UNIVERSITÀ DEGLI STUDI
DI GENOVA

ISTITUTO ITALIANO
DI TECNOLOGIA

UNIVERSITY OF GENOVA

DOCTORAL THESIS

Sense, Think, Grasp: A study on visual and tactile information processing for autonomous manipulation

---

Author:
Giulia Vezzani

Supervisors:
Dr. Lorenzo Natale
Dr. Ugo Pattacini

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Humanoid Sensing and Perception - iCub Facility
Istituto Italiano di Tecnologia (IIT)

March 7, 2019

Declaration of Authorship

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.

Giulia Vezzani
March 7, 2019

Abstract

Giulia Vezzani

Sense, Think, Grasp:
A study on visual and tactile information processing for autonomous manipulation

Interacting with the environment using hands is one of the distinctive abilities of humans with respect to other species. This aptitude reflects on the crucial role played by objects' manipulation in the world that we have shaped for us. With a view of bringing robots outside industries for supporting people during everyday life, the ability of manipulating objects *autonomously* and in *unstructured environments* is therefore one of the basic skills they need. Autonomous manipulation is characterized by great complexity especially regarding the processing of sensors information to perceive the surrounding environment. Humans rely on vision for wide-ranging tridimensional information, prioprioception for the awareness of the relative position of their own body in the space and the sense of touch for local information when physical interaction with objects happens. The study of autonomous manipulation in robotics aims at transferring similar perceptive skills to robots so that, combined with state of the art control techniques, they could be able to achieve similar performance in manipulating objects. The great complexity of this task makes autonomous manipulation one of the open problems in robotics that has been drawing increasingly the research attention in the latest years.

In this work of Thesis, we propose possible solutions to some key components of autonomous manipulation, focusing in particular on the perception problem and testing the developed approaches on the humanoid

robotic platform iCub. When available, vision is the first source of information to be processed for inferring how to interact with objects. The *object modeling and grasping pipeline* based on superquadric functions we designed meets this need, since it reconstructs the object 3D model from partial point cloud and computes a suitable hand pose for grasping the object. Retrieving objects information with touch sensors only is a relevant skill that becomes crucial when vision is occluded, as happens for instance during physical interaction with the object. We addressed this problem with the design of a novel *tactile localization algorithm*, named *Memory Unscented Particle Filter*, capable of localizing and recognizing objects relying solely on 3D contact points collected on the object surface. Another key point of autonomous manipulation we report on in this Thesis work is bi-manual coordination. The execution of more advanced manipulation tasks in fact might require the use and coordination of two arms. Tool usage for instance often requires a proper in-hand object pose that can be obtained via dual-arm re-grasping. In pick-and-place tasks sometimes the initial and target position of the object do not belong to the same arm workspace, then requiring to use one hand for lifting the object and the other for locating it in the new position. At this regard, we implemented a *pipeline for executing the handover task*, i.e. the sequences of actions for autonomously passing an object from one robot hand on to the other.

The contributions described thus far address specific subproblems of the more complex task of autonomous manipulation. This actually differs from what humans do, in that humans develop their manipulation skills by learning through experience and trial-and-error strategy. A proper mathematical formulation for encoding this learning approach is given by Deep Reinforcement Learning, that has recently proved to be successful in many robotics applications. For this reason, in this Thesis we report also on the six month experience carried out at Berkeley Artificial Intelligence Research laboratory with the goal of studying *Deep Reinforcement Learning and its application to autonomous manipulation*.

Publications

This work has been carried out during my Ph.D. course in *Advanced and Humanoid Robotics* from November 2015 to November 2018. This three-year project resulted in the following publications (at the time of writing):

