UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Where's my mesh? An exploratory study on model-free grasp planning

by

Fabrizio Bottarel

Thesis submitted for the degree of *Doctor of Philosophy* (33rd cycle)

April, 2021

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Fabrizio Bottarel
April 2021

Publications

The following has been carried out in between November 2017 and December 2020, as part of the PhD course in Advanced and Humanoid Robotics offered by Università di Genova and carried out within the Humanoid Sensing and Perception research line of the Istituto Italiano di Tecnologia (IIT). The three-year project resulted in the following publications (at the time of writing this document):

- P. D. H. Nguyen, F. Bottarel, U. Pattacini, M. Hoffmann, L. Natale and G. Metta, "Merging Physical and Social Interaction for Effective Human-Robot Collaboration," 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), Beijing, China, 2018, pp. 1-9, doi: 10.1109/HUMANOIDS.2018.8625030
- F. Bottarel[1], G. Vezzani[1], U. Pattacini and L. Natale, "GRASPA 1.0: GRASPA is a Robot Arm graSping Performance BenchmArk," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 836-843, April 2020, doi: 10.1109/LRA.2020.2965865.
- N. Piga, F. Bottarel, G. Vezzani, C. Fantacci, U. Pattacini, L. Natale, "MaskUKF: an Instance Segmentation Aided Unscented Kalman Filter for 6D Object Pose and Velocity Tracking", accepted to Frontiers in Robotics and AI, section Humanoid Robotics.

This Thesis provides a structured discussion on top of these aforementioned papers in order to describe the overall contribution of the PhD. As such, some ideas and figures have already appeared in those publications.

[1]Equal contribution

Abstract

The capability of manipulating objects is fundamental for robots in order to interact and integrate in the environment they are deployed in. Since most manipulation actions involve or begin with picking up objects, grasping is an integral part of any manipulation toolbox. Although planning grasps on known object shapes is a mature research field with decades of work behind it, such information is not available in many realistic scenarios. For robots deployed in dynamic and unstructured environments such as homes, stores or even outdoor settings can be cumbersome or outright unfeasible to know the complete 3D shape of target objects, or even recover it in place with exploration techniques. In such cases, approaches that allow planning for optimal grasps given partial 3D information are essential in order to attempt manipulation actions, and the research field is far from being depleted.

In the attempt to model such a use case, in this Thesis we consider a target scenario consisting of a surface cluttered with everyday household objects and we explore different approaches to grasp planning with single-view point clouds, under the the hypothesis that the object 3D models are not known a priori. Although many state of the art methods can find feasible grasps on the observable part of the scene, explicitly modeling the 3D shape of the target extends this capability to unobservable object sides. Initially, we formulate the hypothesis that geometric primitives such as superquadrics can be effectively used as a modeling tool. According to this line of thought, we propose a grasp planning method that constrains the superquadric parameterization to account for the characteristics of the target scenario. In order to evaluate the performance of our method, we tackle the lack of widespread benchmarking protocols for grasp planning tasks by proposing GRASPA, a complete benchmarking tool inspired by reproducibility and interpretability principles. The nature of GRASPA allowed us to evaluate the performance of grasping pipelines with different features on different robotic setups using the same rigorous experimental procedure and observe the failure cases of each approach.

In the final part of the Thesis, we show a possible way of overcoming the limitations of primitive-based methods by studying shape completion methods that have recently been gaining traction in the computer vision and robot vision fields. Thanks to these methods, the

3D description of the target object can be reconstructed from partial views by leveraging past experience (in the form of a learned model) to infer information on unseen parts of objects. We present a proof of concept of how shape completion deep autoencoders can be effectively integrated in a grasp planning algorithm, and we point to interesting research avenues by showing the importance of their internal representation.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Since its inception in the first half of the 20th century, the term *robot* has always denoted a machine meant to perform work by interacting with the environment and the humans present in it. These interactions include, most of the time, some form of manipulation, be it in the shape of moving parts around in an assembly line, sorting products from bins to store shelves, folding clothes in a laundry room, or loading and unloading parcels and crates in a shipment center. The relevance of manipulation to robotics is so vast that since their first deployment in a factory environment, with the Unimate robot in the 1950s, industrial robots have ever since been addressed to with the term *manipulators*.

Although size, shape and capabilities of robots have changed since the 1950s, manipulation is still a very relevant research topic and constitutes an open challenge for both the industry and the scientific and engineering community [9, 50]. Improving the dexterity of end effectors with smart designs in order to reach and even outclass human levels of effectiveness, researching new materials and sensors to improve tactile sensing capabilities, experimenting with bleeding edge machine learning techniques in order to discover new policies in an unsupervised way; these are but a portion of the ramifications of the robotic manipulation challenge. However, any kind of manipulation task involves grasping the target of the manipulation action at some point. Hence, autonomous robot grasping is conventionally considered to be a fundamental part of the manipulation research field.

Grasping is typically defined [103] as the group of actions necessary in order for a robot to close its end effector around an object and constrain its movement with respect to the robot hand or gripper. Hence, after a successful grasp the movement of the object in terms of position and orientation is entirely defined by the state of the robot[1]. In the literature, the

[1]In other words, the object can be considered fixed to the robot end effector.

typical structure of a grasping algorithm (also often called *grasping pipeline*) can be boiled down to the following stages:

- **Target acquisition.** The system initially has no information about the target object. In this stage, information coming from the perception system of the robot is retrieved and processed. Typically, this information is visual (since no contact between the robot and the target exists just yet) and comes from either a 2D or RGB-D camera. This stage outputs information about the target object, possibly segmented from the rest of the scene
- **Grasp planning.** Given the target, this stage computes the optimal position and orientation of the robot end effector[2] (hand or gripper) that maximizes the probability of a successful grasping action. Different grasp planning approaches might use additional constraints in order to rule out unsuitable candidates, e.g. whether the grasp pose allows the end effector to stay clear of other objects or the scene itself, or some manipulability measure in the target pose. The output is simply the end effector pose in terms of position and orientation of the TCP and the pregrasp configuration, i.e. the initial position of the joints of the end effector itself, such as the angle of the finger joints
- **Motion planning.** This stage takes care of planning the trajectory that will lead the robot end effector in the target pose. This is usually conditioned and constrained by other factors, such as avoiding collisions between the robot links and the scene or avoiding singularity configurations. The output is very setup-specific, but is generally a trajectory of setpoints in task or joint space for the manipulator to follow at runtime
- **Grasp execution.** This stage moves the robot and the end effector in order to follow the target trajectory and perform the planned motions. It is typically specified by the control architectures that are supported by the hardware and software configuration, and is also very setup-specific because of this.

The research work summarized in this Thesis mainly focuses on the grasp planning stage. Nevertheless, the implementation of such work to perform experiments lead to the study and application of popular techniques and tools related to the other stages.

Grasp planning is a field with more than three decades worth of literature and contains a vast landscape of approaches, each with varying assumptions and methodology [50, 100, 10,

[2]In this Thesis, we often use the denomination TCP (Tool Center Point) to denote the reference system attached to the end effector.

7, 55, 51, 27]. Although providing an extensive taxonomy of grasp planning approaches is outside the scope of this Thesis, we hereby attempt to outline some common or distinctive aspects that are relevant to this work.

Target driven vs. target agnostic approaches. Some grasping pipelines are designed to grasp specific objects in a scene [76, 120], while others can produce grasp candidates directly from scene data [82, 62], with no explicit target definition or segmentation.

End effector type. A large part of grasp planners in literature assume a parallel-jaw gripper structure [74, 62, 82]. Other approaches focus on multi-fingered or humanoid hands [117, 102] or even suction grippers [63].

Clutter definition. Although there is no universal and clear-cut definition of clutter and degrees of clutter in a scene, grasping pipelines typically tackle:

- *visually cluttered* scenarios, where some objects are mutually occluded when observed from the viewpoint of the robot, but are not close to each other
- *spatially cluttered* scenarios, where objects lie in a structured or unstructured pile and they are in contact to each other (this typically implies visual clutter as well)
- *isolated* objects, where there is no clutter at all. All the approaches that can only handle one object at a time fall in this category.

Prior on object shape. Different grasp planners make different assumptions about the target object in their formulation. For instance, approaches that rely on pose detection require the 3D mesh in order to locate the object in the scene and plan for grasps (*model-based approaches*), while others only assume the target object can be modeled with a geometric primitive [120, 8]. Some methods do not require any modeling at all [62, 74], while some attempt category-agnostic shape reconstruction based on partial data [114, 60] (we refer to these two last categories with the term *model-free approaches*).

Planar vs. spatial approaches. This distinction relates to the parameterization of the detected grasp candidates. Planar approaches assume the scene is being observed from a fixed point over the target surface and the optical axis is perpendicular to the surface. This type of approach describes grasps either in the form of oriented rectangles or in the form of 3D position and rotation angle around the axis optical axis [62, 63]. Other approaches [82, 74] generate grasps in terms of TCP position and orientation with a 6 DoF parameterizataion, taking full advantage of the manipulator motion capabilities.

Data-driven vs. analytical methods. In general, grasp planners work by sampling candidate poses around the object and then computing some sort of quality index in order to rank them and eventually find the best grasp. Decades worth of research on grasp quality metrics have suggested a wide landscape of options that can be computed when the gripper or hand kinematics and the complete object shape are known. These typically rely on analytical procedures that involve explicit geometrical reasoning or dynamic considerations on the forces and friction in play [7, 97]. In recent years, the rising popularity of machine learning and deep learning has given birth to a whole new class of approaches, where quality metrics are learned from data instead [10, 27].

One of the challenges of the robotics research community has always been to enable robots to operate in highly unstructured or dynamic environments. Classic examples are construction sites, disaster scenes or space missions. However, homes and stores prove just as challenging for an automatic system, to the point where robot challenges have been designed around such human environments [107] to serve as benchmarks for the capabilities of service robots. For instance, a robot whose task is to fetch an object from another room implies navigating around the layout of the house, avoiding static (furniture) and dynamic (pets, humans) obstacles on different surfaces, to finally face a messy shelf or tabletop where the target object resides. In this Thesis work we focus on this last kind of environment, assuming objects lie on a **planar surface**, possibly in **visual and spatial clutter**, where the exact **model of the object to grasp is unknown a priori**. In particular, we consider the model-free hypothesis to be of particular interest. In the robotics and manipulation literature the grasping field is often considered to be mature [9], and the grasp planning problem is often considered to be solved and of little interest. While we agree with the first statement, we argue that the literature offers a large variety of effective and well-established methods [7] only as long as the object model is known and its pose can be reliably estimated within the scene. In the abscence of known 3D models, vision-based grasping approaches cannot perform pose estimation and therefore cannot rely on the straightforward application of classic grasp planning methods. This is a reasonable assumption to make when unstructured environments are concerned, as the system probably would not know the exact shape of the target object, nor would it be able to easily recover it via visual exploration[3]. For this reason, we argue grasp planning still proves to be quite the challenge in realistic applications, as proven by the number of novel approaches that have been published in recent years [55, 27].

Before moving on, we briefly describe some state of the art methods that are relevant to the target scenario of this Thesis, i.e. **grasp planning when the object model is not known a priori**. Dex-Net 2.0 [62] is a planar approach that generates a large number of candidate grasps from a depth image and computes a quality score for each of them with a GQ-CNN (Grasp Quality Convolutional Neural Network). The network is trained using a large synthetic grasp dataset annotated with analytical metrics. Due to its performance and inference speed, Dex-Net 2.0 and its successive revisions have been a cornerstone of data-driven grasp planning in latest years. Since this approach is designed for bin-picking applications, however, it requires the RGB-D camera to be placed on top of the target surface and only provides planar grasps. GPD [82] tackles this shortcoming, being able to provide 6-DoF grasp planning with similar performance on real-robot experiments. This method consumes RGB-D data in the form of a point cloud, and uses a CNN (although extended to a 3D voxel grid) to infer the quality of grasp candidates generated from the point cloud geometry. Also similarly to Dex-Net, GPD is also trained on a synthetic dataset annotated with analytical metrics. 6DoF-GraspNet [74] improves upon GPD by using data-driven methods to directly propose initial candidate guesses from a point cloud by using a Variational Autoencoder (VAE) and later refine them using gradient information from a grasp quality CNN. Unlike previously mentioned approaches, this work uses a physics simulation engine in order to generate ground truth (instead of analytical metrics) for the training process.

The end-to-end nature of the approaches just outlined allows them to quickly generate and rank large numbers of candidates at runtime. However, they lack an explicit target representation which is rather useful in robotics task to guide motion planning, collision avoidance and scene understanding. Some methods propose to explicitly model target objects from single-view point clouds by fitting shape primitives such as cuboids and superquadrics [120, 83] that are also used for grasp planning. These methods are typically based on framing the primitive fitting as an optimization problem, and are therefore quite interpretable and predictable in their output. While this is a desirable feature that is not often present in data-driven and deep learning based models, the simplicity and compactness of the primitive

[3]In case of an object lying in visual clutter or in a constrained environment, for instance, it would be impossible for the robot to move a hand-mounted camera around it to gather enough data for a complete 3D reconstruction.

representation limits the span of possible shapes that can be modeled due to symmetry and lack of detail. Another class of methods attempts to reconstruct the complete object shape from partial point cloud data, borrowing shape completion techniques from the computer vision field. Some of these approaches use 3D deep networks to reconstruct a complete object mesh from single view RGB-D data [116, 60] and then use grasp quality metrics based on the grasp wrench space to obtain good grasp candidates. Lundell et al. [61] employ a similar reconstruction pipeline, but use the obtained shape to render a synthetic depth map and use Dex-Net to compute grasps. These methods are some of the most promising venues in the grasp planning research field, since they allow researchers to employ any well-established grasp planning analytical or data-driven method that requires the full mesh without having it beforehand.

This Thesis work begins (Chapter 2) with an attempt at tackling the aforementioned scenario by proposing a grasp planner that relies on modeling the target object from partial views with expressive and compact geometric primitives, i.e. superquadric surfaces, through a constrained optimization process. The approach was meant to be deployed on the iCub humanoid robot [71], and the features and limitations of the platform were accounted for in the design process. We also show how this approach performs with respect to a similar superquadric-based approach [120] also developed for the iCub robot. Initially, we tested our grasp planner on isolated objects since the perception system implemented on the platform would fail when performing target acquisition on visually cluttered scenes. However, good results in such conditions were obtained by improving the perception capabilities of the robot with the integration of a state of the art deep instance segmentation architecture [45]. Building in this direction, we obtained a versatile tool for tackling a variety of different object sets and scenes (Chapter 3) by adapting a method to quickly generate custom datasets in order to train instance segmentation machine learning models.

The next objective in the PhD project would have been to compare our results with other state of the art approaches [82, 62] to grasp planning on the same cluttered tabletop task. However, we initially found it to be difficult to do for a number of reasons linked to how typically authors experimentally validate their proposed solution on real robot setups. Due to the abscence of a widespread benchmarking protocol, the experiments were performed in non-reproducible conditions and the results reported in a way that was not really comparable with other approaches from other sources. This is the motivation that led to the proposal of GRASPA (Chapter 4), a benchmarking protocol for grasp planners built around the concepts of clarity and granularity of metrics, reproducibility of experimental conditions and adaptability to different robot platforms.

Employing GRASPA to benchmark state of the art model-free grasp planners allowed us to observe some of their failure cases and limitations, due to occlusions and partial information deriving from single-view RGB-D data. Having also observed the limitations that come with modeling objects with geometric primitives such as cuboids and superquadrics, we investigated state of the art machine learning techniques to reconstruct complete shapes from partial measurements, i.e. shape completion and reconstruction techniques. In particular, we show how they can partially compensate for the partial visual information that our target scenario implies, and whether they could be integrated into existing grasping pipelines (Chapter 5). Even though this study did not eventually result in a contribution to the field, it allowed us to obtain precious insight on these techniques and ideas for future research avenues.

Chapter 2

Model-free grasp planning with superquadrics

Thanks to the last two decades of advancements in sensor technology and computer vision, nowadays integrating 3D sensing capabilities into a robot has become cheaper and more practical than ever. Even with these powerful sensing capabilities, robot vision is still limited by occlusion (both auto-occlusion and caused by other objects or elements of the scene) to only ever have a partial perception of objects. One of the stepping stones in the design of algorithms and pipelines for autonomous, vision-based robotic grasping and manipulation is choosing a suitable representation for modeling the objects the robotic system is supposed to perceive, reason about and act on. Despite decades of research and experimentation on the subject, however, the community has not converged to a single representation that is indisputably preferrable to all others in all cases. Recent literature reviews on the topic of vision-based grasp planning [27, 50, 55] show that different approaches model objects according to assumptions on the task and the scene. Some of the most widespread approaches are:

- assume that the object shape is known. This is often the case when the object CAD or mesh is available a priori. In this case, the modeling task turns into a pose estimation task
- assume that the object shape is not known, but it can be decomposed in one or more geometric primitives. In this case, the modeling task turns into a shape fitting and optimization task

- assume that the object shape is not known, but something similar has been seen. The modeling task turns into a shape completion task (this topic is analyzed in detail in Chapter 5)
- no shape is explicitly sought for. There is no explicit modeling of objects, rather the approach proposes to analyze local features of the 3D data.

As seen in the introduction, the state of the art on grasp planning pipelines and grasp planners is rapidly moving towards data-driven and deep learning based approaches [109, 10, 13], inspired by the success of these disciplines in other scientific fields. Nonetheless, recent research Du et al., Kroemer et al., Lin shows that analytical and optimization-based object modeling approaches can still have a place in some scenarios and tasks. This claim has its roots in the following factors.

Availability of intermediate representation. In robotics, explicitly modeling the object geometry is often useful for tasks that are complimentary to manipulation (e.g. scene collision avoidance). This feature is typically not available in most of the data-driven, end-to-end trained grasping approaches that have risen in popularity in latest years.

Cost of a complex shape representation. Although some data-driven grasping pipelines explicitly perform a modeling step before computing grasps (see Chapter 5), such approaches require training on large datasets and have a large footprint in terms of computing power and memory availability. On the other hand, if the target objects have simple shapes (as is often the case with household items) they can be modeled in a very compact way by exploiting geometric primitives.

Hardware constraints. Data-driven grasping pipelines, regardless of whether they provide an interpretable representation of the object, typically are developed on top of deep learning frameworks that require dedicated hardware in order to be efficient and responsive. On the one hand, this is becoming easier to overcome, as training of the models is done offline on large GPU clusters and compact hardware to perform inference on the robot platform is becoming commonplace. On the other hand, methods that perform equally fast without the need for a training procedure or dedicated hardware are desirable (even if this comes at a tradeoff in performance).

In this chapter, we propose a simple grasping pipeline based on explicit object modeling through superquadric functions. At its core, this method is inspired by the work of Vezzani et al. but it is designed to simplify some of its moving parts by:

- eliminating some of the degrees of freedom in the superquadric optimization process by exploiting the scenario structure
- performing the optimization with a well-behaved analytical gradient instead of finite differences
- using a different grasp candidate proposal method that takes advantage of the symmetry of the superquadric representation
- ranking the grasp candidates with a metric based on the robot reachability and end effector geometry.

In 2.1 we present an overview of other superquadric-based approaches to object modeling and grasping drawn from the computer vision and robotics literature. In Section 2.2 we present the approach, outlining our contribution. We quantitatively tested the method (Section 2.3) using two different experimental protocols using the iCub humanoid robot platform. Finally, we review and discuss the experimental results (Section 2.4) and show an application of the method in the context of a complex HRI framework (Section 2.5).

2.1 Related work

The concept of superquadric surfaces was first introduced in 1981 in the field of computer graphics by Barr [4] as a compact and smooth representation for a family of 3D shapes including superellipsoids, superhyperboloids and supertoroids. They were proposed as a unified mathematical formalism that would bring a speedup in computation, by reducing the need for memory-intensive representations such as lists of edges, faces and vertices. They were intended for use in computer aided design and rendering, aid structural and mechanical analysis and verify machine control trajectories. The concept of superquadrics was soon extended by introducing local and global shape deformations [5] and the scope of its representation power was amplified with the introduction of hyperquadrics [44].

Solina and Bajcsy hypothesized the usage of superquadric shapes as components of part-based models, exploring the possibility of using their closed-form description and proposing a mathematical formulation for the problem of segmentation and recovery of superquadrics in both 2D and range images [105, 46]. The intuition behind the concept of part-based models inspired some interest in the computer vision community in finding ways to model 3D object or scene data with multiple superquadrics [18, 19, 29]. This approach allows to model interesting parts of objects (e.g. handles) with expressive primitives or to push the limits of superquadric representation by using a number of convex shapes to model a non-convex shape. This goal is generally obtained by partitioning the 3D point clouds in a number of clusters, and substituting to each cluster the superquadric (superellipsoid) that best fits the points in the cluster. Methods belonging to this family typically differ in the way the point cloud is split, i.e. the way the unsupervised clustering problem is tackled.

Recently, on the wake of the data-driven disciplines, some researchers used 3D deep learning techniques to address the problem of modeling scene and partial object point clouds with multiple superquadrics [83]. These works simultaneously leverage the advantages of both the compact representation offered by superquadric shapes and the fast inference time provided by state of the art DNN (Deep Neural Networks) architectures and modern GPU hardware.

In robotics, being able to reduce the complexity and memory footprint of a scene to a finite number of expressive shapes with a compact representation is useful in a number of context. The implicit formulation of the superquadric surface allows for quick computation of whether points reside inside or outside the shape itself, and this in turn elicits faster collision checking, e.g. when motion planning or grasp planning algorithms are concerned. Over the last two decades, a number of approaches relying on superquadric (either single or multiple) modeling for scene segmentation and grasp planning have been proposed [42, 120, 119, 8, 67, 19, 106]. Relevant to our scenario are works that are employ multi-fingered or humanoids hands [106, 42, 120, 119] since our target robot platform, the iCub robot, features humanoid hands. Some of the works by Vezzani et al. [120, 119] have been developed with the iCub platform in mind and propose a fast solution to the grasp pose detection problem by modeling both the robot hand graspable volume and the target object as superquadric shapes.

2.2 Methodology

The target scenario for this work consists in the robot facing a horizontal surface (e.g. a tabletop, countertop, or shelf) populated with objects whose 3D shape is unknown a priori. There can be partial visual occlusion between the objects, but they cannot be stacked on top of each other. This type of scenario, while unrealistic in an industrial bin picking task, is very relevant for service robotics operating in human environments and is sometimes referred to as *structured clutter* [74, 76].

We hereby describe our approach to a superquadric-based grasping pipeline. Initially, we present the superquadric family of shapes and give some insight on their features for object modeling. After that, we present the approach pipeline itself. Following the typical structure of a model-based grasp planner, our approach can be functionally split in two stages. In the first stage (Section 2.2.2), the information coming from the robot vision system is used to produce a representation of the object. In the following stage (Section 2.2.3), this object description is used to compute a list of feasible grasp candidates and select the best one according to some metric.

2.2.1 Features of superquadric representation

Superquadrics are a family of 3D parametric surfaces that can describe a large variety of shapes in a single and continuous parameter space, and became popular for their capability to represent complex shapes with a limited number of parameters. Within the superquadric family of surfaces, *superellipsoids* are the ones that are the most suitable to model objects, since they define closed and limited surfaces (the others being *supertoroids* and *superhyperboloids*). Hence, with some abuse of notation, this thesis uses the term *superquadric* to refer to *superellipsoids*. Using an object-centric reference frame, superquadric surfaces can be mathematically described by an implicit formulation

$$F(\bar{x},\lambda) = \left(\left|\frac{x}{s_x}\right|^{\frac{2}{\varepsilon_2}} + \left|\frac{y}{s_y}\right|^{\frac{2}{\varepsilon_2}}\right)^{\frac{\varepsilon_2}{\varepsilon_1}} + \left|\frac{z}{s_z}\right|^{\frac{2}{\varepsilon_1}} \quad \lambda = \{s_x, s_y, s_z, \varepsilon_1, \varepsilon_2\} \in \mathbb{R}^5_+ \tag{2.1}$$

where parameters $\{s_x, s_y, s_z\}$ define the size of the three axes of the superquadric while $\{\varepsilon_1, \varepsilon_2\}$ define its shape in terms of edge roundedness. Figure 2.1 shows how these parameters (aptly called *roundedness* parameters) shape a superquadric surface with the same axes size. Using the same five parameters, superquadric surfaces can also be described by an explicit formulation

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \bar{x}(\eta, \omega) = \begin{bmatrix} s_x \cos^{\varepsilon_1}\eta \, \cos^{\varepsilon_2}\omega \\ s_y \cos^{\varepsilon_1}\eta \, \sin^{\varepsilon_2}\omega \\ s_z \sin^{\varepsilon_1}\eta \end{bmatrix} \qquad \begin{matrix} -\pi/2 \leq \eta \leq \pi/2 \\ -\pi \leq \omega \leq \pi \end{matrix} \tag{2.2}$$

By applying a transformation $T_\lambda \in \mathrm{SE}(3)$ to the points belonging to the surface defined by either Equation 2.1 or Equation 2.2, we can obtain the description of an object in the 3D euclidean space together with its pose (defined as position and orientation) with 11 parameters. This compactness of the representation is the first feature of the superquadric representation for object modeling.

A second useful feature of the superquadric representation relies in its implicit formulation. Considering Equation 2.1:

- $\bar{x} : F(\bar{x}, \lambda) = 1$ represent all the 3D points lying on the superquadric surface
- $\bar{x} : F(\bar{x}, \lambda) < 1$ represent all the 3D points lying inside the superquadric boundaries
- $\bar{x} : F(\bar{x}, \lambda) > 1$ represent all the 3D points lying outside the surface.

This property identifies superquadrics as mathematical solid, for it univocally determines the region of the three dimensional space an arbitrary point falls into. Hence, it is also referred to as *in-out* function.

The third feature of the superquadric representation can be shown by observing the behaviour of parameters $\{\varepsilon_1, \varepsilon_2\}$. Each one describes the roundedness of the superquadric along one direction (nort-south and east-west) and behaves in the following way (refer to Figure 2.1 for a visual guide):

- for $\varepsilon < 1$, the shape is increasingly squared with sharper edges
- $\varepsilon = 1$, the shape is rounded
- $\varepsilon \sim 2$, the shape has a flat bevel
- $\varepsilon > 2$, the shape is "pinched" and pointed.

As pointed out in other works [46, 120, 67, 18], constraining the roundedness parameters between 0 and 2 leads to strictly convex shapes. Empirically, we can observe that a large number of common household objects can be approximated to such shapes (e.g. containers such as boxes and bottles) without loss of affordance. The convexity also makes the in-out function well behaved with respect to the optimization problem.

Thanks to the features we just outlined, the superquadric representation is descriptive of a good number of household objects that a service robot might need to grasp on a daily basis. In the next sections, we show how we propose to frame and constrain an optimization problem in order to fit a superquadric shape to a segmented point cloud coming from the visual system of the robot (Section 2.2.2). We also propose a simple algorithm to generate candidate power grasps around the computed superquadric and rank them according to the robot kinematic structure and end effector geometry (Section 2.2.3). We use the term *power grasp* to denote a type of grasp where only the pose of the TCP (Tool Center Point) of the end effector is computed, as opposed to a *precision grasp* where the position of each contact point with the is planned for.

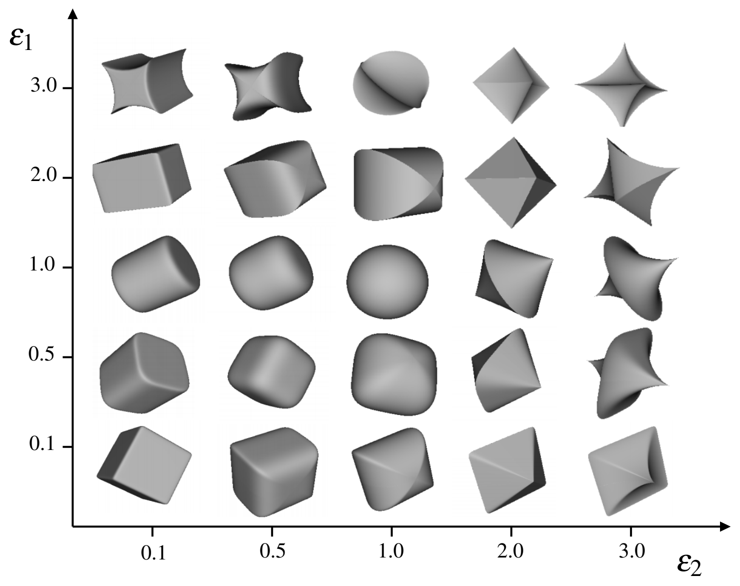


Figure 2.1 Superquadric shapes with varying roundedness parameters $(\varepsilon_1, \varepsilon_2)$ and fixed size $s_x = s_y = s_z$. For values $0 < \varepsilon < 2$, superquadrics are convex shapes with continuous surface gradient. In particular, it can be observed that superquadrics closely approximate more traditional geometric primitives used in object modeling such as spheres ($\varepsilon_1 = 1, \varepsilon_2 = 1$), cylinders ($\varepsilon_1 = 1, \varepsilon_2 \to 0$), octahedrons ($\varepsilon_1 \to 0, \varepsilon_2 = 2$) and cubes/parallelepipeds ($\varepsilon_1 \to 0, \varepsilon_2 \to 0$). Image sourced from [118].

2.2.2 Modeling objects with superquadrics

The aim of this first step is to find the best superquadric representation of an object perceived by the robot, i.e. finding the set of parameters that best fit the 3D data according to some cost function. In this case, such 3D data is supposed to be provided in the form of a partial, single view object point cloud $P \in \mathbb{R}^3$ acquired by the visual perception system of the robot. As initially proposed by Jaklič et al. [46], the best superquadric to model P is the one whose surface is closest (in an euclidean sense) to the point set. Recalling the *in-out function* defined in Equation 2.1, the superquadric surface is defined by the locus of points that satisfy $F(\bar{x}, \lambda) = 1$. Hence, minimizing the distance of the i-th point $p_i = (x_i, y_i, z_i) \in P$, for $i = 1, \dots N$ from the superquadric leads to the parametrization $\lambda = \{s_x, s_y, s_z, \varepsilon_1, \varepsilon_2, T_\lambda\}$ that best fits the 3D data. The search for λ can be cast as a least-squares minimization problem over the point set

$$\lambda = \arg\min \sum_{i=1}^{|P|} \left(\sqrt{s_x s_y s_z} (F(p_i, \lambda) - 1) \right)^2 \tag{2.3}$$

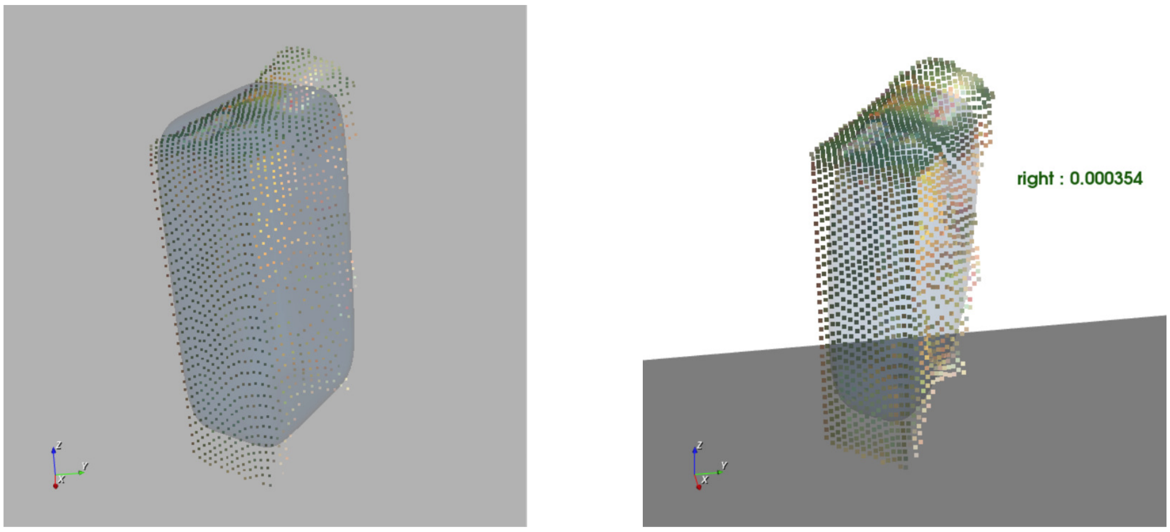
where the term $(F(p_i,\lambda)-1)^2$ is the squared distance of the i-th point from the superquadric surface. The term $\sqrt{s_x s_y s_z}$ is introduced as a regularizer for the superquadric volume, as we are interested to find the smallest surface that fits the point cloud.

Our first contribution with respect to the work of Vezzani et al. [120] is to introduce some prior about the scene where the object is located to simplify the optimization process. In particular, our target scenario consists in the robot observing a tabletop or shelf scenarios with a single object or isolated objects. Thus, instead of estimating the pose of the superquadric T_λ with 6 degrees of freedom, we hypothesize the object to be modeled is laying on a surface. This assumption constrains two of the three axes to lay in a plane parallel to said surface (which is fixed) and reduces the pose estimation problem to 4 degrees of freedom, the first three being the center of the superquadric and the fourth being the rotation ϕ around its z axis. The parameter set λ is reduced in dimensionality from 11 (3 for the position of the center, 3 for the size, 2 for the roundednss, 3 for orientation) to 9. We redefine the notation for the parameter set as $\lambda = \{x_c, y_c, z_c, s_x, s_y, s_z, \varepsilon_1, \varepsilon_2, \phi\} \in \mathbb{R}^9$ where (x_c, y_c, z_c) is the center of the superquadric.

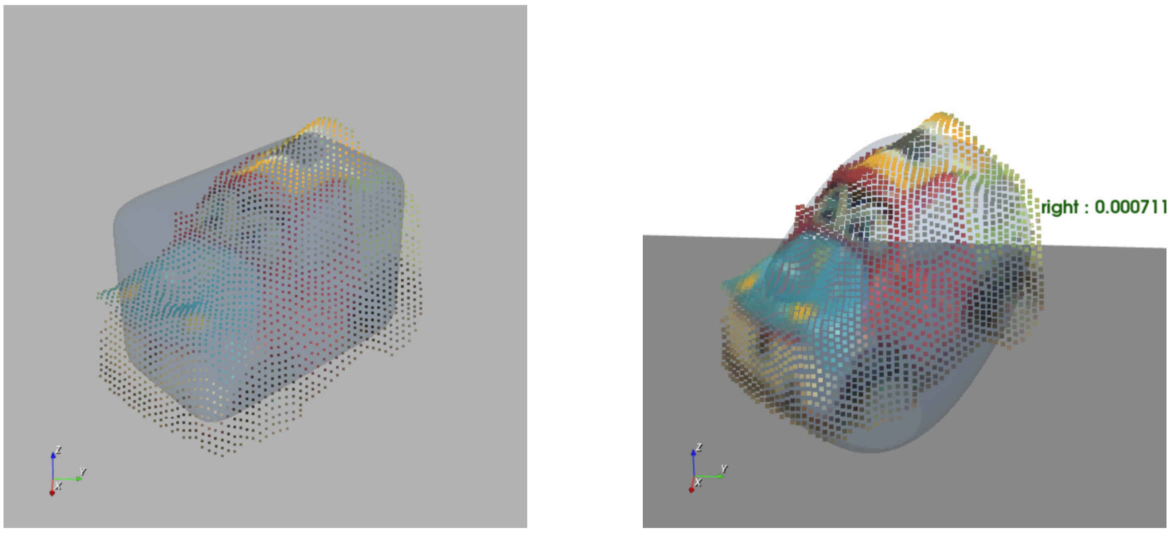
We frame the optimization problem defined in Equation 2.3 as a constrained nonlinear optimization problem, since the target superquadric has to adhere to the following constraints:

- the superquadric has to be convex, therefore $0 < \varepsilon_1, \varepsilon_2 \leq 2$
- since the object is assumed to lie on a surface without intersecting it, $z_c - s_z > z_{surface}$
- the superquadric has to have non-zero volume, i.e. $s_x, s_y, s_z > 0$.

