Task-specific Object Grasps using Primitive Shapes and Symbolic Reasoning

S.F. Zwinkels

TU Delft

Task-specific Object Grasps using Primitive Shapes and Symbolic Reasoning

by

S. F. Zwinkels

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday March 24, 2023 at 13:00.

Student number: 4630726

Thesis committee:	Dr. ir. C. Hernandez Corbato	TU Delft, daily supervisor
	Dr. ir. J. Kober	TU Delft, external committee member
	Dr. A. Gabriel	TU Delft, external committee member

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

In household and retail store environments, humans efficiently identify grasp regions of an object to perform everyday tasks. To enable robots to understand the requirements of a task and the properties of an object, a novel grasp framework, called the Shape Primitive and Reasoning Grasping Engine (SPaRGE), is proposed in this thesis. SPaRGE combines a data-driven approach, a superquadric algorithm, and a logic-based approach to find task-specific object grasp poses from a partial point cloud. Multiple grasp poses are recovered using the data-driven approach, while objects are represented as multiple primitive shapes using the superquadric algorithm and logic-based approach. Logic rules are used to decide which primitive shape meets the criteria of a given task, resulting in a primitive shape indicating a grasp region. The data-driven approach then finds a grasp pose that meets the task requirements. Two versions of the logic-based approach are presented, differing in their design choices that affect the generalizability and scalability of the logic rules. The SPaRGE framework is evaluated on common household and retail store objects, demonstrating an average true positive rate of 81.2% and 86.0%, respectively, in enabling a robot to grasp objects at a desired region. Results show that the SPaRGE framework enables real-time object grasping without prior knowledge of the objects. Further experiments show that the SPaRGE framework can be adapted successfully for real-world object grasping scenarios. The proposed framework provides a significant contribution toward enabling robots to perform human-like grasping in real-world environments.

Preface

This thesis concludes my master's degree in Robotics at Delft University of Technology. Writing a master's thesis was a challenging process that taught me self-discipline, perseverance, and resilience. I take this opportunity to express my gratitude to those who supported me in completing this thesis. Their encouragement, guidance, and invaluable assistance made this endeavor possible.

First and foremost, I would like to thank my supervisor, Dr. Carlos Hernández Corbato. Your support has been invaluable to me and the success of my thesis. Your constructive criticism and feedback helped me grow and improve as a person. During our bi-weekly meetings, your patience and dedication helped me understand and learn from my mistakes. I would also like to thank the people from the KAS group, who provided me with new insights and interesting ideas during our bi-weekly meetings.

And, of course, I am grateful to my family and friends for their unwavering support. Their encouragement and motivation kept me going during the challenging moments of this academic journey. Lastly, I would like to express my sincere appreciation to my friend David, whose discipline and strength inspired me throughout all my years of study.

S. F. Zwinkels
Delft, March 2023

Contents

List of Figures

List of Tables

Nomenclature

Abbrevations

Adam	Adaptive Moment Estimation
DOF	Degrees of freedom
EMS Algorithm	Expectation, Maximization, and Switch algorithm
FOPL	First-order Predicate Logic
GPD	Grasp Pose Detection
MLP	Multi-Layer Perceptron
OMPL	Open Motion Planning Library
Poly	Polynomial kernel
RBF	Radial Basis Function kernel
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
SGD	Stochastic Gradient Descent
SPARGE	Shape Primitive and Reasoning Grasping Engine
SVM	Support Vector Machines
TPE-sampler	Tree-structured Parzen Estimator sampler

Symbols

$\boldsymbol{\alpha}$	Primitive size represented by $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \alpha_3\}$
$\boldsymbol{\epsilon}$	Primitive shape represented by $\boldsymbol{\epsilon} = \{\epsilon_1, \epsilon_2\}$
$\boldsymbol{\Theta}$	Set of multiple recovered primitives $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \dots \boldsymbol{\theta}_i\}$
$\boldsymbol{\theta}$	Primitive shape represented by $\{\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, \alpha_3, t_x, t_y, t_z, q_i, q_j, q_k, q_w\}$
$\boldsymbol{d}'$	Expected spatial distance between two primitives
$\boldsymbol{d}$	Spatial distance between two primitives
$\boldsymbol{P}$	Point cloud that is represented by a set of n points $\boldsymbol{P} = \{p_1, p_2, p_3, \dots p_n\}$
${}^{k}\boldsymbol{q}_i$	quaternion $\{q_x, q_y, q_z, q_w\}$ of a primitive i in the coordinate frame k
${}^{k}\boldsymbol{t}_i$	translation $\{t_x, t_y, t_z\}$ of a primitive i in the coordinate frame k

1 Introduction

Humans thrive at performing complex tasks, allowing them to quickly and accurately identify, grasp, and manipulate everyday objects. This involves deciding where to pick up an object, which allows us to firmly grasp the object and carry out a given task, such as cooking or hammering. Even though these tasks may appear simple for humans, it has been a critical challenge to design robots that can perform these tasks at the same level as humans [32]. To comprehend task requirements, a robot needs to accurately identify objects, identify their properties, and understand how to interact with them. This knowledge is essential to grasp and manipulate objects, since not all grasps on an object result in a suitable grasp for every task. For example, tools usually need to be grasped by their handles to accomplish a given task.

Robots can play an important role in making people's daily lives more convenient. As more and more young people strive for higher education, there is an increasing need for robots to fill the gaps left by traditional low-skill labor [2]. Robots can be employed to carry out menial tasks in fields ranging from households to retail stores, freeing up people to pursue more fulfilling and meaningful activities. Recent advances in robot technology have enabled robots to increase proficiency at performing task-specific grasps in household and retail store environments [7], [24]. This enabled robots to complete more complex tasks, such as assembling components and manipulating objects with greater accuracy and speed [7]. However, these frameworks are not good enough yet to be deployed in household and retail stores, as they are often limited to grasping known objects using hard-coded grasp poses [24], [55], or only focus on finding robust grasp poses [43]. A grasp pose is a configuration of a robot's end-effector that indicates the position and orientation that will enable the robot to grasp an object successfully.

To advance the capabilities of robots and ensure their dependable functioning in real-world scenarios, a robot must be able to include task-specific requirements, allowing it to reason about where known and unknown objects should be grasped. Reasoning allows a robot to make appropriate decisions based on the available information. In the case of performing a grasp, the robot can find a grasp pose that successfully performs a task, such as hammering, pouring, or cooking.

1.1. Motivation

To replace humans in daily situations, such as in a household or retail store, a robot should be able to perform complex tasks such as cooking food with a pan, or hammering a nail. To replicate this human behavior, a robot must understand the relationship between a task and an object. Therefore, many researchers attempted to create robotic grasp frameworks that have the ability to include "*semantic information*", which is useful information that can help a robot to understand how to complete a task [38], [45], [49]. For example, object properties, spatial context, and task description provide meaningful semantic information that influences the outcome of a desired grasp pose. However, existing frameworks are often limited in either their ability to generalize to unknown objects [3], to reason over the ideal grasp pose for a specific task [43], or depend on prior object information (e.g., complete 3D models) [9]. This makes these existing frameworks unable to be used in environments that require a robot to perform many tasks on many unknown objects, such as in a household or retail store environment. A household

environment contains diverse tasks and objects typically placed in random orientations, while a retail store contains many objects with different shapes and sizes. Therefore, a grasp framework that can be used in both environments likely generalizes to other grasp environments such as restaurants or healthcare. The desired grasp framework should be capable of adapting its grasp pose depending on the task requirements, and it should understand the relationship between object properties and be able to grasp many unique objects accurately.

Existing grasp frameworks can be divided into three approaches; analytical, data-driven, and logic-based. *Analytical approaches* require complete object models and fail to include the task to accomplish or to understand semantic information while finding a grasp pose [20], [46], [49]. *Data-driven approaches* can include both the task to accomplish and semantic information but require much data to learn suitable grasp poses [32], [39], [47]. *Logic-based approaches* offer transparency and can reason over many types of semantic information but cannot generate grasp poses from images or point clouds. Therefore, logic-based approaches are not suited to recover grasp poses on unknown objects [29], [30], [53], [57]. In Chapter 2, we will discuss the existing grasp frameworks and their limitations in greater detail.

Despite the extensive research into these individual approaches, there has been little to no research into the combination of using data-driven approaches with logic-based approaches. This combination could enable the proposal of real-time grasp poses on various known/unknown objects using the data-driven approach, whilst the addition of the logic-based approach would provide the robot with the knowledge and reasoning capabilities to determine which proposed grasp pose will lead to successfully performing a task. ***Therefore, this research hypothesizes that by combining the data-driven approach and the logic-based approach, a framework can be developed to create suitable grasp poses for performing tasks in household and retail store environments.***

In this research, we propose to use an existing data-driven approach to generate multiple grasp poses on objects from a partial point cloud [43]. The point cloud is partial as it is obtained by a depth sensor that captures only the visible part of the scene from a certain viewpoint. To find a grasp pose that satisfies the requirements of a task, we use a logic-based approach. Logic-based approaches use *symbolic reasoning*, which is based on a set of facts and logic rules that use semantic information to describe the environment and the robot's capabilities. Because logic-based approaches cannot extract semantic information from images or point clouds, an intermediate step is required. Therefore, we use the state-of-the-art superquadric algorithm that recovers multiple *primitive shapes* (e.g., cuboids, cylinders, and spheres) from a partial object point cloud [31]. This provides the logic-based approach with semantic information to decide where an object should be grasped. The proposed grasp framework is called Shape Primitive and Reasoning Grasping Engine (SPaRGE), which is a combination of the data-driven approach, superquadric algorithm, and the logic-based approach (see Figure 1.1).

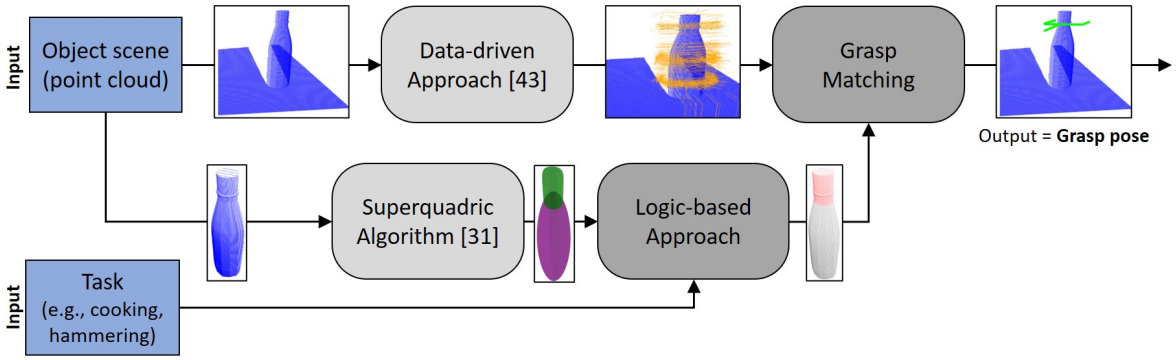

Figure 1.1: Overview of the SPaRGE framework to perform task-specific object grasps. The framework uses an existing data-driven approach to recover multiple grasp poses [43]. In addition, it uses a state-of-the-art superquadric algorithm to recover multiple primitive shapes [31], which is used by the logic-based approach to find a primitive shape that meets the requirements of a given task. Finally, a suitable grasp pose is found by combining the data-driven and logic-based approaches through grasp matching.

1.2. Research Objective

The objective of this research is to combine an existing data-driven approach with a superquadric algorithm and a logic-based approach that uses symbolic reasoning to grasp everyday objects and perform tasks in household and retail store environments. The main focus of this research will be on the design of the logic-based approach to finding task-specific grasp regions. Therefore, the research question in this study is as follows:

> "How can the recovery of primitive shapes with symbolic reasoning allow for task-specific grasps on everyday objects, using a partial point cloud?"

The main research question is divided into five subquestions, whose answers contribute to answering the main research question.

1. *How can a superquadric algorithm provide semantic information on everyday household and retail store objects?*
2. *How to interpret semantic information from primitive shapes using symbolic reasoning?*
3. *How effective is the SPaRGE framework in detecting task-specific grasp regions on objects?*
4. *How effective does the SPaRGE framework generalize to unknown objects?*
5. *How can the SPaRGE framework be extended to new semantic information and tasks?*

1.3. Requirements

In this section, the requirements for the SPaRGE framework, to propose a task-specific grasp pose in household and retail store environments are defined. The requirements descend from the hypothesis (R1), the robotic application (R2), and the working environment (R3, R4, and R5).

R1: *Ability to reason over semantic information.* Being able to reason over semantic information should allow a grasp framework to make informed decisions on where to grasp an object, resulting in a suitable grasp region. This would allow the framework to perform tasks in a household and retail store environment that require specific grasp poses.

R2: *Be applicable to a partial point cloud.* Being able to recover suitable grasp poses from a partial point cloud enables a robot to be more efficient and cost-effective. In addition, it enables a robot to grasp objects in difficult environments where the complete object cannot be perceived, such as in a retail store environment (e.g., shelves, bins, trays).

R3: *Be able to grasp objects from multiple orientations.* Being able to recognize suitable grasp poses on objects despite their orientation improves the framework's flexibility in scenarios where objects are perceived from different perspectives.

R4: *Generalize to unknown objects.* Being able to generalize to unknown objects allows the grasp framework to adapt to new objects. This would enable the grasp framework to interact with objects it has not seen before.

R5: *Extend to new tasks and semantic information.* The framework should be able to extend its reasoning capabilities if either new tasks are added, or more semantic information becomes available.

1.4. Contribution

The main contribution of this thesis is the development of the SPaRGE framework, which combines a data-driven approach, a superquadric algorithm, and a logic-based approach to identify task-specific grasp poses on everyday objects. The SPaRGE framework is capable of providing suitable grasp poses for performing tasks in household and retail store environments, both on known and unknown objects. This framework's unique feature is its reasoning capability to identify a grasp region that will lead to a successful task-specific grasp pose by incorporating the task's requirements.

To support the SPaRGE framework, several additional novel contributions are made.

- For the logic-based approach, two novel symbolic reasoning modules are designed to identify a primitive shape that results in a successful grasp region; reasoning Version-1 and Version-2. Each version features a unique reasoning approach using logic rules. Reasoning Version-1 uses the concept of task-specific logic rules, and reasoning Version-2 uses the concept of part affordance in combination with an OWL ontology to store relations between tasks and affordance labels.
- Hyperparameter tuning of the state-of-the-art superquadric algorithm to enhance the accuracy of the primitive shapes' resemblance with the original object's dimension and shape (see Appendix A.1.1). To achieve this, we have developed a novel cost function that considers the shape and size of the primitive shapes.
- A classification algorithm is made which classifies primitive shapes into cuboids, cylinders, or spheres. It uses a Multi-Layer Perceptron (MLP) that is trained on 550 manually labeled primitive shapes (see Appendix A.1.2). The contribution is the comparison between different machine learning algorithms to classify primitive shapes into their respective labels and the creation of a dataset with labeled primitive shapes.

1.5. Thesis Outline

The thesis is structured as follows. We discuss the work on existing grasp frameworks in Chapter 2. Then, in Chapter 3, we provide the preliminaries needed to understand the SPaRGE framework. In particular, we discuss the state-of-the-art superquadric algorithm. Chapter 4 discusses additional contributions for the superquadric algorithm. In Chapter 5, we give a detailed overview of our proposed SPaRGE framework and the two reasoning versions that use symbolic reasoning. Chapter 6 present the experiments demonstrating the performance of the SPaRGE framework. In Chapter 7, we answer the subquestions and provide an analysis of the limitations of the proposed framework. Finally, we conclude our work and provide suggestions for future research directions in Chapter 8.

2

Related Work

In this chapter, we review the existing grasp frameworks that include the requirements of a task while finding a grasp pose on an object. These task-specific grasp frameworks are equipped with the ability to reason over semantic information, such as object properties (e.g., size, weight, material) and context information to adapt their grasp pose (see Section 2.1). In particular, we discuss the advantages and limitations of data-driven and logic-based approaches used in the SPaRGE framework. In addition, we discuss various shape completion techniques that can provide the logic-based approach with semantic information. We elaborate on the advantages and disadvantages of existing shape-completion techniques (see Section 2.2).

2.1. Task-specific Grasp Frameworks

As introduced in Section 1.1, many researchers have attempted to develop grasp frameworks with the capability to reason over semantic information and account for the requirements of a task. These frameworks can be divided into three distinct approaches; analytical, data-driven, and logic-based. In this section, we explore how each of these approaches reason over semantic reasoning and generates task-specific object grasps. We focus on grasp frameworks that adjust their grasp poses according to the properties of an object or the specific requirements of a task.

2.1.1. Analytical Approaches

Analytic approaches evaluate contact points on the surface of a 3D object model to construct a robust grasp [15]. A robust grasp is a grasp in which a robot seeks a number of contact points on the object's surface that ensures that no slip will occur when the object is grasped. As analytic methods use a heuristic, or dynamic model to find stable grasps, only a few grasp frameworks have considered adjusting their model to include task-specific constraints [8], [20]. This is mainly due to the difficulties of modeling a task and providing criteria to compare the suitability of different grasps and the task requirements [49]. In Prats *et al.* [46], a framework was proposed that takes an object model and the task to be performed as input, and then selects a predefined grasp pose that meets the requirements of a contact model. However, this approach only focuses on the grasp pose, without reasoning at which location an object should be grasped. In addition, including the requirements of a task into the dynamic model is difficult and only works well for a small set of tasks and objects [46]. Therefore, analytic approaches mainly focus on finding a robust grasp, without considering the task to accomplish. In addition, analytical approaches require a complete object model, which is difficult to obtain and not suited to grasp objects in partially visible scenes [46]. Therefore, analytic approaches are not suited for our application to perform task-specific object grasps in a household or retail store environment.

2.1.2. Data-driven Approaches

Data-driven approaches sample grasp poses on objects in scene images or partial point clouds and rank them based on existing grasp experiences [33], [43], [47]. This grasp experience is learned by using machine learning algorithms, such as supervised learning [43]. These methods use sensor data to generate a grasp pose that enables a robot to grasp partially visible objects. Grasp poses are possible

configurations of the robot's end-effector which could be used to create a robust grasp on an object. Moreover, they can learn from past experiences and adapt to varying objects and situations without requiring a contact model or a comprehensive object model, making them more suitable at grasping objects in complex environments [28].

In Mahler *et al.* [33] and Pas *et al.* [43], the authors developed a Neural Network that utilizes supervised learning to predict the quality of grasp poses accurately. These models are trained with a large labeled dataset of robust exemplar grasps. The result is a grasp framework that can propose grasp poses on objects using a scene point cloud. The advantage of both grasp frameworks is their robustness to grasp objects in cluttered environments, and their ability to generalize to unknown objects. The term *unknown* object refers to objects that have not been encountered before, for which no complete object point cloud data is available, and which were not included in the training set. In addition, Pas *et al.* [43] has the advantage of proposing 6-degree of freedom (6-DOF) grasp poses from a scene point cloud, enabling it to find a grasp from any orientation around the object. However, since these "robust" grasp frameworks are trained on a labeled dataset that only contains robust grasp poses, they fail to consider the requirements of a given task.

Therefore, researchers have extended these "robust" grasp frameworks by using an additional neural network to learn correlations between suitable grasp regions and task requirements in object images [26], [32], [47]. These grasp frameworks use the concept of *part affordance*, which focuses on the specific details of how a person can interact with an object and how different parts of an object contain different functions [19]. The part affordance neural network is trained to recognize relations between shape and texture features of segmented object images. This model uses these features to perform pixel-wise segmentation, which involves assigning each pixel to a particular part affordance class. For example, the pixels in an image that indicate the handle of a hammer are labeled differently than the head, as shown in Figure 2.1. The segmented image is then used to select a grasp pose that results in a grasp in the desired affordance region. Data-driven grasp frameworks that include part affordance can thus select a grasp pose at a desired part affordance region, resulting in a task-specific object grasp. However, since the part affordance neural networks only learn relationships between shape and texture features from an image, other object properties (e.g., material, stiffness) cannot be included. Therefore, Liu *et al.* [32] proposed using an additional neural network to learn correlations between object properties (material and object state), part affordance labels (e.g., pound, scoop, grasp), and a given task (e.g., cooking, hammering) to understand where to grasp an object. This required a manually created dataset of 14,400 exemplar grasps for only 44 objects.

Figure 2.1: Object affordances obtained via AffordanceNet [16]. Showing pixel-wise a segmentation for household objects.

The integration of part affordance algorithms into data-driven approaches for learning task-specific object grasps has certain drawbacks. Firstly, since the correlations between tasks and affordances are learned from images, a large amount of accurately labeled data is required to comprehend these relationships. Secondly, identifying new part affordances or including additional semantic information necessitates costly training data, and the model must be retrained. Lastly, part affordance models provide no transparency regarding how the features are learned and how the decisions are made. As a result, it can be difficult to explain why a network is making certain decisions or producing certain outputs. Therefore, data-driven approaches that rely on part affordances are unsuitable for environments with a variety of objects and tasks, such as those found in a household or retail store.

2.1.3. Logic-based Approaches

Logic-based approaches rely on *symbolic reasoning*, which employs a set of facts and logic-based rules that describe the robot's capabilities and its environment. In contrast to analytical and data-driven approaches, logic-based approaches are specifically designed to reason over semantic information. These approaches can be based on either First-Order Logic (FOL) or description logic, which are formal languages created for knowledge representation and reasoning [51]. Description logic is a subset of FOL that is often employed by Web Ontology Language (OWL) to reason over available semantic information. The OWL ontology is a knowledge base that is designed to represent data from various domains and to facilitate reasoning [53].

The studies conducted by Boruah *et al.* [9] and Li and Tian [29] use description logic to extract information from an OWL ontology for finding grasp poses on an object. Both studies employ an OWL ontology to represent data from different domains, allowing the robot to reason with description logic and identify a suitable grasp pose. This approach enables the frameworks to reason over complex information and situations by using the semantic information stored in the OWL ontology. However, to grasp an object from an image or point cloud, prior information is necessary and must be stored in the OWL ontology. In contrast, Antanas *et al.* [3] proposes an alternative logic-based approach that does not require an OWL ontology. Instead, it uses probabilistic logic rules to reason over the semantic information obtained from an object point cloud. The framework decomposes the object point cloud into multiple segmented parts using the Principal Components Analysis (PCA). Probabilistic logic rules are then used to identify the object class and a suitable grasp region based on the segmented parts. Finally, an additional algorithm is employed to recover grasp poses on the point cloud. This framework provides transparency, allows for the addition or removal of rules without affecting other rules, and can incorporate multiple types of semantic information into the rules. However, this grasp framework is limited to a few objects placed in fixed orientations.

Logic-based approaches demonstrate their ability to reason over different types of semantic information to draw new conclusions [29]. However, logic-based approaches are limited in their ability to find grasp poses from images or point clouds and require prior knowledge to decide how to grasp an object. In addition, these approaches often only focus on the grasp type but not where an object should be grasped [9]. Therefore, logic-based approaches are unsuitable for finding suitable grasp poses on unknown objects in household and retail store environments.