- G. Vezzani, N. Jamali, U. Pattacini, G. Battistelli, L. Chisci, and L. Natale, "A novel Bayesian filtering approach to tactile object recognition," *IEEE International Conference on Humanoid Robots*, pp. 256 - 263, 2016, Cancun.
- G. Vezzani, U. Pattacini, G. Battistelli, L. Chisci, and L. Natale, "Memory Unscented Particle Filter for 6-DOF tactile localization," *IEEE Transaction on Robotics*, 33 (5), 1139 - 1155, 2017.
- G. Vezzani, U. Pattacini, and L. Natale, "A Grasping approach based on superquadric models," *IEEE International Conference on Robotics and Automation*, pp. 1579 - 1586, 2017, Singapore.
- G. Vezzani, U. Pattacini, M.Regoli and L. Natale, "A novel pipeline for bi-manual handover task," *Advanced Robotics*, 31 (23-24), 1267 - 1280, 2017.
- G. Vezzani, and L. Natale, "Real-time pipeline for object modeling and grasping pose selection via superquadric functions," *Frontiers in Robotics and AI*, 4, 59, 2017.
- G. Vezzani, U. Pattacini and L. Natale "Improving superquadric modeling and grasping with prior on object shapes", *IEEE Internationl Conference on Robotics and Automation (ICRA)*, pp. 6875 - 6882, 2018, Brisbane.
- C. Fantacci, G. Vezzani, U. Pattacini, V. Tikhanoff and L. Natale "Markerless visual servoing on unknown objects for humanoid robot platforms", *IEEE Internationl Conference on Robotics and Automation (ICRA)*, pp. 3099 - 3106, 2018, Brisbane.

- A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations", *Robotics: Science and Systems (RSS)*, 2018, Pittsburgh.

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

Acknowledgements

First and foremost, I would like to thank Dr. Lorenzo Natale, for giving me this opportunity. I still remember my excitement and happiness when I was accepted for starting my PhD in humanoid robotics. Together with guiding me during my research, Lorenzo has also given me the freedom to try my own ideas and improve my knowledge and skills with wonderful experiences during the past three years.

A special thanks goes to Dr. Ugo Pattacini, that has been and still is a mentor to me. During the last three years, he has taught me with patience and constancy not only countless technical skills but also the proper attitude for working efficiently and effectively.

I am grateful to Prof. Pieter Abbeel, for giving me the opportunity to visit the Berkeley Artificial Intelligence Research lab at UC Berkeley.

I also would like to thank the friends who have always been there and those I met during my PhD. Thank you all for always being close and present in my life, regardless of the distance and the PhD busy schedule. In particular to my ex- and current lab mates: thanks for sharing the fun, anxieties and abroad experiences that made the PhD a wonderful time.

Last, but not least, many thanks go to my family and Claudio, for always believing in me and supporting all my choices in the first lines.

Contents

List of Figures

List of Tables

Preface

"Man grew the most intelligent among animals because of having his hands" is what the philosopher Anaxagoras[1] said. Nearly one century later, Aristotle argued against this claim: *"It is more reasonable to say that man received his hands because he is the most intelligent [...]. Giving a flute to someone who can play it is a better plan than teaching someone which already has a flute how to play it. Considering that nature always acts in the best way, we must conclude that man does not own his intelligence to his hands, but his hands to his intelligence."* [1]

According to Aristotle then, we did not choose the best path towards autonomous robotic manipulation. We equipped robots with hands they did not know how to properly use and we are now struggling to teach them.

Maybe this is the reason why robotic manipulation is such a hard problem to solve...

(Actually, Anaxagoras' intuition has been later on confirmed by several findings of paleoanthropologists, showing that the mechanical dexterity of the human hand has been a major factor in allowing homo sapiens to develop a superior brain (a similar role played by the anatomical structure of the human larynx in relation with speech capabilities has been also recognized). We therefore have hope of solving autonomous robotic manipulation and this thesis makes sense (fortunately)).

[1] (500?–428 BC).

Part I

Introduction

Chapter 1

The importance of perception and autonomous manipulation

"Could you please give me the big bottle of oil?"

This is a task that every human more than five years old can successfully accomplish just taking a little care. Making a robot able to autonomously perform the same actions would require some of the most advanced techniques available in robotics and, most likely, the final outcome would be strongly customized for solving this specific task and would fail if objects positions or vision condition change. This is just one example of the huge gap that still exists between humans' and robots' skills in autonomous manipulation.

Using hands for interacting with the environment has been one of the key features of human evolution. The ability of operating tools allowed us to shape the world into a place where the chances of surviving were higher at the beginning of our history, and then created opportunity to further

improve the quality of our lives with the emergence of technologies and arts.

Aside from its great benefits and power, autonomous manipulation is characterized by great complexity. Together with advanced motor and coordination skills, humans strongly rely on their perception system while interacting with objects: vision provides wide-ranging tridimensional information about the surrounding environment; prioprioception makes aware of relative position of one's own body in the space and the sense of touch gives local information when physical interaction with objects happens. Even if in the lack of one of the mentioned perception skills the other two can compensate, only the combination of all of them leads to the best performance. When trying to grasp an object in the darkness for example, a rough knowledge of the arm and hand positions is given by proprioception and touch helps when first contacts with the objects are detected. However, it might happen that we hit the object and make it drop while blindly looking for it. Equally, striking a matchstick without the sense of touch is almost impossible and it requires several trials to learn how to accomplish the task without the tactile information[1].