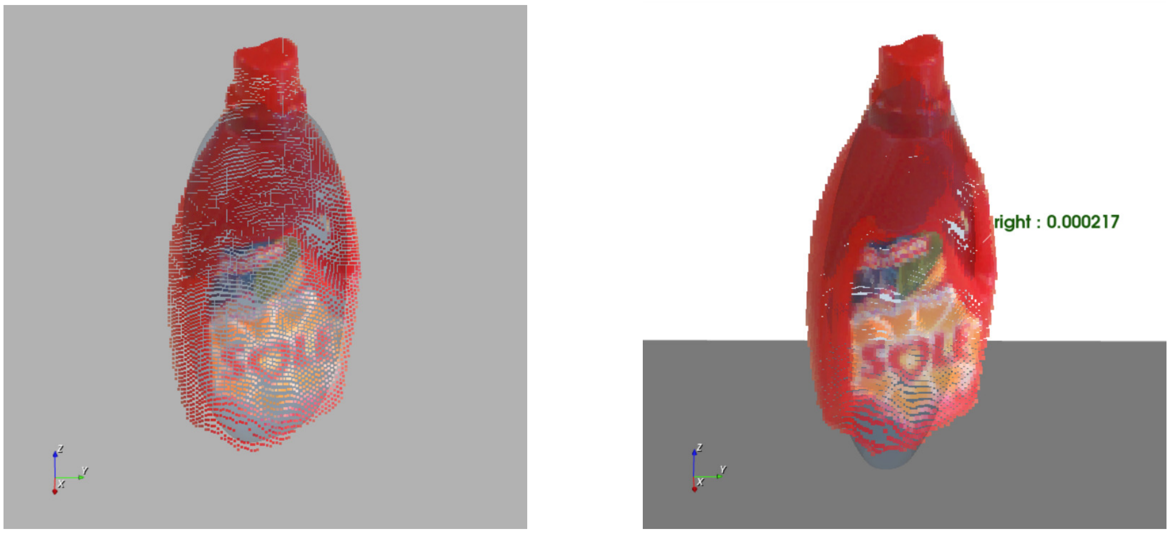
We solve the optimization problem for λ using an interior point filter line search algorithm, as implemented in the IpOpt library [127]. Since this class of methods benefits (in terms of accuracy of the results and computing power) from the availability of a closed-form gradient of the cost function, we take advantage of the analytical nature of the in-out function $F(p_i,\lambda)$ and provide it. This is a second difference with respect to how the problem was solved in the work of Vezzani et al., where a finite difference method was used. Some examples of partial point clouds modeled as superquadrics with our method can be seen in 2.2.



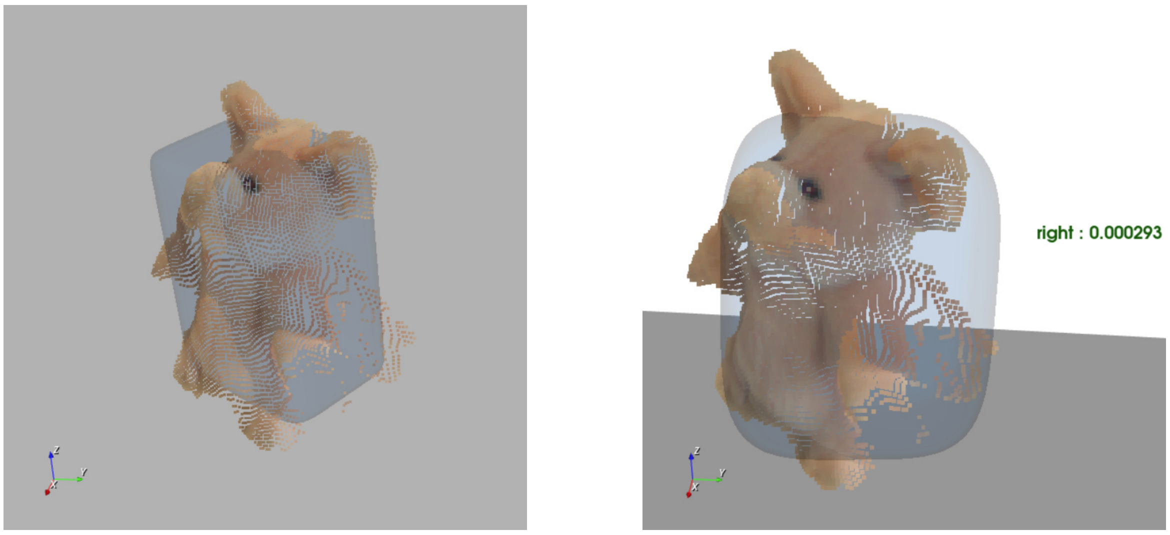
(a) Box object



(b) Toy Car object

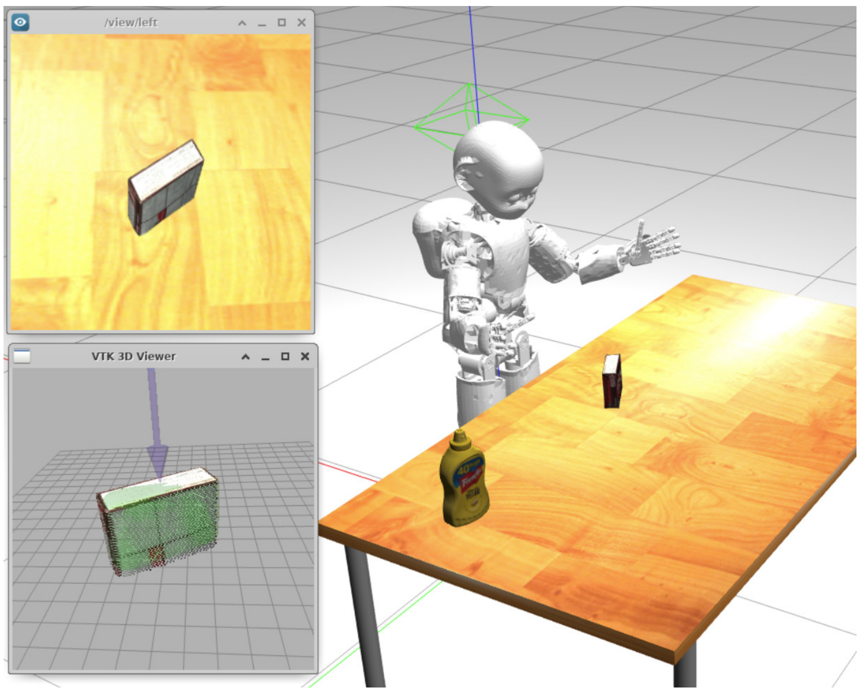


(c) Large Bottle object (not in the dataset, ungraspable)

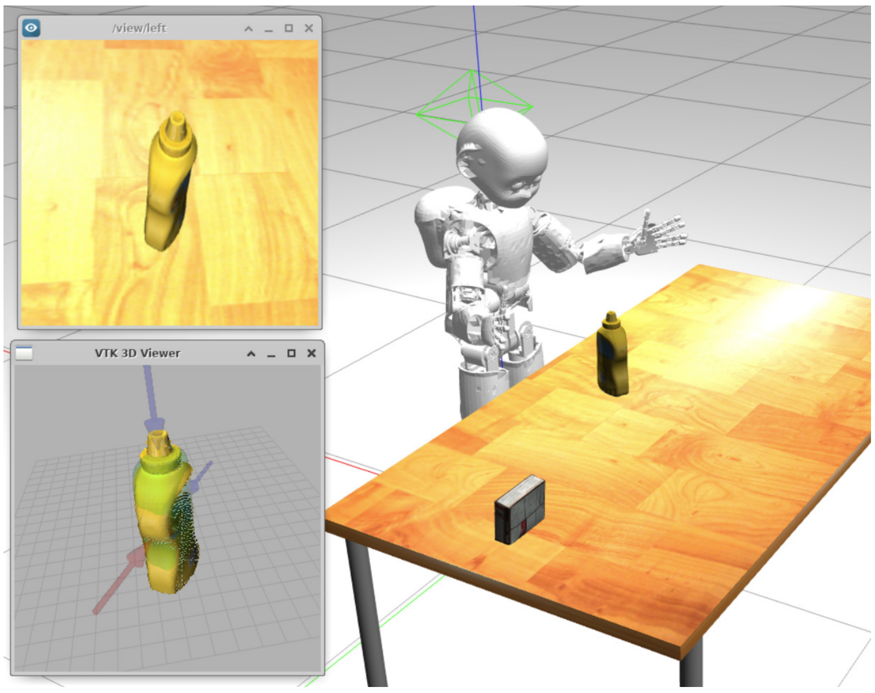


(d) Teddy Pig object

Figure 2.2 Visual comparison of results obtained by modeling partial object point clouds with our method (left) and the one by Vezzani et al. (right). The superquadric obtained for (a) is very similar between methods, since the object is a simple geometric shape. In the case of (b) and (c), the 6D method by Vezzani et al. produces superquadrics that fit the point cloud better, but intersect the surface the object is lying on. Although the plushie in (d) with a single superquadric is impossible, we show the best superquadric fit obtained by both methods.



(a) Simulated grasping scenario with YCB Pudding Box.



(b) Simulated scenario with YCB Mustard Bottle.

Figure 2.3 The superquadric cardinal point grasp planning approach deployed in a tabletop scenario simulated in Gazebo. The test objects are meshes from the YCB object set. The snapshots show the left eye view of the object, the superquadric computed from the partial view (faint green) and the feasible grasps (approach directions marked as arrows). Red and blue arrows indicate candidate grasps for the left and right hand respectively. Grasps that would cause a collision with the table surface are filtered out of the candidate list.

2.2.3 Cardinal point grasp planning

Once the incomplete point set P has been modeled as a superquadric, we wish to determine the best end effector pose for the robot to reach and grasp the object. The motivation for choosing to plan for power grasps (i.e. only the pose of the TCP is specified) instead of precision grasps (i.e. the location of every contact with the object is specified) is mainly twofold:

- the proposed approach was originally designed for humanoids, underactuated hands. Since not every joint can be directly controlled, with this type of hands it is usually preferrable to perform grasps by planning for the pose of the palm and then closing the fingers starting from an initial (pregrasp) configuration. This way, the fingers can naturally adapt to the object shape due to their mechanical design
- planning for precision grasps requires an accurate estimation of the object surface in order to detect contact points. Since superquadrics are approximated representations of objects and lack the power to model both asymmetry and details, planning for contacts on such surfaces might not lead to actual contacts on the object.

During our experience acquired by reproducing and testing the superquadric grasp planning approach proposed by Vezzani et al. on the iCub humanoid robot, we observed that for common household objects the algorithm would most of the time plan for grasps either on the top or the sides of the object. Moreover, the approach did not take into the account whether the pose was reachable by the hand, and this would cause failures. According to these observations, we propose a grasp planning algorithm that generates a number of top and side candidate grasps around the superquadric and then ranks them according to a cost function that takes into account the robot workspace.

Candidate grasp pose generation Given a superquadric parametrized by the set λ, our approach generates grasp candidates in a narrowed search space for position and orientation:

- the position of the TCP is constrained to the cardinal points of the superquadric, so that the palm touches the surface. See Figure 2.4 for a representation of the iCub hand and its TCP reference frame. We define as *cardinal points* the intersections between the superquadric surface and its main axes (Figure 2.5)
- the TCP orientation is constrained so that each pose axis is parallel to one superquadric axis, with the g_x axis pointing towards the superquadric center. This way, the surface of the palm is in contact with the superquadric surface with no collision

- side grasps are constrained to having the thumb always point upwards.

Algorithm 1 Grasp pose candidate generation

```
Input:
    Center $s_c$ of the superquadric, unit vectors for the superquadric axes $\{\vec{a}_x, \vec{a}_y, \vec{a}_z\}$, size
    $\{s_x, s_y, s_z\}$ of the superquadric, maximum hand aperture $w_{hand}$, table surface height $z_t$
Output:
    Grasp candidate set $S_g$
    Grasp pose $g_i = \{R_i, T_i\} \in S_g$
 1: procedure GENERATEGRASPS($\lambda$)
 2:     $S_g = \emptyset$
 3:     $S_{g_x} \leftarrow \{\vec{a}_x, \vec{a}_y, -\vec{a}_x, -\vec{a}_y\}$          ▷ $g_x, g_y$ search spaces
 4:     $S_{g_y} \leftarrow \{\vec{a}_x, \vec{a}_y, \vec{a}_z, -\vec{a}_x, -\vec{a}_y, -\vec{a}_z\}$
 5:     for $\vec{g_{i,x}} \in S_{g_x}$ do
 6:         for $\vec{g_{i,y}} \in S_{g_y}$ do
 7:             $\vec{g_{i,z}} \leftarrow \vec{g_{i,x}} \times \vec{g_{i,y}}$
 8:             $T_i \leftarrow s_c - s_z \vec{g_{i,z}}$
 9:             $R_i \leftarrow [g_{i,x}\ g_{i,y}\ g_{i,z}]$
10:             if ISGRASPFEASIBLE($R_i$) then
11:                 $S_g \leftarrow$OFFSET($R_i, T_i$)
12:     return $S_g$
13: procedure ISGRASPFEASIBLE(R, T)
14:     if ISOBJECTGRASPABLE($s_x, s_y, s_z$) then
15:         if $s_c - T_z > z_t$ then          ▷ make sure grasps are over the table surface
16:         return true
17:     return false
18: procedure ISOBJECTGRASPABLE($R, s_x, s_y, s_z$)
19:     if $g_x \parallel a_x$ and $w_{hand} > 2s_x$ then
20:         return true
21:     if $g_x \parallel a_y$ and $w_{hand} > 2s_y$ then
22:         return true
23:     if $g_x \parallel a_z$ and $w_{hand} > 2s_z$ then
24:         return true
25:     return false
```

Grasp poses are generated as detailed in Algorithm 1, and are represented as homogeneous transformations $g_i = \begin{pmatrix} R_i & T_i \\ 0 & 1 \end{pmatrix}$ linking the robot palm TCP to the root frame. In Operation 18, pose candidates are rejected if the cross section of the superquadric, is larger than the distance between the middle fingertip and the thumb fingertip in pregrasp position (i.e. thumb

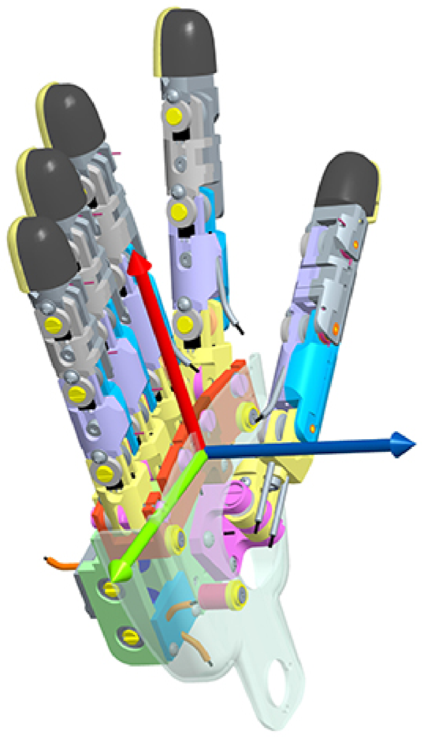


Figure 2.4 The TCP (Tool Center Point) of the iCub hand lies on the palm. The g_x axis is represented in red, g_y in green and g_z in blue.

perpendicular to the palm and open fingers). In Operation 11, the pose is rotated around the wrist pitch to avoid collision between the thumb and the object during approach.

Pose ranking The i-th candidate grasp pose generated on the superquadric cardinal points is then ranked according to a cost function, and the best candidate is sent to the robot for execution. The metric J_i accounts for two contributions, weighted by the parameter w: how reachable the candidate is by the robot TCP ($J_{i,1}$) and the hand geometry with respect to the computed superquadric ($J_{i,2}$)

$$\begin{cases} J_{i,1} = ||\tilde{o}_i \sin \tilde{\theta}|| \\ J_{i,2} = 1 - \frac{s_{hand}}{\max(s_x, s_y, s_z)} \\ J_i = wJ_{i,1} + (1-w)J_{i,2} \qquad w \in [0,1] \end{cases} \tag{2.4}$$

where $\{\tilde{o}_i, \tilde{\theta}\}$ is the axis angle representation of $R_i \hat{R}_i^T$, R_i is the target grasp orientation and $\hat{R}$ is the hand orientation that the robot can actually reach (according to the inverse kinematics solver) with respect to the root reference frame. s_{hand} is the size of the superquadric axis that

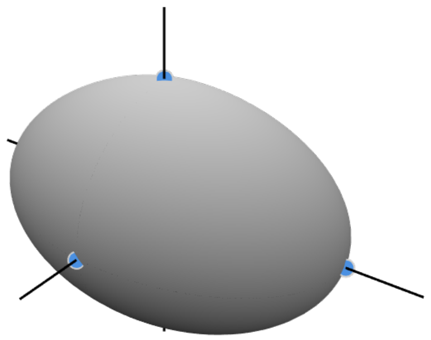


Figure 2.5 Cardinal points are defined as the intersection between the superquadric axes and its surface. The axes are here depicted as black lines, and the visible cardinal points are represented as blue solid dots.

lies in the direction of the fingers (hand x axis). The parameter w weighs the two components of J_i, where $J_{i,1}$ accounts for the orientation accuracy and $J_{i,2}$ favors grasps around the smallest side of the superquadric. Candidates that would bring the hand geometry in collision with the table surface are removed from the candidate list. In case of dual arm grasping, a full list of candidates is generated for each arm and for each cardinal point either the left-handed or right-handed candidate is selected, according to the cost function 2.4. Figure 2.3 shows examples of grasps generated and ranked with this method in a simulated environment.

2.3 Experimental setup

While our proposed approach can be adapted to any robotic setup that features any kind of 3D vision system and a manipulator with a prehensile end effector (e.g. gripper or multifingered hand), we chose the iCub humanoid robot as a test platform.

2.3.1 Robot hardware: the iCub robot as manipulator

The iCub humanoid robot is an open source research platform [71] designed around the size and appearance of a child. While it features 53 degrees of freedom in total, only 41 of these are considered in the scope of its work, since the base of the torso is supposed to be fixed. Figure 2.6 shows a visual overview of the kinematic structure of the robot. The components that are most relevant to this work (i.e. the vision system and the multifingered hands) are briefly reviewed in the following.

The iCub oculomotor system. The vision system of iCub consists in a stereo camera rig built in the robot head, equipped with a color camera (resolution up to 640x480) in each eye. This setup consists of 3 actuated DoF in the robot neck to grant roll, pitch and yaw capabilities to the head, and 3 actuated DoF in order to model the human oculomotor system (tilt, version and vergence). The iCub software library includes a gaze controller [96] that controls the head and eyes joints to fixate on a given point in cartesian coordinates in the robot reference frame. In order to obtain 3D vision capabilities, the eyes vergence is fixed and the camera extrinsics are calibrated in order to enable the use of well-established stereo matching techniques [34] to compute a disparity map. This can be further processed to obtain a depth map and a 3D color point cloud of the scene in front of the robot.

The iCub multifingered hand. iCub is equipped with a 5-fingered hand whose kinematic structure and actuation is inspired to the human physiology (Figure 2.4). Mechanically, all the fingers consist in 3 phalanges (proximal, middle, distal) but are actuated differently. The thumb, index and middle finger have an actuated proximal joint while the middle and distal joints are actuated by the same motor (underactuated). The ring and little finger are actuated by a single motor. The thumb is opposable and the other fingers have an abduction mechanism that spreads them apart. In total, each hand has 9 degrees of freedom and the capability to conform its shape to objects while grasping thanks to the underactuation. Although the torque of the motors actuating the finger motion can be measured and controlled, the torque of the underactuated joints cannot be measured directly and is therefore estimated by modeling the elastic behaviour with a SVM-based method.

2.3.2 Grasping pipeline

In this Section, we describe the main functional blocks of the complete pipeline that was set up in order to test the superquadric cardinal point grasp planner proposed in the previous sections. Figure 2.7 shows the functional blocks of the pipeline and the information flow between them. In order to maximize reusability of the software, the grasping pipeline was developed as an integral part of the Interactive Object Learning (IOL) framework[1], therefore the stereo matching, 2D segmentation, object database and kinematic solver are part of it and have not been implemented as a part of this work.

[1]http://robotology.github.io/iol

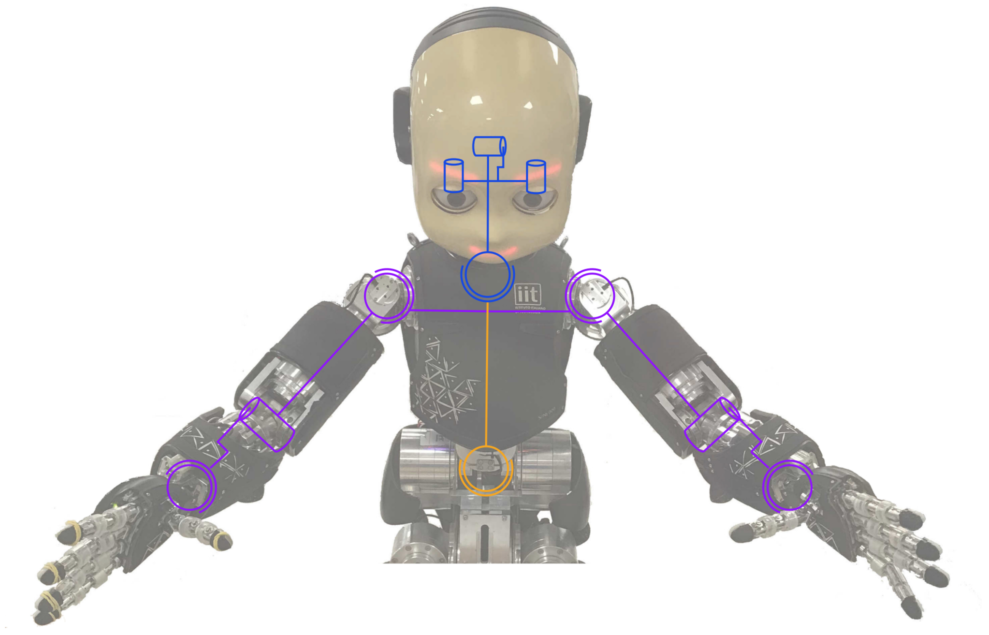


Figure 2.6 Kinematic structure of the iCub humanoid robot. In this work, the legs of the robot are not used and the torso is to be considered fixed to the workbench. The torso (3 DoF) is highlighted in orange, the arms (7 DoF each) in purple and the head and eyes (6 DoF) in blue. Each hand features 9 DoF (the kinematic structure is omitted from this image).

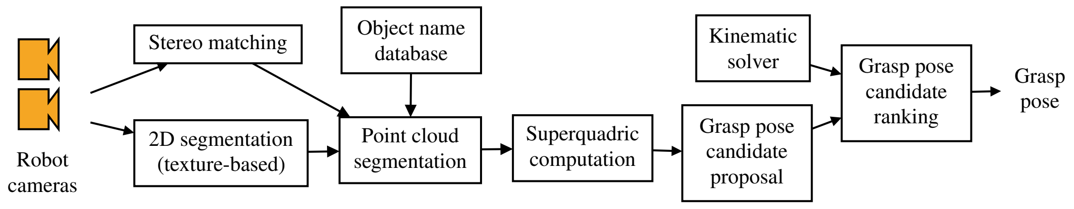


Figure 2.7 Function block representation of the complete grasping pipeline that integrates the superquadric cardinal point grasping approach outlined in this chapter and the iCub system.

Stereo matching. As explained in Section 2.3, the eyes of the iCub can perform tilt, version and vergence movements. For these grasping experiments, the vergence is fixed at 5 degrees (the robot gaze is supposed to fixate on points that are about 50 cm in front of it) and the extrinsics of the stereo setup are computed with a checkerboard pattern. Both cameras acquire 320x240 stereo images, and a disparity map is computed. The knowledge of the camera intrinsics allows to obtain a depth map and point cloud of the scene in front of the robot.

2D segmentation. The target scenario simply involves a tabletop surface with isolated objects on top of it, with no clutter. Since performing an accurate 3D segmentation on the point cloud obtained from the iCub stereo rig is infeasible (the stereo matching algorithm does not behave well on low-texture objects like the tabletop surface at our disposal), we use a simple texture-based 2D segmentation approach to segment object binary objects. It uses Local Binary Patterns [79] to extract 2D texture features from the scene in order to obtain a description of it in terms of texture areas. This information is then processed with a graph cut algorithm to extract isolated object blobs. We used the implementation of this procedure present in the iCub software library in the form of the *lbpExtract* software package[2].

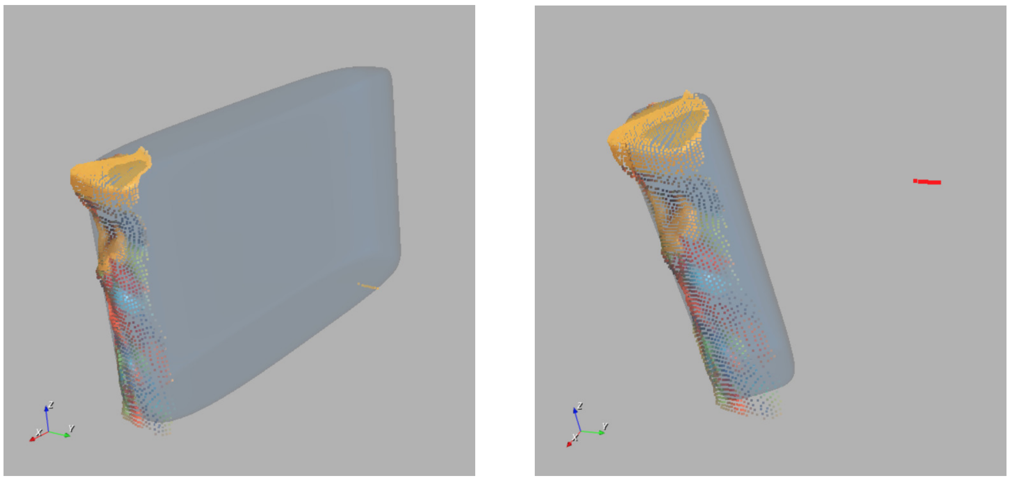
Object name database. The IOL pipeline automatically keeps track of segmented objects, their type (obtained through a classifier) and location in the robot workspace through an object database. If a number of different objects are scattered on the table in front of the robot, the robot can be prompted to fixate its attention on any one of them, selecting the corresponding segmentation mask and triggering the modeling and grasping action on the partial point cloud obtained this way. This is the only interaction required to the user by the grasping pipeline. Figure 2.9 shows an example of how the user input triggers a grasp planning action.

Point cloud segmentation. In order to obtain a partial, single view point cloud of the target object, the binary mask obtained from the segmentation stage is superimposed to the depth map. The point cloud is then computed using the intrinsic parameters of the stereo cameras.

Superquadric and grasp pose computation. Since the segmentation is performed on the camera RGB image instead of directly on the partial point cloud, the partial object point cloud can have outliers. In fact, there is no guarantee that pixels that are contiguous in the

[2]http://robotology.github.io/segmentation

2D image will be close (i.e. their Euclidean distance in the 3D space is small) to each other in the 3D space of the point cloud. For instance, a few pixels belonging to the table surface or to another object could be included in the binary object because of a small segmentation error. Since the cost function in Equation 2.4 does not account for the possibility of outliers, the resulting superquadric is often very deformed (Figure 2.8). To avoid this, our method introduces an outlier rejection stage based on DBSCAN [32], a popular density based spatial clustering algorithm, before the optimization process. An optional random sampling is also performed to reduce the cardinality of the point cloud if necessary. After this preprocessing stage, the segmented point cloud is fed as input to the superquadric modeling stage and the best grasp pose is obtained. The procedure is described in detail in Section 2.2.



(a) (b)

Figure 2.8 The superquadric modeling formulated in Equation 2.4 is not robust to outliers. In (a), the superquadric is deformed by the optimization process to include 8 outliers. In (b), outliers are filtered with DBSCAN.

Kinematic solver. The grasps, superquadrics and point clouds are expressed in the task reference frame. The iCub software library features a cartesian controller [85] that is used to solve the inverse kinematics in order to obtain the joint space configuration of an end effector pose in the 6D task space and to plan for minimum jerk motion trajectories. This software module is prompted by the grasp planner in order to solve the kinematics for a candidate TCP grasp pose (without moving the arm) and return the pose that would be reached if the motion were to be executed. In order to maximize the workspace and dexterity of the robot, we use all the joints in the torso, arms and wrists.

Grasp pose. Once the best grasp is selected, the robot reaches it and closes its hand around the object, using all the hand fingers and joints. The fingers are actuated by setting the joint targets corresponding to the fully closed hand. Each joint stops and maintains its position when a threshold torque value is reached. This allows a proper power grasping motion, allowing the robot hand to conform to the object.

2.4 Results and discussion

In this section we present and discuss the results of our experimental evaluation of the aforementioned grasping pipeline. Using a set of common household objects and toys (see Figure 2.10) we test

- the stability and computation time of the proposed superquadric fitting method
- the performance of the pipeline, evaluated with grasping experiments on the iCub humanoid robot.

In both cases, results are compared to those obtained with the method proposed proposed by Vezzani et al. [120].

2.4.1 Performance of the superquadric fitting method

Our approach shares with the one by Vezzani et al. the software package used in the implementation, as well as the cost function and a similar superquadric parametrization. The main differences between the methods are in the way the superquadric pose is parametrized; we constrain the superquadric to have the z axis perpendicular to the table surface, while the approach by Vezzani et al. uses a full 6 DoF pose. The way the optimization is solved is also different, since our approach takes advantage of analytical gradient of the cost function (Equation 2.4) instead of using a finite differences approach. We acquire 50 partial point clouds for each evaluated object and run the superquadric fitting process for each method. In Table 2.1 we report the final value of the cost function, together with a measure of the computation time. Both approaches are evaluted on the same machine.

2.4.2 Performance of the grasping pipeline

We also tested the overall performance of our grasp planning pipeline against the one proposed by Vezzani et al.. Each object in Figure 2.10 was positioned on a table surface in

Object	Final optimization error				Computation time (s)			
	Ours		Vezzani et al.		Ours		Vezzani et al.	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
Box	**0.032**	0.011	0.085	0.049	0.126	0.035	**0.105**	0.025
Toy Car	0.127	0.050	**0.038**	0.008	0.154	0.055	**0.101**	0.020
Soap Dispenser	0.078	0.082	**0.057**	0.035	0.212	0.055	**0.163**	0.056
Teddy Bear	0.110	0.011	**0.102**	0.017	0.211	0.038	**0.152**	0.030
Soda Can	**0.038**	0.010	0.054	0.018	0.162	0.028	**0.112**	0.034

Table 2.1 Comparative results of two different superquadric modeling strategies. The optimization error refers to how much the computed superquadric fits the point cloud, according to Equation 2.3. A smaller error indicates a better fit. For each object, 50 partial point clouds were acquired and processed.

front of the robot in a natural pose and the system was prompted for a grasp action. For each object, 10 grasps were performed and each grasp was considered valid only if the robot could grasp the object steadily enough to lift it off the table surface and maintain the grasp for 10 seconds. Performance for both methods is reported in Table 2.2.

2.4.3 Results discussion

As far as the superquadric modeling stage is concerned (Table 2.1), there seems to be little difference between the performance of optimization residual error between our method and the one proposed by Vezzani et al.. Unintuitively, despite taking advantage of the analytical gradient our method would seem to be slower in terms of computation time with respect to simply using a finite differences approximation to compute the gradient of the cost function in each iteration of the optimizer. This is mostly due to the analytical gradient of the complete cost function being more expensive to compute with respect to the gradient computed through finite differences. Nonetheless, we observed the outcome of the optimization to be more stable with respect to the random sampling of the point cloud being fed into the process.

In terms of overall grasping performance, however, there are some observable differences (Table 2.2) between the two approaches. While the success rate is solid for both approaches for the majority of objects, the constraints imposed on the modeling of the superquadric in our approach make a difference for some objects (e.g. Teddy Bear, Cup, Ball). In these occurrences, the superquadric shapes found by the approach by Vezzani et al. compenetrate the table surface and only approximate well the observable part of the object. In two instances,

Object	Ours	Vezzani et al.
Cylinder	**100%**	**100%**
Teddy Bear	**100%**	80%
Octopus	**100%**	90%
Juice bottle	90%	**100%**
Soda can	**90%**	**90%**
Cup	**60%**	40%
Soap Dispenser	**90%**	80%
Box 1	**100%**	**100%**
Box 2	90%	**100%**
Box 3	**100%**	**100%**
Sponge	**70%**	60%
Cube	**100%**	**100%**
Ball	**80%**	60%
Small Toy Car	**70%**	50%
Toy Car	**80%**	70%

Table 2.2 Grasping test results for the tabletop scenario on the iCub robot. For each object, 10 grasps were executed. A grasp is considered valid only if the object can be lifted off the tabletop surface and the grasp is maintained for 10 seconds.

this caused the robot hand to collide with the table while reaching for the target pose. By definition of the superquadric constraints, this does not happen in our method. Moreover, including a reachability component in the way our method ranks grasps (Equation 2.4) is a simple and effective way to filter out grasps that are out of the robot workspace or that can only be reached with unacceptable precision by the robot end effector.

In conclusion, testing the pipelines on the iCub robot shows that our approach is better suited to tackle the grasping problem in the target scenario. The constraints imposed on the superquadric in the modeling stage and on the grasp candidates in the grasp planning stage pair well with the simple structure of the task (single object, natural poses, tabletop setting). The tradeoff between computation time and stability of the obtained superquadrics is acceptable, since the scenario does not involve a time-critical aspect. However, the method proposed by Vezzani et al. would perform better in a more complex scene, for instance if the scene were cluttered and the objects were piled together (and therefore lying in a pose that might not be upright), since this would violate the assumptions we leverage in our method to obtain a more reliable object modeling. Finally, this method would be more suitable with respect to ours if a higher rate of superquadric modeling were application-critical (e.g. a moving conveyor belt, or a flying object that needs to be tracked).

2.5 Applications and dissemination

Integration in the IOL toolbox Our approach to superquadric grasp planning was developed in order to provide the iCub robot ecosystem with a relatively simple and easy to debug grasping pipeline that could be used without prior knowledge of the 3D models of the objects to grasp. In this sense, constraining the modeling of the objects in the scene was a deliberate choice in the attempt to obtain a robust pipeline to be deployed in a well-defined scenario. In fact, the target scenario of this pipeline (tabletop surface, isolated objects, simple household objects) coincides with the target scenario of the IOL (Interactive Object Learning) toolbox for iCub. The implementations of the superquadric fitting module[3] and the cardinal point grasp planner[4] are currently part of the IOL suite of open source modules, and are used as a standard grasping action.

The iCub grasping sandbox For its simplicity and interpretability, the superquadric cardinal points grasp planner was chosen as an example approach for the Gazebo iCub Grasping Sandbox [86]. Figure 2.3 shows a snapshot of the Gazebo simulated grasping demo.

2.5.1 Integration in the HRI framework

We proposed our superquadric grasp planner as a building block of the Human Robot Interaction (HRI) proposed by Nguyen et al. [78]. Such framework draws elements of both physical and social HRI (pHRI and sHRI, respectively) and integrates them in a single control system. The features of such a system can be briefly summarized as follows:

- a human-centered visual perception system that employs human keypoint estimation to map the space occupancy of the user in the robot workspace
- a Peripersonal Space representation
- a visuo-tactile reactive controller that allows the robot to safely react in both pre- and post-collision phases by combining visual information about the state of the human with tactile and skin sensor measurements

[3]https://github.com/robotology/find-superquadric
[4]https://github.com/robotology/cardinal-points-grasp

- a simple symbolic “storage” of information about entities (humans, objects, tools) the robot interacts with
- a simple behavior model supporting social interaction.

The architecture of the framework is shown in Figure 2.11. In the paper, the authors show the effectiveness of the framework by having the robot perform simple tasks that involve social and physical collaboration with a human partner. The experimental setup for such tasks is shown in Figure 2.12. The tasks involve safe human-robot and robot-human object handover, in which the robot is asked either to look for an object on a tabletop, grasp it and hand it to the partner or, vice-versa, receive an object from the human and place it on the tabletop surface for later use. In the context of this work, our proposed pipeline was successfully used to pick up objects in the robot-human handover task.

