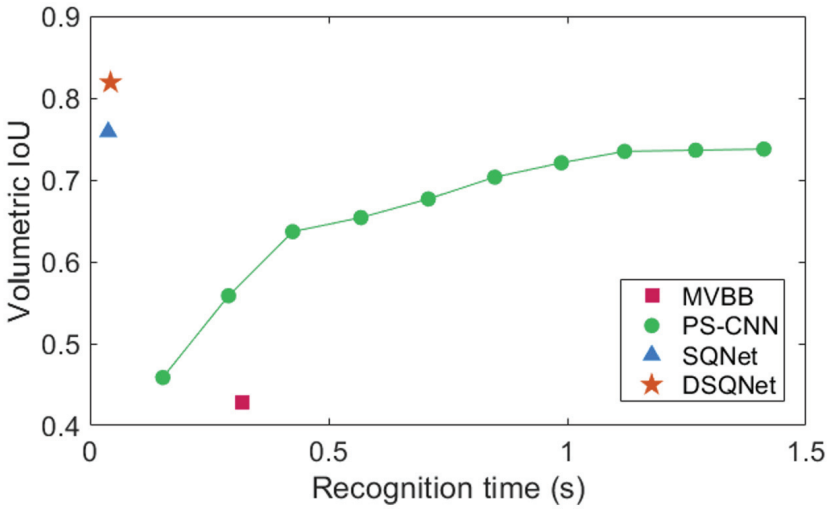


Figure 3.6: Graph of volumetric IoU versus recognition time (calculation time) for MVBB, PS-CNN, SQNet, and DSQNet.

For PS-CNN, a trade-off exists between recognition speed and volumetric IoU; for PS-CNN to achieve high recognition performance, the size of shape templates need to be increased at the cost of longer computation time. MVBB takes 0.318 seconds, similar to that of PS-CNN with 20 shape templates. SQNet and DSQNet are the fastest, taking only 0.038 and 0.043 seconds, respectively, while still showing the best recognition performance.

### 3.5.3 Recognition and Grasping on Real-world Objects

In our final set of experiments, we evaluate the performance of our grasping method against existing methods for real-world objects. Specifically, we compare the performance of four recognition-based grasping methods that rely on different recognition methods (MVBB, PS-CNN, SQNet, and DSQNet), while using the same antipodal points

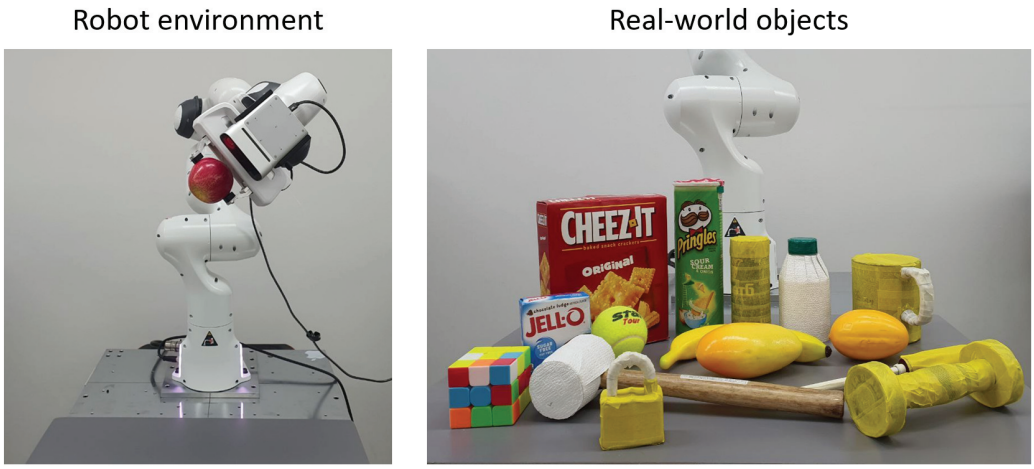


Figure 3.7: Robot manipulator equipped with the vision sensor (left) and real-world objects used in the grasping experiments (right).

sampling-based method for the grasp generation step. For each method, ten trials are conducted on different poses for each object, resulting in a total 150 trials.

We use a set of 15 household objects inspired from the YCB dataset [87] as shown in Figure 3.7. For the robot arm and vision sensor, the seven-DOF Franka Emika Panda robot with a parallel-jaw gripper and an Azure Kinect DK camera sensor mounted on the gripper are used in our experiments. From the vision sensor data, the object point clouds are obtained by discarding points of the table through plane fitting, and then up/down-sampled to 1000 points. After the object point cloud is segmented via the segmentation network, each segmented point cloud is also up/down-sampled to 300 points. We down-sample points using the voxel down-sampling method, and up-sample points by sampling points in the local tangent planes of the observed points [88].

To find feasible grasp poses from the grasp pose candidates, we use the FCL library [89] to check whether a grasp pose collides with the table or the recognized

objects, and also solve the inverse kinematics to determine whether a grasp pose is kinematically feasible. After a grasp pose is determined, collision-free trajectory planning is performed using the Planning Scene module of MoveIt! [90]. Then the gripper is moved to the final grasp pose and closed until contact is detected.

Grasping performance results are shown in Table 3.4. Our proposed DSQNet approach outperforms other recognition-based methods, with an average grasping success rate of 93% across all objects. We believe the high grasping success rate of can be attributed to the high recognition performance of DSQNet. DSQNet is capable of accurate shape recognition of real-world objects, even though it is trained with only synthetic data. Although a precise quantitative assessment of recognition performance for real-world objects is difficult, our experiments confirm empirically that our method recognizes shapes of real-world objects that are close to the actual shapes as shown in Figure 3.8. Accurate shape recognition aids robotic grasping by ensuring that the antipodal points found in the recognized shape correspond to antipodal points on the actual object.

Examining in more detail the causes behind grasping failure cases, the first case is the result of a failure to achieve grasp closure; these can be traced to inaccurate shape recognition caused by noisy vision sensing. The imperfect matching in shape with the actual object can lead the robot to grasp air, or to collide with objects. The second case prevalent is lifting failure, i.e., when the grasp pose cannot support the weight of the object. Such cases occur when the the object is too heavy (for example, in the case of the dumbbell) or when the grasp points are distant from the object center of mass (for example, in the case of the hammer or mug). The analysis of failure cases suggests that grasping performance can be improved by (i) using a more accurate vision sensor, or more precise sensor noise models, to generate synthetic datasets, and (ii) finding grasp poses capable of supporting the object weight by taking into account density prediction,


Figure 3.8: Recognition and grasp pose generation results for real-world objects.

Table 3.4: Real-world object grasping results

| **OBJECT** | MVBB | PS-CNN | SQNet (ours) | DSQNet (ours) | DSQPose |
|---|---|---|---|---|---|
| Cheez-it | 7/10 | 9/10 | 10/10 | 10/10 | 10/10 |
| Jello | 8/10 | 6/10 | 9/10 | 9/10 | 10/10 |
| Cube | 1/10 | 9/10 | 9/10 | 9/10 | 10/10 |
| Pringles | 7/10 | 10/10 | 10/10 | 9/10 | 9/10 |
| Can | 6/10 | 10/10 | 10/10 | 9/10 | 10/10 |
| Tennis ball | 7/10 | 10/10 | 10/10 | 10/10 | 10/10 |
| Pumpkin | 6/10 | 10/10 | 10/10 | 10/10 | 10/10 |
| Mango | 6/10 | 10/10 | 10/10 | 10/10 | - |
| Banana | 10/10 | 9/10 | 4/10 | 10/10 | 10/10 |
| Padlock | 8/10 | 7/10 | 4/10 | 9/10 | 10/10 |
| Hammer | 4/10 | 7/10 | 9/10 | 9/10 | 9/10 |
| Mug | 4/10 | 7/10 | 3/10 | 9/10 | 9/10 |
| Bottle | 7/10 | 9/10 | 7/10 | 9/10 | 9/10 |
| Dumbbell | 1/10 | 6/10 | 9/10 | 8/10 | 8/10 |
| Screwdriver | 7/10 | 10/10 | 10/10 | 9/10 | 10/10 |
| **Average** | 59% | 86% | 83% | **93%** | 96% |

or the center of mass, of the objects.

Our recognition-based method also achieves comparable grasping performance compared to the 80-90% success rates reported in many current end-to-end methods [7, 8, 9, 10], while having the advantages of requiring only a small number of training data, and being applicable to a diverse range of grippers compared to end-to-end methods. While the precise test objects may differ from those used in our experiments, the similarity in the overall object classes lends credibility to the conclusions drawn from our experiments.

To comprehensively evaluate the success rates of these recognition-based methods, we have also implemented and assessed a method named *DSQPose*. DSQPose is essentially a pose estimation-based methodology that utilizes ground-truth object shape information. We first measure the absolute sizes (e.g., the radius and height of a cylinder) of the objects and represent them almost accurately with a set of deformable superquadrics. During the recognition phase, each object's pose is estimated by fitting the partially observed point cloud to the ground-truth shape using the Iterative Closest Point (ICP) algorithm. We use the same grasp pose generation algorithm to the obtained objects' poses and ground-truth shapes as described in Chapter 3.4. The results of DSQPose lead to the following discussions: Firstly, DSQPose demonstrates a high success rate of 96%, suggesting that the deformable superquadric shape primitive itself offers a significant advantage in the task of grasping. Secondly, while DSQPose naturally outperforms all other recognition-based methods due to its use of ground-truth shape information, DSQNet shows a grasping success rate that is nearly comparable to DSQPose. Lastly, even though the objects' shapes are nearly accurately recognized, DSQPose cannot achieve a 100% success rate with certain objects like dumbbells and hammers. This is attributed to the fact that the mass distribution of the objects is not considered, as mentioned earlier. This implies that there is a need to develop a grasping algorithm that

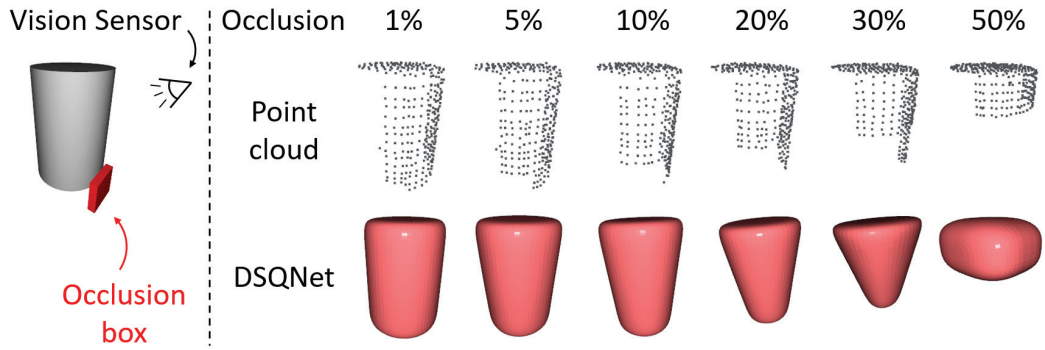


Figure 3.9: Shape recognition results for DSQNet for a cylinder with various occlusion ratios.

accounts for the mass distribution of objects.

## 3.6 Additional Experimental Results

### 3.6.1 Performance of DSQNet with Additional Occlusion

We also explore how DSQNet's shape recognition performance behaves when additional occlusions are caused by other objects. In typical real-world scenarios involving cluttered environments, the target object is often occluded by other objects. We therefore perform experiments to evaluate the robustness of DSQNet against additional occlusions that DSQNet has never experienced.

We first introduce a virtual thin occlusion box (square-shaped), and generate occluded point clouds of the objects as shown in the left of Figure 3.9. The *occlusion ratio* of a point cloud is then defined as the ratio of the number of points occluded by the occlusion box to the total number of points. We note that each point cloud is up/down-sampled to 300 points after occlusion. For each object, we randomly sample the position and size of the occlusion box so that different occluded point clouds can

be obtained. We use the 10 test sets for each single-primitive object (i.e., B, E, CY, C, TC, and TT) from the object dataset in our earlier the occlusion experiment.

The right of Figure 3.9 shows an example of shape recognition results for various occlusion ratios. As the occlusion ratio increases, the object's point cloud becomes more incomplete, and accordingly, the prediction of DSQNet increasingly deviates from the ground-truth shape. However, although DSQNet has never experienced such point clouds, the predicted shapes qualitatively resemble the ground-truth shapes up to an occlusion ratio of 20%.

To measure the recognition performance drop quantitatively, we draw a graph of the average volumetric IoU versus occlusion ratio as shown in Figure 3.10. For each object used in the experiment, ten occluded point clouds are sampled for each bin in the graph (e.g., 5%, 5$\sim$10%) with occlusion ratios in the corresponding range. As the occlusion ratio increases, the volumetric IoU decreases consistently for all objects. However, even when the occlusion ratio is about 25$\sim$30 %, the overall average IoU has a value higher than 0.7, which is higher than the average performance of PS-CNN as shown in Table 3.3. These experimental results verify that our algorithm exhibits a degree of robustness to occlusion by other objects.

### 3.6.2 Enhancing Performance of DSQNet with Segmentation Results

As described in Chapter 3.3.1, the proposed DSQNet converts each segmented point cloud $\mathcal{P}_i = \{\mathbf{x}_{ij} \in \mathbb{R}^3\}_{j=1}^{N_i}$ into deformable superquadric parameters $\{a_1, a_2, a_3, e_1, e_2, k, b, \alpha\}$ and pose $\mathbf{T}$. While DSQNet demonstrates improved shape recognition performance compared to previous works, it still has several limitations. First, if the number of points $N_i$ in the segmented point cloud is fewer than 300, we up-sample the point cloud to 300 to facilitate its use as input for DSQNet. However, this up-sampling method is

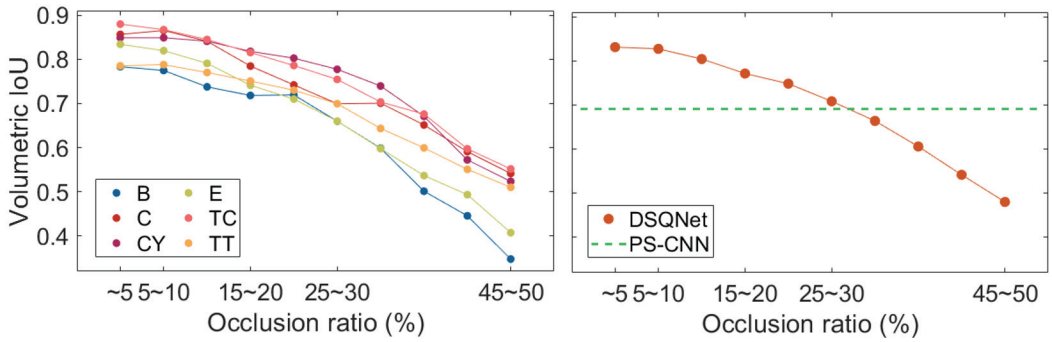


Figure 3.10: Graph of average volumetric IoU versus occlusion ratio for DSQNet for each object (left) and mean across the objects (right).

not entirely accurate, often resulting in an unnatural point cloud, particularly when the original number of points is small. This can lead to decreased recognition performance. A more critical limitation arises when recognizing multi-part objects, especially when one part occludes another in a partially observed point cloud. For instance, consider the case of recognizing a dumbbell as shown in Figure 3.11. Since DSQNet processes only segmented point clouds, it may inaccurately predict the middle part of the dumbbell to be shorter than it is, as depicted on the left, or predict a completely erroneous shape, as shown on the right. Such inaccuracies in recognition can limit the range of feasible grasp poses or result in the generation of unsuccessful grasp poses.

For this reason, we design a new neural network architecture named DSQNet$^+$ that better reflects the overall shape context of multi-part objects utilizing the segmentation results. The most significant difference between DSQNet$^+$ and DSQNet is in the representation of the input. The input to DSQNet$^+$ is a point cloud with 4-dimensional points $\mathcal{P}'_i = \{\mathbf{x}_{ij} \in \mathbb{R}^4\}_{j=1}^{n}$; for each point $\mathbf{x}_{ij}$, the first three components are equal to $\mathbf{x}_j$, and the last element of each point is 1 if $\mathbf{x}_{ij} \in \mathcal{P}_i$ and 0 otherwise, for all $j = 1, ..., n$. This

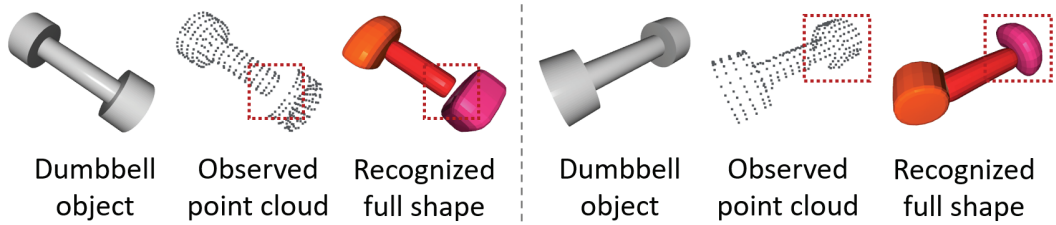


Figure 3.11: Two example cases where the original DSQNet recognizes inaccurate shapes.

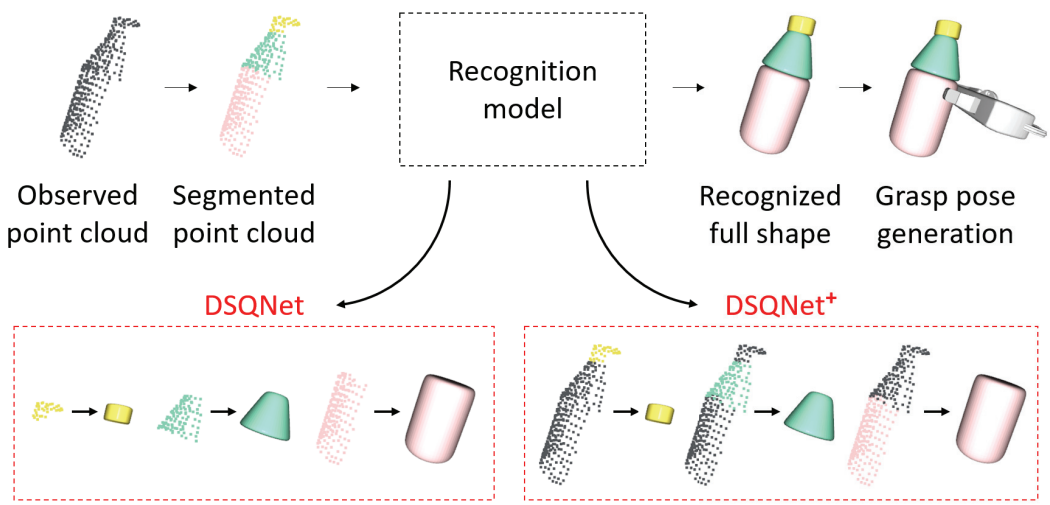


Figure 3.12: Neural network architectures of recognition models DSQNet and DSQNet$^+$


Figure 3.13: Shape recognition results for SQNet, DSQNet, and DSQNet$^+$ for six types of multi-part objects in object dataset.

representation contains information not only about the segmented point cloud of interest but also about the overall shape through the surrounding point cloud. Obviously, the need for down-sampling or up-sampling during the inference phase is effectively eliminated. Other than the input representation, the output representation, the network architecture – except for the first layer of the network –, and the loss function remain the same as those of DSQNet. The differences between DSQNet and DSQNet$^+$ are described in Figure 3.12.

We compare SQNet, DSQNet, and DSQNet$^+$ both qualitatively and quantitatively

on the test sets of the object dataset. Figure 3.13 displays the recognition results for all methods across six types of multi-part objects in the object dataset. We confirm that DSQNet$^+$ generally exhibits superior recognition performance compared to the other methods. In the case of mug and padlock objects, DSQNet tends to predict smaller handles that are only partially visible because they are occluded by the cylinder, whereas DSQNet$^+$ accurately predicts even the hidden parts. For bottle objects, both SQNet and DSQNet demonstrate poor performance, largely due to some parts having a small number of points, leading to inaccuracies in the upsampling and inference processes. Particularly noteworthy is the case of dumbbells, where SQNet and DSQNet often predict a shortened middle part. Additionally, DSQNet sometimes represents one end of the dumbbell as a truncated torus due to the segmented point clouds having a hole. In contrast, DSQNet$^+$ consistently predicts the correct shape, even in these challenging scenarios. Table 3.5 presents the quantitative results for all methods. DSQNet$^+$ achieves the highest shape recognition performance (volumetric IoU), particularly for the dumbbell and bottle objects. In conclusion, by modifying the input representation to more effectively utilize the segmentation results, we have achieved a significant improvement in shape recognition performance.

### 3.6.3 Shape Uncertainty Aware Grasping Algorithm

We have developed a shape uncertainty-aware grasping algorithm to aid grasp pose planning and to increase grasp performance in more difficult cases. Although DSQNet shows the best recognition performance on both synthetic and real-world objects, clearly shape recognition cannot always be perfect, i.e., the volumetric IoU cannot always achieve a perfect score of 1. If the generated grasp pose points to an erroneous part of the recognized shape, the gripper may fail to grasp the object. To limit the possibility of grasping

Table 3.5: IoU comparison between SQNet, DSQNet, and DSQNet$^+$ for object dataset

| **OBJECT** | SQNet | DSQNet | DSQNet$^+$ |
|---|---|---|---|
| Hammer | .8266 | .8143 | **.8359** |
| Mug | .7497 | .7976 | **.8474** |
| Screwdriver | **.8354** | .8310 | .8033 |
| Padlock | .7673 | .7848 | **.8090** |
| Dumbbell | .7583 | .7196 | **.8764** |
| Bottle | .7758 | .7887 | **.8324** |
| **Average** | .7855 | .7893 | **.8324** |

erroneous parts, we introduce a new grasp score that considers shape uncertainty.

We first define an *uncertainty score* of a point on the recognized shape surface. The uncertainty score at a point is defined to be higher when the point is farther from the actual observation. To calculate the uncertainty score at a given point, we calculate the nearest distance from the point to the partially observed point cloud. Then, the uncertainty scores are defined by linearly rescaling these distances so that the surface points nearest and farthest to the partially observed point cloud have scores of 0 and 1, respectively. The uncertainty score of the antipodal point $u$ is then defined as the mean of the uncertainty scores of the corresponding points. Finally, we define a new grasp score $c$ as the weighted sum of the original grasp score and the uncertainty score $u$, i.e., $c = -\mathbf{z}^T\mathbf{z}_t - \beta u$, where the weighting parameter $\beta$ is set to 5. Two grasping experiments based on Algorithm 1, one with and one without this uncertainty score term, have been conducted on five objects with low grasping success rates (Table 3.4). Twenty trials are conducted for each object, with the object's pose fixed at each trial in both experiments to achieve a fair comparison.

Table 3.6: Grasping results with and without uncertainty score

| **METHOD** | Jello | Hammer | Mug | Bottle | Dumbbell |
|---|---|---|---|---|---|
| DSQNet | 18/20 | 18/20 | 17/20 | 17/20 | 15/20 |
| DSQNet + Unc. | 18/20 | 18/20 | 18/20 | 18/20 | 17/20 |

Figure 3.14 shows the recognition results and grasp poses generated with and without the uncertainty score for several real-world objects. We distinguish between the results with and without the uncertainty score by "DSQNet + Unc." or "DSQNet". In the results of the uncertainty-aware case, the color of the object mesh indicates the uncertainty score, e.g., the score is 0 for blue and 1 for red. When grasping the jello, the gripper attempts to grasp the observed part of the recognized object when the uncertainty is taken into account. In this case, both grasp poses are successful in grasping since shape recognition is almost perfect. There remain a few grasping failure cases when the existing algorithm generates grasp poses for mugs and dumbbells, as a result of the gripper attempting to grasp the erroneous part of the recognized shape. By considering uncertainty in Algorithm 1, we verify that some of these cases can be prevented as seen from Figure 3.14.

The grasping performance results with and without the uncertainty score are shown in Table 3.6. There is no performance difference between the two algorithms for the jello and hammer cases, but the performance of the former is higher in other objects. In particular, the performance improvement is most noticeable for dumbbells, which have many erroneous parts in the recognized shapes. Grasping performance has been improved by introducing the uncertainty score into the current algorithm, particularly for cases when shape recognition is not perfect.

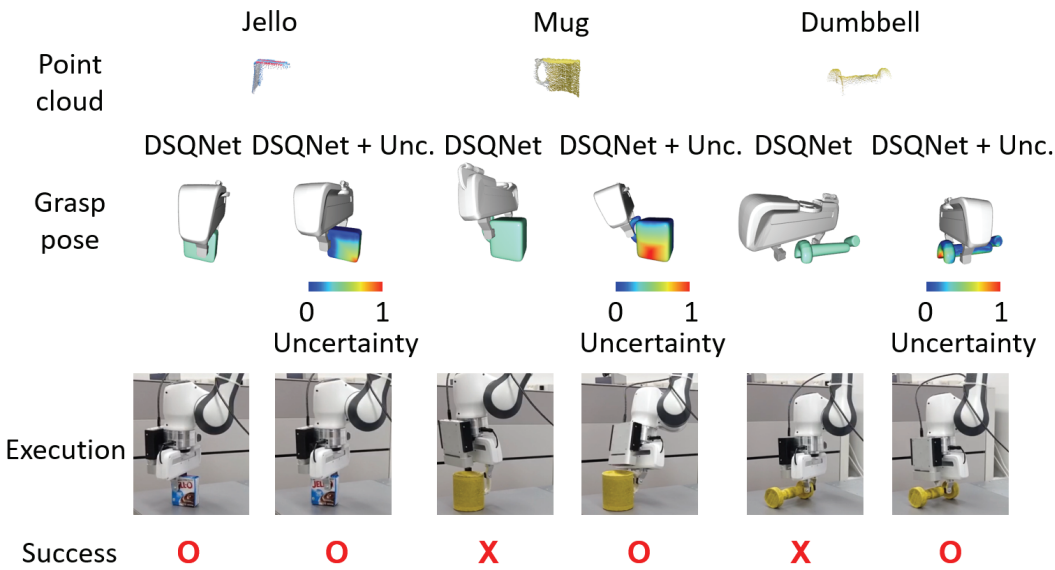


Figure 3.14: Recognition and grasp pose generation results with and without uncertainty score for real-world objects.


Figure 3.15: Examples of human-robot collaboration in scenarios involving the grasping of a Cheez-It box.

### 3.6.4 Real-time Application of DSQNet for Human-Robot Collaboration

We have also used our DSQNet in scenarios where human and robot collaboration is essential. Fast (or real-time) inference speed is crucial for effective human-robot collaboration, and our DSQNet boasts a very fast inference speed. Leveraging these advantages, we have conducted a grasping experiment based on the following scenario: The robot needs to grasp a Cheez-It box, but the object is too flat and large to be grasped directly. With human assistance, the robot attempts to grasp the Cheez-It box.

Figure 3.15 shows some examples of human-robot collaboration in the scenarios described above. Initially, the Cheez-It box is not graspable, so the robot cannot identify an appropriate grasp pose. In such instances, a human aids by either pushing the object toward the edge of the table or lifting it into the air, enabling the robot to grasp the object. The robot, using our DSQNet, recognizes the moving object in real-time and assesses its graspability through our grasp pose generation algorithm. When the object

becomes graspable and its pose stabilizes (i.e., the variation in the center of the recognized object diminishes), the robot initiates grasp planning. Consequently, the robot successfully grasps the Cheez-It box with the aid of a human.

## 3.7 Beyond Superellipsoids: Adopting Superparaboloids for Tableware Objects

As we describe in Chapter 2, we use the term "superquadrics" to exclusively refer to superellipsoids, except for this subsection. In household environments, a wider variety of unknown objects can be present, such as bowls, dishes, and spoons as shown in Figure 3.16. In particular, it is difficult to express the concave objects such as bowls using only superellipsoid primitives. In this subsection, our goal is to a skill that can grasp various tableware objects on the table adopting new shape primitives named superparaboloids.

### 3.7.1 Unified Superquadric Network

We describe the Unified Superquadric Network (USQNet), a deep neural network that takes a partially observed point cloud $\mathcal{P} = \{\mathbf{x}_i \in \mathbb{R}^3\}_{i=1}^{N}$ as input like DSQNet, and outputs the superellipsoid or superparaboloid primitive that best represents the full object shape. To define a unified superquadric shape primitive including superellipsoid and superparaboloid, we use the two implicit functions separately with different notations. Let the implicit equation for a superellipsoid surface 2.2.1 and for a superparaboloid surface 2.2.7 be $f_{se}$ and $f_{sp}$, respectively. Then, in addition to the parameters $(a_1, a_2, a_3, e_1, e_2)$, we define a class parameter $c \in \{0, 1\}$ which determines whether the shape is superellipsoid or superparaboloid, in detail, superparaboloid when $c = 1$


Figure 3.16: Several examples of tableware objects.

and superellipsoid when $c = 0$; the shape parameter vector consists of 6 parameters $\mathbf{q} = (a_1, a_2, a_3, e_1, e_2, c) \in \mathbb{R}^6$. Using this notation, the implicit surface of the unified primitive $f(x, y, z) = 1$ is expressed by:

$$f(x, y, z) = c \cdot f_{sp}(x, y, z) + (1 - c) \cdot f_{se}(x, y, z) = 1. \tag{3.7.9}$$

The basic structure of USQNet is almost similar to DSQNet except for the classifier that classifies the superellipsoid and superparaboloid, and the overall architecture is shown in Figure 3.17. The network consists of (i) the EdgeConv layers [72] with latent space dimension (64, 64, 128, 256) and max pooling operator to produce a global feature vector from $\mathcal{P}$ in a permutation-invariant manner and (ii) five fully-connected layers (MLP) with latent space dimension (512, 256) (with LeakyRelu nonlinearities) to obtain the superquadric parameter $\{a_1, a_2, a_3, e_1, e_2, c\}$ and the pose $\mathbf{T} = [\mathbf{R}; \mathbf{t}]$ from the extracted global feature. Especially, each MLP outputs (i) translation vector $\mathbf{t} \in \mathbb{R}^3$, (ii) quaternion vector $\mathbf{r} \in \mathbb{S}^3$ representing the rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$, (iii) size parameters $\mathbf{a} = (a_1, a_2, a_3) \in \mathbb{R}^3$, (iv) shape parameters $\mathbf{e} = (e_1, e_2) \in \mathbb{R}^2$, and (v) superellipsoid/superparaboloid class parameter $c \in [0, 1]$; the values $e_1$ and $e_2$ are bounded in $[0.2, 1.7]$ since the superquadric equation diverges when $e_1$ and $e_2$ goes to zero and shows too complex shapes when $e_1$ and $e_2$ become large.

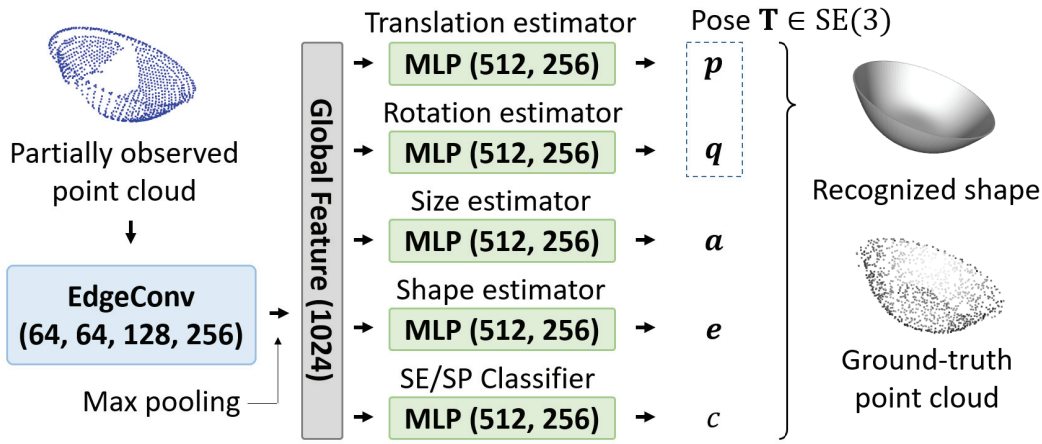


Figure 3.17: Superquadric recognition network.

### 3.7.2 Loss Function for Training

For the predicted superquadric to fit well with the ground-truth shape, the loss function should also be designed to be the difference between the prediction and the ground-truth object shapes. For ground-truth shape, we obtain five kinds of ground-truth labels: (i) *ground-truth point cloud* of the object, uniformly sampled points from the surface of the object, denoted by $\mathcal{P}_g = \{\mathbf{x}_{g,i} \in \mathbb{R}^3\}_{i=1}^{n_g}$, where $n_g = 512$, (ii) *ground-truth surface normals* of the object $\mathcal{N}_g = \{\mathbf{n}_{g,i} \in \mathbb{S}^2\}_{i=1}^{n_g}$ obtained from the surface mesh of the object, (iii) *ground-truth shape class* of the object $c_g$, (iv) *ground-truth height* of the object $h_g$, and (v) *ground-truth z-axis* of the object $\mathbf{z}_g$.

**Superparaboloid loss $\mathcal{L}_{sp}$.** The loss consists of three terms: surface fitting loss, height loss, and $z$-axis loss. The surface fitting loss fits a given ground-truth point cloud to the superparaboloid surface, and since superparaboloid is an infinite surface, additional loss named height loss is required to restrict the object shapes' heights. Also, the objects are placed on a table in general cases, so z-axis loss is also added to make

training easier. Therefore, the final superparaboloid loss is as follows:

$$\mathcal{L}_{sp} = \underbrace{\frac{1}{n_g}\sum_{j=1}^{n_g} f_{sp}^2(\mathbf{T}^{-1}\mathbf{x}_{g,j})}_{\text{Surface fitting loss}} + \underbrace{w_h(a_3 - h_g)^2}_{\text{Height loss}} + \underbrace{w_z(1 - \mathbf{z}^T\mathbf{z}_g)^2}_{z\text{-axis loss}}, \tag{3.7.10}$$

where the hyperparameters are $w_h = 10$ and $w_z = 1$, $f_{sp}$ is defined by the parameters $\{a_1, a_2, a_3, e_1, e_2\}$, and $\mathbf{T}$ is its pose.

**Superellipsoid loss $\mathcal{L}_{se}$.** The loss consists of three terms: surface fitting loss, surface normal loss, and $z$-axis loss. For surface fitting loss, we use the distances from the ground-truth point cloud to the predicted superquadric as the loss function. The distance form is from [84] which is defined as follows. Then, the distance $\delta$ between a point $\mathbf{x}_0 \in \mathbb{R}^3$ and a superellipsoid surface $f_{se}(\mathbf{x}) = 1$ is

$$\delta(\mathbf{x}_0, f_{se}) = ||\mathbf{x}_0|| \left| 1 - f_{se}^{-\frac{e_1}{2}}(\mathbf{x}_0) \right|, \tag{3.7.11}$$

where $\|\cdot\|$ denotes the Euclidean norm. Additionally, the surface normal loss, difference between the ground-truth normal vector $\mathbf{n}_{g,j}$ and predicted normal vector $\mathbf{n}_j = \nabla_{\mathbf{x}} f_{se}(\mathbf{x}_j)$, is added to better abstract the objects (e.g., fork, knife, spoon) suitable for grasping, and a $z$-axis loss is also added for similar reasons described as above. Accordingly, the loss function is defined as:

$$\mathcal{L}_{se} = \underbrace{\frac{1}{n_g}\sum_{j=1}^{n_g} \delta^2(\mathbf{T}^{-1}\mathbf{x}_{g,j}, f_{se})}_{\text{Surface fitting loss}} + \underbrace{\frac{w_n}{n_g}\sum_{j=1}^{n_g}(1 - \mathbf{n}_j^T\mathbf{n}_{g,j})^2}_{\text{Surface normal loss}} + \underbrace{w_z(1 - \mathbf{z}^T\mathbf{z}_g)^2}_{\text{z-axis loss}}, \tag{3.7.12}$$

where the hyperparameters are $w_n = 0.1$ and $w_z = 1$. $f_{se}$ is also defined by the parameters $\{a_1, a_2, a_3, e_1, e_2\}$, and $\mathbf{T}$ is its pose.

**Classification loss $\mathcal{L}_c$.** This is a classical binary classification loss as follows:

$$\mathcal{L}_c = -(c_g \log c + (1 - c_g)\log(1 - c)) \tag{3.7.13}$$

In conclusion, the total loss function used for training USQNet is as follows:

$$\mathcal{L} = \mathcal{L}_c + c_g\mathcal{L}_{sp} + (1 - c_g)\mathcal{L}_{se} \tag{3.7.14}$$

### 3.7.3 Recognition and Grasping on Real-world Tableware Objects

We assume that multiple objects are placed on a table but they are not highly overlapped, and we focus on representing each object as a single shape primitive. In this case, we use a classical method named DBSCAN clustering algorithm as a point cloud segmentation algorithm [91]. DBSCAN is a density-based clustering method that groups each local point cloud cluster. We first perform the point cloud segmentation and then up/downsample the point cloud to match the total number of the points to be 2048. The examples for the results of superquadric shape recognition are shown in Figure 3.18.

When a superquadric representation of the target object is obtained from shape and pose recognition, we can generate candidate grasp poses. In this subsection, we use simple heuristic-based method for grasp pose generation. We use different candidate grasp pose generation strategies for these two shape classes. For superparaboloid shapes, we generate 10 side grasp poses according to the parameters (i.e., size parameters $a_1, a_2, a_3$ and shape parameters $e_1, e_2$) as shown in Figure 3.19. At this time, the two gripper fingers should be on the antipodal points on the object. For superquadric shapes, we generate 10 top-down grasp poses; in this case, grasp poses with a distance between the antipodal points greater than 7cm are removed from the candidates (the maximum gripper width of the Franka gripper is 8cm).

After generating candidate grasp poses, we can easily check the graspability of the objects. Figure 3.20 shows how to check graspability of each object. After generating candidate pre-defined grasp poses for each object, we check the graspability by checking collision between the pre-defined grasp poses and the table or surrounding objects. The

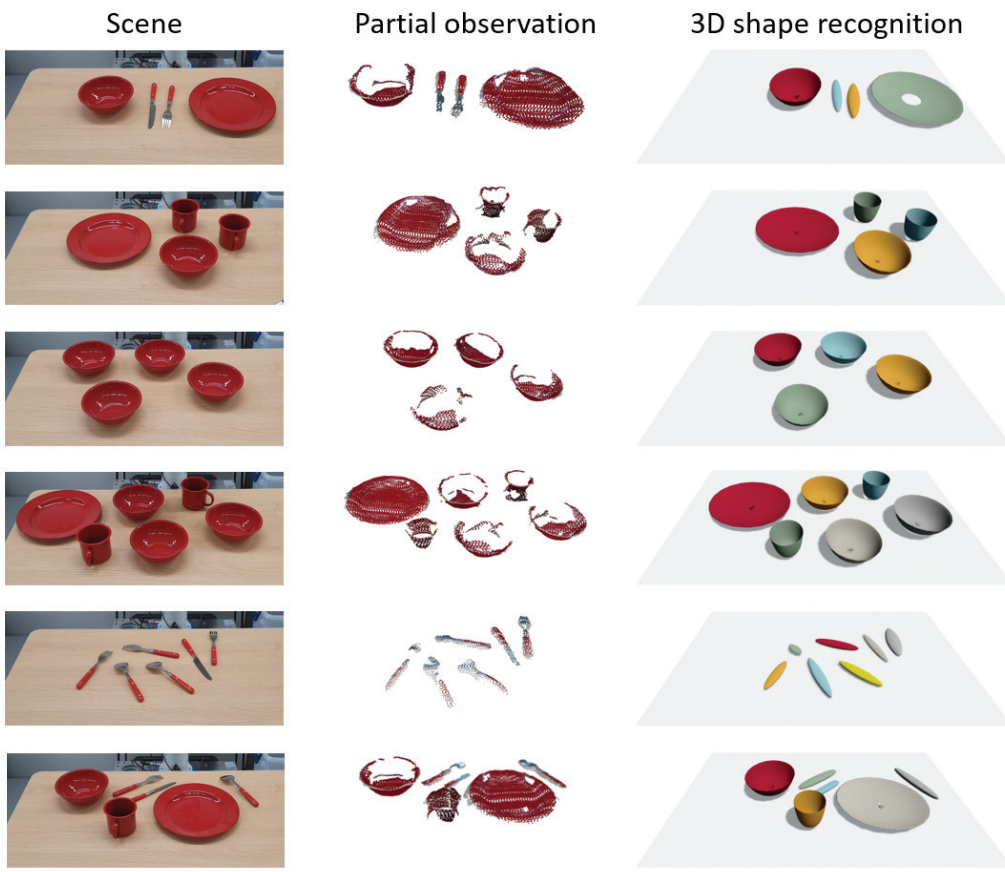


Figure 3.18: The representative examples of the superquadric shape recognition results.