2.2. Shape Completion Methods

Shape completion methods deal with restoring missing parts of an object. These techniques are of significant value as it enables robots to identify and fill in missing parts of an object, which is key to their successful interaction with the world. These methods can provide a logic-based approach with information to understand the overall structure of the object better. It allows a robot to understand important characteristics such as an object's size and shape. Shape completion methods are algorithms that are used to reconstruct a partial shape in three dimensions. These methods can be used to complete an incomplete 3D object or to reconstruct a 3D object from a partial 2D image.

A specific type of shape completion method focuses on representing objects as primitive shapes using a superquadric algorithm, which parameterizes objects using a mathematical equation [23]. It uses an objective function to find a set of primitive shape parameters, such as its position, orientation, size, and shape, that will fit a set of points to represent the object's shape. According to the Recognition By Components theory (RBC) [6], humans identify objects by segmenting them into primitive shapes (e.g., cuboids, cylinders, spheres). This theory can be applied to everyday household and retail store objects. For example, a hammer is composed of a handle and a head, while a spatula is composed of a handle and a blade (see Figure 2.2). According to the RBC theory, recognizing objects as primitive shapes has the advantage of the viewpoint invariance principle, which means that the same object can be identified regardless of the orientation of the object. This is an important advantage, as in real-world scenarios, objects can be observed from different angles [25].

Researchers have attempted to recover primitive shapes using deep learning methods or auxiliary

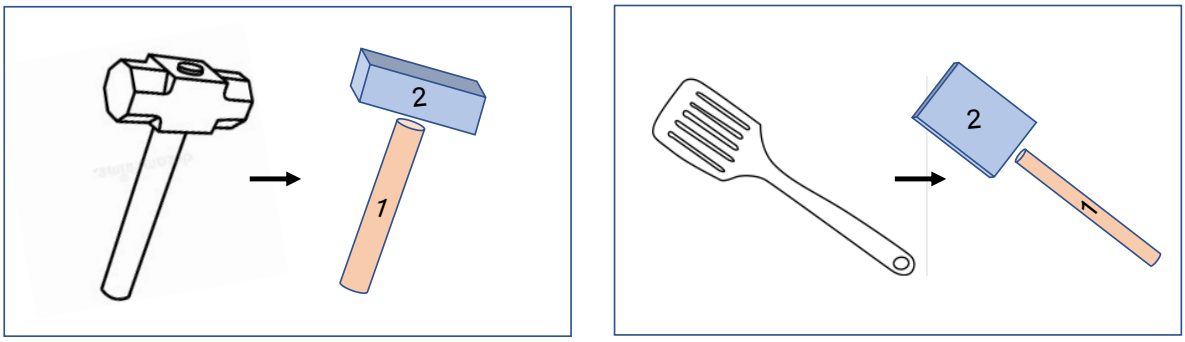

Figure 2.2: An example of a hammer (left) and spatula (right) that are represented by multiple primitive shapes according to the Recognition by Components theory.

functions to enhance the accuracy of recovered primitive shapes [17], [52]. However, these algorithms have limitations in accurately representing objects, as they rely on a single primitive shape or fail to capture the translation and rotation of the primitive shapes in the world frame. In contrast, Liu *et al.* [31] proposed a primitive recovery method that can retrieve multiple primitive shapes from a partial object point cloud using an expectation and maximization algorithm. Their approach can recover primitive shapes from partial object point clouds without requiring prior knowledge of the object, and it provides semantic information about the object's decomposition by recovering multiple primitive shapes. This approach offers improved accuracy in representing objects compared to previous methods that relied on a single primitive shape or failed to capture the object's translation and rotation.

3

Background

This section introduces the data-driven approach (see Section 3.1) [43], the superquadric algorithm (see Section 3.2) [31], and the components Prolog (see Section 3.3) and OWL-ontology (see Section 3.4) that are used by the logic-based approach.

3.1. Data-driven approach

The SPaRGE framework uses the data-driven grasp approach developed by Pas *et al.* [43] to generate multiple grasp poses on a partial point cloud. This off-the-shelf 6-DOF grasp generator uses a Convolutional Neural Network (CNN) to rank each grasp pose based on its robustness. Initially, the grasp framework selects the grasp pose with the highest probability score to grasp an object. However, in the SPaRGE framework, the grasp poses are evaluated based on the location where the object will be grasped. A grasp pose is represented by a position and orientation relative to the object, in which the end-effector, a parallel gripper, approaches the object (see Figure 3.1).

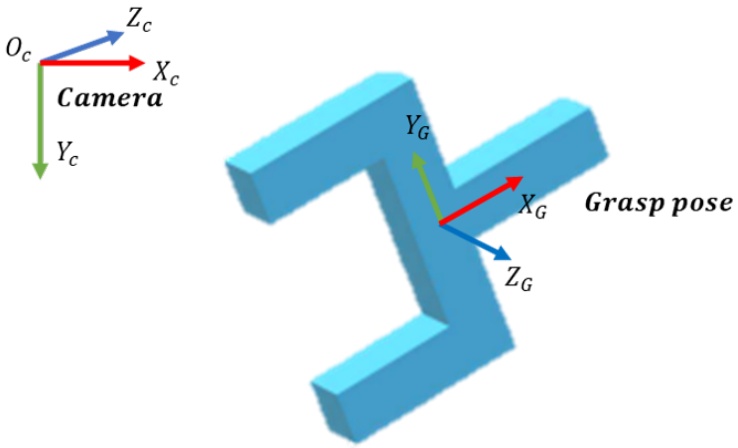

Figure 3.1: Grasp pose representation in the camera frame according to Pas *et al.* [43].

$\{C\}$ is the camera frame, and $\{G\}$ is the grasp pose frame attached to the parallel gripper. Therefore, a grasp pose is defined as:

$$\begin{cases} {}^{C}P = [x, y, z]^T \\ {}^{C}_{G}R = \left[{}^{C}X_G, {}^{C}Y_G, {}^{C}Z_G\right] \end{cases} \tag{3.1}$$

3.2. Superquadric Algorithm

In this section, we elaborate on the components of the superquadric algorithm that are used to recover primitive shapes [31]. It consists of the Expectation, Maximization, and Switch (EMS) algorithm to recover a primitive shape on a set of points, and a hierarchical decomposition approach to recover multiple primitive shapes on a partial object point cloud.

3.2.1. EMS Algorithm

This work employs the EMS algorithm that uses the implicit superquadric function to find a primitive shape (e.g., cuboid, cylinder) that fits the points of a point cloud. The implicit superquadric function is a convenient equation that can represent many shapes in a simple, closed-form expression (see Equation (3.2)). The EMS algorithm uses the implicit superquadric function to find a primitive shape, which is parametrized by the shape, size, translation and rotation parameters.

The implicit superquadric function uses the shape parameters $\{\epsilon_1, \epsilon_2\}$ to represent the form of a primitive shape, while the size parameters $\{\alpha_1, \alpha_2, \alpha_3\}$ are used to vary its dimensions. The function can be used to evaluate whether a point $\boldsymbol{p} = \{x, y, z\}$ lies on the surface of a primitive shape $F_{\boldsymbol{\theta}}(x, y, z) = 1$, inside the primitive shape $F_{\boldsymbol{\theta}}(x, y, z) < 1$, or outside the primitive shape $F_{\boldsymbol{\theta}}(x, y, z) > 1$ of the primitive shape. In this equation, the origin of a primitive shape is located at $(0, 0, 0)$. By adding the parameters for translation and rotation, a primitive shape can be represented in any coordinate frame.

$$F_{\boldsymbol{\theta}}(x, y, z) = \left(\left| \frac{x}{a_1} \right|^{\frac{2}{\epsilon_2}} + \left| \frac{y}{a_2} \right|^{\frac{2}{\epsilon_2}} \right)^{\frac{\epsilon_2}{\epsilon_1}} + \left| \frac{z}{a_3} \right|^{\frac{2}{\epsilon_1}} \tag{3.2}$$

The EMS algorithm uses the implicit superquadric function to measure the distance error between a set of points and the surface of a primitive shape. Because this optimization problem is difficult to solve and computationally expensive, the EMS algorithm uses the Expectation and Maximization algorithm to find a primitive shape that matches a set of points. It works by alternating between two steps: the expectation (E) step and the maximization (M) step. In the E-step, the algorithm computes the expected value of the log-likelihood function given the current parameter values. In the M-step, the algorithm updates the parameter values to maximize the expected value of the log-likelihood. The log-likelihood function minimizes the distance between a primitive and a set of points. This iterative process continues until either convergence is achieved, or until the switch (S) step is triggered. The switch step is a heuristic approach that avoids a local optimum. As shown in Figure 3.2 two superquadrics with distinct shape parameters (ϵ_1, ϵ_2) can possess a similar or even identical shape. Therefore, the switch step changes the order of the dimension parameters $(\alpha_1, \alpha_2, \alpha_3)$ to find a global optimal primitive shape. For a detailed explanation of the EMS algorithm, the reader is referred to Liu *et al.* [31].

The EMS algorithm can translate and rotate a primitive shape to better fit the primitive shape to a set of points. The translation is the distance from the center of the primitive shape to the origin of the camera frame $\boldsymbol{t} = \{t_x, t_y, t_z\}$, while the rotation is denoted in unit quaternions $\boldsymbol{q} = \{q_x, q_y, q_z, q_w\}$. In the end, the EMS algorithm outputs 12 parameters to represent a primitive shape $\boldsymbol{\theta_s}$ in the camera frame, see Equation (3.3). The shape $\{\epsilon_1, \epsilon_2\}$ parameters are bounded to $0 < \epsilon_1, \epsilon_2 < 2$, which results in only a set of convex primitive shapes.

$$\boldsymbol{\theta_s} = \{\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, \alpha_3, t_x, t_y, t_z, q_i, q_j, q_k, q_w\} \tag{3.3}$$

3.2.2. Hierarchical Decomposition

To recover multiple primitive shapes on a partial object point cloud, the EMS algorithm uses a hierarchical decomposition approach, which is an iterative process that starts by sampling a primitive to an object point cloud using the EMS algorithm. Then the output of the EMS algorithm is a primitive shape and a set of outlier points that were not used for fitting the primitive shape. The outlier points are clustered and used by the EMS algorithm to recover a new primitive shape for each cluster. This process is repeated iteratively untill a threshold has been met and multiple primitive shapes have been found that accurately represent the geometry of the object.

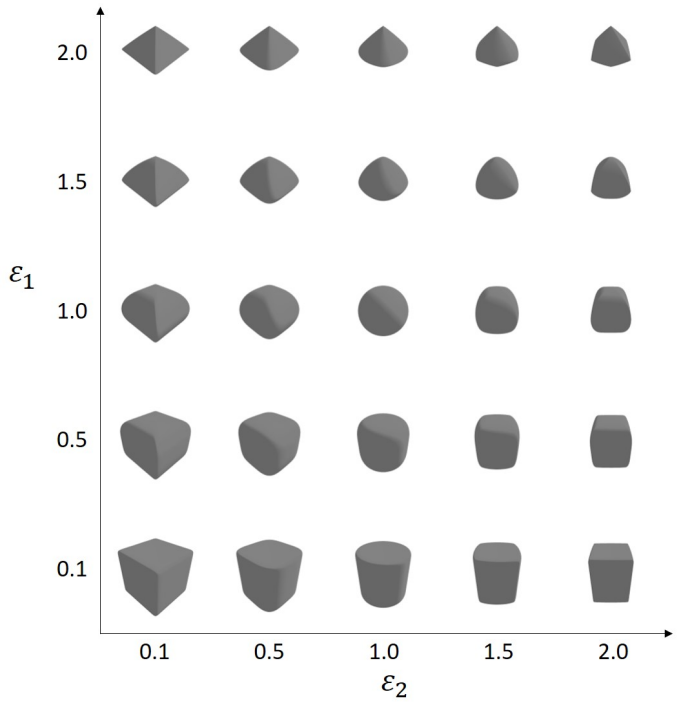

Figure 3.2: The vocabulary of the implicit superquadric function with varying shape parameters $\{\epsilon_1\ \epsilon_2\}$; whereas the range of the shape parameters is bounded to [0,2].

As shown in Algorithm 1, the hierarchical decomposition approach takes a partial point cloud represented by a set of points $\boldsymbol{P}_i$ as input. For each set of points $\boldsymbol{P}_i$ the algorithm fits one primitive shape and returns a set of points marked as $outliers$ that are not used to fit the primitive shape (line 2). The recovered primitive shape is added to the set of earlier found primitive shapes $\boldsymbol{\Theta}$ (line 3). Then, the outliers are clustered using the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) approach, which groups a set of points $(\overline{\boldsymbol{P}}_n)$ together that are closely packed (line 4) [18]. Clusters that have fewer points than the threshold $MinPoints$ are pruned, and the remaining clusters are fed back to the algorithm to recover new primitive shapes (lines 5-6). The optimal threshold value for $MinPoints$ is found using hyperparameter tuning of the superquadric algorithm in Appendix A.1.1. The output of the hierarchical decomposition approach is a set of primitive shapes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_n\} \in \boldsymbol{\Theta}$, as illustrated in Figure 3.3. It demonstrates the process of recovering primitive shapes on a partial object point cloud. Each primitive shape is parametrized according to the parameters in Equation (3.2).

Algorithm 1 Hierarchical decomposition algorithm for recovering multiple primitive shapes [31].

Input: $\{\boldsymbol{P}\}$
Output: $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_s\} \in \boldsymbol{\Theta}$
1: **while** $\boldsymbol{P} \neq \emptyset$ **do**
2: $(\boldsymbol{\theta_i}, outliers) \longleftarrow EMS\_algorithm(\boldsymbol{P}_i)$
3: $\boldsymbol{\Theta} \longleftarrow Append(\boldsymbol{\theta_i})$
4: $\{\overline{\boldsymbol{P}}_1, \overline{\boldsymbol{P}}_2, \dots, \overline{\boldsymbol{P}}_n\} \longleftarrow Clustering(outliers)$ clustering by DBSCAN [18]
5: **if** $Length(\overline{\boldsymbol{P}}_n) < MinPoints$ **then**
6: $\boldsymbol{P} \longleftarrow \overline{\boldsymbol{P}}.remove(\overline{\boldsymbol{P}}_n)$
7: **end if**
8: **end while**
9: **return** $\boldsymbol{\Theta}$

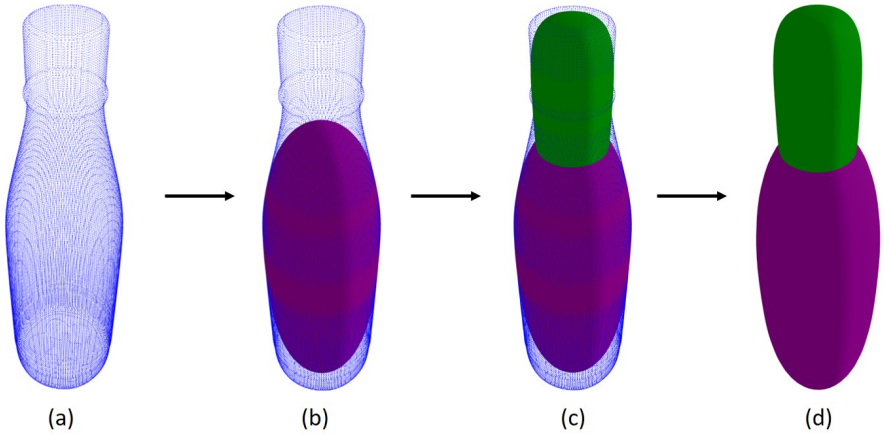

Figure 3.3: Step-by-step overview of the process of recovering multiple primitive shapes using the hierarchical decomposition algorithm. [31]. We demonstrate the process of recovering primitive shapes on a bottle point cloud. (a) The point cloud (blue) to recover primitive shapes on. (b-c) Steps of the hierarchical process of recovering primitive shapes. (d) The primitive shapes representing the object.

3.3. Prolog

Prolog is a logic programming that has its roots in FOL. It is a declarative programming language that expresses relations, represented as facts and rules. This language enables users to pose queries using facts and rules to make deductions through deductive reasoning [59]. Prolog is unique in that it operates under the assumption that its database is a closed world; if it cannot prove something is *true*, it assumes it is *false*. This makes Prolog particularly powerful as it allows for concise, understandable programs that can solve complex problems. This section explains the concepts and terminology necessary to understand the logic programming language Prolog.

Prolog syntax

Prolog is constructed from terms, which is the overarching name to indicate an atom, number, variable, or complex term. Atoms are strings of characters that start with a lower-case letter, or are enclosed by single quotes. Numbers are represented as real numbers by integers or floats. Variables start with an upper-case letter or with an underscore. These atoms, numbers, and variables are the building blocks to create complex terms. Complex terms consist of a functor, followed by a sequence of arguments in parentheses separated by commas. The number of arguments a complex term contains is known as its arity. For example in Figure 3.4 the complex term `robotLocation(tiago, X).`, has the functor `robotLocation()` and two arguments; the atom `tiago` and the variable `X`, which could indicate its location (e.g. room1). In this example, the arity is 2, which is important since functors with the same name can be different based on the arity.

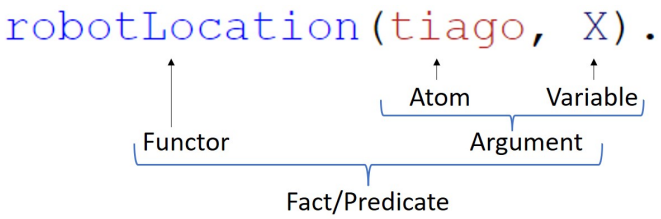

Figure 3.4: Example of Prolog syntax based on predicates, facts, atoms, variables, arguments, and functors.

A functor is referred to as a predicate, whenever it evaluates if something is *true* or *false*. It is incorporated into a knowledge base as either a fact, head of a rule, or part of the body of a rule. Facts are used to state things that are unconditionally true of some situation of interest and are ended with a full stop. For example, the fact `robot(tiago).` states that `tiago` is a robot. A rule is a logical implication to describe a relationship among facts, separated by (:-) to indicate the left side as the head, and

the right side as the body. The body of a rule consists of multiple predicates, which must all be true for the rule itself to be true. A logical conjunction is expressed in Prolog with a comma which means *and*.

Querying information
When using Prolog, a query is submitted to the Prolog interpreter which starts with `?-`. It allows users to query the database, similar to the body of a rule by using a sequence of predicates separated by commas and terminated by a full stop. This is like asking a question, as the purpose of the query is to find values to satisfy the variables in the query. If the query matches the information in the knowledge base, the Prolog interpreter will return *true*, meaning that the fact exists.

We can use Prolog to query the knowledge base in Code 3.1 to ask if the robot TIAGo has a left-sided manipulator suited for grasping; `?- canGrasp(tiago, leftArm).`. This will return *true* for TIAGo, and *false* for the Roomba robot. Furthermore, we can use unification to replace the argument in the query with a variable. For example, if we ask `?- canGrasp(tiago, X).`, Prolog will return `X = leftArm; X= rightArm`, which means TIAGo can use its left and right arms for grasping.

Code 3.1: Example of how in Prolog relations are stored about the robot TIAGo and Roomba.

```
robot(tiago).
robot(roomba).
hasManipulator(tiago, leftArm).
hasManipulator(tiago, rightArm).

canGrasp(X,Y) :-
    robot(X),
    hasManipulator(X,Y).
```

3.4. OWL-Ontology

An ontology provides a common vocabulary to capture relations in a domain [36]. It includes machine-interpretable definitions of concepts in the domain and relations among them. A robotic system can use the relations defined in an ontology to reason about the physical world. The ontology used in this work is based on Web Ontology Language (OWL). OWL is a family of knowledge representation languages used for expressing ontologies [51]. In this section, we explain the concepts and terminology necessary to understand the OWL ontology, which can be implemented using Protégé [36].

Ontology syntax
An OWL ontology is composed of classes, instances, data properties, and object properties [40]. The core of an OWL ontology consists of classes that represent concepts, objects, or ideas. Each class is associated with a set of individuals known as its class extension, which are referred to as instances of the class. Classes may also have subclasses, allowing for further specification and hierarchical structuring of the OWL ontology. Typically, class names are written with an initial uppercase letter, while instance names begin with a lowercase letter. An instance of a class is an individual object that can refer to a specific entity, person, or function. For instance, as depicted in Figure 3.5, the class Robot has the instances Roomba and TIAGo, which indicate that Roomba and TIAGo are types of robots.

In addition to defining classes and instances, an OWL ontology can represent relations between instances and classes through object and data properties. An object property is used to represent a relationship between two classes or instances (e.g. ”hasManipulator”), while a data property is used to represent an attribute of an individual (e.g. ”hasWeight”). For instance, the robot TIAGo is equipped with both a *LeftArm* and *RightArm*, and a relationship between them is established through the object property ”hasManipulator”.

Querying information
Prolog will be used to implement the logic rules of the SPaRGE framework. Therefore, Prolog triples will query the knowledge stored in the OWL ontology, representing the RDF-triple based representation of

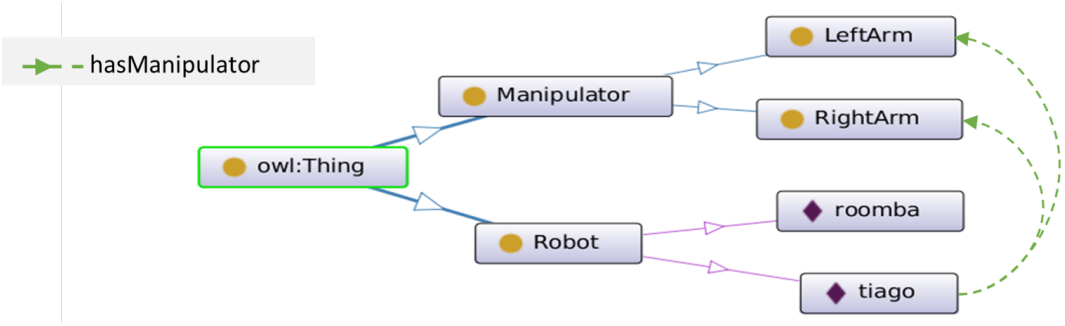

Figure 3.5: An OWL ontology containing the relationship for the robot TIAGo and Roomba.

OWL. A triple is a set of three components representing a relationship between two classes or instances that are defined in the OWL ontology. It is a common data structure used in RDF-based representations of OWL, consisting of a subject, predicate, and object, represented as `?- triple(subject, predicate, object).`. The subject is the person, place, or thing that is the focus of the statement. The predicate is the action or state-of-being that applies to the subject. The object is the person, place, or thing that is affected by the action or state of being expressed in the predicate. For example, in the sentence ”TIAGo has a left arm manipulator”, TIAGo is the subject, has a manipulator is a predicate, and the left arm is the object. This shows that the relations defined by the object or datatype property are used as the predicate in a triple, while the classes and instances are the subject or object. Triples allow the OWL ontology to be represented in a way that is easy for Prolog to process and interpret stored information.