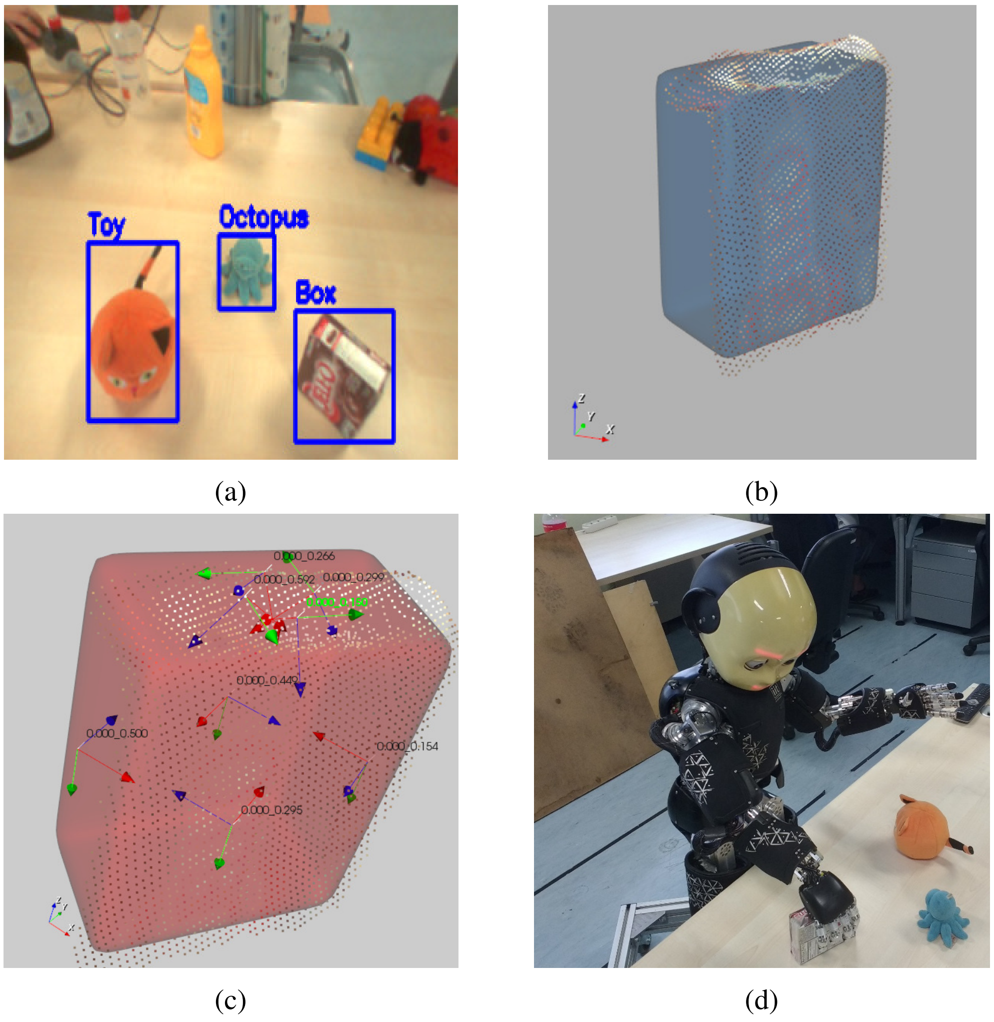


Figure 2.9 In our target scenario, the robot is put in front of a number of different objects. In (a), object detection performed within camera frames from the left eye of the robot. When iCub is prompted to grasp the Cube object, it focuses its attention on the target (if present) and the segmented point cloud is acquired. In (b) the computed superquadric is superimposed for visualization purposes to the object point cloud, and the best grasp pose is selected among candidates for the Box object (c). The candidate grasp poses are drawn as TCP reference frames (recall Figure 2.4 for a visual specification of the TCP with respect to the iCub hand), and each pose is associated to its cost function. In the case of the box, the best feasible grasp is a top grasp (denoted with a green cost function in (c)), and the robot executes it without issues (d).

Figure 2.10 Common household objects used in the experimental evaluation of the superquadric cardinal grasp planning pipeline.

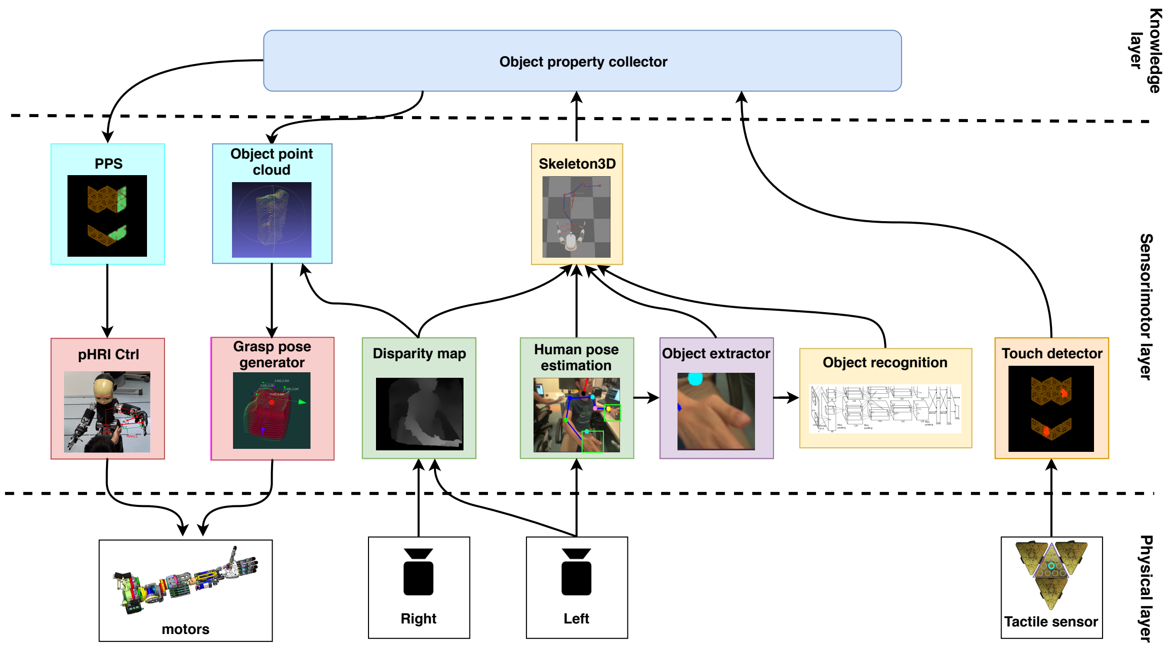


Figure 2.11 Overview of the HRI framework proposed in [78]. At the physical level, perception includes vision and touch. Low-level motor control allows specifying the position trajectories of the joints exploiting a combination of pressure (from the tactile sensors) and force information (from a number of 6-axis force-torque sensors located on the robot structure) as additional feedback. The sensorimotor layer transforms raw sensory data into symbolic tokens (e.g. object identities, posture, 3D shape, human body posture, etc.) that can be easily stored into the "object property collector" database. This symbolic knowledge is used to control action, as for example to avoid contacts rather than to grasp objects, through reasoning modules (i.e. PPS, object point cloud, pHRI controller, and grasp pose planner).

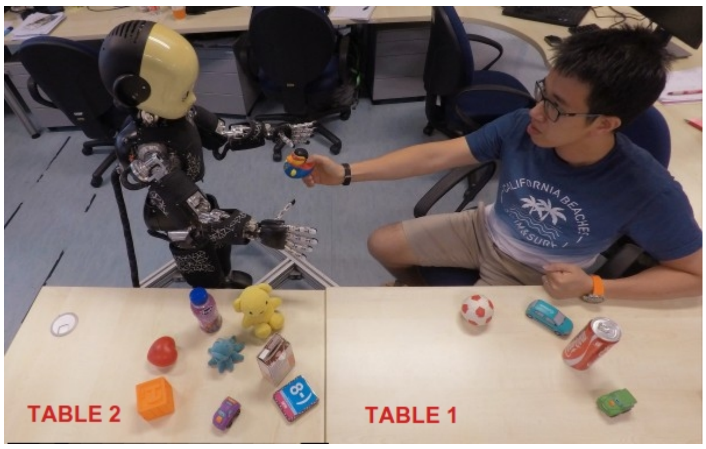


Figure 2.12 The Human Robot Collaboration (HRC) experimental setup for [78]. The human is sitting next to *Table 1* while the iCub is located near *Table 2*, effectively sharing the workspace.

Chapter 3

Fast synthethic dataset generation for instance detection and segmentation

Object localization is a fundamental component in the cognitive architecture of any robotic system that is meant to interact with environments that are not entirely and precisely structured. Vision-based robot manipulation systems, as defined in the Introduction, are no different. From industrial manipulators working in pick and place or assembly work cells to service robots looking around an environment for a specific object, object localization is the means through which they acquire the manipulation target and visually separate it from the surrounding world.

Regardless of whether the vision system provides the robot with 2D, 2.5D or 3D information about the environment, depending on the task and requirements the object localization problem (according to the taxonomy by [27]) can be declined as one of the following:

- **Object localization without classification** when the task consists in finding the location of target objects without explicitly detecting their category
- **Object detection** when the task consists in finding the bounding box (in 2D) or bounding volume (in 3D) while also inferring information about the object category
- **Object instance segmentation** when the task consists in detecting objects and attributing to each instance the pixels or points belonging to it, separating them from the environment.

In Chapter 1, we have shown how we employ instance segmentation to acquire an object point cloud from the scene (Figure 2.7) and fit a superquadric shape on it to then plan for grasps. In that case, performing instance segmentation on the point cloud was unfeasible due to the noisy depth map estimation of the background and we solved the problem by segmenting object blobs in 2D. As explained in Section 2.3, we make use of the IOL segmentation tools to perform this step. However, this method exploit bottom-up information and is therefore viable as long as the objects in the scene do not occlude each other, i.e. their 2D outlines do not touch. This tool determines object bounding boxes according to the segmented blobs, and each blob is then classified. If the object outlines touch, as it happens in cluttered scenarios, this method fails (as shown in detail in Figure 3.1).

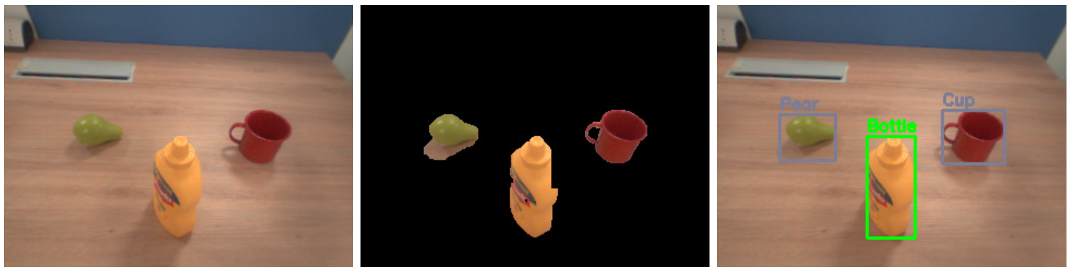
Since visually cluttered scenarios are a target of this thesis work, the limitations of the IOL (Interactive Object Learning suite[1]) segmentation capabilities had to be overcome. In the first part of this Chapter, we show how we use a popular deep learning architecture for instance segmentation to tackle this problem. These approaches outperform classical techniques for object segmentation, because they can succesfully encode semantic information related to the objects to deal with occlusions.

Deep learning methods, although powerful and scalable, are well-known to be especially data-hungry with respect to other machine learning techniques. In the case of 2D object segmentation, the annotations typically require bounding boxes information (for the object detection task) and pixel-wise binary masks for each bounding box (for the segmentation task). Although adding objects to an already existing dataset is a time-consuming and tedious task, in robotics manipulation research it is a pretty common occurrence. While the usage of synthetic rendered image datasets is an increasingly popular solution to this problem since it automatically provides both RGB data and ground truth, large-scale rendering of tens of thousands of images is not always feasible both in terms of hardware and time requirements.

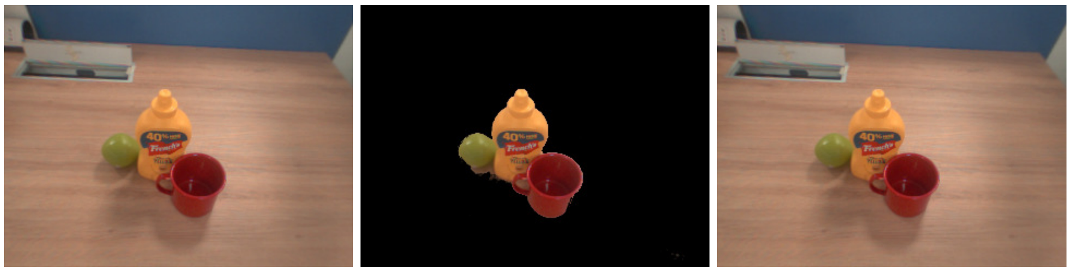
The purpose of the work contained in this Chapter is to produce a highly versatile and reusable tool that can be used as an enhanced instance segmentation tool in the IOL suite. In particular, the feature of such tools are

- provide 2D instance segmentation capabilities in cluttered scenes
- the instance segmentation architecture must be modular in order to facilitate further research and improvements
- provide a way to quickly generate new synthetic annotated datasets when objects are added or removed from experiments
- when a new dataset is produced, the instance segmentation model must be quickly fine-tunable on it.

[1]https://github.com/robotology/iol



(a) LBP-based instance segmentation of isolated objects.



(b) LBP-based instance segmentation of occluded objects.

Figure 3.1 2D instance detection using the IOL toolbox with the LBP segmentation tool. The detection works correctly if the objects are isolated as is shown in (a). If the object outlines touch each other, the heap of objects is segmented as a single instance and detection fails as shown in (b).

In Section 3.1 we give a brief overview of state of the art methods for instance segmentation and automated dataset generation for the task. Then, we describe our approach in producing a tool with the aforementioned features (Section 3.2) by combining the Mask R-CNN architecture [45] with a fast synthetic dataset generation method [30] and proceed to describe experimental results to validate its effectiveness (Section 3.3). We then show some applications of the instance segmentation tool in the context of robotic applications and published work (Section 3.4) and, finally, make some remarks and give some pointers for further research.

3.1 Related work

Before detailing our approach to the problem of quick dataset generation for instance segmentation, we make a brief introduction to the state of the art on object segmentation 3.1.1 and synthetic dataset generation 3.1.2.

3.1.1 Object segmentation methods

In general, the term *object segmentation* is used to refer to the task of labelling the pixels belonging to some object class within an image (or points in a point cloud). However, this is a very broad term and in order to avoid confusion the computer vision and robot vision community typically distinguish between the *semantic segmentation* and *instance segmentation* tasks. *Semantic segmentation* aims to classify each pixel p_i according to to a set S of classes, that can either be binary [89] or include a number of objects, without caring for specific instances. In latest years, the problem has been tackled countless times with a wide landscape of architectures [40, 51] that include fully convolutional neural network. These methods have been steadily rising in popularity thanks to the possible applications in autonomous vehicles and robotics. *Instance segmentation* instead aims to find the pixels p_i belonging to different objects, accounting for different instances of the same object. The current state of the art methods for instance segmentation typically fall in one of three families, that are outlined in the following.

Region-based methods. Broadly speaking, these feature a two-stage approach: while the first stage is tasked with analyzing the input image and finding locations that might contain objects, typically referred to as RoI (Region of Interest), the purpose of the second stage is to establish, for each RoI, whether an object is present in the RoI at all and which pixels of the RoI belong to it. Different approaches might output additional information for each RoI, e.g. the object class most likely contained in the region. A landmark of the region-based methods is Mask R-CNN [45], whose output for a given image consists in a list of bounding boxes, the object class detected in each bounding box, and a binary segmentation mask that indicates pixel-wise foreground/background classification within the bounding box. This architecture is explained in more detailed in Section 3.2.1. Other approaches [23, 54] tackle the problem in a similar way, although in these works the detection phase is performed after computing the segmentation masks instead of in parallel (i.e. what happens with Mask R-CNN).

Pixel-based approaches. The common denominator of these methods is the formulation and definition of an "auxiliary" information metric, computed for each pixel. After the metric has been computed for every pixel in the input image, clustering algorithms are typically used to gather pixels into object instances based on the distribution of such values. The works by Bai and Urtasun and Wolf et al. [3, 124], for instance, combine the watershed transform with a learned "energy field" pixel-wise metric to produce an energy map of the input image, the intuition being that local minima in such a distribution correspond to object instances.

Snake methods. These methods propose to progressively improve an initial, coarse estimation of the object boundaries by optimizing an energy metric with respect to the contour coordinates. An example of this class of methods is the work by Ling et al. [58], where a graph convolutional neural network is employed to predict vertex-wise offsets for progressive contour deformation.

3.1.2 Synthetic dataset generation for object detection and segmentation

Learning the parameters of complex models such as the ones mentioned in Section 3.1.1 requires massive amounts of annotated data. This quickly becomes expensive and impractical, driving the effort either towards datasets that are either small and very task-focused (e.g. YCB-Video [128], used for pose estimation of 21 objects) or large and more generalist (e.g. MS COCO [57] or Pascal VOC [33]). Producing images through rendering is a well-established solution for the dataset creation problem, as it alleviates the cost of manually gathering and annotating vast amounts of examples. Generally speaking, we can distinguish between approaches that produce images by either applying rendered object images on real background images or vice versa [87, 80, 75] and approaches that render the entire scene [94, 43, 36, 101]. Although the giant leaps made in the fields of GPU rendering, hardware architectures and computer graphics have made possible photorealistic scene and object renders [101], these methods are still expensive in terms of time and demanding in terms of computational resources.

In some instances, e.g. in robotics and manipulation research, target objects change frequently according to the task being tackled, and the aforementioned object detection and segmentation methods, if present, must to be retrained. In such context, since the purpose of the research is not to advance the object detection field, it is preferrable to quickly generate a dataset containing a reduced number of objects in order to train a small model with respect to generating a massive dataset containing a comprehensive amount of objects (e.g. all the 77 YCB [14] objects) in order to train a large model. The Cut, Paste and Learn approach [30] fits the role of a fast dataset generator, since it proposes to synthesize dataset images by applying existing objects cropped from real images on a number of backgrounds. While this approach might seem naive, the authors demonstrate that it constitutes a feasible way to augment or create brand new datasets for object detection. In Section 3.2.2 we show that this intuition holds validity for the instance segmentation task as well.

3.2 Methodology

The first part of this section is dedicated to a brief recall of the inner workings of the Mask R-CNN instance segmentation architecture [45] and the Cut, Paste and Learn method [30]. The last part of the section is dedicated to the description of how we used and adapted these building blocks to obtain our versatile instance segmentation tool for robotics.

3.2.1 Mask R-CNN

Mask R-CNN [45] is a deep instance segmentation framework that shares the general two-stage structure of its predecessor Faster R-CNN [93], published by the same research group. Both are aimed at detecting objects in images, with the former extending the functionality to instance segmentation.

The acronym R-CNN stands for *Region-based Convolutional Neural Network*, which suggests the two-stage nature of the approaches. Both Mask R-CNN and Faster R-CNN employ

- a first stage aimed at scanning the image, extracting features and proposing region proposals, i.e. box-bound portions of the image where objects are very likely to be found. This stage is aptly named *Region Proposal Stage*
- a second stage, aimed at refining the region proposals and classifying the content (object detection). Mask R-CNN also outputs a binary segmentation for the object contained in each box.

The Mask R-CNN architecture, depicted in Figure 3.2, can be functionally divided in four modules.

Network backbone. The term *backbone* typically refers to the CNN architecture used to extract features from the input. Agarwal et al. [2] provides more insight on the nature of the term, and how different architectures can share a backbone. As typical with CNNs, the lower layers (i.e. closest to the input) extract low-level features such as edges, corners, and color gradients, while the higher layers combine them in more expressive and complex features. In the original proposal, the authors use ResNet-101 as backbone.

Region Proposal Network (RPN). The RPN is a lightweight network that uses the features produced by the backbone to find rectangles that are likely to contain objects. Although different types of RPN exist, Mask R-CNN inherits its region proposal method from the Faster R-CNN architecture. Briefly, the RPN here scans the feature maps in a very large number of rectangular boxes with different size and aspect ratios. The scanned regions are determined by sliding each box over the feature maps with different strides, in order to cover as much of the image as possible with their receptive fields[2]. The output of the RPN is a list of bounding boxes where objects are likely to be found.

Bounding box regressor and classifier. The feature maps are then cropped according to the candidate regions and further processed in order to perform classification, i.e. establishing the type of the detected object, and regression of the minimal bounding box containing it. Since classifiers typically operate on a fixed input size, Mask R-CNN introduces an operation called ROIAlign to accurately crop feature maps and fit their size and shape to the classifier input. The Mask R-CNN classifier uses a softmax function to determine the likelihood of each dataset class plus a background class. The candidate region is discarded if it is classified as background.

Object segmentation. The main addition of Mask R-CNN over Faster R-CNN is the segmentation stage, outputting a binary mask to indicate which pixels of the image contained in each bounding box belong to the object and which belong to the background. Similarly to the classification module, the feature maps are cropped according to the candidate regions proposed by the RPN and resized to a fixed size (e.g. 28x28). The segmentation stage is essentially a fully convolutional network that outputs, for each pixel in the region, the likelihood of that pixel belonging to the foreground. The segmentation mask is then scaled back to the size of the bounding box.

Beyond its performance with respect to state of the art competitors, Mask R-CNN has some features that make it particularly interesting to our use case. First of all, the segmentation is class-agnostic, i.e. there is only one segmentation CNN for all object classes. This allows the bounding box regression, classification and segmentation tasks to happen in parallel, with a gain in performance, but most importantly it means that for small changes in the task and dataset we can effectively retrain only the classification and regression branches. Another important feature of Mask R-CNN is its modular structure, meaning every module can be swapped out with a different implementation of the same approach or even with a

[2]In deep learning literature, the region of an input that influences the activation value of a neuron is called *receptive field*. In CNNs, a high-level feature element typically has a large receptive field in the input image due to the convolutional nature of the architectures. Refer to Dumoulin and Visin [28] for an in-depth explanation.

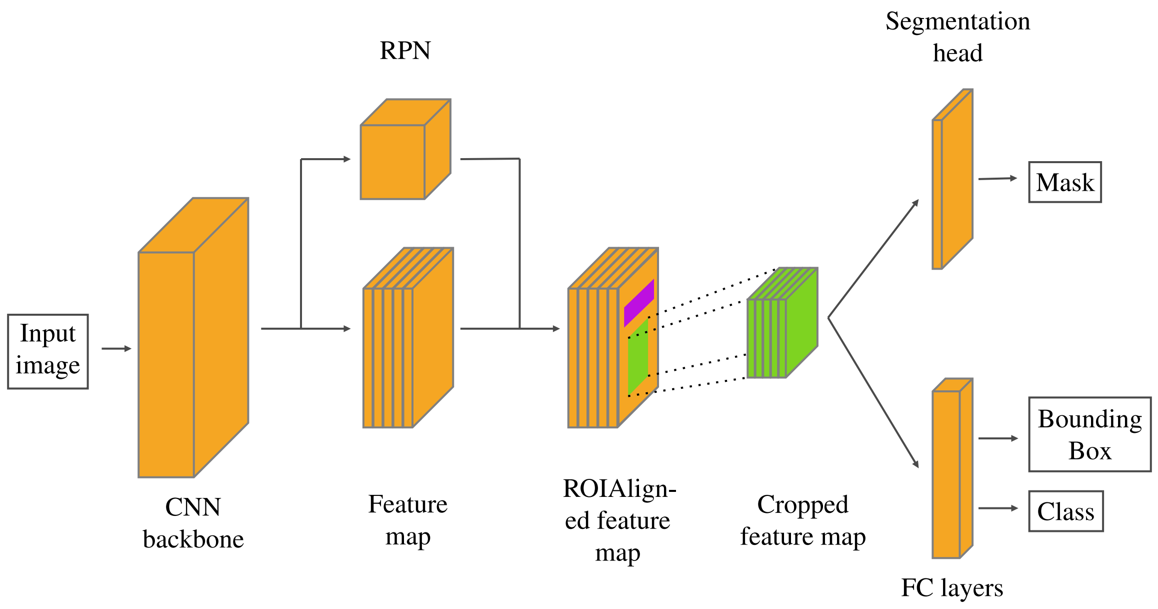


Figure 3.2 The Mask R-CNN architecture for instance segmentation.

different approach (provided the inputs and outputs remain coherent). This is in line with the aim of the work presented in this Chapter, i.e. to provide a platform that can be quickly adapted to different tasks or object sets and that allows for iterating on the design of one or more modules without the need to reimplement the whole architecture.

3.2.2 Cut, Paste, and Learn

Cut, Paste and Learn [30] (abbreviated from now on as CPL) is a method to quickly create or augment existing annotated datasets with custom objects in a way that does not hinder the performance of models trained on them. The problem the authors originally wanted to tackle was to augment existing datasets with particular instances of objects. For instance, the MS COCO dataset for object detection and segmentation features a *Bottle* class, but it is not annotated to distinguish between different bottle shapes, or soda brands. CPL would allow to augment MS COCO with specific bottle instances, each to be detected as a different class.

The proposed approach consists in what the name suggests:

- **cut** object shapes (called *patches* in the paper) from BigBIRD [104] dataset images. For the same objects, images from different viewpoints are used. Objects are segmented from the background using depth information or a simple fully convolutional neural network (in the case of translucent objects, where the depth measurements would fail)

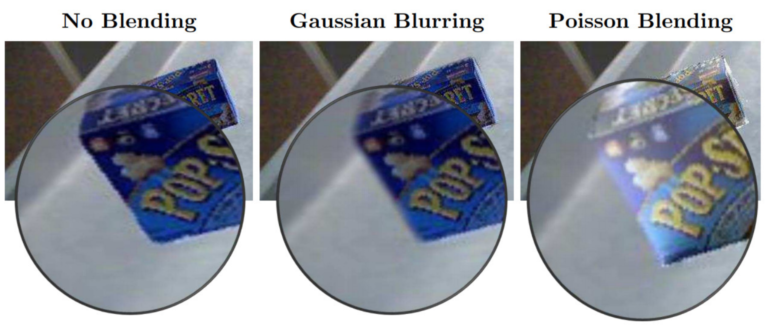


Figure 3.3 Different blending modes used by the Cut, Paste and Learn method. Image sourced from the paper by Dwibedi et al. [30].

- **paste** the object patches on real kitchen backgrounds, applying augmentations such as rotations, scaling and mirroring. To avoid creating boundary artifacts and jagged edges, pasted patches are processed with a number of blending filters. A number of different blending modes are suggested by the authors to make the training more robust to such artifacts, i.e. median blur, gaussian blur and Poisson blending [90]
- **learn**, i.e. train state of the art object detection algorithms (such as Faster R-CNN) on these synthetically-augmented datasets.

The claim of the authors, demonstrated in the paper, is that as the borders of the patches are not too different from the rest of the image, the datasets created this way perform remarkably well in real scenes if compared to a human-annotated dataset when used to train the same algorithm. Their key intuition is that state of the art object detection methods based on CNNs do not focus on global features (e.g. physical feasibility of the object pose in the scene, validity of perspective projection, etc) but on local features such as edges, object texture and color gradients. In our work, we aimed to verify if this claim still stands when instance segmentation architectures (i.e. Mask R-CNN) are trained on datasets generated or augmented in a similar way.

3.2.3 Our contribution

We now present the details of our approach in modifying CPL to automatically generate annotated datasets in order to train a Mask R-CNN model.

Implementation of the Mask R-CNN architecture

We used the out of the box implementation of Mask R-CNN by Abdulla [1] as a starting point. Even though it features some minor implementative differences with respect to the paper, we found its documentation to be more user-friendly and its codebase easier to fit to our needs with respect to the original, and at the time of working on this subject (January 2019) the Detectron2 [126] implementation was not yet public. The most relevant difference in the Mask R-CNN architecture we used with respect to the paper by He et al. is that we used a different backbone, preferring a FPN-enhanced ResNet50 [56] to a ResNet101. The choice was made considering that in our final use case, i.e. live object instance detection on robotic setups, a smaller and lighter backbone would allow faster inference times, and therefore a higher detection rate, when deployed on robot hardware.

Objects and backgrounds

The authors of the CPL paper used the BigBIRD [104] as a dataset to extract object patches from. The availability of physical YCB [14] objects in our laboratory steered us towards the use of such dataset instead; in fact, both BigBIRD and YCB datasets come as a set of object RGB-D images taken from many different viewpoints (the physical turntable and camera rig used to acquire the data is very similar). The subset of YCB objects chosen for this work matches the 21-class subset used in the YCB-Video dataset for pose estimation [128], since such dataset comes with images annotated with bounding boxes and segmentation masks for each object instance in a scene, and is therefore suitable for an architecture such as Mask R-CNN. In order to construct our instance segmentation synthetic dataset, we use real images from the YCB object set, as described in Section 3.2.3. For each of the 21 YCB-Video object classes, we used 600 frames from all the viewpoints available in the YCB dataset, using 1 frame out of 10 consequent frames for the validation set. In total, we used 11361 object images (with relative segmentation masks) to generate object patches for the training set and 1239 for the validation set. Some of these images are shown in Figure 3.4.

In order to provide backgrounds to populate with object patches, we gathered a dataset of real rooms and desks in the lab premises. We shot 19 videos in order to obtain frames for the training sets and 2 videos for the test set. The video frames were downsampled, keeping 1 out of 10 frames in order to eliminate visual similarity between consequent frames. This procedure generated 5373 background frames for the training set and 544 frames for the test set. Some sample background scenes can be seen in Figure 3.4.

Dataset generation using CPL

Here we describe the process we used to generate custom datasets to train Mask R-CNN for our custom task. We mainly follow the CPL method as described in the paper, with some modifications to adapt it to our use case. For the sake of brevity, from this point onwards we shall refer to this type of dataset with the term *tabletop dataset*.

Extracting object patches and segmentation masks. The YCB object set comes with RGB-D images and a segmentation mask for each object, extracted from depth measurement. Since depth maps can be noisy near the object base, we used basic image processing tools (e.g. binary morphology tools) to remove small outliers in the segmentation masks. Since RGB and binary maps have the same size, object patches can easily be cropped from RGB images with such masks (Figure 3.4).

Adding objects to backgrounds. For each dataset image to create, a background scene and a number of object patches are randomly selected, respecting the training/test data split. Although the number of objects that end up in each dataset image is random and can be given lower and upper bounds, we decided to paste between 3 and 5 object patches in each image. This is to mirror the structure of the YCB-Video dataset, that features an average of object instances per frame just below 4. The first difference between the CPL method and our own is how the location of the patches is chosen when they are added to the image. While in the CPL paper this is sampled from 2D uniform distribution, we sample from a 2D Gaussian centered in the image center. This encourages dense scenes with a cluster of objects around the center, which is exactly our target scenario. Another difference with respect to the paper is that, since we need the segmentation masks in order to train Mask R-CNN, each RGB image I_i entry in the tabletop dataset is accompanied by another, gray-scale image M_i containing the segmentation masks. These are formed by pasting the segmentation mask of each object patch on a black image, attributing to each instance a different grey value, even if it refers to instances of the same object. In case of overlap between objects in the RGB images, the corresponding masks also overlap accordingly. Figure 3.4 shows an example of such composite segmentation image.

Patch augmentations. Just as in the paper, we apply scaling and rotation to object patches before pasting them. Regarding scale, scaling up or down a patch too much results in interpolation artifact, therefore our method differs from the one used in the paper. In order to keep resolutions and sharpness consistent, we first scale the backgrounds in order to obtain a similar sharpness to the YCB RGB-D images. This is done by resampling the background image. Only then, we apply a modifier between 0.8 and 1.2 to the patch size and paste it on the dataset image. Regarding rotation, we apply a random rotation sampled from a uniform distribution $\in [0, 2\pi]$. We also add a mirror augmentation to the patch with probability 0.5.

Handling occlusions. Since our target scenario involves cluttered object scenes, we allow overlap between object patches (and relative segmentation masks). While the original CPL method uses the Jaccard index[3] of the object bounding boxes as a measure of occlusion between objects, we use a custom occlusion metric J_t that accounts for both bounding boxes and segmentation masks. If P_i and P_j are two overlapping patches, we define such metric as

$$J_t(P_i, P_j) = \frac{1}{2}\frac{A_{bb}(P_i) \cap A_{bb}(P_j)}{A_{bb}(P_i) \cup A_{bb}(P_j)} + \frac{1}{2}\frac{A_m(P_i) \cap A_m(P_j)}{A_m(P_i) \cup A_m(P_j)} \tag{3.1}$$

where $A_{bb}(P_i)$ defines the area of the bounding box containing the i-th object patch and $A_m(P_i)$ defines the area of the binary segmentation mask of the i-th patch. Both areas are quantified in pixels. For our experiments, we observed $\max(J_t(P_i, P_j)) = 0.3$ for any two patches (i, j) in a synthesized image produce a good level of occlusion.

The original CPT method adds in un-labeled distractor objects to train the detection in a more robust manner. In our own observations, however, this does not seem to lead to relevant improvement in performance. Besides, the tabletop cluttered scenes we aim to segment do not contain any object that is not in the dataset. Hence, we do not add any distractors to our images.

Blending modes. The original CPL method employs 4 blending modes: *Gaussian blur*, *motion blur*, *Poisson blending* and *box filtering*. We make no modifications to the blending modes in our own work, and use them as the original authors intended. The blending is only applied to the RGB data of the dataset images, as segmentation masks have no transparency channel.

3.3 Experiments

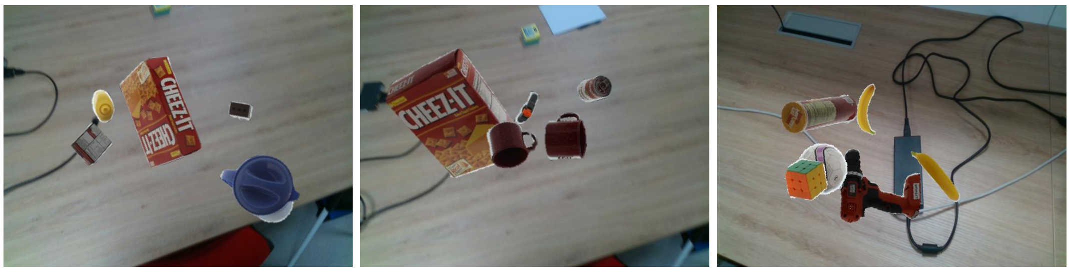
We tested our approach by generating a tabletop database composed of 20K training images and 2K test images. The parameters of the generator were tuned in order to produce images resembling scenes from the target scenario:

[3]colloquially, the Jaccard index is also known as Intersection over Union, IoA.

(a) Images from the YCB dataset

(b) Binary masks for the YCB dataset images

(c) Sample background scenes


(d) Sample images for the tabletop dataset

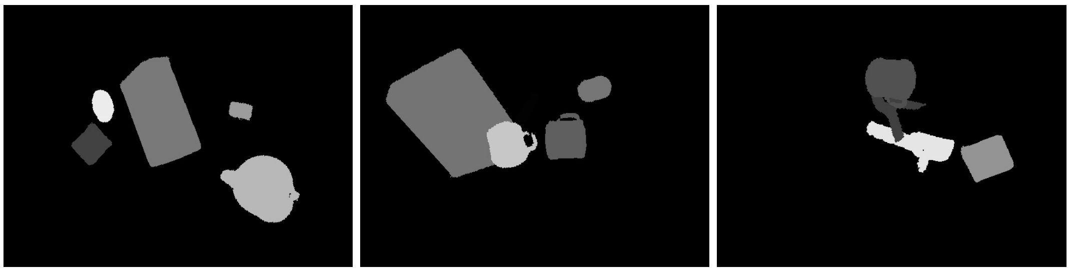
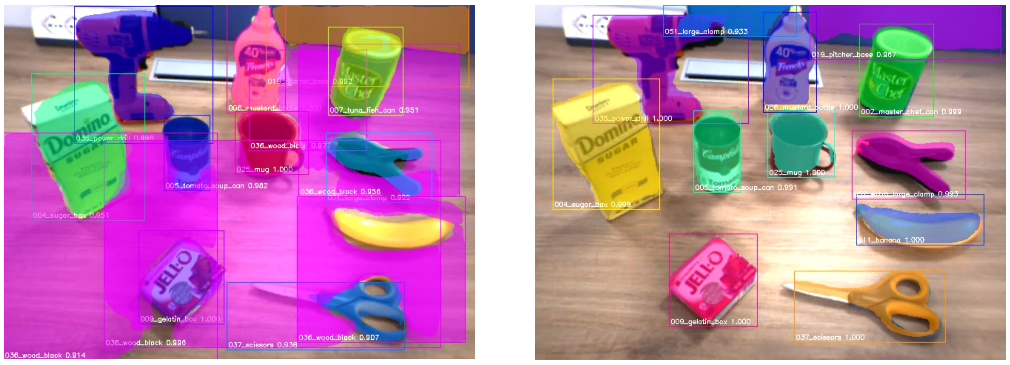

(e) Ground truth masks for the tabletop dataset images

Figure 3.4 Samples from each stage of the synthetic dataset creation procedure.



(a) (b)

Figure 3.5 Samples of instance segmentation output from a real scene using a Mask R-CNN model trained with our approach. In (a), the purple regions correspond to false detections of the *wood_block* object. (b) shows the output of the network when trained on a similar dataset that does not include the YCB *wood_block* object.

- **Objects:** 20 YCB-Video objects
- **Blending modes:** none, box, motion, gaussian, Poisson
- **Objects per image:** 3 to 5
- **Augmentations:** rotation $\in [0, 2\pi]$, scaling factor $\in [0.8, 1.2]$, mirror with probability 0.5
- **Maximum allowed occlusion:** $\max(J_t) = 0.3$.