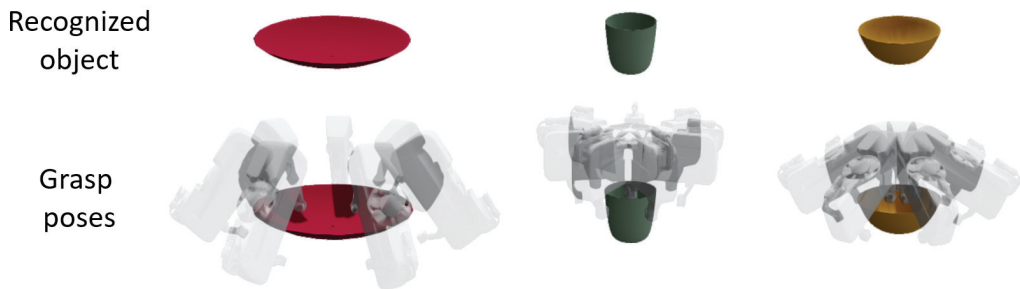


Figure 3.19: Generated candidate grasp poses for various recognized superparaboloid shapes.

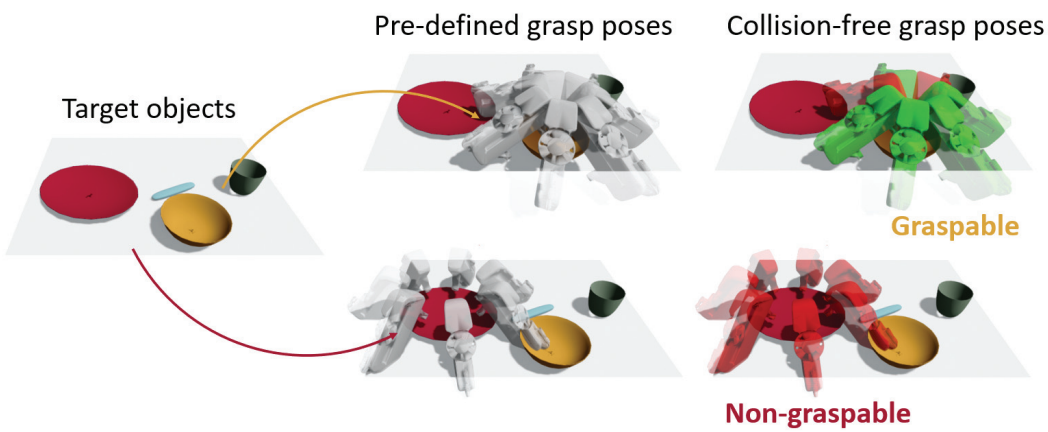


Figure 3.20: Graspability description. The yellow bowl is graspable (upper row) and the red dish is non-graspable (lower row)

green grippers are collision-free grasp poses and the red ones are collide grasp poses. For the yellow bowl, there is at least one of the collision-free grasp poses so the bowl is graspable. Otherwise, there is no collision-free poses of the red dish, so the dish is non-graspable. The collision between the grippers and the environment can be checked in real-time by using (i) object implicit function equation and (ii) batch-wise parallel computing based on PyTorch.

**Collision checking method.** We first sample the points on the gripper mesh; the sampled points are denoted by $\mathcal{P}_{gr} = \{\mathbf{x}_{gr,j} \in \mathbf{R}^3\}_{j=1}^{n_{gr}}$, where $n_{gr} = 2048$. For an implicit object representation $f(\mathbf{x}) = 1$, we note that a point $\mathbf{x}_0 \in \mathbb{R}^3$ is inside the object when $f(\mathbf{x}_0)$ is less than 1 and outside when $S(\mathbf{x}_0)$ is greater than 1. We use this fact to determine whether the gripper collides with the objects or tables or not: when the value

$$\min_j S(\mathbf{T}_{gr}^{-1}\mathbf{x}_{gr,j}), \tag{3.7.15}$$

where $\mathbf{T}_{gr}$ is the pose of the gripper, is less than 1, then the gripper collides with the object. Through this, collision can be checked quickly and efficiently. The grasping results for the real-world tableware objects are shown in Figure 3.21.

## 3.8 Conclusion

This paper has presented a novel recognition-based grasping method using the Deformable Superquadric Network (DSQNet). Our methods adopt deformable superquadrics primitives that are both expressive and computationally simple. Unlike existing optimization based methods, DSQNet uses a supervised learning framework with ground-truth shape labels to find the full shapes of objects (including occluded parts) from partially observed point clouds. An antipodal points sampling-based grasping strategy from the recognized shape is designed that exploits the advantages of the deformable

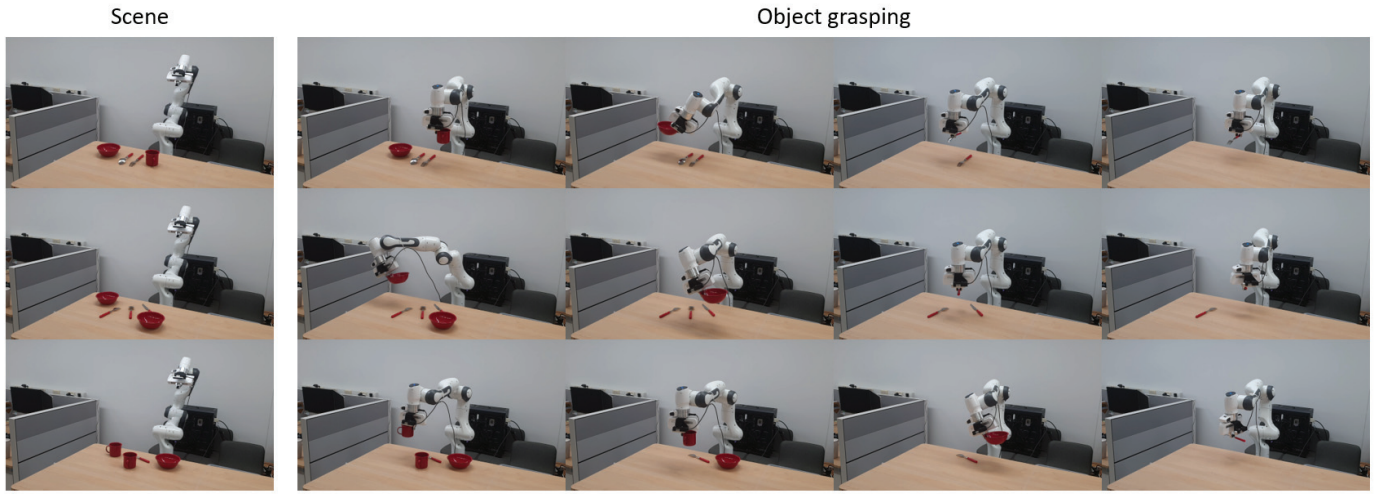


Figure 3.21: Grasping results for real-world tableware objects.

superquadric primitives.

We have verified our recognition-based method with recognition tasks on a synthetic dataset, and grasping tasks on real-world objects. The experimental results confirm that DSQNet can recognize shapes of objects accurately and quickly compared to existing baseline methods. Whereas optimization-based methods cannot predict shapes corresponding to occluded parts of the objects, our supervised learning framework successfully determines full shapes in many cases. Our recognition-based grasping method achieves a success rate of 93% for real-world grasping tasks, which is superior to rates achieved for existing recognition-based grasping methods.

**Limitations and Future Directions** Although our method shows high recognition performance on most of the household objects, recognition performance for more diverse and complex objects can be greatly enhanced by the incorporation of publically available large scale 3-D datasets such as ShapeNet [92]. Since our current method requires difficult-to-obtain primitive shaped point segmentation labels, using public datasets can be a challenge. Possible directions for future work include designing a network that incorporates both the segmentation network and DSQNet, that simultaneously optimizes point cloud segmentation and fitting without segmentation supervision.

Despite the expressiveness of deformable superquadrics, there remain shapes that are difficult or impossible to model. For example, concave shapes such as bowls present a challenge. Among the superquadric sets, we use only superellipsoids, but there exist other possible choices, e.g., superhyperboloids and supertoroids. In particular, the supertoroids can express a variety of shapes with cavities, from the torus to the cylinder shell; with an appropriate deformation model, more varied objects such as bowls can now be represented. The extension of our work using more diverse superquadric primitives remains a topic for future work.

# 4

# SQPDNet: Superquadric Pushing Dynamics Network

## 4.1 Introduction

Robotic visual pushing manipulation – by visual manipulation, we mean that only visual observations (e.g., depth camera) are available – in cluttered environments including unseen objects is an important yet challenging manipulation skill that allows a robot to interact with and change its environment to be suitable for performing downstream tasks. For example, pushing manipulation techniques have been used to move tabletop objects graspable [17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27], rearrange multiple objects [28, 29, 30, 31], sort objects according to specific rules [32, 33], and find a target object occluded by the other objects [93, 94]. We refer to the survey paper for comprehensive reviews for the non-prehensile pushing manipulation methods [95].

We consider model-based approaches for the pushing manipulation that consist of

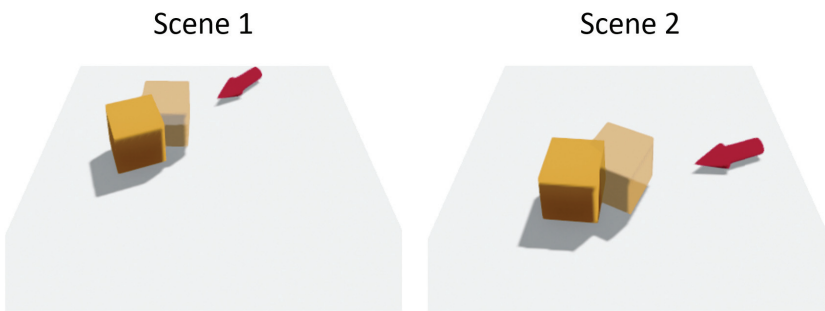


Figure 4.1: The box object and pushing vector in *Scene 1* are transformed by some same planar rigid-body transformation as those in *Scene 2*. An ideal pushing dynamics model should be $\mathrm{SE}(2)$-equivariant, i.e., the resulting motion in *Scene 2* is a transformation of that in *Scene 1*.

the following two components: (i) to construct a pushing dynamics model which predicts the motions of the objects after a robot performs a pushing action to the environment and (ii) to find an optimal sequence of pushing actions that achieves the goal given a predesigned task criteria [96, 97]. Our primary focus is the first step which is to develop an accurate *visual pushing dynamics model* that takes a visual observation as an input. Analytic approaches that precisely model the physical interactions [3, 4, 5, 6] cannot be used since we are given unseen objects with only vision data.

Recently, there has been considerable interest in data-driven methods for *learning* pushing dynamics models [34, 35, 36, 37, 38, 39, 40, 41, 42, 43], but their generalization performances are still far-less-than-satisfying. We claim that one of the important reasons behind this is that neural network models used in existing approaches lack considering the symmetry of the physical systems, and more precisely, *equivariance*. For example, suppose a model is trained with an experience where a robot pushes a box object into a red arrow direction as shown in Figure 4.1 (*Scene 1*). And consider a

new situation where the same box object is located at a different pose and the robot pushes the object in the same relative direction as shown in Figure 4.1 (*Scene 2*). At an intuitive level, a good model should be able to easily generalize to this type of new situation, where tabletop objects are only translated or rotated along the $z$-axis. In more technical terms, the pushing dynamics model needs to be *equivariant* to the $\mathrm{SE}(2)$ transformation.

In this paper, we define the $\mathrm{SE}(2)$-equivariant pushing dynamics model and deliberately design a neural network architecture that by construction has the equivariance property. The core idea to make the model equivariant is to properly transform the coordinates of the pushing action and the objects' poses as needed; details are elaborated in Chapter 4.3. This construction naturally captures the symmetry of the physical systems and significantly improves the generalization performances.

To employ the proposed equivariant pushing dynamics model in environments with only vision data and unseen objects, we need an additional module that can recognize the objects' shapes and poses. In this work, we represent 3d objects' shapes by using the shape class called the *superquadrics*, which can express diverse shapes ranging from boxes, cylinders, and ellipsoids to other complex symmetric shapes. We train the recognition network that predicts the objects' shapes with superquadrics by adopting an idea from [64]. We call our superquadric object representation-based pushing dynamics model a *SuperQuadric Pushing Dynamics Network (SQPD-Net)*.

Experiments and benchmark comparisons against the existing state-of-the-art methods confirm that our dynamics model achieves the highest performance in predicting objects' motions after pushing action. In addition, we validate the effectiveness of our model by using it for model-based optimal controls for various pushing manipulation tasks in both simulation and real-world experiments.

## 4.2 Related Works

### 4.2.1 Model-free Pushing Manipulation

In model-free pushing manipulation methods, a policy that directly maps a visual observation to a sequence of pushing actions is learned without learning a pushing dynamics model. They typically require to design a task-specific reward function and train the policy in an end-to-end manner with a large amount of data. Researchers have attempted to solve diverse tasks with these methods such as grasping [17, 18, 19, 20, 21, 22, 23], singulation [24, 25, 26, 27], object rearrangement [28, 29, 30, 31], object sorting [32, 33], and invisible target object finding [93, 94]. We refer to the survey paper for comprehensive reviews for the end-to-end pushing methods [95]. Model-free methods are known to have very good convergence performance, but they require a lot of data. Also, when a new task is given, the agent must be trained from scratch. On the other hand, in model-based approaches, the learned model is reusable given a new task, and much less additional data is needed.

### 4.2.2 Visual Pushing Dynamics Learning

A few studies have proposed data-driven *visual* pushing dynamics models using trained deep neural network models. These deep neural networks input a visual observation and a pushing action (or action sequences) and predict the motions of the objects after a robot performs a pushing action to the environment.

**Learning dynamics from pixels.** Early works leveraged direct visual observations to learn dynamics in pixel space in a data-driven manner. One of the most naive approaches is to use a basic convolution network to predict (optical) flow in image space, but [34] find these models to have very poor performance. [34] and [35] have modeled object motions in pixel space using several rigid body transformations, as opposed to

predicting pixel-wise flow from observations. These methods have demonstrated impressive results in manipulation tasks, including closed-loop planar pushing. However, they encounter difficulties in accurately predicting outcomes when complex interactions between objects occur. More recent works develop object-centric pushing dynamics models by representing each object by a visual feature [36, 37, 38, 39, 41], or a segmented image using a pre-trained segmentation network [42]. However, these works often predict inaccurate or incomplete object motions as they rely on only partial observations of the objects.

**Self-supervised learning using implicit object shapes.** Recently, self-supervised learning techniques that use only visual observation but do not rely on data collection have been developed. [98] and [99] have parametrized object shapes using implicit signed distance functions and optimized the parameters through a differentiable physics-based simulator to determine accurate dynamics. Additionally, [100] have expanded this approach by incorporating tactile sensor data alongside vision sensor data to determine the dynamics of deformable objects. [43] employs a vision transformer architecture to enhance the performance of pushing dynamics prediction. Further, [101] utilizes compositional neural radiance fields and graph neural networks to learn multi-object dynamics from RGB image observations.

**Object dynamics learning with 3D amodal geometry.** A recent study addresses the limitations of previous methods by inferring the amodal 3D geometry of each object, including unobserved regions, within a 3D voxel space. This approach, detailed in [40], leverages the complete 3D geometry to predict the rigid body motion of the entire object, rather than just the visible surfaces, thereby increasing accuracy and resolving the incompleteness inherent in earlier methods.

Our work is also in the spirit of [40] in utilizing the ground-truth information of the objects to increase the accuracy of the prediction. Still, we use implicit representation

for the objects to utilize them for efficient motion prediction and various manipulation.

### 4.2.3 Shape Recognition from Visual Observation

Many works have been proposed to recognize the full 3D shapes from partial observations such as depth images. Some of them use explicit representation such as occupancy grid [51], point cloud [52], or mesh [53]. Since they often lead to shape prediction not being precise enough due to limited resolutions of the representations, recent works have explored learning implicit 3D representations for the objects using neural implicit functions [55, 56, 57, 58]. In this paper, we adopt superquadric functions that balance the expressiveness of the shapes and efficiency of the computational with a small number of parameters [61]. They have been used for robotic manipulation such as grasping [49, 75, 76, 64, 50]. We note in our paper that we represent each object as a single superquadric function, but our work can be easily extended to general implicit representations, especially deformable superquadric [63, 64] or a set of superquadrics [102, 103, 104, 105].

### 4.2.4 Invariance and Equivariance in Robot Learning

Recently, invariance and equivariance properties turn out to be very important inductive biases for deep learning models to generalize well and be trained data efficiently [106]. Convolutional neural networks (CNNs) have translation equivariance properties, which are particularly suitable for image recognition tasks [107]. Graph neural networks, point cloud neural networks, and set neural networks have the permutation invariant properties [108, 109, 110]. More advanced equivariance properties such as rotational equivariance for image data and $\mathrm{SO}(3)$-equivariance for spherical image data have been achieved by group equivariant CNNs [111, 112, 113]. In the context of robot manipulation, recent

works have adopted these equivariance principles and shown significantly improved sample efficiency and performance. Some of them use group invariant $Q$ function to achieve the group equivariant reinforcement learning [114, 115, 116, 117, 118] or use $\mathrm{SE}(3)$-equivariant object representation for object manipulation tasks [119, 120, 121, 122]. Our work differs from other works in that we learn *equivariant dynamics model* for robot pushing manipulation, and it is novel in that the $\mathrm{SE}(2)$-equivariant dynamics model, which is a suitable inductive bias for pushing dynamics, is formulated. Similar to our approach, there is an approach that performs input transformations to achieve the equivariance [123], but the model is equivariant for only a few rotations and scales; our model is intrinsically equivariant to the continuous SE(2) transformations.

## 4.3 SE(2)-Equivariant Pushing Dynamics Models

In this section, we develop a neural network architecture specialized to learn a $\mathrm{SE}(2)$-equivariant pushing dynamics model. We assume that multiple rigid-body objects are placed on the table whose surface is assumed to be flat and orthogonal to the gravity direction, and the robot interacts with the objects by pushing manipulation. Each object is represented by a pose parameter $\mathbf{T} \in \mathrm{SE}(3)$ ($4 \times 4$ matrix representation) and shape parameter $\mathbf{q}$, where the pose parameter is described with respect to some global fixed frame and the shape parameter is a vector. And the pushing action is defined as a tuple $(\mathbf{p}, \mathbf{v})$ where the tip of the end-effector moves from the position $\mathbf{p} \in \mathbb{R}^3$ to $\mathbf{p} + \mathbf{v} \in \mathbb{R}^3$. As the tip of the end-effector moves, the robot can have contact with environments, pushes objects, and changes the poses of the objects.

Further, we assume there are maximally $M$ rigid-body objects on the table that have the parameters $\{(\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N$ for $N \leq M$. We consider a discrete-time pushing dynamics model $f$ that outputs the object's transformed poses $\{\mathbf{T}'_i\}_{i=1}^N$ when a pushing

action $(\mathbf{p}, \mathbf{v})$ is applied, i.e., $\{\mathbf{T}'_i\}_{i=1}^N = f(\{(\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N, (\mathbf{p}, \mathbf{v}))$, where $N$ can vary as long as $N \leq M$. Assuming the gravity direction is the $z$-axis, we first give a precise definition of the $\mathrm{SE}(2)$-equivariant pushing dynamics model:

**Definition 4.1.** A pushing dynamics model $f$ is $\mathrm{SE}(2)$-equivariant if

$$\{\mathbf{C}\mathbf{T}'_i\}_{i=1}^N = f(\{(\mathbf{C}\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N, (\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t_{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v})) \tag{4.3.1}$$

for all object numbers $N \leq M$ and rigid-body transformations $\mathbf{C}$ that have the following form

$$\mathbf{C} = \begin{bmatrix} \mathbf{Rot}(\hat{\mathbf{z}}, \theta) & \mathbf{t_{xy}} \\ 0 & 1 \end{bmatrix}, \tag{4.3.2}$$

where $\mathbf{Rot}(\hat{\mathbf{z}}, \theta)$ is a $3 \times 3$ rotation matrix for rotations around $z$-axis and $\mathbf{t_{xy}} = (t_x, t_y, 0) \in \mathbb{R}^3$.

To build a $\mathrm{SE}(2)$-equivariant neural network architecture, we first introduce an object pose decomposition method that decomposes an object pose $\mathbf{T}_i \in \mathrm{SE}(3)$ to a pose projected to the table surface denoted by $\mathbf{C}_i \in \mathrm{SE}(3)$ and the relative rigid-body transformation $\mathbf{U}_i \in \mathrm{SE}(3)$ such that $\mathbf{T}_i = \mathbf{C}_i\mathbf{U}_i$.

**Object Pose Decomposition.** Given an object pose $\mathbf{T} \in \mathrm{SE}(3)$, we decompose it to two $4 \times 4$ matrices $\mathbf{C}, \mathbf{U} \in \mathrm{SE}(3)$ as visualized in Figure 4.2. First, $\mathbf{C}$ is defined by projecting $\mathbf{T}$ to the table surface, which has the form in equation (4.3.2). And secondly, $\mathbf{U}$ is defined as $\mathbf{C}^{-1}\mathbf{T}$. More details are in Appendix B.1.1.

Now, we explain our network architecture for the pushing dynamics model $f$; overall architecture is described in Figure 4.3. The model $f$ is defined as $\{f_i\}_{i=1}^N$ where each $f_i$ outputs the $i$-th object's transformed pose, i.e., $\mathbf{T}'_i = f_i(\{(\mathbf{T}_j, \mathbf{q}_j)\}_{j=1}^N, (\mathbf{p}, \mathbf{v}))$. For $f_i$, we first decompose the $i$-th object pose $\mathbf{T}_i = \mathbf{C}_i\mathbf{U}_i$ and transform the other objects' poses (including itself) and pushing action as follows: (i) $\mathbf{T}_j \mapsto \mathbf{C}_i^{-1}\mathbf{T}_j$ for

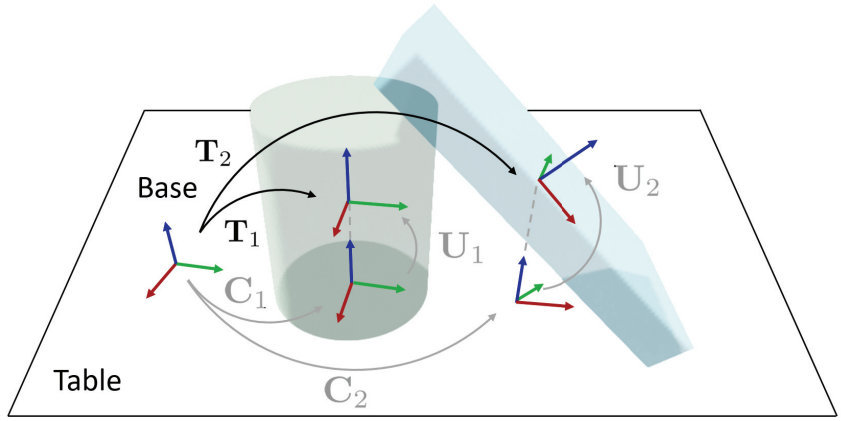


Figure 4.2: Object Pose Decomposition.

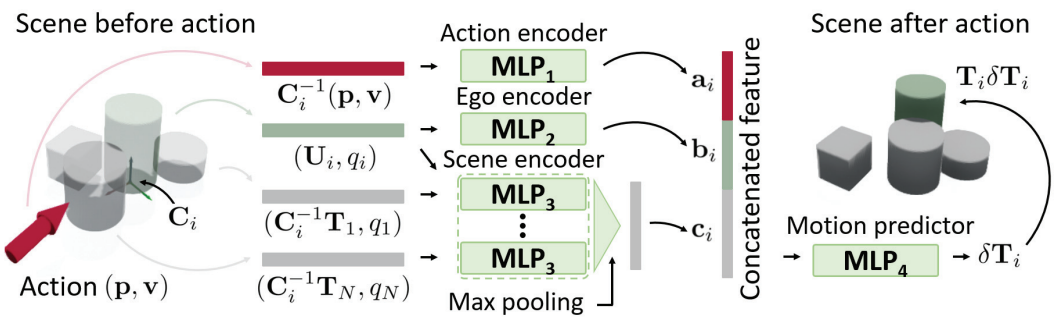


Figure 4.3: SE(2)-equivariant pushing dynamics neural network architecture for an $i$-th object, $f_i$.

$j = 1, \cdots, N$ and (ii) $(\mathbf{p}, \mathbf{v}) \mapsto \mathbf{C}_i^{-1}(\mathbf{p}, \mathbf{v}) := (\mathbf{R}_i^T \mathbf{p} - \mathbf{R}_i^T \mathbf{t}_i, \mathbf{R}_i^T \mathbf{v})$ where $\mathbf{R}_i$ and $\mathbf{t}_i$ are rotation matrix and translation vector parts of $\mathbf{C}_i$. Then, three different multi-layer perceptron (MLP) networks are used to extract $\mathrm{SE}(2)$-invariant feature vectors: (i) the $\text{MLP}_1$ takes the transformed action $\mathbf{C}_i^{-1}(\mathbf{p}, \mathbf{v})$ and outputs a feature vector $\mathbf{a}_i$, (ii) the $\text{MLP}_2$ takes the $i$-th object's parameter $(\mathbf{U}_i, \mathbf{q}_i)$ and outputs a feature vector $\mathbf{b}_i$, and (iii) the $\text{MLP}_3$ takes the transformed object's parameters $(\mathbf{C}_i^{-1}\mathbf{T}_j, \mathbf{q}_j)$ and outputs a feature vector $\mathbf{c}_i^j$ for all $j = 1, \cdots, N$ and then these output vectors pass through some permutation invariant function $h$ as $\mathbf{c}_i = h(\mathbf{c}_i^1, \cdots, \mathbf{c}_i^N)$ such as the element-wise max pooling. These feature vectors are concatenated as $\mathbf{y}_i = (\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i)$, and we have the last layer $\text{MLP}_4$ that takes $\mathbf{y}_i$ and outputs $\delta\mathbf{T}_i \in \mathrm{SE}(3)$. We note that these MLP layers are shared across all $i = 1, \cdots, N$. Then, the transformed poses are defined as $\mathbf{T}_i' = \mathbf{T}_i \delta \mathbf{T}_i$ for all $i = 1, \cdots, N$. As a result, this dynamics model is $\mathrm{SE}(2)$-equivariant by construction; the proof is in Appendix B.1.2.

**Training.** Denote by $\mathbf{s} = \{(\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N$ for some $N \leq M$ and $\mathbf{a} = (\mathbf{p}, \mathbf{v})$. In this paper, we train the pushing dynamics model given a set of tuples $\{(\mathbf{s}, \mathbf{a}, \{\mathbf{T}_i'\}_{i=1}^N)_k\}_{k=1}^K$ where $\mathbf{T}_i'$ is the next pose of the $i$-th object. The loss function $\mathcal{L}$ is defined by comparing the ground-truth next poses $\{\mathbf{T}_i'\}_{i=1}^N$ and the predicted poses $\{\hat{\mathbf{T}}_i'\}_{i=1}^N = f(\mathbf{s}, \mathbf{a})$ as follows:

$$\mathcal{L}(f) = \sum_{i=1}^{N} \left( \|\mathbf{t}_i' - \hat{\mathbf{t}}_i'\|_2^2 + \alpha \cdot d_{\mathrm{SO}(3)}^2(\mathbf{I}_3, \mathbf{R}_i'^{-1}\hat{\mathbf{R}}_i') \right), \tag{4.3.3}$$

where $d_{\mathrm{SO}(3)}$ is a distance measure between two rotation matrices, $\mathbf{I}_3$ is $3 \times 3$ identity matrix, $\mathbf{R}_i', \hat{\mathbf{R}}_i'$ and $\mathbf{t}_i', \hat{\mathbf{t}}_i'$ are rotation matrices and translation vectors parts of $\mathbf{T}_i', \hat{\mathbf{T}}_i'$, respectively, and $\alpha$ is a weighting parameter (for our later experiments we set $\alpha$ to 0.1). The details about the used distance measure are in Appendix B.3.

## 4.4 Object Recognition-based Pushing Manipulation

If we have known objects and can easily estimate the poses of the objects, then it is straightforward to use the learned pushing dynamics model for pushing manipulation. However, for unseen objects, we first need to recognize the objects' shapes and poses. Therefore, our overall framework consists of the following two steps: (i) to recognize objects' shapes and poses and (ii) to push objects by using the learned pushing dynamics model and pre-designed task criteria, of which details are explained in the following subsections.

### 4.4.1 Object Shape and Pose Recognition via Superquadrics

We propose to use implicit functions to represent 3d objects' shapes. In general, an implicit object surface representation is defined by a level set of a function $S(x, y, z; \mathbf{q}, \mathbf{T}) = 0$, where $\mathbf{q}$ is a shape parameter and $\mathbf{T} \in \mathrm{SE}(3)$ is a pose parameter. In our framework, any implicit function approximation model $S(x, y, z; \mathbf{q}, \mathbf{T})$ can be used.

In this work, we employ the shape class called the superquadrics, a family of geometric shapes that resemble ellipsoids and other quadrics, which can be used to represent diverse shapes ranging from boxes, cylinders, and ellipsoids to bi-cones, octahedra, and other complex symmetric shapes. The implicit equation for a superquadric surface at $\mathbf{T} = \mathbf{I}_4$ ($\mathbf{I}_4$ is $4 \times 4$ identity matrix) has the following form:

$$S(x, y, z; \mathbf{q}, \mathbf{I}_4) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}} - 1 = 0, \tag{4.4.4}$$

where $\mathbf{q} = (a_1, a_2, a_3, e_1, e_2) \in \mathbb{R}^5$ is the shape parameter. In particular, $a_1, a_2, a_3$ controls the sizes and $e_1, e_2$ controls the geometric shapes. Some examples are shown in Figure 2.1. At $\mathbf{T} \neq \mathbf{I}_4$, the equation $S(x, y, z; \mathbf{q}, \mathbf{T})$ can be written with the passive coordinate transformation of $(x, y, z)$ by $\mathbf{T}$, i.e., $S(x, y, z; \mathbf{q}, \mathbf{T}) = S(\mathbf{T}^{-1}(x, y, z); \mathbf{q}, \mathbf{I}_4)$;

see Appendix B.2 for details.

The object recognition problem that we address in this paper can then be posed as follows: given a visual input obtained from a depth camera that typically contains partial views of the objects, we need to predict the superquadric parameters $(\mathbf{q}, \mathbf{T})$ for each object. To bridge the gap between synthetic and real-world vision sensor data, we add noise to the visual input as done in [124, 125]. The predicted object represented by $(\mathbf{q}, \mathbf{T})$ should fit the full object, although only a partial view of the object is given as input. This problem has been recently tackled by [64], where two neural network models that take point cloud data as inputs are employed: (i) object segmentation network [72] and (ii) object full shape and pose recognition network [64]. We include details about the visual input noise, the network architectures, and the training methods of these networks in Appendix B.2.

We call our $\mathrm{SE}(2)$-equivariant pushing dynamics model that uses the superquadric representation a *SuperQuadric Pushing Dynamics Network (SQPD-Net)*.

### 4.4.2 Model-based Pushing Manipulation

Given a visual observation of tabletop objects as a point cloud which we denote by $\mathbf{o}$, our goal is to find a sequence of robot pushing actions $(\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_T)$ that changes the environment for some given task. In this section, we assume that we are given (i) a recognition module $R$ that outputs the objects' poses and shapes, i.e. $R(\mathbf{o}_t) = \mathbf{s}_t$ (throughout, we denote by $\mathbf{s}_t = \{(\mathbf{T}_{t,i}, \mathbf{q}_{t,i})\}_{i=1}^{N}$), and (ii) a pushing dynamics model $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$. Given a task-specific objective function $\mathcal{J}$, we solve the following optimal control problem:

$$\min_{\mathbf{a_1}, \cdots, \mathbf{a_T}} \mathcal{J}(\mathbf{o}_1, \mathbf{a}_1, \cdots, \mathbf{a}_T) = \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) + q(\mathbf{s}_{T+1}) \text{ s.t. } \mathbf{s}_1 = R(\mathbf{o}_1), \ \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t). \tag{4.4.5}$$

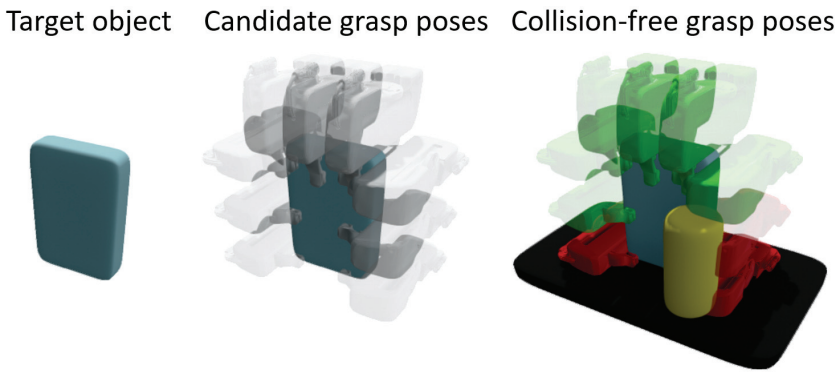


Figure 4.4: Sampling-based grasping criteria.

For tasks we focus in this paper, we set $r(\mathbf{s}_t, \mathbf{a}_t) = 0$ and only use a terminal cost function $q(\mathbf{s}_{T+1})$. We use the sampling-based MPCs [97] (implementation details are in Appendix B.4). Below, we introduce three terminal cost functions for the following pushing manipulation tasks: (i) moving, (ii) singulation, and (iii) grasping. We denote the translation vector and rotation matrix parts of the transformation matrix $\mathbf{T}_{(\cdot)}$ as $\mathbf{t}_{(\cdot)}, \mathbf{R}_{(\cdot)}$, respectively.

**Moving** is a task to move objects to their desired poses. The desired poses are given as $\{\mathbf{T}_{d,i}\}_{i=1}^{N}$, then we define a terminal cost function as

$$q(\mathbf{s}_{T+1}) = \sum_{i=1}^{N} \left( \|\mathbf{t}_{T+1,i} - \mathbf{t}_{d,i}\|_2^2 + \beta \cdot d_{\mathrm{SO}(3)}(\mathbf{I}_3, \mathbf{R}_{d,i}^{-1}\mathbf{R}_{T+1,i}) \right). \tag{4.4.6}$$

**Singulation** is a task to separate objects by more than a certain distance $\tau$. We define a terminal cost function as

$$q(\mathbf{s}_{T+1}) = - \min_{\{(i,j)\in\{1,\cdots,N\}|i>j\}} \big( \min(\|\mathbf{t}_{T+1,i} - \mathbf{t}_{T+1,j}\| - \tau, 0) \big). \tag{4.4.7}$$

**Grasping** is a task to make a target object graspable. Given a target object index $i$, we generate candidate grasp poses for the recognized target object as shown in Figure 4.4 and check collisions with the environment and the other recognized objects;

green grasp poses are collision-free and red poses are not. The terminal cost $q(\mathbf{s}_{T+1})$ is defined to be 0 if at least one collision-free grasp pose exists and 1 otherwise. Further details are provided in Appendix B.5.

## 4.5 Pushing Manipulation Dataset

To train pushing dynamics models, we generate a pushing manipulation dataset in simulation environment (Pybullet). We use the 7-dof Franka Emika Panda robot with a parallel-jaw gripper and an Azure Kinect DK camera sensor mounted on the gripper. The raw input visual observation is a depth image, which is then pre-processed to other 3d representations (e.g., point cloud) as needed.

We generate the pushing manipulation dataset as follows: (i) we place random objects at random poses in the workspace, (ii) sample an action, and (iii) execute the robot pushing action. In this process, we note that the gripper's other parts than the tip can also make contact with the environment.

**Object configuration.** The objects consist of cubes and cylinders with various shape parameters (i.e., width, height, depth for the cube, and radius, and height for the cylinder). The shape parameters are randomly generated, and 18 different objects are generated for each box and cylinder as shown in Figure 4.5. Then, the objects are divided into 9/9 known and unknown sets per shape class. The known objects are used for training the pushing dynamics models, and the unknown objects are used for the performance evaluation of the trained models. For data generation, these objects are dropped on the workspace of 0.512m $\times$ 0.512m $\times$ 0.192m.

**Pushing Action.** To execute an action $(\mathbf{p}, \mathbf{v}) \in \mathbb{R}^6$, (i) the robot first moves so that the gripper's tip is placed at $\mathbf{p}$ and its orientation is set as visualized in Figure 4.6, and then (ii) the robot moves in a way that the gripper's tip moves to $\mathbf{p} + \mathbf{v}$ with a fixed

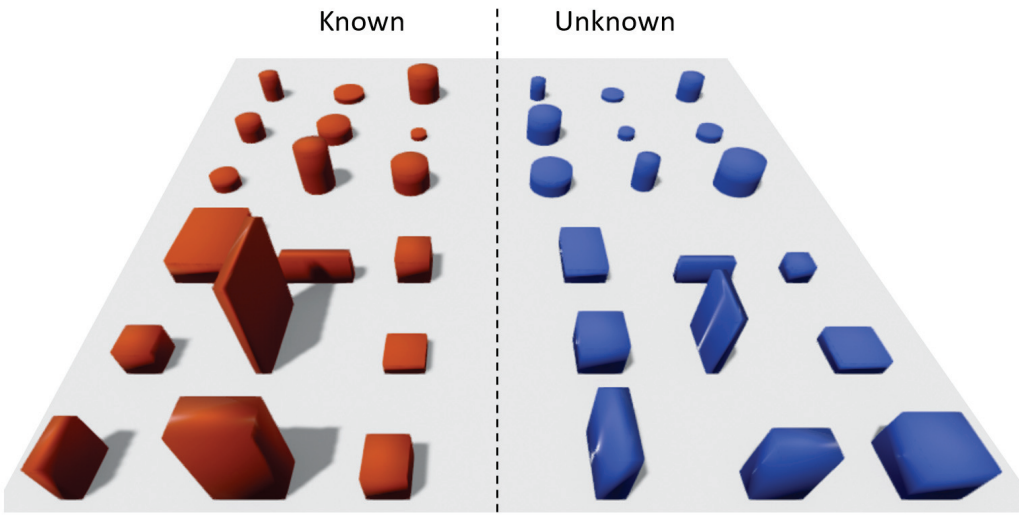


Figure 4.5: Known (red) and unknown (blue) object shapes used for data generation.

orientation.

**Action sampler.** To sample an action $(\mathbf{p}, \mathbf{v}) \in \mathbb{R}^6$, we first randomly choose an object to push and randomly choose the pushing direction among pre-defined 8 angles divided equally between 0 to $2\pi$ towards the center of the object pose. The vector $\mathbf{v} \in \mathbb{R}^3$ is then defined from the pushing direction vector. To determine the pushing start point $\mathbf{p} \in \mathbb{R}^3$, the height of the pushing point is randomly chosen within the candidate heights which belong to 5 pre-defined heights divided equally by the workspace height (i.e., 0.192m) and less than the object's height. The starting point in the x-y plane is also chosen within the 4 candidates on the pushing line as shown in Figure 4.7. For each object, a maximum of 160 action candidates can be generated. For each action, the trajectory of the robot pushing motion is divided into 10 via points and the end effector of the robot reaches the via points sequentially and slowly, making the object as quasi-static as possible.