4

Primitive Shape

In this Chapter, we present two additional algorithms that are utilized in conjunction with the superquadric algorithm in the SPaRGE framework to provide the logic-based approach with more informative semantic information. Specifically, an overlap removal algorithm (see Section 4.1) is proposed to eliminate primitive shapes with excessive overlap. This is necessary due to the tendency of the superquadric algorithm to recover multiple overlapping primitive shapes in certain object regions. Additionally, a classification algorithm is introduced that classifies the recovered primitive shapes into cuboids, cylinders, or spheres based on their shape and size parameters. It provides the logic-based approach with additional semantic information to improve the robot's reasoning capabilities to decide where to grasp an object.

4.1. Primitive Overlap Removal

Due to the incompleteness and noise of partial object point clouds, recovering primitive shapes can be challenging. When a partial object point cloud is fed into the EMS algorithm, primitive shapes may be recovered in the same area. This can be problematic, as primitive shapes with similar shapes and positions are likely to represent the same part of an object, making them redundant and providing no extra semantic information. The authors of the original paper on the superquadric algorithm [31] did not address this limitation, as they only tested their algorithm to recover multiple primitive shapes on a complete point cloud, or a single primitive shape on a partial point cloud. Therefore, to address this problem, the overlap removal algorithm (see Algorithm 2) removes primitive shapes that show too much mutual overlap.

The algorithm starts with two for loops, iterating over the set of recovered primitive shapes $\boldsymbol{\Theta}$ to compute their respective mutual overlap (lines 1-2). For every iteration, $\boldsymbol{\theta}_j$ is the reference primitive which is used to compute the amount of overlap with the primitive shape $\boldsymbol{\theta}_i$ (line 3-4). The coordinate system in which the primitive shapes are taken is denoted as the $origin$, indicated as ${}^{origin}\boldsymbol{\theta}j$. The implicit superquadric function (see Equation (3.2)) is used to evaluate the amount of overlap between two primitive shapes by determining if a given point lies inside or outside a respective primitive shape. To determine this overlap, we sample 5000 points inside the volume of the reference primitive shape ${}^{origin}\boldsymbol{\theta}_j$ (line 5). A rejection sampling strategy is used to generate a uniformly distributed set of points within the volume. This sampling allows us to compute the amount of the primitive shape volume that overlaps with another primitive. The points ${}^{origin}\boldsymbol{X}_j$ are transformed to the coordinate system of the primitive from which the overlap will be measured (line 6). Then, the distances of the points ${}^{i}\boldsymbol{X}_j$ to the primitive $\boldsymbol{\theta}_i$ are computed using the implicit superquadric function (line 8). If $F < 1$, the given point ${}^{i}x_j$ lies inside the volume of a primitive shape $\boldsymbol{\theta}_i$, while if $F > 1$ the point lies outside the volume of the primitive shape. Thus, the amount of overlap primitive $\boldsymbol{\theta}_j$ has with primitive $\boldsymbol{\theta}_i$ can be determined by computing the ratio of the number of overlapping points within the total set of points (line 10). Resulting in $overlap_{j,i}$, which indicates how much the volume of primitive j overlaps with primitive i. If primitive j has more than 50% overlap with another primitive, it will not be included in the set of filtered primitive shapes $\boldsymbol{\Theta}$. If two primitive shapes share more than 50% overlap, the one with the least overlap will be kept. As an example, if $overlap_{1,2} = 65\%$ and $overlap_{2,1} = 55\%$, then primitive θ_2 will be added to

Algorithm 2 Overlap removal algorithm

Input: $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \dots, \boldsymbol{\theta}_s\} \in \boldsymbol{\Theta}$
Output: $\{\tilde{\boldsymbol{\theta}}_1, \tilde{\boldsymbol{\theta}}_2, \dots, \tilde{\boldsymbol{\theta}}_s\} \in \tilde{\boldsymbol{\Theta}}$
for $i = 1, 2, \dots, n$ **do**
  for $j = 1, 2, \dots, n$ **do**
    ${}^{i}\boldsymbol{\theta}_i \longleftarrow$ overlap primitive
    ${}^{origin}\boldsymbol{\theta}_j \longleftarrow$ reference primitive
    ${}^{origin}\boldsymbol{X}_j \longleftarrow sample\_volume\_primitive({}^{origin}\boldsymbol{\theta}_j)$
    ${}^{i}\boldsymbol{X}_j \longleftarrow \boldsymbol{T}^{i}_{origin}\,{}^{origin}\boldsymbol{X}_j$ Transformation points $\boldsymbol{X}_j$ to ${}^{i}\boldsymbol{\theta}_i$ frame
    for ${}^{i}x_j \in {}^{i}\boldsymbol{X}_j$ **do**
      $dist_j \longleftarrow F({}^{i}x_j, {}^{i}\boldsymbol{\theta}_i)$ Equation (3.2)
    end for
    $overlap_{j,i} = \frac{length(dist_j<1)}{length(dist_j)}$
    if $overlap_{j,i} < Threshold$ **then**
      $\boldsymbol{\theta}_j \in \tilde{\boldsymbol{\theta}}$
    end if
  end for
end for
return $\tilde{\boldsymbol{\theta}}$

the filtered primitive shapes $\boldsymbol{\Theta}$. The outcome of the algorithm will be a set of filtered primitive shapes $\tilde{\boldsymbol{\Theta}}$. Figure 4.1 illustrates two overlapping primitive shapes, with the overlapping areas represented in orange.

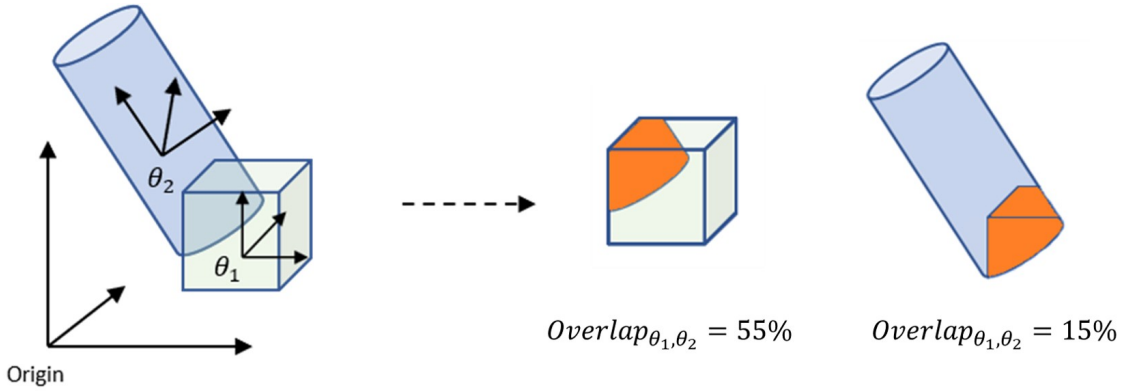

Figure 4.1: Example of two recovered primitive shapes $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ that have mutual overlap. Primitive shape $\boldsymbol{\theta}_1$ overlaps 55% of its own volume with primitive shape $\boldsymbol{\theta}_2$, and primitive shape $\boldsymbol{\theta}_2$ overlaps 15% of its own volume with primitive shape $\boldsymbol{\theta}_1$. Meaning that primitive shape $\boldsymbol{\theta}_1$ is removed.

4.2. Primitive Shape Classification

Classifying primitive shapes into cuboids, cylinders, and spheres is important for identifying the appropriate shape for a particular task and facilitating shape comparisons. The superquadric algorithm recovers primitive shapes that are defined by two shape parameters $\{\epsilon_1, \epsilon_2\}$ and three size parameters $\{\alpha_1, \alpha_2, \alpha_3\}$. However, without further analysis, it can be challenging to identify the type of 3D-form based solely on these parameters. Therefore, a classification algorithm is necessary to classify primitive shapes into their respective shape classes: cuboids, cylinders, or spheres. Figure 4.2 provides an example of how the shape parameters influence the 3D-form.

To decide which classification algorithm is most suitable, we conducted an experiment using a synthetic dataset of 550 labeled primitive shapes, consisting of cuboids, cylinders, and spheres. The labels were assigned manually based on our criteria. We compared four different multi-classification algorithms. A heuristic classification algorithm is compared with three machine learning classifiers:

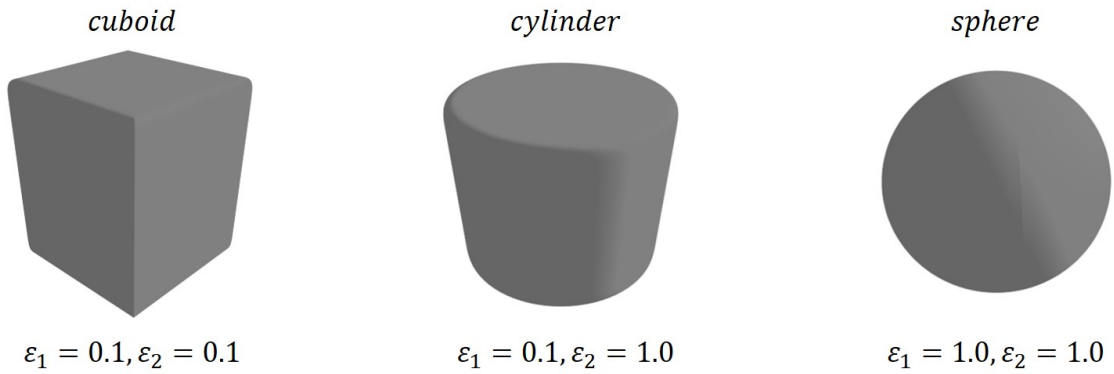

Figure 4.2: Example of classified primitive shapes based on ϵ_1 and ϵ_2 into cuboids, cylinders, and spheres.

Random Forest Classifier [10], Support Vector Machine (SVM) [60] and Multi-Layer Perceptron (MLP) [21]. The experiment results indicated that the MLP algorithm is most suited for classifying primitive shapes. This research will move forward using the trained MLP to classify primitive shapes into cuboids, cylinders, or spheres. A detailed overview of the experiment and the parameters of the MLP algorithm can be found in Appendix A.1.2.

4.3. Conclusion

In conclusion, this chapter presented two additional algorithms that are utilized in conjunction with the superquadric algorithm. The overlap removal algorithm is an algorithm to eliminate primitive shapes with excessive overlap. This algorithm helps to address the problem of recovering primitive shapes on incomplete and noisy partial object point clouds, which can lead to the recovery of overlapping primitive shapes in the same region. Furthermore, the classification algorithm classifies the primitive shapes into cuboids, cylinders, or spheres. It provides the SPaRGE framework with more informative semantic information, which helps to decide where to grasp an object.

5

Method: SPaRGE Framework

This chapter presents the Shape Primitive and Reasoning Grasping Engine (**SPaRGE**), a grasp framework that combines a data-driven, superquadric, and logic-based approach to identify task-specific grasp poses on partially visible objects. This involves performing tasks, such as hammering and cooking, on objects that can be decomposed into multiple primitive shapes (e.g., hammer, spatula, pan).

Section 5.1 explains how the SPaRGE framework merges the data-driven approach from Section 3.1, the superquadric algorithm from Section 3.2, and the auxiliary algorithms from Chapter 4 into a cohesive framework. In addition, the SPaRGE framework uses one of the two logic-based versions to perform reasoning on semantic information; see Section 5.2.1 and Section 5.2.2. Both versions reason over semantic information obtained from primitive shapes through symbolic reasoning. In general, both logic-based approaches determine in which region a robot should grasp an object to meet the requirements of a task.

5.1. Overview Framework

The SPaRGE framework enables a robot to conduct real-time reasoning on everyday objects to identify task-specific grasp poses. The SPaRGE framework is illustrated in Figure 5.1.

As depicted in Figure 5.1, the SPaRGE framework consists of two pathways. The first path involves a data-driven approach, as explained in Section 3.1 [43], which takes a scene point cloud as input to generate multiple grasp poses. The second pathway takes a partial object point cloud as input, which is then processed by the primitive recovery modules. The primitive recovery module uses the superquadric algorithm to sample primitive shapes [31] (see Section 3.2), which are filtered using the overlap removal algorithm to eliminate any overlapping primitive shapes (see Section 4.1), and classifies the remaining primitive shapes according to their shape (cuboids, cylinders, or spheres) (see Section 4.2). The output of the primitive recovery modules is used as input by the logic-based approach to identify a primitive shape that results in a suitable grasp region. This primitive shape is then segmented onto the partial object point cloud to identify a suitable region for grasping. Finally, using the set of grasp poses and the segmented partial object point cloud, a grasp pose is selected in the suitable grasp region.

The logic-based approach uses semantic information from the primitive recovery modules and a specified task (e.g., Pouring, Hammering) to identify a primitive shape that results in a suitable grasp region. It uses logic rules defined in Prolog based on FOL. This allows for symbolic reasoning using semantic information to draw conclusions based on the relationship. We have made two versions of the logic-based approach, both containing different design choices that influence the generalizability and scalability of the SPaRGE framework. In Figure 5.1, the first approach, indicated in purple, is reasoning Version-1, which employs logic rules. While the second approach, indicated in orange, is reasoning Version-2, which utilizes logic rules in conjunction with an OWL ontology. The difference between reasoning Version-1 and Version-2 is as follows:

- ***Reasoning Version-1***: Captures the relationship between primitive shapes, the shape labels, and specific tasks, using logic rules. A logic rule is created for each task in Prolog. Then, by

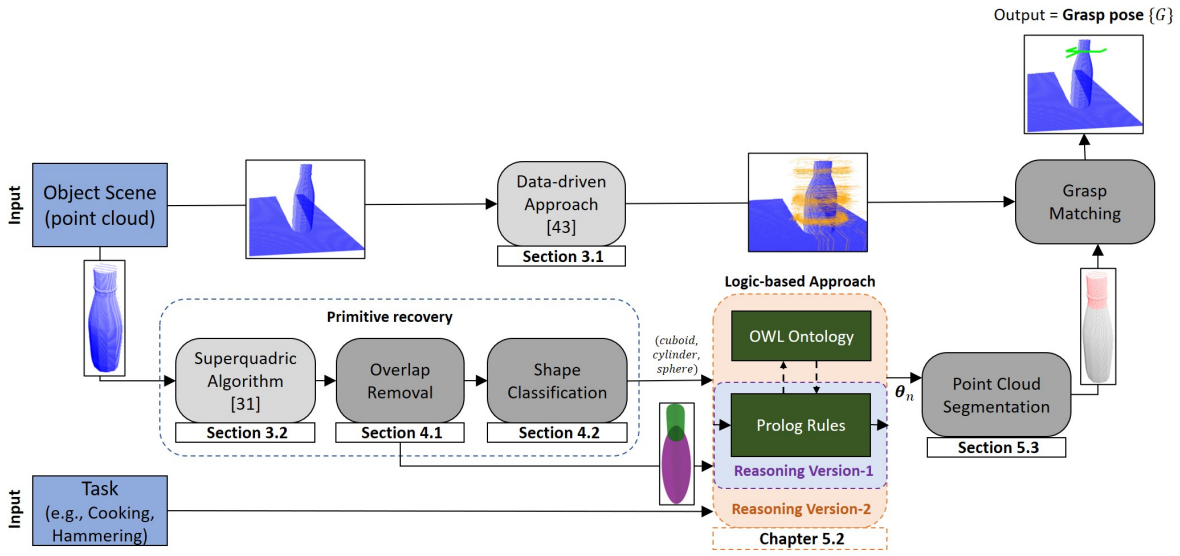

Figure 5.1: Complete overview of the SPaRGE framework to perform task-specific object grasps. The framework uses an existing data-driven approach to recover multiple grasp poses [43]. In addition, it uses a state-of-the-art superquadric algorithm in combination with an overlap removal algorithm and a shape classification algorithm [31]. The logic-based approach consists of two versions, reasoning Version-1 using only Prolog logic rules and reasoning Version-2 using Prolog logic rules with our OWL ontology. Finally, a suitable grasp pose is found by combining the output of the data-driven and logic-based approaches through grasp matching.

using a Prolog query, we search for a primitive shape that meets the requirements of a task as defined by the logic rules. This version enables the creation of task-specific logic rules that must be satisfied by a collection of primitive shapes.

- ***Reasoning Version-2***: This version incorporates the concept of part affordance, which uses logic rules to classify primitive shapes into part affordance labels (e.g., contain, pound, grasp) based on the primitive shapes, and shape labels. A logic rule is created in Prolog for each part affordance. Then, the relationships between the part affordances required for a task are stored in an OWL ontology. By using a Prolog query, we identify the available part affordances and extract the relations from the OWL ontology to verify whether the requirements of a task are met.

The following section details the design choices behind reasoning Version-1 and Version-2 and their respective symbolic reasoning abilities.

5.2. Reasoning Module

This section explains two logic-based approaches for identifying a task-specific grasp region on everyday objects, such as those found in households and retail stores. To choose the predicates and parameters for a particular task or part affordance, we identify the relevant objects and look for similarities among them. It assumes that objects with the same task or part affordance share the same primitive shape properties. This enables us to create logic rules that generalize among multiple objects. Both approaches leverage the reasoning capabilities of Prolog, using primitive shapes and corresponding shape labels (cuboids, cylinders, or spheres) to reason over semantic information.

To reason over the primitive shapes and their respective shape classes using logic rules, the semantic information is stored in Prolog as facts. An example of how a recovered primitive shape is stored as a fact is shown in Code 5.1. For each unique primitive shape, a fact stores the following properties: ID, dimension, and shape class. The ID refers to a number that identifies each unique primitive. The dimension $(\alpha_1, \alpha_2, \alpha_3)$ is stored as a list, whereas the ID and shape class (cuboid, cylinder, or sphere) are stored as single atoms.

Code 5.1: Example of how the properties of primitive shapes are stored as Prolog facts. For each primitive shape, the ID, dimension, and shape class are stored.

```
primitive_values(ID, [Dimension], Shape_class).
primitive_values(1, [0.03, 0.02, 0.04], 'cylinder').
primitive_values(2, [0.06, 0.03, 0.07], 'cuboid').
```

The logic-based approach uses logic rules to determine whether the dimensions and shape of a primitive shape fulfill the requirements of a task or part affordance. As explained in Section 3.3, a logic rule consists of a head and a body. The head identifies a specific task or part affordance, whilst the body consists of predicates that a primitive shape must satisfy. If all the predicates in the body yield a *True* result, the head also returns *True*, signifying that the logic rule is satisfied and the primitive shape(s) belong to a specific task (reasoning Version-1) or part affordance (reasoning Version-2). To simplify the process of selecting predicates for the body, we have created a set of predefined predicates that the logic rules can reuse in both reasoning Version-1 and Version-2.

The predefined predicates use geometric constraints to find a primitive shape that matches the criteria. First, there are the `min_dimension(Dimension, [Value])` and `max_dimension(Dimension, [Value])` predicates. These predicates constrain the minimum and maximum dimensions a primitive shape can have. The variable *Value* is replaced by a list consisting of 3 values, which can be an integer or float. The predicate sorts the given dimensions from the lowest to the highest number, meaning it is possible to constrain values based on their indices in the list. For example, the predicate `max_dimension(Dimension, [1,1,3])` means that the values of the dimension must be no greater than 1 for the smallest two elements, and no greater than 3 for the largest element. In addition, the predicates `min_ratio(Dimension, [Value])` and `max_ratio(Dimension, [Value])` constrain the difference allowed between the minimum and maximum values of the dimensions. This enables the addition of constraints based on whether a shape is symmetrical or elongated. The operator `==` is used to determine if two terms are identical to restrict the required shape class that a primitive should meet, which can be either a cuboid, cylinder, or sphere. For example, by creating the relation `Shape_class == 'cylinder'`, the expected primitive should be classified as a cylinder to return *True*.

Code 5.2: General logic rule used by reasoning Version-1 and Version-2 to set requirements for a task or part affordance.

```
affordTask(ID, Task/Part_affordance) :-
    primitive_values(ID, Dimension, Shape_class),
    min_dimension(Dimension, [Value]),
    max_dimension(Dimension, [Value]),
    max_ratio(Dimension, Value),
    max_shape(ID),
    Shape_class == (cuboid/cylinder/sphere).
```

In Code 5.2, an example is shown of a logic rule. Depending on whether reasoning Version-1 or Version-2 is used, the predicate name of the head might be different, as well as the predicates and values that have been chosen in the body. However, the structure of the logic rule remains similar for both reasoning versions. For every task or part affordance, a unique logic rule is created. The body of the logic rule starts with the predicate `primitive_values(ID, Dimension, Shape_class)`, which searches the knowledge base for the available primitive shapes. Then, primitive shape properties that are stored as facts, are used by the predicates to verify whether they meet the requirements of the task or part affordance. The predicates and values of the logic rules are hand-picked based on the requirements of a task or part affordance. If all predicates in the body evaluate to *True*, the primitive shape satisfies the geometric constraints, and the corresponding primitive shape ID number will be returned as output.

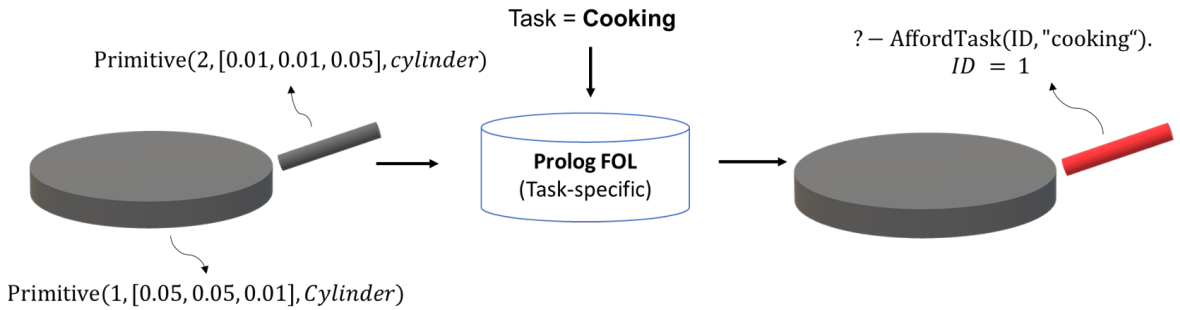

Figure 5.2: Overview of reasoning Version-1 that uses the properties of a primitive shape and task as input to output an ID that meets the task requirements. In this example, the object pan and task ”cooking” are used.

5.2.1. Reasoning Version-1. Task-specific Reasoning

Reasoning Version-1 is a logic-based approach consisting of logic rules that captures the relationship between primitive shapes and a specific task (see Figure 5.2). It assumes that objects that share the same task have similar primitive shapes, allowing the establishment of a task-specific logic rule that can be generalized to various objects. It uses a unique logic rule for each task (e.g., cooking, hammering), which contains a set of predicates based on geometric constraints to find a primitive that meets the task's requirements. The logic rules form a knowledge base, which is used to reason over the relationship between tasks and primitive shapes. To perform reasoning, the built-in search engine of Prolog is used to query over the logic rules. This query consists of the head of a logic rule that takes the task and the variable ID as input arguments. Consequently, it can answer the question: ”*Is there a primitive shape that meets the requirements of a given task?*”.