The generation of 22K images according to our method took 45 minutes using 8 cores of an Intel Xeon server-grade processor. Note that YCB-Video includes 21 objects, while we used 20. The last object is a wooden brick, and we did not include it in our experimental dataset as it exhibits similar wood grain and texture to the desks that are present in the background images. This would cause a massive number of false positives, as shown in Figure 3.5.

As already mentioned in Section 3.2, we used a Mask R-CNN model with a ResNet50 Feature Pyramid Network backbone as target architecture. In order to minimize training time, we used a set of weights pre-trained on MS COCO and fine-tuned only the network heads[4], i.e. classifier, bounding box regressor and segmentation mask CNN. While fine-tuning more

[4]This is a common practice, known in the deep learning literature as *transfer learning*. Training large architecture models from scratch every time might not be feasible, in terms of required time and amount of training data. To circumvent this, deep learning practicioners often "transfer the knowledge" between a model learned on a large dataset to a similar architecture that has to be retrained for a different task, or dataset.

Detection threshold	Metric	Tabletop synthetic dataset	YCB-Video
70%	mAP	0.69	0.58
	AP_{50}	0.88	0.84
	AP_{75}	0.66	0.66
90%	mAP	0.66	0.55
	AP_{50}	0.83	0.79
	AP_{75}	0.76	0.64

Table 3.1 Quantitative performance of the Mask R-CNN when trained on the synthetic tabletop dataset. Results are reported for two different detection thresholds, i.e. the minimum confidence value for a single class.

stages of the model could produce higher segmentation performance, it would have taken significantly longer to train. We used a simple early stopping technique to stop the training when the validation error (half of the test images were held out for performance evaluation, while the other half was used for validation) would register a significant rising trend. On a nVidia P100 card, in most of the experiments the model would overfit after 2 hours of training and 6 epochs.

In order to probe the generalization capabilities of a synthetic-only dataset training on real scenes, we tested the model against the test set of the tabletop dataset used for training and a subset of YCB-Video test sequences. We report the performance in terms of average precision[5], obtained by generating 10 different tabletop datasets and training 10 different models. The evaluation is performed using two different minimum detection threshold, i.e. the minimum confidence for a single class that must obtain in the classification stage in ordered for the bounding box to not be discarded. Performance is reported in Table 3.1.

Quantitatively, the results indicate that the model fine-tuned on a dataset that only contains images generated with a CPL method can be used on real scene images. As expected, the model performs worse on YCB-Video test images than its own test set, but proves that the CPL method can also be extended to instance segmentation tasks if the proper modifications to the dataset generation phase are made.

Qualitatively, this approach turned out to be sufficient for our use case. The work outlined in this Chapter was not meant to be a comprehensive and exhaustive study of the CPL

[5]Average precision (AP) is a widespread metric for measuring the performance of object detection approaches. It computes the precision value for different recall values. mAP indicates medium average precision, and averages the precision for recall values from 0 to 1. For more details, refer to the Pascal VOC paper [33], where a definition and explanation of these metrics is given.

technique applied to the instance segmentation problem. As remarked in Section 3.5, it was merely supposed to work well enough to surpass the limitations of the LBP-based segmentation module. Moreover, while not performing as well as other state of the art approaches in terms of either object detection rate or segmentation quality, reducing the time and effort needed to fine-tune the network to a different object set or a different scene composition proved to be a valid tool. In particular, its usefulness became clear in some applications that are part of this Thesis and also in the context of collaborative efforts, as outlined in the next Section.

3.4 Applications

In this Section, we show some applications of the work in this Chapter by briefly presenting some instances where having a tool to quickly adapt a powerful instance segmentation model to a custom dataset is fundamental.

3.4.1 Replacement of *lbpExtract* in the IOL suite

As explained in the introductory section to this Chapter, the standard segmentation tool in the IOL suite of software modules for the iCub robot shows some limits in dealing with cluttered scenes and surfaces with non-uniform textures. Although its usefulness is unquestionable in some scenarios, in terms of both deployability and throughput, a YARP[6] wrapper for Mask R-CNN was inserted in the IOL suite of modules in order to take over the instance segmentation functionality in more complex scenario. As described in Section 4.4 and shown in Figure 4.6, a custom dataset for GRASPA objects using markers as distractors was created and used to train a Mask R-CNN model to perform instance segmentation on the benchmark layouts, where the *lbpExtract* module would fail.

3.4.2 Segmentation input for MaskUKF

We contributed to the development of MaskUKF[88], an approach to 6DoF pose estimation from RGB-D images that proposes an alternative to state of the art pipelines. This work proposes a step back from the current paradigm of end-to-end deep learning pose estimation network by using a two stage system; the first stage consists in a Mask R-CNN model trained on target objects and provides segmentation masks for detected objects whose pose has to

[6]YARP, or Yet Another Robot Platform, is the middleware used by the iCub software ecosystem.

be detected, while the second stage uses this information to segment input RGB-D data and estimate the object pose with an Unscented Kalman Filter [121]. The paper shows how this approach not only outperforms some state of the art approaches, but also provides an estimation of the object motion state and direction, that is crucial for closed loop control in robotic applications.

Another noteworthy feature of MaskUKF is that it does not require a large dataset annotated with 6D poses, since the instance segmentation network works in the 2D domain and it is the only component that requires a training phase. Hence, adapting this model to a different set of target objects is only a matter of training the Mask R-CNN network and providing 3D models of the objects to the UKF stage. With our contribution, a dataset can be quickly generated and the network can be fine-tuned in only a few epochs. This provides a very convenient advantage over deep pose estimation architectures such as Pose-CNN [128] or PoseRBPF [25] in terms of dataset size, type and training time requirements.

3.5 Remarks and Conclusions

In this Chapter, we have shown how we combined a relatively straightforward synthetic dataset generator and a state of the art instance segmentation architecture to obtain a versatile tool that enabled us to adapt to fast experiment prototyping and deployment on the robot platform. Altough the generated images do not look nearly as good as their photorealistic rendered counterparts and the trained models do not exhibit state of the art performance, its purpose was mainly to provide a segmentation stage good enough for the tasks at hand. On the basis of the applications in terms of both research work and lab demos, we believe our approach fulfilled the needs that sparked its inception.

Even though this Thesis work is not focused on improving the state of the art on instance segmentation tasks, we can point out one relevant shortcoming of the work outlined in this Chapter. In our review of the object segmentation state of the art (Section 3.1.1), we mainly focused on deep architectures that are trained in an end-to-end fashion using backpropagation and stochastic gradient descent. Although it can be argued that the training can be limited to the most high-level stages of the architecture like we did, effectively reducing training time and transfering parameters learned on much larger datasets, this process weakens the performance of the model quite significantly. Moreover, factoring in the time needed to optimize the generator parameters and create a new dataset results in a method that can still be too slow for many applications (e.g. in our target tasks, generating a dataset and training the model on it took typically took no less than three hours with dedicated hardware).

As a development and research direction for this work, we point to some approaches that address online learning for robotic applications. Such methods employ hybrid architectures that integrate CNNs a feature extractors and efficient kernel-based methods [98] to allow online learning for object classification [84], object detection [65, 66] and instance segmentation [16] tasks. This kind of approach solves the need for an annotated database and for stochastic gradient optimization by freezing the feature extraction part of the end-to-end architectures and allowing the user to simply acquire visual object samples from the robot itself at runtime and wait about 20 minutes for the retraining.

Chapter 4

Benchmarking grasp planners

The pillars and defining features of the scientific method are the repeatability and reproducibility of experiments, the objectivity in evaluation protocols and the clarity in the way results are reported. In many disciplines, the use of benchmarks is a widespread and scientifically meaningful practice to validate the performance of different approaches to the same task. In the computer vision field, for instance, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [24, 99] has become a landmark for researchers tackling image classification and object detection tasks. Over the last decade, wildly different architectures and algorithms have competed in the same evaluation environment, on the same images, and their performance was quantified with the same metrics. Unfortunately, it is not yet possible to draw a parallel to the ILSVRC in the context of robot manipulation. Although robot challenges exist [20] and in recent years the use of standardized object sets has started to become common practice, no dominant protocols and metrics to test grasping pipelines have taken root yet. In an attempt to address this lack of standardization we have proposed a benchmark to test the effectiveness of grasp planners and grasping pipelines on physical robot setups. In Section 4.2 we describe in detail how the approach tackles the complexity of such pipelines by proposing different metrics that account for the features and limits of the test platform and an experimental protocol needed to evaluate such metrics. As an example application (Sections 4.4 and 4.5) we have successfully deployed our benchmark on two very different robot platforms in order to assess and compare the performance of several state of the art grasp planners. The final section of this chapter contains a discussion on the results we obtained, detailing how the indicators included in the proposed benchmarking approach can provide insight into the performance of every step of the grasping pipelines. According to the experience gathered with the deployment of the benchmark, we also propose possible improvements towards a successive release.

4.1 Fairly benchmarking grasp planners on real robots: the GRASPA benchmark

In recent years, many robotic grasping pipelines have been proposed in the literature featuring consistent differences in hypotheses, methodology and experimental evaluation, in particular with respect to the objects and robotic platform used [10]. Given such variability, reproducible test conditions, a standardized set of objects, a benchmarking protocol and a suite of metrics are fundamental to make fair performance comparisons. Although a subset of the manipulation research community has already converged on a standard set of objects (i.e. the YCB object and model set [15, 14]), a widespread protocol and a system of metrics for properly comparing different pipelines are still missing.

Validation of candidate grasps in simulation alone with force closure quality measures [95] has been proven to be unreliable [48]. Such a limitation, together with the lack of a dominant metric, led to the common practice of empirically testing grasp pipelines with a simple success rate over a given number of trials and objects [82, 74, 61, 62, 120]. However, this kind of binary metric is somewhat limited, since it has no means of decoupling limitations of the algorithm itself from those of the test platform. To clarify this, consider the following scenarios:

- the grasp planner is not aware of the robot workspace, and outputs a good grasp pose that is not reachable with precision by the end effector
- the grasp planner is aware of the robot workspace, but it plans for a poor grasp pose
- the grasp planner is aware of the robot workspace and it outputs a good grasp pose, but the camera extrinsics have not been calibrated properly.

In all of these, the experimenter using a simple binary success/failure metric would simply record a failure, since telling the different scenarios apart is rarely straightforward in practice. Testing the same pipeline on a different robot setup could easily lead to different results, and the experimenter would record a different performance. While this is acceptable if the goal of the experiment is simply to benchmark the performance of the overall system (e.g. the system is being sold commercially as a package), on the other hand being able to decouple and measure the performance of the single components is desirable (e.g. the experimenter is benchmarking grasp planners). As a side benefit, this feature would enable the experimenter to debug the system as well.

According to these observations, we proposed **GRASPA v1.0 (GRASPA is a Robot Arm graSping Performance benchmArk)**, a benchmarking protocol and a set of metrics to evaluate the performance of grasping pipelines. It aims to fairly compare methodologies tested on different robots by measuring and accounting for platform limitations that might hinder the overall performance. The proposed benchmark features:

- printable layouts of predefined grasping scenarios (populated with YCB object subsets) equipped with localization markers to enhance test reproducibility
- a protocol to assess the robot reachability and the calibration of the vision system within the defined grasping setup area
- a widely-used grasp quality metric to evaluate candidate grasping poses before their physical execution
- a score to assess grasp stability during the physical execution on the robot
- possibility to benchmark the pipeline either in isolation or in clutter, with the definition of a further metric to evaluate the obstacle avoidance in the latter case
- a composite score to quantify the overall performance of the pipeline.

We published on GitHub[1] the code for computing the benchmark scores and instructions on how to collect the required data. Additionally, we made available a Docker container to ease installation and a cloud hosted environment to test the code without requiring any installation.

We employed GRASPA to assess the performance of several grasping pipelines, on two different robot setups. We used the iCub humanoid robot platform to benchmark the grasping pipeline proposed in [78], while we used a Franka Emika Panda manipulator to benchmark pipelines based on GPD [82], Dex-Net [62] and the superquadric grasp planner by Vezzani et al. [120]. The results we obtained are available[2] as well as an example implementation of the procedure used to gathered such data.

4.1.1 State of the art and related work

In recent years, the success of data-driven methods has brought new ideas and advancements in the field of robotic manipulation [10, 13, 82, 74, 61, 62], at the same time pushing the

[1]https://github.com/robotology/GRASPA-benchmark
[2]https://github.com/robotology-playground/GRASPA-test

community towards testing applications on common sets of both real objects [68, 47, 14] and meshes [104, 17, 35] to develop benchmarking protocols. Notably, the YCB object set [14] has had great impact on the robotic manipulation community, enhancing the reproducibility of experiments by providing both physical and mesh representation of 77 objects drawn from different categories (food, kitchen items, tools, shapes and task items). Despite the complexity of grasping pipelines and the variability in test setup design, however, most of the available benchmarks meant to be deployed on real robots are based on simple success/failure binary evaluation metrics[3].

Challenges such as the Amazon Picking Challenge [20] and RoboCup@Home [107] proved to be quite effective in benchmarking entire autonomous pipelines by defining strict rules and tasks. However, in these contexts the tasks themselves are often difficult to reproduce and the number of accepted teams is typically small.

The VisGraB benchmark [49] presents a toolbox to evaluate vision-based grasp planners in simulation. VisGraB provides real stereo images of objects in various conditions and a software environment to analyze the quality of user-planned grasps in a simulated environment. A similar idea lead to the recent GraspNet-1Billion [35], where the authors propose a large scale dataset of RGB-D images annotated with more than one billion grasp poses and a unified evaluation system based on analytical metric computation in a simulated environment. The dataset and evaluation system can be used for training grasp detectors (as the authors propose in the paper) but also as a benchmark for other approaches. Despite the usefulness of the datasets and the simulated environment evaluation suite, neither VisGraB nor GraspNet-1Billion account for any real execution of the task, the type of the manipulator and gripper or the aspects of grasping pipelines that lie beyond grasp detection (for instance the visual calibration and robot workspace).

The ACRV benchmark [52] and the one published by Triantafyllou et al. [111] tackle the issue of reproducibility by proposing a set of objects and layouts for industrial shelving and pick and place applications. Both argue that physical execution of the task is essential in evaluating the performance of pick and place pipelines, although their protocols do not account for test platform limitations and the score metrics do not provide insight on the performance of single pipeline steps.

[3]http://www.ycbbenchmarks.com/protocols-and-benchmarks

4.2 Benchmarking protocol

In this Section we outline the proposed benchmarking protocol, focusing on the design of the grasping layouts, and the metrics to evaluate the individual pipeline steps.

4.2.1 Benchmark layouts

GRASPA is designed to evaluate grasping pipelines on an area located in front of the robot with dimensions 594x420 mm (A2 standard paper size), resulting in the setup shown in Fig. 4.1. GRASPA uses a subset of the YCB object set (see Fig. 4.2), selected in order to include a range of shapes, dimensions and challenges for the grasping task. We propose 3 scenarios of increasing complexity in terms of number, shape and pose of the included objects (see Fig. 4.3a to 4.3c). Moreover, GRASPA can evaluate pipelines that work both *in isolation* (i.e. one object at a time in the layout) and *in clutter*[4] (i.e. all objects at the same time). In the latter case, the added challenge is accounted for in the final score.

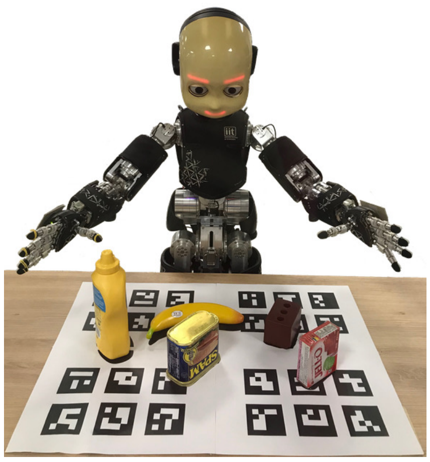
The 6D object poses are fixed and expressed with respect to the layout reference frame shown in Fig. 4.3a to 4.3c. To this end, an ArUco marker board [41] is embedded in the printable layouts to enable the experimenter[5] to estimate the layout reference frame pose in a robust way. Users need to express all the information collected during the benchmark procedure with respect to the layout reference frame so as to be independent from the position of the physical board. Finally, we provide printable layouts of dimensions 594x420 mm (i.e. A2 format) that include markers and object footprints (e.g. Fig. 4.3d).

4.2.2 Reachability within the layouts

Depending on the testing platform, the robot arm size, mechanical structure or joint range limits may impair the capability of the end-effector to reach some layout regions with accuracy. Therefore, grasps in these regions might fail regardless of the performance of the planner. To avoid penalizing planners for the limits of the test platform, an index of reachability over the layout area must be included in the benchmark. In GRASPA, we adopt an empirical approximation of such measure by dividing the layout area in 6 regions, each with a reachability score $S0_i$ for $i = 1, \dots, 6$ (Fig. 4.4a). The reachability score $S0_i$ for each region is defined over a set of poses uniformly distributed over the layout area with different

[4]In this work, we refer to clutter as a situation where the objects are visually occluded (as long as a top down view is not used) and the presence of objects limits the task space of the robot while planning for grasp and avoiding collisions.

[5]From this point onwards, we refer to the experimenter as "the user".


(a) iCub robot setup.

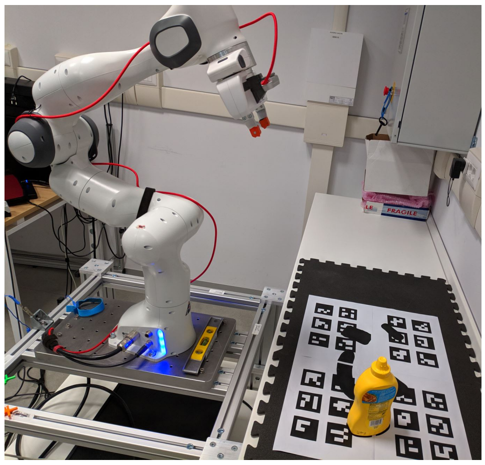

(b) Panda robot setup.

Figure 4.1 The robot setups where GRASPA was deployed.

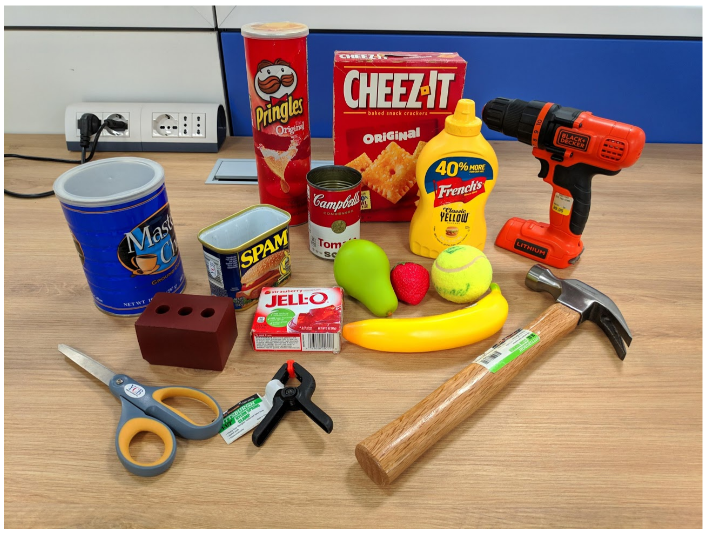
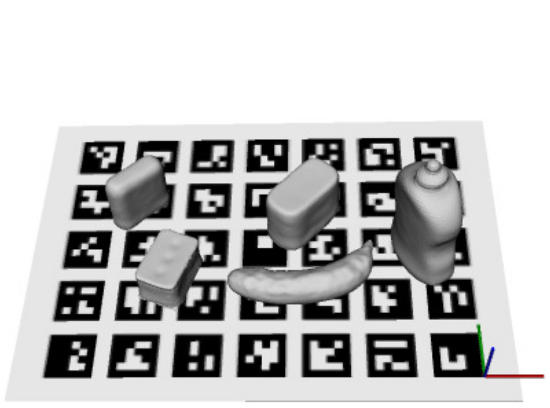
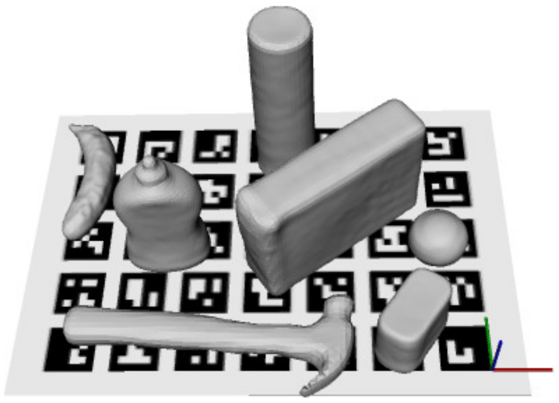
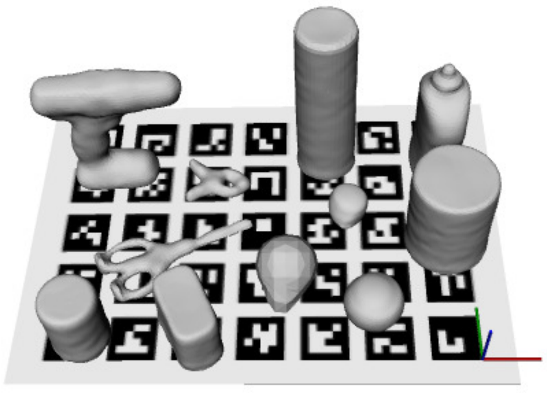


Figure 4.2 The subset of YCB objects selected for GRASPA 1.0.


(a) Rendered view of layout 0.


(b) Rendered view of layout 1.


(c) Rendered view of layout 2.

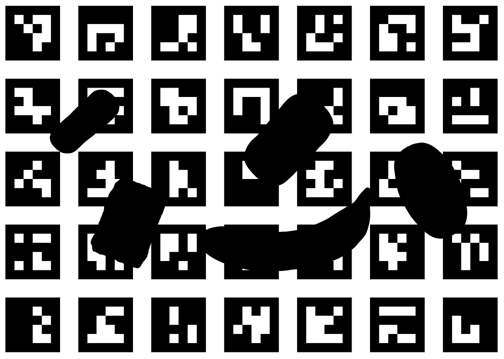
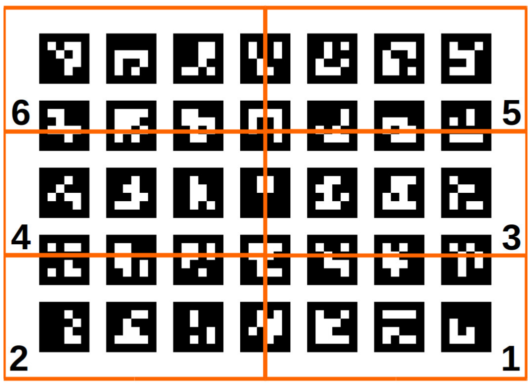
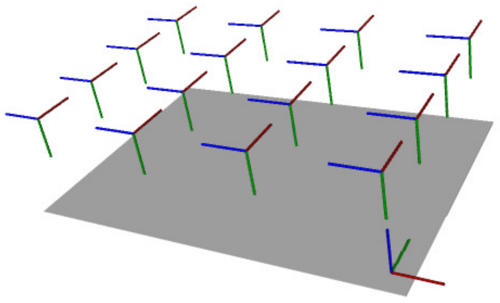

(d) Printout of layout 0.

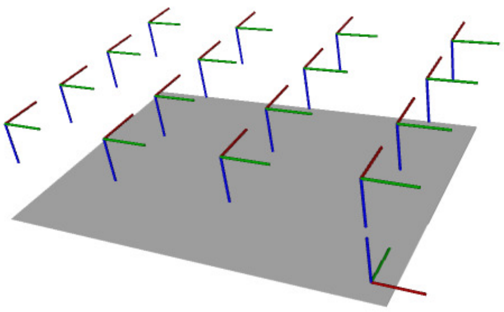
Figure 4.3 From (a) to (c): the 3D renders of the three layouts defined within the benchmark. (d) shows one of the provided printable boards that allow for reproducibile object placement on a physical setup.



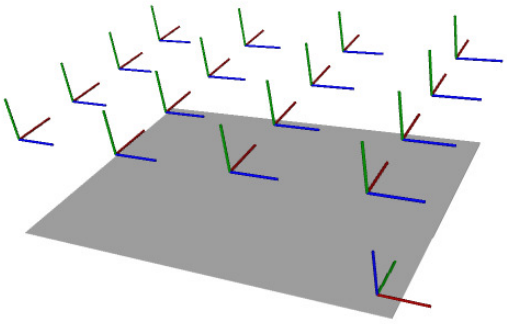
(a) Reachability/calibration regions.



(b) Reachability pose set 0.



(c) Reachability pose set 1.



(d) Reachability pose set 2.

Figure 4.4 (a) Regions used to determine the robot reachability and the calibration of the vision system over the layouts. (b) to (d) Poses the end effector of the robot has to reach for in order to test how well its workspace covers the layouts. Reachability pose set 1 is also used later to test camera calibration.

orientations (Fig. 4.4b). The user makes the robot reach (or attempt to) for these pre-defined poses and then acquire the ones actually reached by querying the forward kinematics. Poses placed on the boundary of contiguous regions are considered to belong to both regions.

The score $S0_i$ for the i-th region is given by:

$$S0_i = \frac{N_i^{reached}}{N_i^{tot}} \in [0,1], \tag{4.1}$$

where $N_i^{reached}$ is the number of poses in region i actually reached by the robot with a given accuracy and N_i^{tot} is the number of poses belonging to the region i.

A pose l is considered to be reached if the position and orientation errors (E_l^p, E_l^o) are smaller than the thresholds (τ_p^r, τ_o^r) defined by the user. In Section 4.3.2 we elaborate more on such thresholds. The errors are computed as follows:

$$E_l^p = \|p_l^{reached} - p_l^{desired}\|, \tag{4.2}$$

$$E_l^o = \sin(\alpha_l^{error}), \tag{4.3}$$

where α_l^{error} is the angle of the equivalent axis-angle representation of the matrix:

$$R_l^{error} = R_l^{desired} R_l^{reached}, \tag{4.4}$$

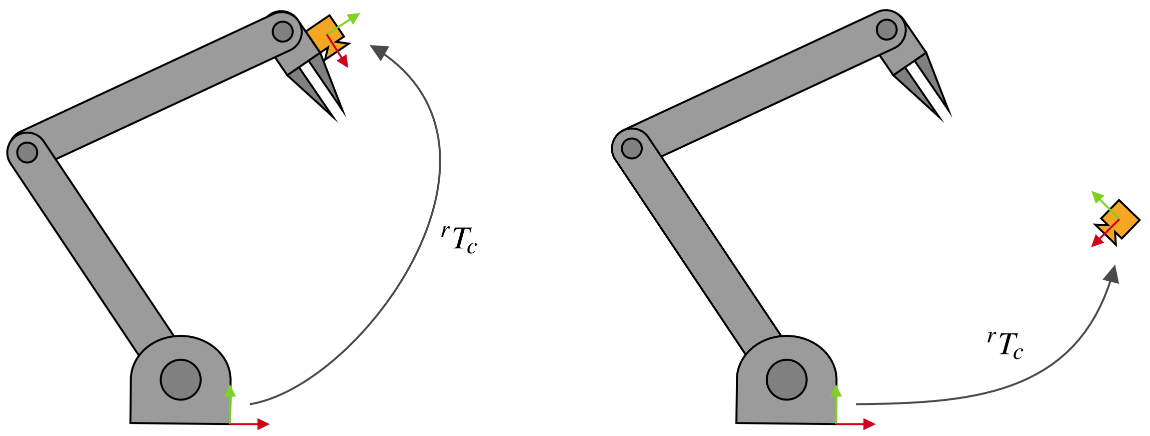
with $R_l^{desired}$ and $R_l^{reached}$ respectively the desired and reached orientation matrix relative to pose l [103].

For each benchmark layout $L \in \{0,1,2\}$, we associate to each object $k = 1,\dots,N_L^{obj}$ (with N_L^{object} being the number of objects included in layout L) the reachability score $S0_{i^*}$ of the region i^* where the object is located. For simplicity, an object belongs to the region its center of mass falls into. Thus, for each object $k = 1,\dots,N_L^{objects}$ in each layout we obtain the *reachability score* $S0_k^L$:

$$S0_k^L = S0_{i^*} = \frac{N_{i^*}^{reached}}{N_{i^*}^{tot}} \in [0,1]. \tag{4.5}$$

4.2.3 Camera calibration within the layouts

Drawing a parallel to the reachability problem, GRASPA aims to assess the precision of the manipulator when reaching for poses acquired by the visual system in the camera reference frame. The main goal of this stage is to assess the calibration of the camera extrinsics, i.e. in this case the pose of the camera with respect to some robot link (robot-mounted camera) or the scene reference frame (world-mounted camera). In the former case, the calibration being checked is the pose of the camera with respect to the robot links (e.g. Figure 4.5a), while in the latter the camera is mounted outside of the robot arm links so that its pose with respect to the world reference frame does not change when the robot arm moves (e.g. Figure 4.5). Our benchmark defines a camera calibration score $S1_i$ for each i-th region introduced in the Section 4.2.2. In order to evaluate the scores $S1_i$, the robot is asked to reach a subset of the poses defined for the reachability evaluation (Fig. 4.4c) and the reached poses are compared to the targets. Similarly to the reachability stage, these target poses are defined with respect to the layout reference frame. This stage however differs from the reachability stage in the fact that the 6D pose reached by the end-effector have to be acquired through the visual system and transformed in the GRASPA board reference frame. A straightforward



(a) Robot-mounted camera (b) World-mounted camera

Figure 4.5 Difference between robot-mounted camera and world-mounted camera. In (a) ${}^rT_c$ denotes the transformation between root frame and camera frame, expressed in the root reference frame.

way for the user to detect the pose of the end effector is, for instance, by affixing a fiducial marker to it in a known position and orientation with respect to the kinematic chain.

The score $S1_i$ is then computed as:

$$S1_i = \frac{N_i^{reached}}{N_i^{tot}} \in [0,1], \tag{4.6}$$

where $N_i^{reached}$ is the number of poses in region i actually reached by the robot with a given accuracy and N_i^{tot} is the number of poses belonging to the region i.

A pose l is considered to be reached if the position and orientation errors $(E_l^p,\ E_l^o)$ (computed according to Eq. (4.2) - (4.4)) are smaller than respective thresholds (τ_p^c, τ_o^c) defined by the user. As mentioned, the only difference with respect to the scores $S0_i$ is that the poses actually reached by the robot are acquired through the robot vision system and not from the forward kinematics.

Also in this case, for each benchmark layout $L \in \{0,1,2\}$, we associate to each object $k = 1,\dots,N_L^{obj}$ the camera-calibration score $S1_{i^*}$ of the region i^* where the object is located. Thus, for each object $k = 1,\dots,N_L^{obj}$ we obtain the *camera calibration score* $S1_k^L$:

$$S1_k^L = S1_{i^*} = \frac{N_{i^*}^{reached}}{N_{i^*}^{tot}} \in [0,1]. \tag{4.7}$$

Since GRASPA layouts are defined with respect to the board reference frame, the benchmark protocol can be applied to grasping pipelines that do not process visual input (provided the user can reliably define a transform between the robot and the board reference frames). In such case, the benchmark does not take into account the scores $S1_k^L$.

4.2.4 Graspability

Different robots might have diverse grasping capabilities due to the arm maximum payload and the end-effector design and size. According to the fairness claims of GRASPA, grasping pipelines should not be benchmarked on objects that are impossible for the robot to manipulate. By doing otherwise, the grasping attempt would result in an automatic failure for one of or both the following reasons:

- the grasp planner cannot find a feasible candidate. This can happen when the gripper fingers cannot encompass any object feature, e.g. when the object is too large
- the object weight exceeds the robot payload. Even if a good grasp can be found, it is impossible for the robot to successfully grasp and lift the object

GRASPA encodes this information in the *graspability score* $S2_k^L = p_k^L \wedge g_k^L$, defined for each object $k = 1, \dots, N_L^{object}$ in layout L. p_k^L is 1 if the weight of the object is compatible with the robot payload and 0 otherwise. g_k^L is 1 if the end effector aperture is larger than the smaller dimension of the object and 0 otherwise. For simple objects such as a box, this dimension is the shorter edge of the enclosing 3D bounding box, while for complex objects (e.g. the power drill) this can be the diameter of the grip. Objects can also be declared un-graspable by other criteria, if sufficient motivation is given.

4.2.5 Grasp quality

This index evaluates grasps planned by the pipeline before execution, regardless of reachability. GRASPA uses a metric that relies on computation of the Grasp Wrench Space (GWS) and Object Wrench Space (OWS) [11]. This metric, while not being the most robust to uncertainty [48], is still widely used in many grasping toolboxes such as Simox, OpenRAVE and GraspIt! [112, 26, 72].

The user is required to provide the kinematic structure and the collision mesh model of their end-effector. Grasps have to be parametrized in terms of end effector pose and pregrasp configuration of the joints, making GRASPA compatible with both grippers and multifingered hands. Grasps are tested by first moving the end effector model to the desired pose with the desired pregrasp configuration, and then simulating the finger closure motion (in case of multifingered hands, joints are moved with equal velocity). When contact points are detected (via collisions between the object and end effector meshes), joints attached to the links that have collided are stopped. While this approach is straightforward for power grasps, pipelines that plan the contact locations need to be tested by setting the final hand configuration as a pregrasp.

We assume a *hard point contact with friction* model [77] with a fixed friction coefficient. Non-graspable objects (according to Subsection 4.2.4) do not receive any score. The *grasp quality* $\bar{S3}_k^L$ for each graspable object $k = 1, \ldots, N_L^{object}$ in layout L can be expressed as

$$\bar{S3}_k^L = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{\bar{r}(GWS_{k,t})}{r(OWS_k)} \right) \in [0,1], \tag{4.8}$$

where $\{\bar{r}(GWS_{k,t}), r(OWS_k)\}$ are the radii of the largest spheres contained, respectively:

- in the GWS defined by the t-th grasp planned for the k-th object. $\bar{r}(GWS_{t,k})$ is obtained by perturbing the grasping pose (before closing the fingers) in both position and orientation to ensure robustness, and then averaging the results;
- in the OWS of the k-th object, and is computed regardless of the grasp.

GRASPA v1.0 uses the implementation of the aforementioned metric included in Grasp-Studio [112].

4.2.6 Grasp execution and stability

GRASPA combines all the previously defined scores with grasp executions on physical robots. A *binary success score* $\bar{S4}_k^L$ for each object $k = 1, \ldots, N_L^{object}$ in layout L is evaluated over $T = 5$ grasp executions:

$$\bar{S4}_k^L = \frac{1}{T} \sum_{t=1}^{T} \left(S4_{k,t}^L \right), \tag{4.9}$$

where

$$S4_{k,t}^L = \begin{cases} 1, & \text{if object } k \text{ has been grasped at trial } t, \\ 0, & \text{otherwise.} \end{cases}$$

The object is considered grasped if it can be lifted by $\delta_p = 0.15$ m and held without falling for at least five seconds. Contact slip is acceptable as long as it does not ultimately cause the object to fall. The score $\bar{S4}_k^L$ can be evaluated by executing both grasping in isolation or in the cluttered scene, assuming the same modality is kept for each object and layout.

Finally, the benchmark evaluates the stability of the grasp during the execution of a fixed trajectory. This trajectory simply consists of rotations around the end effector approach axis and in the vertical plane such axis passes through. Given the grasping pose (p_{gr}, R_{gr}), with $p_{gr} \in \mathbb{R}^3$ as position and $R_{gr} \in SO(3)$ as the rotation matrix representing the orientation, the trajectory consists of 5 waypoints:

$$
\begin{aligned}
p_0 &= p_{gr} + \delta_p & R_0 &= R_{gr} && (4.10)\\
w_1 : p_1 &= p_0 & R_1 &= R_{gr} \cdot R^{+} && (4.11)\\
w_2 : p_2 &= p_0 & R_2 &= R_{gr} && (4.12)\\
w_3 : p_3 &= p_0 & R_3 &= R_{gr} \cdot R^{-} && (4.13)\\
w_4 : p_4 &= p_0 & R_4 &= R_{gr} && (4.14)\\
w_5 : p_5 &= p_0 & R_5 &= R_{gr} \cdot R^{\perp} && (4.15)
\end{aligned}
$$

where $R^{+/-}$ represents a rotation of ± 45 degrees around the approach axis of the end effector, and $R^{\perp}$ a rotation of 30 degrees (towards the table surface) in the vertical plane that contains this axis. The reference duration for each rotation trajectory is two seconds. We define the *grasp stability* score $\bar{S5}_k^L$ for each object $k = 1, \dots, N_L^{object}$ in layout L over T as:

$$
\bar{S5}_k^L = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{N_{w,t}^{reached}}{N_w^{tot}} \right) \in [0,1], \tag{4.16}
$$

where $N_{w,t}^{reached}$ is the number of the trajectory waypoints reached without dropping the object at trial t and $N_w^{tot} = 5$ is the total number of the trajectory waypoints. Again, contact slip is acceptable if it does not lead to a fall.