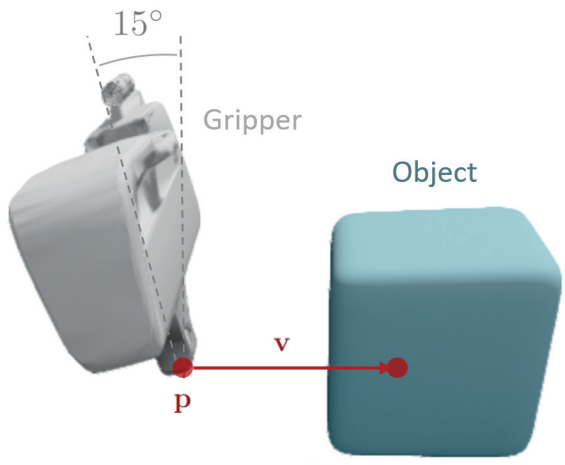


Figure 4.6: Execution of a pushing action.

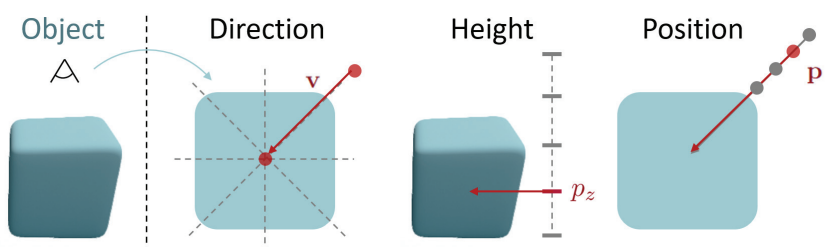


Figure 4.7: Pushing action sampling method for a chosen object.

## 4.6 Experimental Results

In this section, we empirically show that (i) our proposed pushing dynamics model, the SQPD-Net, outperforms the existing state-of-the-art data-driven pushing dynamics models, and (ii) our SQPD-Net can be used for various downstream pushing manipulation tasks, e.g., object moving, singulation, and grasping.

**Shape alignment.** Since we use symmetric shapes for the object sets, various solutions can appear with the different poses of objects when recognition is performed. We introduce a processing technique that standardizes the recognized object shapes in terms of their poses to reduce the complexity of data statistics and accelerate the training of the SQPD-Net.

We recall the superquadric equation for convenience of explanation:

$$S(x, y, z; \mathbf{q}, \mathbf{I}_4) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}} - 1 = 0. \tag{4.6.8}$$

If shape parameters $e_1$ and $e_2$ of the predicted superquadric object are almost equal (i.e., $||e_1 - e_2|| < 0.01$), we rearrange the object pose so that the z-axis is close to the surface normal vector of the table and x-axis is close to the action vector. As the object poses are rearranged, the size parameters $a_1, a_2, a_3$ are also rearranged, e.g., if the x-, y-, and z-axis of the original object pose are rearranged to z-, x-, and y-, the size parameters are changed by $(a_1, a_2, a_3) \mapsto (a_3, a_1, a_2)$. Otherwise, there is no solution for z-axis alignment. So we only rearrange the x-axis of the object pose to be close to the action vector. The illustration of the shape alignment method is shown in Figure 4.8.

**Baseline Methods.** We compare our SQPD-Net with the following baseline methods: *2DFlow* and *SE3-Net* adopted from [34], *SE3Pose-Net* adopted from [35], and *3DFlow* and *DSR-Net* adopted from [40]. The 2DFlow, SE3-Net, and SE3Pose-Net take an organized point cloud as a visual input and predict the flow vectors of the points.

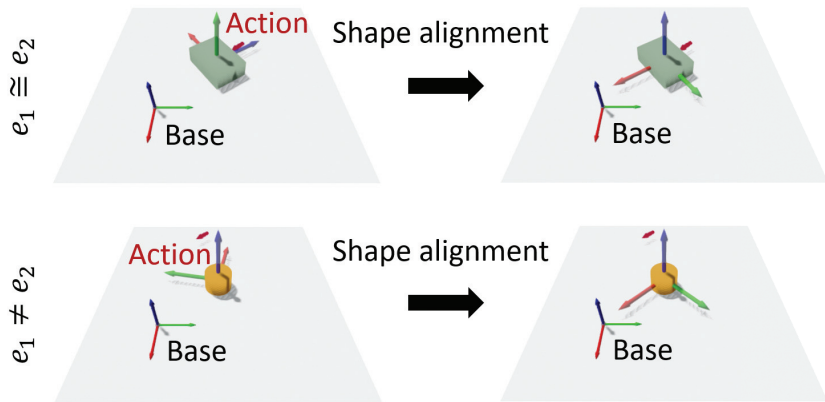


Figure 4.8: Illustration of the shape alignment method for superquadric objects.

The 3DFlow and DSR-Net take a voxelized truncated signed distance field (TSDF) as a visual input and predict the voxel flow. Our SQPD-Net takes the estimated objects' poses and superquadric shape parameters as input and predicts the objects' next poses. While, in the existing approaches, the models directly predict motions from the pre-processed raw visual observations, our model consists of two modules: (i) a pre-trained recognition network $R$ that predicts objects' poses and shape parameters and (ii) the SQPD-Net that predicts the objects' next poses. We denote these two networks together by *R-SQPD-Net*. For the comparison purpose, we also test the case where the ground-truth objects' poses and shape parameters are used as an input for the SQPD-Net and denote it by *GT-SQPD-Net*.

**Evaluation Metrics.** Throughout, we use two types of evaluation metrics for the learned pushing dynamics models: (i) flow error (the lower the better) and (ii) mask intersection over union (mask IoU, the higher the better). First of all, we consider the visible and full flow errors. The visible flow error is the root mean squared error (RMSE) between the ground-truth flows and predicted flows of the points on the visible surface

of the objects, while the full flow error is the RMSE computed with all points from the objects' volumes. Second, we consider the 2D and 3D mask IoUs. The 2D mask IoU is computed by using the depth images and thus only visible surfaces are taken into consideration. On the other hand, the 3D mask IoU is computed with the complete 3D occupancy grid. The full flow error and mask IoU cannot be computed in 2DFlow, SE3-Net, and SE3Pose-Net, because they do not estimate the complete objects' shapes as an intermediate step of the prediction of the pushing dynamics.

### 4.6.1 Equivariance Study

For the purpose of testing the equivariance of the models, we design the following experiment: we train the models with *only one* pushing manipulation data – a 3-tuple $\{\mathbf{o}, \mathbf{a}, \mathbf{o}'\}$ where $\mathbf{o}$ and $\mathbf{o}'$ are current and next observations respectively and $\mathbf{a}$ is pushing action – so that the models overfit the given data. Then, we compare the models' generalization capabilities with test data that are generated by applying random $\mathrm{SE}(2)$-transformation to the data. An ideal equivariant model should produce almost zero error in the test data.

The models are trained with only one pushing data with a single object. Next, we generate nine test data by applying nine random $\mathrm{SE}(2)$ transformations to the training data, and then we evaluate the trained model using these test datasets. The experiments are conducted using ten different training data, each with different objects and poses. Consequently, the total number of test data is 90.

Table 4.1 shows average visible flow errors of the baseline methods and SQPD-Nets, obtained by running the above experiment multiple times with different training data. The 3DFlow and DSR-Net are omitted in this experiment because they cannot make estimations if the transformed actions do not belong to the pre-defined discrete

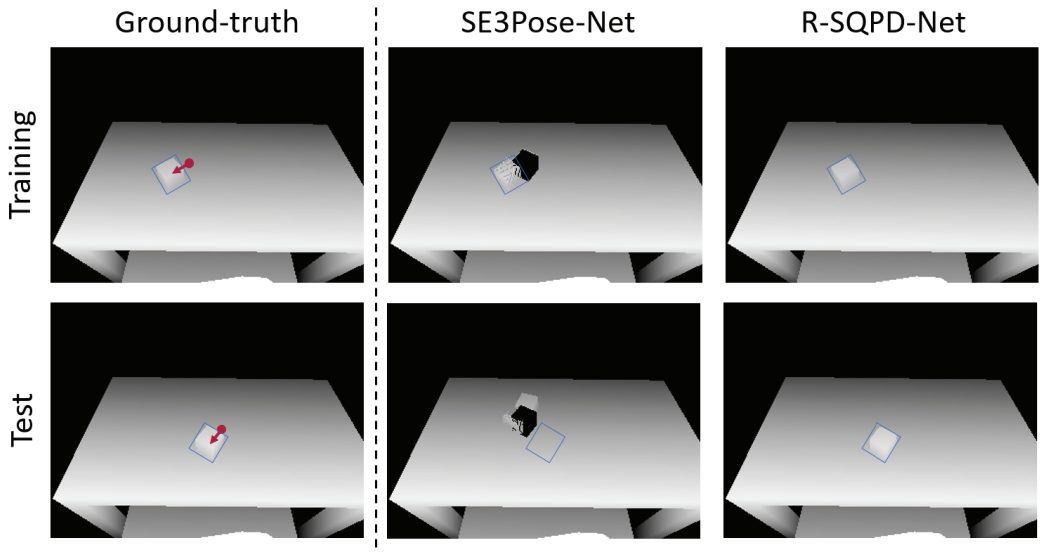


Figure 4.9: Depth images of prediction results. For SE3Pose-Net, after the point cloud moves, the space occupied before is colored black.

Table 4.1: Test visible flow error (cm).

| **METHOD** | visible flow (↓) |
|---|---|
| 2DFlow [34] | 4.73 |
| SE3-Net [34] | 4.73 |
| SE3Pose-Net [35] | 4.72 |
| R-SQPD-Net (ours) | 0.73 |
| GT-SQPD-Net (ours) | 0.02 |

set of actions. The GT-SQPD-Net produces almost zero error as expected while the R-SQPD-Net produces a little error originating from the recognition error. Our SQPD-Nets are much more $\mathrm{SE}(2)$-equivariant compared to the existing works. Figure 4.9 shows an example prediction result from the SE3Pose-Net and R-SQPD-Net; the blue bounding box represents the ground-truth next pose of the object. For the test data, the SE3Pose-Net predicts a completely wrong motion.

More examples of the results of the equivariance study are shown in Figure 4.10. The experimental results, including 2DFlow and SE3-Net, are given in this figure.

### 4.6.2 Pushing Dynamics Learning

We compare the learning performances of the SQPD-Nets and the baseline methods with a large-scale pushing dataset where the training/validation/test data consist of 12000, 1200, and 1200 numbers of 3-tuples $(\{\mathbf{o}, \mathbf{a}, \mathbf{o}'\})$, respectively.

The detailed explanation for the pushing dataset is as follows. For each sequence, the objects are randomly chosen from the known objects, and randomly dropped on the workspace. The number of objects per sequence varies from 1 to 4. We sample and execute a maximum of 20 random pushing actions per sequence, and we stop the sequence when objects' positions are out of the workspace or objects fall down. We collect data until the total numbers of data tuples are 12000/1200/1200 for training/validation/test, respectively. To test the generalization performance for unknown objects, we additionally generate 300 data tuples per the number of objects from 1 to 4, where the objects are randomly chosen from the unknown objects. So, the unknown test dataset consists of 1200 data tuples. We evaluate the trained models on both the known and unknown test datasets.

Figure 4.11 shows the predicted depth images and 3D masks for an example pushing

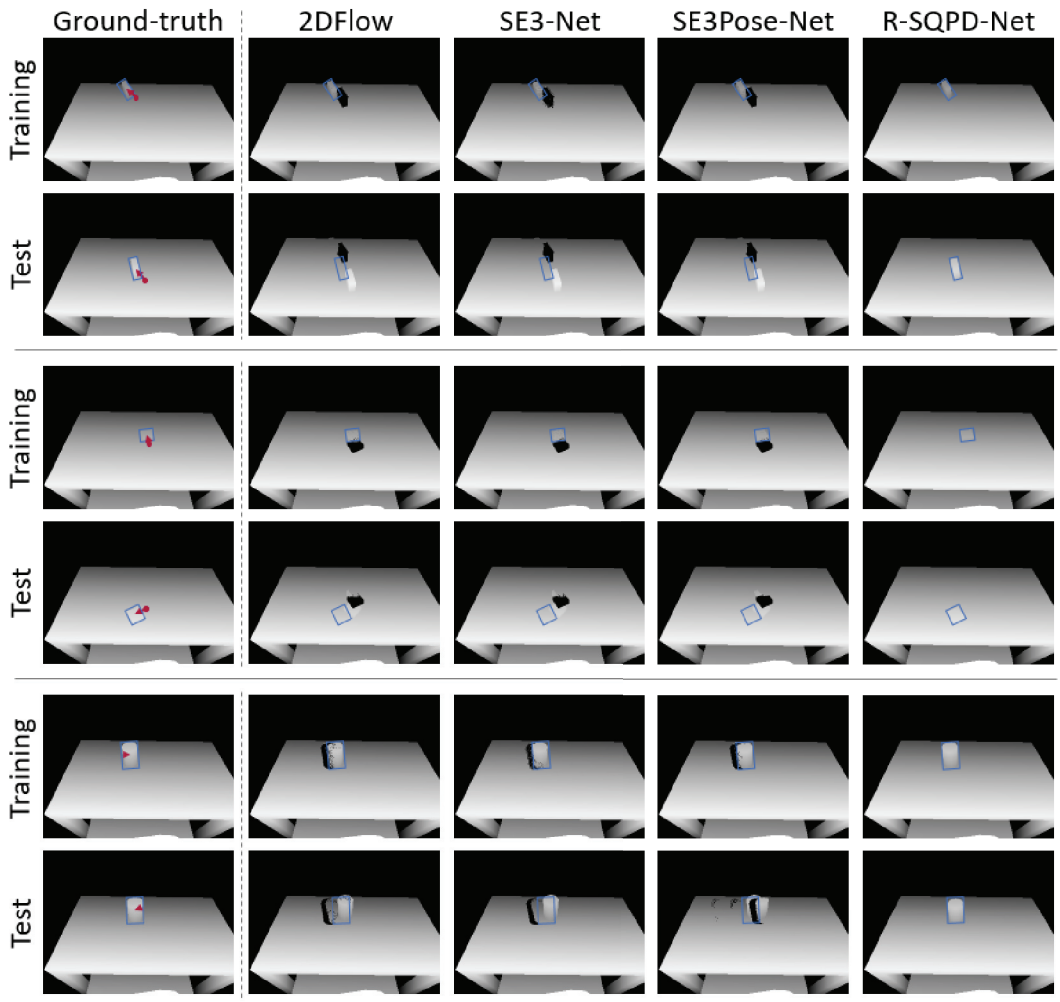


Figure 4.10: The representative three examples of the equivariance study experiments.

Table 4.2: Evaluation metrics computed within test dataset (the unit of flow error is cm).

| | Known | | | | Unknown | | | |
|---|---|---|---|---|---|---|---|---|
| | Flow error (↓) | | Mask IoU (↑) | | Flow error (↓) | | Mask IoU (↑) | |
| **METHOD** | visible | full | 2D | 3D | visible | full | 2D | 3D |
| 2DFlow [34] | 2.179 | - | - | - | 2.180 | - | - | - |
| SE3-Net [34] | 1.631 | - | - | - | 1.701 | - | - | - |
| SE3Pose-Net [35] | 1.639 | - | - | - | 1.712 | - | - | - |
| 3DFlow [40] | 1.818 | 1.859 | 0.747 | 0.699 | 1.697 | 1.719 | 0.755 | 0.698 |
| DSR-Net [40] | 1.325 | 1.331 | 0.720 | 0.705 | 1.531 | 1.524 | 0.665 | 0.632 |
| R-SQPD-Net (ours) | **0.575** | **0.610** | **0.844** | **0.798** | **0.710** | **0.726** | **0.834** | **0.781** |
| GT-SQPD-Net (ours) | 0.519 | 0.379 | 0.903 | 0.888 | 0.638 | 0.485 | 0.888 | 0.868 |

data in the test dataset. As shown in the ground-truth motions (left of Figure 4.11), the red, green, and gray objects are in contact with each other and these three objects move together when the red object is pushed. In this case, our R-SQPD-Net only successfully predicts the complex interactive motions of the objects. Table 4.2 shows the evaluation metrics computed within the test data and shows that our SQPD-Nets outperform the other baseline methods by significant margins.

More examples for the results of pushing dynamics learning are shown in Figure 4.12 and Figure 4.13. The trend of the experimental results is also similar to the experimental results in Figure 4.11. The results of 2DFlow, SE3-Net, and SE3Pose-Net, which predict the flow of the point cloud, show that they have difficulties predicting the motions of the objects at all. 3DFlow and DSR-Net attempt to predict the motion more than the above methods, but the prediction accuracy is still not enough. Our SQPD-Net accurately predicts the motion of moving objects.

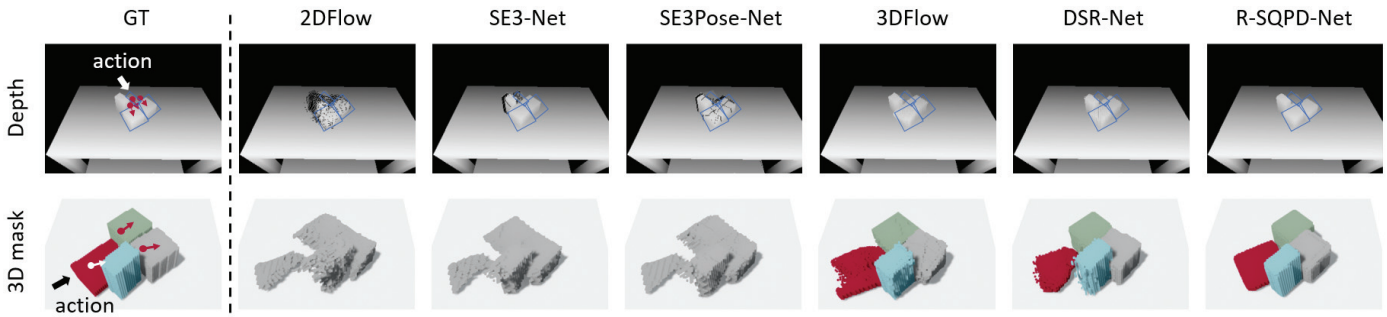


Figure 4.11: Depth images and 3D masks of the ground-truth next scene and predicted scenes. *Upper*: Depth images where the blue bounding boxes represent the ground-truth next poses of the green and gray objects. *Lower*: (i) (incomplete) 3D masks converted from the depth images for 2DFlow, SE3-Net, and SE3Pose-Net and (ii) predicted complete 3D masks for 3DFlow, DSR-Net, and R-SQPD-Net.

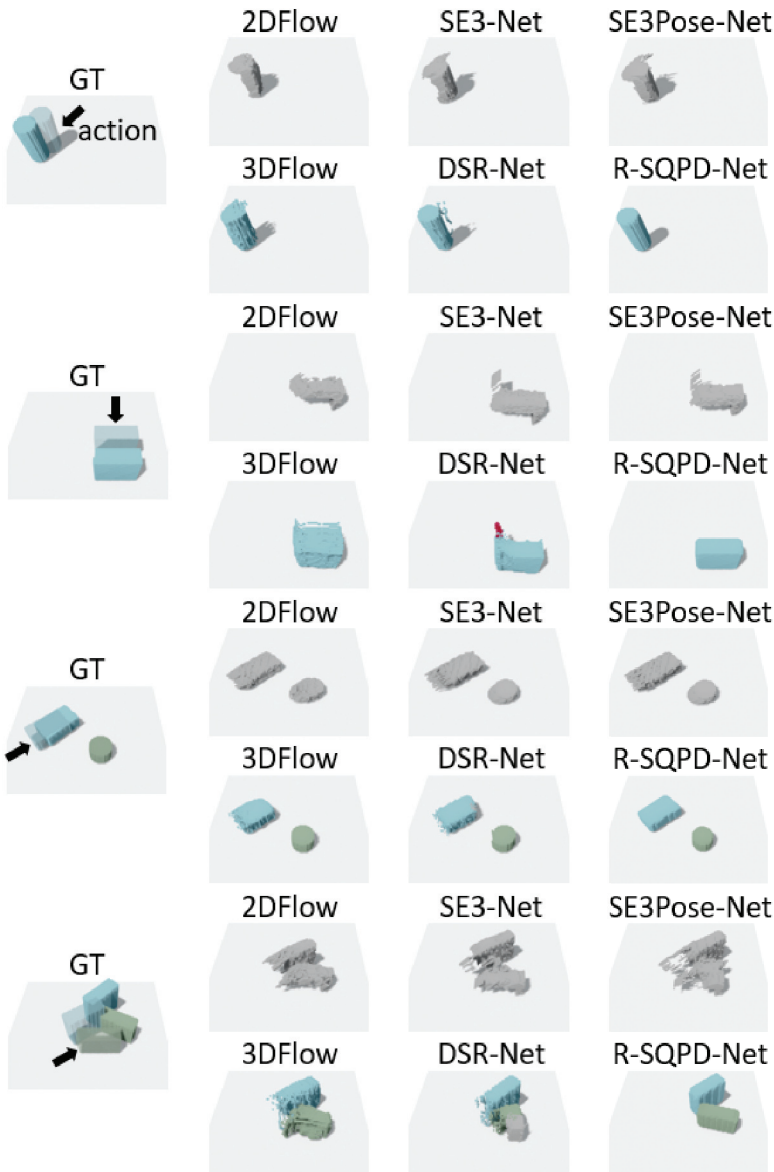


Figure 4.12: The representative examples of the pushing dynamics learning experiments for the number of objects 1 and 2.

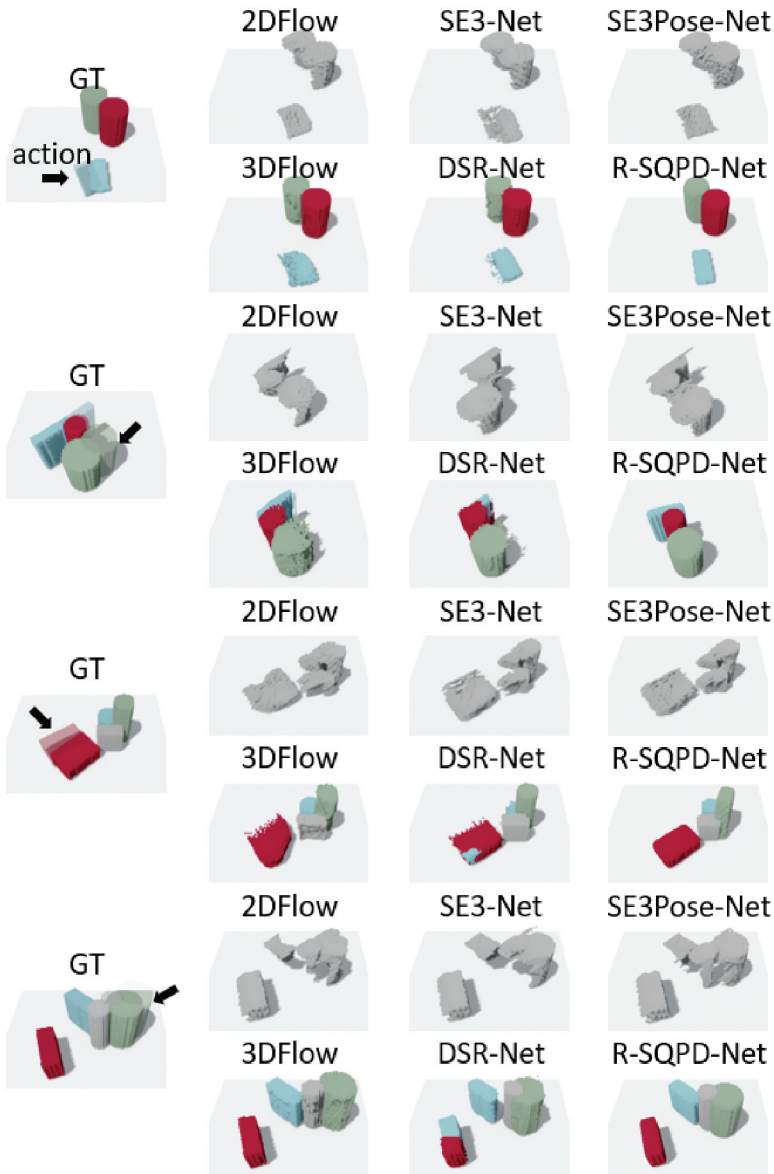


Figure 4.13: The representative examples of the pushing dynamics learning experiments for the number of objects 3 and 4.

### 4.6.3 Pushing Manipulation using R-SQPD-Net

In this section, we use the R-SQPD-Net trained in Chapter 4.6.2 and conduct the pushing manipulation tasks introduced in Chapter 4.4.2 (moving, singulation, and grasping) in both simulation and real-world. For the real-world experimental setup, we use various box- or cylinder-like objects as shown in Figure 4.14; the same objects are used in simulation experiments. Since we directly apply the R-SQPD-Net trained in simulation to the real physical environment, it is reasonable to ask about the sim-to-real transfer issue. In our experiments, we use slow pushing motions to generate quasi-static movements of the objects and thus minimize the sim-to-real gap (for quasi-static object movements, the dynamical properties of the objects and environment, e.g., mass, friction coefficient, become less affective [126]).

For pushing manipulation experiments on simulation and real-world experiments, we also use box- and cylinder-like objects inspired by YCB dataset [87]. For each task, we use the object sets shown in Figure 4.15. In the moving and singulation tasks, a total of 10 experiments – 5 experiments for 2 object sets – are performed. In the grasping tasks, the "grasping large" and the "grasping clutter" tasks include 5 experiments for one object set. The objects have different poses for each experiment. Simulation manipulation experiments are conducted by making objects of the same size as these.

For action sampling, we use the same action sampler used in data generation as described in Chapter 4.5 on the recognized object shapes. When sampling the pushing actions, we reject the actions that collide with the objects or the table are rejected using the gripper collision detection method introduced in Appendix B.5.

The success criterion for each task is as follows: (i) moving is success when the distance between the positions of the objects and the goal positions is less than 5cm on average, (ii) singulation is success when all distances between the objects are more


Figure 4.14: Real-world experimental setting.

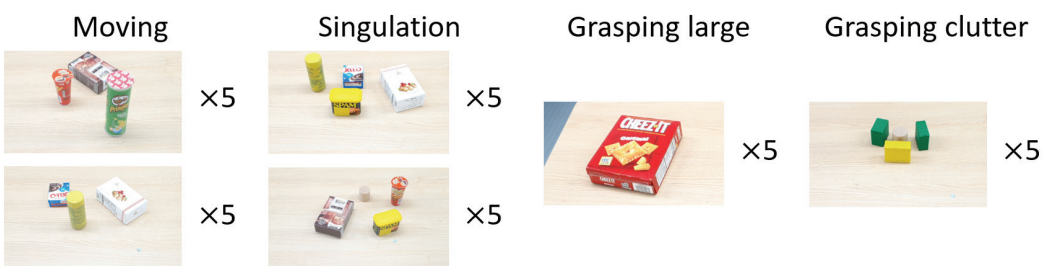


Figure 4.15: Object sets used in moving, singulation, and grasping tasks.

than $\tau$ = 20cm, and (iii) grasping is success when at least one grasp pose is found. We note that the candidate grasp poses for the target object are resampled every timestep $t$. If the task trial does not succeed within 10 timesteps, the trial is considered as a failure.

Figure 4.16 shows some real-world manipulation results for various tasks. For the moving task (*first row*), we set the desired positions $\mathbf{t}_{d,i}$ as $(0.3, \mathbf{t}_{0,i,y}, \mathbf{t}_{0,i,z})$ and $\beta = 0$ in equation (4.4.6). For the singulation task (*second row*), we set $\tau = 20$ (cm) in equation (4.4.7). For the grasping tasks (*third and fourth row*), we sample about 15 to 30 candidate grasp poses for the target recognized objects. For all three examples, our approach can find a series of pushing actions that successfully perform the desired tasks. Notably, for the grasping tasks, without using ad hoc objective functions, the robot realizes how to re-configure the objects so that feasible grasp poses can be found for the target objects: (i) the robot pushes the large and flat object to the edge of the table and (ii) the robot pushes the surrounding objects to make the surrounded target object graspable.

Table 4.3 shows the manipulation success rates in simulation and real-world experiments. There are some failure cases for pushing manipulation, and some representative examples of the failure cases are shown in Figure 4.17. The first case is the result of a failure to recognize the shape; this leads to inaccurate calculation of task objective function or unexpected collision with the objects (left of the Figure 4.17). Also, to prevent the object from falling down, we provide a constraint so that the center of the object is inside the workspace. The constraint sometimes did not work and the object falls down from the table since recognition and trained dynamics are not perfectly accurate (center of the Figure 4.17). The second case prevalent is the failure of sim-to-real transfer of the pushing dynamics model. Such cases occur when pushing the standing long cylinder, in detail, when the same pushing action is performed, the cylinder does not fall

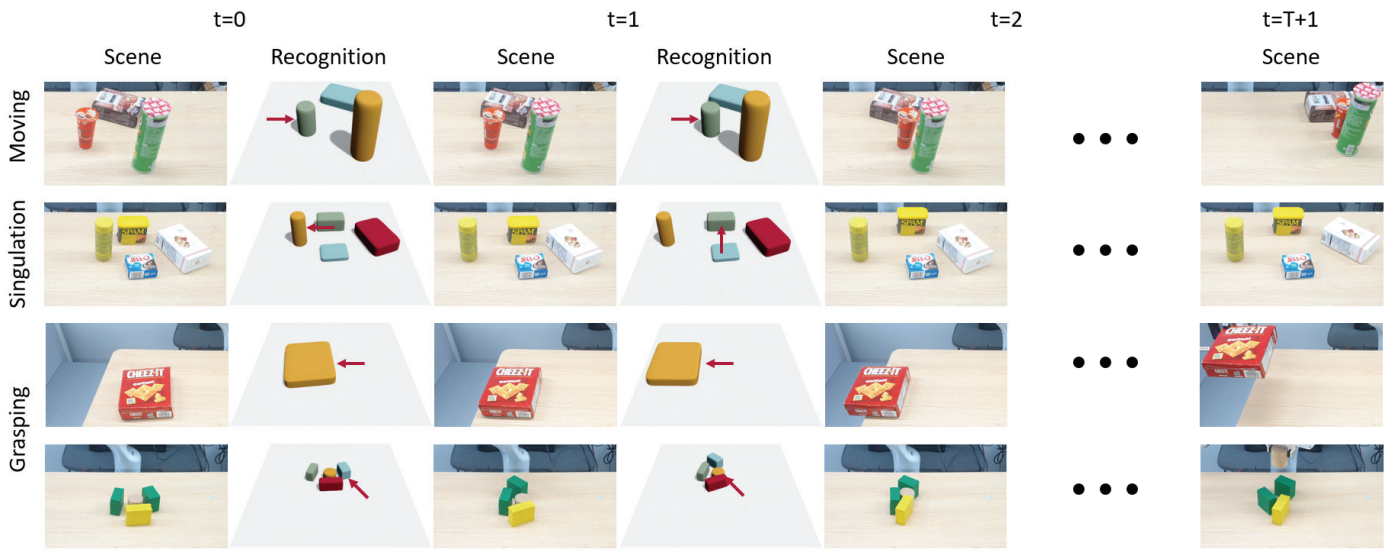


Figure 4.16: Real-world manipulation results using R-SQPD-Net for moving, singulation, and grasping tasks (for the *fourth row* case, the target object is the cylinder surrounded by the three cubes). The red arrow at each recognition step means the optimal pushing action.


Figure 4.17: The representative examples of the failure cases for pushing manipulation.

Table 4.3: Simulation and real-world manipulation results.

| **TASK** | Simulation | Real |
|---|---|---|
| Moving | 9/10 | 8/10 |
| Singulation | 9/10 | 8/10 |
| Grasping clutter | 4/5 | 4/5 |
| Grasping large | 4/5 | 3/5 |

over in the simulation but sometimes falls over in the real-world experiment (right of the Figure 4.17). These unexpected object motions cause the performance drop of the pushing manipulation tasks.

## 4.7 Additional Experimental Results

### 4.7.1 Comparison with Data Augmentation

In this subsection, we qualitatively compare the performance of the pushing dynamics model with and without $\mathrm{SE}(2)$-equivariant module and shape alignment module. We use the same dataset, network architecture, and training method, except the fact that we perform random data augmentation with $\mathrm{SE}(2)$ transformations and object coordinate frame transformations during training as shown in Figure 4.18.

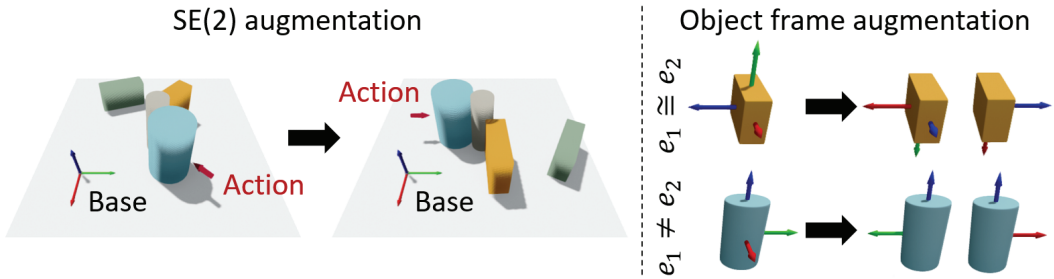


Figure 4.18: Random data augmentation during training.

**Baseline Methods.** We compare the the SQPD-Nets with and without $\mathrm{SE}(2)$ equivariant module and shape alignment module. We distinguish between these models using $\mathrm{SE}(2)$ equivariant module and shape alignment module by an "+ $\mathrm{SE}(2)$" and "+ Shape" after the network name. For example, the SQPD-Net with only $\mathrm{SE}(2)$ equivariant module is denoted by "SQPD-Net + $\mathrm{SE}(2)$". For the comparison purpose, the ground-truth objects' poses and shape parameters are used for the all SQPD-Nets (i.e., GT-SQPD-Nets). Except for this subsection, "SQPD-Net" denotes "SQPD-Net + $\mathrm{SE}(2)$ + Shape".

**Evaluation Metrics.** We use two types of evaluation metrics: (i) flow error and (ii) pose error. The flow errors including visible and full flow errors are the same ones described in Chapter 4.6. The pose error measures the mean position error and orientation error between the ground truth and the predicted object poses.

Figure 4.19 shows an example of pushing data in the test dataset. In this example, we check whether the trained models (with and without the $\mathrm{SE}(2)$-equivariant module and shape alignment module) can generate equivariantly transformed objects' next poses given transformed inputs. We consider one input scene and two transformed input scenes (see the first row of Figure 4.19). The second and third row of Figure 4.19 show the predicted objects' next poses of the models, where we draw the output scene from $\mathrm{SE}(2)$-transformed input in the same view angle with the original scene by applying

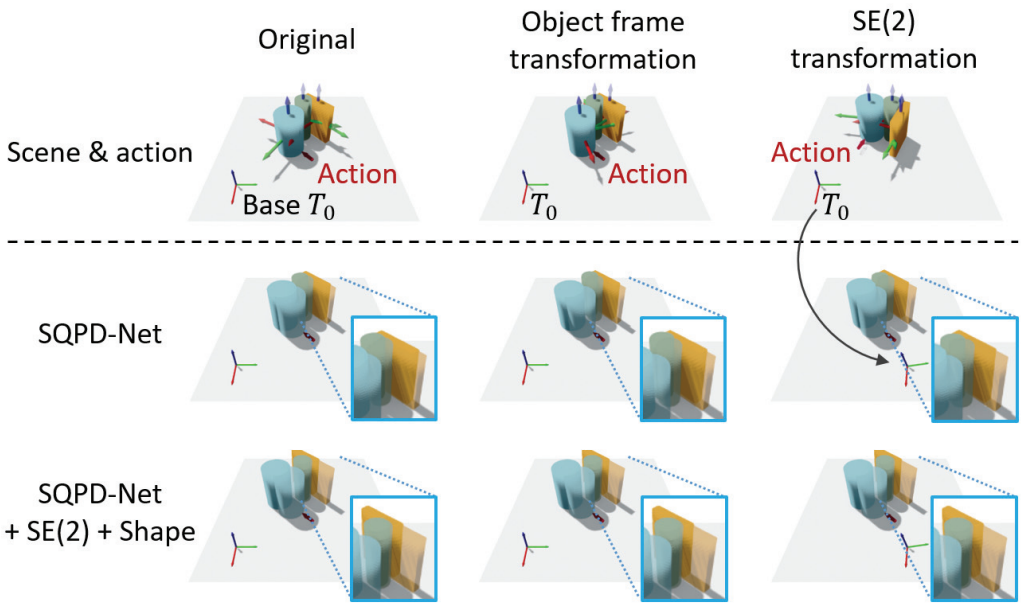


Figure 4.19: A qualitative comparison of the pushing dynamics models trained with and without our $\mathrm{SE}(2)$-equivariant module and shape alignment module. For each figure, transparent and bold objects represent the scene before and after a pushing action is applied, respectively.

inverse transformation to the scene for easy comparison.

For the vanilla method without the module, despite the use of data augmentation in the training process, the predictions are not equivariant with respect to the group actions (see the zoomed-in part of the second row of Figure 4.19). The predictions by our method show equivariant transformations as the input scene transformed (third row of Figure 4.19). This implies that our model is fully equivariant to the input transformations, whereas the baseline model with data augmentation (i.e., pure SQPD-Net) is not.

Table 4.4 shows the evaluation metrics for the learned pushing dynamics models.

Table 4.4: Evaluation metrics for the learned pushing dynamics models; the units for the flow error and pose error (pos. and ori.) are cm and (cm and degree), respectively.

| | Flow error (↓) | | Pose error (↓) | |
|---|---|---|---|---|
| **METHOD** | visible | full | pos. | ori. |
| SQPDNet | 1.327 | 1.083 | 1.270 | 10.427 |
| SQPDNet + Shape | 1.195 | 0.995 | 1.175 | 7.960 |
| SQPDNet + $\mathrm{SE}(2)$ | 1.183 | 0.950 | 1.158 | 8.846 |
| SQPDNet + $\mathrm{SE}(2)$ + Shape | 0.712 | 0.550 | 0.676 | 7.606 |

The $\mathrm{SE}(2)$ equivariance significantly enhances the accuracy when compared to the vanilla model with data augmentation, and employing the additional shape alignment method yields even greater performance improvements. For qualitative comparison, we consider another example of pushing data in the test dataset as shown on the left of Figure 4.20. As shown in the ground truth, the green, blue, and gray objects move together due to the contact when the green object is pushed. Ideally, the learned pushing dynamics model should be able to predict not only the next pose of the *pushed object* but also the next poses of *other surrounding objects* due to these complex interactions. The right of Figure 4.20 shows the predicted objects' next poses of the models trained with and without the modules. The vanilla model performed poorly despite data augmentation. The models using the only one module succeeded in making predictions closer to the ground truth than the vanilla model, but there are still some errors, especially in predicting the motion of the blue object. The model considering both modules shows more accurate performance than the other models (see the blue object in Figure 4.20).