Code 5.3: Logic rule for the task ”cooking”.

```
affordTask(ID_2, "cooking") :-
    primitive_values(ID_1, Dimension_1, _),
    min_dimension(Dimension_1, [0.0, 0.08, 0.08]),
    max_shape(ID_1),

    primitive_values(ID_2, Dimension_2, Shape_class),
    min_ratio(Dimension_2, 1.5),
    max_dimension(Dimension_2, [0.05, 0.05, 1.00]),
    min_dimension(Dimension_2, [0.005, 0.01, 0.05]),
    Shape_class == 'cylinder',
    ID_1 \= ID_2.
```

The logic rule for the task ”cooking” (see Code 5.3) is created as follows: the predicates and values in the body have been hand-picked to reflect the expected dimensions and shape of objects suitable for the task. The logic rule searches for two primitive shapes that will be identified by `ID_1` and `ID_2`. For the first primitive, a shape with dimensions greater than $[0.0, 0.08, 0.08]$ is sought to reflect the cooking area. The second primitive that must represent the handle should have a maximum dimension of $[0.05, 0.05, 1.00]$, a minimum dimension of $[0.005, 0.01, 0.05]$, a minimum ratio of 1.5, and is identified as a cylindrical shape. At last, the two primitive shapes should be unique, as stated by `ID_1 \= ID_2`. As a result, if two primitive shapes meet the requirement of the logic rule, the primitive shape identified as `ID_2` indicates the suitable grasp region.

The Prolog predicate `?- AffordTask(ID, "cooking")` is used to query the knowledge base for a primitive shape that will meet the requirements of the task ”cooking”. If the requirements of the logic rule is met, Prolog will return the ID number of the primitive shape.

Using tasks-specific logic rules makes it possible to set constraints for primitive shapes directly. This is illustrated by the cooking example, where constraints are defined for multiple primitive shapes. Furthermore, implementing the logic rules in Prolog allows easily adapting and modifying the rules as needed. This includes adding or removing predicates and new tasks (see Section 6.2). If the constraints

of a task need to be adjusted or an additional task needs to be added, this will not affect the existing rules. However, to add a new task, an additional logic rule is required that requires expert knowledge to understand how to construct these rules.

5.2.2. Reasoning Version-2. Part Affordance Reasoning

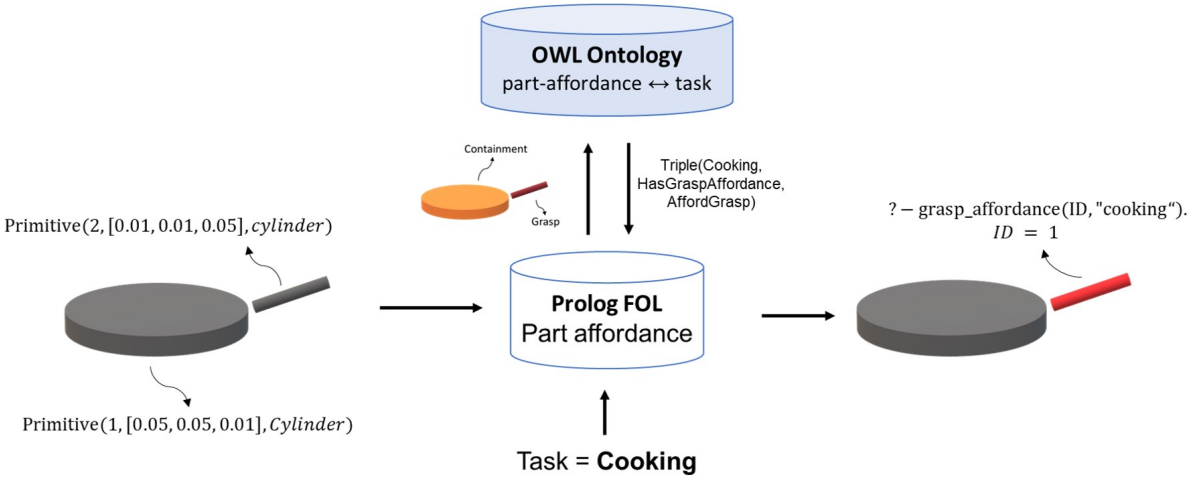

Figure 5.3: An example of the logic-based approach for reasoning Version-2 to find a primitive shape for the task "cooking" using a pan. It uses logic rules to classify the primitive shapes into part affordance labels. It uses logic rules to query the OWL ontology to understand which part affordance label should be grasped. The output is the ID of a primitive shape that meets the task's requirements.

Reasoning Version-2 is a logic-based approach consisting of logic rules in combination with an ontology. The logic rules are based on classifying primitive shapes into part affordance labels (e.g., pound, grasp, cut) [16]. It assumes that objects with the same type of part affordance share a similar primitive shape. For each part affordance, a unique logic rule is created containing a set of predicates based on geometric constraints to find a primitive shape that meets the task's requirements. The logic rules form a knowledge base, which is used to reason over the relationship between part affordances and primitive shapes. The relations between tasks and part affordance are defined in an OWL ontology. This information can be accessed by the logic rules using a Prolog query. Consequently, reasoning Version-2 can answer the questions: "*What types of part affordances does an object hold?*" and "*At which part affordance should a robot grasp an object to perform the task?*".

Code 5.4: Logic rules for the part affordances AffordContain and AffordGrasp.

```
affordance(ID, AffordContain) :-
    primitive_values(ID, Dimension, Shape_class),
    min_dimension(Dimension, [0.03,0.03,0.03]),
    max_shape(ID),
    Semantic_shape == 'cylinder'.

affordance(ID, AffordGrasp) :-
    primitive_values(ID, Dimension, Semantic_shape),
    min_dimension(Dim, [0.01, 0.01, 0.05]),
    max_dimension(Dim, [0.05, 0.05, 1]),
    min_ratio(Dim, 2),
    (Semantic_shape == 'cylinder' ;
        Semantic_shape == 'cuboid').
```

Code 5.4 presents the logic rules for the part affordances "AffordContain" and "AffordGrasp" that are both required to perform the task "cooking". The predicates and values in the body have been hand-picked to reflect the expected dimension and shape of object parts representing the part affordances. The part affordance AffordContain requires a primitive shape with a minimum dimension of

$[0.03, 0.03, 0.03]$, the largest of the primitive shapes, and is identified as a cylinder. While the part affordance AffordGrasp requires a primitive shape with a minimum dimension of $[0.01, 0.01, 0.05]$, a maximum dimension of $[0.05, 0.05, 1.0]$, a min ratio of 2, and is either identified as a cylinder or cuboid.

To classify all primitive shapes according to their part affordances, the query `?- affordance(ID, Affordance)` is used, and the corresponding ID number is matched through the part affordance label. The part affordance labels are used to determine where an object should be grasped using the relationships between part affordances and tasks that are stored in the OWL ontology.

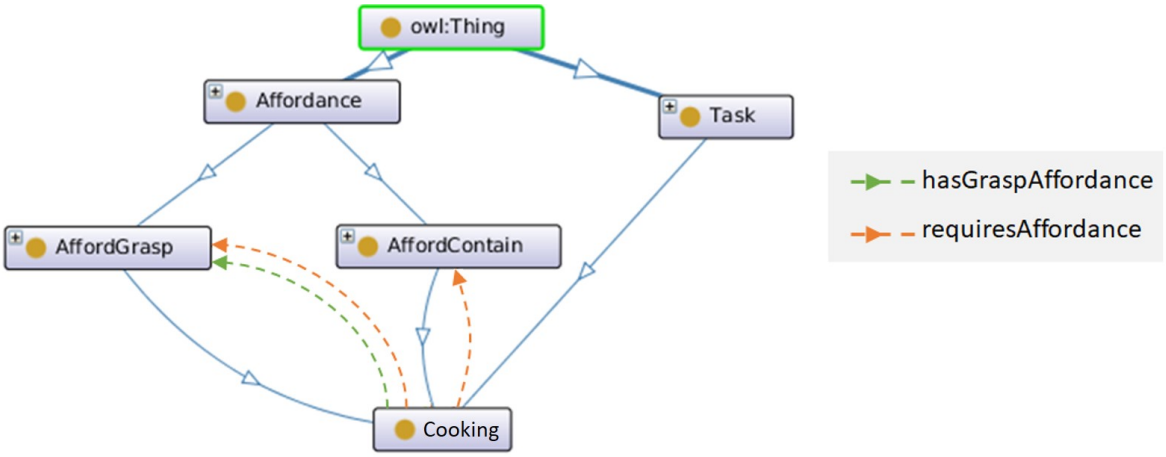

Figure 5.4: Overview of the relationships defined in the OWL ontology to understand which part affordances are required for a given task, such as ”cooking”.

Figure 5.4 shows part of the OWL ontology that depicts the relations associated with the task ”cooking”. Our ontology uses the class”*Affordance*”, which contains the part affordance labels (AffordGrasp and AffordContain), and the class *Task*, which contains the tasks (e.g., cooking). Between both classes, a relation is made to connect the type of part affordances that belong to a task. By using the object property *requiresAffordance*, a relation is made between a task and part affordance that indicates the part affordances an object must have to perform the task. For example, to perform the task ”cooking”, the primitive shapes must be labeled as AffordGrasp and AffordContain. In addition, to indicate the part affordance that should be grasped, the object property *hasGraspAffordance* links a task to a specific part affordance. The task ”cooking” is related to the part affordance AffordGrasp.

Using the Prolog logic rules in combination with the OWL ontology enables us to reuse the logic rules from one task to another. We can use the part affordance labels to define new relations between a new task and some existing part affordances in the OWL ontology. In Section 6.2, an experiment is shown on how this version can be extended to additional tasks.

5.3. Point Cloud Segmentation

The output of the logic-based approach, a selected primitive shape, does not include information about possible obstacles. While by using the object point cloud, we can approximate a suitable approach direction by selecting a grasp pose near the points of the partial object point cloud. Therefore, the region of the selected primitive shape is segmented to the points of the object point cloud.

The partial object point cloud is segmented by computing the shortest distance between the points, and the primitive shapes that have been recovered, to assign each point to its closest primitive shape. The Radial Euclidean Distance [23] computes the distance between a point to the surface of a primitive shape, shown in Equation (5.1). The function $F(\boldsymbol{p}, \boldsymbol{\theta})$ is the implicit superquadric function Equation (3.2), which takes a point $\boldsymbol{p}$ and the parameters of a primitive shape $\boldsymbol{\theta}$ as input. The distance between all primitive shapes is computed to determine which primitive shape a point belongs to.

$$distance(\boldsymbol{p}, \boldsymbol{\theta}) = ||\boldsymbol{p}|| \left| 1 - F(\boldsymbol{p}, \boldsymbol{\theta})^{\frac{\epsilon_1}{2}} \right| \tag{5.1}$$

Figure 5.5 provides an example of a primitive shape selected to perform a task, which is segmented into the points of the partial object point cloud.

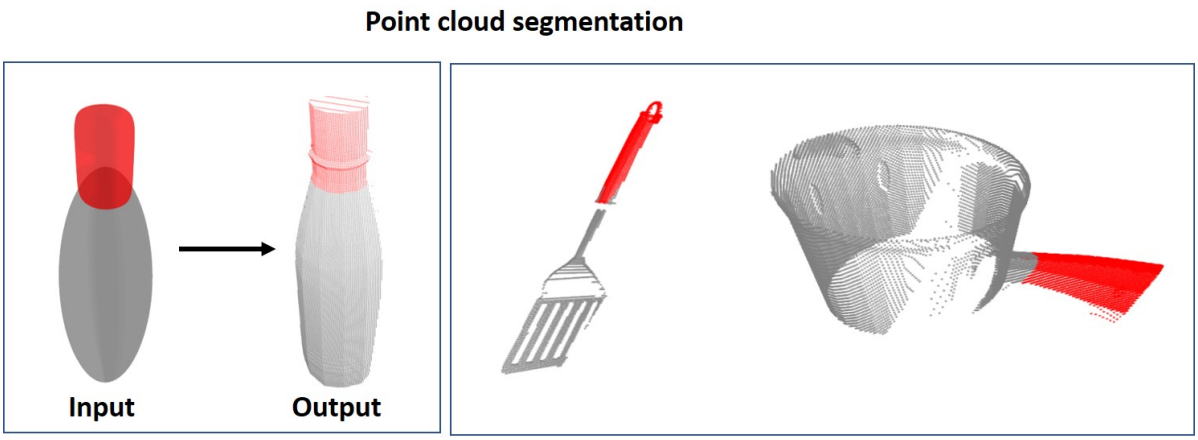

Figure 5.5: The selected primitive shape of a bottle, shown in red, is segmented onto the partial object point cloud (left). In addition, other partial object point clouds are segmented based on a selected primitive shape (Right).

5.4. Conclusion

In conclusion, we have proposed the SPaRGE framework that combines a data-driven, a superquadric algorithm, and a logic-based approach to generate a task-specific grasp pose. The data-driven approach takes a scene point cloud as input and generates multiple grasp poses. In addition, the logic-based approach takes multiple primitive shapes as input and identifies a suitable region for grasping. We have proposed two versions of the logic-based approach, which differ in how the logic rules reason over the primitive shapes. The logic-based approach enables a robot to reason at which region an object should be grasped.

This chapter has answered the research question: **"How to interpret semantic information from primitive shapes using symbolic reasoning?"**.

Reasoning Version-1 is a logic-based approach that uses Prolog logic rules to identify the relationship between primitive shapes and a specific task. This version has the advantage of modeling a task as a logical rule, allowing for constraints to be modified and new rules to be added without affecting the other tasks. Nevertheless, to create a new logical rule for a task, a thorough understanding of the set of primitive shapes to be recovered is essential.

Reasoning Version-2 is a logic-based approach that utilizes logic rules to classify primitive shapes into part affordance labels. Additionally, an OWL ontology is used to store relations between tasks and part affordances. One advantage of this approach is the ability to reuse the logic rules for part affordances when defining a new task. For example, if the part affordances required for a new task have already been specified as logic rules, the only thing required is to create new relations in the OWL ontology between the task and part affordances. However, creating logic rules that can generalize among multiple types of objects that share the same part affordance can be challenging.

Both reasoning Version-1 and Version-2 have their own merits, and both can be used for understanding the semantic information obtained from the primitive shapes. To evaluate their efficiency, we must assess how logic rules can identify suitable grasp regions (see Section 6.3). Furthermore, we should investigate the extensibility of the logic rules to new tasks (see Section 6.2).

6

Experiments & Results

In this chapter, the performance of the SPaRGE framework will be analyzed through multiple experiments. At first, the experimental setup section discusses the implementation details of the SPaRGE framework into the Robot Operating System (ROS), the object dataset used to validate the SPaRGE framework, and gives an overview of the tasks and part affordances that are used in the experiments. Next, the performance and generalizability of the SPaRGE framework will be evaluated using the following experiments:

- **Experiment 1: Extension To New Tasks** demonstrates how to extend both reasoning Version-1 and Version-2 to a novel task.
- **Experiment 2: Grasp Region Performance** evaluates the performance of the primitive recovery modules and the logic-based approach by analyzing how accurately a selected primitive shape matches the expected grasp region of a specific task.
- **Experiment 3: Grasp Pose Performance** evaluates the performance of the SPaRGE framework in finding a suitable grasp pose. A comparison is made between the data-driven approach and the SPaRGE framework.
- **Experiment 4: Computation Time Analysis** evaluates the computation time of the SPaRGE framework to produce a suitable grasp pose for a given task, and if additional tasks influence the total computation time.
- **Experiment 5: Real-world Demo** performs a real-world experiment, in which an object is placed in different orientations, whereby the robot's task is to grasp the object at the desired region.

6.1. Experimental Setup

In this section, the implementation of the SPaRGE framework in ROS is described. Additionally, the acquisition of the dataset used for experiments (2,3) is discussed. Lastly, the tasks and part affordances are discussed in relation to the object dataset.

6.1.1. ROS Implementation SPaRGE

ROS provides an open-source collection of frameworks for developing robotic software and facilitates collaboration among researchers in the robotics community [48]. The SPaRGE framework is implemented in ROS Melodic[1]. The implementation details are discussed in accordance with the SPaRGE framework outlined in Figure 5.1.

1. **Partial object point cloud**: The SPaRGE framework requires a scene point cloud as input, obtained from an RGB-D sensor. Because the superquadric algorithm requires only an object point cloud, the scene point cloud must be filtered. In our approach, the assumption is made that

[1] http://wiki.ros.org/melodic

objects are placed on a table in front of a robot. Therefore, within the scene point cloud, the table is removed using plane segmentation using the open3D open-source library [61].

2. **Data-driven approach**: The data-driven is implemented using the open-source Grasp Pose Detection library[2] from Pas *et al.* [43]. The code is integrated as a client-server node that takes as input the scene point cloud and outputs multiple grasp poses.

3. **Primitive recovery modules**: The recovery of primitives is performed using the superquadric algorithm[3] [31]. As a contribution to the existing code, which could only recover single primitives, we extended their code by implementing the hierarchical decomposition algorithm from MATLAB to Python. The superquadric algorithm is implemented as a client-server node, which takes as input the object point cloud and outputs (multiple) primitive shapes.

   Lastly, the MLP algorithm is trained to classify the recovered primitive shapes into cuboids, cylinders, or spheres (see Appendix A.1.2). This algorithm was prepared with Sci-kit learn [44], and is only compatible with Python3. Therefore, to make it function within the ROS Melodic framework, which is based on Python2, a client-server node was created to run the MLP algorithm on Python3.

4. **Logic-based approach**: The logic-based approach is implemented using components of the KnowRob framework[4]. KnowRob is an open-source Knowledge Representation and Reasoning (KRR) framework for robotic cognition [4]. The SPaRGE framework makes use of the KnowRob components SWI-Prolog and the MongoDB database. SWI-Prolog is an open-source implementation of the Prolog programming language and MongoDB is used to store and manage an OWL ontology. To make SWI-Prolog communicate with ROS, the Rosprolog[5] interface is used. In addition, Protégé is used for developing the OWL ontology [36], providing a graphical interface for designing and editing ontologies.

5. **Point cloud segmentation**: The fourth step of the SPaRGE framework segments the chosen primitive to the partial object point cloud. A Python function implements the algorithm described in Section 5.3.

6. **Grasp matching**: The last step of the SPaRGE framework consists of matching a grasp pose with a found grasp region. For each grasp pose the closest point on the object point cloud is computed. The grasp pose that has the highest score and is located in a suitable grasp region is chosen as the final grasp pose.

The steps above describe the main libraries and dependencies that are used by the SPaRGE framework. In additon to the SPaRGE framework, we utilize the open-source robotic manipulation platform Motion Planning Platform (MoveIt) [14] to perform the grasp pose with the robot TIAGo. MoveIt integrates the Open Motion Planning Library (OMPL) which consist of multiple planning algorithms. The code for the SPaRGE framework is made publicly available[6].

6.1.2. Dataset Acquisition

To evaluate the SPaRGE framework, a simulation dataset is created consisting of multiple partial object point clouds to evaluate the performance of the superquadric algorithm and logic-based approach. The partial object point clouds were generated by gathering a collection of 3D models and placing them in the Gazebo simulation environment[7] to obtain the point clouds. To further assess the efficacy of predicting the desired grasp region, the 3D models are manually segmented to identify the specific grasp regions for the given task. The following steps describe the process for creating the partial object point clouds and generating the ground truth data.

A total of 15 3D models were sourced from various open-source websites, which included household and retail store objects, resulting in five object classes, each containing three unique objects:

[2] https://github.com/atenpas/gpd_ros/
[3] https://github.com/bmlklwx/EMS-superquadric_fitting
[4] https://github.com/knowrob/knowrob
[5] https://github.com/knowrob/rosprolog
[6] https://github.com/stanzwinkels/grasp_generator
[7] https://gazebosim.org/home

(pan, ladle, hammer, spatula, and bottle), as shown in Figure 6.1. In addition, three object models were selected from the Yale-CMU-Barkeley (YCB) dataset [11], namely 027_skillet, 033_spatula, and 48_hammer. The YCB dataset consists of 3D models of everyday objects that are used to evaluate object recognition and manipulation frameworks. The three YCB object models are included to evaluate the generalizability of the SPaRGE framework to unknown objects. All 3D models are converted to the object datatypes .ply and .dae format. The .dae format is used to import and export 3D models that can be incorporated in the Gazebo simulations, while the .ply format is used to store 3D objects as a collection of points.

Figure 6.1: Overview of the different objects that are obtained from open-source websites. The dataset includes five object classes, namely bottle, pan, hammer, ladle, and spatula, with three objects in each class.

The second step involves obtaining multiple partial object point clouds from the 3D models. To achieve this, each object is placed on a table at two different heights (0.8 and 1.0 meters) in the Gazebo simulation environment. The objects are then perceived from the perspective of the robot TIAGo, which is equipped with an Intel Realsense D435 sensor, capable of capturing 3D data in the form of point clouds. These point clouds represent the objects in three-dimensional space, consisting of a set of points with X, Y, and Z coordinates within the coordinate system of the camera frame. Figure 6.2 illustrates TIAGo perceiving two objects placed on a table in front of it. The 3D models are sequentially placed in front of the robot, and the table is removed in the scene point cloud through plane segmentation to obtain a partial object point cloud. Ten partial object point clouds are captured for each 3D model, in random orientations and positions. Additionally, the homogeneous transformations between the camera frame and 3D models are stored to later on match the ground truth point cloud with the partial object point cloud in the experiments, which is used to evaluate the performance of the SPaRGE framework. The dataset consists of 180 partial object point clouds of everyday household and retail store objects in varying orientations.

At last, to create the ground truth data for evaluating the performance of the SPaRGE framework, the complete 3D models are manually labeled using the open-source 3D mesh processing software MeshLab [13]. This labeling process serves as the basis for evaluating the task region prediction of the framework in experiments 2. The labeling process involves segmenting the 3D models based on the concept of where an object should be grasped for a given task. This process is informed by our understanding of how objects should be used for different tasks. For instance, a pan is segmented into two parts, based on our understanding that it is used for the task ”handover” and ”cooking” that both require a different grasp region. Using color-coding, each point in the 3D models is segmented, and this segmentation is later used to evaluate the effectiveness of the selected primitive shapes in representing the task-specific grasp regions.

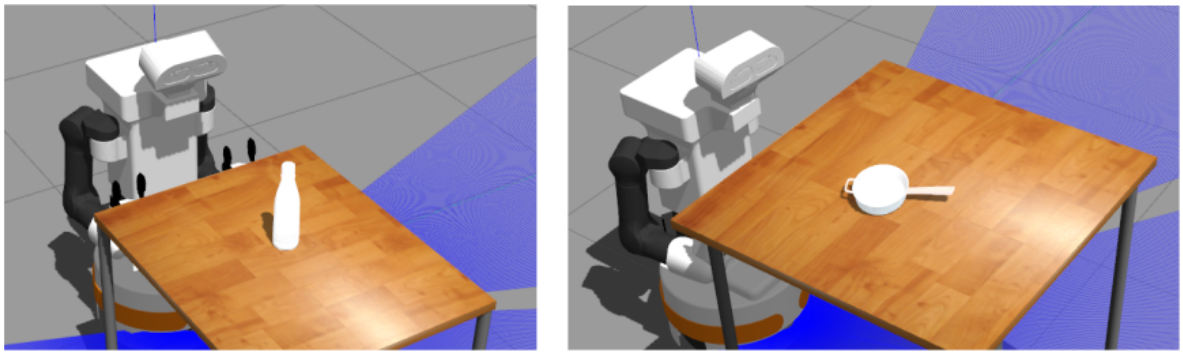

Figure 6.2: Example of the setup that is used to obtain the object dataset. In this example, TIAGo perceives bottle1 on a table with a height of 0.8 meters, and pan1 on a table of 1.0 meters high.