If the pipeline under test allows for it, GRASPA can measure its ability to grasp while avoiding other objects. We define the *obstacle avoidance* score $\bar{S6}_k^L$ for $k = 1, \dots, N_L^{obj}$ over T trials:

$$
\bar{S6}_k^L = \frac{1}{T} \sum_{t=1}^{T} \left(1 - \frac{N_{hit,t}}{N_L^{obj}} \right) \in [0,1], \tag{4.17}
$$

where $N_{hit,t}$ is the number of objects hit by the robot while approaching the target object at trial t. The score is 1 if the robot is able to avoid all the objects and 0 if it collides with every object. If no obstacle avoidance is accounted for, the objects must be placed on the layout and grasped one at a time and $S6$ is not computed. We refer to this case as *benchmarking in isolation* as opposed to *benchmarking in clutter*.

4.3 Reporting benchmark scores

In this Section, we explain how the single step metrics are combined into a single composite score. We outline how the benchmark scores are reported, giving guidelines on how to interpret the outcome and how to choose the required user-defined thresholds.

4.3.1 Final composite score and summary table

All the scores proposed thus far contribute to the computation of a composite score $\bar{S}_L$ to evaluate the grasping pipeline performance in each layout L, accounting for the limits of the testing platform. To this aim, the final score is computed considering only objects $m = 1, \ldots, M_L^{obj}$ such that:

- m is graspable by the robot, i.e. $S2_m^L = 1$
- m is in a reachable region, i.e. $S0_m^L > 0.5$. A region is not considered to be reachable if less than half the test poses were not reached with precision
- m is in a region with a good calibration of the vision system, where at least half the calibration poses were reached with acceptable precision, i.e. $S1_m^L > 0.5$.

The expression of the final score $\bar{S}_L$ is the following:

$$\bar{S}_L = \frac{1}{M_L^{obj}} \sum_{m=1}^{M_L^{obj}} \bar{S}_m^L, \tag{4.18}$$

where, if benchmarking with objects in isolation:

$$\bar{S}_m^L = \frac{1}{T} \sum_{t=1}^{T} (S3_{m,t}^L + S5_{m,t}^L) \cdot S4_{m,t}^L \in [0,2], \tag{4.19}$$

whereas, if benchmarking in clutter:

$$\bar{S}_m^L = \frac{1}{T} \sum_{t=1}^{T} (S3_{m,t}^L + S5_{m,t}^L + S6_{m,t}^L) \cdot S4_{m,t}^L \in [0,3]. \tag{4.20}$$

where L indicates the layout, m indicates the object and t indicates the trial, $S3_{m,t}^L$ is the *grasp quality* score, $S5_{m,t}^L$ is the *grasp stability* score, $S6_{m,t}^L$ is the *obstacle avoidance* score (if the pipeline allows for it), and $S4_{m,t}^L = 1$ only if the object has been successfully grasped at trial t. The scores computed by the benchmark are summarized in Table 4.1.

Score formula	Score nomenclature	Brief description
$S0_k^L = S0_{i^*} = \frac{N_{i^*}^{reached}}{N_{i^*}^{tot}} \in [0,1]$	Reachability score	Accounts for whether the object is located in a region characterized by a good reachability of the robot
$S1_k^L = S1_{i^*} = \frac{N_{i^*}^{reached}}{N_{i^*}^{tot}} \in [0,1]$	Camera calibration score	Accounts for whether the object is located in a region characterized by a good calibration of the vision system
$S2_k^L = p_k^L \wedge g_k^L \in \{0,1\}$	Graspability score	Accounts for whether the object can be physically grasped and lifted by the robot, considering its shape and weight.
$\bar{S3}_k^L = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{\bar{r}(GWS_{k,t})}{r(OWS_k)}\right) \in [0,1]$	Grasp quality score	Accounts for how contacts are distributed on the object by simulating grasp closure in simulation and computing the grasp wrench space.
$\bar{S4}_k^L = \frac{1}{T}\sum_{t=1}^{T}\left(S4_{k,t}^L\right) \in [0,1]$	Binary success score	Accounts for whether the robot actually managed to grasp the object in real tests.
$\bar{S5}_k^L = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{N_{w,t}^{reached}}{N_w^{tot}}\right) \in [0,1]$	Grasp stability score	Evaluates the stability of the grasp during the execution of a fixed trajectory.
$\bar{S6}_k^L = \frac{1}{T}\sum_{t=1}^{T}\left(1 - \frac{N_{hit,t}}{N_L^{obj}}\right) \in [0,1]$	Obstacle avoidance score	*(Only in cluttered mode)* Accounts for how many objects the robot has hit while executing the grasp.
$\bar{S}_L = \frac{1}{M_L^{obj}}\sum_{m=1}^{M_L^{obj}} \bar{S}_m^L$ where in isolation: $\begin{cases}\bar{S}_m^L = \frac{1}{T}\sum_{t=1}^{T}(S3_{m,t}^L + S5_{m,t}^L) \\ \quad S4_{m,t}^L \in [0,2]\end{cases}$ in the clutter: $\begin{cases}\bar{S}_m^L = \frac{1}{T}\sum_{t=1}^{T}(S3_{m,t}^L + S5_{m,t}^L + S6_{m,t}^L) \\ \quad S4_{m,t}^L \in [0,3]\end{cases}$	Compound per object score	Combines all the scores in order to evaluate the grasping pipeline performance taking into account any limitation of the robotic platform used in real world tests.

Table 4.1 Summary of the benchmark scores.

The final output of the benchmark consists of a summary table (see Table 4.4 for an example). In the second column, the value of the final score S_L for each layout $L = 0, 1, 2$ is reported. In the rest of the table, each row collects all the scores computed for each object $k = 1, \ldots, N_{obj,L}$. Analyzing such scores can give insight about the performance of different parts of the grasping pipeline, down to the hardware. For instance, if the *grasp quality score* $\bar{S3}_k^L$ is high but the robot could not grasp the object ($\bar{S4}_k^L = 0$), the *reachability score* $S0_k^L$ and the *camera-calibration* score $S1_k^L$ can outline whether the vision system calibration or the robot reachability are to blame for the failure in the execution of the grasp. On the other hand, if $\bar{S3}_k^L$ is low, but $\bar{S4}_k^L$ and $S5_k^L$ are large, this may indicate that the physical execution is able to compensate for the poor grasp quality (e.g. the gripper is compliant and can conform to the object, or the object pose changes during the grasp execution).

4.3.2 Defining reachability and camera calibration thresholds

As previously mentioned, GRASPA requires position and orientation thresholds used during the reaching test ((τ_p^r, τ_o^r), see Paragraph 4.2.2) and the camera calibration test ((τ_p^c, τ_o^c), see Paragraph 4.2.3). Since GRASPA is meant to adapt to different robot platforms, these thresholds cannot be fixed a priori by the benchmark and have to be chosen by the user according to the robot platform and vision system. The pair (τ_p^r, τ_o^r) defines how precise the robot kinematics is over the GRASPA layout space. For dexterous and precise arms (e.g. industrial manipulators), small values of the reachability thresholds are advisable (e.g. $\tau_p^r = 0.005$ m, $\tau_o^r = 0.1$ rad). For less precise robots (e.g. research-oriented platforms such as iCub, PR2, Baxter) higher values are needed (e.g. $\tau_p^r = 0.02$ m, $\tau_o^r = 0.5$ rad). On the other hand, (τ_p^c, τ_o^c) depend on camera resolution and the method used to visually infer the end effector pose. Upper bounds on these parameters are $\tau_p^r = 0.05$ m, $\tau_o^r = 0.5$ rad, that we found borderline acceptable for low-resolution cameras such as the ones used on the iCub humanoid robot (320x240 pixels).

Note that the aforementioned thresholds are mostly useful in the presence of hardware limits, inverse kinematics solver or calibration. In this scenario, low thresholds will likely mark some regions as unreachable or not well calibrated and will allow only grasps in regions where their execution can be more precise. With high thresholds, grasps will be executed and scored in regions where lack of precision might lead to unstable grasps and unfair scoring.

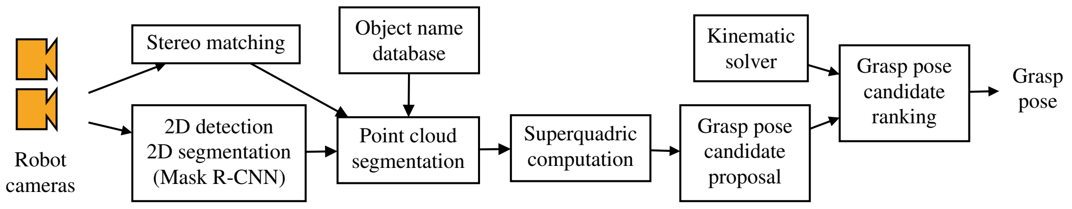


Figure 4.6 The cardinal point grasp detection pipeline as deployed on the iCub platform.

4.4 Deployment of the benchmark - iCub humanoid robot

In the upcoming two sections we give an overview of the experimental setups for the deployment of GRASPA on real robots. We used the iCub robot platform to benchmark a grasping pipeline based on the Cardinal Point Grasp approach outlined in Chapter 2 and in Nguyen et al. [78]. Since we were mainly interested in the performance of the methods in the simplest case, we benchmarked the pipelines on isolated GRASPA objects.

4.4.1 Pipeline description - Cardinal Point Grasps

What follows is a brief overview of the structure of the grasping pipeline set up on the iCub humanoid robot. Since it replicates the approach proposed in Section 2.2.3, it closely resembles the pipeline outlined in 2.3. A functional block diagram of the pipeline used in this instance is shown in Figure 4.6, and we hereby give some detail to better document the experimental setup.

Stereo matching. One of the features of the iCub robot is a stereo camera rig built in the robot head, equipped with a 320x240 resolution color camera in each eye. This setup consists of 3 actuated DoF in the robot neck to grant roll, pitch and yaw capabilities to the head, and 3 actuated DoF in order to model the human oculomotor system (tilt, version and vergence). More details about this system are provided in Section 2.3.

2D-driven point cloud segmentation. Due to technical limitations of the aforementioned setup, effectively and reliably segmenting the object from the tabletop surface in the point cloud domain proved to be infeasible. Hence, we overcame this limitation by performing 2D object segmentation and using this information to parse the depth map. While using GRASPA, however, objects can visually occlude each other and the 2D segmentation method used for the experimental validation of the Cardinal Point Grasp planner (described in Section 2.3) cannot be used in this instance. To overcome this limitation, we adapt an off-the-shelf

Tensorflow implementation [1] of Mask R-CNN [45] in order to process monocular images and obtain 2D segmentation masks of the objects. We use a ResNet-50 backbone pre-trained on MS Coco, further training it on a subset of YCB-Video [128] and then fine-tuning it on a custom synthetic dataset. The latter was obtained by augmenting real images with YCB object crops following the Cut, Paste and Learn approach [30] enhanced with segmentation masks. The dataset features the 16 YCB objects used in GRASPA as classes, and ArUco marker crops as distractors. Further details on how we generate custom synthetic datasets can be found in 3.2.2.

Superquadric computation. As described in [78] and Section 2.2.3 we approximate the object with the smallest superquadric fitting the point cloud. The superquadric and its 6D pose are estimated by solving a constrained optimization problem, imposing one of the axes of the superquadric to be perpendicular to the table surface.

Grasp pose candidate proposal and ranking We generate grasping pose candidates from the cardinal points of the superquadric (i.e. where axes intersect the surface). The candidates are then ranked according to the superquadric and hand size, and the capability of the robot to reach them with sufficient accuracy. Further details can be found in 2.2.3 and Nguyen et al. [78].

Grasp execution The grasp execution was split in two concatenated trajectories. The first trajectory consists in an approach motion, i.e. the robot reaches for a pose identical to the target pose (p_{gr}, R_{gr}) translated along the grasp approach axis, while the second trajectory brings the robot hand to the the target pose itself and proceeds to close around the object. The closure is performed by moving all the finger joints from the pregrasp configuration, i.e. stretched fingers and opposed thumb, with the same angular velocity until a strain threshold is reached.

4.4.2 Data collection

We hereby outline how we deployed the protocol to collect the benchmark data from the real robot. The explanation is split in subsections, each pertaining to a single GRASPA indicator. The code necessary to these experiments has been collected, together with experiment data, in a public repository[6].

[6]https://github.com/robotology-playground/GRASPA-test

Reachability score $S0$. Data for the computation of the reachability scores $S0_i$ has been acquired by having iCub reach for the poses defined within the benchmark with its right hand, querying the forward kinematics to obtain the poses that were actually reached. We used OpenCV in order to estimate the pose of the layout marker boards (Fig. 4.3) with respect to the robot. We used this information to express the target poses in the robot reference frame and save the reached poses in the layout reference frame. Fig. 4.7 shows some samples of the outcome.

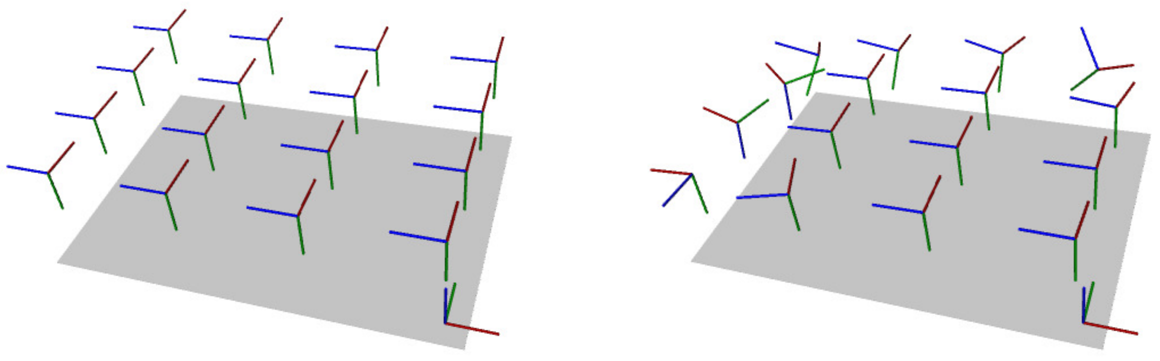

(a) Desired poses (Reachability pose set 1) (b) Reached poses (Reachability pose set 1)

Figure 4.7 Reachability test results: comparison between the objective poses and those actually reached by iCub.

Camera-calibration score $S1$. Data for the computation of the camera calibration score $S0$ was obtained by following a similar procedure. As imposed by the benchmarking protocol, the reached poses have to be acquired through visual means instead of simply querying the forward kinematics. Since the whole grasping pipeline relies on the stereo camera rig, the camera calibration score basically tests the camera extrinsics with respect to the robot root reference frame. This is critical for humanoid robots such as iCub, since the camera rig resides at the end of a kinematic chain that can be quite articulated. In the iCub case, the kinematic chain between the eyes and the root reference frame consists of 9 DoF (3 for the torso, 3 for the neck, 3 for the eyes). In order to visually detect the pose of the hand, we resorted to affixing two ArUco markers located on the back and the side of the hand (Figure 4.8).

Graspability $S2$. We considered an object to be graspable by the iCub if at least one of its dimensions was smaller than the iCub hand aperture and its weight was compatible with the

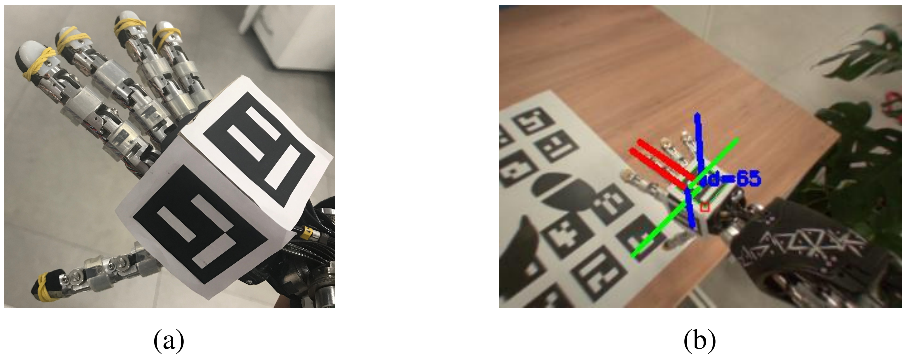


Figure 4.8 Detection of the iCub TCP using fiducial markers. In (a), ArUco fiducial marker placement on the iCub hand. In (b) visual estimation of the iCub TCP (blue axis pointing down) through detection of the marker pose (blue axis pointing up).

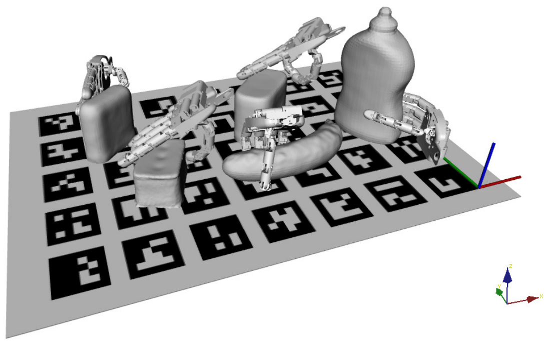


Figure 4.9 Rendering of the grasp poses planned for layout 0 with the tested algorithm on the iCub humanoid. For visual clarity, only one pose is rendered for each object.

maximum arm payload (0.5 kg). We considered un-graspable by iCub objects that have a very low profile (i.e. scissors, clamp) when laid flat on the table.

Grasp Quality $S3$**.** For each object visible in each layout, we planned for $T = 5$ 6D grasping poses according to Section 4.4.1, expressed in the layout reference frame by using the estimated pose of the ArUco marker board. During the grasp quality computation using GWS and OWS, the iCub hand used in the GraspStudio [112] suite has been modeled from the official CAD in order to make contact detection as similar as possible to the real case. A graphical rendering of some of the planned poses can be seen in Figure 4.9.

Binary Success and Stability scores $S4, S5$**.** We executed in isolation the $T = 5$ grasps computed by the algorithm for each object. Whenever the robot managed to grasp the object we also had it execute the trajectory defined in Section 4.2.6. We added a layer of rubber on the robot fingertip to actually have friction on the contact points.

4.5 Deployment of the benchmark - Franka Emika Robot Arm

Explicitly modeling graspable objects as superquadrics turned out to be quite reliable on iCub, since the mechanical complexity and adaptability of its underactuated hand allow power grasps to be advantageous. However, other state of the art grasp planners are designed to perform precision picking tasks using parallel jaw grippers and require precise depth sensing cameras. The reason we decided to deploy GRASPA on a different robot setup is twofold: on the one hand, it allowed us to avoid designing and installing major upgrades on the robot hardware; on the other hand, it allowed us to prove the claim that the GRASPA benchmark is adaptable to different robot setups and work cells. Additionally, since the Panda arm comes by default with a parallel-jaw gripper, this also enabled us to evaluate and benchmark grasping pipelines designed for this class of end effector natively, without having to adapt them to a humanoid hand.

We used a 7 DoF robot arm to benchmark a simple pipeline based on two state of the art grasp planners, namely GPD [82] and Dex-Net [62]. These were chosen for the recognition and impact on the grasping community, relevance to the proposed grasping task and code availability. Similarly to the benchmarking on iCub, we considered the simplest scenario and benchmarked the pipelines on isolated objects.

GPD. This approach, proposed by Pas et al., detects 6 DoF grasps on point clouds, assuming a parallel jaw gripper. It works by uniformly sampling the point cloud (that can be either partial or complete) and creating a candidate in each sampled location by setting the approach direction to the surface normal. The grasp candidates are filtered out using a list of geometric criteria such as approach direction, gripper size, maximum aperture and desired workspace. Eventually, the quality of each candidate is estimated by considering the cloud points that fall into the grasped volume of the gripper and projecting them in different ways to obtain a tensor that is fed into a simple CNN. The output of the CNN is the grasp quality, and it is trained on a number of synthetic grasps. For more details, refer to the paper [82]. An example of a GPD-generated grasp can be found in 4.16b.

Dex-Net. Proposed by Mahler et al., Dex-Net is a planar grasp detector that consumes range images (in the form of depth maps) in order to perform bin-picking tasks in cluttered scenes. It requires a setup where the camera is mounted on the top of the scene, with its image plane parallel to the scene background (e.g. table or bin floor), and produces 4 DoF planar grasps (i.e. the approach axis is orthogonal to the horizontal scene plane) that are parametrized with 3D position and angle around the approach axis. Dex-Net has seen different revisions and the addition of features in recent years (e.g. vacuum instead of parallel jaw gripper structure, or a mix of the two with different policies) but at its core it is an end-to-end grasping pipeline trained on a synthethic grasp database (also proposed by the same authors [64]) that uses CNNs to extract features from depth maps and propose a number of grasp candidates with associated estimated quality. An example of a DexNet-generated grasp can be found in 4.16a.

4.5.1 Pipeline description - Dex-Net and GPD grasp planners

The experimental setup consists of a Franka Emika Panda using the stock Franka Hand as a gripper. An Intel Realsense D415 was attached to the end effector in order to acquire RGBD and point cloud from the scene (Figure 4.1b). The stock Franka Hand fingertips are made of a stiff rubber material and require quite a large clamping force in order to generate friction. While this does not constitute a problem for some YCB objects used in GRASPA (e.g. hammer), repeated grasps would have damaged some of the more fragile objects (e.g. cardboard boxes). We applied some soft rubber tape on the fingertips in order to limit the amount of force applied during a grasp to the minimum the Franka Hand can sense (about 10 N) and still generate friction on the contacts.

Since the experimental setup would be the same for different grasp planners, we designed a lightweight software framework aimed at wrapping different implementations of grasp planners and reusing as many software modules of the same pipeline. The main components of the frameworks are the following:

- a Python package that provides base classes and helper functions for camera data and grasp poses. It also defines a common interface for any grasp planner that deals with such data. This way, any grasp planner that consumes RGB images, depth maps or scene point clouds can inherit the same interface class and implement a number of common methods (e.g. planning for grasps given camera data, or displaying the planned grasps). This package has very little dependencies and can wrap planners written in Python (natively) and other languages (through compilable bindings)

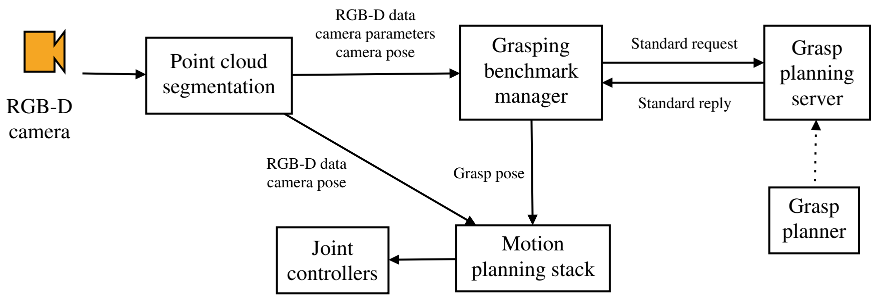


Figure 4.10 The grasp planner benchmarking pipeline as deployed on the Franka Emika Panda setup.

- a ROS-based **grasp planning server** that uses the Python common interface to expose the grasp planning functionality to other nodes through a standard request (containing visual data and camera parameters) and response (6D grasp candidates)
- a ROS-based **grasping benchmark manager** that receives 3D visual data from the camera and ensures coherent communication between the grasp planner and the motion planning stack.

At the time of writing, the grasp planners whose functionality has been implemented in the benchmarking framework are Dex-Net [62], GPD [82] and the 6D superquadric-based grasp planner proposed by Vezzani et al. [120].

The rest of the pipeline (see Figure 4.10 for a functional representation of the different blocks) features a **point cloud segmentation** stage and a **motion planning stack**.

Point cloud segmentation. This simple ROS node consumes point clouds coming from the camera and filters it in order to remove the table plane, restricting the perceived workspace whatever object is on top of the GRASPA board in front of the robot. Tabletop plane removal is performed by fitting a plane to the point cloud through a random sample consensus (RANSAC) algorithm.

Motion planning stack. This ROS node receives input from the user, the point cloud segmentation module and the grasping benchmark manager. It is built on top of the MoveIt! framework and its function is to compute a feasible trajectory for all the joints, given any target for the end effector in the 6D space while avoiding collision with itself and the workbench. This node also detects the pose of the GRASPA marker board with respect to the robot root reference frame.

4.5.2 Data collection

Similarly to Section 4.4.2, we now document the experimental procedure carried out to collect the benchmark data from the real robot setup, according to the GRASPA protocol. Again, we split the explanation in subsections, each pertaining to a single GRASPA indicator. The full pipeline code, although split in different repositories to ease reusability, is available in full at our GitHub organization[7]. The data gathered from these experiments is also available online[8].

Reachability score $S0$. The robot setup was placed in front of the GRASPA board as shown in Figure 4.11a, positioning the end effector so that the gripper-mounted camera could detect the board pose ${}^{r}T_{b}$ with respect to the robot reference frame. At that point the robot can move its end effector towards the reachability targets and record the reached 6D poses using only the direct kinematics.

Camera-calibration score $S1$. Differently with respect to the iCub experimental setup, the camera used to acquire RGB-D data for grasp planning cannot detect the end effector if a marker were affixed to it. In order to visually estimate the position of the end effector, we set up a second Intel Realsense, identical to the hand-mounted one, in front of the setup (Figure 4.11a). Using the hand-mounted camera we can detect the pose of the GRASPA marker board with respect to the world root reference frame (${}^{r}T_{b} \in \mathrm{SE}(3)$). We can decompose this as

$$ {}^{r}T_{b} = {}^{r}T_{h}\,{}^{h}T_{c_1}\,{}^{c_1}T_{b} \qquad {}^{r}T_{b} \in \mathrm{SE}(3) \tag{4.21} $$

where ${}^{r}T_{h} \in \mathrm{SE}(3)$ denotes the pose of the hand tool center point (TCP) with respect to the root reference frame, ${}^{h}T_{c_1} \in \mathrm{SE}(3)$ denotes the pose of the hand-mounted camera c_1 with respect to the TCP and ${}^{c_1}T_{b} \in \mathrm{SE}(3)$ denotes the board pose in the camera reference frame. Since ${}^{r}T_{h}$ and ${}^{c_1}T_{b}$ do not depend upon camera c_1 extrinsics, we can use an external camera c_2 to evaluate the accuracy of the hand-camera calibration ${}^{h}T_{c_1}$. By estimating the GRASPA board pose in the c_2 reference frame (${}^{c_2}T_{b} \in \mathrm{SE}(3)$) we obtain the pose of camera c_2 in the root reference frame

$$ {}^{r}T_{c_2} = {}^{r}T_{b}\,{}^{b}T_{c_2} = {}^{r}T_{b}\,{}^{c_2}T_{b}^{-1} \qquad {}^{r}T_{c_2} \in \mathrm{SE}(3) \tag{4.22} $$

[7]https://github.com/hsp-panda

[8]https://github.com/robotology-playground/GRASPA-test/tree/master/experiment_data

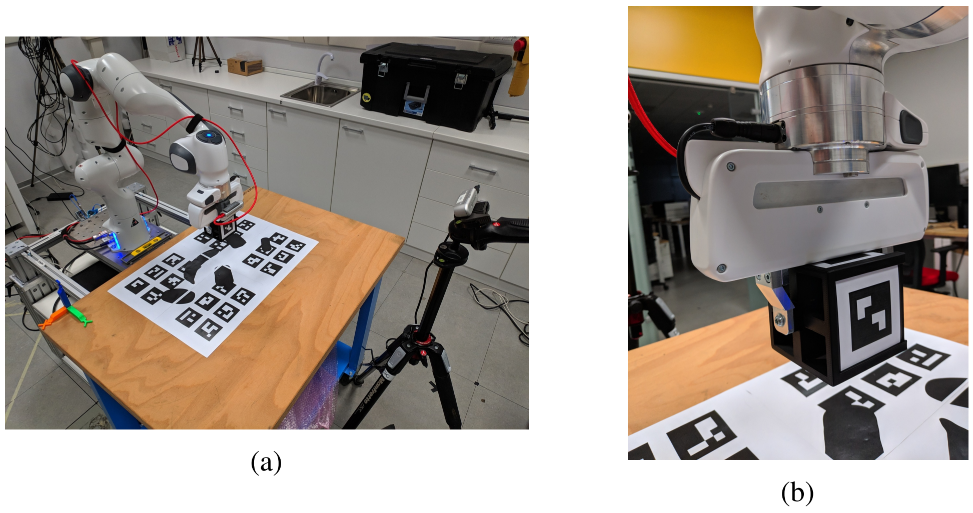

(a) (b)

Figure 4.11 The Panda arm set up for the computation of metrics $S0$ and $S1$. An additonal camera was mounted in front of the robot (a) to visually estimate the end effector pose (b).

and, by detecting the pose of the TCP in the c_2 reference frame (${}^{c_2}T_h \in \mathrm{SE}(3)$) we can obtain the pose of the TCP in the GRASPA board reference frame

$$ {}^bT_h = {}^bT_{c_2}\,{}^{c_2}T_h \qquad {}^bT_h \in \mathrm{SE}(3) \tag{4.23} $$

which is the required input for metric $S1$. In order to estimate ${}^{c_2}T_h$ we augmented the marker board pose estimation node to also detect single markers and we 3D-printed a custom structure to rigidly attach said markers to the robot hand in a known configuration, as seen in Figure 4.11b.

Graspability $S2$. The Panda arm features a payload several times larger than the iCub arm, therefore the only objects that were deemed ungraspable are the ones whose smaller dimension is larger than 0.08m, the maximum aperture of the Franka Hand gripper.

Grasp Quality $S3$. In order for the $S3$ metric to be computed, a description of the Franka Hand must be specified in the Simox framework. Just as other grasp analysis tools (e.g. GraspIt! [72]), the Simox toolbox requires the complete kinematic structure of the end effector, complete with collision meshes in order to detect contacts during closure. Due to the inner workings of the collision detection engine employed by the toolbox, only one contact point will be generated for each pair of colliding meshes. The Franka Hand can be therefore simply modeled with three meshes (hand main body and two fingers) and when the gripper closes around an object two contact points will be found in total. However, this results in null volume for the Grasp Wrench Space (GWS) due to the contact model, the grasp shape and the low number of contacts. In the *point contact with friction* model used by Simox (refer to Figure 4.2), the wrench w_{c_i} applied at the i-th contact point has a null component around the contact normal $\overline{n_i}$. This does not constitute a problem for multi-finger grasps since contact normals are typically non-collinear, but in the case of a parallel jaw grasp there are two contact points and their contact normals are collinear. This hinders the capability of the grasp to resist any perturbation (in wrench space) acting on the object around said normals, effectively flattening the GWS along that dimension. One possible solution is to use a more sophisticated contact model, e.g. the *soft finger model* [77]. This allows each contact to exert a force along the normal (point contact with friction model) as well as a torque around it, and allows the grasp to resist wrenches around $\overline{n_i}$ even if the contact normals are collinear. Unfortunately, the Simox toolbox does not support this model and extending the software framework was beyond the scope of this work. Another solution is to split the fingertip collision volume, effectively increasing the number of contacts.

Choosing the number of contacts. In order to model the behaviour of the contact surfaces, we would ideally need to split the contact surface of the fingertips in an infinite number of infinitesimal contact surfaces[9]. Since this is computationally intractable, we sought to approximate this with a finite number of contact surfaces.

We divided the contact surface of the fingertips (Figure 4.13a) in an increasing number of equally spaced contact volumes so that, during closure, Simox may compute the GWS over a larger number of contact points. We computed grasp quality through Simox on the same grasp, varying the number of subdivisions of the contact surface. As shown in Figure 4.12, the grasp quality converges for a large number of contacts. We can observe that the point contact with friction model used by Simox causes the grasp quality metric $S3$ to directly depend on the maximum distance between the contact normals and only indirectly on the number of collision volumes used; the number of the collision volumes is inversely proportional to their size (because of the fixed size of the fingertip), and since in this case each collision volume can generate a contact, there will be contact normals closer and closer to the fingertip edge (if the grasp allows for it). We show this by placing 4 collision volumes of edge 1mm at the vertices of the fingertips contact surface. (Figure 4.13b). For a given grasp (that allows the fingertips to fully rest on the obect), grasp quality metric $S3$ is equal to the case in which the fingertip surface is divided in 100 collision volumes with edge size 1 mm ($S3 \sim 0.15$ in both

[9]It is important to notice that this was not necessary while modeling the iCub hand, as the power grasps planned for it ensure there are always at least more than two contacts with the object, and the contacts being far apart (with contact normals that are rarely parallel) avoids the collapse of the GWS volume.

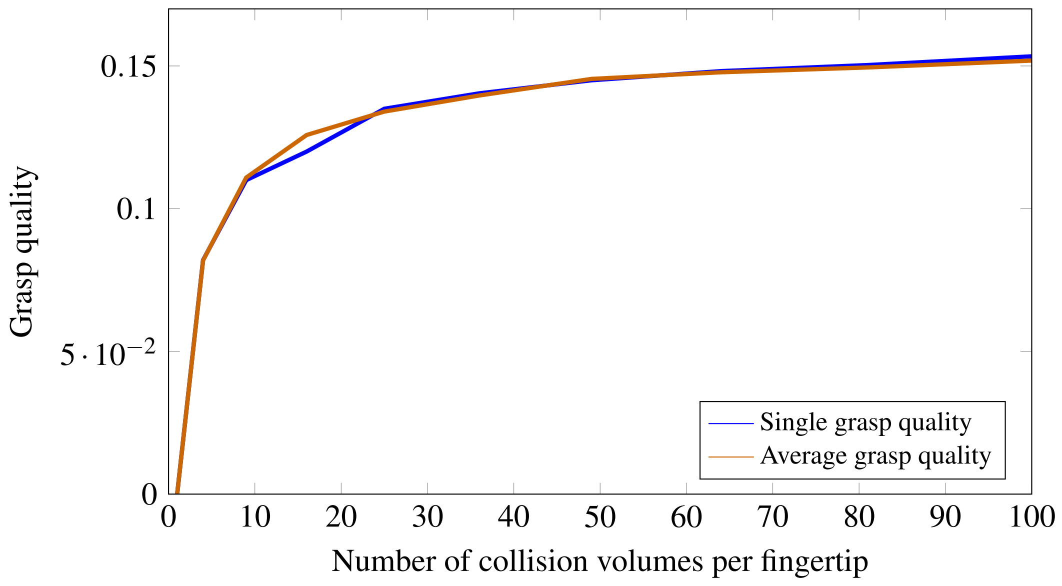
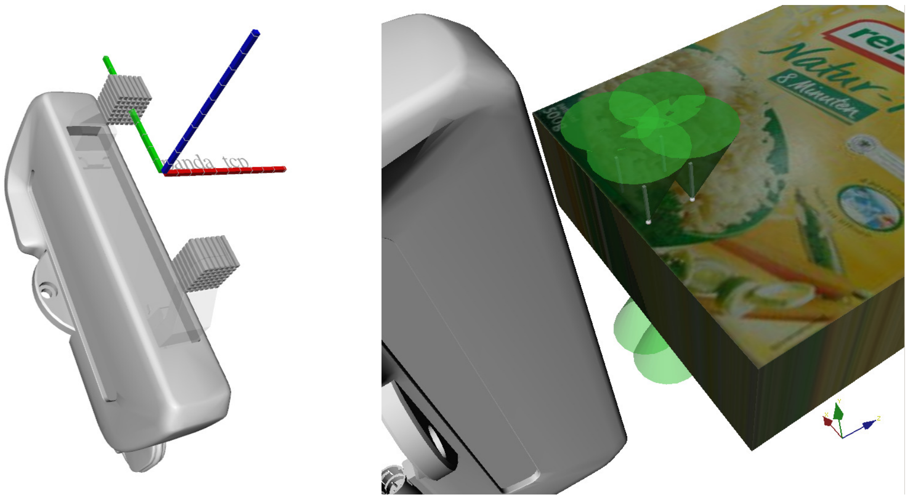


Figure 4.12 Modeling of the collision volumes of the Franka Hand fingertips. For each subdivision of the fingertips, grasp quality has been measured on the test grasp and by perturbing it 50 times. Perturbations are drawn from uniform distributions: $\Delta p \in [0, 1]$cm is the Euclidean norm and orientation $\Delta p \in [0, 10]$deg is the sum of three euler angles.

cases). However, using 4 small, maximally spaced contact surfaces on each fingertip is not feasible, as not all grasps cause the fingertips to fully contact the object. This, in turn, may cause missed contacts and this makes the metric unreliable and subject to large swings when the pose is perturbed during computation of $\bar{S3}^L_k$.