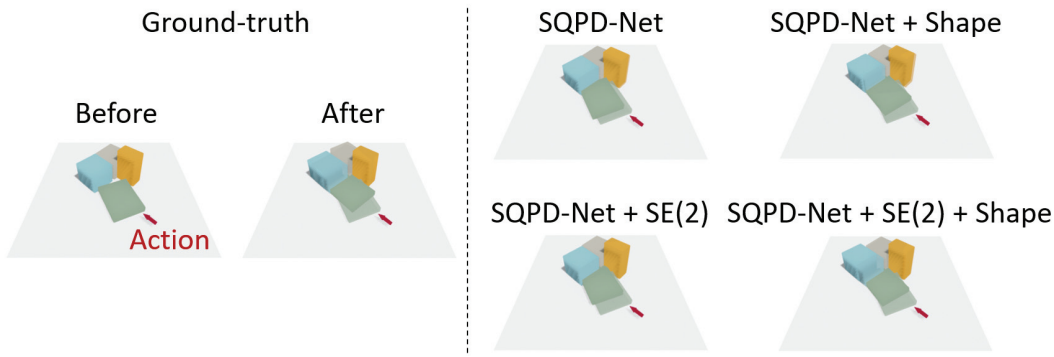


Figure 4.20: A qualitative comparison of the pushing dynamics models trained with and without our modules. For each figure, transparent and bold objects represent the scene before and after a pushing action is applied, respectively.

### 4.7.2 Pushing Dynamics Learning on Real-world Pushing Data

In this experiment, we train our SQPD-Net on the real-world dataset and compare the performance with the physics-based simulator (PyBullet). We generate a real-world pushing manipulation dataset using four cube-shaped objects of different sizes shown in Figure 4.21. In this case, one scene contains only one of these four objects. For each scene, the object is placed in a fixed pose (rotated 22.5° degrees with respect to the robot's base frame), then we sample and execute one of the 20 different random pushing actions per object; so the total number of data tuples is 80. To annotate the pose of the object before and after the pushing action, we use the Iterative Closest Point (ICP) algorithm that matches the point cloud observation to the ground-truth object model. We then divide the four objects into three known objects and one unknown object; the known objects are used to train our R-SQPD-Net, and the unknown objects are used to evaluate the trained model. The number of cases of dividing objects is four, and the model is trained and evaluated in all these cases. We denote the model trained in box


Figure 4.21: Objects for real-world pushing data.

$a, b, c$ and evaluated in box $d$ as "$a, b, c \rightarrow d$". To get the predictions of PyBullet, we drop the superquadric object given by our trained recognition model into the simulator and let the robot simulator perform the same pushing action that is performed in the real-world.

Figure 4.22 shows the motion of the objects predicted by the PyBullet simulator and trained R-SQPD-Net. As shown, both PyBullet and R-SQPD-Net tend to predict the position and approximate direction of the objects after movement. However, R-SQPD-Net predicts the orientation of the predicted object much better. This is due to some differences between the dynamic natures of PyBullet and the real-world. Since our model is trained directly from data collected in the real-world, it can predict the dynamics

Table 4.5: Translation and rotation errors computed with real-world data.

| | PyBullet | | R-SQPD-Net | |
|---|---|---|---|---|
| **MODEL** | translation (cm) | rotation (°) | translation (cm) | rotation (°) |
| 2,3,4 → 1 | 1.025 ± 0.520 | 8.129 ± 7.206 | **0.681 ± 0.349** | **4.304 ± 4.091** |
| 1,3,4 → 2 | 1.007 ± 0.599 | 4.000 ± 3.658 | **0.972 ± 0.580** | **3.581 ± 3.110** |
| 1,2,4 → 3 | 0.963 ± 0.702 | 13.038 ± 7.357 | **0.800 ± 0.347** | **4.197 ± 3.508** |
| 1,2,3 → 4 | 0.847 ± 0.541 | 4.584 ± 3.308 | **0.674 ± 0.512** | **2.995 ± 2.346** |

of pushing objects well. In order to quantitatively verify this fact, translation and rotation errors are measured for the poses of the predicted objects as shown in Table 4.5. We have confirmed that R-SQPD-Net outperforms PyBullet overall, and especially, our model performs much better in terms of rotation errors. In conclusion, we verify that our model can be successfully trained on real-world data. Also, the simulator is somewhat accurate, but to perform more accurate pushing manipulation in the real-world, we should collect a dataset from the real-world and train the model on this dataset.

### 4.7.3 Pushing Manipulation via Interaction

To verify that our model R-SQPD-Net works well for cases where multi-object interactions are essential to achieve the goal, we have conducted a new pushing manipulation task named **interactive moving**.

**Interactive moving** is similar to **moving** task in that the goal is to move some objects, but here it is a task that moves one target object to the desired pose. In this case, the robot *should not push the target object.* The current pose of the target object and the desired pose are given as $\mathbf{T_t}$ and $\mathbf{T}_d \in \mathrm{SE}(3)$ respectively, then we define a terminal cost function as

$$q(\mathbf{s}_{T+1}) = \|\mathbf{t}_{T+1} - \mathbf{t}_d\|_2^2, \tag{4.7.9}$$

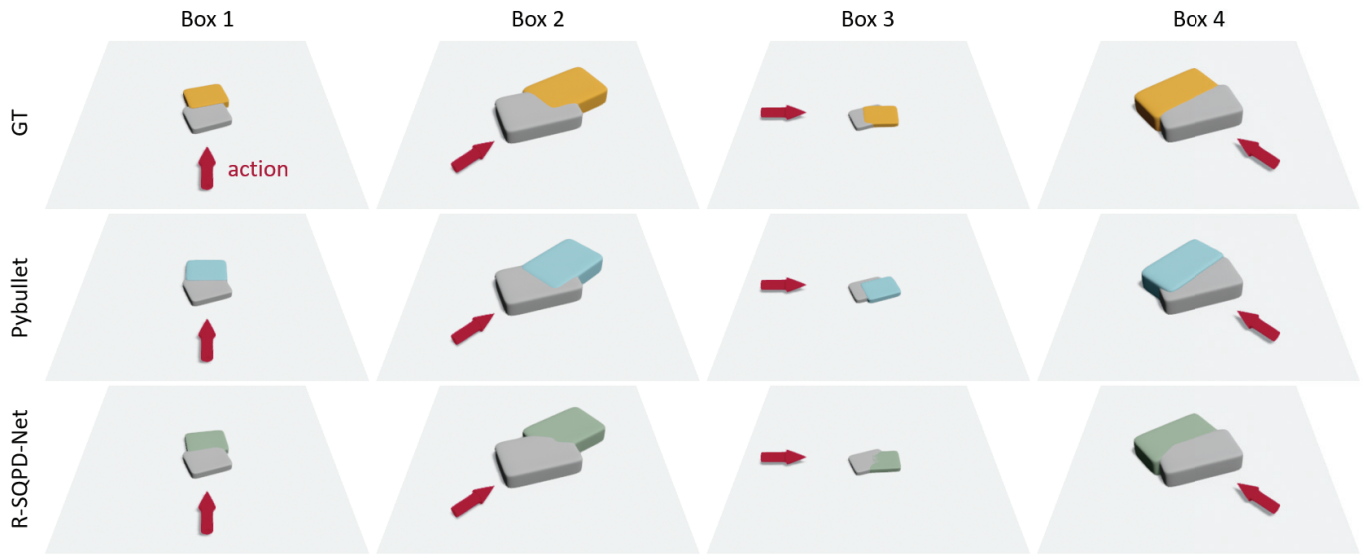


Figure 4.22: Real-world ground-truth pushing data (yellow) and pushing dynamics prediction results of PyBullet physics simulator (blue) and trained R-SQPD-Net (green). The initial pose of the object before being pushed is indicated in gray color.

where the notation $\mathbf{t}_{(\cdot)}$ denotes the translation vector of $\mathbf{T}_{(\cdot)}$. We set the desired positions $\mathbf{t}_d$ as $(0.3, 0.0, \mathbf{t}_{0,z})$; the task is described in the left of Figure 4.23. Since the robot cannot directly push the target object, it must move the target object to the desired position by pushing other objects. That is, this task essentially requires interaction between multi-objects. We sample 100 action sequences and the time horizon of each sequence is set to one.

Figure 4.23 shows some manipulation results for interactive moving tasks. In the case of the first row, it succeeded in moving the target object to the desired position by pushing the yellow box. Even in the case of the slightly difficult case in the second row, our approach can find the series of actions that sequentially push the yellow and green boxes so that successfully perform the desired task. In conclusion, We verify that R-SQPD-Net can learn multi-object interaction well and that it can be used properly in manipulation tasks.

### 4.7.4 Pushing Manipulation using Physics-based Simulator

In this experiment, we compare the performance of our R-SQPD-Net and the physics-based simulator (Pybullet) in pushing manipulation tasks, highlighting the importance of learning a data-driven pushing dynamics model. Since our method recognizes the shapes of tabletop objects at an intermediate stage, a straightforward approach to using a physics simulator as a dynamics model involves inserting the recognized shapes into the simulator. Accordingly, we compare the following baselines with dynamics models: R-SQPD-Net, Pybullet with recognition results, denoted as "Pybullet (w/ Recog.)," and Pybullet with ground-truth object shapes, denoted as "Pybullet (w/ GT.)." We test these dynamics models on the same tasks used in Chapter 4.4.2 – including moving, singulation, grasping clutter, and grasping large – in the simulation environment. In the

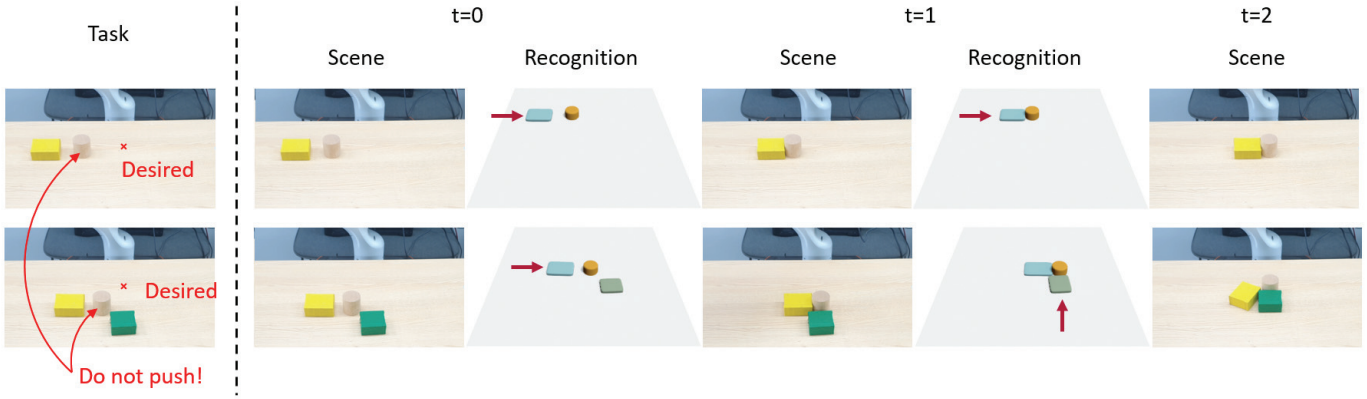


Figure 4.23: Real-world manipulation results using R-SQPD-Net for the interactive moving task (the target object is the cylinder surrounded by cubes). The red arrow at each recognition step means the optimal pushing action.

Table 4.6: Comparison of R-SQPD-Net and the Pybullet simulator in simulation-based manipulation experiments.

| | Moving | | Singulation | | Grasping clutter | | Grasping large | |
|---|---|---|---|---|---|---|---|---|
| **MODEL** | Succ. | Steps | Succ. | Steps | Succ. | Steps | Succ. | Steps |
| R-SQPD-Net | 19/20 | - | **18/20** | **5.17** | **9/10** | **2.50** | 9/10 | 2.56 |
| Pybullet (w/ Recog.) | 19/20 | - | 15/20 | 5.43 | 7/10 | 3.29 | 9/10 | **2.22** |
| Pybullet (w/ GT) | 20/20 | - | 19/20 | 4.73 | 9/10 | 2.56 | 9/10 | 2.22 |

moving and singulation tasks, a total of 20 experiments with random object sets are performed. For the grasping tasks, the "grasping large" and "grasping clutter" tasks each comprise 10 experiments for one object set, with the objects in different poses for each experiment.

Table 4.6 presents the manipulation success rates and the average number of action steps required for success in simulation experiments. In the case of moving tasks, we do not report the number of steps due to the large variation depending on the scenario. We have observed that our R-SQPD-Net outperforms the PyBullet simulator with recognition results, particularly in singulation and grasping clutter tasks that involve interactions between objects. This superiority is evident both in terms of success rates and average action steps. The performance of Pybullet with ground-truth shapes is similar to, or better than, the other baselines. This suggests that inaccuracies in recognition can lead to performance degradation when using a physics-based simulator. By contrast, our R-SQPD-Net achieves higher performance than Pybullet by compensating for these recognition inaccuracies through its data-driven model. In conclusion, we find that jointly learning the data-driven model SQPD-Net with a recognition model yields better performance in vision-based pushing manipulation problems than using a physics simulator combined with a recognition model.

## 4.8 Conclusion

This paper has proposed a $\mathrm{SE}(2)$-equivariant pushing dynamics model. Using the superquadric representations of object shapes, we have proposed a SuperQuadric Pushing Dynamics Network (SQPD-Net). Through extensive empirical validations, we confirm that the SQPD-Net significantly outperforms the existing state-of-the-art visual pushing dynamics models. Moreover, we have verified that the SQPD-Net can be used for various pushing manipulation tasks.

**Limitations and Future Directions.** First, since SQPD-Net considers single superquadric shaped objects, it is not trivial to apply it directly to more complex or non-convex shapes. As the researches on representing objects in multiple superquadrics progress [102, 103], extending our approach to multiple superquadric-shaped objects remains a future work. Second, the dynamics prediction task becomes challenging when pushing an object with a non-uniform mass distribution since different mass distributions will lead to different motions. In this case, if we can consistently predict the reference poses of the objects (e.g., pre-specified poses in CAD models), our $\mathrm{SE}(2)$-equivariant model is applicable regardless of the mass distribution. Since predicting reference poses is not easy with only depth images, this is a limitation of our approach. As one possible solution, additional information such as RGB images should be utilized [127, 128]. Lastly, there could be some situations where the $\mathrm{SE}(2)$-equivariance does not apply; in this case, our approach can be detrimental. One example is that the friction coefficients are different in different regions of the table. As a research direction to overcome this, a locally $\mathrm{SE}(2)$-equivariant model – $\mathrm{SE}(2)$ space is divided into several subspaces and the model is equivariant only within each subspace – can be considered.

# 5

# Search-for-Grasp: Superquadric Recognition for Mechanical Search

## 5.1 Introduction

Finding and grasping a desired target object on a cluttered shelf – where the target is occluded by unknown objects and initially not visible to a vision sensor – is a significant challenge in robotic manipulation. This task is further complicated when the pose of the vision sensor is fixed. In such scenarios, the robot must rearrange surrounding objects to identify the target's pose and grasp it, all while avoiding collisions with the shelf and nearby objects. The geometric characteristics of the shelf, which allow visual observations solely from the front and limit the manipulator's workspace, add another layer of complexity.

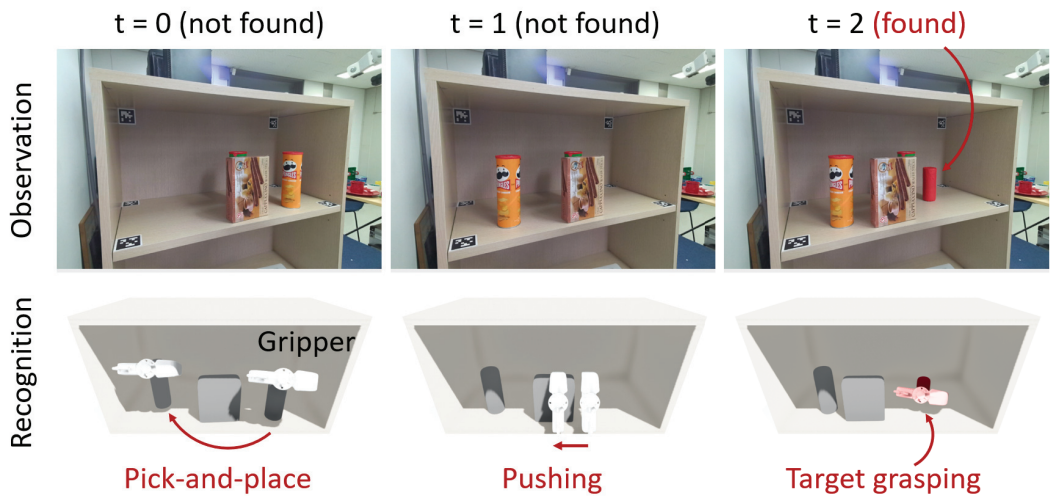


Figure 5.1: A 3D recognition-based mechanical search and grasping of the target object (red cylinder).

Previous research on finding and grasping the target object has focused on (i) simpler environments, such as flat tables or bins [129, 130, 131, 132, 133, 134, 94, 135], or (ii) complex shelf environments but with the assumption that all object shapes and poses are known (e.g., assuming the shelf's inside can be seen through its transparent top) [136, 137, 138, 139, 140, 141, 142]. Recent studies have ventured into more realistic shelf scenarios, where the surrounding non-target objects are unknown, termed the *mechanical search*. However, these often depend on specialized, elongated tools to manipulate the objects within tight shelf spaces [143, 144, 145]. As a result, the challenge of finding and grasping the target object in a densely populated shelf environment using a standard robot gripper, devoid of specialized tools, remains open.

To our knowledge, we are the first to offer a practical method for finding and grasping the target object on a cluttered shelf populated with unknown objects, using a standard robot gripper (as shown in Figure 5.1). Like previous studies, we utilize pushing

and pick-and-place actions. However, we employ a standard two-finger gripper, introducing several practical challenges. For instance, generating a collision-free robot path during the grasp the target or during a pick-and-place action becomes non-trivial when only partial visual information is accessible.

In this paper, we propose a comprehensive framework for mechanical search and grasping. Specifically, we introduce two indicator functions (denoting the target's candidate pose by $x \in \mathrm{SE}(3)$): (i) an existence function $f(x)$ that indicates if the target can be present at $x$ and (ii) a graspability function $g(x)$ that indicates if the target at $x$ is graspable. The objective then becomes to rearrange the objects until only one existable and graspable pose $x^*$ remains, i.e., there exists a unique $x^* \in \mathrm{SE}(3)$ such that $f(x^*) = 1$ and $g(x^*) = 1$. Leveraging the dynamics models of these functions, we formulate a model-based optimal control with a suitably-designed objective function.

Furthermore, we provide practical algorithms that leverage a 3D object recognition model to effectively estimate of the functions $f_t, g_t$ and their corresponding dynamics models. We employ a recent 3D recognition model rooted in *superquadric* primitives [64]. Notably, the superquadric representation allows for rapid collision checks, depth image rendering, and the utilization of pushing dynamics models [65]. To mitigate accumulated estimation errors during optimal control, we adopt the model predictive control with a short time horizon. A distinguishing feature of our method, compared with existing methods, is the use of a 3D recognition model and leveraging the recognition results for mechanical search.

Through experiments in both simulations and real-world scenarios, we have validated the effectiveness of our 3D reconstruction-based approach. Our method consistently finds and grasps the target object using a standard two-finger robot gripper, even in the presence of noise from vision sensor data in real-world settings.

## 5.2 Related Works

### 5.2.1 Mechanical Search on Shelves

**Object search and mechanical search.** The goal of object search (sometimes called as target object retrieval) is to find a target object from unknown environments. Some works have focused on *active perception* problem of making decisions of the sequence of camera poses to find a target object using a camera-mounted mobile robot [146, 147, 148, 149, 150, 151, 152]; recently, deep learning-based methods have been proposed in terms of target-driven visual navigation [153, 154, 155, 156]. However, in a more complex environment, such as a cluttered environment on a tabletop or an environment where objects are placed on a shelf, it may be impossible to find a target object by controlling only the position of the camera. To solve these issues, *interactive perception*-based methods – in which the robot can change the environment to find the target object – have been proposed. Object search using interactive perception is recently called mechanical search.

**Mechanical search methods.** The earlier works have attempted to solve the problem of searching the target object via performing pushing or grasping actions to the surrounding objects in algorithmic manners [129, 130, 131]. Although these methods have made a significant contribution to the research topic of mechanical search, many assumptions are made in the environment to make the problem tractable, and they are generally computationally complex and therefore slow. To improve these methods (e.g., relaxing the assumptions), several works have proposed a POMDP model and its solver (e.g., DESPOT [132] or POMCP [133]) for mechanical search. A recent work provides a generalized formulation of mechanical search and solve this problem effectively using

deep learning-based perception module (e.g., object segmentation and recognition network) and grasping module (e.g., pre-trained Dex-Net) [134]. The follow-up paper proposes a novel perception module and a policy that minimizes the support of learned occupancy distributions obtained from the perception, and claims that the proposed method outperforms the previous methods [94]. Another work propose a 3D shape recognition-based approach that predicts the occluded geometries from the vision sensor image and then utilize this information to efficiently find the target object [135]. Our work is also in the spirit of [135] in utilizing the 3D shape recognition module to solve the mechanical search problem efficiently (e.g., reduce the number of total actions), but we use implicit representation for the recognized objects to utilize them for efficient and effective action decision (see Appendix 5.2.3).

**Mechanical search on shelves.** As the shelves are often used to store the objects in home environments or logistic warehouses, mechanical search on shelves are being studied as an important research topic [143, 144, 145]. Object manipulation on the shelves is more challenging because of the several task constraints: the manipulator must not collide with the shelf, the objects cannot be removed from the shelf, and only a nearly-lateral camera view is available. These constraints limit the action space of the manipulator and the amount of visual information that can be obtained from the vision sensor. An earlier work proposes an extension of the previous method named lateral access X-ray [94] to solve laterally-accessible mechanical search [143]. The follow-up studies use novel tools to extend the robot action space from just pushing to pushing-and-grasping [144] and stacking [145].

The main difference from these works is that the graspability of the target object is taken into consideration when performing mechanical search. The existing studies use a custom long suction gripper specialized for mechanical search, so the graspability of the target object does not need to be considered separately. On the other hand, when using

the standard robot gripper (e.g., parallel-jaw gripper), we need to find a trajectory that does not collide with the other surrounding objects and the shelf to grasp the target object. To find a non-collide trajectory, the 3D reasoning of the current scene is inevitable for 6-DOF grasping, and accordingly, a 3D recognition model has been adopted in our method. The existing works only perform 2D reasoning for the scene since they do not have to consider the graspability, and the point that we perform mechanical search task adopting 3D recognition is also different from other works.

### 5.2.2 Object Rearrangement for Target Object Grasping

The object rearrangement generally refers to the problem of finding the feasible paths of the objects that move the objects from their initial configuration to desired final configuration, and in fact, a lot of various object rearrangement studies has been conducted; in this subsection, we only focus on the object rearrangement researches for grasping the target object. An earlier work propose an algorithm to remove the surrounding objects using prehensile manipulation to grasp the target object without robot-object collisions [136]. Since the action space is limited only by prehensile manipulation, object rearrangement algorithms using non-prehensile manipulation have also been conducted; for example, these algorithms are based on tree-search [137], persistent homology [138], and semi-autonomous tele-operation [139]. We note that unlike mechanical search, these papers assume that the information about the target object (and sometimes information about the environment) is known. Other works focus on more general cases where the target object is possibly occluded [140, 141, 142]. If the target object is occluded, the proposed performs an algorithm to find the target similar to the mechanical search. It is worthy to note that our problem is more challenging since the surrounding objects can be removed in previous studies, but cannot in our case. Also, these studies first find the

target object and then grasp it when the target is occluded; we argue in this paper that finding a target object while simultaneously considering whether it can be graspable is more efficient.

**Grasping the invisible.** It is valuable to note that our problem setting is the closest to the problem considered in [93]. Their work also considers the problem of grasping the target object while considering the mechanical search problem. They named this problem *grasping the invisible* and introduce a deep learning-based end-to-end method, more specifically, a critic function that maps the visual observations to the expect rewards of robot pushing or grasping actions. This paper is the same in that it addresses the same problem as ours, but the proposed methods so far are limited to a specific environment and may require a lot of data for the model to generalize to other environments. We develop a method that can be applied in various environments by using object recognition, which is known to be well generalizable to unseen scenes [64, 65], rather than an end-to-end method.

### 5.2.3 Shape Recognition-based Robot Manipulation

We refer to Chapter 4.2.3 for detailed reviews of shape recognition-based robot manipulation methods.

## 5.3 A General Framework for Mechanical Search and Grasping

In this section, we propose a general framework for mechanical search and provide two specific algorithms: (i) search-and-grasp and (ii) search-for-grasp, using the model-based optimal control formulation. The search-and-grasp method executes actions first

to find the target and then executes additional actions to rearrange the objects to make the target graspable, while the search-for-grasp method integrates the search and grasp processes into a cohesive framework so that the target object's graspability is taken into account in the search phase.

Our focus lies specifically on scenarios where the target object is known – where its information is given as the color and geometry – and is fully occluded by other unknown objects. We assume that the robot can interact with objects either by pushing or pick-and-place actions to rearrange the objects. When the robot performs the pick-and-place action, it is only allowed to place an object in another empty space on the shelf (i.e., object stacking is not allowed). For the target object, we assume that it is placed in a straight-up pose (i.e., not tilted) and not stacked on top of the other surrounding objects. Also, when the robot rearranges the surrounding non-target objects, the target object is assumed to remain stationary.

We denote the target object's pose by $x \in \mathrm{SE}(3)$; since the target object is not tilted and stacked, it is sufficient to represent the object pose as $x \in \mathrm{SE}(2)$. Instead of dealing with the continuous pose space $\mathrm{SE}(2)$, to simplify numerical computations, we restrict our attention to a finite subset $\mathcal{X} \subset \mathrm{SE}(2)$. The two core components in our framework are (i) an existence function $f : \mathcal{X} \to \{0, 1\}$ and (ii) a graspability function $g : \mathcal{X} \to \{0, 1\}$. The existence function $f(x)$ indicates whether the target object can be present at the pose $x$ or not. For example, given an observation shown in Figure 5.2 (*Upper*), consider two candidate target poses $x \in \mathcal{X}$ (i.e., the red cylinders in Figure 5.2 *Lower*). For $f(x)$ to be 1, the rendered image must match the observation (there are other conditions as well, details are in the next section), otherwise, $f(x) = 0$. In practice, considering the discretization resolution of $\mathcal{X}$, $f(x)$ is considered to be 1 if the target object pose can exist near $x$. Naturally, it captures the uncertainty of the target object's pose because $\sum_{x \in \mathcal{X}} f(x)$ represents the number of possible object poses; the

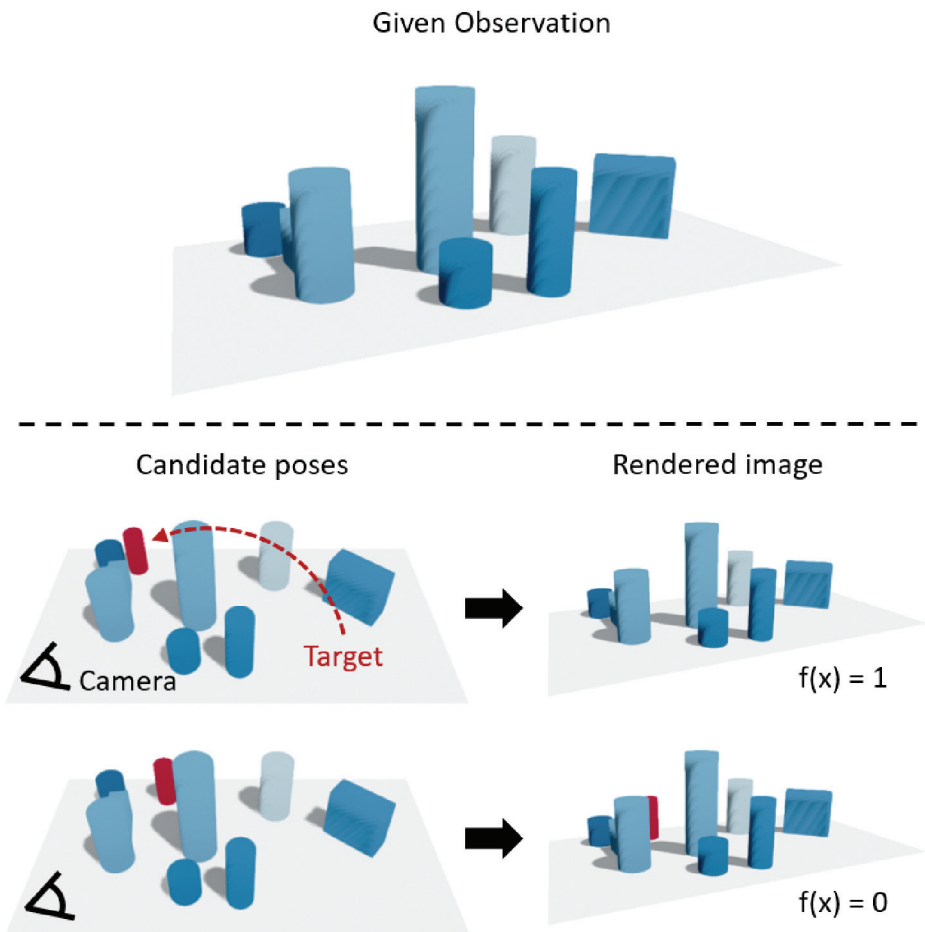


Figure 5.2: Illustration on the existence function $f(x)$. *Upper*: Observation. *Lower*: Candidate poses and hypothetical rendering results.

greater $\sum_{x\in\mathcal{X}} f(x)$, the more uncertain. We assume $\mathcal{X}$ is sufficiently densely discretized so that $\sum_{x\in\mathcal{X}} f(x)$ is lower bounded by 1 (i.e., the target object must exist in at least one $x \in \mathcal{X}$). The graspability function $g(x)$ indicates whether the target object at the pose $x$ is graspable or not.

Denote the existence and graspability functions at a discrete timestep $t$ by $f_t(x)$ and $g_t(x)$ and a pushing or pick-and-place action by $a_t$. We assume that the dynamics models for $f_t(x)$ and $g_t(x)$ are given and denoted by $\mathcal{F}$ and $\mathcal{G}$ such that $f_{t+1}(x) := \mathcal{F}(f_t, a_t)(x)$ and $g_{t+1}(x) := \mathcal{G}(g_t, a_t)(x)$. Additionally, we assume that the uncertainty of the target object pose is not increasing in $t$, i.e., $\sum_{x\in\mathcal{X}} f_t(x) \geq \sum_{x\in\mathcal{X}} \mathcal{F}(f_t, a_t)(x)$, regardless of $a_t$. This non-increasing uncertainty assumption comes from the prior assumption that the target object does not move when an action is applied because it is impossible for an object suddenly can exist where it could not originally exist.

Given initial $f_0$ and $g_0$ and dynamics models $\mathcal{F}$ and $\mathcal{G}$, we formulate a *search-for-grasp method* as a model-based optimal control. Specifically, the goal is to find a sequence of actions $a_0, \ldots, a_T$ for a fixed time $T$, so that $f_T(x^*) = 1$ for only one $x^* \in \mathcal{X}$ – if there are more than one, then we cannot specify at which $x$ the target object exists – and that is graspable, i.e., $g_T(x^*) = 1$. For this purpose, we formulate the following model-based optimal control problem:

$$\min_{a_1,\ldots,a_T} \sum_{x\in\mathcal{X}} f_T(x) + \alpha f_T(x)(1 - g_T(x)), \tag{5.3.1}$$

$$\text{s.t. } f_{t+1}(x) = \mathcal{F}(f_t, a_t)(x), \tag{5.3.2}$$

$$g_{t+1}(x) = \mathcal{G}(g_t, a_t)(x), \tag{5.3.3}$$

where $\alpha$ is a hyperparameter. Minimizing the first term in (5.3.1) makes $f_t$ have only one $x^*$ such that $f_T(x^*) = 1$ – since the existence function is lower bounded by 1 –, and minimizing the second term enforces $g_T(x^*) = 1$. Without the second term, even

though the target object becomes visible, there is no guarantee that it will be graspable.

In practice, we prefer to perform a minimum number of actions needed for the task, rather than for a pre-determined fixed length $T$. We introduce a discount factor $\gamma \in (0, 1]$ and propose a modified version of the optimal control problem:

$$\min_{a_1,\ldots,a_T} \sum_{t=0}^{T-1} \gamma^t (\mathcal{J}_{t+1} - \mathcal{J}_t) = \gamma^T \mathcal{J}_T + \frac{1-\gamma}{\gamma} \sum_{t=1}^{T} \gamma^t \mathcal{J}_t - \mathcal{J}_0, \tag{5.3.4}$$

where $\mathcal{J}_t := \sum_{x \in \mathcal{X}} f_t(x) + f_t(x)(1 - g_t(x))$. As $\gamma \to 1$, the new objective function converges to the original one in (5.3.1) since $\mathcal{J}_0$ is constant. For $\gamma < 1$, $\mathcal{J}_t$ for $t < T$ are now taken into account, and thus minimizing (5.3.4) leads to accomplishing the task quickly. A $\gamma$ that is too small can make the problem difficult, so an appropriate level of $\gamma$ needs to be found.

Additionally, we propose a computationally lighter version, a *search-and-grasp method*. In the cost term $\mathcal{J}_t$ in (5.3.4), the computation cost of $g_t(x)$ is very high since it requires the planning of multiple collision-free paths (as we explain in the next section in more details). The function $g_t(x)$ must be calculated for all $x$ such that $f_t(x) = 1$, thus as $\sum_x f_t(x)$ is bigger, the computation cost increases. In the search-and-grasp method, we first find the target by using the first term only $\mathcal{J}_t = \sum_x f_t(x)$. Then, after the target is found at $x^*$, we use the second term $\mathcal{J}_t = 1 - g_t(x^*)$ to rearrange the objects to make the target graspable. Since the target pose is already specified at the search phase, in the second stage, we need to compute the graspability function $g_t$ only at one $x^*$, which significantly reduces the total computation time.

## 5.4 3D Object Recognition-based Mechanical Search

In this section, as one practical way to implement our framework, we propose an object recognition-based approach. We assume that a scene contains multiple unknown objects

and a known target object, and the RGB-D camera's pose is fixed so can only capture one side of the scene (e.g., a shelf containing objects from the front view). From the partial view information of the scene, our strategy is to first recognize the shapes and poses of the objects and then to use the recognition results to estimate the existence and graspability functions $f$ and $g$ and the dynamics models $\mathcal{F}$ and $\mathcal{G}$. For the 3D recognition model, we include the details about the architectures and the training details in Appendix C.1.

### 5.4.1 Existence and Graspability Function Estimates $\hat{f}$ and $\hat{g}$

Given a partial visual observation $\mathbf{o}$, let a set of estimated superquadric shape parameters and poses be denoted by $\mathbf{s} = \{(\mathbf{q}_i, \mathbf{T}_i)\}$. Further, we assume that an indicator variable $c \in \{0, 1\}$ – indicates whether the target object is visible (i.e., $c = 1$) or not (i.e., $c = 0$) in the observation $\mathbf{o}$ – is available. If target is visible (i.e., $c = 1$), we assume that the target object's pose can be estimated accurately. Using these estimates $\mathbf{s}$ and $c$, we obtain the estimates for the existence and graspability functions denoted by $\hat{f}(x; \mathbf{s}, c)$ and $\hat{g}(x; \mathbf{s})$.

The key idea is to locate the recognized superquadric objects in the simulation, as well as the target object at a hypothetical pose $x \in \mathcal{X}$. If $c = 1$, the existence function $\hat{f}$, since we know at which $x^*$ the target object exist, is 1 only at $x^*$, i.e., $\hat{f}(x^*; \mathbf{s}, c = 1) = 1$ and 0 at other $x \in \mathcal{X}$. If $c = 0$, $\hat{f}(x; \mathbf{s}, c = 0)$ is defined to be 1 if (i) depth rendering results with and without the target object at $x$ are identical and (ii) there is no collision between the recognized objects, the environment, and the target object. Otherwise $\hat{f}(x; \mathbf{s}, c = 0) = 0$. In detail, the existence function $\hat{f}(x)$ can be calculated by the multiplication of the inverse-visibility function $\hat{f}_d(x)$ and collision

function $\hat{f}_c(x)$, i.e.,

$$\hat{f}(x) = \hat{f}_d(x)\hat{f}_c(x).$$

The inverse-visibility function $\hat{f}_d : \mathcal{X} \to \{0, 1\}$ is a function in which $\hat{f}_d(x) = 1$ if the depth rendering results with and without the target object at $x \in \mathcal{X}$ are identical (i.e., target object is invisible) and $\hat{f}_d(x) = 0$ otherwise. The collision function $\hat{f}_c : \mathcal{X} \to \{0, 1\}$ is a function in which $\hat{f}_c(x) = 1$ if there is no collision between the recognized objects, the environment, and the target object at $x \in \mathcal{X}$ and $\hat{f}_c(x) = 0$ otherwise. The depth rendering and collision checking can be performed efficiently using 3D recognition results, and the details on the computation of these functions are given in Appendix C.2.

The graspability function $\hat{g}(x; \mathbf{s})$ is defined to be 1 if we can find a collision-free grasping trajectory of the robot gripper, where all possible collisions between the robot arm, gripper, environment, and multiple objects should be taken into account. The grasp planning can also be performed efficiently using 3D recognition results, and the details on the computation of graspability functions are given in Appendix C.3.

### 5.4.2 Approximate Dynamics Models $\hat{\mathcal{F}}$ and $\hat{\mathcal{G}}$

Then, we construct the approximate dynamics models for $f_t$ and $g_t$ by using (i) the dynamics of $\mathbf{s}_t$ denoted by $\mathbf{s}_{t+1} = \mathcal{S}(\mathbf{s}_t, \mathbf{a}_t)$ – where we just transform the selected object for the pick-and-place action and use pre-trained SE(2)-equivariant pushing dynamics model for pushing action [65] – and (ii) the function estimates $\hat{f}$ and $\hat{g}$. Since $\hat{f}$ takes the classification label $c$ as an input, we need a dynamics model for $c_t$, but it is hardly possible to know if the target will be visible or not in the future given an action $\mathbf{a}_t$. We take a conservative strategy and assume that the visibility of the target object $\tilde{c}_{t+1}$ does not change at $t + 1$, i.e., $\tilde{c}_{t+1} = c_t$. Then, for given $f_t$ and $g_t$ at time

$t$, the approximate dynamics models are defined as follows:

$$\hat{\mathcal{F}}(f_t, \mathbf{a}_t)(x) := f_t(x)\hat{f}(x; \mathbf{s}_{t+1}, \tilde{c}_{t+1}); \qquad \hat{\mathcal{G}}(g_t, \mathbf{a}_t)(x) := \hat{g}(x; \mathbf{s}_{t+1}), \tag{5.4.5}$$

where $\mathbf{s}_{t+1} = \mathcal{S}(\mathbf{s}_t, \mathbf{a}_t)$ and $\tilde{c}_{t+1} = c_t$. The existence function $\hat{\mathcal{F}}(f_t, \mathbf{a}_t)(x)$ is 1 if both $f_t(x)$ and $\hat{f}(x; \mathbf{s}_{t+1}, c_{t+1})$ are 1, which guarantees the non-increasing uncertainty condition for $\mathcal{F}$, i.e., $\sum_{x\in\mathcal{X}} f_t(x) \geq \sum_{x\in\mathcal{X}} \hat{\mathcal{F}}(f_t, \mathbf{a}_t)(x)$.

### 5.4.3 Sampling-based Model Predictive Control

With these approximated functions and their dynamics, we solve an approximated model -based optimal control as described below:

$$\min_{a_1,\ldots,a_T} \sum_{t=0}^{T-1} \gamma^t (\mathcal{J}_{t+1} - \mathcal{J}_t), \tag{5.4.6}$$

$$\text{s.t. } \mathcal{J}_t = \sum_{x\in\mathcal{X}} \hat{f}_t(x) + \alpha \hat{f}_t(x)(1 - \hat{g}_t(x)), \tag{5.4.7}$$

$$\hat{f}_t(x) = \hat{f}_d(x)\hat{f}_c(x), \tag{5.4.8}$$

$$\hat{f}_{t+1}(x) = \hat{\mathcal{F}}(\hat{f}_t, a_t)(x), \tag{5.4.9}$$

$$\hat{g}_{t+1}(x) = \hat{\mathcal{G}}(\hat{g}_t, a_t)(x), \tag{5.4.10}$$

where an initial observation $\mathbf{o}_0$ and function estimates $\hat{f}_0(x) = \hat{f}(x; \mathbf{s}_0, c_0), \hat{g}_0(x) = \hat{g}(x; \mathbf{s}_0)$ are given. Since both the function estimates and their dynamics models have numerical errors, we perform the model predictive control (MPC) where we iterate the following procedure for $T$ times: for $t = 0, \ldots, T-1$ (i) update initial estimates of $\hat{f}_t, \hat{g}_t$ from a new observation $\mathbf{o}_t$, (ii) solve the above model-based optimal control with a short time horizon $M < T$, and (iii) take only the first action $\mathbf{a}_t$.