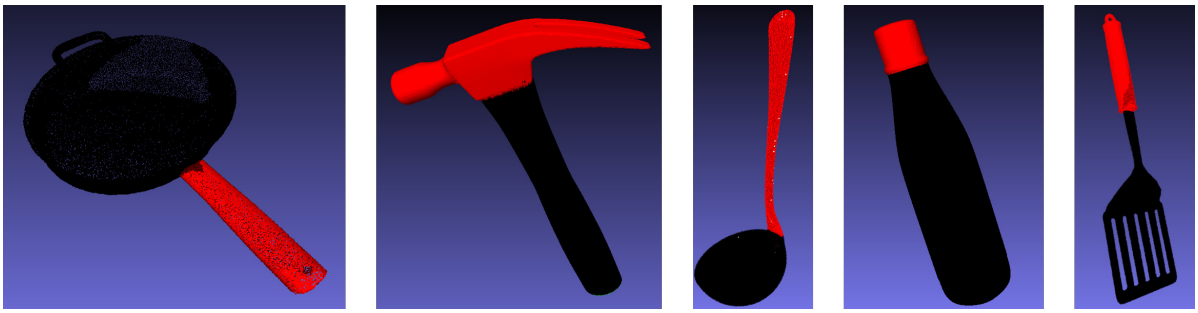

Figure 6.3: Segmented complete object point clouds that are used to evaluate the performance of the SPaRGE framework in finding a task-specific grasp region. Starting from left to right: {pan1, hammer2, ladle1, bottle1, spatula1}.

6.1.3. Overview Tasks

To evaluate the performance of the SPaRGE framework, logic rules for reasoning Version-1 and Version-2 have been created based on the description of the tasks and part affordance. The tasks and part affordance labels are taken from existing part affordance task-oriented grasp frameworks [32], [37]. An overview of the part affordance descriptions with objects that can provide these affordances is presented in Table 6.1.

Table 6.1: An overview of the part affordance labels with their respective descriptions and corresponding object classes.

part affordances	Description	Object Class
contain	Storing/holding liquid/objects	Pan, Bottle
pound	Striking other objects with a solid part	Hammer
scoop	A curved surface with a mouth for gathering soft material	Ladle
support	Holding other objects with a flat surface	Turner
wrap-grasp	Wrapping by hand for holding	Bottle
grasp	Enclosing by hand for manipulation	(Hammer, Pan, Ladle, Turner)

In reasoning Version-1, the logic rules are created based on the task description itself, whereas in Version-2, the part affordance labels are linked to a specific task. As presented in Table 6.2, an overview is provided of the tasks, their corresponding task descriptions, and the association between the part affordances. This table serves as a reference for understanding the relationship between the tasks and the part affordances in the experiments conducted using the SPaRGE framework.

In order to enable the robot to grasp objects appropriately for specific tasks, it is expected that a particular color in the segmented region displayed in Figure 6.3 will correspond to the desired grasp region for that task. For instance, to perform the ”cooking” task, the robot should be able to identify the handle of the pan and grasp it in that region. Thus, in the case of the object pan, the desired grasp region for the ”cooking” task is represented by the red region.

Table 6.2: An overview of the tasks, the part affordance labels associated with each task, and the object categories which are relevant to each task.

Task	Task description	part affordance	Object Class
Cooking	Plans, prepares and cooks food items.	grasp + contain	Pan
Hammering	To beat, drive, or shape with repeated blows of a hammer	grasp + pound	Hammer
Pouring	Send (a liquid, fluid, or anything in loose particles) flowing or falling, from one container to another, or into, over, or on something.	grasp/w-grasp + contain	Bottle
Handover	Give someone or something, or the responsibility for someone or something, to someone else.	grasp/w-grasp	All
Scooping	Picking up a quantity of food such as ice cream or ingredients such as flour.	grasp + scoop	Ladle
Turning	Change or transform an object into a different orientation.	support + grasp	Turner

6.2. Experiment 1: Extension To New Tasks

In this experiment, we aim to showcase the adaptability and scalability of the SPaRGE framework by introducing a new task. To achieve this, we will extend reasoning Version-1 and Version-2 using additional logic rules and relations defined in the OWL ontology. We will discuss the approach we took to create the logic rules for both reasoning versions, emphasizing the thought process behind enabling the SPaRGE framework to reason over semantic information. Through this experiment, we will showcase the flexibility of the SPaRGE framework in accommodating new tasks.

6.2.1. Setup

We extend the logic rules of reasoning Version-1 and Version-2 to include the new task of ”cutting”. The constraints for this task have been carefully defined, and the relevant relations have been established. It provides a theoretical explanation of both reasoning versions. It should be noted that both reasoning versions assumes that tasks are only assigned to objects that are capable of performing them. Thus, the logic rules for ”cutting” only consider objects that have the capability to cut through various materials.

6.2.2. Reasoning Version-1

The logic rules of reasoning Version-1 are extended to include the task ”cutting”. A systematic approach is used to add this new task using the following steps:

1. The first step is to define the constraints of a task, including the relevant objects, and determine the geometric factors that influence the desired grasp pose. To perform the task ”cutting”, we assume objects with a sharp edge that can cut through different materials. Next, we identify which types of objects meet the task description and can be found in a household and retail store environment. In this case, objects belonging to the object categories knives and scissors share geometric features such as a thin blade used to cut material and a handle to grasp the object. These objects also share the same property that the preferred grasp region to perform the task ”cutting” is not on the thin blade. Therefore, in the next steps, we aim to identify a primitive shape that characterizes itself as the handle of the object.

2. The second step focuses on defining the constraints for the logic rules. The head of the logic rule is formed using the predicate `AffordTask(ID, Surface, 'cutting')`, while the body of the rule is composed of concatenated predicates. The choice of predicates is based on our understanding of the task and associated objects, in this case, knives and scissors. The predicates that have been chosen for task ”cutting” are presented in Code 6.1.

Code 6.1: The logic rule of reasoning Version-1 for the task ”cutting”.

```
1  AffordTask(ID_1, 'Cutting') :-
2      primitive_values(ID_1, Dimension_1, Shape_Class_1),
3      min_dimension(Dimension_1, [Value]),
4      max_dimension(Dimension_1, [Value]),
5      min_ratio(Dimension_1, [Value]),
6      Shape_Class_1 == Shape,
7
8      primitive_values(ID_2, Dimension_2, _),
9      min_ratio(Dimension2, [Value]),
10     ID_1 \= ID_2.
```

The logic rule is designed to search for two primitive shapes that are not identical, indicated by the numbers *"1"* and *"2"*. To ensure that the logic rule finds the correct primitive shape, predicates such as `min_dimension()`, `max_dimension()`, and `min_ratio()` are used to set constraints on general dimensions expected of a handle (lines 3-5). In addition, to increase the likelihood of always grasping the correct primitive shape, the logic rule searches for another primitive that shoud reflect the dimension and shape of a thin blade, which is only constraint by the predicate `min_ratio()` (line 9).

3. The third step focuses on determining the values for the predicates, which should be able to generalize to objects with varying shapes and dimensions. To achieve this, a set of objects are used to find correlations between the recovered primitive shapes.

The described steps outline the process of extending the existing logic rules of reasoning Version-1 for the task "cutting" by adding a new logic rule that contains constraints specific to the task.

6.2.3. Reasoning Version-2

The logic rules and the OWL ontology that are used by reasoning Version-2 are extended to include the task "cutting". Similar to reasoning version-1, a systematic approach is used to add this new task using the followings steps:

1. The first step focses on identifying the relevant part affordances for the task "cutting". Given that scissors and knives are the suitable objects to perform this task, the affordances "AffordGrasp" and "AffordCut" are considered as relevant part affordances. The "AffordGrasp" affordance refers to the handle of an object, while the "AffordCut" affordance relates to a thin blade suitable for cutting material. It is important to note that an object might afford multiple part affordances, and in this scenario, both "AffordGrasp" and "AffordCut" affordances should be recognized by the logic rules to perform the task "cutting". The desired part affordance to grasp the object for the task is "AffordGrasp".

2. The second step focuses on defining the constraints for the logic rules to classify primitive shapes into part affordance labels. The head of the logic rules are formed using the predicates `Affordance(ID, 'AffordGrasp')` and `Affordance(ID, 'AffordCut')`. Similar to reasoning Version-1, the body of each logic rule contains multiple predicates that are chosen based on requirements of the part affordances. The logic rules for the AffordGrasp and AffordCut affordances are illustrated in Code 6.2 and Code 6.3, respectively.

Code 6.2: The logic rule of reasoning Version-2 for the part affordance "AffordGrasp".

```
1  Affordance(ID, 'AffordGrasp') :-
2      primitive_values(ID, Dimension, Shape_Class),
3      min_dimension(Dimension, [Value]),
4      max_dimension(Dimension, [Value]),
5      min_ratio(Dimension, [Value]),
6      max_ratio(Dimension, [Value]),
7      Shape_Class == Shape.
```

Reasoning Version-2 assumes that each primitive can represent a part affordance based on its geometric properties. For the part affordance ”AffordGrasp”, the predicates are chosen to represent the shape of a cylindrical handle.

Code 6.3: The logic rule of reasoning Version-2 for the part affordance ”AffordCut”.

```
Affordance(ID, 'AffordCut') :-
    primitive_values(ID, Dimension, Shape_Class),
    min_dimension(Dimension, [Value]),
    max_dimension(Dimension, [Value]),
    min_ratio(Dimension, [Value]),
    Shape_Class == Shape.
```

For the part affordance ”AffordCut” the predicates will resemble the thin blade of a knife or scissor. Therefore, the associated values will be chosen to constrain the dimensions and shape of the primitive shape.

3. The third step involves finding values for the predicates, which should be able to generalize to parts of objects that have the same part affordances.

4. The fourth step focuses on creating a relationship between a given task and the relevant part affordances, utilizing our OWL ontology. In cases where the part affordance labels or task are not currently represented within the OWL ontology, the part affordance labels are added to the Affordance class, while the task ”cooking” is included in the Task class. Next, the textitrequiresAffordance object property is used to establish a relationship between the task ”cutting” and the part affordances necessary to perform the task. Additionally, to enable a robot to identify the appropriate part affordance to grasp, the object property *hasGraspAffordance* is employed, connecting the task ”cutting” to the AffordGrasp part affordance. By establishing these relationships within the OWL ontology, the Prolog logic rules can interpret which classified primitive shape is most suitable for a given task. Thus, reasoning Version-2 enables the OWL ontology to contain the necessary relations required to support effective reasoning about task-part affordance relationships.

6.2.4. Discussion

Both reasoning Version-1 and Version-2 required the development of additional logic rules. For reasoning Version-1, a new logic rule was created to directly link the properties of the primitive shape to the task of ”cutting”. This version offers the advantage of allowing the inclusion of constraints between multiple recovered primitive shapes in the same rule. Moreover, specific constraints can be defined for a task, making it easier to create new logic rules for new tasks.

In the case of reasoning Version-2, the reuse of existing logic rules is possible if the part affordance label has already been defined. This approach uses the OWL ontology to create relations, thereby offering a user-friendly method to express new tasks. This makes the approach more generalizable to a wider range of objects. However, finding a suitable set of values for the predicates is more challenging, since it is difficult to find part affordance constraints that generalize among multiple objects sharing the same part affordances.

Both in reasoning Version-1 and Version-2, the logic rules are subject to bias towards objects that are familiar to the expert. As a result, for unknown objects with unique shapes, the logic rules may not able to identify a suitable primitive shape.

6.3. Experiment 2: Grasp Region Performance

This experiment compares the reasoning capabilities of the SPaRGE framework’s reasoning Version-1 and Version-2. The experiment’s primary focus is to assess the ability of the SPaRGE framework to predict task-specific regions on everyday objects. The framework’s performance is evaluated on an object dataset, which is described in detail in Section 6.1.2.

6.3.1. Setup

This study aimed to evaluate the accuracy of grasp region prediction using the SPaRGE framework by utilizing a dataset consisting of partial object point clouds of various objects, as discussed in Section 6.1.2. The accuracy was measured by computing the overlap between the predicted partial point clouds and the segmented ground truth point clouds. For each task and part affordance in Section 6.1.3, logic rules were created with predicates and their respective values (see Appendix A.2).

To evaluate the performance of reasoning Version-1 and Version-2, we selected 30 partial object point clouds from the dataset for each task, which were not from the YCB object dataset. Furthermore, to test the generalization of the logic rules for both versions, their performance was assessed on three objects from the YCB dataset. This experimental setup allowed for a comprehensive evaluation of the reasoning capabilities of the SPaRGE framework.

In this experiment, the accuracy and true positive rate (TPR) were computed for the predicted grasp region. The partial object point clouds were transformed into the base frame of the ground truth models using a homogeneous transformation. The accuracy score was computed by dividing the number of correctly labeled points by the total number of points after matching the points of the partial object point cloud with the closest point of the ground truth point cloud. The true positive rate was determined by taking the ratio of correctly identified positive points out of all positive points.

6.3.2. Results

The experiment's results are presented in Table 6.3 for the 15 objects in the dataset. The True/False columns in the table indicate how often a primitive shape is identified by the logic rules that meet the task's criteria. It was observed that both reasoning Version-1 and Version-2 could not find a primitive shape that meets the task criteria for all partial object point clouds. However, for partial object point clouds where a primitive was found, an average true positive rate of 81.2% was achieved for reasoning Version-1 and 86.0% for reasoning Version-2.

Table 6.3: For each task, 3 objects in 10 orientations are sampled to evaluate if a primitive shape is detected that could potentially fulfill a given task. Then, the segmented partial object point clouds are evaluated against the ground truth data to obtain the accuracy score and true positive rate (TPR).

Task	**Object Class**	**Version-1**	**Version-2**	**Version-1**	**Version-2**	**Version-1**	**Version-2**
		True/False		Accuracy Mean $\pm$ STD		TPR Mean $\pm$ STD	
Cooking	Pan	22/30	23/30	95% $\pm$ 3.2	94% $\pm$ 13	99% $\pm$ 0.4	89% $\pm$ 23
Hammering	Hammer	21/30	25/30	76% $\pm$ 19	61% $\pm$ 33	83% $\pm$ 23	62% $\pm$ 39
Pouring	Bottle	26/30	11/30	80% $\pm$ 11	87%$\pm$ 11	97% $\pm$ 4.0	97% $\pm$ 4.1
Scooping	Ladle	10/30	12/30	92% $\pm$ 20	90% $\pm$ 11	100% $\pm$ 0.0	95% $\pm$ 8.2
Turning	Spatula	17/30	11/30	83% $\pm$ 20	93% $\pm$ 14	81% $\pm$ 36	93% $\pm$ 14
Handover	Spatula	30/30	28/30	33% $\pm$ 23	75% $\pm$ 28	50% $\pm$ 50	79% $\pm$ 28
Handover	Bottle	30/30	23/30	86% $\pm$ 18	65% $\pm$ 33	53% $\pm$ 39	83% $\pm$ 17
Average		164/210	133/210	79.1	74.5	81.2	86.0

Furthermore, to assess the generalization capability of the SPaRGE framework, it was tested on the three objects from the YCB dataset. The results indicate that the SPaRGE framework achieved an average true positive rate of 69.0% for reasoning Version-1 and 68.7% for reasoning Version-2.

Table 6.4: For each task, an YCB object is placed in 10 unique orientations for which the accuracy score and true positive rate (TPR) are obtained.

Task	**Object Class**	**Version-1**	**Version-2**	**Version-1**	**Version-2**	**Version-1**	**Version-2**
		True/False		Accuracy Mean $\pm$ STD		TPR Mean $\pm$ STD	
Cooking	027_skillet	6/10	9/10	91% $\pm$ 7.3	90% $\pm$ 9.1	99% $\pm$ 0.4	64% $\pm$ 36
Hammering	048_hammer	10/10	10/10	68.4% $\pm$ 19	78% $\pm$ 16	81% $\pm$ 17	82% $\pm$ 15
Turning	033_spatula	7/10	7/10	62% $\pm$ 18	83% $\pm$ 7.8	27% $\pm$ 34	60% $\pm$ 11
Average		23/30	26/30	73.8	83.7	69.0	68.7

6.3.3. Discussion

The accuracy of identifying a primitive shape that satisfies a given task is affected by the orientation of the object from which the point cloud was captured, as seen in the True/False scores reported in Table 6.3. In addition, objects with simple shapes, like bottles and pans, were more likely to have a primitive shape detected that meets the task requirements. In contrast, objects with more complex shapes, like ladles, are more challenging to represent with primitive shapes.

The ”pouring” task showed a stark contrast in True/False scores between reasoning Version-1 and Version-2, with reasoning Version-1 detecting a primitive shape 26 out of 30 trials compared to only 11 times out of 30 trials for reasoning Version-2. The difference can be explained due to reasoning Version-2 using part affordance rules, which need to generalize to multiple objects with the same part affordance, while still selecting the correct primitive shape. This leads to a trade-off between the specificity of the logic rules and the framework's ability to generalize to unknown objects. Furthermore, a high TPR score is observed for most tasks in both reasoning Version-1 and Version-2. However, a low TPR score is observed for the ”handover” task in reasoning Version-1, indicating that the logic rules often selected the wrong primitive shape. In contrast, reasoning Version-2 showed a much higher TPR score when using the same partial object point clouds for the handover task.

The results of testing the SPaRGE framework on objects from the YCB dataset with similar geometric characteristics to the trained objects indicate high accuracy and TPR score performance, as reported in Table 6.4. This suggests that the SPaRGE framework can generalize well to unknown objects with similar geometric characteristics.

In conclusion, the findings suggest that the performance of the SPaRGE framework is affected by the shape of the primitive shapes, the design choices of the logic rules, and the orientation and complexity of the objects. The SPaRGE framework performs well when the logic rules are specific enough to represent the objects accurately. An overview of the process for recovering primitive shapes and selecting a suitable primitive shape for the robot to grasp the object can be found in Figure 6.4.

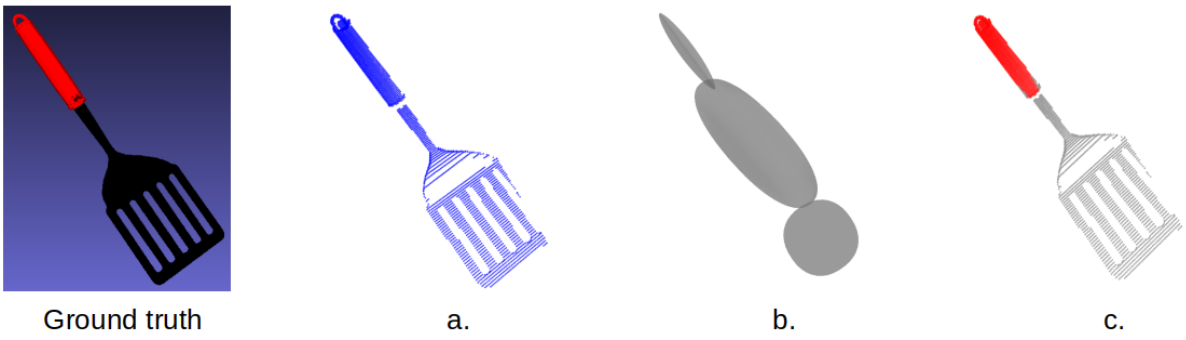

Figure 6.4: A spatula being recovered with primitive shapes, represented by (a) the partial point cloud, (b) the recovered primitive shapes, and (c) the resulting segmented partial object point cloud, with red indicated as the suitable grasp region.

6.4. Experiment 3: Grasp Pose Performance

This experiment compares the effectiveness of task-specific grasp poses generated by the SPaRGE framework with those generated by the data-driven approach. The data-driven approach relies on pre-trained knowledge to generate robust grasp poses, but it does not take into account the specific task requirements (see Section 3.1). In contrast, the SPaRGE framework provides reasoning capabilities to the data-driven approach, which allows it to generate a grasp pose while including the task requirements.

6.4.1. Setup

In this experiment, the grasp poses generated by the data-driven approach were evaluated based on their success/failure rate in finding a suitable grasp pose. The approach generated multiple grasp poses and assigned a probability score to each pose, indicating the likelihood of a successful grasp.

The grasp pose with the highest score was selected for evaluation. Similarly, for the SPaRGE framework, the closest point on the object point cloud was computed to choose the final grasp pose that had the highest score and was located in a suitable grasp region.

The performance was evaluated for three tasks, including cooking, handover, and pouring, using a pan and a bottle as objects. The experiment was conducted in simulation, and the objects were placed in ten different orientations on a table of 0.8 meters. To ensure that at least one or more grasp poses were found close to the suitable grasp region, the number of sampled grasp poses by the data-driven approach was set to 300 grasp poses.

To validate the grasp poses, both the data-driven approach and the SPaRGE framework were evaluated against the ground truth object model from Section 6.1.2. The validation was performed by determining whether the grasp pose was closest to a point that had been designated as a suitable grasp region. Only grasp poses meeting this criterion were considered successful.

6.4.2. Results

The results of the experiment are presented in Table 6.5. The resulting grasp poses from the data-driven approach rarely resulted in a successful grasp pose at the desired region, while the SPaRGE framework succeeded for every task more than 7 out of 10 times.

Table 6.5: Evaluation on the success/failure rate of selecting a grasp pose at the designated task-specific grasp region between the data-driven approach and the SPaRGE framework.

Task	Object	Data-driven	SPaRGE Version-1	SPaRGE Version-2
		Success/Failure		
Cook	Pan1	3/10	7/10	8/10
HandOver	Bottle1	2/10	8/10	7/10
Pour	Bottle1	8/10	7/10	9/10

6.4.3. Discussion

In this experiment, we evaluated the SPaRGE framework versus the data-driven approach. The data-driven approach is known to prioritize grasp poses that result in a robust grasp and tend to favor grasping an object at the body, which often leads to a wide or general grasp. Consequently, the data-driven approach performs poorly for the task ”handover”, which requires a grasp away from the body.

On the other hand, the SPaRGE framework can modify its grasp pose based on the task requirements. By utilizing Reasoning Version-1 or Version-2, the SPaRGE framework can choose a grasp pose from the data-driven approach within a suitable grasp region. As shown in Figure 6.5, the SPaRGE framework chooses a different grasp pose than the data-driven approach.