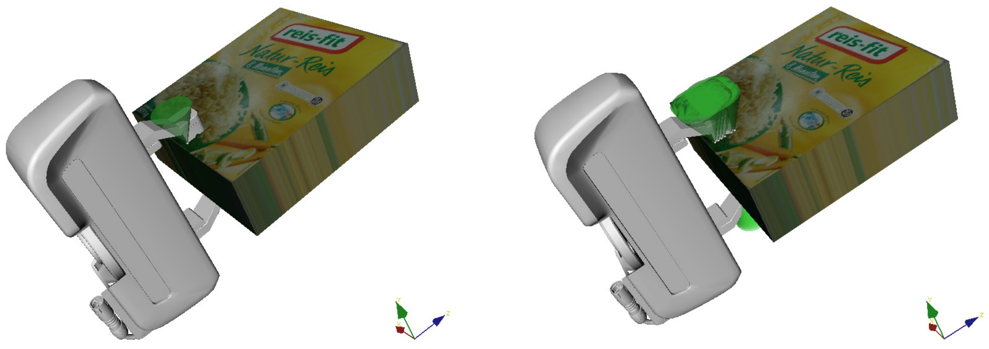
To recap, in the case of a parallel jaw gripper and a point contact with friction model, the GWS volume is limited by the maximum distance between contact normals. Hence, since it is not feasible to use a small number of maximally spaced collision volumes, we need to model the fingertip surface with a large number of different collision volumes, that diverges to approach the ideal case. However, using a very large number (e.g. 100) of contacts for each fingertip exponentially increases the time needed for computing collisions. As a tradeoff between the two extremes, we chose to model the each fingertip with 49 contact volumes (7x7 grid), covering the whole fingertip pad surface. As shown in Figure 4.12, using 49 contacts instead of 100 leads to a 5% loss in grasp quality, which we believe is a good enough approximation for this use case.

Binary Success and Stability scores $S4, S5$. Following the benchmarking protocol defined in Section 4.2.6, we executed 5 grasps for each object, in isolation, and tested the grasps by lifting the object and moving it through the protocol stability trajectories. While closing



(a) (b)

Figure 4.13 Subdivision of the Franka Hand collision volumes in order to study grasp quality variations. In (a), an example subdivision of each fingertip in a 7x7 grid of collision volumes. Finger meshes are in transparency. The TCP reference frame is highlighted (approach axis in blue). In (b), the "edge case" where collision volumes are very small and on the fingertip edges. The green shapes represent the computed friction cones.



(a) Panda hand with 2 contacts (b) Panda hand with 49 contacts

Figure 4.14 The Franka Hand as modeled in Simox, grasping an example object. The green shapes represent the computed friction cones. In (a) each fingertip has been modeled as a single collision volume. The two contact normals are almost collinear and the volume of the GWS (V_{GWS}) is $\sim 10^{-9}$. In (b) each fingertip has been split in 49 (7x7 grid) collision volumes, for a total of 98 contact points. In this case, the grasp can resist an external torque around the contact normal direction ($V_{GWS} = 0.152$). The green shapes represent the computed friction cones.

the gripper fingers, we thresholded the squeezing force to 10 N in order not to damage the objects while simultaneously applying enough normal force to cause surface friction.

4.6 Results and Discussion

In this Section we aim to present and discuss the results of the benchmarking done on the iCub and Panda robots. Since one of the features of GRASPA is to allow vastly different grasping pipelines, tested on different robot setups, to be evaluated fairly, we compare the outcome of the experiments in order to highlight the pros and cons of using the proposed benchmark instead of others.

4.6.1 On GRASPA thresholds for each setup

Table 4.3 collects the user-defined parameters used for data acquisition and score computation. Although the GRASPA protocol gives users some guidelines (refer to Section 4.2.3) regarding how to select the threshold values $(\tau_p^r, \tau_o^r, \tau_p^c, \tau_o^c)$, ultimately it is up to the user themselves to choose appropriate values for the platform under evaluation. For the deployment on iCub, for instance, the τ thresholds have been chosen conservatively considering the limited task space and the visual calibration limits of the robot. The pipeline under test plans for power grasps (i.e. it computes a pose for the hand palm and not for each fingertip) and is, therefore, robust with respect to reachability errors in position of $\tau_p^r = 0.02$ m and in orientation of $\tau_o^r = 0.5$ rad. Due to the complexity of the iCub oculomotor system, we chose looser tolerances for the GRASPA relative thresholds ($\tau_p^c = 0.045$ and $\tau_o^c = 0.8$) in order to account for some error in the calibration of the vision system. For the deployment on the Panda robot, on the other hand, all the thresholds have been lowered by about 75% to account for the higher precision the arm is capable of.

4.6.2 GRASPA computed scores

Computed scores are reported in Tables 4.4 to 4.6. Before moving on, we highlight:

- the values of the reachability score $S0_k^L$ and the camera calibration score $S1_k^L$ when $S0_k^L < 0.5$ and $S1_k^L < 0.5$. In these cases, the object is in a region unreachable by the robot or with an unacceptable visual calibration error

- the value of the graspability score when the object is not graspable, i.e. $\bar{S2}_k^L = 0$. In these cases, the final score $\bar{S}_k^L$ is not computed by the benchmark and is replaced with the placeholder N/A
- because of the proximity of some objects to the robot torso, the stereo vision could not reliably acquire partial point clouds. In these cases, no further score is reported.

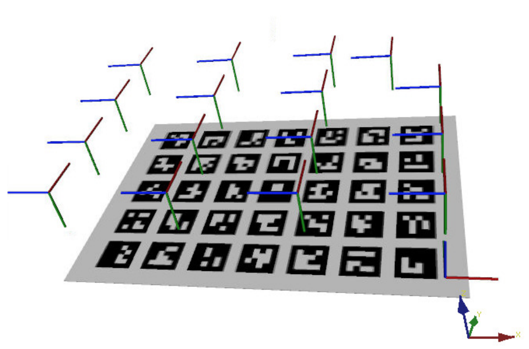
As stated in the benchmark description, the granularity of the GRASPA metrics allows for an in-depth look into the performance of the experimental pipeline. In the following paragraphs we will analyze the indicators, giving an example of how these can be used to infer useful information about success and failure cases.

Analysis of the GRASPA scores on the iCub setup. Table 4.4 shows how our benchmark fairly tests the capabilities of the superquadric cardinal point grasping pipeline, without penalizing its performance wherever the test platform proved its limits. A meaningful example is the *foam brick* in layout 0. The grasp quality score $\bar{S3}_k^L$ is good, meaning that the algorithm computes proper grasping poses for the object. However, in practice, the robot could grasp the object only once over the 5 trials ($\bar{S4}_k^L = 0.2$). The reason of such failure can be attributed to the poor vision system calibration in the region of the object ($S1_k^L = 0.25$). For the considerations explained in Section 4.3.1, the foam brick scores do not contribute to the computation of the final composite score. On the other hand, the grasps computed for other objects (e.g. *potted meat can*, *cracker box* and *tennis ball*) show a poor analytical grasp quality (low $\bar{S3}_k^L$) in layouts 0 and 1, but the respective values for $\bar{S4}_k^L$ and $\bar{S5}_k^L$ show good performance in practical graspa. This indicates that the execution of these grasps by the real robot ended up being more successful than anticipated by the analytical grasp quality metric. Since reachability and visual calibration metrics ($S1_k^L$ and $S0_k^L$) are maxed out, it cannot be blamed non-ideal behaviour of the system while executing the approach trajectory. By looking at experimental video footage, we observed that in these cases the mechanical underactuation in the iCub hand allows the fingers to conform to the object. This behaviour is not modeled in the Simox toolbox used to compute $\bar{S3}_k^L$, and this would explain the discrepancy in performance between the simulated closure and the actual closure.

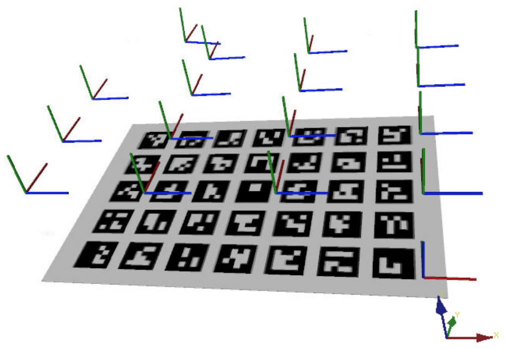
Analysis of the GRASPA scores on the Franka Emika Panda setup. Tables 4.5 and 4.6 list the GRASPA indicators computed on the Panda setup for the GPD and Dex-Net based pipelines. As highlighted by the reachability metric $S0_k^L$, the robot can easily reach the objects over the GRASPA board in a variety of orientations. In fact, despite having the same number of the DoF with respect to the iCub arm:

- the links of the Franka arm are almost twice as long
- the robot inverse kinematic is more precise and repeatable. In iCub, the torso, shoulder and elbow joints are tendon-driven, introducing small non-idealities that are difficult to compensate completely. This is not the case for the Franka arm, whose joint are directly driven through a gearbox
- the joint ranges are larger for the Franka arm.

This reflects in an almost perfect reachability score (compare Figure 4.15 with Figure 4.7). Due to the cameras and extrinsics calibration used, values of $S1_k^L$ show that camera calibration is very reliable. The value range of $S3_k^L$ is consistent between Dex-Net and GPD, but it is on average smaller with respect to the same metric in the iCub experiments. This does not come as a surprise, as the volume of the GWS is generally larger when a grasp is composed by contacts that are further apart (as outlined in Section 4.5.2) from each other. On average, while $S4_k^L$ and $S5_k^L$ are both higher with respect to the pipeline tested on iCub, when comparing the performance of GPD against Dex-Net the former seems to have an advantage. Dex-Net is a top-down grasp planner, effectively providing candidates with 4 degrees of freedom in cartesian space (the approach axis is always facing the horizontal surface the objects are supposed to lie on), while GPD outputs 6D grasp poses. This is especially advantageous for objects that are not easily graspable from the top (e.g. the *mustard bottle*, whose cap is thin and slippery) but are very easily graspable from the side. Figure 4.16 shows the difference in approach between the two planners.



(a) Reached poses for reachability set 0



(b) Reached poses for reachability set 2

Figure 4.15 View of the poses reached by the Panda arm during reachability tests. Target poses are those shown in 4.4b and 4.4d. The Panda arm has excellent reachability over the board, with only two poses being unreachable (top right for (a) and top left for (b)). The same position can be reached without issues when the end effector is pointing down (reachability set 1), but the length of the last link of the robot arm (between the TCP and the center of the wrist) is just large enough that the poses are out of the 6D workspace.

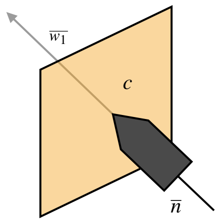
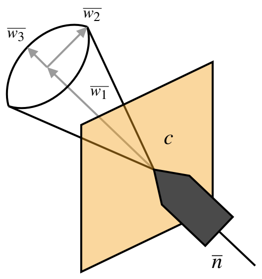
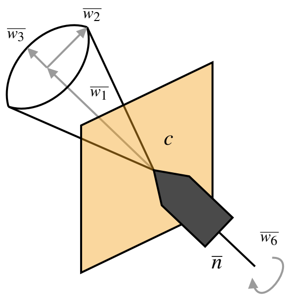
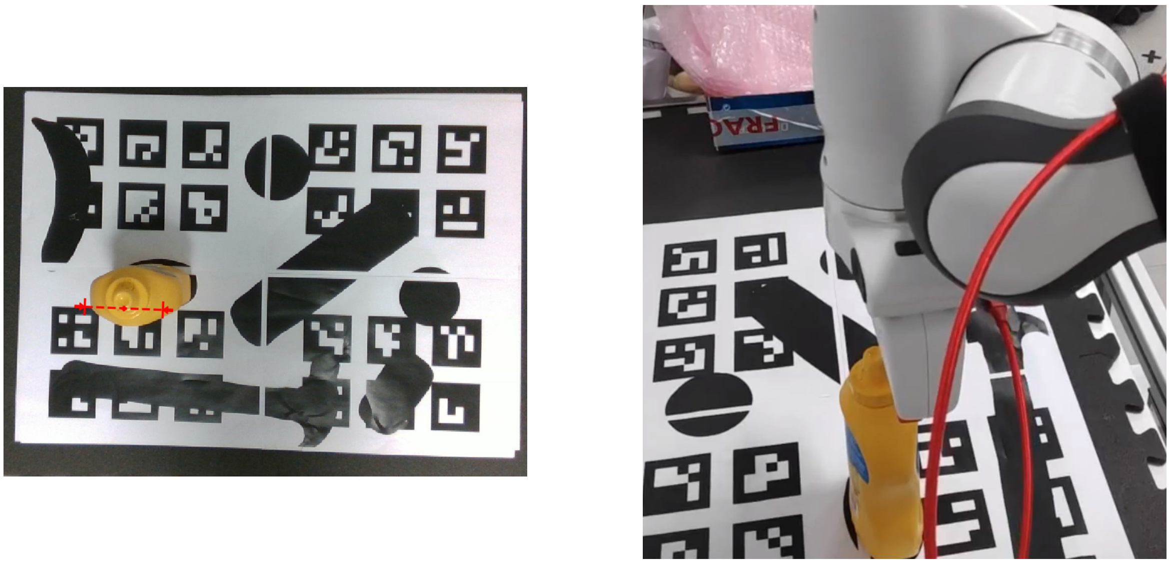
Contact model	Representation	Contact wrench w_c	Constraints
Point contact		$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} f_1$	$f_1 \geq 0$
Point contact with friction	 $\overline{w_2}$ $\overline{w_3}$ $\overline{w_1}$ c $\bar{n}$	$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}$	$\begin{cases} \sqrt{f_2^2 + f_3^2} \leq \mu f_1 \\ f_1 \geq 0 \end{cases}$
Soft finger contact	 $\overline{w_2}$ $\overline{w_3}$ $\overline{w_1}$ c $\overline{w_6}$ $\bar{n}$	$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$	$\begin{cases} \sqrt{f_2^2 + f_3^2} \leq \mu f_1 \\ f_1 \geq 0 \\ \lvert f_4 \rvert \leq \gamma f_1 \end{cases}$

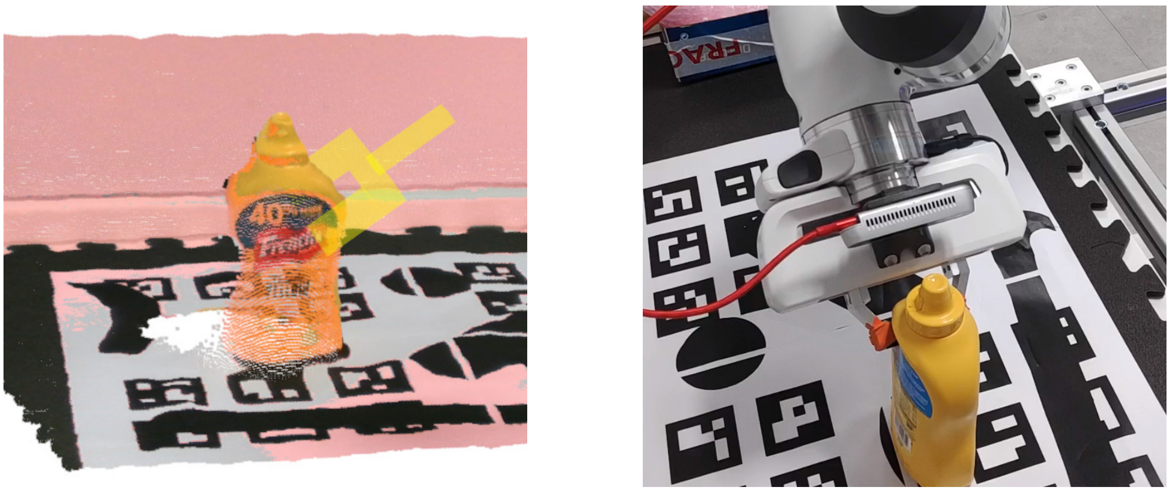
Table 4.2 Contact models typically used in manipulation. $\bar{n}$ indicates the contact normal, μ and γ indicate the Coulomb friction coefficients and f_i for i=1...4 represent the generalized forces acting on the contact. An in-depth explanation of these models can be found in [77].

Robot	End-effector	Modality	τ_p^r	τ_o^r	τ_p^c	τ_o^c
iCub	Right hand	In isolation	0.02 m	0.5 rad	0.045 m	0.8 rad
Panda arm	Franka Hand	In isolation	0.005 m	0.1 rad	0.01 m	0.2 rad

Table 4.3 User-defined parameters used during benchmarking procedure.



(a) Top grasp planned by Dex-Net



(b) Side grasp planned by GPD

Figure 4.16 Advantages of 6D grasp planning. Dex-Net only plans for top grasps and cannot take advantage of the full object geometry like GPD. (a) results in a failure, while (b) results in a success.

Layout		Per object scores							
Layout 0	$\bar{S}_L$	Object	$S0^L_k$	$S1^L_k$	$S2^L_k$	$\bar{S3}^L_k$	$\bar{S4}^L_k$	$\bar{S5}^L_k$	$\bar{S}^L_k$
	0.59	mustard bottle	1.0	1.0	1.0	0.15	1.0	0.8	0.95
		banana	1.0	0.75	1.0	0.32	0.8	0.2	0.36
		potted meat can	1.0	1.0	1.0	0.01	0.8	0.45	0.46
		gelatin box	0.75	0.25	1.0	0.07	0.2	0.0	N/A
		foam brick	0.75	0.25	1.0	0.27	0.2	0.2	N/A
Layout 0	$\bar{S}_L$	Object	$S0^L_k$	$S1^L_k$	$S2^L_k$	$\bar{S3}^L_k$	$\bar{S4}^L_k$	$\bar{S5}^L_k$	$\bar{S}^L_k$
	0.70	hammer	0.75	0.25	0.0	N/A	N/A	N/A	N/A
		tennis ball	1.0	1.0	1.0	0.23	0.2	0.2	0.29
		chips can	0.5	0.5	1.0	0.25	1.0	1.0	1.25
		banana	0.25	0.0	1.0	0.19	0.0	0.0	N/A
		potted meat can	1.0	0.75	1.0	0.01	0.8	0.7	0.71
		cracker box	1.0	1.0	1.0	0.04	0.8	0.5	0.54
		mustard bottle	0.75	0.25	1.0	0.23	0.8	0.15	N/A
Layout 0	$\bar{S}_L$	Object	$S0^L_k$	$S1^L_k$	$S2^L_k$	$\bar{S3}^L_k$	$\bar{S4}^L_k$	$\bar{S5}^L_k$	$\bar{S}^L_k$
	0.77	pear	1.0	0.75	1.0	0.0	0.0	0.0	0.0
		master chef can	1.0	1.0	0.0	N/A	N/A	N/A	N/A
		chips can	0.5	0.5	1.0	0.48	1.0	1.0	1.48
		medium clamp	0.75	0.25	0.0	N/A	N/A	N/A	N/A
		scissors	0.75	0.25	0.0	N/A	N/A	N/A	N/A
		power drill	0.25	0.0	0.0	N/A	N/A	N/A	N/A
		potted meat can	0.75	0.25	1.0	N/A	N/A	N/A	N/A
		tomato soup can	0.75	0.25	1.0	N/A	N/A	N/A	N/A
		tennis ball	1.0	0.75	1.0	0.07	0.4	0.4	0.43
		strawberry	1.0	1.0	1.0	0.13	0.6	0.55	0.51
		mustard bottle	0.5	0.5	1.0	0.25	1.0	1.0	1.25

Table 4.4 Results obtained when testing the superquadric cardinal point grasp planner on the iCub humanoid robot.

Layout		Per object scores							
Layout 0	$\bar{S}_L$	Object	$S0_k^L$	$S1_k^L$	$S2_k^L$	$\bar{S}3_k^L$	$\bar{S}4_k^L$	$\bar{S}5_k^L$	$\bar{S}_k^L$
	0.77	Mustard Bottle	1.0	1.0	1.0	0.09	0.2	0.05	0.08
		Banana	1.0	1.0	1.0	0.33	1.0	1.0	1.33
		Potted Meat Can	1.0	1.0	1.0	0.01	0.8	0.8	0.81
		Gelatin Box	1.0	1.0	1.0	0.09	0.8	0.65	0.72
		Foam Brick	1.0	1.0	1.0	0.07	1.0	0.85	0.92
Layout 0	$\bar{S}_L$	Object	$S0_k^L$	$S1_k^L$	$S2_k^L$	$\bar{S}3_k^L$	$\bar{S}4_k^L$	$\bar{S}5_k^L$	$\bar{S}_k^L$
	0.65	Hammer	1.0	1.0	1.0	0.12	0.6	0.0	0.07
		Tennis Ball	1.0	1.0	1.0	0.19	0.6	0.6	0.71
		Chips Can	1.0	0.92	0.0	N/A	N/A	N/A	N/A
		Banana	1.0	0.92	1.0	0.29	0.8	0.8	1.03
		Potted Meat Can	1.0	1.0	1.0	0.09	0.8	0.8	0.82
		Cracker Box	1.0	1.0	1.0	0.03	0.6	0.6	0.61
		Mustard Bottle	1.0	1.0	1.0	0.09	0.6	0.6	0.66
Layout 0	$\bar{S}_L$	Object	$S0_k^L$	$S1_k^L$	$S2_k^L$	$\bar{S}3_k^L$	$\bar{S}4_k^L$	$\bar{S}5_k^L$	$\bar{S}_k^L$
	0.77	Pear	1.0	1.0	1.0	0.14	0.0	0.0	0.0
		Master Chef Can	1.0	1.0	0.0	N/A	N/A	N/A	N/A
		Chips Can	0.92	1.0	0.0	N/A	N/A	N/A	N/A
		Medium Clamp	1.0	1.0	1.0	0.16	0.6	0.6	0.72
		Scissors	1.0	1.0	1.0	0.17	1.0	1.0	1.17
		Power Drill	0.92	1.0	1.0	0.08	1.0	0.85	0.92
		Potted Meat Can	1.0	1.0	1.0	0.12	0.8	0.8	0.92
		Tomato Soup Can	1.0	1.0	1.0	0.15	0.0	0.0	0.0
		Tennis Ball	1.0	1.0	1.0	0.13	0.0	0.0	0.0
		Strawberry	1.0	1.0	1.0	0.31	1.0	1.0	1.31
		Mustard Bottle	0.92	1.0	1.0	0.12	0.8	0.8	0.92

Table 4.5 Results obtained when testing Dex-Net on the Panda.

Layout		Per object scores							
Layout 0	$\bar{S}_L$	Object	$S0_k^L$	$S1_k^L$	$S2_k^L$	$\bar{S3}_k^L$	$\bar{S4}_k^L$	$\bar{S5}_k^L$	$\bar{S}_k^L$
	1.2	Mustard Bottle	1.0	1.0	1.0	0.16	1.0	1.0	1.34
		Banana	1.0	1.0	1.0	0.34	1.0	1.0	1.16
		Potted Meat Can	1.0	1.0	1.0	0.14	1.0	1.0	1.14
		Gelatin Box	1.0	1.0	1.0	0.14	1.0	1.0	1.15
		Foam Brick	1.0	1.0	1.0	0.22	1.0	1.0	1.22
Layout 1	$\bar{S}_L$	Object	$S0_k^L$	$S1_k^L$	$S2_k^L$	$\bar{S3}_k^L$	$\bar{S4}_k^L$	$\bar{S5}_k^L$	$\bar{S}_k^L$
	0.78	Hammer	1.0	1.0	1.0	0.13	0.8	0.25	0.36
		Tennis Ball	1.0	1.0	1.0	0.28	0.2	0.2	0.25
		Chips Can	0.92	1.0	0.0	N/A	N/A	N/A	N/A
		Banana	0.92	1.0	1.0	0.28	0.8	0.7	0.93
		Potted Meat Can	1.0	1.0	1.0	0.14	1.0	1.0	1.14
		Cracker Box	1.0	1.0	1.0	0.07	1.0	1.0	1.07
		Mustard Bottle	1.0	1.0	1.0	0.15	0.8	0.8	0.94
Layout 2	$\bar{S}_L$	Object	$S0_k^L$	$S1_k^L$	$S2_k^L$	$\bar{S3}_k^L$	$\bar{S4}_k^L$	$\bar{S5}_k^L$	$\bar{S}_k^L$
	0.83	Pear	1.0	1.0	1.0	0.23	0.2	0.2	0.25
		Master Chef Can	1.0	1.0	0.0	N/A	N/A	N/A	N/A
		Chips Can	0.92	1.0	0.0	N/A	N/A	N/A	N/A
		Medium Clamp	1.0	1.0	1.0	0.20	0.8	0.8	0.99
		Scissors	1.0	1.0	1.0	0.10	0.6	0.55	0.59
		Power Drill	0.92	1.0	1.0	0.10	1.0	0.95	1.06
		Potted Meat Can	1.0	1.0	1.0	0.17	1.0	1.0	1.18
		Tomato Soup Can	1.0	1.0	1.0	0.18	1.0	1.0	1.17
		Tennis Ball	1.0	1.0	1.0	0.17	0.4	0.4	0.44
		Strawberry	1.0	1.0	1.0	0.27	0.6	0.5	0.65
		Mustard Bottle	0.92	1.0	1.0	0.14	1.0	1.0	1.12

Table 4.6 Results obtained when testing GPD on the Panda.

4.7 Remarks and further development

So far, in this chapter we have outlined the GRASPA benchmarking protocol and its objectives, explained the motivation behind the choices made in designing its metrics and shown an application on two different robot platforms. By recalling the main points this work aimed to address (see Section 4.1) and the results obtained experimentally, we can draw some conclusions and point to relevant improvement directions.

GRASPA favours reproducibility of grasping experiments. The reproducibility claims heavily rely on the usage of a marker board where objects have to be placed in specific poses. The board is inexpensive and easy to print on different paper formats. Placing the objects on the printout shapes with precision is an easy task, since the object footprints are obtained from object meshes. On one hand, the choice to use a subset of YCB objects allows many different users around the world to replicate the same experimental conditions, given the spread of the object set. On the other hand, users that are not already in possession of the dataset will not be able to adopt the benchmark. A viable way to lower this barrier to entry is to rework the object set used by GRASPA in a way to be easily and inexpensively 3D-printable. While this is a relatively easy fix, consumer-grade 3D printers cannot yet faithfully produce exact replicas of the YCB objects in terms of color, material and texture of the YCB objects. In order to lower the barrier even more, GRASPA could be reworked to replace YCB objects with shapes generated as meshes in the first place, and then rendered as real objects through 3D printing instead of the other way around. As Morrison et al. show in their work [73], it is possible to use generative models to produce 3D-printable objects based on how challenging they are to being grasped. This is an interesting possible development, since it would also give a more grounded and objective explanation for the choice of objects in the benchmarking set.

GRASPA is adaptable to different robotic platforms. We tested the approach on two platforms with different kinematic structure, capabilities and end effectors. Although GRASPA provides a convenient and easy to install way to compute the benchmark scores and metrics, the burden of writing a custom software application to gather the required data from the target platform is left to the user. This allows for maximum flexibility, but it also could turn down potential users since the time and effort overhead it adds is non-negligible. Given the variability in robot setups, middleware and hardware used in the robotics research community, balancing versatility and deployment effort is a zero-sum game. The best improvement direction for GRASPA in this sense is to turn it into a software package integrated in ROS, as at the time of writing this document it is the most widespread and open-source middleware for robotics.

GRASPA can be adapted to many grasping pipelines. We deployed GRASPA in order to test three grasping pipelines that are quite different in terms of assumptions and performance. For all of them, we managed to compute and gather all the data required by the benchmark to compute its metrics and problems in the deployment of the benchmark were mostly due to the effort required in adapting the pipelines to the robotic platform, instead of the protocol itself.

GRASPA proposes a granular way to test grasping pipelines. While discussing the outcome of the experiments (Section 4.6.2) we have shown how having a performance index for different aspects of grasping pipelines can explain away some apparent inconsistency in the results. In particular, we found the reachability index $S0_k^L$ and the calibration index $S1_k^L$ for each object to be especially useful in diagnosing failure cases. While the concept of breaking away from simple success/failure metrics is useful, we identified some shortcomings in how some of these metrics are defined. In particular, the way the metric $\bar{S3}_k^L$ is computed also has room for improvement. As explained in Section 4.5.2, the way grasp quality is computed at the moment is disadvantageous while evaluating grasps performed with parallel jaw grippers, due to the contact model implemented in the Simox toolbox. While testing grasps in a virtual environment certainly helps disentangling the "ideal performance" from the one experimentally observed on the real robot, such virtual environment must treat grippers and multifingered hands in the same way. Hence, an improvement direction for GRASPA is to rework the contact modeling within the Simox toolbox or to migrate towards the usage of a different physics-based framework. The method used to sample the workspace in order to compute these scores is quite coarse in the current state and, more critically, the computation of $S0_k^L$ and $S1_k^L$ requires the user to pick error thresholds without a rigorously defined protocol. More on this in the next point.

GRASPA accounts for the limits of the test platform. As already stated, GRASPA aims to be flexible with respect to the platform it is being deployed to, however we recognize that the choice of these parameters is unbalanced in nature. As they are curently defined, in specific conditions $S0_k^L$ and $S1_k^L$ reduce the negative impact of a grasp failure while simultaneously not amplifying successes. For example, consider the *mustard bottle* in Layout 1 in Table 4.4 and the *tennis ball* in Table 4.5. In the former case, iCub could not compute a good quality grasp for the object, nor grasp it in a stable manner, nonetheless since $S0_k^L \leq 0.5$ this failure case does not affect the overall score $\bar{S_L}$. In the latter case, both reachability and visual calibration metrics obtained full scores and the low $\bar{S3}_k^L$, $\bar{S4}_k^L$ and $\bar{S5}_k^L$ affected the compound score $\bar{S_L}$. In other words, the arbitrary choice of the τ thresholds lead to the fact that having a more precise, better calibrated robot makes the failure cases count more while having a less accurate robot with poor vision calibration is forgiving with respect to grasp failures. This could be improved upon by reworking the way $\bar{S}_k^L$ is computed.

4.8 Impact of GRASPA on the community

Version 1.0 of GRASPA was accepted [12] in the context of the RA-L Special Issue in Benchmarking Protocols for Robotic Manipulation, and has been included in the YCB official website, in the Protocols and Benchmarks section[10].

At the time of writing this thesis, the GRASPA protocol is having some resonance in the robotics research community. As already mentioned, the benchmark in its entirety requires quite some effort on the user side in order to be set up on the target robot platform, nonetheless the way GRASPA proposes to deal with experiment repeatability has attracted some attention. For instance, some works in the grasping and pose detection field [6, 110] have adopted the layouts and printable marker board to validate the proposed approaches. In addition, GRASPA was adopted in the context of the HEAP CHIST-ERA project as explained in the next section.

4.8.1 GRASPA and project HEAP

GRASPA has had a relevant impact on the HEAP CHIST-ERA project, a project including a number of academic European partners in the field of robotic manipulation and grasping. The project proposes to tackle current challenges in automatic and human-guided manipulation of objects in scenarios where the objects are not known a priori or have been deformed or separated in parts. Moreover, HEAP puts the focus on building an end-to-end benchmarking framework, which includes rigorous scientific methodology and experimental tools for application in realistic scenarios.

One of the initial efforts of the project was to establish a common robot setup and build a customized simulation environment to obtain its digital clone. The partner consortium settled on using Franka Emika Panda robot arms and Intel Realsense cameras in a tabletop

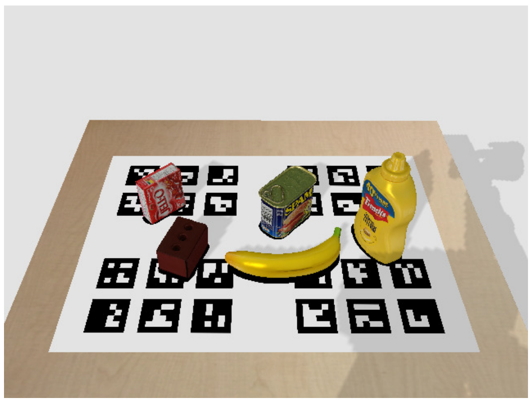
[10]http://www.ycbbenchmarks.com/protocols-and-benchmarks

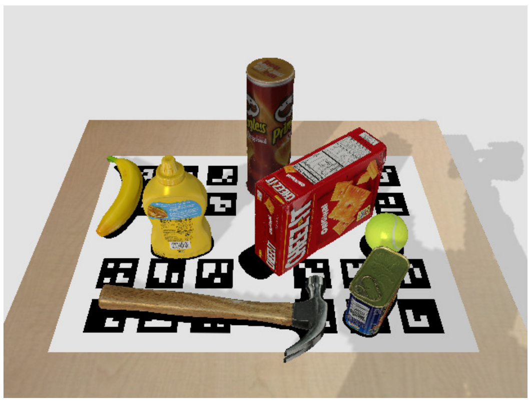
scenario. Our contribution to this initial effort consisted in the integration of the GRASPA object set and layout structure in the aforementioned simulation environment (based on the Bullet physics engine and PyBullet [21]). Since such environment would have to be used to test grasping algorithms and pipelines starting from the acquisition step, we implemented the GRASPA layouts as scenes using the most high-quality mesh renditions of the YCB object subset available to date. We used the meshes provided with the YCB-Video dataset [128] since they are more accurate and texture-complete with respect to those in the original YCB dataset. Rendered examples of such scenes are shown in Figure 4.17.

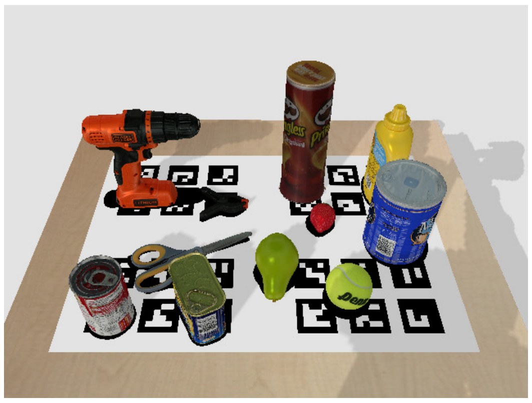
Even though the intended use for GRASPA is to be deployed on real robotic setups, we adapted our benchmarking framework (refer to Section 4.5.1 for details) to the simulated environment to field-test its performance and report issues to the development team. In fact, the consortium is currently striving to fully integrate the simulated robot setup in the ROS ecosystem, in order to be able to seamlessly switch between it and the real setup to speed up development and experimentation.

Parts of the GRASPA framework, in particular the layout, evaluation metrics and general idea were adopted as standard in the HEAP project also for what concerns the real-world experimental evaluation of grasp planning algorithms. As a matter of fact, the evaluation of GPD and Dex-Net using the GRASPA benchmarking protocol is part of an effort aimed at establishing a performance baseline using state of the art grasp planning and grasp detection approaches.

HEAP is also providing challenges and interesting directions for further developments of GRASPA. On the one hand, in the context of the project some aspects of GRASPA related to the adaptability of the benchmark to different setups are not really relevantm since the consortium has agreet to use a common setup. On the other hand, the ambitions of the project point towards including more challenging layouts that include heaps of deformed or broken objects that go beyond what the YCB dataset proposes. In fact, a research direction consists in integrating a number of challenging objects obtained from deep 3D generative models (on the line of works such as EGAD [73]). Another possible development direction points towards extending the way GRASPA computes metric $S3$ in simulation to include more advanced contact models and an ensemble of grasp quality measures of different nature instead of a single one based on the computation of the GWS [97].


(a) Layout 0


(b) Layout 1


(c) Layout 2

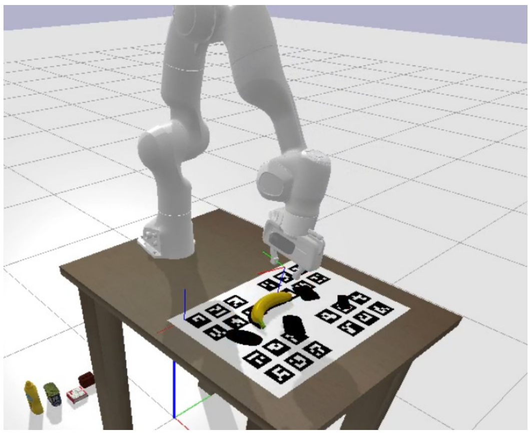

(d) The complete scene environment

Figure 4.17 The GRASPA layouts in the HEAP simulation environment. From (a) to (c), the GRASPA boards as rendered by the simulated Intel Realsense camera mounted on the robot hand. in (d), a render of the complete scene during grasp execution.