**Implementation details.** To solve the iterative optimization in MPC (where we set $T = 10$), we use $M = 3$ and take a sampling-based approach. The actions space $\mathcal{A}$,

whose elements are either pushing or pick-and-place actions, is defined based on the object recognition results and as a discrete set that can be efficiently searched through sampling (sampling details can be found below). We stop the MPC iteration if the target is visible and graspable, so the total number of actions can be less than $T$. The objective function is slightly modified so that, in $\mathcal{J}_t = \sum_{x \in \mathcal{X}} \hat{f}_t(x) + \hat{f}_t(x)(1 - \hat{g}_t(x))$, the graspability function $\hat{g}_t(x)$ – which originally could take 0 or 1 – now can take $1, 0, -1, -2, -3, \ldots$ where $\hat{g}_t(x) = -n + 1$ indicates that generated grasping trajectories for the target at $x$ collides with at least $n$ objects. This modified objective function provides more dense signals, making the optimization problem easier to solve, while not changing the optimal solution. The details for modified graspability function can be found in Appendix C.3.

### 5.4.4 Action Space and Action Sampling Method

As noted above, the action space $\mathcal{A}$ is composed of pushing or pick-and-place actions. In this section, we describe the details of the pushing and pick-and-place actions, and then describe the action sampling methods for manipulation. The details of the pushing action and the pick-and-place action including sampling processes and scene prediction are described in Figure 5.3.

**Pushing action.** A pushing action is defined by $(\mathbf{T}_{\text{push}}, \mathbf{d}) \in \text{SE}(3) \times \mathbb{R}^3$, where $\mathbf{T}_{\text{push}} \in \text{SE}(3)$ is an initial gripper pose and $\mathbf{d} \in \mathbb{R}^3$ is a displacement of the gripper. To sample the initial gripper pose $\mathbf{T}_{\text{push}}$, an index $i_r$ from the recognized object indices $i = 1, ..., N$ is selected and then a direction (left or right) for pushing the selected object $i_r$ is selected. Then $\mathbf{T}_{\text{push}}$ is defined from the pose and the shape parameters of the selected object $(\mathbf{q}_{i_r}, \mathbf{T}_{i_r})$ and the direction to push; (i) the gripper is tilted about 30 degrees along the $y$-axis of the gripper frame and (ii) the distance between the selected

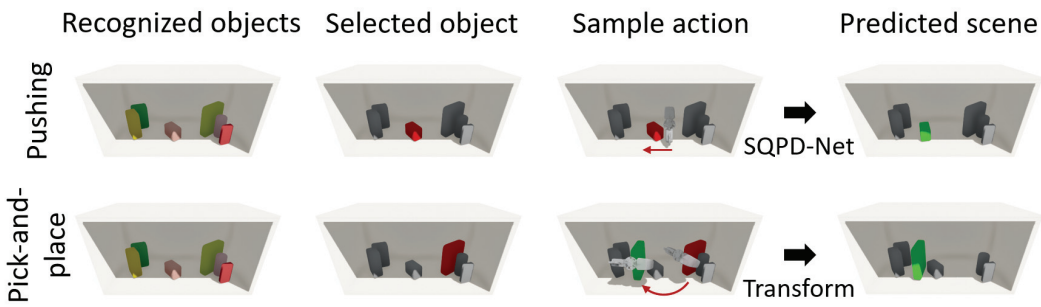


Figure 5.3: Sampling process and predicted scene after applying the action for pushing and pick-and-place actions.

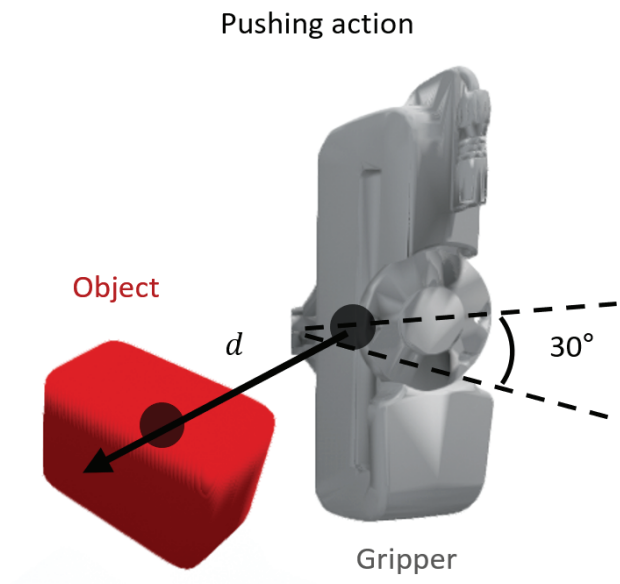


Figure 5.4: Visual description of the pushing action.

object and the gripper (i.e., Chamfer distance between point clouds sampled from the meshes) is set to 1cm. The pushing action is described in Figure 5.4. If the gripper at the pose $\mathbf{T}_{\text{push}}$ collide with the surrounding objects, the action is rejected; for collision checking, we use the method used in Appendix C.2. The displacement of the gripper $\mathbf{d}$ is sampled from a discrete set $\{5, 10, 15\}$cm. For each sampled action, we predict the next state $\mathbf{s}_{t+1}$ using a pre-trained pushing dynamics model named SQPD-Net [65].

**Pick-and-place action.** A pick-and-place action is defined by $(\mathbf{T}_{\text{grasp}}, \mathbf{T}_{\text{place}}) \in \text{SE}(3) \times \text{SE}(3)$, which are gripper poses when it grasps and places the object, respectively. To sample the grasp pose $\mathbf{T}_{\text{grasp}}$, an index $i_g$ from the recognized object indices $i = 1, ..., N$ is selected and a grasp pose is selected from the candidate grasp poses of the selected object $(\mathbf{q}_{i_g}, \mathbf{T}_{i_g})$; we generate the grasp poses using the same strategy we used in Appendix C.3. Then a gripper pose to place $\mathbf{T}_{\text{place}}$ is selected from the poses in which the grasped object does not overlap (i.e., do not collide with) other surrounding objects and the shelf; we use the collision checking method used in Appendix C.2. We check the collision of the trajectory when grasping and placing and the action is rejected if there is a collision; we use the same method in Appendix C.3 when checking collision. For each sampled action, we predict the next state $\mathbf{s}_{t+1}$ by just applying the transformation $\mathbf{T}_{\text{grasp}}^{-1}\mathbf{T}_{\text{place}}$ to the selected object.

**Action sampler.** We first check which objects can be pushed or grasped (for pick-and-place action); whether an object can be pushed or grasped is noted in the action description section above. Let $I_p = \{i_{p1}, \ldots, i_{pN_p}\}$ and $I_g = \{i_{g1}, \ldots, i_{gN_g}\}$ be the set of indices of the graspable and pushable objects, respectively. Then, we randomly choose 30 indices from the multi-set $I_g + I_p := \{i_{g1}, \ldots, i_{gN_g}, i_{p1}, \ldots, i_{pN_p}\}$ allowing duplicates. For each chosen index, we select one action (pushing action if index is in $I_p$ and pick-and-place action if index is in $I_g$) through the method described above. The action sequences for MPC are sampled in the similar manner.

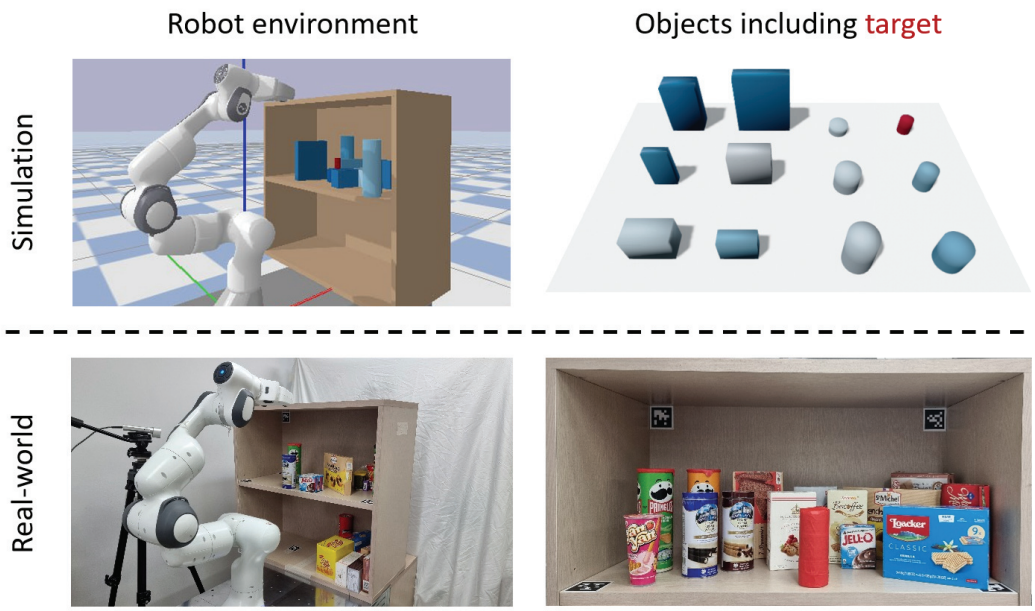


Figure 5.5: The left column shows the simulation and real environments, and in the right column, objects used in each environment are visualized. In particular, the target object is marked in red in the simulation; the red-taped can is the target object in the real experiment.

## 5.5 Experiments

In this section, we conduct a comparative analysis of the two proposed methods, the search-and-grasp and search-for-grasp methods, and evaluate both methods to show robust performance to find and grasp the occluded target object in both simulation and real-world environments.

**Environment.** The simulation and real-world experiments share the same robot, the same RGB-D camera, and the same shelf environment; the simulation experiments are conducted by the Pybullet simulator. We use the 7-dof Franka Emika Panda robot with

a parallel-jaw gripper and an Azure Kinect DK camera sensor looking into the shelf at a fixed location (See Figure 5.5). The raw input visual observation from the camera is an RGB-D, which is then converted to point cloud data. We use cylinder-shaped and cube-shaped objects of various sizes which are visualized in Figure 5.5.

**Target object.** Throughout the experiments, we restrict our attention to a cylindrical target object. Considering the rotational symmetry of the target object and the assumption that the target object stands upright, the target pose space $\mathcal{X}$ can be reduced to $\mathbb{R}^2$. We note that if the target object is a box, the whole $\mathrm{SE}(2)$ space should be considered as the target pose space; the experimental results when the target object is a box can be found in Chapter 5.6.3.

**Evaluation metrics.** We report the find and grasp success rates separately. If a sequence of our actions makes the target visible within 10 time steps, it is considered as a find-success. If it makes the target not only visible but also graspable and we can find a collision-free path for taking out the object within 10 time steps, it is considered a grasp success. Additionally, we measure the average number of actions required to find or grasp the target.

### 5.5.1 Simulation Experiments Results

In this section, we evaluate our methods in the simulation experiments and empirically show that they can find and grasp the fully-occluded target object. The details on simulation experiment settings are provided below.

**Object configuration.** To evaluate our method, we have created 180 scenarios for each number of surrounding objects in $\{2, 4, 6, 8\}$, so a total of 720 scenarios; for each scenario, a random selection (allowing duplicates) is made among the given objects in Figure 5.5.

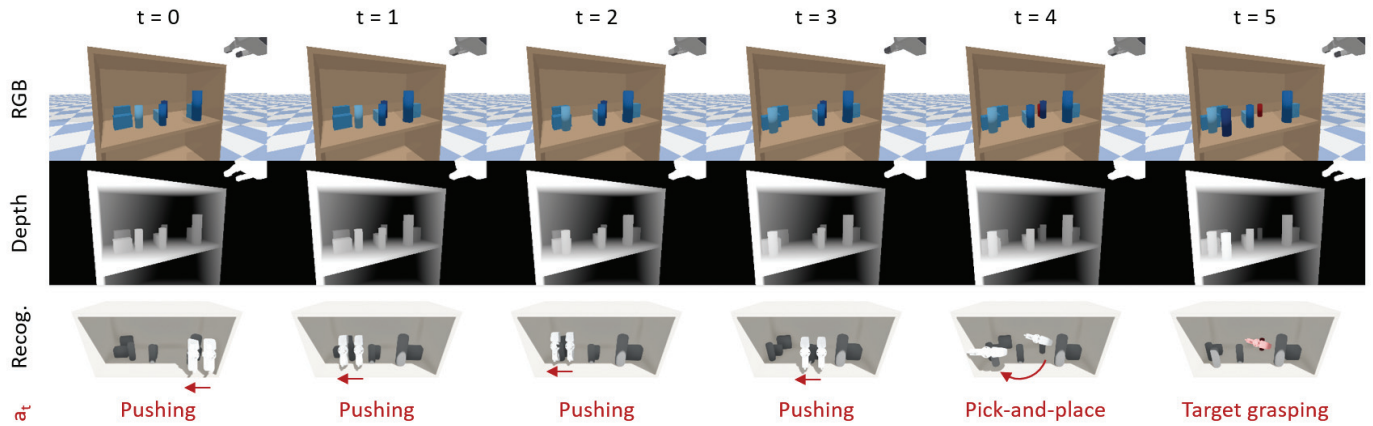


Figure 5.6: An example trajectory of simulation manipulation. Each column shows the camera input and action selection at each time step. In the simulation, surrounding objects are blue and the target object is red.

Table 5.1: Simulation manipulation results

| | | The number of objects | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 2 | | 4 | | 6 | | 8 | |
| **METHOD** | | Find | Grasp | Find | Grasp | Find | Grasp | Find | Grasp |
| O-Search-and-Grasp | Succ. | 0.978 | **0.939** | **0.939** | 0.761 | 0.878 | 0.622 | 0.844 | 0.528 |
| | Steps | 1.392 | 1.562 | 2.178 | 2.635 | 2.589 | 3.473 | 2.664 | 3.947 |
| O-Search-for-Grasp | Succ. | **0.983** | 0.928 | 0.933 | **0.794** | **0.900** | **0.678** | **0.889** | **0.578** |
| | Steps | 1.407 | 1.647 | 2.077 | 2.769 | 2.377 | 3.574 | 2.831 | 4.125 |
| R-Search-and-Grasp | Succ. | **0.983** | **0.928** | 0.928 | **0.789** | 0.889 | 0.656 | 0.878 | 0.606 |
| | Steps | 1.362 | 1.611 | 2.222 | 2.739 | 2.581 | 3.432 | 2.981 | 3.78 |
| R-Search-for-Grasp | Succ. | 0.972 | 0.922 | **0.933** | 0.756 | **0.922** | **0.672** | **0.894** | **0.656** |
| | Steps | 1.331 | 1.651 | 2.196 | 2.765 | 2.301 | 3.281 | 2.901 | 3.975 |

**Initial scene setting.** We first randomly drop the selected objects on the shelf. Then, we place the (red) target object on the shelf where it is not visible from the current camera image. If there is no place to place the target object, or the target object is not graspable because of the collision of the gripper with the shelf, the scenario is discarded.

**Target detection.** In the simulation environment, a ground-truth segmentation mask can be used from the synthetic camera. If a part of the target object is observed for more than 100 pixels on the camera image (c.f. the resolution of the camera is $1280 \times 720$), it is considered to have succeeded in finding the target object in that scenario.

We note that there are some scenarios where the target object is not graspable even with the maximum number of actions, so the maximum average success rate is slightly lower than 1.

To evaluate our 3D reconstruction-based mechanical search method, and in particular to see how much recognition error affects the task performance, we also test the cases where the ground-truth poses and shape parameters of the surrounding objects (not the

target object) are available, and denote them as *oracle*. In these cases, the the recognition module is not used and the ground-truth information of the surrounding objects are directly used for solving optimal control. We put the letter 'O' in front of the method name in the oracle experiments, and 'R' in the experiments using the 3D recognition (e.g., O-Search-and-Grasp and R-Search-for-Grasp).

Figure 5.6 shows an example of how our recognition-based search-for-grasp method acts on the simulation experiment. The search-for-grasp method succeeds in finding the target object in four pushing actions and then makes the target object graspable by performing an additional pick-and-place action. The success rates and the average number of actions for finding and grasping the target are shown in Table 5.1. First, the performance differences between *oracle* and *recognition* are not significant, which means that the 3D recognition error does not significantly affect performance. Second, search-for-grasp has no difference in performance from search-and-grasp when the number of surrounding objects is small, but shows better performance when the number is large. This suggests that considering graspability is of great help when finding the target.

### 5.5.2 Real-world Experiments Results

We adopt the R-Search-for-Grasp method for finding and grasping the desired target object in real-world environment. The details on real-world experiment settings are provided below.

**Object configuration.** We have created 5 scenarios for each number of surrounding objects in $\{3, 4, 5, 6\}$, so a total of 20 scenarios; for each scenario, a pre-defined object set is used according to the number of surrounding objects as shown in Figure 5.8.

**Initial scene setting.** We put the given surrounding objects and the target object so that the target object (red cylinder) is not visible in the initial camera view.

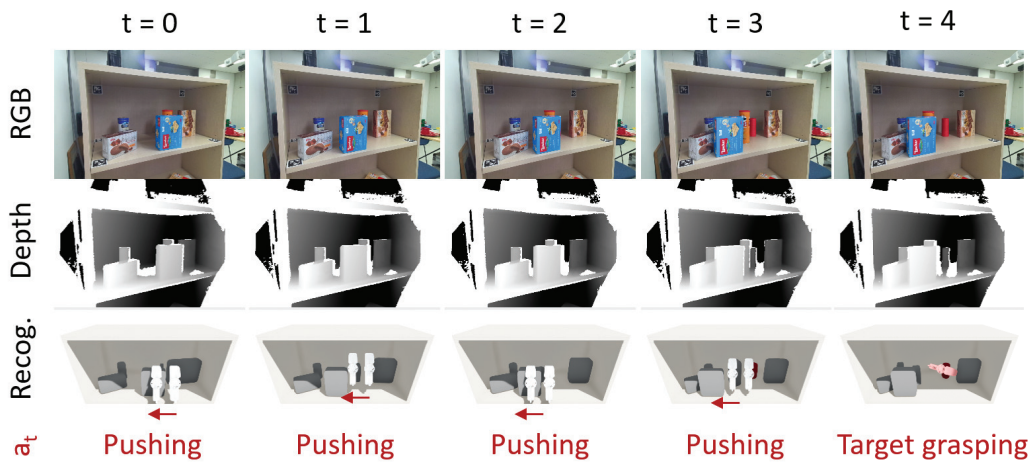


Figure 5.7: Search-for-grasp real-world manipulation results


Figure 5.8: Pre-defined object set used for real-world experiments.

Table 5.2: Search-for-grasp real-world manipulation results.

| Num. | Find | Grasp |
|---|---|---|
| 3 | 5/5 | 5/5 |
| 4 | 5/5 | 4/5 |
| 5 | 4/5 | 2/5 |
| 6 | 3/5 | 2/5 |

**Target detection.** In the 3D recognition process, we segment the observed point cloud. We first calculate the average RGB value of the points in each segmented point cloud. Then, we calculate the MSE between these average RGB values and the RGB of the target object (in this case, $[0.7282, 0.1558, 0.2099]$), and if there is exactly one segmented point cloud with MSE smaller than 0.1, the target object is said to be found.

Figure 5.7 shows a real-world manipulation result. The target object is occluded by the two objects, and the target object is found through three pushing actions. The found target object is not graspable at $t = 3$, an additional pushing action is applied to make the target object graspable.

**Failure cases.** Table 5.2 shows the manipulation success rates in real-world experiments. A few failure failure cases occur, especially when the number of the surrounding objects increases. Most of the reasons for failure cases are (i) that there's no solution of rearranging the objects in our designed action space (see Chapter 5.4.4) and (ii) that incorrect 3D recognition causes erroneous updates of existence and graspability maps; for example, if the recognition model recognizes an object as inaccurately large, the existence map may be underestimated, i.e., it is decided that $f(x) = 0$, but the target object can exist at $x \in \mathrm{SE}(2)$.

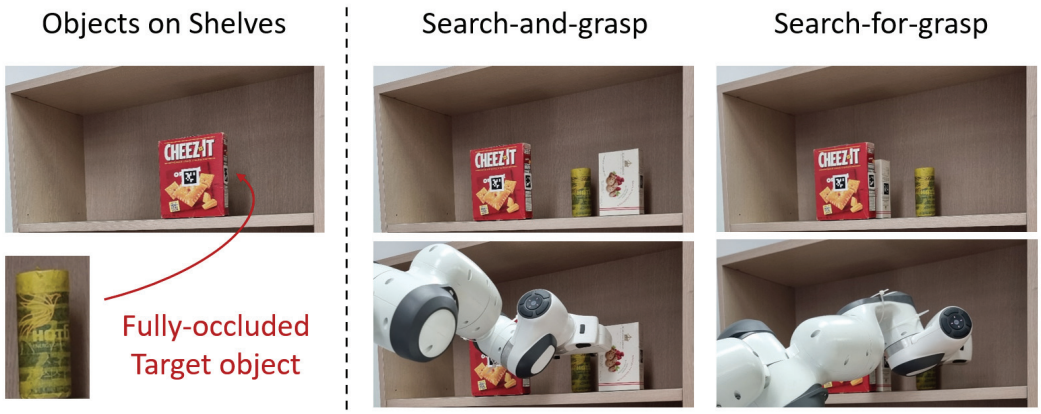


Figure 5.9: Comparison of *search-and-grasp* and *search-for-grasp* methods to find the target object (yellow cylinder). *This figure is a conceptual figure, not the result of implementing the methods.*

## 5.6 Additional Experimental Results

### 5.6.1 3-Object Toy Experiment

In this experiment, we compare the performance of the two proposed methods, search-and-grasp and search-for-grasp, and especially highlight the advantages of the search-for-grasp method. Let consider a situation where a yellow cylindrical target object is fully occluded by two larger boxes as described in the left of Figure 5.9. The search-and-grasp method may successfully find the target object by rearranging the two boxes, but it does not guarantee the target object's graspability. As illustrated in the middle of Figure 5.9, the identified object may not have a collision-free robot trajectory for grasping in some cases. Consequently, additional actions would be necessary to manipulate the environment and make the target object graspable. On the other hand, in the case of the search-for-grasp method, the target object's graspability is taken into account in

the searching phase, so the two boxes are rearranged in a way that the target object becomes not only visible but also graspable as shown in the right of Figure 5.9. In summary, when the target object is occluded by multiple objects, the search-for-grasp method can be efficient in terms of the number of actions. To verify this, we design an additional simulation experiment named *3-object toy experiment* as described below.

**Object configuration.** We have created 200 scenarios in the pybullet simulation environment with two large cylindrical objects and one small cylindrical target object. The radius and height of the target cylinder is fixed, and those of two large cylinders are randomly selected big enough to occlude the target.

**Initial scene setting.** The position of the camera center and the positions of three cylindrical objects are on a straight line in the x-y plane so that the target object is occluded by the other two objects on the shelf. The position of the target object is fixed, while the $(x, y)$ coordinates of the other objects' positions are randomly selected on the straight line.

**Target detection.** Target detection method is the same with Chapter 5.5.1.

Figure 5.10 shows the mechanical search results by search-and-grasp and search-for-grasp. In this subsection, we only use recognition-based (i.e., R-search-and-grasp and R-search-for-grasp). The search-and-grasp method succeeds in finding the target object in two pick-and-place actions as desired, but when target object is found, it cannot be grasped due to collision with surrounding objects. So, this method makes the target object graspable by performing two additional actions as shown in the left of Figure 5.10. In the case of the search-for-grasp method, it also succeeds finding the target object in two pick-and-place actions, and at this time, the target object becomes graspable at the same time as shown in the right of Figure 5.10. This difference of the number of actions is due to the difference in whether the graspability of the found object is taken into account when searching for the object. Although the search-for-grasp method is slightly

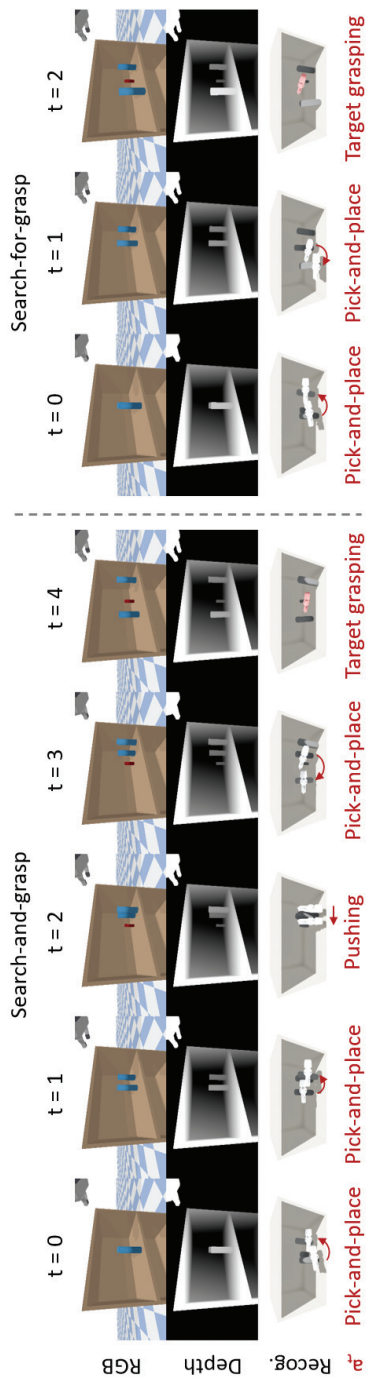


Figure 5.10: Example trajectories of simulation manipulation for R-search-and-grasp (*Left*) and R-search-for-grasp (*Right*). Each column shows the camera input and action selection at each time step. In the simulation, surrounding objects are blue and the target object is red.

Table 5.3: Manipulation results for 3-object toy experiment.

| **METHOD** | Succ. | Find | Grasp |
|---|---|---|---|
| R-Search-and-Grasp | 0.985 | **2.065** | 2.919 |
| R-Search-for-Grasp | **0.995** | 2.255 | **2.497** |

more computationally expensive than the search-and-grasp method, the search-for-grasp method can be efficient in terms of number of actions in these situations.

In order to quantitatively verify this fact, we measure (i) the average grasp-success rate and the average number of actions to (ii) find and (iii) grasp the target object for the two methods, and the results are shown in Table 5.3. We note that the minimum number of actions required to find and grasp is 2. The average number of actions to *find* is lower for search-and-grasp than for search-for-grasp since the search-and-grasp method only focuses on finding the target object. On the other hand, the average number of actions to *grasp* is lower for search-for-grasp than for search-and-grasp. Unlike the search-and-grasp methods, the search-for-grasp simultaneously search for a target object while increasing the probability of grasping the found target object. As a result, this fact has the effect of reducing the average number of actions. In conclusion, we verify that the search-for-grasp method can be more efficient than the search-and-grasp method in terms of the number of actions when the target object is occluded by multiple objects.

### 5.6.2 Mechanical Search via Only Pushing or Only Pick-and-place

This experiment evaluates the performance of two cases where only one action type (e.g., pick-and-place or pushing) is allowed, demonstrating that both motions are essential for mechanical search tasks on shelves. Instead of action sampler described in Appendix 5.4.4, if only push is allowed, the action is sampled only from $I_p$. If only

pick-and-place is allowed, the action is sampled only from $I_g$. We use recognition-based method in this experiment.

**Object configuration.** We have created 50 scenarios for each number of surrounding objects in $\{4, 8\}$, so a total of 100 scenarios; for each scenario, a random selection (allowing duplicates) is made among the given objects.

**Initial scene setting** Initial scene setting is the same with Chapter 5.5.1.

**Target detection.** Target detection method is the same with Chapter 5.5.1.

Figure 5.11 shows the mechanical search results by our methods using only pick-and-place and only pushing. In the first example, using only pushing succeeds in finding and grasping the target object in one action as desired, but using only pick-and-place fails since there is no valid pick-and-place action. In the second example, using only pick-and-place succeeds in finding and grasping in one action, and using only pushing succeeds in finding the target object but there is no pushing action to move the blue cylinder to be removed since it becomes close to the wall of shelf. As such, there are cases where using only pick-and-place and only pushing show different trajectories.

To quantitatively investigate the roles of the pushing and pick-and-place, and the results are shown in Table 5.4. First, allowing only one type of action degrades the performance significantly as shown in the table. Specifically, using only pick-and-place highly degrades the performance of finding the target object compared to the case where both actions are used, and on the other hand, using only pushing highly degrades the performance of grasping the target object, even though the success rate in finding is higher than using only pick-and-place.

This result is because pick-and-place and pushing have different strengths and weaknesses. The pick-and-place is efficient in terms of the average number of actions because it can move the objects farther away, but the actions which can be performed are limited due to the fewer objects the robot can grasp in a cluttered scene. The pushing is

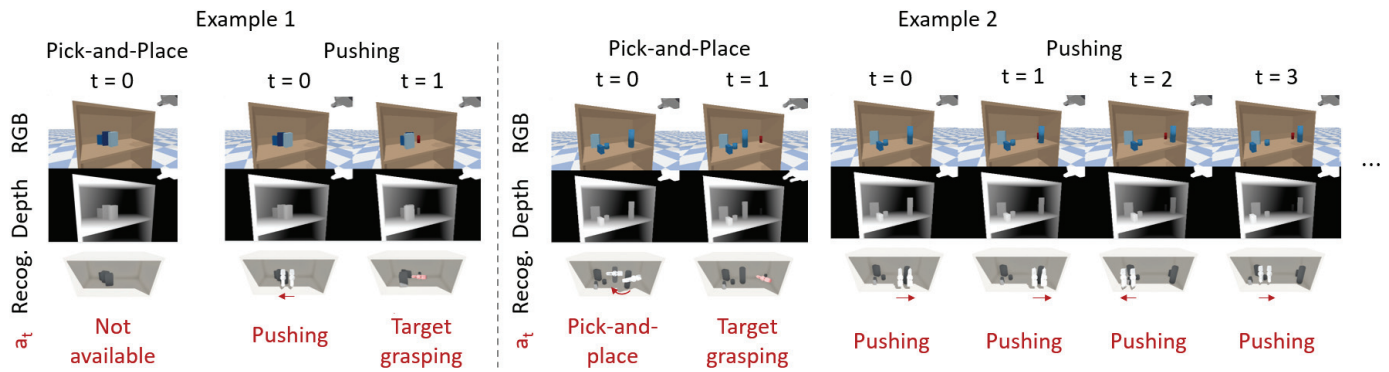


Figure 5.11: Example trajectories of simulation using only pick-and-place and only pushing. Each column shows the camera input and action selection at each time step. In the simulation, surrounding objects are blue and the target object is red. (*Left*) A scenario where only pick-and-place fails but only pushing succeeds. (*Right*) A scenario where only pick-and-place succeeds but only pushing fails.

Table 5.4: Simulation manipulation results for the one-type action experiments.

| | | | The number of objects | | | |
|---|---|---|---|---|---|---|
| | | | 4 | | 8 | |
| **METHOD** | **ALLOWED ACTION** | | Find | Grasp | Find | Grasp |
| R-Search-and-Grasp | Both | Succ. | 0.96 | 0.84 | 0.98 | 0.56 |
| | | Steps | 1.562 | 2.065 | 2.102 | 3.73 |
| | Pick-and-Place | Succ. | 0.76 | 0.64 | 0.6 | 0.5 |
| | | Steps | 1.5 | 1.656 | 2.167 | 2.96 |
| | Push | Succ. | 0.92 | 0.58 | 0.82 | 0.58 |
| | | Steps | 2.196 | 3.793 | 3.073 | 4.828 |
| R-Search-for-Grasp | Both | Succ. | 1.0 | 0.88 | 0.98 | 0.6 |
| | | Steps | 1.74 | 2.543 | 1.653 | 3.846 |
| | Pick-and-Place | Succ. | 0.76 | 0.66 | 0.64 | 0.46 |
| | | Steps | 1.395 | 1.667 | 1.938 | 3.043 |
| | Push | Succ. | 0.9 | 0.58 | 0.78 | 0.6 |
| | | Steps | 2.444 | 3.483 | 3.667 | 4.733 |

better than pick-and-place for finding target objects because the robot can manipulate more objects than pick-and-place, but because the robot can't move the objects that far, it requires a higher number of actions because the can't move the objects that far. In summary, both pick-and-place and pushing are required to find the target and grasp the target object.

### 5.6.3 Mechanical Search with Box Target Object

This experiment evaluates the performance of our methods for mechanical search tasks with a box target object. In this case, the target pose space should be $\mathcal{X} = \mathrm{SE}(2)$, i.e. position $(x, y)$ of the target box on the shelf and its z-axis angle $\theta$ as orientation. We

only use recognition-based method in this experiment.

**Object configuration.** We have created 40 scenarios for each number of surrounding objects in $\{4, 8\}$, so a total of 100 scenarios; for each scenario, a random selection (allowing duplicates) is made among the given objects. The target object is replaced with a box shape instead of a cylinder.

**Initial scene setting** Initial scene setting is the same with Chapter 5.5.1.

**Target detection.** Target detection method is the same with Chapter 5.5.1.

Figure 5.12 shows the mechanical search results by search-for-grasp on the task of the box target object. The search-and-grasp method succeeds in finding the target object in three pick-and-place actions and succeeds in grasping the target object in additional one pick-and-place action. Table 5.5 shows the performance of our methods with the box target object. Compared to the cylinder target object case, the grasp success rate is lower when the number of objects is $4$. This is because the approach direction of the grasping trajectory is limited to one in the case of the box as shown in the Figure C.5; we note that the cylinder can be grasped in any direction due to rotational symmetry. However, when the number of objects are large (i.e., the number of objects is 8), the box target object case shows similar performance to the cylinder target object case. This is because the advantage of having various grasping approach directions of cylinders is lost in highly cluttered environment. In summary, we conclude that our algorithm also works on the target pose space of $\mathrm{SE}(2)$ on the box target object experiment.

### 5.6.4 Ablation Study on Hyperparameter $\alpha$

This experiment evaluates the performance of our methods for mechanical search tasks concerning the hyperparameter $\alpha$. To comprehend the role of $\alpha$, we revisit the objective

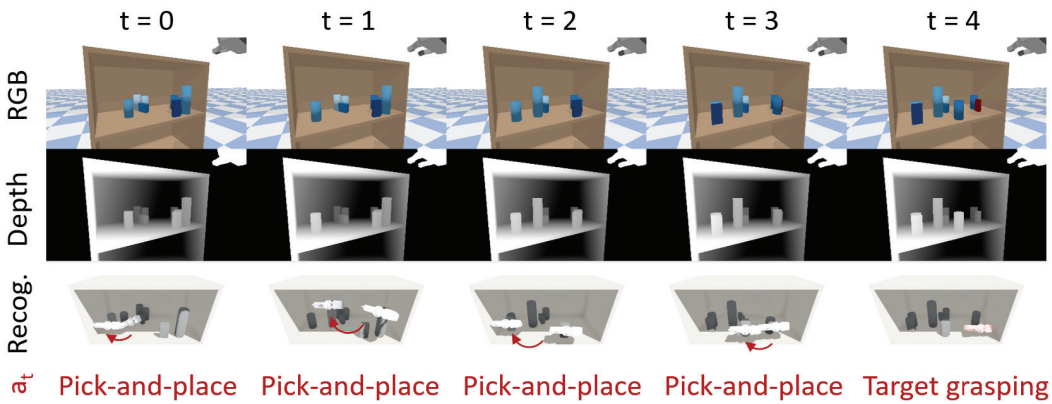


Figure 5.12: An example trajectory of simulation manipulation for R-search-for-grasp for the box-shaped target object. Each column shows the camera input and action selection at each time step. In the simulation, surrounding objects are blue and the target object is red.

Table 5.5: Simulation manipulation results for the box target object

| | | The number of objects | | | |
|---|---|---|---|---|---|
| | | 4 | | 8 | |
| **METHOD** | | Find | Grasp | Find | Grasp |
| R-Search-and-Grasp | Succ. | 0.925 | 0.775 | 0.825 | 0.525 |
| | Steps | 1.568 | 3.323 | 2.394 | 4.81 |
| R-Search-for-Grasp | Succ. | 0.95 | 0.7 | 0.85 | 0.6 |
| | Steps | 1.652 | 3.286 | 2.412 | 5.167 |

function of our optimal control:

$$\min_{a_1,\ldots,a_T} \sum_{x\in\mathcal{X}} f_T(x) + \alpha f_T(x)(1 - g_T(x)). \tag{5.6.11}$$

Figure 5.13 depicts a graph illustrating the find and grasp success rates of our R-Search-for-Grasp algorithm concerning different values of $\alpha$. Initially, discerning major trends proves challenging due to the interdependence between finding and grasping. Notably, the grasp success rate inherently relies on the find success rate, as successful grasps are a subset of successful finds. Moreover, candidate target poses where the existence function equals 1 (i.e., $f(x) = 1$) often imply occlusion by other objects, resulting in the graspability function of 0 (i.e., $g(x) = 0$) in most cases. Consequently, minimizing the second term in the objective function can somewhat aid in finding the target object and thereby may reduce the first term. As such, identifying significant trends becomes challenging. Nonetheless, the critical insight from this graph reveals that (i) excessively small $\alpha$ values lead to decreased success rates in grasping the target object, particularly evident when many objects are present on the shelf (i.e., 6 or 8), and (ii) higher $\alpha$ values indicate a certain level of algorithmic effectiveness.

## 5.7 Conclusion

We have proposed a general mechanical search and grasping framework on cluttered shelves and implemented practical algorithms, the search-and-grasp and search-for-grasp, by using the pre-trained 3D object recognition model. We use pushing and pick-and-place actions to rearrange the obstructing objects and to make the target object visible and graspable. Our method does not rely on specialized tools and is not restricted to the suction gripper as the previous studies, but rather uses the standard two-finger gripper and 6-DoF grasping for pick-and-place actions. We have confirmed that our proposed

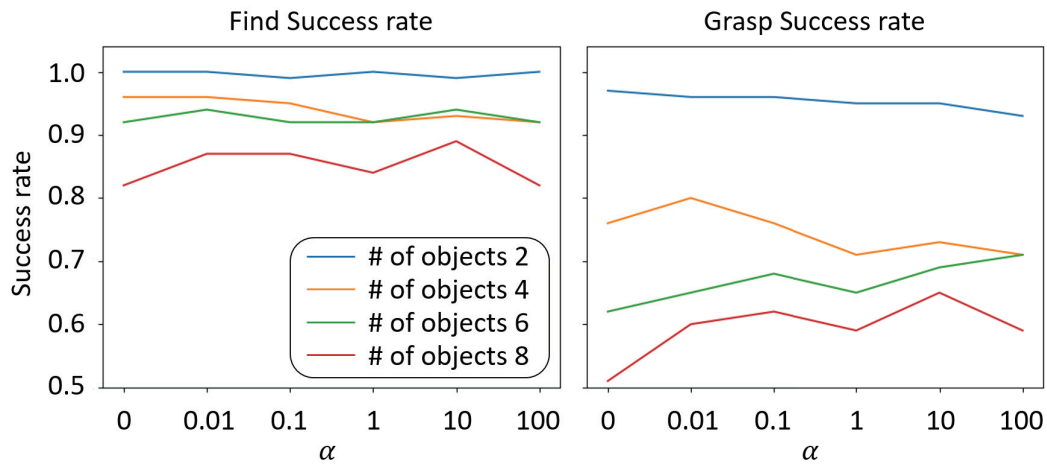


Figure 5.13: Find and grasp success rates of R-Search-for-Grasp according to $\alpha$.

method works well in both simulation and real-world shelf environments.