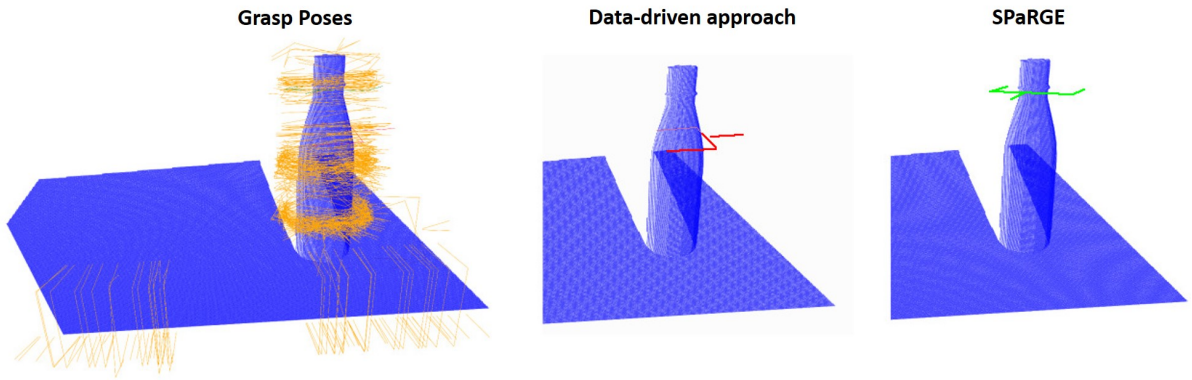

Figure 6.5: Comparison of grasp poses proposed by the data-driven approach and the SPaRGE framework. The Left illustrates all grasp poses proposed by the data-driven approach, followed in the middle by the selected grasp pose from the data-driven approach, and the right illustrates the grasp pose selected by the SPaRGE framework.

In conclusion, our experiment demonstrated that the SPaRGE framework outperforms the data-driven approach in terms of adapting grasp poses to specific task requirements. The SPaRGE framework can select a grasp pose based on task requirements by utilizing reasoning Version-1 or Version-2 and selecting a suitable grasp pose from the data-driven approach within a suitable grasp region. Our result highlights the potential of the SPaRGE framework to improve the performance of data-driven approaches for a variety of real-world tasks.

6.5. Experiment 4: Computation Time Analysis

In this experiment, we evaluated the computational cost of the SPaRGE framework by measuring the time required for the framework to search for a suitable grasp pose. Additionally, we aimed to examine how the computational complexity of reasoning Version-1 and Version-2 is affected by the inclusion of additional tasks.

6.5.1. Setup

To evaluate the computational cost of the SPaRGE framework, we conducted an experiment using a subset of the dataset created in Section 6.1.2. Specifically, we measured the computation time for three tasks with the objects: HandOver; *Spatula*, HandOver; *Bottle*, and Hammering; *Hammer*. For each task, a total of 20 partial point clouds were selected. This experiment only focuses on the computation from recovering primitive shapes to segmenting the partial object point cloud using the selected primitive shape. Obtaining the partial point cloud and detecting grasp poses with the data-driven approach were not included in this experiment.

The Prolog rules are implemented in SWI-Prolog, which uses a backtracking algorithm for searching its knowledge base. The engine attempts to match the query with the available facts and rules, and if it fails, it backtracks and tries another unification. The process continues until a solution is found or all options have been exhausted, resulting in query failure [58]. In addition, for reasoning Version-2, the OWL ontology is loaded into MongoDB, which enables SWI-Prolog to access and query it.

Additionally, we examined the SPaRGE framework's scalability by increasing the number of tasks in steps of [5, 10, 20, 50, 100, 200]. While increasing the number of tasks, we used the same task and object as the end goal. To evaluate reasoning Version-1, we added additional logic rules, while for reasoning Version-2, we established new relations between fabricated tasks and existing affordances in the OWL ontology. The experiment was conducted on a Lenovo Thinkpad p1 gen2 using an Intel(R) Core i7-9750H CPU @ 2.60GHz processor.

6.5.2. Results

The experiment results are presented in Table 6.6. The longest computation time was observed for the removal of overlapping primitive shapes, which took 4.3 seconds. The logic-based approach took 0.1 seconds for reasoning Version-1 and 0.2 seconds for reasoning Version-2. The total time to find a suitable grasp region takes respectively 9.3 and 9.4 seconds.

Table 6.6: Computation time of reasoning Version-1 and Version-2 of the SPaRGE framework.

	Superquadric Algorithm	**Overlap Removal**	**Shape Classif.**	**Logic-based Approach**	**Point cloud segmentation**	**Total Time**
			Mean $\pm$ STD [s]			
SPaRGE Version-1	2.3 ± 0.8	4.3 ± 1.5	$1.0 \pm 2.0e\text{-}4$	0.15 ± 0.01	1.5 ± 0.5	9.3
SPaRGE Version-2				0.25 ± 0.01		9.4

Furthermore, as depicted in Figure 6.6, there is no observed increase in the overall time when additional tasks are included in either reasoning Version-1 or Version-2.

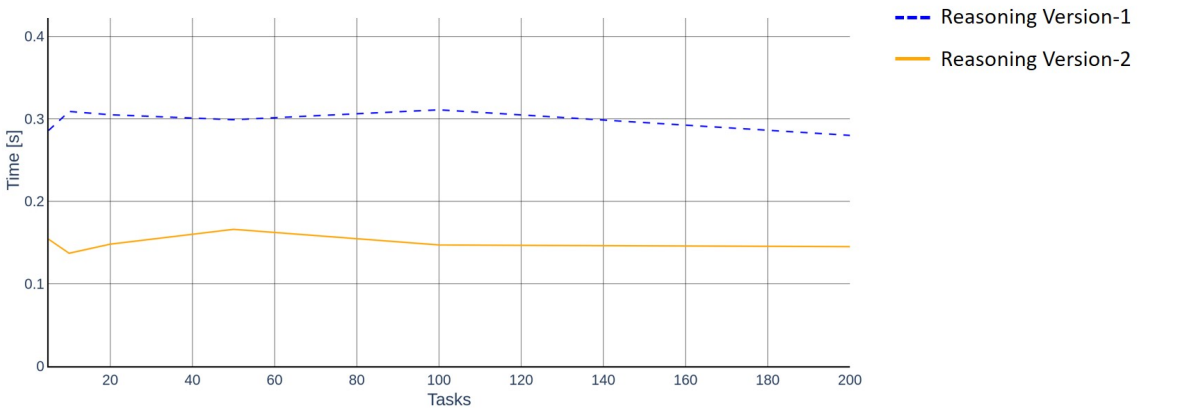

Figure 6.6: The computation time for reasoning Version-1 and Version-2 with having an increasing number of potential tasks. For Version-1 additional logic rules are added, while for Version-2 new relations are made in the OWL ontology.

6.5.3. Discussion

The results show that the superquadric algorithm (2.3 seconds) and removal of overlapping primitive shapes (4.3 seconds) takes up the majority of the computation time, with the overlap removal algorithm being the most time-consuming. The overlap removal algorithm uses a volume-point completion approach to determine the degree of overlap between primitive shapes, which is an operation that is computationally intensive and hence consumes the most time within the framework. In addition, it is noteworthy that the computational time for reasoning Version-2 is slightly longer than that of Version-1. This outcome is expected since reasoning Version-1 only executes a single logic rule, whereas reasoning Version-2 performs multiple logic rules to assign part affordance labels to the primitive shapes and requires an extra query to identify the most appropriate primitive shape for grasping.

Furthermore, as demonstrated in Figure 6.6, our experiment indicates that increasing the number of logic rules in Prolog or the number of relations to reason over in the OWL ontology does not have a significant impact on computational time. This outcome can be explained because our approach assumes that the object can perform the given task and the task itself is known beforehand. As a result, reasoning Version-1 only needs to verify a single logic rule that matches the task description, which saves computation time.

In conclusion, our evaluation of the SPaRGE framework showed that the current implementation takes 9.3 and 9.4 seconds for execution, which may not be suitable for real-time applications. Nevertheless, the logic-based approach demonstrated promising results, taking only 0.15 and 0.25 seconds, respectively. The time-consuming algorithms, such as the superquadric algorithm and the overlap removal algorithm, could be further optimized by exploring different approaches or techniques. These findings suggest that the SPaRGE framework has the potential for future development.

6.6. Experiment 5: Real-world Demo

In this experiment, the SPaRGE framework is evaluated by performing a grasp in a real-world scenario.

6.6.1. Setup

This experiment is conducted in collaboration with Artificial Intelligence for Retail Lab Delft (AIRLab Delft) in a retail store environment [50], using the mobile manipulator TIAGo[8]. TIAGo consists of a moveable base with two 7-DOF arms and a two-finger parallel gripper as an end-effector and uses a realsense D4355 camera to perceive the environment. In addition, a motion plan was computed and performed using MoveIt[9] motion planning framework.

The experiment aimed to test the robot's ability to perform a "handover" task, which required it to grasp a 1L Arizona bottle[10] placed in two distinct orientations (flat and upright) on a 0.8m high table. The experiment focused on identifying a preferred grasp region that would allow a human to take over the object from the robot.

6.6.2. Results

Figure 6.7 indicates that the framework was able to find a suitable grasp region for both an upright and a flat bottle position.

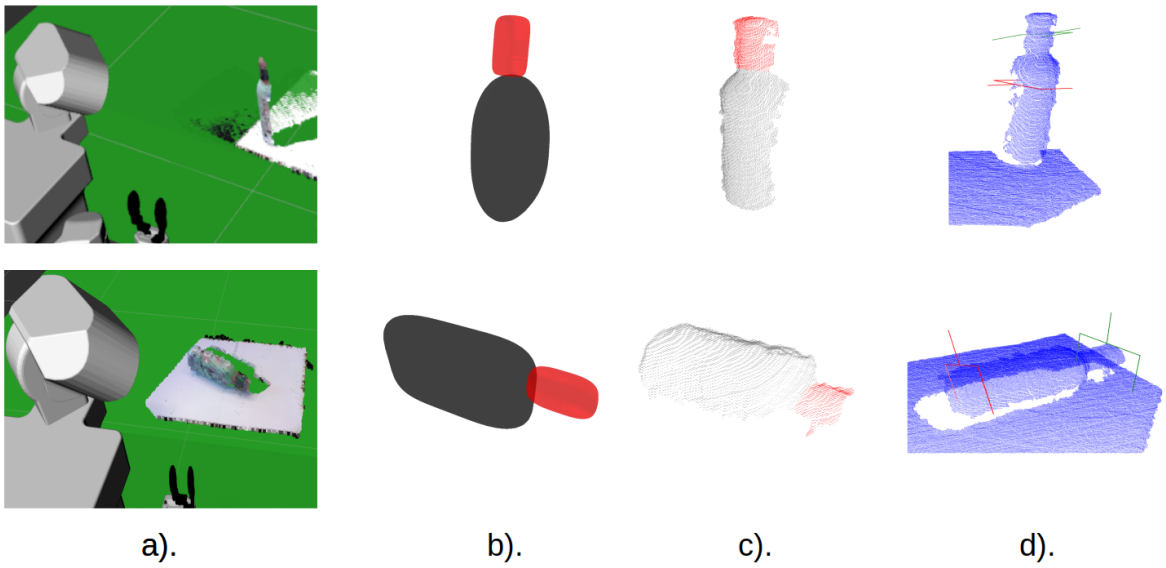

Figure 6.7: Step-by-step overview of finding a suitable grasp pose in a real-world experiment with the robot TIAGo. a). Perceiving the object as a scene point cloud, b). recovery of primitive shapes, c). reasoning with the logic-based approach to finding a suitable grasp region, and d). the proposed grasp pose by the SPaRGE framework in green, and the proposed grasp pose by the data-driven approach in red.

In addition, Figure 6.8 shows that the resulting grasp pose leads to TIAGo successfully grasping the object at the desired region.

6.6.3. Discussion

The experiment demonstrated that the SPaRGE framework is capable of reliably grasping an object at the desired grasp location for the 'handover' task, regardless of the object's orientation. By leveraging primitive shapes, the framework could identify suitable grasp regions and achieve successful grasping. This was illustrated in Figure 6.7 where the framework was able to recover accurate primitive shapes even in the presence of real-world noise.

[8] https://pal-robotics.com/
[9] https://moveit.ros.org/
[10] https://www.ah.nl/producten/product/wi490036/arizona-green-tea-with-honey

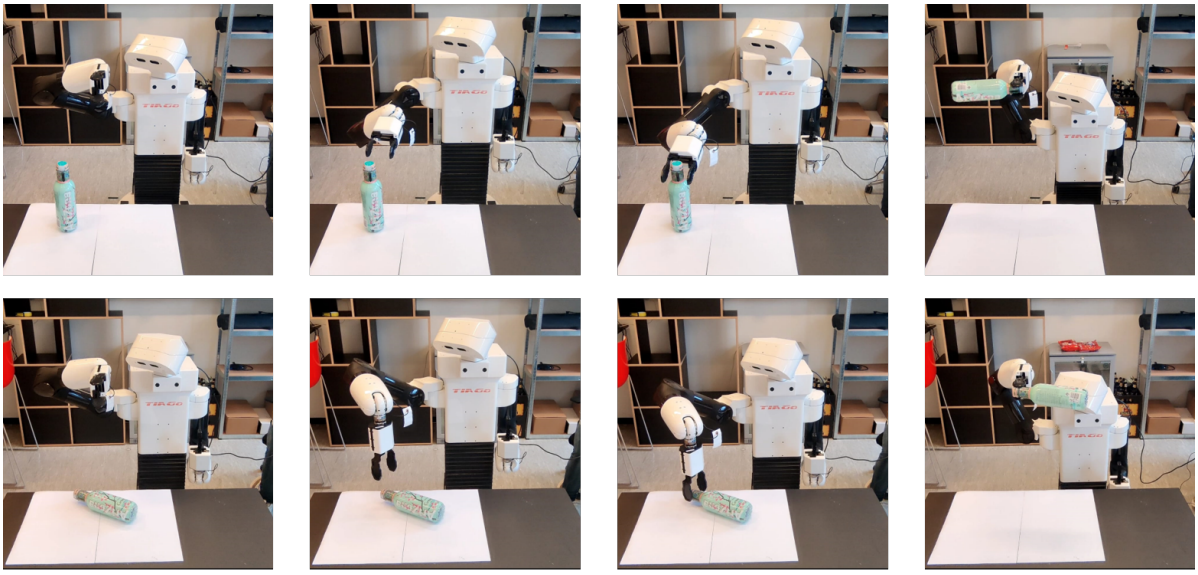

Figure 6.8: A real-world grasp using the grasp pose obtained from the SPaRGE framework using the robot TIAGo. a) TIAGo's initial predefined grasp position, b) a pre-grasp position in front of the object, followed by c) grasping the object using the grasp pose, and d) manipulating the object.

However, the experiment also revealed several challenges that must be addressed to improve the performance of the SPaRGE framework. One of these challenges was identifying the object as a potential collision object, which was addressed by introducing a pre-grasp position. Another challenge was the difficulty in finding a trajectory to the pre-grasp pose, which could be resolved by using both arms or a robot with a larger reach. Lastly, noise around the object caused by table points after the plane segmentation process made it difficult for the superquadric algorithm to recover accurate primitive shapes.

7

Discussion

The discussion evaluates the SPaRGE framework and answers the subquestions from Section 1.2. For each subquestion, the problems will be addressed in relation to the SPaRGE framework.

7.1. Primitive Recovery

How can a superquadric algorithm provide semantic information on everyday household and retail store objects?

The superquadric algorithm, proposed in Liu *et al.* [31], allowed for real-time recovery of primitive shapes on a partial object point cloud to represent objects as simple shapes. By decomposing the object's geometry into multiple primitive shapes, a robot can reason over the recovered primitive shapes to understand how to interact with the object. Because the superquadric algorithm uses the implicit superquadric function (see Equation (3.2)), the algorithm can represent a wide variety of objects, including those with complex shapes and features. This was seen in Section 6.3, where the superquadric algorithm had to represent complex objects (e.g., hammer, ladle, spatula) using primitive shapes. In addition, We can conclude from Section 6.3, where the objects were randomly oriented in front of the robot, that the algorithm can represent objects regardless of orientation accurately. This makes it applicable in various settings, such as household and retail store environments.

The main challenge of recovering primitive shapes from partial object point clouds is sparseness and noise, which can lead to a misrepresentation of the object's shape when represented by primitive shapes. Therefore, we optimized the parameters of the superquadric algorithm using a cost function to minimize the mismatch between the object's geometry and the recovered primitive shapes (see Appendix A.1.1). The parameters of the superquadric algorithm were tuned on ground truth labels that contained manually defined expected dimensions and spatial distances of the recovered primitive shapes to ensure that the recovered primitive shapes were consistent with the object geometry. The found parameters scaled well to objects from the YCB dataset, but the algorithm still has difficulty representing dense objects, such as a spatula. In addition, the parameter tuning restricts the number of clustered points required to recover a new primitive. This ensures that no primitive shapes are recovered in areas with high noise. However, it also means that small object features such as knobs, buttons, and needles that are represented by only a few points will not be identified by the superquadric algorithm.

In addition, another limitation of the superquadric algorithm is its limited vocabulary of recovered primitive shapes, which only includes convex shapes such as cuboids, cylinders, and spheres. Although previous studies suggest that these three primitive shapes can represent many everyday objects [6], increasing the range of recovered primitive shapes to include curved, tapered, or concave shapes would provide better identification of the semantic meaning and differences of object parts. For example, this expansion in vocabulary could help in recognizing the hollow handle of a mug or the bent shape of a banana [56]. As the limited vocabulary of the superquadric algorithm affects the available

semantic information, the designer of the logic rule should be aware of this constraint. A follow-up study should be conducted to include tapered, curved, and concave shapes in the superquadric algorithm to expand the vocabulary of recovered primitive shapes [54]. Furthermore, since the hierarchical algorithm recovers overlapping primitive shapes, it is time-consuming to filter and provides additional noise. A potential solution could be to use the ”split and merge approach” suggested by Chevalier *et al.* [12], which recovers multiple primitive shapes with no overlapping shapes.

7.2. Logic-based Approach

How to interpret semantic information from primitive shapes using symbolic reasoning?

This research proposes two symbolic reasoning versions using Prolog logic rules to interpret the semantic information provided by the superquadric algorithm. The logic rules are used to identify a recovered primitive shape that satisfies the requirements of a specific task. Both reasoning Version-1 and Version-2 utilize distinct approaches to reason over the available semantic information and fulfill task criteria.

Reasoning Version-1 Reasoning Version-1 utilizes task-specific logic rules to identify recovered primitive shapes that meet a set of constraints. These constraints consist of multiple predicates that evaluate whether a primitive shape satisfies the constraints. The benefit of using task-specific logic rules is that they can be adjusted whenever the task requirements change. However, creating a new logic rule for each new task can be time-consuming and requires expert knowledge.

Reasoning Version-2 In reasoning Version-2, the concept of part affordance was used to classify primitive shapes into part affordance labels based on the shape and size of the recovered primitive shapes. Using part affordances offers a significant advantage by associating tasks with part affordance labels in an OWL ontology. If the logic rules for the required part affordance already exist, only a new relationship must be made between the task and the part affordance label in the OWL ontology. This enables the reuse of existing part affordance logic rules, which is beneficial in terms of reducing time and effort. Furthermore, using an OWL ontology to describe the relations between tasks and part affordances provides a structured and consistent approach to describing tasks, which is particularly helpful for non-experts who may not have expertise in constructing Prolog logic rules.

However, since there is no universal rule for classifying particular primitive shapes into a specific type of part affordance, the constraints that define the part affordance labels are biased towards the experts’ conceptualization of part affordance and the potential set of objects. Additionally, choosing predicates and values for a part affordance label is a complex task that requires domain-specific knowledge about the objects and a clear understanding of the concepts of part affordance. For instance, the logic rules should be general enough to cover many objects that enable the same type of part affordance, while also excluding primitive shapes that do not afford the task.

A limitation of Reasoning Version-2 is that the OWL ontology is not based on a standardized approach. This affects the possibility of sharing semantic information between different robotic systems. In a recent study by Beßler *et al.* [5], the authors propose a novel conceptualization of affordances and their realization as a ”description logic ontology”. The resulting affordances are integrated into the robots’ understanding of the physical world, enabling them to make new decisions based on this information. By incorporating our ontology into their ontology, the part affordance information can be used to answer a set of questions highly relevant in the domain of autonomous robotics, which pertains to item usage and action potentialities. Several frameworks exist that aim to improve and create such a world representation using an ontology [5], [29], [34], [35], [41]. Therefore, for future research, it is essential to investigate how our ontology can be integrated into standardized robotic ontologies.

For a household and retail store environment, reasoning Version-1 and Version-2 have their own advantages, and the choice between them depends on the specific situation. Reasoning Version-2 has the advantage that changing the requirements of a task only requires changing the relation between the desired part affordance and task in the OWL ontology. However, if the preferred primitive shape changes, adjusting the logic rules for reasoning Version-2 becomes more challenging as part

affordances are not directly linked to a specific task. On the other hand, Reasoning Version-1 allows us to modify the values of the logic rules without affecting the logic rules of other tasks. This cannot be done with data-driven approaches that use affordance Neural Networks [16], where changing the requirements and labels requires new data and retraining of the Neural Network

7.3. SPaRGE Performance

How effective is the SPaRGE framework in detecting task-specific grasp regions on objects?

To evaluate the effectiveness of the SPaRGE framework, we tested it on household and retail store objects to predict task-specific grasp regions. Both Reasoning Version-1 and Reasoning Version-2 achieved high average true positive rates, with 81.2% and 86.0%, respectively (see section 6.3). This suggests that the logic rules can distinguish between different primitive shapes and find a shape that meets the task's requirements. However, we also found that the accuracy of the recovered primitive shapes is affected by the object's shape, with the superquadric algorithm performing better on objects with a more defined shape, such as pans and hammers.

It is important to note that the objects used in our dataset met the criteria for segmentation into primitive shapes, and more complex objects like mugs were not included. Moreover, the dataset was created based on our assumptions regarding what qualifies as a suitable task-specific grasp region. Despite these assumptions, we can conclude that the logic rules can successfully identify the requested grasp region on an object, which can subsequently be used to find task-specific grasp poses.

7.4. SPaRGE Generalization

How effective does the SPaRGE framework generalize to unknown objects?

The YCB dataset was used to evaluate the SPaRGE framework's ability to generalize to unknown objects. The results demonstrated that both versions of the logic-based approach effectively identified suitable grasp regions on unknown objects (see Section 6.3). However, the logic rules only generalize to unknown objects that have familiar shapes similar to the objects for which the logic rules were created. For objects that have distinct shapes and do not match the constraints of the logic rules, it may be necessary to modify or add new logic rules to include these objects.

7.5. SPaRGE Extendability

How can the SPaRGE framework be extended to new semantic information and tasks?

The use of Prolog logic rules in the SPaRGE framework offers a modular framework that can be extended easily with new logic rules. The was demonstrated in Section 6.5 by extending reasoning Version-1 and Version-2 with additional tasks. The results showed that adding new logic rules or relations in the OWL ontology did not increase the computation time. This is because the SPaRGE framework assumes that the object can perform the given task and the task is known beforehand, so it does not have to reason over all logic rules.

Although this study did not test the SPaRGE framework's extendability to additional semantic information, new predicates or relationships can be added to the OWL ontology to extend the logic rules with new semantic information. Expanding reasoning Version-1 can be done by adding predicates, while reasoning Version-2 can have new relationships defined in the OWL ontology. By introducing new relationships between concepts in the OWL ontology, we can define new constraints and incorporate new semantic information. However, adding more predicates or relationships to the rules can lead to decreased generalizability and increased complexity.