Chapter 5

Towards shape completion strategies for grasp planning

This Thesis work has been focused on scenarios where the shape of objects to be grasped is not known a priori. We have hypothesized that objects can be modeled with a superquadric in Chapter 2 and we have included the object 2D appearance to only perform instance segmentation in Chapter 3, while in Chapter 4 we have benchmarked two model-free state of the art grasp planners. At runtime, we have always assumed that a complete and detailed 3D description of the object is not available a priori in order to plan for grasps. We have also implicitly stated that such a description cannot be recovered from a single view observation because of the auto-occlusion[1] property of non-clear objects.

In order to plan for grasps given the auto-occlusion property, we have several options:

- grasps can be planned without explicit modeling of the unseen parts. This is the case, for instance, of many approaches that detect grasps from partial observations like GPD or Dex-Net (see Section 4.5 for specific details)
- more information can be gathered by integrating several views acquired from a variety of viewpoints, before planning for grasps. This can be done with a multiple calibrated cameras setup, or if the camera is mounted on the robot end effector and can be moved around
- make assumptions on the object shape given the observations, i.e. performing a modeling operation, and planning grasps on the model.

[1]The term *auto-occlusion* simply refers to the intuitive fact that we cannot observe points that are on the other side of the object.

Combining the first and second point seems to be the most effective solution. If the robot does not need to be moved, a number of calibrated RGB-D cameras can be placed around the workspace, or a single robot-mounted camera can be moved around the object if the robot is not fixed. This however does not help with cluttered scenes like object piles or structured clutter, which is often the case in grasping tasks. In this case, the shape of the object cannot be perceived without modeling it.

A wide variety of assumptions on object shapes can be made in order to formulate a modeling problem. Shapes can be approximated by geometric primitives, under the assumption that said primitives are fit for grasp planning. In Chapter 2, for instance, we have presented methods that use such approximations. Heuristics can also be put in place to reconstruct the missing parts, e.g assuming they are symmetrical to the observed parts. We could also assume similarity with objects whose 3D model is known, or the objects themselves are known, as in the pose estimation problem. Finally, we could attempt to reconstruct the whole object shape by using a model that has learned to complete a variety of different objects given partial measurements. This last approach is called *shape completion* in the literature, and has gained traction in latest years thanks to the popularity of 3D deep learning frameworks.

As it often happens, the rising interest in shape completion methods has recently spilled over from the computer vision field into robotics research. Algorithms and architectures that can reconstruct the complete 3D shape from partial views constitute a "superpower" that can be useful in a number of applications, from motion planning to navigation and scene understanding. Arguably, the robotics subfields that can benefit from shape completion the most are manipulation and grasping, since vision-based approaches are no longer limited by partial measurements and can rely on complete 3D information instead.

Given the novelty of the field, this Chapter is supposed to be an exploratory study for shape completion techniques for robot grasping. In Section 5.1, we review some recent relevant work on the subject. In Section 5.2 we describe the shape completion problem in more detail, delving into the types of representations and the most promising architectures. In Section 5.3 we replicate the results of two approaches from the state of the art, analyzing one of them in depth in order to provide insight on its features and limitations. Finally, in Section 5.4, we summarize our findings and propose interesting research directions towards improving the state of the art on the subject.

5.1 Related Work

To the best of our knowledge, the first approach to deep learning-driven shape completion for robot grasping is the work by Varley et al. [116]. The proposed pipeline feeds a partial object point cloud into a 3D CNN in the form of a voxel grid. The output is different voxel grid representing the completed shape, which is transformed into a mesh via marching cubes [59]. The best grasp is then computed via a sampling-based scheme with analytical metrics, by using the GraspIt! [72] grasp simulator. Lundell et al. [60] inherit the same pipeline, substituting a VoxNet-based deep autoencoder to the 3D CNN. The architecture employs dropout to sample the latent space at inference time in order to produce a number of possible shape voxel grids that are then averaged and meshed. Lundell et al. [61] attempts the reconstruction of the whole scene, segmenting the 3D point cloud of a cluster of objects and completing every shape using a very similar approach to the previous work by the same authors [60]. After the whole scene is completed, a virtual depth map is rendered from a fixed number of points around it, and candidate grasps are obtained through the out-of-the-box Dex-Net implementation [62]. This produces (almost) 6-DoF grasps, i.e. planar grasps detected on a number of different planes.

The methods cited so far cast the input 3D data in a voxel grid representation to be used both as input and output, but other representations have been used as well. Van der Merwe et al. [114] directly use the input point cloud to approximate the signed distance field (SDF) of the completed shape. In this case, the authors use the compactness of the SDF representation in order to plan for grasps by accounting for the feasibility of the motion with respect to the scene. Gao and Tedrake [39] propose a novel representation embedding geometric information (e.g. mesh or point cloud) with semantic keypoints (e.g. top and bottom object centers). This approach employs state of the art deep learning methods for both shape completion and keypoints detection and combines them in order to plan for grasps and approach trajectories.

Some methods attempt shape completion with a multimodal approach, i.e. visual and tactile inputs. Watkins-Valls et al. [123] use a similar pipeline to [116], employing a 3D deep autoencoder with a multimodal input (consisting of tactile measurements and partial point cloud) to probe the occluded side of the object and produce a more accurate mesh on which grasps or manipulation actions can be planned. Wang et al. [122] incorporate information from both 2D vision and tactile sensors. The approach leverages prior knowledge learned from a large shape dataset to obtain an initial estimation of the complete shape from a color image, which is then refined using tactile sensing.

5.2 Background

In this Section, we provide some notions in order to ease understanding the general structure of the typical autoencoder-based shape completion pipeline. We also provide insight on some popular representations used in the field, and explain the inner workings of two interesting approaches from recent literature.

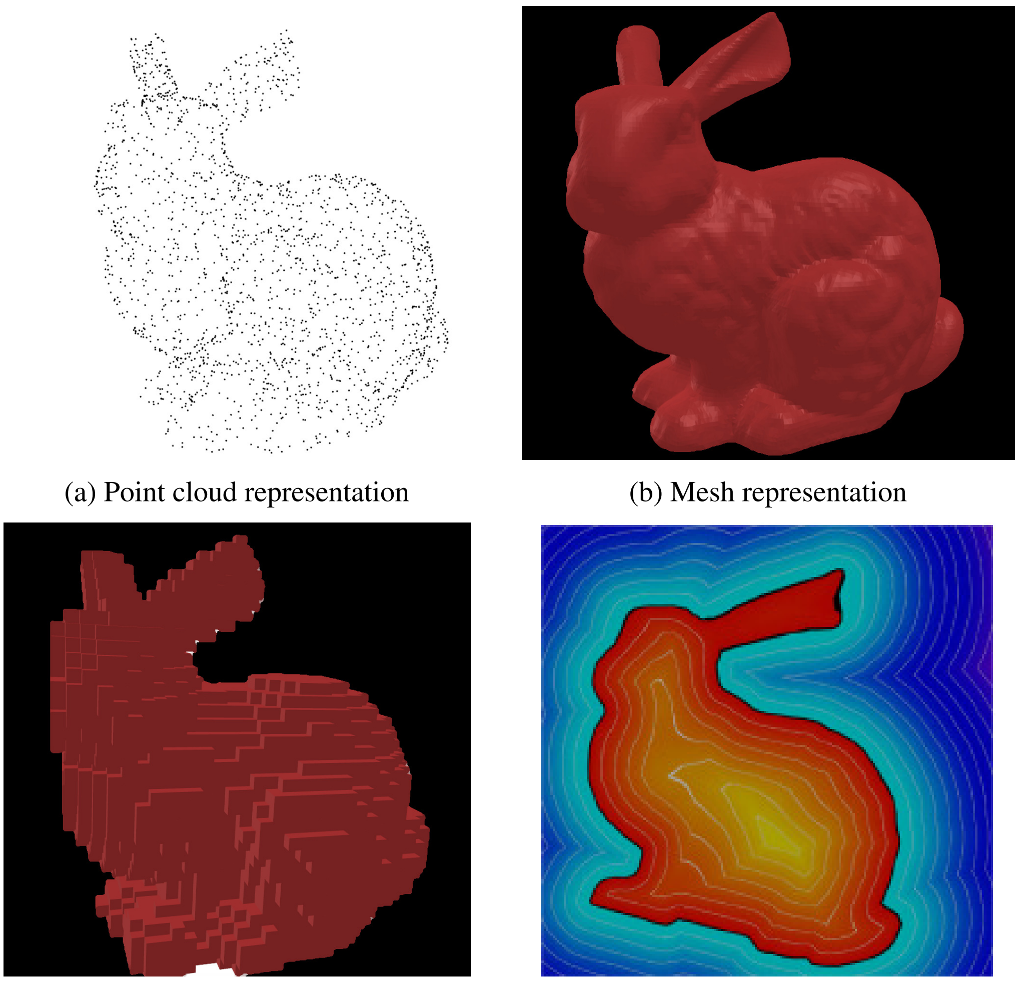
5.2.1 Different representations of 3D shapes

Over the years, a number of different representations for 3D data and shapes have been employed according to the target task and constraints. We hereby recall the definition of the most relevant ones in the shape completion field. A visual reference for these representations is shown in Figure 5.1.

Point cloud. This is the most popular and simple 3D representation used in robotics. It consists in a sparse set of points $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ expressed in some reference frame, typically in a Cartesian space, that can also contain information such as the point color and normal vector. Point clouds are a natural representation to use in robotics, since each depth map pixel from a RGB-D camera or stereo rig corresponds to a point. Due to their sparse and unordered nature, they are typically converted into other representations before being used as input of machine learning and deep learning models. In very recent years, nonetheless, a number of models [91, 92, 53, 125, 108] that take a point cloud as input have surfaced, becoming a feasible backbone for deep shape completion.

Mesh. In its simplest form, a mesh is a collection of vertices, edges and polygonal faces. In computer vision and robotics, meshes are typically used when face planes and normals are useful, for instance rendering applications in computer graphics and collision checking and/or grasp planning in robotics. Because of this reason, this representation is typically the output of a shape completion pipeline, often obtained from a voxel or point cloud representation using popular mesh construction algorithms such as marching cubes [59].

Voxel. Voxels are an extension of the pixel concept to the 3D space. Voxel grids discretize a limited portion of space in cells of equal size, producing a dense representation. Voxel grids are very popular in the field of shape completion and deep learning for 3D vision: on the one hand, it is trivial to convert point clouds into voxel representations, therefore removing



(a) Point cloud representation (b) Mesh representation

(c) Voxel grid representation (d) Signed Distance Field representation

Figure 5.1 Different 3D representations of the Stanford Bunny model.

the problem of sparse representation; on the other hand, their similarity to their 2D pixel counterparts implies the convolution operator can be adapted in a straightforward way [69].

Continuous representations. The representations outlined so far are inherently descrete, or are limited in resolution. To overcome this limitation, recent works have proposed the usage of continuous representations as output of a deep neural network relying on concepts such as Signed Distance Function (SDF) [81], occupancy functions [70] or Gaussian Process Implicit Functions (GPIS) [38]. Some of the features that make these representations desirable are a potentially infinitesimal sampling granularity, smoothness and, in the case of GPIS, the native estimation of uncertainty.

5.2.2 Deep autoencoders for shape completion tasks

Deep autoencoders are at the core of most state of the art data-driven shape completion approaches and architectures. In this Section, we briefly show its definition and the basic idea behind its usage for shape completion, in order to ease comprehension of the next Section.

Autoencoders are dimensionality reduction methods that compress the number of features used to describe data. Assuming some input data $x \in \mathbb{R}^n$, we wish to compress it into a lower-dimensional feature space through an *encoder* function $z = e(x),\ \ \mathbb{R}^n \rightarrow \mathbb{R}^m,\ m < n$ (Figure 5.2a. The co-domain of the encoder, i.e. the compressed feature space, is typically called *latent space*. In order to be able to decompress the data mapped in the latent space, a *decoder* function is needed, i.e. $\hat{x} = d(z),\ \ \mathbb{R}^m\ \rightarrow \mathbb{R}^n$.

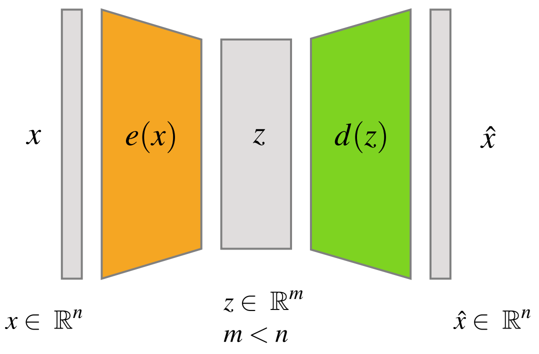
The choice of $e(\cdot)$ and $d(\cdot)$ is what defines an autoencoder. The ideal goal is to make sure the encoding/decoding process can restore the initial data with no information loss, i.e. $x = \hat{x} = d(e(x))$. More realistically, if we indicate possible encoder and decoder functions as E and D, for a given m the choice of $e(\cdot)$ and $d(\cdot)$ must minimize the reconstruction information loss ε:

$$(e^*, d^*) = \underset{(e,d) \in E \times D}{\arg\min}\ \varepsilon(x, \hat{x}) \tag{5.1}$$

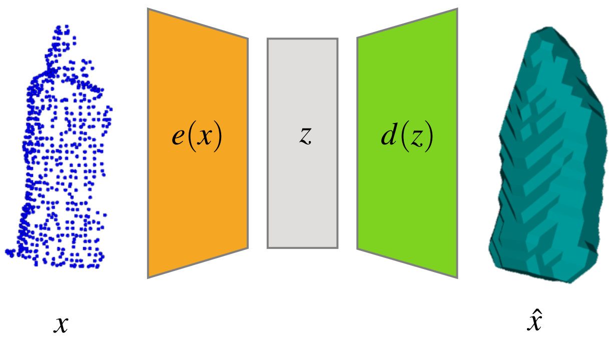
$$= \underset{(e,d) \in E \times D}{\arg\min}\ \varepsilon(x, d(e(x))) \tag{5.2}$$

where the loss ε is a criterion to measure the difference between the input and output data. For instance, $e(\cdot), d(\cdot)$ can be chosen within the linear family. In this case, it can be shown that the PCA (Principal Component Analysis) technique can be a valid linear encoding-decoding scheme. In order to enhance the compression capability of the scheme $e(\cdot), d(\cdot)$ can be chosen within any neural network family, making them highly nonlinear. In this case, the reconstruction problem in Equation 5.2 can be cast as an iterative optimization process, and tackled with backpropagation and gradient descent techniques.

The dimensionality m of the latent space is also a critical parameter to the autoencoder, and it must be tuned according to the capacity of the encoder and decoder. A latent space that is too small might not be enough to grant minimum reconstruction error, and therefore the majority in the information will be embedded in the parameters learned in $e(\cdot)$ and $d(\cdot)$. The edge case of this scenario is $m = 1$ with infinite capacity of the encoder and decoder: in this case, every data point could be mapped into a different value of the scalar indicating the latent space (e.g. $z = 1, 2, \ldots, N$ where N is the number of training data points). A latent



(a) Visualization of the general autoencoder architecture.



(b) A 3D autoencoder used to reconstruct a complete shape.

Figure 5.2 Intuition for shape completion deep autoencoders. In (b), the reconstructed shape is transformed into a mesh for visualization purposes.

space that is too large might lead to overfitting the training dataset, with each example being mapped in a different region of the latent space, with minimal overlap. Note that framing the optimization as Equation 5.2, even though granting minimum reconstruction error on the training dataset, does not constrain distribution of the latent space in any way, leading to the abscence of structures that could be exploited during inference, i.e. lack of *regularity*.

Autoencoder models are used for a large variety of tasks. Shape completing autoencoders are inspired to the *denoising* (also known as *reconstructive*) kind, tasked with learning to reconstruct a noisy or corrupted input from uncorrupted ground truth. For shape completion tasks, during training the encoder is presented with single view, partial 3D data $\tilde{x}$ and the reconstructed output $\hat{x}$ is used to compute the reconstruction error with respect to the ground truth x, i.e. the complete version of $\tilde{x}$. If (e^*, d^*) are the parameters of the encoder and decoder deep networks, Equation 5.2 becomes

$$(e^*, d^*) = \underset{(e,d) \in E \times D}{\arg\min} \; \varepsilon(x, d(e(\tilde{x}))) \tag{5.3}$$

At inference time, the autoencoder is presented with a partial object observation, and the completed shape is retrieved at the output (Figure 5.2b). As example, in the following we describe the main structure of the autoencoders for shape completion used by Lundell et al. [60] and Van der Merwe et al. [114], which are used in the rest of this Chapter for an exploratory study.

A voxel-based shape completion autoencoder. The work by Lundell et al. proposes an autoencoder based on the 3D CNN autoencoder proposed by Dai et al.[22], augmented by skip connections and a latent vector enclosed by dropout layers. The input is cast from a partial point cloud into a fixed size voxel grid, and the output is obtained in the form of a voxel grid that has to be transformed into a mesh. At inference time, the same partial input voxel grid is fed through the completion process multiple times, each with a different dropout scheme around the latent space. Then, a mean shape is computed and grasps are detected on it by using the GraspIt! [72] simulation tool. This process of exploiting dropout to probe the uncertainty of a learned deep model is called MC-Dropout [37] and has probabilistic foundations that are outside the scope of this study.

We employ this method in Section 5.3 in order to test the possibility of using such a method to replace partial point clouds with completed point clouds as input of an out of the box 6DoF grasp planner.

A hybrid shape completion autoencoder. The work by Van der Merwe et al. proposes a shape completion autoencoder that uses partial object point clouds as input and outputs an SDF function. It uses PointConv [125] layers as encoder and a simple MLP[2] as decoder. After the partial point cloud is embedded as a latent vector $z = e(\tilde{x})$, the decoder uses z and query points $q_i = (x_i, y_i, z_i) \in \mathbb{R}, i = 1, \ldots, N$ to sample the SDF of the completed shape at those points, $\text{SDF}(q_i) = d(z, q_i)$. The idea is to estimate the shape of the completed object in an implicit form via the signed distance function instead of an explicit formulation like a voxel grid. There are many advantages to this approach:

- the absence of 3D convolutions make both training and inference fast and with a smaller memory footprint

[2]Multi Layer Perceptron, i.e. a deep neural network with fully connected layers.

- the SDF is that it is continuous and differentiable
- a point cloud, mesh or voxel grid representation can be obtained from the SDF by sampling points q_i the 3D space with arbitrary resolution. The shape is obtained by following the surface boundary where $\text{SDF}(q_i) = 0$.

The SDF representation is used by the authors to perform geometry-aware grasp and motion planning. In Section 5.3.1, we try to analyze the internal representation of the PointSDF in order to identify any structure or exploitability.

5.3 Out-of-the-box shape completion for grasp planning

In Chapter 4.5 we have benchmarked two state of the art grasp planners that make use of learned models (i.e. GPD [82] and Dex-Net [62]) and experimentally showed that they performed well out of the box, with some tuning due to the differences in robot setup. The benchmarking setup used a single hand-mounted camera, hence both planners operated with a partial view of the scene objects. During the experiments on the real robot, the successful candidates were mostly limited to visible parts of the objects and often failed when contacting unseen parts of the object. This is somewhat expected, since that is the task the algorithms were trained for. While working on superquadric-based grasp planners (Chapter 2), we observed that modeling complete objects, although with simple shapes such as superquadrics, would lead to candidates on the hidden sides of the object that would eventually prove to be good grasps during execution.

According to these observations, we hypothesize that a state of the art 6DoF grasp planner trained on partial point clouds can efficiently leverage information obtained with shape completion methods in order to obtain a larger number of good candidates. Lundell et al. [61] already proposed something similar by completing a single view scene data, rendering synthetic depth maps from different points of view and planning for grasps using the same out of the box Dex-Net planner we benchmarked in Section 4.5. Although interesting, we argue this approach limits the grasp orientation of the candidates due to the finite number of viewpoints (12 in the paper) and does not offer the versatility of a fully 6DoF approach.

As a proof of concept, we have tested the same approach to shape completion shown in Lundell et al. [60] in combination with GPD in order to show the difference between 6DoF grasps computed on a partial point cloud and on a completed shape. We used a Intel Realsense to gather partial point clouds from 10 real YCB objects, using a simple RANSAC plane removal to segment the object from the tabletop surface, and we fed them through the entire shape completion pipeline. We produced 50 different completions for every single partial point cloud, in order to obtain a meaningful average completed shape. We then sampled the resulting point clouds with a resolution similar to that of the camera we used during acquisition, obtaining completed point clouds. At this point, we detected 6DoF grasps using the out of the box GPD algorithm on both sets. We set no constraints in terms of position and orientation on the candidates, generating a set of 500 grasps on each point cloud and reporting the best 10. Visual results of this procedure are shown in Figure 5.3.

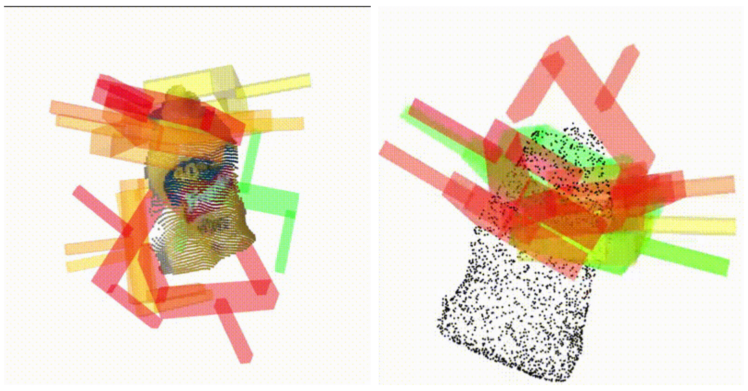
Inspecting these results, it appears like the GPD grasp detector can potentially benefit from the completion procedure. This seems to be an improvement for some of the objects:

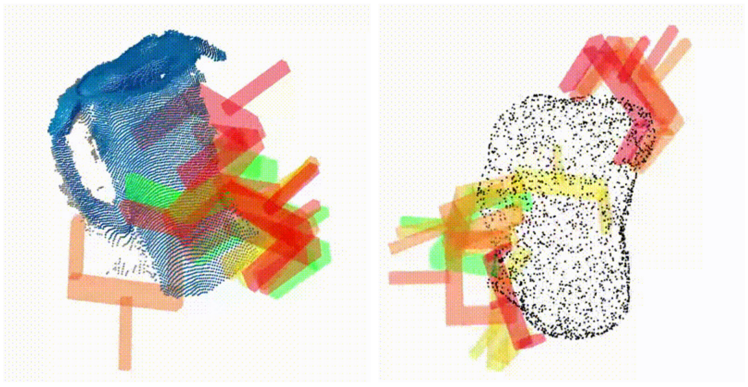
- on the *mustard_bottle*, planning on the partial point cloud produces clusters of grasps on the cap and base area, while planning on the completed shape outputs clusters on the sides of the object, where the gripper can exploit more surface area (Figure 5.3a)
- on the *pitcher*, all the candidates planned on the partial point cloud are clearly infeasible. The pitcher is a large object, only graspable by the gripper on the edges and the handle, which is where GPD detects feasible grasps in the completed shape (Figure 5.3b)
- grasps planned on the *power_drill* were good on both the partial and the completed cloud. While the grasps obtained from the partial point cloud clustered on the top of the drill, the ones on the completed shape were clustered mostly on the base and handle (Figure 5.3c).

Quantitatively, the grasp quality of candidates computed by GPD on the completed shapes resulted to be around 15% higher on average than those computed on partial point clouds. This grasp quality metric comes from the CNN in the GPD pipeline, and it is prone to an overestimation of the quality of grasps computed around edges (as highlighted by the pitcher results); nonetheless, this is a promising research direction and suggests further analysis and experimentation with other methods.

5.3.1 Studying the internal representation

So far, the only meaningful value metric we have considered for shape completion autoencoders is the reconstruction error, i.e. how good the completion is with respect to the ground truth and how it can be used for grasping. However, in Section 5.2.2 we have introduced autoencoders as dimensionality reduction methods and we have pointed out that the latent space representation is just as important as the output, as it represents the quality of the encoding. A well-structured latent space can be exploited for both shape completion and


(a) Grasps computed on Mustard Bottle point clouds.


(b) Grasps computed on Pitcher point clouds.

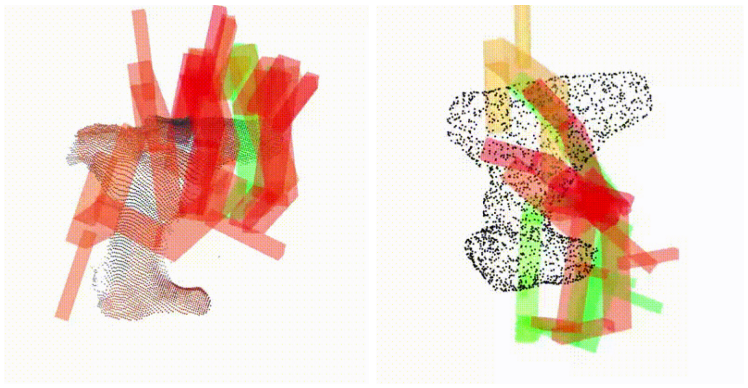

(c) Grasps computed on Power Drill point clouds.

Figure 5.3 Grasps computed by GPD on partial point clouds (left) and on the relative completions (right) obtained with Lundell et al.. Candidates shown in green indicate better grasp quality, as computed by the GPD candidate ranking model.

grasping tasks; for instance, an interpretable latent space might reveal information about the shape prior [22], or might contain hints about the object size, keypoints, or the presence of concavities. However, to the best of our knowledge, none of the works mentioned in Section 5.1 include conditioning the structure of the latent space through a regularization term in the training loss function. We decided to investigate the latent space of PointSDF, due to the simplicity of the architecture (the decoder is a simple MLP), looking for structures and patterns in the encoded inputs.

The objective of this analysis was to embed a number of different partial views of YCB objects in the latent space of the PointSDF model in an attempt to observe the distribution of the embeddings. Since the pre-trained model provided by the authors of the paper did not seem to behave well with respect to the noise model of our RGB-D camera, we opted for synthetic point clouds with a controlled noise model. We rendered 50 partial point clouds for each object, uniformly sampling viewports on a sphere centered on the center of mass of the object center. We added noise to the rendered depth map before converting it to a point cloud, as described in the paper. Then, we embedded each partial point cloud and analyzed the distribution of the embeddings.

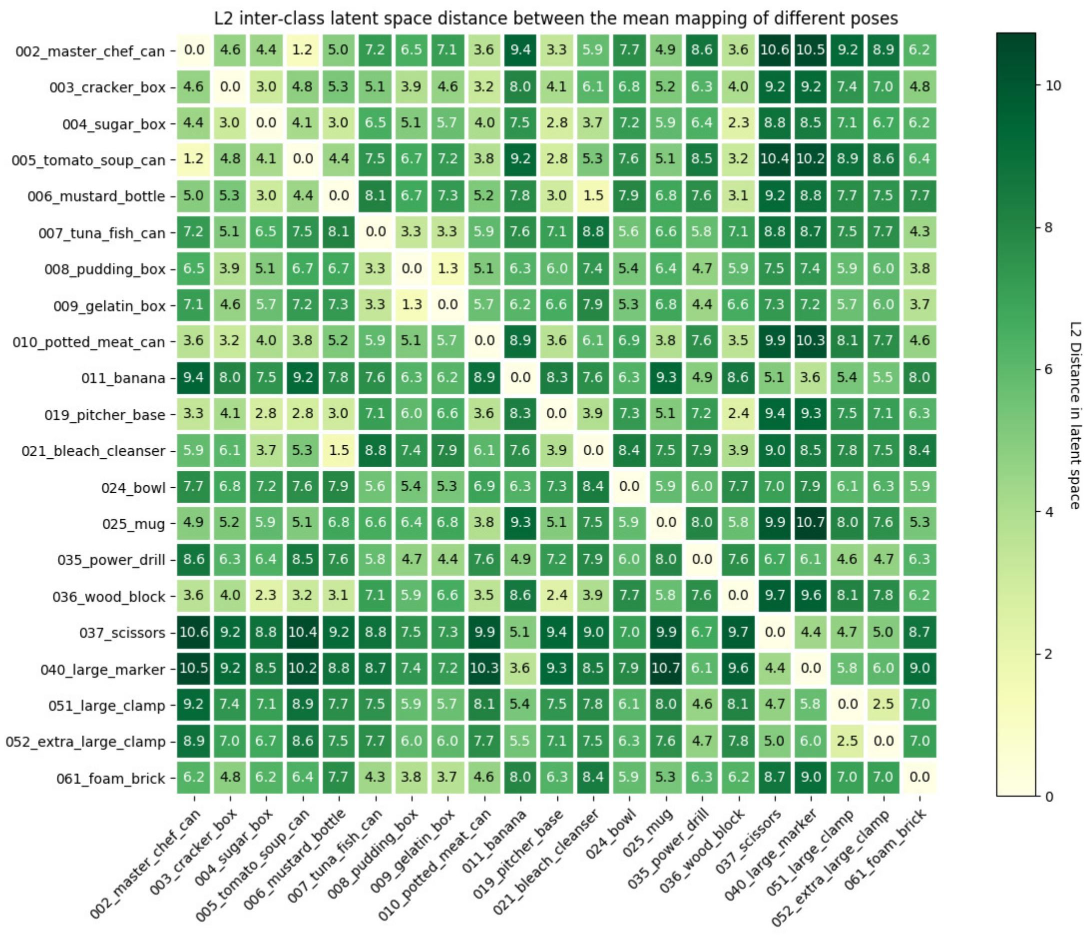
The latent space of PointSDF has dimension 256, therefore is well beyond being directly interpretable. In order to visualize the distribution of the embeddings, and detect any clustering, we used two methodologies:

- the first involves computing the distance between embeddings of different objects. We compute this distance $D(o_i, o_j)$ by averaging the Euclidean distance of every embedding of the i-th object from every embedding of the j-th object. Results according to this metric are shown in Figure 5.4
- the second involves using t-SNE [113] to visualize the embeddings in a 2D interpretable space. Results according to this metric are shown in Figure 5.5.

This experiment shows that, for some objects, the resulting embeddings clusters are somewhat separated even if there is no explicit regularization term in the loss function used for training PointSDF (refer to the paper for further details). This is the case, for instance, of the clamps (*extra_large_clamp* and *large_clamp*).

Objects that have a similar shape are also somewhat clustered in different portions of the latent space[3]: elongated objects such as the *banana*, *large_marker* and the aforementioned clamps are well separated from the box-like objects, that in turn are separated from cylindrical objects such as *tomato_soup_can* and *master_chef_can*. Other objects, for instance the *wood_block*, *mug* and *bowl* do not feature clustered or organized representations, and are scattered all over the latent space.

[3]Although t-SNE reduces the dimensionality from 256 to 2, the method guarantees the best projection of global and local structures



	002_master_chef_can	003_cracker_box	004_sugar_box	005_tomato_soup_can	006_mustard_bottle	007_tuna_fish_can	008_pudding_box	009_gelatin_box	010_potted_meat_can	011_banana	019_pitcher_base	021_bleach_cleanser	024_bowl	025_mug	035_power_drill	036_wood_block	037_scissors	040_large_marker	051_large_clamp	052_extra_large_clamp	061_foam_brick
002_master_chef_can	0.0	4.6	4.4	1.2	5.0	7.2	6.5	7.1	3.6	9.4	3.3	5.9	7.7	4.9	8.6	3.6	10.6	10.5	9.2	8.9	6.2
003_cracker_box	4.6	0.0	3.0	4.8	5.3	5.1	3.9	4.6	3.2	8.0	4.1	6.1	6.8	5.2	6.3	4.0	9.2	9.2	7.4	7.0	4.8
004_sugar_box	4.4	3.0	0.0	4.1	3.0	6.5	5.1	5.7	4.0	7.5	2.8	3.7	7.2	5.9	6.4	2.3	8.8	8.5	7.1	6.7	6.2
005_tomato_soup_can	1.2	4.8	4.1	0.0	4.4	7.5	6.7	7.2	3.8	9.2	2.8	5.3	7.6	5.1	8.5	3.2	10.4	10.2	8.9	8.6	6.4
006_mustard_bottle	5.0	5.3	3.0	4.4	0.0	8.1	6.7	7.3	5.2	7.8	3.0	1.5	7.9	6.8	7.6	3.1	9.2	8.8	7.7	7.5	7.7
007_tuna_fish_can	7.2	5.1	6.5	7.5	8.1	0.0	3.3	3.3	5.9	7.6	7.1	8.8	5.6	6.6	5.8	7.1	8.8	8.7	7.5	7.7	4.3
008_pudding_box	6.5	3.9	5.1	6.7	6.7	3.3	0.0	1.3	5.1	6.3	6.0	7.4	5.4	6.4	4.7	5.9	7.5	7.4	5.9	6.0	3.8
009_gelatin_box	7.1	4.6	5.7	7.2	7.3	3.3	1.3	0.0	5.7	6.2	6.6	7.9	5.3	6.8	4.4	6.6	7.3	7.2	5.7	6.0	3.7
010_potted_meat_can	3.6	3.2	4.0	3.8	5.2	5.9	5.1	5.7	0.0	8.9	3.6	6.1	6.9	3.8	7.6	3.5	9.9	10.3	8.1	7.7	4.6
011_banana	9.4	8.0	7.5	9.2	7.8	7.6	6.3	6.2	8.9	0.0	8.3	7.6	6.3	9.3	4.9	8.6	5.1	3.6	5.4	5.5	8.0
019_pitcher_base	3.3	4.1	2.8	2.8	3.0	7.1	6.0	6.6	3.6	8.3	0.0	3.9	7.3	5.1	7.2	2.4	9.4	9.3	7.5	7.1	6.3
021_bleach_cleanser	5.9	6.1	3.7	5.3	1.5	8.8	7.4	7.9	6.1	7.6	3.9	0.0	8.4	7.5	7.9	3.9	9.0	8.5	7.8	7.5	8.4
024_bowl	7.7	6.8	7.2	7.6	7.9	5.6	5.4	5.3	6.9	6.3	7.3	8.4	0.0	5.9	6.0	7.7	7.0	7.9	6.1	6.3	5.9
025_mug	4.9	5.2	5.9	5.1	6.8	6.6	6.4	6.8	3.8	9.3	5.1	7.5	5.9	0.0	8.0	5.8	9.9	10.7	8.0	7.6	5.3
035_power_drill	8.6	6.3	6.4	8.5	7.6	5.8	4.7	4.4	7.6	4.9	7.2	7.9	6.0	8.0	0.0	7.6	6.7	6.1	4.6	4.7	6.3
036_wood_block	3.6	4.0	2.3	3.2	3.1	7.1	5.9	6.6	3.5	8.6	2.4	3.9	7.7	5.8	7.6	0.0	9.7	9.6	8.1	7.8	6.2
037_scissors	10.6	9.2	8.8	10.4	9.2	8.8	7.5	7.3	9.9	5.1	9.4	9.0	7.0	9.9	6.7	9.7	0.0	4.4	4.7	5.0	8.7
040_large_marker	10.5	9.2	8.5	10.2	8.8	8.7	7.4	7.2	10.3	3.6	9.3	8.5	7.9	10.7	6.1	9.6	4.4	0.0	5.8	6.0	9.0
051_large_clamp	9.2	7.4	7.1	8.9	7.7	7.5	5.9	5.7	8.1	5.4	7.5	7.8	6.1	8.0	4.6	8.1	4.7	5.8	0.0	2.5	7.0
052_extra_large_clamp	8.9	7.0	6.7	8.6	7.5	7.7	6.0	6.0	7.7	5.5	7.1	7.5	6.3	7.6	4.7	7.8	5.0	6.0	2.5	0.0	7.0
061_foam_brick	6.2	4.8	6.2	6.4	7.7	4.3	3.8	3.7	4.6	8.0	6.3	8.4	5.9	5.3	6.3	6.2	8.7	9.0	7.0	7.0	0.0

Figure 5.4 Matrix of distances between embeddings of different objects rendered from different poses. Embeddings are obtained with the PointSDF encoder. Higher distances indicates that the clusters are more separated in the latent space.