**Limitations and Future Directions** First, since our method considers single superquadric shaped objects, it is not trivial to apply it directly to more complex objects. One future direction to extend our method is to exploit the researches that attempt to represent the complex objects as multiple superquadrics [102, 64, 103, 105]. Even if each object is represented with several superquadrics, we can compute the existence function and the graspability function in the same manner. Second, our method highly depends on the 3D recognition model, inaccurate recognition can cause several problems. There can be robot-object collisions because we check the collision between the robot action and recognized objects. Moreover, some inappropriate action decisions can be made due to erroneous existence map estimation. To overcome the limitation, additional information such as RGB images should be utilized to enhance the recognition performance [128, 157]. Lastly, since the graspability of the object is checked using only a limited number of grasping trajectories (see Appendix C.3), there can be cases

where it is determined that the object cannot be grasped even if there is a grasping trajectory. As a research direction to overcome this, the graspability can be checked with more diverse trajectories quickly using a network-based planner [158].

# 6 Conclusion

## 6.1 Summary

This thesis has demonstrated that numerous challenges encountered in object manipulation can be effectively addressed using shape recognition methods. The crucial insight is that 3D reasoning of a scene from visual observations, particularly through 3D shape recognition using (deformable) superquadrics, significantly simplifies and enhances the handling of various challenging vision-based object manipulation problems. Building on this perspective, we have developed shape recognition-based object manipulation algorithms for grasping unknown objects, learning pushing dynamics models for tabletop object manipulation, and conducting mechanical searches in cluttered shelf environments. The main contributions of each chapter are summarized as follows.

- **DSQNet: Deformable Superquadric Network for Unknown Object Grasping**

  We have introduced a novel shape recognition-based grasping method that integrates a wider range of shape templates, specifically deformable superquadrics,

into a deep learning framework. This method is encapsulated in the *Deformable SuperQuadric Network (DSQNet)*, proficient at recognizing complete object shape. In particular, we employ a supervised learning framework enabling DSQNet to generate these eight parameters and pose of the deformable superquadric, aligning it with the complete shape of the object, including occluded parts. We have discovered that these deformable superquadrics can represent a broad array of shapes using only eight continuous parameters. Additionally, we capitalize on the benefit of closed-form surface equations for the efficient computation of point-to-surface distances, which is essential for precise fitting. We also propose an algorithm that effectively and efficiently generates grasp poses using these deformable superquadric shapes. Our findings demonstrate that DSQNet surpasses existing shape recognition-based methods in terms of both speed and accuracy in object shape recognition. Furthermore, our method outperforms others in grasping success rates, owing to its precise shape recognition capabilities. Remarkably, our recognition-based approach achieves grasping success rates comparable to those of widely-used end-to-end methods, while it requires minimal training data and offers adaptability to various types of grippers.

- **SQPDNet: Superquadric Pushing Dynamics Model for Pushing Manipulation**

  We have extended the advantages of shape recognition to non-prehensile manipulation, with a particular emphasis on learning pushing dynamics models. A significant benefit of incorporating shape recognition in this domain is the natural ability to define an $\mathrm{SE}(2)$-equivariant pushing dynamics model. This insight has led us to develop a unique neural network architecture, the *SuperQuadric Pushing Dynamics Network (SQPD-Net)*, which inherently embodies the equivariance property. A central idea of our model is to utilize explicit relationships between

the poses of objects and the pushing action. This methodology acknowledges the symmetry present in physical systems, greatly enhancing the model's ability to generalize. Our findings indicate that our shape recognition-based model outperforms existing vision-based pushing dynamics models, especially with the integration of $\mathrm{SE}(2)$-equivariance. The effectiveness of our model is further validated in its application to model-based optimal controls for various pushing manipulation tasks. This is corroborated by results from both simulations and real-world experiments, highlighting the practicality and efficiency of our approach.

- **Search-for-Grasp: Superquadric Recognition for Mechanical Search**

  Lastly, we address the practical yet challenging task of mechanical search on cluttered shelves using a shape recognition model. This task entails finding and grasping a specific target object, which is occluded by other objects and initially invisible by vision sensors, on a cluttered shelf. The geometric configuration of the shelf, permitting only frontal visual observations and restricting the manipulator's workspace, further compounds the task's complexity. To overcome this challenge, we have utilized the shape recognition models, particularly the superquadric recognition model. This model facilitates quick and efficient reasoning about the potential poses of the occluded target object by enabling rapid computations for various tasks such as depth image rendering and collision checking. Additionally, we have incorporated the shape recognition-based prehensile and non-prehensile manipulation techniques developed in earlier chapters. This integration allows the robot to effectively and safely find the target object. Our method's efficacy has been confirmed through both simulation and real-world experiments, proving its capability to successfully identify and grasp target objects using a standard two-finger robot gripper. Importantly, our approach demonstrates robustness even in

the presence of the typical noise in vision sensor data in real-world scenarios.

## 6.2 Future Work

There are several potential avenues for extending our current shape recognition-based prehensile and non-prehensile manipulation methods. While some of these extensions have been directly referenced in the preceding chapters, here we highlight key directions and open problems for future research that we believe warrant further investigation.

- **More accurate and effective recognition model for object manipulation**

  In this thesis, we propose object manipulation methods using superquadric-based (Chapters 4 and 5) and deformable superquadric-based (Chapter 3) shape recognition models. A natural question that arises is whether deformable superquadrics can adequately represent the diversity and complexity of many objects. Several studies have been conducted to recognize full 3D shapes from partial observations, such as depth images, using explicit object representations like occupancy grids [51], point clouds [52], or meshes [53]. Furthermore, recent works have explored learning implicit 3D representations of objects using neural implicit functions [55, 56, 57, 58]. While these representations could potentially replace deformable superquadrics (an interesting extension of our method), deformable superquadrics are still highly effective for object manipulation, not just recognition. For instance, they enable rapid generation of grasp poses, are efficient in computations related to various manipulations such as collision checking, and most importantly, provide interpretable information by abstracting objects into their basic parts. This capability makes them highly advantageous for the range of calculations required in various object manipulation.

One method to enhance the expressiveness of object shapes, while adhering to our core philosophy, is the incorporation of a more diverse set of shape primitives, particularly superquadrics. In our work, we primarily utilize superellipsoids (although superparaboloids are briefly discussed in Chapter 3.7), but there are other viable options, such as superparaboloids and supertoroids, as outlined in Chapter 2. Specifically, superparaboloids are capable of expressing a variety of highly concave shapes, including dishes, bowls, and the heads of wine glasses. Supertoroids, on the other hand, can represent a wide range of shapes with cavities, ranging from tori to cylindrical shells. By employing an appropriate deformation model, it becomes possible to represent an even broader array of objects. The exploration of more diverse superquadric primitives as an extension of our work is an intriguing prospect for future research.

In addition to utilizing more diverse and expressive shape primitives, the performance of our recognition model could be significantly improved for a broader range of complex objects by incorporating publicly available, large-scale 3D datasets such as ShapeNet [92]. However, as outlined in Chapter 3, our current method relies on primitive-shaped part segmentation labels, which are challenging to obtain. Consequently, direct utilization of raw public datasets presents certain difficulties. Future work could explore developing a model that does not require part-segmentation supervision and is capable of fitting multiple shape primitives to given ground-truth object shapes. Recent research has made progress in representing objects as sets of superquadrics when their full shapes are available [102, 103, 104, 105]. In our scenario of vision-based object manipulation, which typically involves only partial observation of unknown objects, directly applying the aforementioned studies is not straightforward. The prospect of recognizing objects

as sets of deformable superquadrics, without relying on part segmentation labels and from only partial observations, presents a promising research direction.

- **Object manipulation methods considering physical object properties**

  While we currently propose vision-based object manipulation methods that do not take into account the physical properties of objects, in practice, considering these properties is crucial for successful manipulation. For instance, when generating grasp poses using only the recognized shape of an object (e.g., assuming a uniform mass distribution), there's a risk that the grasping point may be far from the object's center of mass. This is particularly evident in objects like hammers, mugs, or dumbbells, where ignoring the mass distribution can lead to unstable grasping. Similarly, in pushing manipulation, predicting dynamics becomes challenging when dealing with objects that have non-uniform mass distributions, as varying distributions result in different motions. Moreover, during pushing manipulation that involves interaction between objects, the absolute values of mass and friction coefficients significantly impact the motion of the objects. Understanding the physical properties of objects would allow us to (i) identify grasp poses that can support the object's weight, taking into account factors like density or the center of mass, and (ii) enhance the accuracy of dynamics models in learning object motions. However, estimating these physical properties using only a vision sensor remains a significantly challenging task.

  To address this challenge, several studies have proposed methods to ascertain the physical properties of unknown objects through multi-step dynamic interactions. One approach involves learning abstract representations (or features) of physical

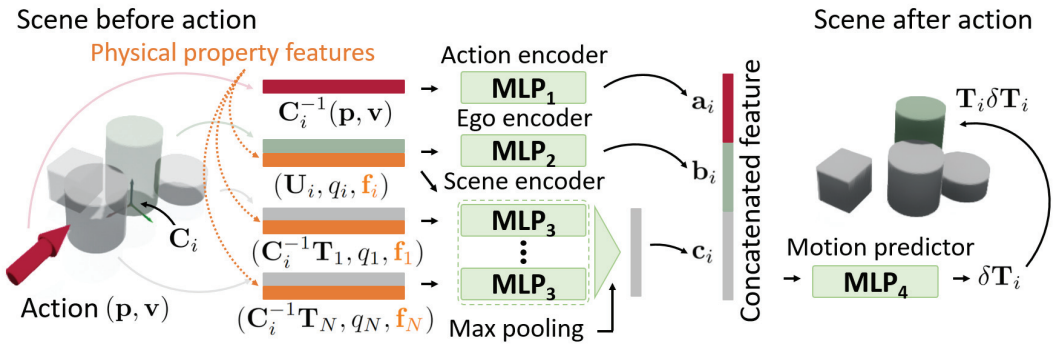


Figure 6.1: SE(2)-equivariant pushing dynamics neural network architecture considering physical object properties for an $i$-th object.

object properties [126]. Utilizing data from vision sensors, this study employ a recurrent structure to accumulate information from multiple robot-object or object-object interactions. This enables the model to more effectively infer and encode the objects' physical properties over time. By operating without an explicit physics model, the system is not only capable of exploring various types of interactions but also able to infer and separate physical properties from these interactions. Building on this concept, an interesting research direction is to develop a modified pushing dynamics model (SQPDNet) that incorporates physical properties, as illustrated in Figure 6.1. This new model processes input information about the poses and shapes of objects, along with physical property features $\mathbf{f}_i$'s of each object. By suitably updating these features through multi-step interactions and appropriate recurrent structures, the model could be capable of considering diverse object mass distributions and accordingly predicting accurate motions of the objects. While these methods show effectiveness with only vision sensor data, a notable challenge lies in generalizing the physical property features learned from one task (e.g., pushing manipulation) to other tasks (such as grasping).

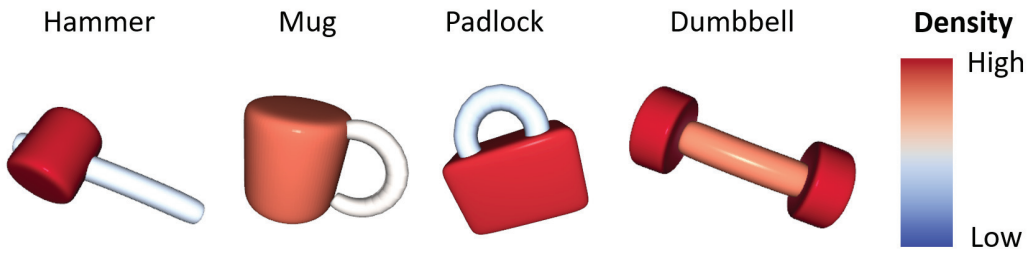


Figure 6.2: Recognized shapes via deformable superquadrics and their mass distributions.

Another approach involves the use of differentiable simulators to determine explicit physical properties of objects, including mass distribution and friction coefficient, as explored by [159]. Through multi-step dynamic interactions, they initially calculate the analytical gradient of the physical simulation error in relation to the mass and friction distributions, using a cuboid representation. The analytical gradients are then used to identify physical properties by analyzing videos of their sliding motions. Subsequently, the identified physical properties are applied to execute dynamic non-prehensile manipulations, such as sliding. Although this method allows the explicit physical properties to be directly utilized for various manipulation tasks, it currently operates under the assumption that the objects' shapes are known. To estimate the physical properties of completely unknown objects, shape recognition must be conducted in conjunction with the estimation process. In this context, our deformable superquadric-based DSQNet presents a promising opportunity. An interesting avenue for future research would be to develop a method that employs a set of deformable superquadrics to recognize object shapes and simultaneously estimate the mass density and friction coefficient of each superquadric through multi-step dynamic interactions, as shown in Figure 6.2.

- **Using multi-modal sensor beyond using only vision sensor**

  The vision sensor is a critical component for robots handling unknown objects. However, similar to how humans rely on multiple senses for complex, dexterous manipulation, robots may also require multi-modal sensors for more sophisticated and intricate tasks. Tactile sensors, for instance, can significantly aid in precise object grasping, especially when vision sensor data is noisy [78, 100]. They are particularly useful for accurately controlling the robot while handling of fragile items, such as eggs, or manipulating deformable objects, like dough and clothes [160, 161]. Moreover, most robotic arms are equipped with built-in force-torque sensors, or these can be mounted on their end-effectors. These sensors are beneficial for manipulating objects with diverse mass-inertia properties and are crucial for discovering the physical properties of objects through multi-step interactions, as previously mentioned [159]. Force-torque sensors may enable robots to better adapt to new objects in dynamic manipulation tasks, such as sliding [162, 163] or tossing [164]. Additionally, some studies have incorporated audio data to further enhance a robot's perception in various tasks, including bin packing and water pouring [165]. Exploring the use of these multi-modal sensors in conjunction with the shape recognition-based object manipulation methods proposed in this thesis presents another exciting avenue for future research.

## 6.3 Concluding Remark

Robotic object manipulation methods are increasingly vital, especially in logistics and household settings. However, relying solely on vision sensor information to manipulate various unknown objects across diverse tasks often proves impractical due to numerous constraints. We assert that our contribution holds significance for practitioners by

presenting robust methods capable of overcoming these practical challenges. Our suite of shape recognition-based object manipulation methods outlined in this thesis provides potential methods to address a broader spectrum of unknown objects and manipulation tasks. Moreover, these methods enable for further advancements, enabling robots to undertake more dynamic manipulation tasks than previously feasible.

# A

# Appendix: DSQNet

This section describes implementation details of our algorithm. We first describe the synthetic dataset used for training and evaluation of the recognition stage, followed by preprocessing of the point cloud, training the segmentation network and DSQNet, and hardware setup used for real-world grasping experiments.

## A.1 Synthetic Data Generation

The recognition stage consists of the segmentation network and DSQNet. For training and evaluation purposes, we generate synthetic datasets as shown in Figure A.1 (*object dataset* for the segmentation network and *primitive dataset* for DSQNet).

The synthetic objects are generated to match a variety of everyday objects. Inspired by YCB datasets [87], we define six primitive types: box (B), ellipsoid (E), cylinder (CY), cone (C), truncated cone (TC), and truncated torus (TT). From these six primitives we generate twelve object types: six single-primitive shapes, and six multi-primitive shapes (bottle, mug, dumbbell, hammer, padlock, and screwdriver). The primitives and

object types are shown in Figure A.1.

In detail, Figure A.2 shows the synthetic dataset configurations, consisting of (i) primitive types with the shape parameters required to define the shape, and (ii) object types with the assembly configuration of the shape primitives to construct each object. For each type in the primitive dataset, we randomly sample 100 different tuples of shape parameters within a predefined range of values; the range of parameters are shown in Table A.1. In the case of object types (i.e., multi-primitive shapes), we similarly randomly sample 100 shape parameters of all used primitives that satisfy prescribed shape continuity conditions. The ranges of the shape parameters and the conditions for generating the objects are described in Table A.2.

After determining the synthetic object types, we construct the object dataset and primitive dataset used for training and evaluation. The *object dataset* contains a total of 1200 object shapes: 100 objects with different parameters for each object type. For each object, a partially observed point cloud with 1000 points is obtained from 16 different camera viewpoints. The segmentation label for each point is annotated according to the primitive to which the point belongs. The *primitive dataset* includes a total of 800 primitive shapes: 100 primitives with different parameters for each primitive type with the exception of the truncated torus (TT). We use a total of 300 primitives for TT, resulting in a total of 800 training primitives for the six object types. When trained with a total of 600 primitive shapes (i.e., 100 shapes are also used for TT), DSQNet always predicts values for the bending parameter $b$ that are close to zero (the lower bound for $b$ is set to $0.01$). This finding can be explained with an imbalance in the training data, in which the objects representable by positive $b$ are relatively scarce compared to those representable by zero $b$. To address this training data imbalance, we use additional 200 primitives for TT. For each primitive, we obtain 16 paired data from different camera viewpoints: (i) a partially observed point cloud with 300 points, and (ii) the ground-truth

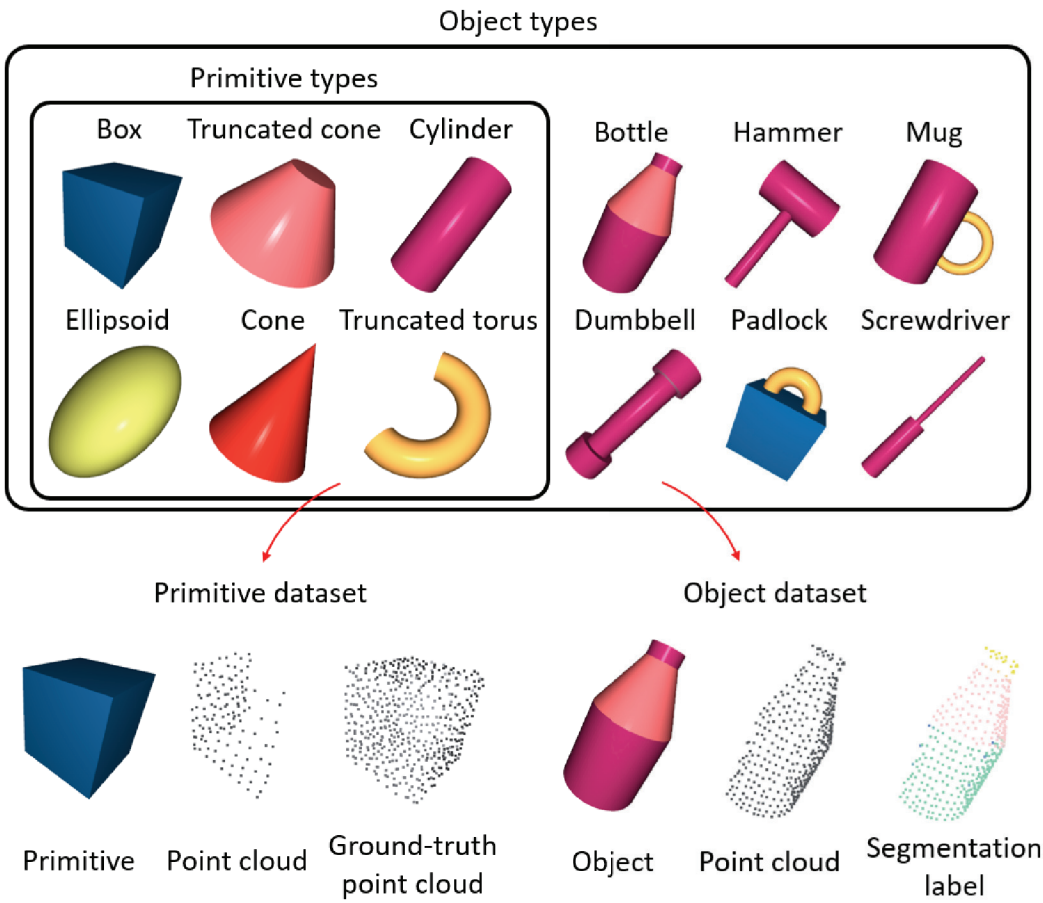


Figure A.1: Types of synthetic objects (primitive types and object types) and dataset generated from the synthetic objects (primitive dataset and object dataset).

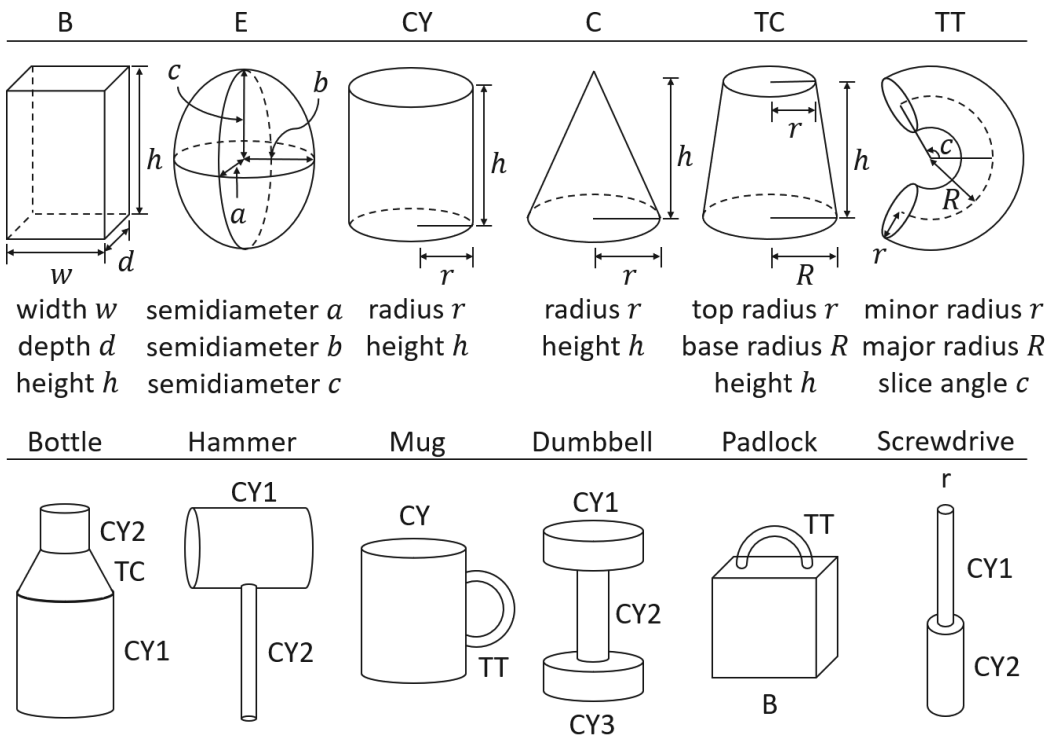


Figure A.2: Synthetic dataset configurations: primitive types with shape parameters (upper row), and object types with assembly configuration of the shape primitives (lower row).

Table A.1: Range of parameter values for primitive dataset

| Primitives | Parameters |
|---|---|
| B | $w \in [0.02, 0.3], d \in [0.02, 0.3], h \in [0.02, 0.3]$ |
| E | $a \in [0.03, 0.07], b \in [0.03, 0.07], c \in [0.03, 0.07]$ |
| CY | $r \in [0.01, 0.075], h \in [0.03, 0.3]$ |
| C | $r \in [0.02, 0.07], h \in [0.03, 0.15]$ |
| TC | $R \in [0.02, 0.1], r \in [0.02, 0.8R], h \in [0.02, 0.3]$ |
| TT | $r \in [0.01, 0.015], R \in [0.02, 0.075], c \in [\frac{\pi}{3}, \frac{2\pi}{3}]$ |

point cloud with 1500 points sampled uniformly from the ground-truth shape. Examples from the primitive dataset and object dataset are shown in Figure A.1.

We exclude data whose partially observed point cloud consists of a single plane, since the lack of three-dimensional information makes full shape estimation difficult. Consequently the primitive dataset and the object dataset respectively contain a total of 11,998 and 17,136 items, with each dataset divided into 90%/10% train and test sets.

To bridge the gap between the synthetic data and real-world vision sensor data, noise is added to each sample point $\mathbf{x}$ of the partially observed point cloud according to $\mathbf{x} \mapsto \mathbf{x} + m\mathbf{v}$, where $\mathbf{v}$ is uniformly sampled on the unit sphere and $m$ is sampled from a zero-mean Gaussian distribution with standard deviation 0.001.

## A.2 Training Segmentation Network and DSQNet

For the segmentation network, we use the same architecture and loss function used in [72] for DGCNN. Since the main purpose is to separate the point cloud, the network should learn permutation-invariant segmentation labels, so the loss function should be invariant to prediction permutations. To achieve this, we first find a bipartite matching

Table A.2: Range of parameter values for object dataset

| Objects | Prim. | Parameters |
|---|---|---|
| Bottle | CY1 | $r_1 \in [0.02, 0.07], h_1 \in [0.06, 0.15]$ |
| | TC | $R = r_1, r \in [0.01, 0.04]$, $h \in [0.02, 0.1]$ |
| | CY2 | $r_2 \in [r, 0.02], h_2 \in [0.01, 0.04]$ |
| Hammer | CY1 | $r_1 \in [0.025, 0.06], h_1 \in [0.09, 0.2]$ |
| | CY2 | $r_2 \in [0.012, 0.02], h_2 \in [0.05, 0.3]$ |
| Mug | CY | $r \in [0.03, 0.06], h \in [0.08, 0.15]$ |
| | TT | $r \in [0.003, 0.01], R \in [0.03, 0.05]$, $c \in [1.27, 2.09]$ |
| Dumbbell | CY1 | $r_1 \in [0.02, 0.04], h_1 \in [0.02, 0.05]$ |
| | CY2 | $r_2 \in [0.012, 0.018], h_2 \in [0.09, 0.11]$ |
| | CY3 | $r_3 = r_1, h_3 = h_1$ |
| Padlock | B | $w \in [0.015, 0.03], d \in [0.03, 0.045]$, $h \in [0.04, 0.06]$ |
| | TT | $r \in [0.003, 0.005], R \in [0.01, 0.02]$, $c \in [1.37, 1.77]$ |
| Screwdriver | CY1 | $r_1 \in [0.0025, 0.004], h_1 \in [0.08, 0.12]$ |
| | CY2 | $r_2 \in [0.01, 0.02], h_2 \in [0.06, 0.1]$ |

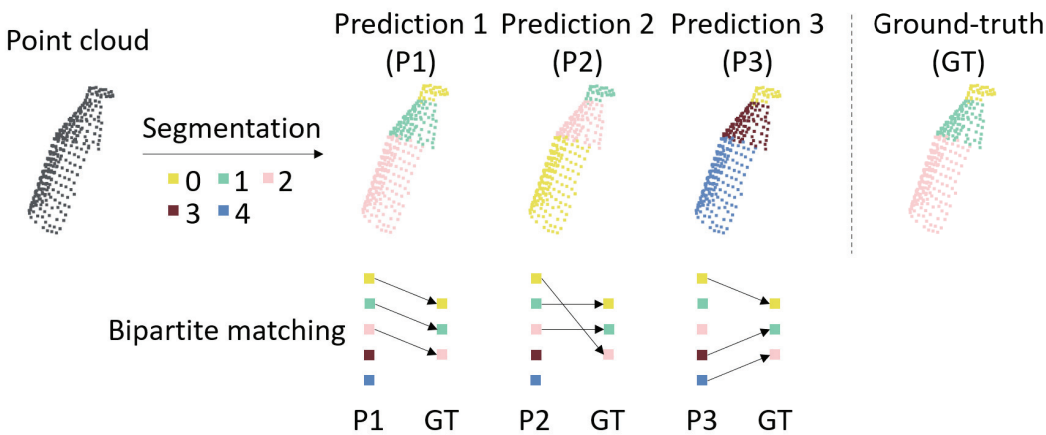


Figure A.3: Three predictions of segmentation labels, ground-truth labels, and the bipartite matching between prediction and ground-truth labels.

between ground-truth and prediction segmentation labels using the Hungarian matching algorithm [166], and then compute the usual segmentation loss between the matched labels. This makes the loss function permutation-invariant, and ensures that the network can properly separate the point cloud.

Figure A.3 provides a more conceptual explanation of the paragraph. Each color indicates the segmentation label; for example, the ground-truth labels consist of labels $\{0, 1, 2\}$. As shown in the figure, the partially observed point cloud is segmented in three prediction ways (P1, P2, P3), all of which are the desired results for the segmentation network of our framework. P1's segmentation labels exactly match the ground-truth labels. The labels for P2 and P3 do not exactly match the ground-truth labels, but they still are the right solution. To learn permutation-invariant segmentation labels in this way, a loss function that is permutation-invariant to the prediction must be used. For P2 and P3, there exists a bipartite matching between prediction and ground-truth

labels as shown in the lower part of Figure R1. As such, by first finding such a bipartite matching (this is called Hungarian matching), and calculating the segmentation loss between the matched labels, a permutation-invariant loss can be achieved.

We train the segmentation network and DSQNet with the object dataset and primitive dataset, respectively. We train the segmentation network with point clouds consisting of 1000 data points, and DSQNet with point clouds consisting of 300 data points. During inference, point clouds are preprocessed to match these numbers; a detailed description of these methods are provided in the next section. To optimize both networks, we use ADAM [167] with a learning rate of 0.001 and batch size of 16. DSQNet and the segmentation network are trained up to 3M and 1.3M iterations, which take approximately 46.5 hours and 41 hours on RTX3090, respectively.

# B

# Appendix: SQPDNet

## B.1 Details for SE(2)-Equivariant Dynamics Model

### B.1.1 Pose Decomposition

In the manuscript, we introduce an object pose decomposition method that decomposes an object pose $\mathbf{T} \in \mathrm{SE}(3)$ to a projected pose to the table denoted by $\mathbf{C} \in \mathrm{SE}(3)$ and the remaining rigid-body transformation $\mathbf{U} \in \mathrm{SE}(3)$ such that $\mathbf{T} = \mathbf{CU}$ as shown in the left of Figure B.1.

To achieve this, we first calculate the projection matrix $\mathbf{U}^{-1}$ as shown in the right of Figure B.1. The projection matrix is decomposed as $\mathbf{M}_1\mathbf{M}_2$, where $\mathbf{M}_1 \in \mathrm{SE}(3)$ is the rotation matrix that aligns the z-axis of the object pose with the z-axis of the base frame and $\mathbf{M}_2 \in \mathrm{SE}(3)$ is the translation matrix that projects the z-axis-aligned frame to the table surface. If $\mathbf{M}_1\mathbf{M}_2$ is calculated, we can obtain $\mathbf{U} = (\mathbf{M}_1\mathbf{M}_2)^{-1}$ and $\mathbf{C} = \mathbf{T}\mathbf{M}_1\mathbf{M}_2$ accordingly.

To calculate $\mathbf{M}_1$, we follow the following steps. The vector $\mathbf{z}$ and $\mathbf{z_0}$ is the z-axis

vector of the base frame $\mathbf{T}_0$ and object frame $\mathbf{T}$ expressed in the base frame $\mathbf{T}_0$ (e.g., $\mathbf{z_0} = (0, 0, 1)^T$). First, we calculate the inner product and cross product between $\mathbf{z}$ and $\mathbf{z_0}$ to obtain the following results:

$$\cos\phi = \mathbf{z} \cdot \mathbf{z}_0, \quad \sin\phi = ||\mathbf{z} \times \mathbf{z}_0||, \quad \mathbf{w} = \frac{\mathbf{z} \times \mathbf{z}_0}{||\mathbf{z} \times \mathbf{z}_0||},$$

where $\mathbf{w}$ is the rotation axis expressed in $\mathbf{T}_0$ and $\phi$ is the rotation angle; $\phi$ can be obtained by $\phi = \text{atan2}(\sin\phi, \cos\phi)$. Then the matrix $\mathbf{M}_1$ is of the form

$$\mathbf{M}_1 = \begin{bmatrix} \exp([\mathbf{R}^{-1}\mathbf{w}]\phi) & 0 \\ 0 & 1 \end{bmatrix}, \tag{B.1.1}$$

where $\mathbf{R} \in \text{SO}(3)$ is the rotation matrix part of $\mathbf{T}$, the bracket $[\cdot] : \mathbb{R}^3 \to \text{so}(3)$ is the skew-symmetric operation, and $\exp : \text{so}(3) \to \text{SO}(3)$ is the exponential map from rotation vector $\text{so}(3)$ to rotation matrix $\text{SO}(3)$. In other words, $\mathbf{M}_1$ rotates the frame $\mathbf{T}$ by $\phi$ with the rotation axis $\mathbf{R}^{-1}\mathbf{w}$ which is the axis of the rotation expressed in $\mathbf{T}$. The matrix $\mathbf{M}_2$ is simply of the form

$$\mathbf{M}_2 = \begin{bmatrix} \mathbf{I}_3 & \mathbf{t_z} \\ 0 & 1 \end{bmatrix}, \tag{B.1.2}$$

where $\mathbf{t_z} = (0, 0, -t_z) \in \mathbb{R}^3$. We note that (i) $\mathbf{M}_1$ and $\mathbf{M}_2$ are uniquely defined for given $\mathbf{T}$, (ii) the projected transformation matrix $\mathbf{C} = \mathbf{T}\mathbf{M}_1\mathbf{M}_2$ is of the form

$$\mathbf{C} = \begin{bmatrix} \mathbf{Rot}(\hat{\mathbf{z}}, \theta) & \mathbf{t_{xy}} \\ 0 & 1 \end{bmatrix}, \tag{B.1.3}$$

where $\mathbf{Rot}(\hat{\mathbf{z}}, \theta) \in \text{SO}(3)$ is a $3 \times 3$ rotation matrix for rotations around $z$-axis and $\mathbf{t_{xy}} = (t_x, t_y, 0) \in \mathbb{R}^3$, and (iii) given a new object frame $\mathbf{T}' = \mathbf{C}_a\mathbf{T}$ ($\mathbf{C}_a$ has the form of Equation (B.1.3)) and $\mathbf{T} = \mathbf{C}\mathbf{U}$, the frame is uniquely expressed by $\mathbf{T}' = \mathbf{C}'\mathbf{U}$ where $\mathbf{C}' = \mathbf{C}_a\mathbf{C}$.

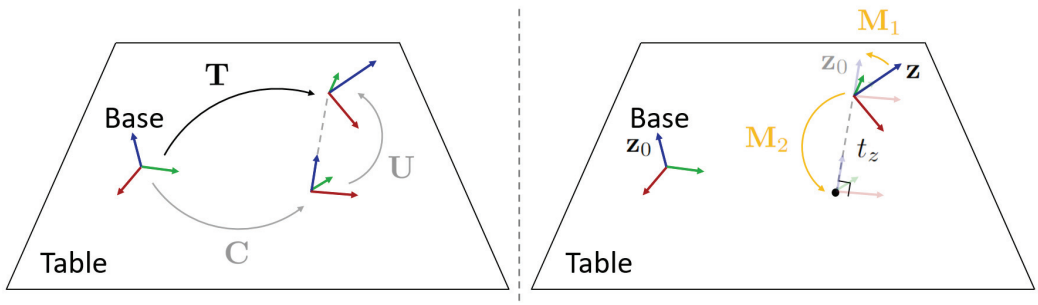


Figure B.1: Object Pose Decomposition.

### B.1.2 Proof for Equivariance

**Proposition B.1.** *A pushing dynamics model* $f = \{f_i\}_{i=1}^N$ *proposed in Section 2 is* SE(2)*-equivariant, i.e., given the inputs and outputs*

$$\{\mathbf{T}'_i\}_{i=1}^N = f(\{(\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N, (\mathbf{p}, \mathbf{v})), \tag{B.1.4}$$

*and for all rigid-body transformations that have the following form*

$$\mathbf{C} = \begin{bmatrix} \mathbf{Rot}(\hat{\mathbf{z}}, \theta) & \mathbf{t_{xy}} \\ 0 & 1 \end{bmatrix}, \tag{B.1.5}$$

*where* $\mathbf{Rot}(\hat{\mathbf{z}}, \theta)$ *is a* $3 \times 3$ *rotation matrix for rotations around* $z$*-axis and* $\mathbf{t_{xy}} = (t_x, t_y, 0) \in \mathbb{R}^3$*, the model satisfies*

$$\{\mathbf{C}\mathbf{T}'_i\}_{i=1}^N = f(\{(\mathbf{C}\mathbf{T}_i, \mathbf{q}_i)\}_{i=1}^N, (\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t_{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v})). \tag{B.1.6}$$

*Proof.* It is enough to show that one element $f_i$ is SE(2)-equivariant.

Since $\mathbf{C}\mathbf{T}'_i = (\mathbf{C}\mathbf{T}_i)\delta\mathbf{T}_i$, the claim that "$f_i$ is SE(2)-equivariant" is equivalent to the claim that "$\delta\mathbf{T}_i$ is invariant to the arbitrary transformation $\mathbf{C}$". This claim suffices to show that the network inputs are invariant to the transformation $\mathbf{C}$. Below is a description of whether each input is invariant. We note that when $\mathbf{T}_i$ is decomposed to

$\mathbf{T}_i = \mathbf{C}_i\mathbf{U}_i$, the frame $\mathbf{CT}_i$ is also decomposed to $\mathbf{CT}_i = (\mathbf{CC}_i)\mathbf{U}_i$ from the property (iii) above.

**Action.** Through a series of calculations, we below confirm that the term for the action is invariant.

$$
\begin{aligned}
&(\mathbf{CC}_i)^{-1}(\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t}_{\mathbf{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v}) \\
&= \mathbf{C}_i^{-1}\mathbf{C}^{-1}(\mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{p} + \mathbf{t}_{\mathbf{xy}}, \mathbf{Rot}(\hat{\mathbf{z}}, \theta)\mathbf{v}) \\
&= ... \\
&= \mathbf{C}_i^{-1}(\mathbf{p}, \mathbf{v})
\end{aligned}
$$

**Ego.** The frames $\mathbf{U}_i$ are invariant.

**Scene.** Since $(\mathbf{CC}_i)^{-1}(\mathbf{CT}_j) = \mathbf{C}_i^{-1}\mathbf{T}_j$ for $j = 1, ..., N, j \neq i$, the terms for the surrounding objects are also invariant.

In conclusion, $\delta\mathbf{T}_i$ is invariant to the arbitrary transformation $\mathbf{C}$, so $f_i$ is SE(2)-equivariant. Therefore, the pushing dynamics model $f = \{f_i\}_{i=1}^N$ described in Section 2 is SE(2)-equivariant. ■

## B.2 Details for Object Shape and Pose Recognition

The goal of the object shape and pose recognition is to design an algorithm that takes a partial point cloud of the objects in the scene $\mathcal{P} \subset \mathbb{R}^3$, observed from a (synthetic or real-world) depth camera, as input and outputs the superquadrics $\{\mathbf{q}_i, \mathbf{T}_i\}_{i=1}^N$, where $\mathbf{q}_i$ is the shape parameter, $\mathbf{T}_i \in \mathrm{SE}(3)$ is the pose, and $N$ is the number of the objects. We note that a noise is added to each point $\mathbf{x} \in \mathcal{P}$ – in detail, $\mathbf{x} \mapsto \mathbf{x} + m\mathbf{v}$ where $\mathbf{v}$ is uniformly sampled on $\mathbb{S}^2$ and $m$ is sampled from a Gaussian with zero-mean and standard deviation 0.001 – to bridge the sim-to-real gap on vision sensor data as described in [124, 125]. To achieve this goal to design the algorithm, we first segment a partially

observed point cloud $\mathcal{P}$ into a set of object point clouds $\{\mathcal{P}_i\}_{i=1}^N$, then convert them into superquadric representations $\mathbf{q}_i$ and $\mathbf{T}_i$.