Autonomous robotic manipulation shows the same complexity as the human counterpart but no equal dexterous and powerful skills have been reached yet, making it one of the most challenging open problem in nowadays robotic research. While state of the art control techniques allow robots to follow complex trajectories with high speed and precision, equally efficient and reliable methods for sensors information processing and decision making are still missing. In particular, the ability of modeling the surrounding environment and autonomously inferring how to interact with objects do represent the core skills that mark the boundary between *preprogrammed* and *autonomous* manipulation.

This work of Thesis addresses some of the key problems of autonomous manipulation such as *object modeling* and *localization*, *grasping* and *bi-manual coordination*. The vision and motivation that accompanied the Ph.D. research activity are shown in Section 1.1. Then, Section 1.2 summarizes the main contributions and outlines the Thesis structure.

[1] This experiment was conducted by Prof. Roland Johansson at the University of Umeå, Sweden. More information are available here: http://roboticmaterials.com/rm/why-making-robotic-manipulation-harder-than-it-is/.

1.1 Vision and motivation

Industrial manipulators are already in use in factories for many years by now. They can achieve great performance in terms of speed, precision and reliability, because advanced control techniques are flanked with very *accurate knowledge of the environment and the objects to manipulate* (Fig. 1.1). Such an information level is possible since industrial manipulators operate

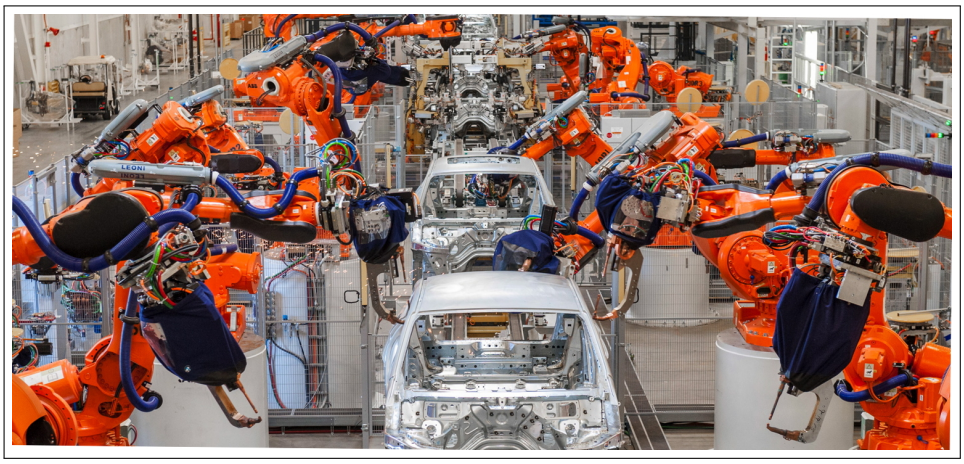


FIGURE 1.1: Industrial manipulators for car assembly.

in strongly structured environments. Though very effective, this restricts the operational domain of the such robots and requires engineering process of the environment. By removing such a constraint, production chains would be built more easily, but also, and more importantly, robots could be brought outside industries for supporting people in their everyday life. Robots ability to manipulate objects *autonomously* and in *unstructured environment* would be in fact crucial in several applications, such as cargo handling, people and object recovery after disasters, elder people assistance (Fig 1.2). Moving towards this new robots generation requires the design of novel hardware and cognitive skills. Taking inspiration from humans[2], the required features are the following:

1. A motor system reproducing the human upper body, in particular including two arms and hands with sufficient degrees of freedom (DOFs) and dexterous workspaces.

[2]Human-inspired robots are not necessary the optimal choice but taking inspiration from humans seems reasonable given their impressive dexterity in manipulation. However, a proper discussion about whether human-inspired robots are the best choice for autonomous manipulation does not fall within the scope of this work.