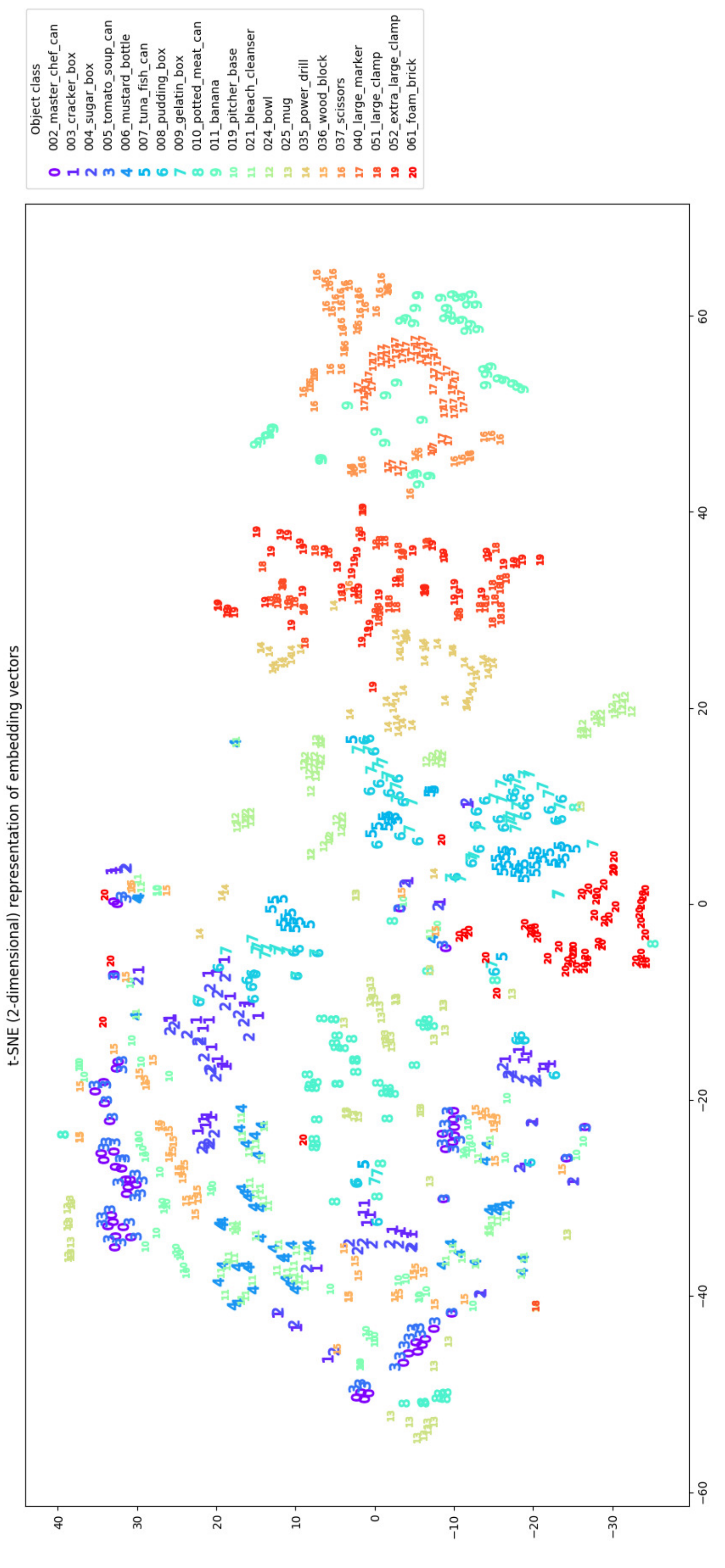


Figure 5.5 2D t-SNE visualization of embeddings of partial point clouds rendered from different YCB-Video objects. Embeddings are obtained with the PointSDF encoder.

In order to further verify that the clusterization of the embeddings does not happen for all the objects, we pick different embeddings of the same object and interpolate in the latent space, similarly at what is done in Zhao et al. [129]. Given two embeddings, we generate a 50-steps trajectory in the latent space and generate a completed shape at every step. Meshes are obtained by sampling the SDF in a voxel 3D space of edge 128, and then meshed through a marching cubes algorithm [59]. As an example, in Figure 5.6 we show this process for the objects *banana* and *mug*. According to the insight obtained in the previous phase (Figures 5.4 and 5.5), the former is well clustered and the latter is not, therefore the interpolation provides an interpretable shape in all the trajectory points only in the first case. While the banana in Figure 5.6 looks like a banana for the whole trajectory, the mug loses its concavity at around half the trajectory. Moreover, the latent representations for these two objects are quite far apart (Figure 5.4), therefore the fact that reconstructed shapes of the banana are very different from the ones of the mug is quite intuitive.

5.4 Remarks and further work

In this Section, we have given an overview of the shape completion problem and how it has been tackled, in recent times, with powerful deep autoencoder architectures. Although such architectures have appeared in the latest robotic manipulation research literature as part of an integrated grasp planning pipeline, we have shown that the idea of pairing shape completion with out of the box, state of the art, 6 DoF grasp planners is promising. We have also argued that the capability of an autoencoder to reconstruct training examples should not be the only driver when optimizing these model for real-world applications, and more attention should be put on the regularization of the latent space. Enabling these architectures to not only have good reconstruction accuracy but also an interpretable and exploitable latent space structure is an interesting research avenue. For instance, the optimization of the network weights can be steered towards a target distribution (e.g. a Gaussian multivariate) by inserting a Kullback-Leibler divergence term into the loss function alongside the reconstruction error. This idea has led to the usage of Variational Autoencoders (VAE), that are very popular in current times as generative models. To the best of our knowledge, this approach is already used to generate 3D shapes from random seeds, and some works propose a VAE as a grasp candidate proposal system [74], but very little yet exists towards the shape completion task.

The results showed in this Chapter are mere proof of concepts, and elicit further studies on the subject. Since the work by Varley et al. [116], shape completion has started to gain traction in the robotics community and, thanks to the performance of modern-day GPUaccelerated deep learning models, could in the next years become a common tool in any robotic manipulation pipeline just as much as RGB-D cameras did in the 2000s.

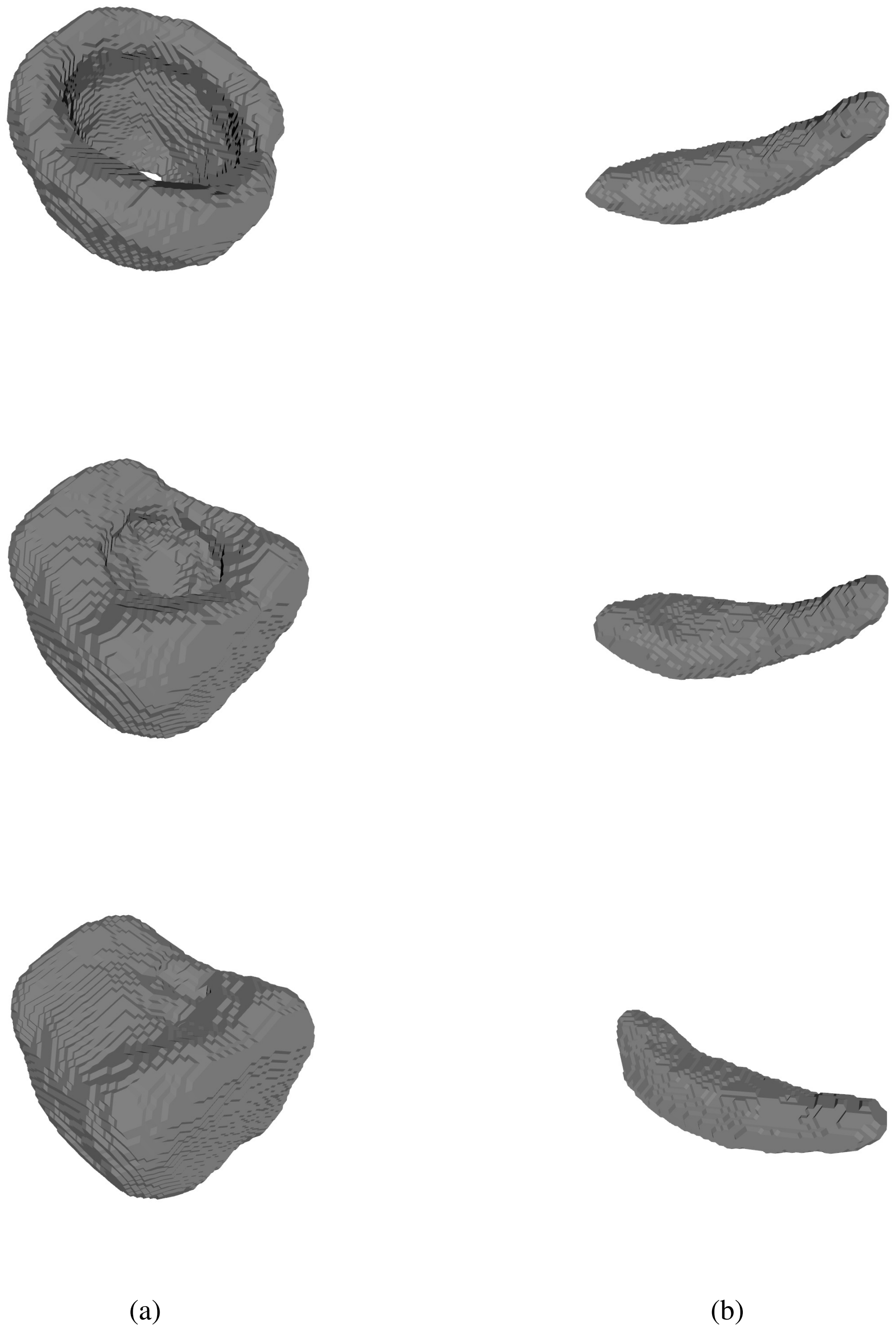


Figure 5.6 Shapes reconstructed from an interpolation in embedding space for the object *mug* (a) and *banana* (b). The top row represents the completion related to the 10-th embedding of the latent space trajectory, the middle row corresponds to the 25-th embedding and the bottom row corresponds to 40-th embedding.

Chapter 6

Conclusion

This Thesis addresses the problem of vision-based grasp planning, that is the detection of suitable end effector poses for a robot to reach for in order to pick up objects from the surrounding environment. This is an essential part of any robot task involving physical interaction and manipulation actions. In particular, the purpose of the Thesis work is twofold. On the one hand, we target the grasp planning problem in contexts where the robot system can only leverage partial views of the scene and has no a priori information about the 3D model of the object. On the other hand, we address an issue regarding existing and scientifically valid ways to effectively benchmark the performance of grasp planning approaches already present in the state of the art by proposing a benchmarking protocol ourselves.

In some high-volume, application-specific environments such as robotic assembly lines, the manipulation system (in terms of gripper, manipulator kinematic structure, and perception system) and the robot workspace are optimized with respect to the task at hand. In this case, the 3D geometry of the target objects are well known in advance and the grasping task simply resolves into a pose detection problem, since the grasp planning itself can be addressed with mature analytical techniques [7]. For appplications involving environments where it is not feasible to scan and store the 3D representation of every object, such as homes, stores or warehouses, this approach is not feasible and robots can usually only rely on partial 3D views of the scene. In this Thesis work, we distilled this into a target scenario where everyday objects are distributed on a horizontal surface with increasing levels of occlusion and clutter. In Chapter 2 we show how we attempted to tackle such scenario by fitting superquadric surfaces to partial object point clouds to approximate the general shape and graspable volume of the target. The main novelty of our method resides in the way the optimization problem is framed to constrain the orientation of the superquadric with respect to the horizontal surface. Moreover, we also proposed a simple method to generate candidates around the superquadric shape and rank them using a metric that accounts for kinematic feasibility of the grasp and the geometry of the superquadric with respect to the robot hand. This approach was deployed and tested on the iCub robot and its results compared with a similar method that also employs superquadrics.

The proposed solution performed well on the test scenario, and ended up being integrated in the iCub ecosystem as a standard routine for grasping. For its effectiveness and simplicity, it was also used as grasping action in the context of a Human-Robot Interaction framework. However, the method works well on the assumption that target objects are symmetric with respect to one or more planes. While the assumption holds for simple objects such as boxes, cans and bottles, the method is not expressive enough to reliably model asymmetric targets and generate good grasp candidates. Improvement directions for this kind of primitive-based method involve using a larger number of superquadrics to model the target object [115] or using superquadric functions with local deformations, or even better hyperquadrics [44], to tackle asymmetric targets with irregular shapes.

At the time of performing the experiments, the iCub software module ecosystem did not yet include solutions to perform object segmentation on cluttered surfaces and occluded objects. In Chapter 3 we have shown how we adapted a state of the art deep learning architecture for instance segmentation to solve this problem and allow us to tackle visually cluttered environments. However, for a number of grasping experiments we had to frequently switch the number and type of objects in our test set, and this required to retrain the network as fast as possible. To this end, we extended a fast dataset generation algorithm to automatically annotate images for instance segmentation tasks. We therefore obtained a versatile tool that helped us adapt the learned model to different object sets in the matter of hours, instead of days. Although this tool proved useful in various kinds of experiments, including serving as segmentation front end for a novel pose detection approach, it can be improved in terms of time and effort needed to train the model for a new task. In its nature, it is still a method that requires offline training, and while it might be good enough for manipulation experiments, it has limited use in the world of service robotics. In this sense, recent works in literature show that online and continuous learning techniques [65, 16] have a faster retraining wait time and can perform just as good as offline methods like ours, given that the robot is able to acquire annotated ground truth automatically, which is a challenge in itself. Such techniques would allow robots that work in human environments to quickly and naturally adapt to new object sets.

The desire to test our superquadric-based approach against other state of the art methods and compare results obtained on different robot setups lead us to realize how daunting this task really is, since the robotics manipulation community has never adopted common benchmarks and protocols for the kind of scenario targeted in this Thesis. Although some benchmarks exist for very specific tasks, and some object sets [14] saw a widespread adpotion, nothing specific to grasping on a cluttered tabletop surface existed, despite the large number of papers on the subject published in the last 5 years. With this goal in mind, we designed GRASPA (Chapter 4). GRASPA is a benchmarking protocol whose main contributions are the reproducibility of target scenes, a procedure to assess the good calibration and reachability of the robot over the scenes, a way to test planned grasps both in simulation and *on the metal*, and a number of granular metrics that attempt to fairly evaluate successes and failures. We showed how GRASPA may be used to measure the performance of different grasp planners can be assessed using different robot platforms, and this allowed us to make interesting observations about the failure cases of the approaches being tested. This does not mean, however, that GRASPA cannot be improved. For instance, some grasping approaches are designed to tackle bin picking and heap sorting scenarios, and the current GRASPA scenes do not provide enough of a challenge. Another part of GRASPA that has a lot of improvement headroom is the metric used to quantify the simulated grasp quality, that turned out to be unreliable for parallel grippers because of the limits of the simulation tool used. With the evolution of software physics simulation, it makes sense to foresee a grasping simulator that focuses on proper friction simulation [31] to replace the current implementation in GRASPA. Overall, the most important lesson learned with GRASPA is that benchmarks imply a delicate balancing act: on the one hand, they should be adaptable to every robot, every setup, every sensor and every gripper, in order to encompass every possible scenario; on the other hand, the larger the appeal, the higher the difficulty in making all scenarios comparable to each other. This also implies direct correlation between flexibility and the effort required of the user (in the form of writing code and adhering to the protocol). Therefore, we believe any improvement towards a version 2.0 of GRASPA would have to aim to hit a sweet spot between being useful enough for a large enough portion of the research community and avoiding the imposition of a cumbersome overhead to experiments.

Testing both our own superquadric modeling approach and model-free approaches taken from the state of the art (nominally, GPD [82] and Dex-Net [62]) on the same task helped us observe failure cases on both. Primitive-based modeling approaches trade off precise modeling and grasping for an hypothesis about the hidden parts of the object, while model-free approaches like GPD and Dex-Net can efficiently exploit local geometries for precise grasps, but they cannot generate candidates on the hidden part of objects. A question arises then: *is it possible to combine the pros of both, without the respective cons?* We believe it is, and the answer may lie in shape completion methods. Since these methods rely on training a deep autoencoder model to reconstruct complete shapes from partial readings, they can provide an explicit and interpretable model of the target object from single view observations based on previous experience. In the last Chapter (Chapter 5), we have provided a proof of concept by combining an existing shape completion approach with a state of the art grasp planner, with encouraging results. In the same Chapter we have also outlined the importance of the dimensionality reduction these models implicitly perform, and how such a feature can be exploited in order to augment the capabilities of grasp planners that make use of such shape completion approaches.

As a closing remark, and considering the exploratory journey this PhD Thesis turned out to be, we reiterate an observation made in the Introduction: despite some lines of thought might consider grasp planning a solved problem, we believe that is not the case when thinking forward to some of the realistic challenges autonomous robots might have to face. We also believe that in order to make significant advancements that adhere to the scientific principles, the manipulation research community should aim at producing usable benchmarks that foster reproducibility of experiments and fair, unbiased comparison of results even at the cost of some overhead in the experimental validation phase. Finally, we foresee that shape completion methods will, in the coming years, gain even more traction and become the go-to solution for bridging the gap between model-based and model-free approaches, by leveraging the performance of the former and the assumptions (or rather, lack thereof) of the latter.

References

[1] Abdulla, W. (2017). *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. Github. Publication Title: GitHub repository.

[2] Agarwal, S., Terrail, J. O. D., and Jurie, F. (2018). Recent advances in object detection in the age of deep convolutional neural networks. *arXiv preprint arXiv:1809.03193*.

[3] Bai, M. and Urtasun, R. (2017). Deep watershed transform for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5221–5229.

[4] Barr (1981). Superquadrics and Angle-Preserving Transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23. Conference Name: IEEE Computer Graphics and Applications.

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

[6] Bauer, D., Patten, T., and Vincze, M. (2020). VeREFINE: Integrating Object Pose Verification With Physics-Guided Iterative Refinement. *IEEE Robotics and Automation Letters*, 5(3):4289–4296. Conference Name: IEEE Robotics and Automation Letters.

[7] Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 348–353 vol.1. ISSN: 1050-4729.

[8] Biegelbauer, G. and Vincze, M. (2007). Efficient 3D Object Detection by Fitting Superquadrics to Range Image Data for Robot's Object Manipulation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1086–1091. ISSN: 1050-4729.

[9] Billard, A. and Kragic, D. (2019). Trends and challenges in robot manipulation. *Science*, 364(6446). Publisher: American Association for the Advancement of Science Section: Review.

[10] Bohg, J., Morales, A., Asfour, T., and Kragic, D. (2014). Data-Driven Grasp Synthesis-A Survey. *IEEE Transactions on Robotics*, 30(2):289–309.

[11] Borst, C., Fischer, M., and Hirzinger, G. (2004). Grasp planning: how to choose a suitable task wrench space. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, pages 319–325 Vol.1, New Orleans, LA, USA. IEEE.

[12] Bottarel, F., Vezzani, G., Pattacini, U., and Natale, L. (2020). GRASPA 1.0: GRASPA is a Robot Arm graSping Performance BenchmArk. *IEEE Robotics and Automation Letters*, 5(2):836–843. Conference Name: IEEE Robotics and Automation Letters.

[13] Caldera, S., Rassau, A., and Chai, D. (2018). Review of Deep Learning Methods in Robotic Grasp Detection. *Multimodal Technologies and Interaction*, 2(3):57. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[14] Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015a). The YCB object and Model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517.

[15] Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015b). Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set. *IEEE Robotics Automation Magazine*, 22(3):36–52.

[16] Ceola, F., Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2020). Fast Object Segmentation Learning with Kernel-based Methods for Robotics. *arXiv:2011.12805 [cs]*. arXiv: 2011.12805.

[17] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012 [cs]*. arXiv: 1512.03012.

[18] Chevalier, L., Jaillet, F., and Baskurt, A. (2003). Segmentation and superquadric modeling of 3D objects. Publisher: UNION Agency–Science Press.

[19] Cociaş, T. T., Grigorescu, S. M., and Moldoveanu, F. (2012). Multiple-superquadrics based object surface estimation for grasping in service robotics. In *2012 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, pages 1471–1477. ISSN: 1842-0133.

[20] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. (2018). Analysis and Observations From the First Amazon Picking Challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188. Conference Name: IEEE Transactions on Automation Science and Engineering.

[21] Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning.

[22] Dai, A., Qi, C. R., and Nießner, M. (2017). Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis. *arXiv:1612.00101 [cs]*. arXiv: 1612.00101.

[23] Dai, J., He, K., and Sun, J. (2016). Instance-aware semantic segmentation via multi-task network cascades. pages 3150–3158.

[24] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. ISSN: 1063-6919.

[25] Deng, X., Mousavian, A., Xiang, Y., Xia, F., Bretl, T., and Fox, D. (2019). Poserbpf: A rao-blackwellized particle filter for 6d object pose tracking. *arXiv preprint arXiv:1905.09304*.

[26] Diankov, R. and Kuffner, J. (2008). Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79.

[27] Du, G., Wang, K., Lian, S., and Zhao, K. (2020). Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. *Artificial Intelligence Review*.

[28] Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[29] Duncan, K., Sarkar, S., Alqasemi, R., and Dubey, R. (2013). Multi-scale superquadric fitting for efficient shape and pose recovery of unknown objects. pages 4238–4243. IEEE.

[30] Dwibedi, D., Misra, I., and Hebert, M. (2017). Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1310–1319, Venice. IEEE.

[31] Eppner, C., Mousavian, A., and Fox, D. (2020). ACRONYM: A Large-Scale Grasp Dataset Based on Simulation. *arXiv:2011.09584 [cs]*. arXiv: 2011.09584.

[32] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. volume 96, pages 226–231. Issue: 34.

[33] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338. Publisher: Springer.

[34] Fanello, S. R., Pattacini, U., Gori, I., Tikhanoff, V., Randazzo, M., Roncone, A., Odone, F., and Metta, G. (2014). 3D stereo estimation and fully automated learning of eye-hand coordination in humanoid robots. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 1028–1035. ISSN: 2164-0580.

[35] Fang, H.-S., Wang, C., Gou, M., and Lu, C. (2020). GraspNet-1Billion: A Large-Scale Benchmark for General Object Grasping. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11441–11450, Seattle, WA, USA. IEEE.

[36] Gaidon, A., Wang, Q., Cabon, Y., and Vig, E. (2016). Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4340–4349.

[37] Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *International Conference on Machine Learning*, pages 1050–1059. PMLR. ISSN: 1938-7228.

[38] Gandler, G. Z., Ek, C. H., Björkman, M., Stolkin, R., and Bekiroglu, Y. (2020). Object shape estimation and modeling, based on sparse Gaussian process implicit surfaces, combining visual data and tactile exploration. *Robotics and Autonomous Systems*, 126:103433. Publisher: Elsevier.

[39] Gao, W. and Tedrake, R. (2019). kPAM-SC: Generalizable Manipulation Planning using KeyPoint Affordance and Shape Completion. *arXiv:1909.06980 [cs]*. arXiv: 1909.06980.

[40] Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., Martinez-Gonzalez, P., and Garcia-Rodriguez, J. (2018). A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70:41–65.

[41] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., and Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292.

[42] Goldfeder, C., Allen, P. K., Lackner, C., and Pelossof, R. (2007). Grasp Planning via Decomposition Trees. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4679–4684. ISSN: 1050-4729.

[43] Handa, A., Pătrăucean, V., Stent, S., and Cipolla, R. (2016). Scenenet: An annotated model generator for indoor scene understanding. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5737–5743. IEEE.

[44] Hanson, A. J. (1988). Hyperquadrics: Smoothly deformable shapes with convex polyhedral bounds. *Computer Vision, Graphics, and Image Processing*, 44(2):191–210.

[45] He, K., Gkioxari, G., Dollar, P., and Girshick, R. (2017). Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Venice. IEEE.

[46] Jaklič, A., Leonardis, A., and Solina, F. (2000). *Segmentation and Recovery of Superquadrics*, volume 20 of *Computational Imaging and Vision*. Springer Netherlands, Dordrecht.

[47] Kasper, A., Xue, Z., and Dillmann, R. (2012). The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934. Publisher: SAGE Publications Ltd STM.

[48] Kim, J., Iwamoto, K., Kuffner, J. J., Ota, Y., and Pollard, N. S. (2013). Physically Based Grasp Quality Evaluation Under Pose Uncertainty. *IEEE Transactions on Robotics*, 29(6):1424–1439.

[49] Kootstra, G., Popović, M., Jørgensen, J. A., Kragic, D., Petersen, H. G., and Krüger, N. (2012). VisGraB: A Benchmark for Vision-Based Grasping. *Paladyn, Journal of Behavioral Robotics*, 3(2).

[50] Kroemer, O., Niekum, S., and Konidaris, G. (2020). A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms. *arXiv:1907.03146 [cs]*. arXiv: 1907.03146.

[51] Lateef, F. and Ruichek, Y. (2019). Survey on semantic segmentation using deep learning techniques. *Neurocomputing*, 338:321–348.

[52] Leitner, J., Tow, A. W., Sünderhauf, N., Dean, J. E., Durham, J. W., Cooper, M., Eich, M., Lehnert, C., Mangels, R., McCool, C., Kujala, P. T., Nicholson, L., Pham, T., Sergeant, J., Wu, L., Zhang, F., Upcroft, B., and Corke, P. (2017). The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4705–4712.

[53] Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. (2018). PointCNN: Convolution on xi-transformed points. pages 828–838.

[54] Li, Y., Qi, H., Dai, J., Ji, X., and Wei, Y. (2017). Fully convolutional instance-aware semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2359–2367.

[55] Lin, H. (2020). Robotic Manipulation Based on 3D Vision: A Survey. pages 1–5.

[56] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125.

[57] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

[58] Ling, H., Gao, J., Kar, A., Chen, W., and Fidler, S. (2019). Fast interactive object annotation with curve-gcn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5257–5266.

[59] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169. Publisher: ACM New York, NY, USA.

[60] Lundell, J., Verdoja, F., and Kyrki, V. (2019). Robust Grasp Planning Over Uncertain Shape Completions. pages 1526–1532. IEEE.

[61] Lundell, J., Verdoja, F., and Kyrki, V. (2020). Beyond top-grasps through scene completion. pages 545–551. IEEE.

[62] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Aparicio, J., and Goldberg, K. (2017). Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. Robotics: Science and Systems Foundation.

[63] Mahler, J., Matl, M., Liu, X., Li, A., Gealy, D., and Goldberg, K. (2018). Dex-Net 3.0: Computing Robust Vacuum Suction Grasp Targets in Point Clouds Using a New Analytic Model and Deep Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5620–5627. ISSN: 2577-087X.

[64] Mahler, J., Pokorny, F. T., Hou, B., Roderick, M., Laskey, M., Aubry, M., Kohlhoff, K., Kroger, T., Kuffner, J., and Goldberg, K. (2016). Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards. pages 1957–1964. IEEE.

[65] Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2018). Speeding-up object detection training for robotics with falkon. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5770–5776. IEEE.

[66] Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2020). On-line object detection: a robotics challenge. *Autonomous Robots*, 44(5):739–757. Publisher: Springer.

[67] Makhal, A., Thomas, F., and Gracia, A. P. (2017). Grasping Unknown Objects in Clutter by Superquadric Representation. *arXiv:1710.02121 [cs]*. arXiv: 1710.02121.

[68] Matheus, K. and Dollar, A. M. (2010). Benchmarking grasping and manipulation: Properties of the Objects of Daily Living. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5020–5027.

[69] Maturana, D. and Scherer, S. (2015). VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Hamburg, Germany. IEEE.

[70] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy Networks: Learning 3D Reconstruction in Function Space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4455–4465, Long Beach, CA, USA. IEEE.

[71] Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. (2010). The iCub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125–1134.

[72] Miller, A. T. and Allen, P. K. (2004). Graspit! A versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4):110–122. Conference Name: IEEE Robotics Automation Magazine.

[73] Morrison, D., Corke, P., and Leitner, J. (2020). EGAD! An Evolved Grasping Analysis Dataset for Diversity and Reproducibility in Robotic Manipulation. *IEEE Robotics and Automation Letters*, 5(3):4368–4375. Conference Name: IEEE Robotics and Automation Letters.

[74] Mousavian, A., Eppner, C., and Fox, D. (2019). 6-DOF GraspNet: Variational Grasp Generation for Object Manipulation. pages 2901–2910.

[75] Movshovitz-Attias, Y., Kanade, T., and Sheikh, Y. (2016). How useful is photo-realistic rendering for visual learning? In *European Conference on Computer Vision*, pages 202–217. Springer.

[76] Murali, A., Mousavian, A., Eppner, C., Paxton, C., and Fox, D. (2020). 6-DOF Grasping for Target-driven Object Manipulation in Clutter. *arXiv:1912.03628 [cs]*. arXiv: 1912.03628.

[77] Murray, R. M., Li, Z., and Sastry, S. S. (2017). *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1 edition.

[78] Nguyen, P. D., Bottarel, F., Pattacini, U., Hoffmann, M., Natale, L., and Metta, G. (2018). Merging Physical and Social Interaction for Effective Human-Robot Collaboration. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9, Beijing, China. IEEE.

[79] Ojala, T., Pietikinen, M., and Harwood, D. (1996). A comparative study of texture measures with classification with local binary patterns. *Pattern Recognition*, 29(1):51–59.

[80] Park, D. and Ramanan, D. (2015). Articulated pose estimation with tiny synthetic videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 58–66.

[81] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, Long Beach, CA, USA. IEEE.

[82] Pas, A. t., Gualtieri, M., Saenko, K., and Platt, R. (2017). Grasp Pose Detection in Point Clouds. *arXiv:1706.09911 [cs]*. arXiv: 1706.09911.

[83] Paschalidou, D., Ulusoy, A. O., and Geiger, A. (2019). Superquadrics Revisited: Learning 3D Shape Parsing beyond Cuboids. *arXiv:1904.09970 [cs]*. arXiv: 1904.09970.

[84] Pasquale, G., Ciliberto, C., Odone, F., Rosasco, L., and Natale, L. (2019). Are we done with object recognition? The iCub robot's perspective. *Robotics and Autonomous Systems*, 112:260–281. Publisher: Elsevier.

[85] Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. pages 1668–1674. IEEE.

[86] Pattacini, U. and Piga, N. (2021). robotology/icub-gazebo-grasping-sandbox: Release v1.3.6.

[87] Peng, X., Sun, B., Ali, K., and Saenko, K. (2015). Learning deep object detectors from 3d models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286.

[88] Piga, N. A., Bottarel, F., Fantacci, C., Vezzani, G., Pattacini, U., and Natale, L. (2021). MaskUKF: an Instance Segmentation Aided Unscented Kalman Filter for 6D Object Pose and Velocity Tracking. *Frontiers in Robotics and AI*, 8. Publisher: Frontiers.

[89] Pinheiro, P. O., Collobert, R., and Dollar, P. (2015). Learning to Segment Object Candidates. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1990–1998. Curran Associates, Inc.

[90] Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pages 313–318.

[91] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. pages 652–660.

[92] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.

[93] Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149.

[94] Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer.

[95] Roa, M. A. and Suárez, R. (2015). Grasp quality measures: review and performance. *Autonomous Robots*, 38(1):65–88.

[96] Roncone, A., Pattacini, U., Metta, G., and Natale, L. (2016). A Cartesian 6-DoF Gaze Controller for Humanoid Robots. In *Robotics: Science and Systems XII*. Robotics: Science and Systems Foundation.

[97] Rubert, C., León, B., Morales, A., and Sancho-Bru, J. (2018). Characterisation of Grasp Quality Metrics. *Journal of Intelligent & Robotic Systems*, 89(3):319–342.

[98] Rudi, A., Carratino, L., and Rosasco, L. (2017). FALKON: an optimal large scale kernel method. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 3891–3901, Red Hook, NY, USA. Curran Associates Inc.

[99] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

[100] Sahbani, A., El-Khoury, S., and Bidaud, P. (2012). An overview of 3D object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336.

[101] Schwarz, M. and Behnke, S. (2020). Stillleben: Realistic Scene Synthesis for Deep Learning in Robotics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10502–10508. ISSN: 2577-087X.

[102] Shao, L., Ferreira, F., Jorda, M., Nambiar, V., Luo, J., Solowjow, E., Ojea, J. A., Khatib, O., and Bohg, J. (2019). UniGrasp: Learning a Unified Model to Grasp with N-Fingered Robotic Hands. *arXiv:1910.10900 [cs]*. arXiv: 1910.10900.

[103] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Springer Science & Business Media.

[104] Singh, A., Sha, J., Narayan, K. S., Achim, T., and Abbeel, P. (2014). BigBIRD: A large-scale 3D database of object instances. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 509–516, Hong Kong, China. IEEE.

[105] Solina, F. and Bajcsy, R. (1990). Recovery of parametric models from range images: the case for superquadrics with global deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):131–147. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[106] Strand, M., Xue, Z., Zoellner, M., and Dillmann, R. (2010). Using superquadrics for the approximation of objects and its application to grasping. In *The 2010 IEEE International Conference on Information and Automation*, pages 48–53.

[107] Stuckler, J., Holz, D., and Behnke, S. (2012). RoboCup@Home: Demonstrating Everyday Manipulation Skills in RoboCup@Home. *IEEE Robotics Automation Magazine*, 19(2):34–42.

[108] Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.-H., and Kautz, J. (2018). Splatnet: Sparse lattice networks for point cloud processing. pages 2530–2539.

[109] Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., and Milford, M. (2018). The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420. Publisher: SAGE Publications Sage UK: London, England.

[110] Thalhammer, S., Leitner, M., Patten, T., and Vincze, M. (2020). PyraPose: Feature Pyramids for Fast and Accurate Object Pose Estimation under Domain Shift. *arXiv:2010.16117 [cs]*. arXiv: 2010.16117.

[111] Triantafyllou, P., Mnyusiwalla, H., Sotiropoulos, P., Roa, M. A., Russell, D., and Deacon, G. (2019). A Benchmarking Framework for Systematic Evaluation of Robotic Pick-and-Place Systems in an Industrial Grocery Setting. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6692–6698. ISSN: 2577-087X, 1050-4729.

[112] Vahrenkamp, N., Kröhnert, M., Ulbrich, S., Asfour, T., Metta, G., Dillmann, R., and Sandini, G. (2013). Simox: A Robotics Toolbox for Simulation, Motion and Grasp Planning. In Lee, S., Cho, H., Yoon, K.-J., and Lee, J., editors, *Intelligent Autonomous Systems 12*, volume 193, pages 585–594. Springer Berlin Heidelberg, Berlin, Heidelberg.

[113] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

[114] Van der Merwe, M., Lu, Q., Sundaralingam, B., Matak, M., and Hermans, T. (2020). Learning Continuous 3D Reconstructions for Geometrically Aware Grasping. *arXiv:1910.00983 [cs]*. arXiv: 1910.00983.

[115] Varadarajan, K. M. and Vincze, M. (2011). Object part segmentation and classification in range images for grasping. In *2011 15th International Conference on Advanced Robotics (ICAR)*, pages 21–27.

[116] Varley, J., DeChant, C., Richardson, A., Ruales, J., and Allen, P. (2017). Shape completion enabled robotic grasping. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2442–2447. IEEE.

[117] Varley, J., Weisz, J., Weiss, J., and Allen, P. (2015). Generating multi-fingered robotic grasps via deep learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4415–4420.

[118] Vezzani, G. (2019). *Sense, Think, Grasp: A study on visual and tactile information processing for autonomous manipulation*. Dissertation, Universita di Genova, Genova.

[119] Vezzani, G. and Natale, L. (2017). Real-time Pipeline for Object Modeling and Grasping Pose Selection via Superquadric Functions. *Frontiers in Robotics and AI*, 4.

[120] Vezzani, G., Pattacini, U., and Natale, L. (2017). A grasping approach based on superquadric models. pages 1579–1586. IEEE.

[121] Wan, E. A., Van Der Merwe, R., and Haykin, S. (2001). The unscented Kalman filter. *Kalman filtering and neural networks*, 5(2007):221–280. Publisher: Wiley Online Library.

[122] Wang, S., Wu, J., Sun, X., Yuan, W., Freeman, W. T., Tenenbaum, J. B., and Adelson, E. H. (2018). 3d shape perception from monocular vision, touch, and shape priors. pages 1606–1613. IEEE.

[123] Watkins-Valls, D., Varley, J., and Allen, P. (2019). Multi-Modal Geometric Learning for Grasping and Manipulation. *arXiv:1803.07671 [cs]*. arXiv: 1803.07671.

[124] Wolf, S., Schott, L., Kothe, U., and Hamprecht, F. (2017). Learned Watershed: End-To-End Learning of Seeded Segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[125] Wu, W., Qi, Z., and Fuxin, L. (2019a). Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630.

[126] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. (2019b). *Detectron2*.

[127] Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57. Publisher: Springer.

[128] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2017). PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *arXiv:1711.00199 [cs]*. arXiv: 1711.00199.

[129] Zhao, S., Gu, C., Lu, C., Huang, Y., Wu, K., and Guan, X. (2019). PointDoN: A Shape Pattern Aggregation Module for Deep Learning on Point Cloud. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. ISSN: 2161-4407.