8

Conclusion & Future Work

8.1. Conclusion

This study aimed to add reasoning capabilities to a data-driven approach that allows it to perform task-specific grasps. The data-driven approach is able to propose robust grasp poses on multiple objects. However, it does not consider the task and therefore does not adapt the resulting grasp pose accordingly. Therefore, this research introduces the Shape Primitive and Reasoning Grasping Engine (SPaRGE) framework, which is a grasp framework with the reasoning capacity to determine at which region an object can be grasped and select a grasp pose at this region using the data-driven approach. This study has answered the following research question:

"How can the recovery of primitive shapes with symbolic reasoning allow for task-specific grasps on everyday objects, using a partial point cloud?"

The SPaRGE framework takes a partial object point cloud as input, which generates multiple primitive shapes that represent an object using simple 3D shapes such as cuboids, cylinders, and spheres. These primitive shapes are classified into their respective shape classes (cuboids, cylinders, or spheres) using an MLP algorithm. Then, a logic-based approach is employed, which uses the primitive shapes and their shape classes to determine which primitive shape satisfies the task requirement, leading to a desired grasp region. The logic-based approach is composed of logical rules, where each rule consists of geometric constraints to identify a primitive shape that matches the task's criteria. In this research, two versions of the logic-based approach have been developed: Reasoning Version-1 and Version-2. Reasoning Version-1 consists of Prolog logic rules, each of which contains multiple constraints designed for a specific task, aimed at identifying a primitive shape that meets the task's requirements. Reasoning Version-2 uses Prolog logic rules to classify primitive shapes into part affordance labels. To determine which primitive shape should be grasped according to a task, the relationships between the task and the part affordance labels are stored in an OWL ontology. This allows Reasoning Version-2 to identify a primitive shape that affords a specific part affordance. Both Reasoning Versions output a selected primitive shape, which results in a desired grasp region for the given task. Additionally, the selected primitive shape is segmented on the partial object point cloud to indicate the desired grasp region. This output is used by the data-driven approach to finding a grasp pose at the desired grasp region, resulting in a grasp pose that meets the requirements of the given task.

Following the discussions of the experiments in Chapter 6, we can now evaluate to what extent the SPaRGE framework fulfills the requirements R1-R5, as defined in Section 1.3.

R1 Ability to reason over semantic information.

- The SPaRGE framework has the ability to reason over semantic information using a logic-based approach. The superquadric algorithm provides semantic information in the form of primitive shapes such as cuboids, cylinders, and spheres. By using Prolog, which is based on First-Order logic rules, the SPaRGE framework can leverage the properties of these primitive shapes to

identify a specific shape that satisfies the requirements of a given task. Experiment 1 presented an example of how the logic rules can be applied to query the available semantic information and identify a primitive shape that meets the task requirements (see Section 6.2).

R2 Be applicable to objects from a single-shot object view.

- The SPaRGE framework's data-driven approach and use of the superquadric algorithm allow it to propose grasp poses from a partial point cloud. However, the superquadric algorithm requires a partial *object* point cloud as input, which can be more challenging to obtain because it requires the framework to extract object-specific points. Despite this challenge, the primitive shapes in simulation and the real-world experiments provide sufficient semantic information to determine a suitable grasp pose.

R3 Be applicable to objects in every orientation.

- The orientation from which a point cloud is perceived can impact the recovered primitive shapes, as viewing an object from a different perspective may result in a different primitive shape being recovered. However, the results of experiment 2, 3 and 5 (see Chapter 6) demonstrate that the SPaRGE framework can consistently identify primitive shapes that accurately represent the object. This enables the logic-based approach to select the most suitable primitive shape for generating a grasp region.

R4 Generalize to unknown objects.

- In experiment 2 (see Section 6.3), both reasoning Version-1 and Version-2 were generalized to objects from the YCB dataset. The average true positive rate was found to be 69.0% and 68.7%, which was lower compared to the performance achieved on objects for which the constraints of the logic rules were initially chosen. Nevertheless, the results indicate that the logic rules can generalize to objects with different shapes. However, the extent of generalization is limited to objects with familiar shapes that satisfy the constraints of the logic rules.

R5 Extend to new tasks and semantic information.

- Reasoning Version-1: experiment 1 (see section 6.2) demonstrates that the logic-based approach of the SPaRGE framework can be extended to include new tasks. Although extending the logic rules to new tasks requires expert knowledge on the possible set of objects and additional logic rules, it does not require retraining a neural network. This makes the logic rules flexible to new tasks.

- Reasoning Version-2: Extending the logic rules of part affordances is a complex task that requires domain-specific knowledge about objects and a clear understanding of the concept of part affordances. Therefore, instead of modifying the existing logic rules, a relation can be made between the new task and some existing part affordance logic rules in the OWL ontology. Reusing the part affordance logic rules for other tasks improves scalability.

The extension of the SPaRGE framework to new semantic information was not tested in our study. However, it is theoretically possible to extend the logic rules since a rule consists of multiple concatenated predicates that provide constraints based on the geometry of a primitive shape. Therefore, additional predicates can be added based on material properties or the preferred end-effector to perform a task. However, it is important to note that adding more requirements to the logic rules can increase complexity, making it more challenging to create and manage them effectively.

The SPaRGE framework has demonstrated its ability to reason over objects in order to find a suitable grasp pose, meeting the criteria for successful grasping. By leveraging a logic-based approach and the superquadric algorithm, the framework is able to reason over the geometry of an object from a partially visible object and select an appropriate grasp pose for a given task. Our results indicate that the SPaRGE framework is a promising approach for object grasping without requiring prior knowledge of the object's 3D model.

8.2. Future Work

This section identifies directions for future work based on the limitations that arose in our study, as described in earlier chapters. We propose expansions in several areas, including improving the primitive shape algorithm, extending the SPaRGE framework to incorporate additional semantic information, enhancing the reasoning capabilities of the SPaRGE framework, and conducting further experiments to validate its performance.

8.2.1. Improving the Primitive Algorithm

Currently, the superquadric algorithm is limited to recovering only three shapes: cuboids, cylinders, or spheres. However, some objects cannot be accurately represented by these shapes, such as a coffee mug with a curved handle. Therefore, future work could investigate how to extend the EMS algorithm to include tapered, curved, and concave shapes, thus expanding the vocabulary of the recovered primitive shapes [56].

Moreover, we propose further improving the hierarchical decomposition approach by using the split and merge approach proposed by Chevalier *et al.* [12] to enhance how well multiple primitive shapes represent an object. This approach ensures that resulting primitive shapes do not overlap and may better resemble the original geometry of the object. Furthermore, it eliminates the need to measure the overlap of the primitive shapes, resulting in improved computational efficiency for the SPaRGE framework.

8.2.2. Expanding Semantic Information

To enhance the performance of the SPaRGE framework, it is suggested to reason over additional semantic information beyond the dimensions and shape of a primitive shape.

One suggestion is to break down the primitive shapes further by reasoning over their respective surfaces. For example, a cylindrical shape can be decomposed into ”flat” or ”round” surfaces, which the logic-based approach can use to output more specific grasp regions for a given task. Moreover, surface information can provide the robot with knowledge of the suitable region for placing or stacking an object. For instance, it is preferable to place an object on the region with the largest flat surface. As shown in Figure 8.1, a bottle can be divided into multiple flat and round surfaces.

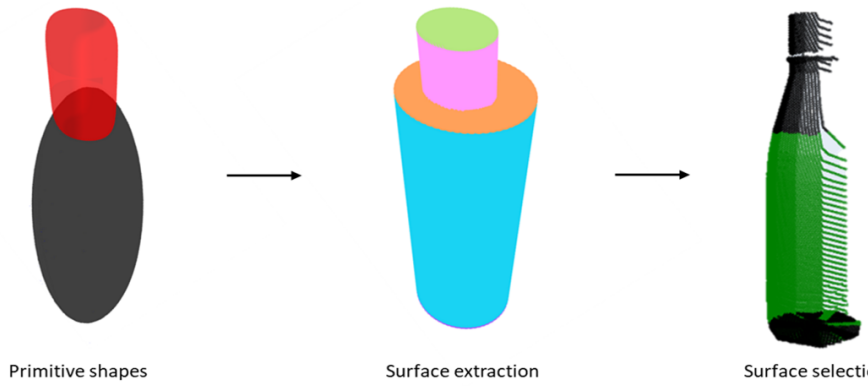

Figure 8.1: Potential future work, where a primitive shape (left) is further decomposed based on the ”flat” and ”round” surfaces (middle). These surfaces can specify a more specific grasp region (right).

To further enhance the SPaRGE framework, additional semantic information such as color, material, or object state can be integrated. These properties can influence the desired grasp pose, but integrating them can increase the complexity of the logic rules. Therefore, future research should investigate the impact of including more semantic information in the logic rules.

8.2.3. Expanding Reasoning Capabilities

To improve the reasoning capabilities of the SPaRGE framework, it would be beneficial to integrate the logic-based approach with a standardized ontology to enhance its world representation. This integration would enable the framework to use more sophisticated reasoning techniques, allowing robots

to perform advanced queries using the available semantic information. Using standardized ontologies makes the OWL ontology more easily understood and interpreted, resulting in more efficient and accurate information retrieval. Additionally, integrating the ontology into a standardized ontology framework would enhance its interoperability with other systems, increasing its potential for various other applications.

Moreover, the current implementation of the SPaRGE framework does not include constraints based on the type of end-effector used. However, the logic rules can be modified to include constraints specific to the requirements of a particular end-effector. For example, the dimensions of primitive shapes can be used to verify whether an object fits between parallel grippers, whereas for suction grippers, the area of the object's surface is of importance. Incorporating end-effector constraints would make the SPaRGE framework more versatile and applicable to a broader range of robotic systems.

8.2.4. Scene Extension

As discussed in Section 6.1.2, the objects used in our study were recorded on a table in front of the robot. To further evaluate the performance of the SPaRGE framework, it would be interesting to assess its performance when objects are placed in situations where they are partially occluded by obstacles, such as objects placed on a shelf or in a bin. To address this scenario, an additional method is required to generate an object point cloud from the scene point cloud. Future research should investigate how to obtain partial object point clouds from these scenes and evaluate how this affects the overall performance of the SPaRGE framework. This has the potential to expand the framework's applications to real-world scenarios where objects are not always fully visible and occlusions are common. Overall, investigating the performance of the SPaRGE framework under such scenarios and evaluating additional methods to generate object point clouds can contribute to improving its robustness and applicability in real-world applications.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[2] M. van Algemene Zaken, *Tekort aan personeel vraagt blijvende inzet*, Feb. 2023. [Online]. Available: https://www.rijksoverheid.nl/actueel/nieuws/2023/02/03/tekort-aan-personeel-vraagt-blijvende-inzet.

[3] L. Antanas, P. Moreno, M. Neumann, *et al.*, "Semantic and geometric reasoning for robotic grasping: A probabilistic logic approach," *Autonomous Robots*, vol. 43, no. 6, pp. 1393–1418, 2019.

[4] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels, "Know rob 2.0—a 2nd generation knowledge processing framework for cognition-enabled robotic agents," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 512–519.

[5] D. Beßler, R. Porzel, M. Pomarlan, M. Beetz, R. Malaka, and J. Bateman, "A formal model of affordances for flexible robotic task execution," in *ECAI 2020*, IOS Press, 2020, pp. 2425–2432.

[6] I. Biederman, "Recognition-by-components: A theory of human image understanding.," *Psychological review*, vol. 94, no. 2, p. 115, 1987.

[7] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, no. 6446, eaat8414, 2019. DOI: 10.1126/science.aat8414. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aat8414.

[8] C. Borst, M. Fischer, and G. Hirzinger, "Grasping the dice by dicing the grasp," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, IEEE, vol. 4, 2003, pp. 3692–3697.

[9] A. Boruah, N. M. Kakoty, and T. Ali, "Reasoning on objects' geometric shapes for prosthetic hand grasping," in *Proceedings of the Advances in Robotics 2019*, 2019, pp. 1–6.

[10] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[11] B. Calli, A. Singh, J. Bruce, *et al.*, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[12] L. Chevalier, F. Jaillet, and A. Baskurt, "Segmentation and superquadric modeling of 3d objects," 2003.

[13] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds., The Eurographics Association, 2008, ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[14] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[15] M. R. Cutkosky and R. D. Howe, "Human grasp choice and robotic grasp analysis," in *Dextrous robot hands*, Springer, 1990, pp. 5–31.

[16] T.-T. Do, A. Nguyen, and I. Reid, "Affordancenet: An end-to-end deep learning approach for object affordance detection," in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 5882–5889.

[17] K. Duncan, S. Sarkar, R. Alqasemi, and R. Dubey, "Multi-scale superquadric fitting for efficient shape and pose recovery of unknown objects," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4238–4243.

[18] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, 1996, pp. 226–231.

[19] J. G. Greeno, “Gibson’s affordances.,” 1994.

[20] R. Haschke, J. J. Steil, I. Steuwer, and H. Ritter, “Task-oriented quality measures for dextrous grasping,” in *2005 International Symposium on Computational Intelligence in Robotics and Automation*, IEEE, 2005, pp. 689–694.

[21] G. E. Hinton, “Connectionist learning procedures,” in *Machine learning*, Elsevier, 1990, pp. 555–610.

[22] T. Horikoshi and H. Kasahara, “3-d shape indexing language,” in *1990 Ninth Annual International Phoenix Conference on Computers and Communications*, IEEE Computer Society, 1990, pp. 493–494.

[23] A. Jaklic, A. Leonardis, F. Solina, and F. Solina, *Segmentation and recovery of superquadrics*. Springer Science & Business Media, 2000, vol. 20.

[24] Y. Kageyama, *Robot that stocks drinks is newest thing at the corner store*, Sep. 2022. [Online]. Available: `https://techxplore.com/news/2022-09-robot-stocks-corner.html`.

[25] K. Kirkpatrick, “Object recognition,” *Avian visual cognition*, vol. 43, 2001.

[26] M. Kokic, J. A. Stork, J. A. Haustein, and D. Kragic, “Affordance detection for task-specific grasping using deep learning,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, IEEE, 2017, pp. 91–98.

[27] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[28] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[29] C. Li and G. Tian, “Transferring the semantic constraints in human manipulation behaviors to robots,” *Applied Intelligence*, vol. 50, no. 6, pp. 1711–1724, 2020.

[30] Y. Lin, C. Tang, F.-J. Chu, and P. 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)*, IEEE, 2020, pp. 10 494–10 501.

[31] W. Liu, Y. Wu, S. Ruan, and G. S. Chirikjian, “Robust and accurate superquadric recovery: A probabilistic approach,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2676–2685.

[32] W. Liu, A. Daruna, and S. Chernova, “Cage: Context-aware grasping engine,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2550–2556.

[33] J. Mahler, J. Liang, S. Niyaz, *et al.*, “Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics,” *arXiv preprint arXiv:1703.09312*, 2017.

[34] W. M. Mohammed, P. M. Fresnillo, S. Vasudevan, Ž. Gosar, and J. L. M. Lastra, “An approach for modeling grasping configuration using ontology-based taxonomy,” in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, IEEE, vol. 1, 2020, pp. 507–513.

[35] B. Moldovan, P. Moreno, D. Nitti, J. Santos-Victor, and L. De Raedt, “Relational affordances for multiple-object manipulation,” *Autonomous Robots*, vol. 42, no. 1, pp. 19–44, 2018.

[36] M. A. Musen, “The protégé project: A look back and a look forward,” *AI matters*, vol. 1, no. 4, pp. 4–12, 2015.

[37] A. Myers, C. L. Teo, C. Fermüller, and Y. Aloimonos, “Affordance detection of tool parts from geometric features,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 1374–1381.

[38] R. Newbury, M. Gu, L. Chumbley, *et al.*, “Deep learning approaches to grasp synthesis: A review,” *arXiv preprint arXiv:2207.02556*, 2022.

[39] A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, “Object-based affordances detection with convolutional neural networks and dense conditional random fields,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[40] N. F. Noy, D. L. McGuinness, *et al.*, *Ontology development 101: A guide to creating your first ontology*, 2001.

[41] A. Olivares-Alarcos, D. Beßler, A. Khamis, *et al.*, "A review and comparison of ontology-based approaches to robot autonomy," *The Knowledge Engineering Review*, vol. 34, 2019.

[42] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems," in *Proceedings of the 2020 genetic and evolutionary computation conference*, 2020, pp. 533–541.

[43] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] P. K. Prasad and W. Ertel, "Knowledge acquisition and reasoning systems for service robots: A short review of the state of the art," in *2020 5th International Conference on Robotics and Automation Engineering (ICRAE)*, IEEE, 2020, pp. 36–45.

[46] M. Prats, P. J. Sanz, and A. P. Del Pobil, "Task-oriented grasping using hand preshapes and task frames," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 1794–1799.

[47] K. Qian, X. Jing, Y. Duan, *et al.*, "Grasp pose detection with affordance-based task constraint learning in single-view point clouds," *Journal of Intelligent & Robotic Systems*, vol. 100, pp. 145–163, 2020.

[48] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[49] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3d object grasp synthesis algorithms," *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 326–336, 2012.

[50] C. Salmi, *AIRLab Delft*. [Online]. Available: `http://airlab.3me.tudelft.nl/`.

[51] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*. Springer, 2008, vol. 200.

[52] J. Šircelj, T. Oblak, K. Grm, *et al.*, "Segmentation and recovery of superquadric models using convolutional neural networks," *arXiv preprint arXiv:2001.10504*, 2020.

[53] B. Smith, "Ontology," in *The furniture of the world*, Brill, 2012, pp. 47–68.

[54] F. Solina and R. Bajcsy, "Recovery of parametric models from range images: The case for superquadrics with global deformations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 2, pp. 131–147, 1990.

[55] M. Sparkes, *Robotic nurse can dress a mannequin in a hospital gown*, Apr. 2022. [Online]. Available: `https://www.newscientist.com/article/2315052-robotic-nurse-can-dress-a-mannequin-in-a-hospital-gown/`.

[56] T. Torii and M. 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)*, IEEE, 2018, pp. 580–585.

[57] E. Tosello, Z. Fan, and E. Pagello, "A semantic knowledge base for cognitive robotics manipulation," *University of Padova*, 2016.

[58] J. Wielemaker, "Swi-prolog 5.4. 1 reference manual," *Dept. of Social Science Information (SWI), University of Amsterdam, Roeterstraat*, vol. 15, p. 1018, 2002.

[59] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "Swi-prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.

[60] T.-F. Wu, C.-J. Lin, and R. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[61] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

A

Appendix

A.1. Additional experiments

A.1.1. Experiment 1: Parameter Selection Superquadric Algorithm

In this section, we aim to explore the optimal settings for the superquadric algorithm to accurately recover primitive shapes that represent an object's geometry. The initial settings chosen by the authors in Liu *et al.* [32] were not optimized to recover primitive shapes on partial object point clouds. They only minimized the distance between a primitive shape and a set of points. However, to reason effectively over the recovered primitive shape, it is equally important that the primitive shape matches the original geometry of the object. Therefore, we employ hyperparameter tuning to find the optimal setting that minimizes the distance between a set of points while accurately representing the original geometry of the object. To perform hyperparameter tuning, we create a cost function that computes the difference between the recovered primitive shape and the expected sizes, as well as the relative spatial differences. The details of recovering multiple primitive shapes on a partial object point cloud are discussed in Section 3.2. We aim to ensure that the found settings generalize across a wide range of objects, and hence, we perform experiments on objects from the object dataset described in Section 6.1.2.

Parameters

A total of 8 parameters influence the outcome of the superquadric algorithm. These parameters influence the shape, size, and spatial relationship of the recovered primitive shape. The algorithm consists of the following 8 parameters: The 'Outlier Ratio' is the prior probability [0.0, 1.0]. 'Max Iteration EM' is the maximum number of EM iterations [0, 30]. 'Tolerance EM' is the absolute tolerance of EM [1e-4, 1e-2]. 'Relative Tolerance EM' is the relative tolerance of EM [0.01, 1]. 'MaxiSwitch' is the maximum number of switches the S step is allowed to perform [1,8]. 'Adaptive Upperbound' introduces an upper bound to restrict the volume of a primitive [True, False]. 'Eps' sets the range for sampling outliers in DBSCAN [0.001, 0.1], and 'MinPoints' is the threshold for the minimum number of required points to recover a new primitive shape [50, 500]. The reader is referred to Liu *et al.* [31] for a better understanding of the parameters and method to recover a primitive shape.

Cost Function

To find the optimal setting for the superquadric algorithm, we use a cost function that computes the difference in size and spatial distance between the recovered primitive shape and the expected primitive shape. By minimizing the cost function using hyperparameter tuning, the best setting can be found.

The size error is computed as the difference between the size of a recovered primitive shape and an expected primitive shape. The main challenge is that the size parameters $\alpha_1, \alpha_2, \alpha_3 = \boldsymbol{\alpha}$ of a primitive shape do not directly match the order of the expected size $s_x, s_y, s_z = \boldsymbol{s}$, thus $\alpha_1 \neq s_x, \alpha_2 \neq s_y, \alpha_3 \neq s_z$. However, it is assumed that the combination that results in the minimum size error is the correct combination. Therefore, we use the Hungarian algorithm to find the minimum size error, which is used to solve the linear assignment problem [27]. The Hungarian algorithm takes a cost matrix of the different sizes between the recovered primitive shape and the expected sizes as input. Each individual cost is

computed by taking the sum of the absolute values of the size differences. The Hungarian algorithm determines the optimal order of parameters for the recovered primitive shape so that the difference between the expected and actual sizes is minimized. Here, $\boldsymbol{\alpha}_i$ represents the size parameters of a recovered primitive shape, while $\boldsymbol{s}_j$ indicates the expected primitive shape from a set of expected primitive shapes. The outcome of the Hungarian algorithm is a single cost that indicates the total size difference between a set of recovered primitive shapes and the expected sizes. The value is normalized using the hyperbolic tangent function, which maps the value onto a range from [0,1]. Equation (A.1) shows the equation to compute the cost function for the size error.

$$error\_size(\boldsymbol{\alpha}, \boldsymbol{s}) = \tanh\left(\min \text{Hungarian}\left(|\boldsymbol{\alpha}_i - \boldsymbol{s}_j|\right)\right) \tag{A.1}$$

In addition, the spatial distance is included in the cost function to improve the positioning of recovered primitive shapes relative to each other. To compute the spatial distance, the partial object point clouds on which the primitive shapes are recovered are transformed to the origin frame, eliminating any rotation or translation. This allows the ground truth values to be compared with the recovered primitive shape. To compute the spatial error between two recovered primitive shapes, the difference is taken, resulting in the spatial distance between two recovered primitive shapes denoted by $\{d_x, d_y, d_z\} = \boldsymbol{d}$. Whereas the ground truth spatial distance is denoted by $\{d'_x, d'_y, d'_z\} = \boldsymbol{d}'$. To compute the difference between the spatial distance of the recovered primitive shape and the expected primitive shape, the difference is summed to find the total offset. In Equation (A.2), the cost function for the spatial distance is shown. The resulting spatial error is normalized using the hyperbolic tangent function, which maps the value onto a range from [0,1].

$$error\_spatial(\boldsymbol{d}, \boldsymbol{d}') = \tanh\left(\sum |\boldsymbol{d} - \boldsymbol{d}'|\right) \tag{A.2}$$

The total cost is a combination of the size error and the spatial distance error, as seen in Equation (A.3). These two errors are given equal weight, meaning that the total error is determined by taking the average of the two. For partial object point clouds where the number of recovered primitive shape is less or more than 2, a cost is assigned of ”1”.

$$cost\_function(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{d}, \boldsymbol{d}') = \frac{size\_error(\boldsymbol{a}, \boldsymbol{s}) + spatial\_error(\boldsymbol{d}, \boldsymbol{d}')}{2} \tag{A.3}$$

The resulting cost function measures how well a primitive shape represents the original geometry of an object. However, to find a setting that generalizes to multiple objects, the average is taken over multiple objects. This value will be used for hyperparameter tuning to find a setting that generalizes among multiple objects.