From the raw vision sensor data $\mathcal{P} \subset \mathbb{R}^3$, the table points are discarded through plane fitting and then up/down-sampled to 2048 points. In the other words, the partially observed point cloud $\mathcal{P}$ is processed to $\mathcal{P} = \{\mathbf{x}_j \in \mathbb{R}^3\}_{j=1}^n$, where $n = 2048$.

**Point cloud segmentation.** We use the same architecture and loss function used in [72]. Since our purpose is just to separate the point cloud, the network should learn permutation-invariant segmentation labels, so the loss function should be invariant by a permutation of prediction. To achieve this, we first find a bipartite matching between ground-truth and prediction segmentation labels using the Hungarian algorithm [166], then compute the usual segmentation loss between the matched labels. This makes the loss function permutation-invariant, and guarantees that the network can separate the point cloud properly. The trained segmentation network separates the point cloud $\mathcal{P}$ into several object point clouds $\{\mathcal{P}_i\}_{i=1}^N$.

**Superquadric recognition.** Our remaining goal is to convert each segmented point cloud $\mathcal{P}_i$ to superquadric representation $\mathbf{q}_i$ and $\mathbf{T}_i$. We first construct the input representation using not only the segmented point cloud $\mathcal{P}_i$ but also surrounding point cloud $\mathcal{P}_1, \cdots, \mathcal{P}_{i-1}, \mathcal{P}_{i+1}, \cdots, \mathcal{P}_N$. In detail, we concatenate 1 after each segmented point $\mathbf{x}_j \in \mathcal{P}_i$ (i.e., $\mathbf{x}_j = (x, y, z) \mapsto (x, y, z, 1)$), and 0 after each surrounding point $\mathbf{x}_j \in \mathcal{P} \backslash \mathcal{P}_i$. We denote this newly created 4-dimensional point from a point $\mathbf{x}_j \in \mathbb{R}^3$ as $\mathbf{x}_{s,j} \in \mathbb{R}^4$, and denote the set of all these points as $\mathcal{P}_{s,i} = \{\mathbf{x}_{s,j} \in \mathbb{R}^4\}_{j=1}^n$; the set $\mathcal{P}_{s,i}$ still has $n$ points.

Then, inspired from [64], we design a neural network that takes the point cloud $\mathcal{P}_{s,i} = \{\mathbf{x}_{s,j} \in \mathbf{R}^4\}_{j=1}^n$ as input and outputs the superquadric parameter $\mathbf{q}_i$ and its pose $\mathbf{T}_i$ that best represents the full object shape as shown in Figure B.2. The network consists of (i) the EdgeConv layers [72] with latent space dimension (64, 64,

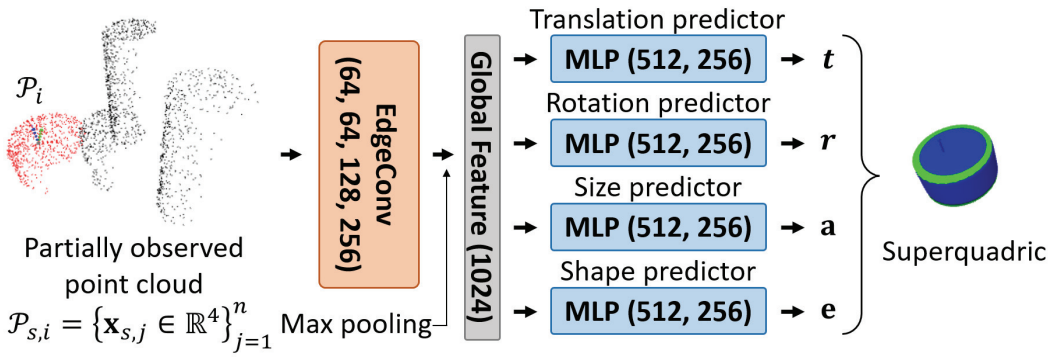


Figure B.2: Superquadric Recognition network. The red dots are the points with label 1 and the black dots are the points with label 0 in the partially observed point cloud.

128, 256) and max pooling operator to produce a global feature vector from $\mathcal{P}_{s,i}$ in a permutation-invariant manner and (ii) four fully-connected layers (MLP) with latent space dimension (512, 256) (with LeakyRelu nonlinearities) to obtain the superquadric parameter $\{a_1, a_2, a_3, e_1, e_2\}$ and the pose $\mathbf{T} = [\mathbf{R}; \mathbf{t}]$ from the extracted global feature. Especially, each MLP outputs (i) translation vector $\mathbf{t} \in \mathbb{R}^3$, (ii) quaternion vector $\mathbf{r} \in \mathbb{S}^3$ representing the rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$, (iii) size parameters $\mathbf{a} = (a_1, a_2, a_3) \in \mathbb{R}^3$, and (iv) shape parameters $\mathbf{e} = (e_1, e_2) \in \mathbb{R}^2$; the values $e_1$ and $e_2$ are bounded in $[0.2, 1.7]$ since the superquadric equation diverges when $e_1$ and $e_2$ goes to zero and shows too complex shapes when $e_1$ and $e_2$ become large.

For the predicted superquadric to fit well with the ground-truth shape, the loss function should also be designed to be the difference between the prediction and the ground-truth object shapes. For ground-truth shape, we uniformly sample the points from the surface of the object by $\mathcal{P}_{g,i} = \{\mathbf{x}_{g,j} \in \mathbb{R}^3\}_{j=1}^{n_g}$, where $n_g = 512$, and we call $\mathcal{P}_{g,i}$ the ground-truth point cloud of the $i$'the object. Then we use the distances from the

ground-truth point cloud to the predicted superquadric as the loss function. The distance form is from [84] which is defined as follows. Only in this Appendix C.1, the notation for $S$ is abused as follows:

$$S(x,y,z) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}}, \tag{B.2.7}$$

Then, the distance $\delta$ between a point $\mathbf{x}_0 \in \mathbb{R}^3$ and a superquadric surface $S(\mathbf{x}) - 1 = 0$ is

$$\delta(\mathbf{x}_0, S) = ||\mathbf{x}_0|| \left| 1 - S^{-\frac{e_1}{2}}(\mathbf{x}_0) \right|, \tag{B.2.8}$$

where $\| \cdot \|$ denotes the Euclidean norm. Accordingly, the loss function is defined as:

$$\mathcal{L} = \frac{1}{n_g} \sum_{j=1}^{n_g} \delta^2(\mathbf{T}^{-1}\mathbf{x}_{g,j}, S), \tag{B.2.9}$$

where $S$ is defined by the superquadric parameters $\{a_1, a_2, a_3, e_1, e_2\}$ and $\mathbf{T}$ is its pose.

## B.3 Details for SQPD-Net

SuperQuadric Pushing Dynamics Network (SQPD-Net) has the structure of SE(2) equivariant pushing dynamics model described in Section 2. The detail of the network architecture is shown in Figure B.3. The input dimensions are as follows: (i) the planar pushing action $\mathbf{C}_i^{-1}(\mathbf{p}, \mathbf{v})$ is represented by a 5-dimensional vector where the start point is $\mathbf{p} \in \mathbb{R}^3$ and the direction $\mathbf{v}$ is represented by $(\cos\theta, \sin\theta) \in \mathbb{R}^2$ where $\theta$ is the pushing direction in the x-y plane, (ii) the $i$'the object $(\mathbf{U}_i, \mathbf{q}_i)$ is a 12-dimensional vector where $\mathbf{U}_i \in \mathrm{SE}(3)$ is expressed by 7-dimensional (3-dimensional translation and 4-dimensional rotation quaternion) and $\mathbf{q}_i \in \mathbb{R}^5$, and (iii) the surrounding objects $\mathbf{C}_i^{-1}\mathbf{T}_j$ for $j = 1, ..., N, j \neq i$ is also 12-dimensional similar to (ii).

Action encoder, ego encoder, and scene encoder consist of the shallow MLP layers with latent space dimension (64, 128) and output feature dimension 256; for scene

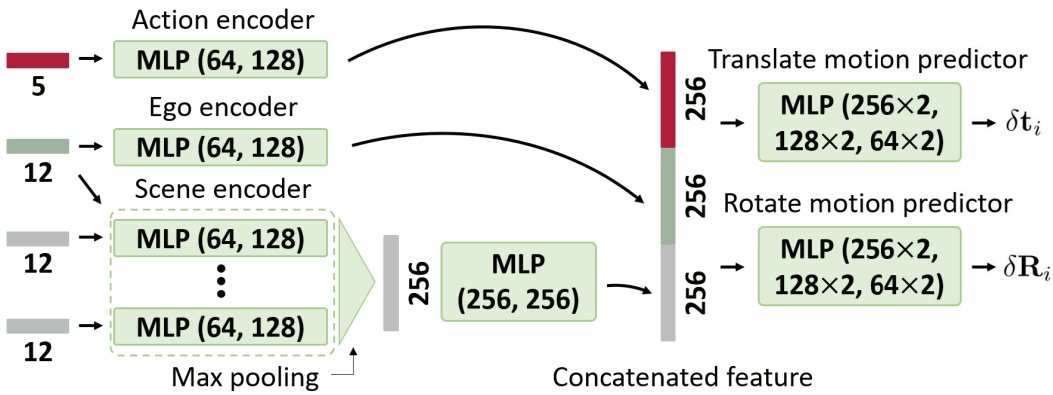


Figure B.3: Detail network architecture of SQPD-Net.

encoder, an additional MLP layer with latent dimension (256, 256) is also included. After the three feature vectors are obtained from the inputs, they are concatenated and a 768-dimensional global feature is obtained. The global feature then passes through the last MLP layers with size (256, 256, 128, 128, 64, 64) to produce the motion $\delta \mathbf{T}_i = [\delta \mathbf{R}_i; \delta \mathbf{t}_i]$, consists of the translate motion vector $\delta \mathbf{t}_i$ and rotation motion vector $\delta \mathbf{R}_i$ expressed in quaternion. In this work, we consider the planar pushing motions of the objects, so the predicted motion $\delta \mathbf{T}_i$ is of the form Equation (B.1.3). We note that all MLPs are followed by LeakyRelu nonlinearities.

**Distance measure on** $\mathrm{SO}(3)$**.** The general distance measure is the Frobenius norm of the difference of two rotation matrices as follows:

$$d_{\mathrm{SO}(3)}(\mathbf{R}_1, \mathbf{R}_2) = \|\mathbf{R}_1 - \mathbf{R}_2\|_F, \tag{B.3.10}$$

where $\mathbf{R}_1, \mathbf{R}_2 \in \mathrm{SO}(3)$ are the rotation matrices. We can use this distance measure for training, but we only consider planar pushing motions of the objects in this paper, we use simpler distance metric as follows:

$$d_{\mathrm{SO}(3)}(\mathbf{R}_1, \mathbf{R}_2) = 1 - \cos(\theta_1 - \theta_2), \tag{B.3.11}$$

where $\theta_1$ and $\theta_2$ are the rotation angle of $\mathbf{R}_1$ and $\mathbf{R}_2$, when the rotation matrices are represented as $\mathbf{R}_i = \mathbf{Rot}(\hat{\mathbf{z}}, \theta_i)$

## B.4 Details for Pushing Manipulation

We use the sampling-based MPCs [97]. At each timestep $t$, when the observation $\mathbf{o}_t$ (for our case, the partially observed point cloud $\mathcal{P}_t$) is obtained, the shape recognition is performed and obtain the shape parameters $\mathbf{q}_{t,i}$ and poses $\mathbf{T}_{t,i}$. We denote by $\mathbf{s}_t = \{(\mathbf{T}_{t,i}, \mathbf{q}_{t,i})\}_{i=1}^{N}$. From the recognized objects $\mathbf{s}_t$, we sample 100 action sequences; the time horizon of each sequence is one for moving and singulation tasks and three for grasping tasks. Then, using our trained SQPD-Net, we compute the next objects' poses (i.e., $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$) and accordingly compute cost functions using $\mathbf{s}_{t+1}$ for all sampled actions. Then we find an optimal action that best minimizes the cost function. The action sequences are resampled and the optimal action is chosen every timestep $t$.

## B.5 Details for Grasping Cost Function

This subsection includes the details for the calculation of the grasping cost function.

**Candidate grasp poses.** When a superquadric representation of the target object is obtained from the shape and pose recognition, we generate candidate grasp poses. Grasp poses can be generated in a general superquadric through sampling-based methods [64]; in this work, we use a simple rule-based method for grasp pose generation. We generate top-down and side grasp poses (with 4 directions) according to the shape of the superquadric. At this time, the two gripper fingers should be on the antipodal points on the object. For each approaching direction, 6 grasp poses are generated (maximum 30 grasp poses). Grasp poses with a distance between the antipodal points greater than 7cm

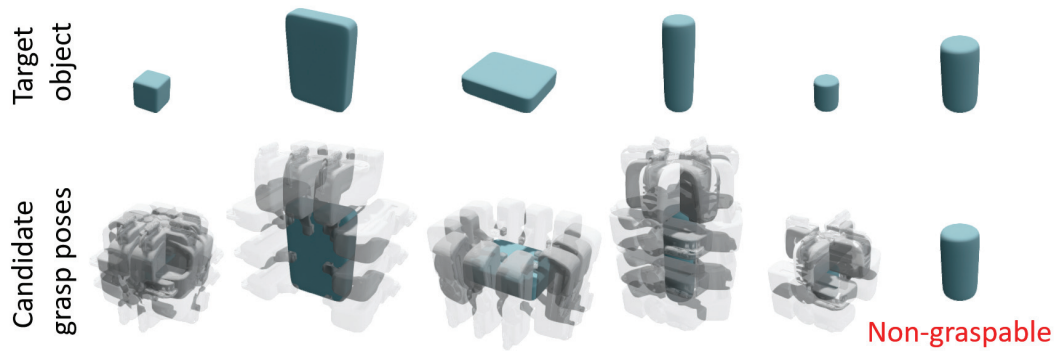


Figure B.4: Candidate grasp poses for various recognized superquadric shapes.

are removed from the candidates (the maximum gripper width of the Franka gripper is 8cm). The grasp poses generated from various object shapes are shown in Figure B.4.

**Gripper collision detection.** After generating candidate grasp poses, we check the collisions with the environment and the other recognized objects. The collision detection has to be computed thousands of times to calculate the cost in one step of sampling-based MPC (exactly, it is the product of the number of sampled actions and the number of grasp poses generated above), so it is difficult to use the traditional collision checking algorithm between meshes. Instead, we introduce an efficient method that utilizes the advantage of shape recognition through an implicit function. We first sample the points on the gripper mesh in the maximum open state as shown in Figure B.5; the sampled points are denoted by $\mathcal{P}_{gr} = \{\mathbf{x}_{gr,j} \in \mathbf{R}^3\}_{j=1}^{n_{gr}}$, where $n_{gr} = 512$. For an implicit object representation $S(\mathbf{x}) = 0$, we note that a point $\mathbf{x}_0 \in \mathbb{R}^3$ is inside the object when $S(\mathbf{x}_0)$ is less than 0 and outside when $S(\mathbf{x}_0)$ is greater than 0. We use this fact to determine whether the gripper collides with the objects or tables or not: when the value

$$\min_j S(\mathbf{T}_{gr}^{-1}\mathbf{x}_{gr,j}), \tag{B.5.12}$$

where $\mathbf{T}_{gr}$ is the pose of the gripper, is less than 0, then the gripper collides with the

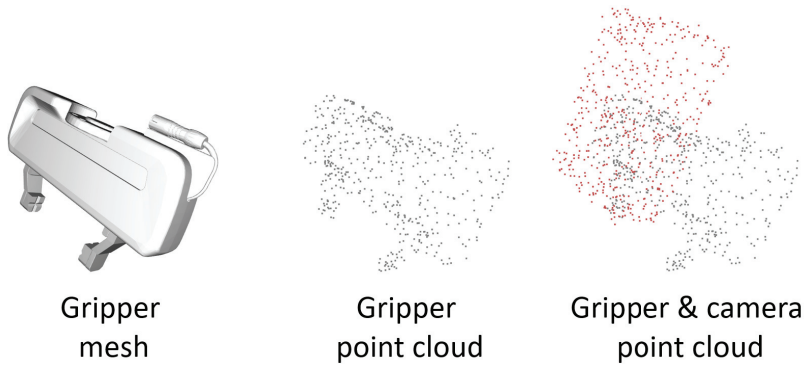


Figure B.5: Gripper mesh, sampled gripper point cloud from the mesh, and point cloud with the camera's point cloud.

object. Through this, collisions can be checked quickly and efficiently.

In real-world pushing manipulation experiments, the gripper is equipped with an Azure Kinect camera. To account for this, we also sample and use the camera point cloud to check the collision, as shown in the right of Figure B.5. In this case, 1024 points are sampled (i.e., $n_{gr} = 1024$) on both the gripper and the camera.

**Grasping criteria.** Let the recognized shapes' implicit representations be $S_1(\mathbf{x}) = 0, S_2(\mathbf{x}) = 0, \cdots, S_N(\mathbf{x}) = 0$, and additionally, the table's implicit representation be $S_{N+1}(\mathbf{x}) = 0$ (the box-shaped table can also be represented by superquadric equation). Let $\mathbf{T}_{gr,1}, \cdots, \mathbf{T}_{gr,N_c} \in \mathrm{SE}(3)$ be the candidate grasp poses. Then the terminal cost is defined by:

$$q(\mathbf{s}_{T+1}) = 1 - \max_k \left[ \mathbb{1}(\min_{i,j} S_i(\mathbf{T}_{gr,k}^{-1}\mathbf{x}_{gr,j}) > 0) \circ \mathbb{1}(\mathbf{T}_{gr,k} \text{ is kinematically feasible}) \right], \tag{B.5.13}$$

where $i = 1, \cdots, N+1$ is the object index, $j = 1, \cdots, n_{gr}$ is the gripper point cloud index, $k = 1, \cdots, N_c$ is the candidate grasp pose index, the indicator function $\mathbb{1}(\cdot)$ is $N_c$-dimensional vector, and $\circ$ is the element-wise multiplication. The terminal cost

is 0 if at least one kinematically feasible and collision-free grasp pose exists and 1 otherwise. When the terminal cost achieves 0, grasping proceeds by selecting one of the grasp poses that satisfy both conditions, i.e., $\mathbf{T}_{gr,k}$ such that

$$\min_{i,j} S_i(\mathbf{T}_{gr,k}^{-1}\mathbf{x}_{gr,j}) > 0 \text{ and } \mathbf{T}_{gr,k} \text{ is kinematically feasible.} \tag{B.5.14}$$

# C

# Appendix: Search-for-Grasp

## C.1 Details for Object Shape Recognition

The object shape recognition is an algorithm that takes a partial observation from a (synthetic or real-world) RGB-D camera as input and outputs the 3D shapes of the objects in the scene. Especially, the input is a partial point cloud of the scene obtained from a depth camera $\mathcal{P} \subset \mathbb{R}^3$ (the RGB image is only used to check whether the target object is detected or not) and output is the superquadric representations $\{\mathbf{q}_i, \mathbf{T}_i\}_{i=1}^N$, where $\mathbf{q}_i$ is the shape parameter, $\mathbf{T}_i \in \mathrm{SE}(3)$ is the pose, and $N$ is the number of the objects. To design a model that performs this task, we use the same method as in the previous work [65]. We first segment a partially observed point cloud $\mathcal{P}$ into a set of object point clouds $\{\mathcal{P}_i\}_{i=1}^N$ and then convert each segmented point cloud $\mathcal{P}_i$ to superquadric representation $(\mathbf{q}_i, \mathbf{T}_i)$. The segmentation and superquadric recognition processes are based on neural network models. Each model is trained from synthetic dataset obtained through simulation environment, and the trained models are directly applied to both the simulation environment and the real-world environment. The overall

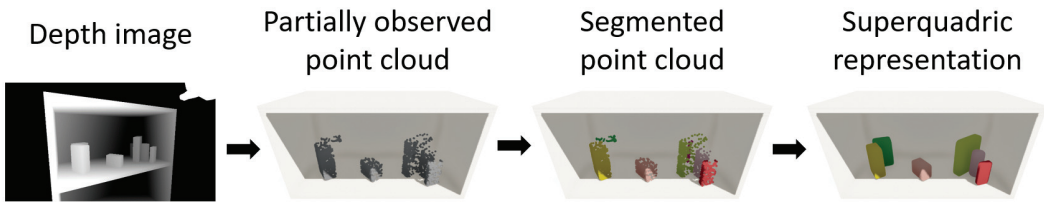


Figure C.1: Overall process for object shape recognition.

object shape recognition process is described in Figure C.1.

**Point cloud processing.** For a partially observed point cloud $\mathcal{P}$ obtained from **simulation environment**, to bridge the sim-to-real gap, a noise is added to each point of the point cloud $\mathbf{x} \in \mathcal{P}$ using the map $\mathbf{x} \mapsto \mathbf{x}+m\mathbf{v}$ where $\mathbf{v}$ is uniformly sampled on $\mathbb{S}^2$ and $m$ is sampled from a Gaussian with zero-mean and standard deviation 0.001. The all points corresponding to the shelf in the point cloud are removed and up/downsampled so that the number of points is 2048, i.e., $\mathcal{P} = \{\mathbf{x}_j \in \mathbb{R}^3\}_{j=1}^{n}$, where $n = 2048$. For a partially observed point cloud $\mathcal{P}$ obtained from **real-world environment**, the shelf points are removed through the known shelf shape information and pose; since there's noise on shelf pose and observed point cloud, we additionally remove the points corresponding to the floor of the shelf where the objects are placed through RANSAC plane fitting. Then as in the case of the simulation environment, the point cloud is up/downsampled so that the number of points is 2048.

**Point cloud segmentation.** After the point cloud $\mathcal{P}$ is processed, it is separated into several object point clouds $\{\mathcal{P}_i\}_{i=1}^{n}$ via a segmentation network. For the network architecture, we use the same architecture used in [72]. To train the segmentation network, we first find a bipartite matching between ground-truth and predicted segmentation labels using the Hungarian matching algorithm [166], and then define the loss function as the segmentation loss between the matched labels. This loss function is invariant

to the point permutation of prediction, so the network can learn permutation-invariant segmentation labels and can be trained faster and more accurately accordingly.

**Superquadric recognition.** Each segmented object point cloud $\mathcal{P}_i$ is then converted to the 3D full shape represented by superquadric $\mathbf{q}_i \in \mathbb{R}^5$ and $\mathbf{T}_i \in \mathrm{SE}(3)$ via a recognition network proposed in [64, 65]. For the network architecture, we use the same architecture used in [65]; the input representation is a point cloud with 4-dimensional points $\mathcal{P}'_i = \{\mathbf{x}_{ij} \in \mathbb{R}^4\}_{j=1}^{n}$ – for each point $\mathbf{x}_{ij}$, the first three components of $\mathbf{x}_{ij}$ is equal to $\mathbf{x}_j$ and the last element of each point is 1 if $\mathbf{x}_{ij} \in \mathcal{P}_i$ and 0 otherwise for all $j = 1, ..., n$ – and output representation is $(\mathbf{q}_i, \mathbf{T}_i)$. The input and output of the superquadric recognition model are described in Figure C.2. To train this recognition network, we adopt the training loss function as the difference between ground-truth and predicted object shapes. For the ground-truth shape, we use the point cloud uniformly sampled from the surface of the object $\mathcal{P}_i^g = \{\mathbf{x}_j^g \in \mathbb{R}^3\}_{j=1}^{n_g}$ where $n_g = 512$. Then we use the distances from the ground-truth point cloud to the predicted superquadric as the loss function. Especially, the distance form $\delta$ proposed in [84] is used. The distance $\delta$ is defined as follows: for the superquadric surface equation $S$ expressed as

$$S(x, y, z) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}}, \tag{C.1.1}$$

the distance $\delta$ between a point $\mathbf{x} \in \mathbb{R}^3$ and a superquadric surface $S(\mathbf{x}) - 1 = 0$ defined by

$$\delta(\mathbf{x}_0, S) = ||\mathbf{x}|| \left| 1 - S^{-\frac{e_1}{2}}(\mathbf{x}) \right|, \tag{C.1.2}$$

where $\| \cdot \|$ denotes the Euclidean norm. Accordingly, the training loss function for the recognition network is defined as:

$$\mathcal{L} = \frac{1}{n_g} \sum_{j=1}^{n_g} \delta^2(\mathbf{T}^{-1}\mathbf{x}_j^g, S). \tag{C.1.3}$$

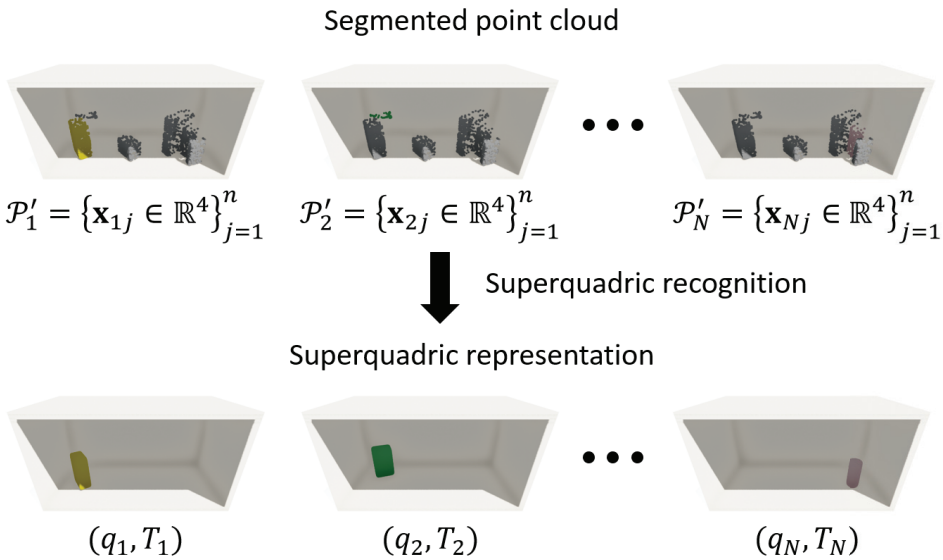


Figure C.2: Input and output representation of the superquadric recognition model.

where $S$ is defined by the predicted superquadric parameters $\mathbf{q} = \{a_1, a_2, a_3, e_1, e_2\}$ and $\mathbf{T}$ is the predicted object pose.

We generate a training dataset to train the above two networks, the segmentation network and the superquadric recognition network. To generate data, we randomly generate $N$ objects consisting of cubes and cylinders with various shape parameters (i.e., width, height, depth for the cube, and radius, and height for the cylinder); the number of objects $N$ varies from 2 to 8. The generated objects are then placed in a random position and orientation on the shelf. After that, we construct a data tuple consisting of 3 components: (i) partially observed point cloud $\mathcal{P}$ from depth camera, (ii) segmentation label for each point on $\mathcal{P}$, and (iii) ground-truth point cloud (i.e., point cloud sampled from ground-truth shape) for each object $\mathcal{P}_i^g$ for $i = 1, ..., N$. For each number of objects $N$, we collect data until the numbers of the data tuples become 5000/100 for training/validation set, respectively. As a result, the total numbers of the data tuples are

35000/700 for training/validation set, respectively. The validation set is used to select the best model for the segmentation network and the superquadric recognition network.

## C.2 Details for Existence Function Estimate $\hat{f}$

In this section, we describe how to calculate the existence function estimate $\hat{f}$. The function $\hat{f}(x; \mathbf{s}, c)$ is defined by the recognized superquadric parameters $\mathbf{s} = \{(\mathbf{q}_i, \mathbf{T}_i)\}$ and the visibility of the target object $c \in \{0, 1\}$. The function's input is a hypothetical pose $x \in \mathcal{X} \subset \mathrm{SE}(3)$ and output is an indicator whether the target object can be present at the pose $x$ (i.e., $\hat{f}(x; \mathbf{s}, c) = 1$) or not (i.e., $\hat{f}(x; \mathbf{s}, c) = 0$). The calculation of $\hat{f}$ is trivial if $c = 1$ since the existence function $\hat{f}$ – we know at which $x^*$ the target object exist – is 1 only at $x^*$, i.e., $\hat{f}(x^*; \mathbf{s}, c = 1) = 1$ and 0 at other $x \in \mathcal{X}$. So we consider only the case where $c = 0$, i.e., the target object is not visible. If $c = 0$, $\hat{f}(x; \mathbf{s}, c)$ is defined to be 1 if (i) depth rendering results with and without the target object at $x \in \mathcal{X}$ are identical and (ii) there is no collision between the recognized objects, the environment, and the target object. Otherwise $\hat{f}(x; \mathbf{s}, c) = 0$. We describe how to compute the above two conditions by taking advantage of the superquadric as an implicit function.

**Depth rendering condition.** We check the depth rendering condition, i.e., calculate the function $\hat{f}_d : \mathcal{X} \to \{0, 1\}$ where $\hat{f}_d(x) = 1$ if the depth rendering results with and without the target object at $x \in \mathcal{X}$ are identical and $\hat{f}_d(x) = 0$ otherwise. To calculate this, a depth rendering function that takes the 3D object shapes $\mathbf{s}$ as input and outputs the corresponding depth image $\mathbf{D} \in \mathbb{R}^{H \times W}$ is required; the intrinsic and extrinsic parameters of the camera used for depth rendering are known. In this paper, depth image can be rendered quickly using the superquadric implicit function. The overview of the depth rendering process from recognized superquadric functions is described in

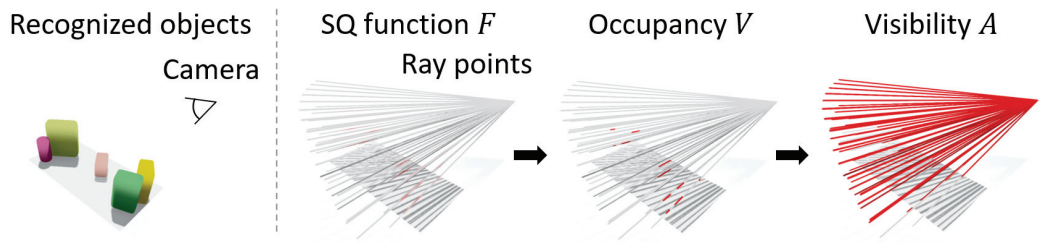


Figure C.3: Overview of depth image rendering process from recognized superquadric functions.

Figure C.3.

We first obtain the camera rays from the camera's intrinsic parameters, extrinsic parameters, and the resolution of the camera as described in [168], and we denote each ray's equation corresponding to the $(k, l)$-th pixel of the depth image as $\mathbf{r}_{kl} : [t_n, t_f] \rightarrow \mathbb{R}^3$ where $k = 1, ..., H$, $l = 1, ..., W$, and $t_n$ and $t_f$ are the near and far bounds to measure distance; in this paper, we set this values as 0 and 1.5, respectively. Each ray is a straight line $\mathbf{r}_{kl}(t) = \mathbf{o}_c + t\mathbf{d}_{kl}$ where $\mathbf{o}_c \in \mathbb{R}^3$ is the position of the camera pose and $\mathbf{d}_{kl} \in S^2$ is a direction vector of the ray. The camera rays are shown in Figure C.3.

After recognizing the objects, we obtain $N$ superquadric parameters and their poses $\mathbf{s} = \{(\mathbf{q}_i, \mathbf{T}_i)\}_{i=1}^N$. We recall that the superquadric equation with the superquadric parameters $\mathbf{q}_i$ is expressed by

$$S(x, y, z; \mathbf{q}_i) = \left( \left| \frac{x}{a_1} \right|^{\frac{2}{e_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{e_2}} \right)^{\frac{e_2}{e_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{e_1}}, \tag{C.2.4}$$

where $\mathbf{q}_i = (a_1, a_2, a_3, e_1, e_2)$. Then, we convert the parameters from recognition to implicit functions $S_i(\mathbf{x}) = S^{e_1}(\mathbf{T}_i^{-1}\mathbf{x}; \mathbf{q}_i) = 1$ for $i = 1, ..., N$. Using the obtained implicit functions, we calculate the signed distance-like superquadric function $F$ as:

$$F(\mathbf{x}) = \min_i S_i(\mathbf{x}) \tag{C.2.5}$$

Using this superquadric function, we can approximately calculate the occupancy function $V(\mathbf{x})$ where $V(\mathbf{x}) = 1$ if the point $\mathbf{x}$ is occupied and $V(\mathbf{x}) = 0$ otherwise. The occupancy can be calculated using the equation

$$V(\mathbf{x}) = \sigma(p(1 - F(\mathbf{x}))), \tag{C.2.6}$$

where $\sigma$ is the sigmoid function and $p$ is a scaling factor that adjusts the sharpness of $V$ – we set $p = 1000$ – as proposed in [169, 170]. Then the visibility function $A_{kl}(t)$ – which indicates whether a point is visible in the rendered camera image – on a ray point $\mathbf{r}_{kl}(t)$ can be obtained as

$$A_{kl}(t) = \exp(-\tau \int_{t_n}^{t} V(\mathbf{r}_{kl}(t'))dt'), \tag{C.2.7}$$

with large enough $\tau$ – we set $\tau = 100$ – as proposed in [171]. Finally, the depth value $\mathbf{D}_{kl}$ on the $(k, l)$-th pixel of the rendered depth image $\mathbf{D}$ is calculated as

$$\mathbf{D}_{kl} = t_n + \int_{t_n}^{t_f} A_{kl}(t)dt, \tag{C.2.8}$$

where $k = 1, ..., H$ and $l = 1, ..., W$. The integrals above can be calculated numerically after uniformly dividing $t \in [t_n, t_f]$.

The integration is required for all $(k, l)$ pairs to obtain a perfect depth image $\mathbf{D}$, but it is not necessary to calculate for all $(k, l)$ pairs to check the depth rendering condition. Instead, we additionally propose an algorithm that (i) finds pixels around the recognized object (i.e., $S_i$'s) and the target object and (ii) calculates the depth value only for those pixels.

We additionally propose a modified method to calculate the depth rendering condition efficiently. The key idea is that since our region of interest is the objects on the shelf, the above calculation will also be done only near the objects. The modified algorithm is based on the fact that an arbitrary superquadric $(\mathbf{q}, \mathbf{T})$ can be contained in

an ellipsoid $E$; in detail, for $\mathbf{q} = (a_1, a_2, a_3, e_1, e_2)$, the ellipsoid equation $E$ is given by

$$E(x, y, z) = \frac{x^2}{3a_1^2} + \frac{y^2}{3a_2^2} + \frac{z^2}{3a_3^2}, \tag{C.2.9}$$

and the superquadric $(\mathbf{q}, \mathbf{T})$ is contained in the ellipsoid $E(\mathbf{T}^{-1}\mathbf{x}) \leq 1$.

Using this fact, the modified method (i) reduces the number of camera rays $\mathbf{r}_{kl}$ required for calculation, and (ii) efficiently samples the ray points $\mathbf{r}_{kl}(t)$ required for calculation in each ray $\mathbf{r}_{kl}$. For the first one, for each ray's equation $\mathbf{r}_{kl}(t) = \mathbf{o}_c + t\mathbf{d}_{kl}$, we calculate whether $\mathbf{r}_{kl}(t)$ meets the ellipsoids $E(\mathbf{T}^{-1}\mathbf{x}) \leq 1$. If $\mathbf{r}_{kl}(t)$ does not meet any ellipsoid of the object, the depth value $\mathbf{D}_{kl}$ is set to $t_f$, and otherwise, $\mathbf{D}_{kl}$ is calculated following (C.2.8). Therefore, we only need to consider the rays that meet at least one of the object ellipsoids. For the second one, instead of uniformly dividing $[t_n, t_f]$ for numerical integration of (C.2.7) and (C.2.8), we use more efficient method for dividing; we obtain the intersection points between the ray and the ellipsoid, and then uniformly divide the chord corresponding to the intersections. Since all of the above calculations exist in closed-form, they have little effect on the amount of calculation.

Using the above depth rendering module, we finally obtain the function $\hat{f}_d : \mathcal{X} \to \{0, 1\}$ to check the depth rendering condition; especially, $\hat{f}_d(x) = 1$ if the MSE (Mean Squared Error) between the rendered depth images with and without the target object at $x \in \mathcal{X}$ is lower than the threshold – we set this threshold as 0.001 – and $\hat{f}_d(x) = 0$ otherwise.

**Collision condition.** We check the collision condition, i.e., calculate the function $\hat{f}_c : \mathcal{X} \to \{0, 1\}$ where $\hat{f}_c(x) = 1$ if there is no collision between the recognized objects, the environment, and the target object at $x \in \mathcal{X}$ and $\hat{f}_c(x) = 0$ otherwise. This function also can be quickly evaluated using the superquadric implicit function.

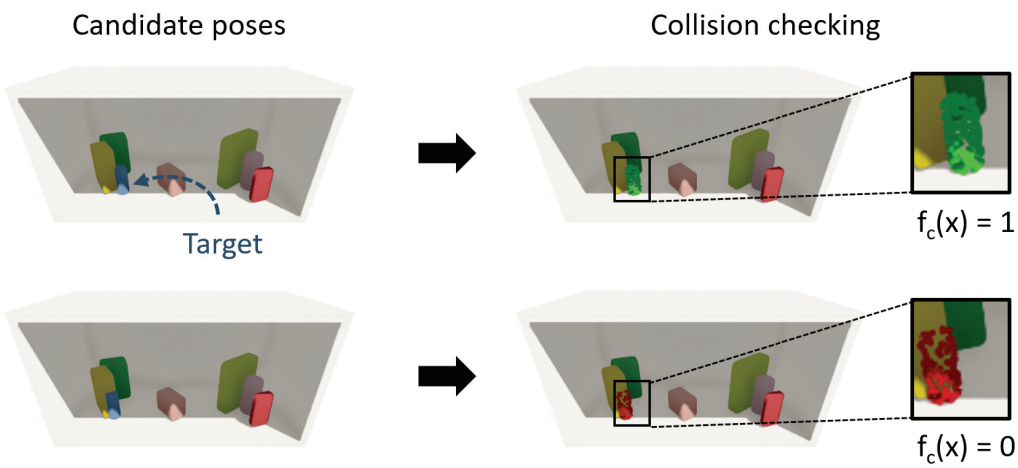


Figure C.4: Illustration on the collision condition $f_c(x)$. Candidate poses and collision checking results.

We first obtain the target object point cloud sampled from the surface of the target object $\mathcal{P}_o = \{\mathbf{x}_j^o \in \mathbb{R}^3\}_{j=1}^{n_o}$, where $n_o = 512$. Similar to the case of depth rendering condition, we also get the superquadric implicit functions $S_i$ for $i = 1, ..., N$ from the recognized objects. We note that a point $\mathbf{x} \in \mathbb{R}^3$ is inside the $i$'th object $S_i$ when the value $S_i(\mathbf{x})$ is less than 1 and otherwise outside. We utilize this fact to check whether the target object collides with the recognized objects or not. The function $\hat{f}_c : \mathcal{X} \rightarrow \{0, 1\}$ is defined as:

$$\hat{f}_c(x) = \mathbb{1}(\min_{i,j} S_i(\mathbf{x}_j^o) > 1) \tag{C.2.10}$$

where $i = 1, ..., N$ is the object index, $j = 1, ..., n_o$ is the point index of the target object point cloud, and $\mathbb{1}(\cdot)$ is the indicator function. The calculation of the collision condition is described in Figure C.4.

**Existence function estimate.** Using the above functions to check the conditions, $\hat{f}_d$

and $\hat{f}_c$, we define the existence function estimate $\hat{f} : \mathcal{X} \to \{0, 1\}$ as follows,

$$\hat{f}(x) = \hat{f}_d(x)\hat{f}_c(x).$$

In the other words, the existence function is 1 at a pose $x \in \mathcal{X}$ if both the depth rendering condition and the collision condition holds and 0 otherwise.

**Computational cost.** The calculation time takes $0.013$ seconds in average where $WH = 18,564$, $n_p = 7$, $N = 4$, and $n_o = 512$ with GeForce RTX 3090 and Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz. We design the existence function is calculated in batch-wise with respect to the target pose $x$ to accelerate the computation time for MPC.