Experiment Setup

The hyperparameter tuning is performed on 4 partial object point clouds from every unique object in the created object dataset (see Section 6.1.2), resulting in a total of 60 partial object point clouds. For each unique object, ground truth values were created based on the expected primitive shapes and their respective spatial distance.

The hyperparameter-tuning approach uses the Tree-structured Parzen Estimator algorithm (*TPE-Sampler*) to optimize the set of hyperparameters [42]. The algorithm is available through the hyperparameter optimization framework *Optuna*[1], which works well with optimizing mathematical models [1]. For the batch of 60 unique partial point clouds, 300 trials are conducted to minimize the cost function.

In addition, hyperparameter tuning was performed on a set of real object point clouds. This experiment was conducted on 10 partial point clouds of a bottle. However, since the homogeneous transformation between the object and the camera frame is unknown, the spatial error cannot be computed. Therefore, for the real-world experiment, the cost function only consists of the size error.

Results

We present the result of the hyperparameter tuning in Table A.1. The optimization history of all trials can be visualized in Figure A.1. For the simulation experiment, the lowest cost function was achieved in trial 113, with a cost of 0.2.

[1] https://optuna.org/

Table A.1: An overview of the parameters, and the resulting values of the superquadric algorithm using hyperparameter tuning.

Parameter	Description	Range	Simulation	Real-world
OutlierRatio	Prior outlier probability	[0.0, 1.0]	0.965	0.996
MaxIterationEM	Maximum number of EM iterations	[10, 30]	18	18
ToleranceEM	Absolute tolerance of EM	[1e-4, 1e-2]	0.00468	0.00430
RelativeToleranceEM	Relative tolerance of EM	[1e-2, 1]	0.867	0.804
MaxiSwitch	Maximum number of switches allowed	[1, 8]	2	2
AdaptiveUpperBound	Introduces adaptive upper bound to restrict the volume of SQ	True/False	False	True
Eps	Tuning the distance for DBScan	[0.001, 0.1]	0.0131	0.0397
MinPoints	Threshold for minimum required leftover point clouds	[50, 500]	100	150

Discussion
In Figure A.1, a decrease in the cost function can be seen, indicating that the hyperparameter tuning had a positive effect on the representation of objects via primitive shapes. Thanks to hyperparameter tuning, someone without prior knowledge of the superquadric algorithm can find an appropriate setting for any object dataset. It can be further observed that for certain settings, the cost is ”1”, suggesting that for many of the 60 object point clouds, either too few or too many primitive shapes were recovered. The recovery of too few or too many primitive shapes also influences the cost of other trials. In addition, a high cost may also be attributed to the sparseness of the partial object point cloud, which can result in a primitive shape that poorly represents the object.

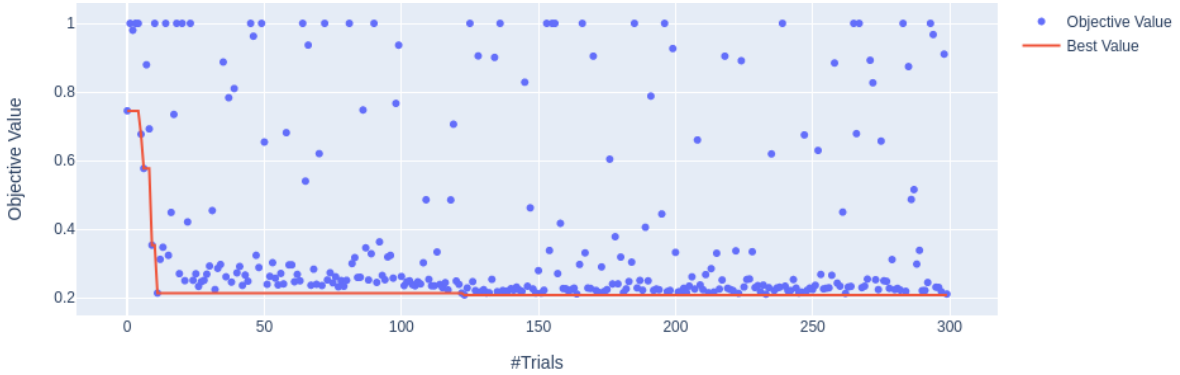

Figure A.1: Optimization history of all trials in the study over the selected range of parameters to reduce the cost function. Trial 113 yielded in the optimal result.

From Table A.1, both the outcome of the simulation and real-world settings are almost similar. A difference is seen for the Epsilon (Eps) values 0.0131 and 0.0397. This is because the data points in the simulation are usually more evenly spread out and there are fewer outliers. Therefore, the epsilon value that defines the distance between two points to be considered as neighbors can be set lower than in the real-world scenario.

In conclusion, using the settings found in this experiment minimizes the error between the size and spatial distance of the recovered primitive shape. Furthermore, the found settings in simulation cannot be generalized to real-world objects. The simulation settings are used for all experiments in simulation, while the real-world settings are used for the real-world demo with TIAGo.

A.1.2. Experiment 2: Shape Classification

In this experiment, we compare the performance of four shape classification approaches to label primitive shapes as a *cuboid*, *cylinder*, or *sphere*. These labels will enhance the SPaRGE framework by providing additional semantic information that can be used to identify suitable grasp regions more ac-

curately.

Setup
We conducted experiments in simulation by creating a synthetic dataset consisting of 550 primitive shapes that were recovered using the implicit superquadric function (see Equation (3.2)). This dataset was created by sampling random primitive parameters $(\epsilon_1, \epsilon_2, \alpha_1, \alpha_2, \alpha_3)$. The shape parameters ϵ_1, ϵ_2 have a range of $(0.1, 2.0)$, whereas the size parameters $\alpha_1, \alpha_2, \alpha_3$ have a range of $[0.01, 0.15]$. The parameters were randomly sampled and labeled by visualizing the primitive shape using the implicit superquadric function. The dataset consists of 105 cuboids, 335 cylinders, and 60 spheres. Additionally, the parameters were normalized due to the shape parameters being of a larger magnitude than the size parameters. The dataset was randomly split into 412 training samples and 138 test samples.

To estimate the correct shape label, four different multi-classification algorithms were compared. The first algorithm is a heuristic approach that uses linear equations to classify the primitive shapes (see Equation (A.4), A.5, A.6). The constraints are based on a modified version of Horikoshi and Kasahara [22]. This approach uses both shape and size to classify primitive shapes into their respective labels. The parameters of the heuristic approach were chosen manually. The shape constraints used by the heuristic approach are visualized in Figure A.2.

$$cuboid = \begin{cases} 0 \leq \epsilon_1, \epsilon_2 \leq 0.7, \\ -\epsilon_1 - \epsilon_2 + 0.7 \geq 0 \end{cases} \quad or \quad \begin{matrix} 0 \leq \epsilon_1 \leq 0.7,\ 1.3 \leq \epsilon_2 \leq 2.0, \\ -\epsilon_1 + \epsilon_2 - 1.3 \geq 0 \end{matrix} \tag{A.4}$$

$$cylindrical = \begin{cases} -\epsilon_1 - \epsilon_2 + 0.7 < 0, \\ \epsilon_1 - \epsilon_2 + 1.3 > 0, \\ 0 \leq \{\epsilon_1, \epsilon_2\} \leq 0.9, \\ 1.1 \leq \{\epsilon_1, \epsilon_2\} \leq 2, \\ \left(\frac{\max\{\alpha_1, \alpha_2, \alpha_3\}}{\min\{\alpha_1, \alpha_2, \alpha_3\}}\right) > 1.3 \end{cases} \tag{A.5}$$

$$spherical = \begin{cases} 0.9 < \{\epsilon_1, \epsilon_2\} < 1.1, \\ \left(\frac{\max\{\alpha_1, \alpha_2, \alpha_3\}}{\min\{\alpha_1, \alpha_2, \alpha_3\}}\right) \leq 1.3 \end{cases} \tag{A.6}$$

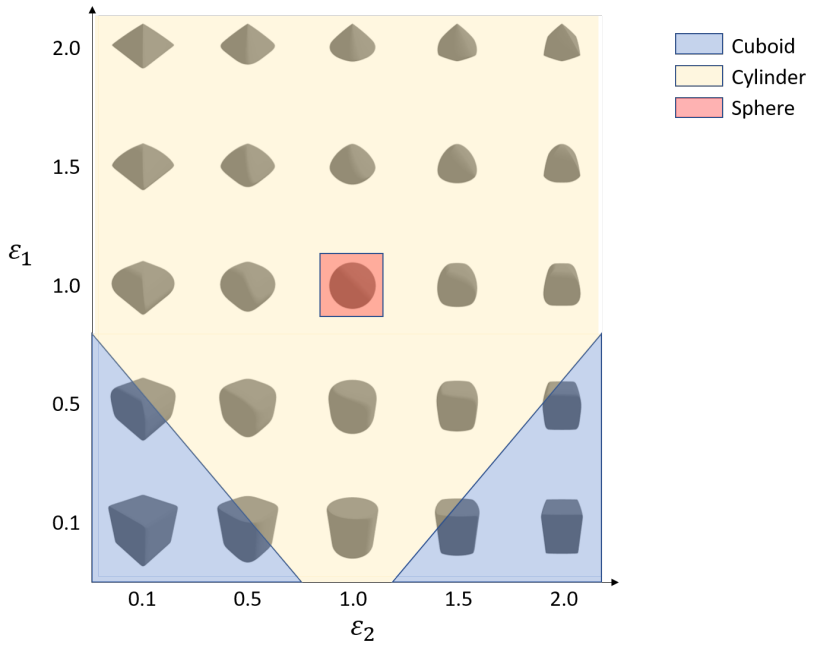

Figure A.2: Illustration of how the heuristic approach segments primitive shapes into cuboids, cylinders, and spheres based on the shape parameters $\{\epsilon_1, \epsilon_2\}$

The other three approaches are based on common machine learning techniques: Support Vector

Machine (SVM), Random Forest, and Multi-layer Perceptron (MLP). These are all supervised learning methods that use labeled data to train a model. To achieve the best results, these models require parameter tuning. To find the optimal parameters, a grid search is performed in combination with a 5-fold cross-validation approach using the training dataset. This involves splitting the data into five partitions, with four partitions used for training and the remaining partition for testing. This process is repeated five times, with each partition being used as the testing set once. The results are then averaged to obtain the best possible model parameters. The machine learning models, cross-validation, and grid-search techniques are implemented using the Scikit-learn library [44].

The following hyperparameters are used by the machine learning techniques:

- In the case of a Support Vector Machine (SVM), the hyperparameters include the kernel type ('Linear', 'Polynomial' (Poly), or 'Radial Basis Kernel' (RDF)), the regularization parameter (C) that controls the width of the margin around the decision boundary and helps to prevent overfitting $\{1, 10, 100, 100\}$, the gamma parameter which is a kernel coefficient for Poly and RBF and is used to control the curvature of the decision boundary $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. The kernel type determines the function that is used to map data into a higher dimensional space.
- In the case of a Multi-Layer Perceptron (MLP) algorithm, the hyperparameters include the size of each hidden layer $(50, 50, 50)$, $(50, 100, 50)$, $(100, 100, 100)$, the learning rate which is either 'constant' or 'adaptive', the activation functions 'Tanh' or 'Relu', the regularization parameter $\alpha \in \{0.0001, 0.001, 0.01, 0.1, 1.0\}$, and the optimizer which is either Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (Adam). Each of these hyperparameters affects the performance of the MLP algorithm.
- The Random Forest is an ensemble learning algorithm that uses multiple decision trees to make predictions. The hyperparameters include the number of trees $[200, 500]$, the number of features looking to consider when looking for the best split using 'auto', 'sqrt', or 'log2', the maximum depth of the tree $[4, 5, 6, 7, 8]$, and the criterion that measures the quality of a split which is either 'gini' or 'entropy'.

The accuracy, precision, recall, and F1-score are used as evaluation metrics to measure the performance of the four shape classification approaches. The accuracy is used to measure the overall correctness of the classification. The precision measures the ratio of correctly classified samples to the total number of samples predicted to belong to a specific class. The recall measures the ratio of correctly classified samples to the total number of samples that actually belong to a specific class. Finally, the F1-score is the harmonic mean of precision and recall, and it is used to evaluate the balance between precision and recall.

Result

The results of the different algorithms are presented in Table A.2. For the precision, recall, and F1-score, the macro average is taken, which is the average unweighted mean. The result shows that the performance of the heuristic rules performs lowest, while according to the F1-score MLP algorithm performs best with the following hyperparameters: {hidden layers : $(50, 100, 50)$, activation : $Relu$, solver : SGD, α : 0.0001, learning : $adaptive$}. While the hyperparameters for the Random Forest are: {criterion : $gini$, max_depth : 8, max_features :$'$ $auto'$, n_estimators : 200}, and for SVM: {C : 1, gamma : 0.5, kernel : RBF}.

Table A.2: The performance of the Heuristic, Random Forest, Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP) algorithms is demonstrated to classify primitive shape into cuboids, cylinders, and spheres.

Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-score
Heuristic classification	0.84	0.77	0.82	0.78	0.73
Random Forest	1.00	0.93	0.93	0.89	0.91
SVM	0.96	0.93	0.93	0.92	0.92
MLP	1.00	0.94	0.95	0.92	0.93

The MLP algorithm performs best on the F1 score. In addition, the confusion matrices show that the MLP algorithm is best at classifying cuboids and cylinders.

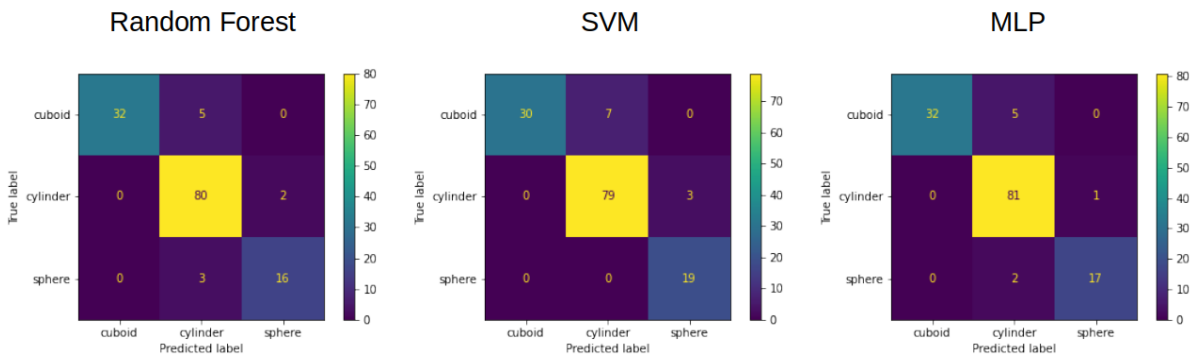

Figure A.3: Confusion matrix of Random Forest, SVM, and MLP on the test dataset.

Discussion

The labeling of each primitive is subjective, as it is performed by visualizing the primitive shape as 3D objects. Because there is no clear boundary between the three categories, noise may occur in the labeling of the categories. To obtain more consistent labels for primitive shapes in transition areas, the values are compared to previously assigned labels. This process leads to bias in the dataset, but on the other hand, improves the relevance of the dataset. It is important to acknowledge that the results of the classification algorithm are based on the authors' definition of cuboid, cylindrical, and spherical shapes.

Comparing the Heuristic classification, Random Forest, SVM, and MLP, Table A.2 shows that the MLP works best at understanding the non-linear relationship between the properties of the primitive shape. The Random Forest approach overfits the data, as can be seen by a training accuracy of 1.00 but a test accuracy of 0.93. The Heuristic classification approach yields the lowest F1-score of 0.73, and it fails to categorize many primitive shapes correctly. Additionally, it requires a deep understanding of the recovered primitive shape and ground truth labels to select the parameters. The results suggest that using the classification approach by Horikoshi and Kasahara [22] does not work well to classify primitive shapes. SVM, on the other hand, uses an RBF kernel to map the given data points into a higher-dimensional space, allowing it to form non-linear decision boundaries. Compared to the heuristic approach, this allows for a more complex decision boundary. From Figure A.3, it can be seen that the Random Forest approach was biased toward predicting cylinders, as it most often classified cuboids and spheres as cylinders.

To evaluate the performance on the precision, recall, and F1-score, the macro average is used to compute the unweighted mean average for the classes. This metric fits the purpose because it takes into account the performance of all classes equally, regardless of the number of samples belonging to each class. The results demonstrate that the MLP algorithm achieved the best F1-score of 0.93, which may be due to its greater robustness to noise. Additionally, the MLP algorithm may have been able to capture more nuanced relationships between the input features and the output labels, resulting in better performance on this classification problem.

In conclusion, classifying primitive shapes is a difficult task due to the subjective nature of labeling primitive shapes. However, the MLP algorithm achieves the best results with an F1-score of 0.93. The trained MLP algorithm is used in this research to classify primitive shapes into cuboids, cylinders, or spheres and thereby enhance the symbolic reasoning framework with additional semantic information.

A.2. Prolog Logic Rules

Code A.1: Prolog rules reasoning version-1

```
affordTask(ID, pap:'Cooking') :-
    primitive_values(ID_tool, _, Dim_tool, _,_),
    min_dimension(Dim_tool, [0.0, 0.08, 0.08]),
    max_shape(ID_tool),

    primitive_values(ID, _, Dim, _,Semantic_shape),
    min_ratio(Dim, 1.5),
    max_dimension(Dim, [0.05, 0.05, 1.00]),
    min_dimension(Dim, [0.005, 0.01, 0.05]),
    Semantic_shape == 'cylinder',
    ID \= ID_tool.

affordTask(ID, pap:'Hammering') :-
    primitive_values(ID_tool, _, Dim_tool, _, _),
    max_ratio(Dim_tool, 4),

    primitive_values(ID, _, Dim, _, _),
    max_dimension(Dim, [0.07, 0.07, 1.00]),
    min_dimension(Dim, [0.005, 0.01, 0.05]),
    min_ratio(Dim, 1.5),
    max_shape(ID),
    ID_tool \= ID.

affordTask(ID, pap:'Pouring'):-
    primitive_values(ID, _, Dim, _,Semantic_shape),
    max_shape(ID),
    min_ratio(Dim, 1.5),
    max_dimension(Dim, [0.08, 0.08, 1.00]),
    min_dimension(Dim, [0.03, 0.03, 0.03]),
    Semantic_shape == 'cylinder'.

affordTask(ID, pap:'Scooping') :-
    primitive_values(ID_tool, _, Dim_tool, _,_),
    max_ratio(Dim_tool, 2),
    min_dimension(Dim_tool, [0.01, 0.01, 0.01]),

    primitive_values(ID, _, Dim, _, _),
    max_dimension(Dim, [0.005, 0.03, 1.00]),
    min_dimension(Dim, [0.00, 0.00, 0.03]),
    ID \= ID_tool.

affordTask(ID, pap:'Turning') :-
    primitive_values(ID_tool, _, Dim_tool, _,_),
    max_dimension(Dim_tool, [0.03, 0.1, 0.1]),
    min_dimension(Dim_tool, [0.00, 0.03, 0.03]),

    primitive_values(ID, _, Dim, _, Semantic_shape),
    min_ratio(Dim, 2),
    max_dimension(Dim, [0.04, 0.04, 1.00]),
    min_dimension(Dim, [0.00, 0.00, 0.03]),
    (Semantic_shape == 'cylinder';
        Semantic_shape == 'cuboid'),
    ID \= ID_tool.

affordTask(ID, pap:'Handover') :-
    primitive_values(ID, _, _, _,_),
    min_shape(ID).
```

Code A.2: Prolog rules reasoning version-2

```
partAffordance(ID, pap:'AffordGrasp') :-
    primitive_values(ID, _, Dim, _,Semantic_shape),
    min_dimension(Dim, [0.001, 0.01, 0.04]),
    max_dimension(Dim, [0.05, 0.05, 1]),
    min_ratio(Dim, 2),
```

```
    (Semantic_shape == 'cylinder' ;
        Semantic_shape == 'cuboid').

partAffordance(ID, pap:'AffordWgrasp') :-
    primitive_values(ID, _, Dim, _, Semantic_shape),
    min_dimension(Dim, [0.05, 0.05, 0.05]),
    min_ratio(Dim, 2),
    Semantic_shape == 'cylinder'.

partAffordance(ID, pap:'AffordContain') :-
    primitive_values(ID, _, Dim, _, Semantic_shape),
    Semantic_shape == 'cylinder',
    hard_condition_min(Dim, [0.03,0.03,0.03]),
    max_shape(ID).

partAffordance(ID, pap:'AffordPound') :-
    primitive_values(ID, _, Dim, _, Semantic_shape),
    min_dimension(Dim, [0.015, 0.015, 0.015]),
    (Semantic_shape == 'cylinder' ;
        Semantic_shape == 'cuboid').

partAffordance(ID, pap:'AffordSupport') :-
    primitive_values(ID, _, Dim, _, _),
    min_dimension(Dim, [0.0, 0.05, 0.05]),
    max_dimension(Dim, [0.03, 1, 1]),
    min_ratio(Dim, 3).

partAffordance(ID, pap:'AffordScoop') :-
    primitive_values(ID, _, Dim, _, _),
    min_dimension(Dim, [0.01, 0.01, 0.01]),
    max_dimension(Dim, [0.1, 0.1, 0.1]),
    max_ratio(Dim, 2).

partAffordance(Affordance, Task) :-
    disjoint_with(Task, Description),
    triple(Description, _, Affordance),
    subclass_of(Affordance, pap:'Affordance').
```

Code A.3: Prolog rules reasoning version-2 querying the ontology.

```
task_object(Affordance, Task) :-
    subclass_of(Task, D),
    triple(D, _, Affordance),
    subclass_of(Affordance, pap:'Affordance'),
    triple(D, _, pap:'RequiresAffordance').

grasp_affordance(ID, Surface, Task) :-
    affordAffordance(ID, Surface, Affordance),
    subclass_of(Task, Description1),
    triple(Description1, _, Affordance),
    subclass_of(Affordance, pap:'Affordance'),
    triple(Description1,_, pap:'hasGraspAffordance').
```