## C.3 Details for Graspability Function Estimate $\hat{g}$

In this section, we describe how to calculate the graspability function estimate $\hat{g}$. The function $\hat{g}(x; \mathbf{s})$ is defined by the recognized superquadric parameters $\mathbf{s} = \{(\mathbf{q}_i, \mathbf{T}_i)\}$. The function's input is a hypothetical pose $x \in \mathcal{X} \subset \mathrm{SE}(3)$ and output is an indicator whether a collision-free grasping trajectory of the robot gripper – all possible collisions between the robot arm, the robot gripper, the shelf, and the surrounding objects should be taken into account – can be find at the pose $x$ (i.e., $\hat{g}(x; \mathbf{s}) = 1$) or not (i.e., $\hat{g}(x; \mathbf{s}) = 0$). We then describe how to compute the graspability function by also taking the advantage of the superquadrics as an implicit function similar to the calculation of the existence function.

**Candidate grasping trajectories.** To check the graspability, i.e., to check whether a collision-free grasping trajectory exists, we generate candidate grasping trajectories to grasp the target object. To achieve this, we first generate candidate grasp poses for the target object using a simple rule-based method as introduced in [65]. In this paper,

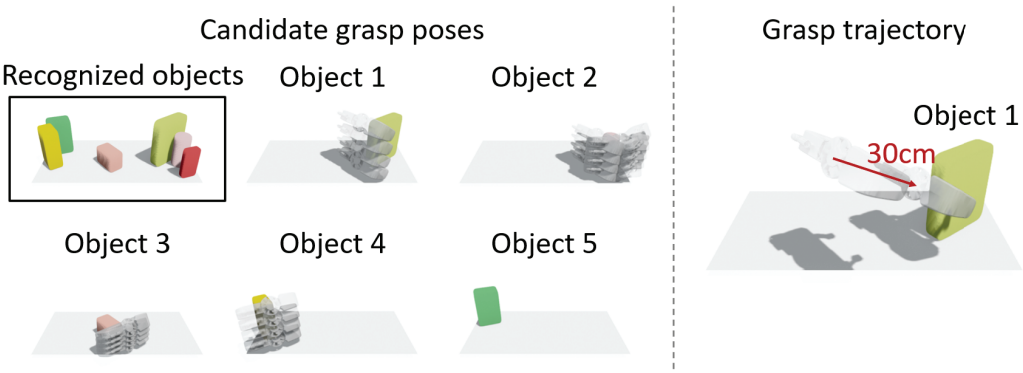


Figure C.5: The examples of the candidate grasp poses for various object shapes (*Left*) and the robot trajectory for a selected grasp pose (*right*).

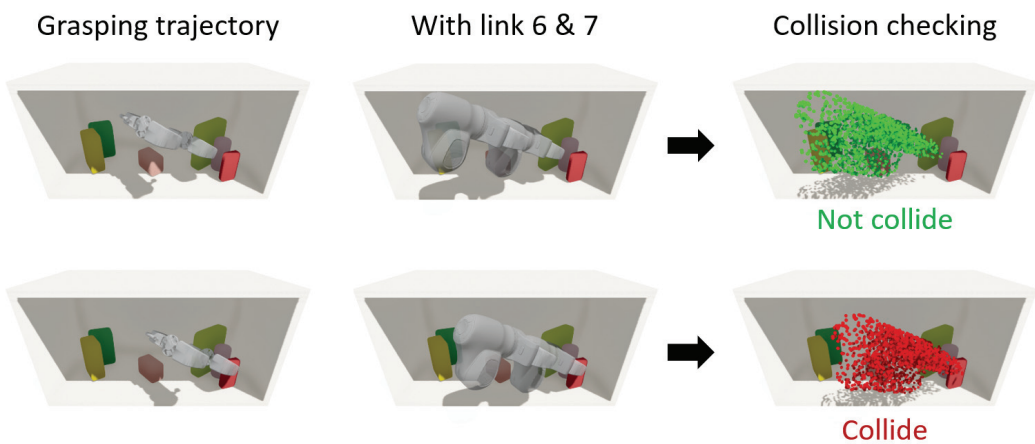


Figure C.6: Illustration on the grasp trajectory collision detection. Candidate grasp trajectories and collision checking results.

we generate side grasp poses according to the shape and size of the superquadric. The pairs of the antipodal points of the target objects are sampled since we use two-finger gripper, and the grasp poses with a distance of bigger than 7.5cm between antipodal points are rejected since the maximum gripper width of the Franka gripper is 8cm. To stably grasp the target object on the shelf, we tilted the grasp poses about 20° relative to the ground. Accordingly, we generate $N_{gr}$ grasp poses $\{\mathbf{T}_k^{gr}\}_{k=1}^{N_{gr}}$ where $N_{gr}$ is between 10 and 30; the examples of the grasp poses for various object shapes are shown in the left of Figure C.5. Then, we additionally have to plan the trajectory of the robot arm for each grasp pose. For a grasp pose $\mathbf{T}_k^{gr} \in \mathrm{SE}(3)$, we design a robot trajectory where the gripper approaches about 30cm along the z-direction of the gripper frame as shown in the right of Figure C.5. Finally, we get $N_{gr}$ candidate grasping trajectories for the target object.

**Grasp trajectory collision detection.** After generating grasping trajectories, we should check whether the robot arm following the trajectory collides with the surrounding objects or the shelf. We first obtain the afterimage mesh of the gripper and Franka's links 7 and 6 when the robot follows the trajectory of the grasp pose $\mathbf{T}_k^{gr}$ and obtain the point cloud $\mathcal{P}_k^{gr} = \{\mathbf{x}_{kj}^{gr} \in \mathbb{R}^3\}_{j=1}^{n_{gr}}$ sampled from the afterimage mesh, where $n_{gr} = 2048$. We get the superquadric implicit functions $S_i(\mathbf{x}) = S^{e_1}(\mathbf{T}_i^{-1}\mathbf{x}; \mathbf{q}_i) = 1$ for $i = 1, ..., N$ from the recognized surrounding objects. We additionally represent the shelf as the superquadric implicit functions; a shelf can be represented by five boxes so five implicit functions $S_{N+1}(\mathbf{x}), ..., S_{N+5}(\mathbf{x})$ are additionally considered. We recall from the collision condition of existence function that a point $\mathbf{x} \in \mathbb{R}^3$ is inside the $i$'th object $S_i$ when the value $S_i(\mathbf{x})$ is less than 1 and otherwise outside. The collision

function $\hat{g}_c : \mathcal{X} \to \{0, 1\}$ is defined as:

$$\hat{g}_c(x) = \mathbb{1}(\max_k \min_{i,j} S_i(\mathbf{x}_{kj}^{gr}) > 1) \tag{C.3.11}$$

$$= 1 - \mathbb{1}(\max_k \min_{i,j} S_i(\mathbf{x}_{kj}^{gr}) <= 1) \tag{C.3.12}$$

where $i = 1, ..., N+5$ is the object index, $j = 1, ..., n_{gr}$ is the point index of the target object point cloud, $i = 1, ..., N_{gr}$ is the grasp pose index, and $\mathbb{1}(\cdot)$ is the indicator function. The calculation of the collision of the grasp trajectories is described in Figure C.6. In practice, the modified collision detection function $\hat{g}_{c,m} : \mathcal{X} \to \{0, 1\}$ we used is:

$$\hat{g}_{c,m}(x) = 1 - \min_k \sum_i \mathbb{1}(\min_j S_i(\mathbf{x}_{kj}^{gr}) <= 1) \tag{C.3.13}$$

**Graspability function.** We define the graspability function estimate $\hat{g} : \mathcal{X} \to \{0, 1\}$ as the same with $\hat{g}_c$, i.e.,

$$\hat{g}(x) = \hat{g}_c(x). \tag{C.3.14}$$

The modified graspability function estimate is the same with $\hat{g}_{c,m}$, i.e.,

$$\hat{g}(x) = \hat{g}_{c,m}(x). \tag{C.3.15}$$

**Computational cost.** The calculation time takes $0.0128$ seconds in average, where $N + 5 = 6$, $n_{gr} = 2048$, and $N_{gr} = 32$ with GeForce RTX 3090 and Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz.

# Bibliography


[1] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 348–353. IEEE, 2000.

[2] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, 2012.

[3] Matthew T Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986.

[4] Kevin M Lynch. Estimating the friction parameters of pushed objects. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 1, pages 186–193. IEEE, 1993.

[5] Kevin M Lynch and Matthew T Mason. Stable pushing: Mechanics, controllability, and planning. *The international journal of robotics research*, 15(6):533–556, 1996.

[6] Jiaji Zhou, Yifan Hou, and Matthew T Mason. Pushing revisited: Differential flatness, trajectory planning, and stabilization. *The International Journal of Robotics Research*, 38(12-13):1477–1489, 2019.

[7] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[8] 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.

[9] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[10] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2901–2910, 2019.

[11] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F Huber. A survey on learning-based robotic grasping. *Current Robotics Reports*, pages 1–11, 2020.

[12] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE, 2021.

[13] Xibai Lou, Yang Yang, and Changhyun Choi. Collision-aware target-driven object grasping in constrained environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6364–6370. IEEE, 2021.

[14] Xupeng Zhu, Dian Wang, Ondrej Biza, Guanang Su, Robin Walters, and Robert Platt. Sample efficient grasp learning using equivariant models. *arXiv preprint arXiv:2202.09468*, 2022.

[15] Julen Urain, Niklas Funk, Jan Peters, and Georgia Chalvatzaki. Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5923–5930. IEEE, 2023.

[16] Rhys Newbury, Morris Gu, Lachlan Chumbley, Arsalan Mousavian, Clemens Eppner, Jürgen Leitner, Jeannette Bohg, Antonio Morales, Tamim Asfour, Danica Kragic, et al. Deep learning approaches to grasp synthesis: A review. *IEEE Transactions on Robotics*, 2023.

[17] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[18] Michael Danielczuk, Jeffrey Mahler, Chris Correa, and Ken Goldberg. Linear push policies to increase grasp access for robot bin picking. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1249–1256. IEEE, 2018.

[19] Kechun Xu, Hongxiang Yu, Qianen Lai, Yue Wang, and Rong Xiong. Efficient learning of goal-oriented push-grasping synergy in clutter. *IEEE Robotics and Automation Letters*, 6(4):6337–6344, 2021.

[20] Eddie Sasagawa and Changhyun Choi. Fixture-aware ddqn for generalized environment-enabled grasping. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3151–3158. IEEE, 2022.

[21] Enbo Li, Haibo Feng, Songyuan Zhang, and Yili Fu. Learning target-oriented push-grasping synergy in clutter with action space decoupling. *IEEE Robotics and Automation Letters*, 7(4):11966–11973, 2022.

[22] Min Zhao, Guoyu Zuo, Shuangyue Yu, Daoxiong Gong, Zihao Wang, and Ouattara Sie. Position-aware pushing and grasping synergy with deep reinforcement learning in clutter. *CAAI Transactions on Intelligence Technology*, 2023.

[23] Gaoyuan Liu, Joris De Winter, Denis Steckelmacher, Roshan Kumar Hota, Ann Nowe, and Bram Vanderborght. Synergistic task and motion planning with reinforcement learning-based non-prehensile actions. *IEEE Robotics and Automation Letters*, 8(5):2764–2771, 2023.

[24] Marios Kiatos and Sotiris Malassiotis. Robust object grasping in clutter via singulation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1596–1600. IEEE, 2019.

[25] Iason Sarantopoulos, Marios Kiatos, Zoe Doulgeri, and Sotiris Malassiotis. Split deep q-learning for robust object singulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6225–6231. IEEE, 2020.

[26] Iason Sarantopoulos, Marios Kiatos, Zoe Doulgeri, and Sotiris Malassiotis. Total singulation with modular reinforcement learning. *IEEE Robotics and Automation Letters*, 6(2):4117–4124, 2021.

[27] Houjian Yu and Changhyun Choi. Self-supervised interactive object segmentation through a singulation-and-grasping approach. In *European Conference on Computer Vision*, pages 621–637. Springer, 2022.

[28] Weihao Yuan, Kaiyu Hang, Danica Kragic, Michael Y Wang, and Johannes A Stork. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems*, 119:119–134, 2019.

[29] Ahmed H Qureshi, Arsalan Mousavian, Chris Paxton, Michael C Yip, and Dieter Fox. Nerp: Neural rearrangement planning for unknown objects. *arXiv preprint arXiv:2106.01352*, 2021.

[30] Ankit Goyal, Arsalan Mousavian, Chris Paxton, Yu-Wei Chao, Brian Okorn, Jia Deng, and Dieter Fox. Ifor: Iterative flow minimization for robotic object rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14787–14797, 2022.

[31] Bingjie Tang and Gaurav S Sukhatme. Selective object rearrangement in clutter. In *Conference on Robot Learning*, pages 1001–1010. PMLR, 2023.

[32] Eric Huang, Zhenzhong Jia, and Matthew T Mason. Large-scale multi-object rearrangement. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 211–218. IEEE, 2019.

[33] Haoran Song, Joshua A Haustein, Weihao Yuan, Kaiyu Hang, Michael Yu Wang, Danica Kragic, and Johannes A Stork. Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9433–9440. IEEE, 2020.

[34] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. IEEE, 2017.

[35] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3339–3346. IEEE, 2018.

[36] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRL*, 12:16, 2017.

[37] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[38] Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *arXiv preprint arXiv:1812.10972*, 2018.

[39] Yufei Ye, Dhiraj Gandhi, Abhinav Gupta, and Shubham Tulsiani. Object-centric forward modeling for model predictive control. In *Conference on Robot Learning*, pages 100–109. PMLR, 2020.

[40] Zhenjia Xu, Zhanpeng He, Jiajun Wu, and Shuran Song. Learning 3d dynamic scene representations for robot manipulation. *arXiv preprint arXiv:2011.01968*, 2020.

[41] Jiayu Wang, Chuxiong Hu, Yunan Wang, and Yu Zhu. Dynamics learning with object-centric interaction networks for robot manipulation. *IEEE Access*, 9:68277–68288, 2021.

[42] Baichuan Huang, Shuai D Han, Abdeslam Boularias, and Jingjin Yu. Dipn: Deep interaction prediction network with application to clutter removal. In *2021 IEEE*

*International Conference on Robotics and Automation (ICRA)*, pages 4694–4701. IEEE, 2021.

[43] Sheng Yu, Di-Hua Zhai, and Yuanqing Xia. A novel robotic pushing and grasping method based on vision transformer and convolution. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[44] Corey Goldfeder, Peter K Allen, Claire Lackner, and Raphael Pelossof. Grasp planning via decomposition trees. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4679–4684. IEEE, 2007.

[45] Kai Huebner, Steffen Ruthotto, and Danica Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. In *2008 IEEE International Conference on Robotics and Automation*, pages 1628–1633. IEEE, 2008.

[46] Clemens Eppner and Oliver Brock. Grasping unknown objects by exploiting shape adaptability and environmental constraints. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4000–4006. IEEE, 2013.

[47] Siddarth Jain and Brenna Argall. Grasp detection for assistive robotic manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2015–2021. IEEE, 2016.

[48] Natsuki Yamanobe and Kazuyuki Nagata. Grasp planning for everyday objects based on primitive shape representation for parallel jaw grippers. In *2010 IEEE International Conference on Robotics and Biomimetics*, pages 1565–1570. IEEE, 2010.

[49] Abhijit Makhal, Federico Thomas, and Alba Perez Gracia. Grasping unknown objects in clutter by superquadric representation. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 292–299. IEEE, 2018.

[50] Yuwei Wu, Weixiao Liu, Zhiyang Liu, and Gregory S Chirikjian. Learning-free grasping of unknown objects using hidden superquadrics. *arXiv preprint arXiv:2305.06591*, 2023.

[51] 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.

[52] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737. IEEE, 2018.

[53] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018.

[54] Takuya Torii and Manabu Hashimoto. Model-less estimation method for robot grasping parameters using 3d shape primitive approximation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 580–585. IEEE, 2018.

[55] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision. *Advances in Neural Information Processing Systems*, 32, 2019.

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

[57] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[58] 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.

[59] Yunzhi Lin, Chao Tang, Fu-Jen Chu, and Patricio A Vela. Using synthetic data and deep networks to recognize primitive shapes for object grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10494–10501. IEEE, 2020.

[60] Marios Kiatos, Sotiris Malassiotis, and Iason Sarantopoulos. A geometric approach for grasping unknown objects with multifingered hands. *IEEE Transactions on Robotics*, 37(3):735–746, 2020.

[61] Terrance E Boult and Ari D Gross. Recovery of superquadrics from depth information. In *Proc. Workshop on Spatial Reasoning and Multi-Sensor Fusion*, pages 128–137, 1987.

[62] Erik Roeland Van Dop and Paul PL Regtien. Fitting undeformed superquadrics to range data: improving model recovery and classification. In *Proceedings. 1998*

*IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*, pages 396–401. IEEE, 1998.

[63] Franc Solina and Ruzena Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE transactions on pattern analysis and machine intelligence*, 12(2):131–147, 1990.

[64] Seungyeon Kim, Taegyun Ahn, Yonghyeon Lee, Jihwan Kim, Michael Yu Wang, and Frank C Park. Dsqnet: A deformable model-based supervised learning algorithm for grasping unknown occluded objects. *IEEE Transactions on Automation Science and Engineering*, 2022.

[65] Seungyeon Kim, Byeongdo Lim, Yonghyeon Lee, and Frank C Park. Se (2)-equivariant pushing dynamics models for tabletop object manipulations. In *Conference on Robot Learning*, pages 427–436. PMLR, 2023.

[66] Seungyeon Kim, Young Hun Kim, Yonghyeon Lee, and Frank C Park. Leveraging 3d reconstruction for mechanical search on cluttered shelves. In *7th Annual Conference on Robot Learning*, 2023.

[67] Alan H Barr. Global and local deformations of solid primitives. In *Readings in Computer Vision*, pages 661–670. Elsevier, 1987.

[68] Yunzhi Lin, Chao Tang, Fu-Jen Chu, Ruinian Xu, and Patricio A Vela. Primitive shape recognition for object grasping. *arXiv preprint arXiv:2201.00956*, 2022.

[69] Wei Gao and Russ Tedrake. kpam-sc: Generalizable manipulation planning using keypoint affordance and shape completion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6527–6533. IEEE, 2021.

[70] Marcus Gualtieri and Robert Platt. Robotic pick-and-place with uncertain object instance segmentation and shape completion. *IEEE robotics and automation letters*, 6(2):1753–1760, 2021.

[71] Nikhil Chavan-Dafle, Sergiy Popovych, Shubham Agrawal, Daniel D Lee, and Volkan Isler. Simultaneous object reconstruction and grasp prediction using a camera-centric object shell representation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1396–1403. IEEE, 2022.

[72] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[73] Sahar El Khoury, Miao Li, and Aude Billard. Bridging the gap: One shot grasp synthesis approach. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2027–2034. IEEE, 2012.

[74] Ana Huamán Quispe, Benoît Milville, Marco A Gutiérrez, Can Erdogan, Mike Stilman, Henrik Christensen, and Heni Ben Amor. Exploiting symmetries and extrusions for grasping household objects. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3702–3708. IEEE, 2015.

[75] Giulia Vezzani, Ugo Pattacini, and Lorenzo Natale. A grasping approach based on superquadric models. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1579–1586. IEEE, 2017.

[76] Giulia Vezzani, Ugo Pattacini, Giulia Pasquale, and Lorenzo Natale. Improving superquadric modeling and grasping with prior on object shapes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6875–6882. IEEE, 2018.

[77] David Schiebener, Andreas Schmidt, Nikolaus Vahrenkamp, and Tamim Asfour. Heuristic 3d object shape completion based on symmetry and scene context. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 74–81. IEEE, 2016.

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

[79] Xinchen Yan, Jasmined Hsu, Mohammad Khansari, Yunfei Bai, Arkanath Pathak, Abhinav Gupta, James Davidson, and Honglak Lee. Learning 6-dof grasping interaction via deep geometry-aware 3d representations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3766–3773. IEEE, 2018.

[80] Jens Lundell, Enric Corona, Tran Nguyen Le, Francesco Verdoja, Philippe Weinzaepfel, Grégory Rogez, Francesc Moreno-Noguer, and Ville Kyrki. Multi-fingan: Generative coarse-to-fine sampling of multi-finger grasps. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4495–4501. IEEE, 2021.

[81] Jens Lundell, Francesco Verdoja, and Ville Kyrki. Ddgc: Generative deep dexterous grasping in clutter. *IEEE Robotics and Automation Letters*, 6(4):6899–6906, 2021.

[82] Zhenyu Jiang, Yifeng Zhu, Maxwell Svetlik, Kuan Fang, and Yuke Zhu. Synergies between affordance and geometry: 6-dof grasp detection via implicit representations. *arXiv preprint arXiv:2104.01542*, 2021.

[83] Daniel Yang, Tarik Tosun, Benjamin Eisner, Volkan Isler, and Daniel Lee. Robotic grasping through combined image-based grasp proposal and 3d reconstruction. In

*2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6350–6356. IEEE, 2021.

[84] Ari D Gross and Terrance E Boult. Error of fit measures for recovering parametric solids. In *ICCV*, 1988.

[85] I-Ming Chen and Joel W Burdick. Finding antipodal point grasps on irregularly shaped objects. *IEEE transactions on Robotics and Automation*, 9(4):507–512, 1993.

[86] Joseph O'Rourke. Finding minimal enclosing boxes. *International journal of computer & information sciences*, 14(3):183–199, 1985.

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

[88] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003.

[89] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[90] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

[91] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *kdd*, 96(34):226–231, 1996.

[92] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[93] Yang Yang, Hengyue Liang, and Changhyun Choi. A deep learning approach to grasping the invisible. *IEEE Robotics and Automation Letters*, 5(2):2232–2239, 2020.

[94] Michael Danielczuk, Anelia Angelova, Vincent Vanhoucke, and Ken Goldberg. X-ray: Mechanical search for an occluded object by minimizing support of learned occupancy distributions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9577–9584. IEEE, 2020.

[95] Marwan Qaid Mohammed, Lee Chung Kwek, Shing Chyi Chua, Arafat Al-Dhaqm, Saeid Nahavandi, Taiseer Abdalla Elfadil Eisa, Muhammad Fahmi Miskon, Mohammed Nasser Al-Mhiqani, Abdulalem Ali, Mohammed Abaker, et al. Review of learning-based robotic manipulation in cluttered environments. *Sensors*, 22(20):7938, 2022.

[96] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[97] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free

fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[98] Michael Strecke and Joerg Stueckler. Diffsdfsim: Differentiable rigid-body dynamics with implicit shapes. In *2021 International Conference on 3D Vision (3DV)*, pages 96–105. IEEE, 2021.

[99] Danny Driess, Jung-Su Ha, Marc Toussaint, and Russ Tedrake. Learning models as functionals of signed-distance fields for manipulation planning. In *Conference on Robot Learning*, pages 245–255. PMLR, 2022.

[100] Youngsun Wi, Pete Florence, Andy Zeng, and Nima Fazeli. Virdo: Visio-tactile implicit representations of deformable objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3583–3590. IEEE, 2022.

[101] Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint. Learning multi-object dynamics with compositional neural radiance fields. In *Conference on Robot Learning*, pages 1755–1768. PMLR, 2023.

[102] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10344–10353, 2019.

[103] Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S Chirikjian. Robust and accurate superquadric recovery: a probabilistic approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2676–2685, 2022.

[104] Yuwei Wu, Weixiao Liu, Sipu Ruan, and Gregory S Chirikjian. Primitive-based shape abstraction via nonparametric bayesian inference. In *European Conference on Computer Vision*, pages 479–495. Springer, 2022.

[105] Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S Chirikjian. Marching-primitives: Shape abstraction from signed distance function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8771–8780, 2023.

[106] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[107] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[108] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[109] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[110] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[111] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[112] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018.

[113] Taco S Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant cnns on homogeneous spaces. *Advances in neural information processing systems*, 32, 2019.

[114] Elise van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4199–4210, 2020.

[115] Dian Wang, Robin Walters, Xupeng Zhu, and Robert Platt. Equivariant $q$ learning in spatial action spaces. In *Conference on Robot Learning*, pages 1713–1723. PMLR, 2022.

[116] Haojie Huang, Dian Wang, Robin Walters, and Robert Platt. Equivariant transporter network. *arXiv preprint arXiv:2202.09400*, 2022.

[117] Dian Wang, Mingxi Jia, Xupeng Zhu, Robin Walters, and Robert Platt. On-robot learning with equivariant models. *arXiv preprint arXiv:2203.04923*, 2022.

[118] Colin Kohler, Anuj Shrivatsav Srikanth, Eshan Arora, and Robert Platt. Symmetric models for visual force policy learning. *arXiv preprint arXiv:2308.14670*, 2023.

[119] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields:

Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400. IEEE, 2022.

[120] Hyunwoo Ryu, Hong-in Lee, Jeong-Hoon Lee, and Jongeun Choi. Equivariant descriptor fields: Se (3)-equivariant energy-based models for end-to-end visual robotic manipulation learning. *arXiv preprint arXiv:2206.08321*, 2022.

[121] Anthony Simeonov, Yilun Du, Yen-Chen Lin, Alberto Rodriguez Garcia, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Pulkit Agrawal. Se (3)-equivariant relational rearrangement with neural descriptor fields. In *Conference on Robot Learning*, pages 835–846. PMLR, 2023.

[122] Zhengrong Xue, Zhecheng Yuan, Jiashun Wang, Xueqian Wang, Yang Gao, and Huazhe Xu. Useek: Unsupervised se (3)-equivariant 3d keypoints for generalizable manipulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1715–1722. IEEE, 2023.

[123] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022.

[124] Yonghyeon Lee, Jonghyuk Baek, Young Min Kim, and Frank Chongwoo Park. Imat: The iterative medial axis transform. *Computer Graphics Forum*, 40(6):162–181, 2021.

[125] Yonghyeon Lee, Seungyeon Kim, Jinwon Choi, and Frank Park. A statistical manifold framework for point cloud data. In *International Conference on Machine Learning*, pages 12378–12402. PMLR, 2022.

[126] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019.

[127] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[128] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 37(5):1343–1359, 2021.

[129] Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Manipulation-based active search for occluded objects. In *2013 IEEE International Conference on Robotics and Automation*, pages 2814–2819. IEEE, 2013.

[130] Megha Gupta, Thomas Rühr, Michael Beetz, and Gaurav S Sukhatme. Interactive environment exploration in clutter. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5265–5272. IEEE, 2013.

[131] Mehmet R Dogar, Michael C Koval, Abhijeet Tallavajhula, and Siddhartha S Srinivasa. Object search by manipulation. *Autonomous Robots*, 36:153–167, 2014.

[132] Jue Kun Li, David Hsu, and Wee Sun Lee. Act to see and see to act: Pomdp planning for objects search in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5701–5707. IEEE, 2016.

[133] Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen, and Christopher Amato. Online planning for target object search in clutter under partial observability. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8241–8247. IEEE, 2019.

[134] Michael Danielczuk, Andrey Kurenkov, Ashwin Balakrishna, Matthew Matl, David Wang, Roberto Martín-Martín, Animesh Garg, Silvio Savarese, and Ken Goldberg. Mechanical search: Multi-step retrieval of a target object occluded by clutter. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1614–1621. IEEE, 2019.

[135] Andrew Price, Linyi Jin, and Dmitry Berenson. Inferring occluded geometry improves performance when retrieving an object from dense clutter. In *Robotics Research: The 19th International Symposium ISRR*, pages 376–392. Springer, 2022.

[136] Jinhwi Lee, Younggil Cho, Changjoo Nam, Jonghyeon Park, and Changhwan Kim. Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 183–189. IEEE, 2019.

[137] Jinhwi Lee, Changjoo Nam, Jonghyeon Park, and Changhwan Kim. Tree search-based task and motion planning with prehensile and non-prehensile manipulation for obstacle rearrangement in clutter. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8516–8522. IEEE, 2021.

[138] Ewerton R Vieira, Daniel Nakhimovich, Kai Gao, Rui Wang, Jingjin Yu, and Kostas E Bekris. Persistent homology for effective non-prehensile manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1918–1924. IEEE, 2022.

[139] Sangbeom Park, Yoonbyung Chai, Sunghyun Park, Jeongeun Park, Kyungjae Lee, and Sungjoon Choi. Semi-autonomous teleoperation via learning non-prehensile manipulation skills. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9295–9301. IEEE, 2022.

[140] Changjoo Nam, Jinhwi Lee, Younggil Cho, Jeongho Lee, Dong Hwan Kim, and ChangHwan Kim. Planning for target retrieval using a robotic manipulator in cluttered and occluded environments. *arXiv preprint arXiv:1907.03956*, 2019.

[141] Changjoo Nam, Jinhwi Lee, Sang Hun Cheong, Brian Y Cho, and ChangHwan Kim. Fast and resilient manipulation planning for target retrieval in clutter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3777–3783. IEEE, 2020.

[142] Changjoo Nam, Sang Hun Cheong, Jinhwi Lee, Dong Hwan Kim, and ChangHwan Kim. Fast and resilient manipulation planning for object retrieval in cluttered and confined environments. *IEEE Transactions on Robotics*, 37(5):1539–1552, 2021.

[143] Huang Huang, Marcus Dominguez-Kuhne, Vishal Satish, Michael Danielczuk, Kate Sanders, Jeffrey Ichnowski, Andrew Lee, Anelia Angelova, Vincent Vanhoucke, and Ken Goldberg. Mechanical search on shelves using lateral access x-ray. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2045–2052. IEEE, 2021.

[144] Huang Huang, Michael Danielczuk, Chung Min Kim, Letian Fu, Zachary Tam, Jeffrey Ichnowski, Anelia Angelova, Brian Ichter, and Ken Goldberg. Mechanical search on shelves using a novel “bluction” tool. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6158–6164. IEEE, 2022.

[145] Huang Huang, Letian Fu, Michael Danielczuk, Chung Min Kim, Zachary Tam, Jeffrey Ichnowski, Anelia Angelova, Brian Ichter, and Ken Goldberg. Mechanical search on shelves with efficient stacking and destacking of objects. In *Robotics Research*, pages 205–221. Springer, 2023.

[146] Yiming Ye and John K Tsotsos. Sensor planning for 3d object search. *Computer Vision and Image Understanding*, 73(2):145–168, 1999.

[147] Kristoffer Sjö, Dorian Gálvez López, Chandana Paul, Patric Jensfelt, and Danica Kragic. Object search and localization for an indoor mobile robot. *Journal of Computing and Information Technology*, 17(1):67–80, 2009.

[148] Thomas Kollar and Nicholas Roy. Utilizing object-object and object-scene context when planning to find things. In *2009 IEEE International Conference on Robotics and Automation*, pages 2168–2173. IEEE, 2009.

[149] Jeremy Ma, Timothy H Chung, and Joel Burdick. A probabilistic framework for object search with 6-dof pose estimation. *The International Journal of Robotics Research*, 30(10):1209–1228, 2011.

[150] Alper Aydemir, Kristoffer Sjöö, John Folkesson, Andrzej Pronobis, and Patric Jensfelt. Search in the real world: Active visual object search based on spatial relations. In *2011 IEEE International Conference on Robotics and Automation*, pages 2818–2824. IEEE, 2011.

[151] Marc Hanheide, Charles Gretton, Richard Dearden, Nick Hawes, Jeremy Wyatt, Andrzej Pronobis, Alper Aydemir, Moritz Göbelbecker, and Hendrik Zender. Exploiting probabilistic knowledge under uncertain sensing for efficient robot behaviour. In *IJCAI*, pages 2442–2449, 2011.

[152] Alper Aydemir, Moritz Göbelbecker, Andrzej Pronobis, Kristoffer Sjöö, and Patric Jensfelt. Plan-based object search and exploration using semantic spatial knowledge in the real world. In *ECMR*, pages 13–18. Citeseer, 2011.

[153] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2616–2625, 2017.

[154] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

[155] Arsalan Mousavian, Alexander Toshev, Marek Fišer, Jana Košecká, Ayzaan Wahid, and James Davidson. Visual representations for semantic target driven navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8846–8852. IEEE, 2019.

[156] Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33:4247–4258, 2020.

[157] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[158] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *Conference on Robot Learning*, pages 967–977. PMLR, 2023.

[159] Changkyu Song and Abdeslam Boularias. Learning to slide unknown objects with differentiable physics simulations. *arXiv preprint arXiv:2005.05456*, 2020.

[160] Félix Nadon, Angel J Valencia, and Pierre Payeur. Multi-modal sensing and robotic manipulation of non-rigid objects: A survey. *Robotics*, 7(4):74, 2018.

[161] Hang Yin, Anastasia Varava, and Danica Kragic. Modeling, learning, perception, and control methods for deformable object manipulation. *Science Robotics*, 6(54):eabd8803, 2021.

[162] Harshit Khurana, Michael Bombile, and Aude Billard. Learning to hit: A statistical dynamical system based approach. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9415–9421. IEEE, 2021.

[163] Harshit Khurana and Aude Billard. Motion planning and inertia-based control for impact aware manipulation. *IEEE Transactions on Robotics*, 2023.

[164] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[165] Hao Li, Yizhi Zhang, Junzhe Zhu, Shaoxiong Wang, Michelle A Lee, Huazhe Xu, Edward Adelson, Li Fei-Fei, Ruohan Gao, and Jiajun Wu. See, hear, and feel: Smart sensory fusion for robotic manipulation. *arXiv preprint arXiv:2212.03858*, 2022.

[166] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[167] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[168] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[169] Despoina Paschalidou, Luc Van Gool, and Andreas Geiger. Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1060–1070, 2020.

[170] Tim Oblak, Jaka Šircelj, Vitomir Štruc, Peter Peer, Franc Solina, and Aleš Jaklič. Learning to predict superquadric parameters from depth images with explicit and implicit supervision. *IEEE Access*, 9:1087–1102, 2020.

[171] Matheus Gadelha, Rui Wang, and Subhransu Maji. Shape reconstruction using differentiable projections and deep priors. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22–30, 2019.

# 국문초록

비전 기반의 물체 조작은 로봇 공학에서 중요한 연구 분야로 부상하였으며, 물체를 파지하거나, 물체를 밀거나, 복잡한 환경 속에서 물체를 재배열하는 등의 작업을 포함한다. 최근 기술의 발전으로 로봇은 비전 센서를 활용하여 주변 물체들이나 환경과 상호작용을 할 수 있게 되었다. 그러나 이러한 비전 기반 물체 조작 기법들은 여전히 해결해야 할 많은 도전 과제들을 가지고 있다. 딥 러닝 기반 방식에서 나타나는 데이터의 비효율성부터 실제 환경에서 실용적으로 적용하기 위해 해결해야 할 다양한 제약 사항까지, 이러한 도전 과제들을 해결하는 것은 로봇의 물체 조작 응용 범위를 확장하는 데 매우 중요하다.

본 학위 논문에서는 형상 인식 기법을 통해 비전 기반 물체 조작과 관련된 많은 도전 과제를 완화할 수 있음을 입증한다. 이 접근 방식은 비전 데이터로부터 물체의 3D 형상을 인식하고, 이를 기반으로 로봇이 취해야할 행동 (예: 물체 파지 및 또는 물체 밀기)을 생성한다. 이러한 방법은 데이터의 비효율성을 해결할 뿐만 아니라 정확성과 효율성을 희생하지 않고도 광범위한 데이터 수집의 필요성을 줄인다. 본 논문에서는 3D 형상 인식이 물체 조작과 관련된 수많은 도전 과제를 효과적으로 해결함을 발견했으며, 특히 실제 환경에서의 실용적인 적용에서 다양한 어려운 비전 기반 물체 조작 문제를 현저하게 간소화하고 개선하는 데 크게 기여한다. 이렇듯 본 학위 논문에서는 형상 인식 기술과 비전 기반 물체 조작을 통합함으로써 로봇의 환경과의 상호 작용 능력을 상당히 향상시킬 수 있다는 점을 강조한다. 본 학위 논문의 주요 기여는 형상 인식 기반 물체 조작 기법들을 소개하는 데 있다.

첫째로, 본 논문에서는 새로운 형상 인식 기법을 기반으로 한 파지 기법을 소개한다. 이 방법은 심층 학습 네트워크와 함께 변형 가능한 슈퍼쿼드릭이라고 하는 다양한 형상 템플릿을 통합하여 DSQNet (Deformable SuperQuadric Network) 이라고 하는 새로운 모델을 제안한다. 이 모델은 부분적인 포인트 클라우드 데이터에서 변형 가능한 슈퍼쿼드릭을 추론하여 완전한 물체 형상을 식별하는 데 사용된다. 지도 학습을 통해

DSQNet은 변형 가능한 슈퍼쿼드릭의 여덟 가지 매개변수와 자세를 생성하고, 가려진 부분까지 정확하게 고려하여 전체 물체 형상과 일치시킨다. 파지를 위한 후속 전략은 그리퍼의 운동학적 및 구조적 특징과 더불어, 변형 가능한 슈퍼쿼드릭과 이것의 폐형식 방정식을 활용한다. 비교 분석 결과, DSQNet은 정확성과 속도 측면에서 기존의 형상 인식 방법들을 능가하는 것으로 나타났다. 특히 이 형상 인식 기반 방법은 정교한 형상 인식 능력 덕분에 기존 기법을 뛰어넘는 파지 성공률을 기록했다.

다음으로, 형상 인식의 장점을 활용하여 밀기 역학 모델의 학습을 용이하게 한다. 우선, 테이블 위에 놓인 물체들의 형상을 또한 슈퍼쿼드릭을 사용해 인식한다. 형상 인식과 밀기 역학 모델 학습을 통합하는 것의 내재적인 장점은 SE(2)-등변 밀기 역학 모델을 자연스럽게 정의할 수 있는 능력이다. 그리고 이는 SQPDNet (SuperQuadric Pushing Dynamics Network) 라는 SE(2)-등변 밀기 역학 모델을 개발하는 데 기반이 된다. 이 접근 방식은 물리적 시스템 내의 대칭성을 내재적으로 인식하여 역학 모델의 일반화 성능을 상당히 향상시킨다. 비교 평가 결과, 이 형상 인식 기반 모델이 기존의 비전 기반 밀기 역학 모델을 능가하는 것으로 나타났는데, 특히 이는 추가로 부여된 SE(2)-등변성 때문이라고 본 논문에서 주장한다. 더불어, 이 모델의 효과성은 시뮬레이션 및 실제 실험을 통해 검증된 다양한 물체 밀기 조작 작업에서 모델 기반 최적 제어에 대한 응용을 통해 추가로 입증되었다.

마지막으로, 형상 인식 방법을 사용한 복잡한 선반에서의 기계적 탐색 작업을 다룬다. 이 작업은 지정된 특정 대상 물체를 복잡한 선반 환경 안에서 찾고 파지하는 것을 포함하며, 비전 센서가 특정 대상 물체를 처음에 감지하지 못하는 경우도 포함된다. 이러한 상황에서 로봇의 임무는 주변 물체를 전략적으로 재배열하여 대상 물체의 위치를 확인하고, 동시에 선반과 주변 물체와의 충돌을 피하는 것이다. 이러한 도전에 대처하기 위해 이 작업에서도 슈퍼쿼드릭 형상 인식 모델을 활용한다. 이 모델은 본 학위 논문에서 개발된 형상 인식 기반 물체 조작 기술을 사용 가능하게 하며, 가려진 대상 물체의 잠재적인 자세에 대한 신속하고 효율적인 추론을 가능하게 한다. 이를

통해 로봇은 대상 물체를 효과적으로 안전하게 찾아 파지 할 수 있다. 이 방법은 시뮬레이션 및 실제 세계에서 일반적인 로봇 그리퍼를 사용하여 대상 물체를 찾고 붙잡는 데 성공했다.

**주요어:** 비전 기반 물체 조작, 형상 인식, 로봇 물체 파지, 밀기 동작, 밀기 역학 학습, 기계적 탐색, 물체 재정렬.

**학번:** 2019-39029