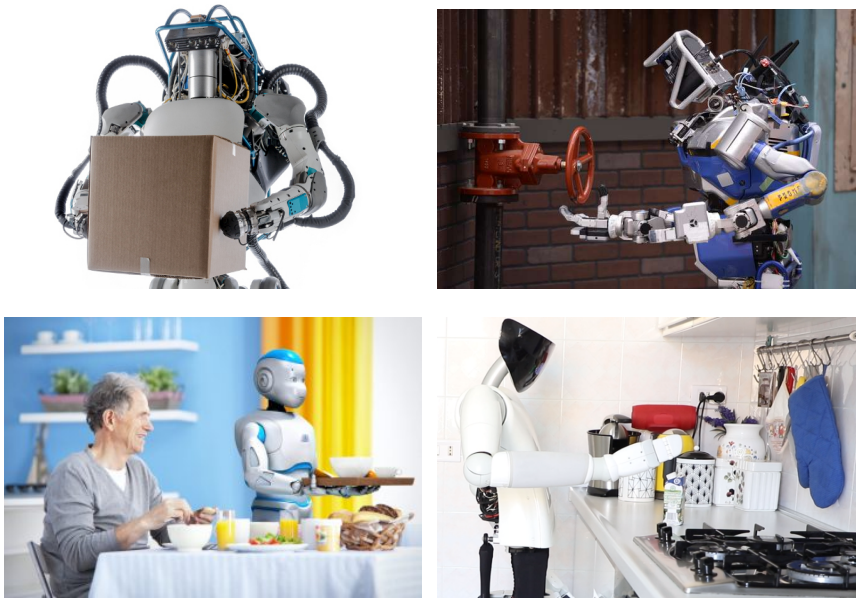


FIGURE 1.2: How autonomous robots could have beneficial effects on the society: some examples.

2. A proper sensor system including: cameras for acquiring images and depth information, tactile sensors on the robot hands and encoders for prioprioception.

3. A control system of the upper body for reaching desired poses in a reliable, safe and accurate way.

4. The ability of processing perceptive information generated by vision and tactile sensors.

5. Motor coordination with respect to the environment and the objects to manipulate and between the robot arms themselves.

The hardware design of several recent humanoid robots, such as the iCub [2] - the platform used for testing all the contributions presented in this Thesis work -, presents good, even still upgradeable, solutions for the fulfillment of the first three requirements in the aforementioned list. For this reason, the attention of this research activity focuses on the other two points, i.e. on the processing of the perceptive information and planning of proper movements for accomplishing manipulation tasks. The techniques derived from this research are still general and can be applied to any other humanoid robots satisfying points 1., 2. and 3. .

1.2 Contribution and outline

During the Ph.D. activity, our research focused on the following problems:

- **Object localization**: the problem of estimating the pose of a known object, using the information acquired from the robot perceptive system (vision and/or tactile sensors).
- **Bi-manual coordination**: the planning of proper actions to be executed with two arms for the accomplishment of a unique task.
- **Object modeling**: the reconstruction of an efficient mathematical representation of an unknown objects in terms of their shape, dimensions and pose, by using information provided by the robot sensors.
- **Object grasping**: the design of a suitable pose of the robot hand with respect to the object in order to enable grasps characterized by given properties (e.g. robustness, precision etc).

For each problem, we proposed specific solutions that together provide the main contributions of this Thesis:

- A *localization algorithm*, named *Memory Unscented Particle Filter*, capable of estimating the object poses using only 3D points collected on the object surface.
- A complete pipeline for the execution of *object bi-manual handovers* with the robot iCub. In other words, the robot is asked to pass one object from his hand on to the other.
- An *object modeling and grasping pose computation approach based on superquadric function*, that uses vision information to reconstruct the object model and compute a robot hand pose for grasping the object.

The solutions described thus far address specific key-parts of the general manipulation task. In particular, the solution of a complex and general problem such as manipulation is turned into the solution of many simpler sub-problems. This is the standard way to go in robotics, but it actually differs from what humans do. Humans develop their manipulation skills by *learning* them through experience and trial-and-error strategy. A mathematical formulation that encodes this approach is Reinforcement

Learning (RL) [3]. While previously applied mostly to game playing, recent works [4, 5, 6] show successful applications of Deep RL[3] to robotic manipulation. For this reason, six months of the Ph.D. activity presented in this work took place at Berkeley Artificial Intelligence Research laboratory at UC Berkeley with the main goal of studying Deep Reinforcement Learning and its application to autonomous manipulation.

This Thesis is organized as follows. A review of the state of the art regarding manipulation and more in particular the four problems we tackled is provided in Chapter 2, followed by a brief description of the iCub humanoid robot in Chapter 3. Chapters 4 and 5 respectively describe the Memory Unscented Particle Filter, including the mathematical derivation and experimental evaluation, and its application on a challenging tactile recognition task. In Chapter 6, we describe the entire pipeline implemented on the iCub robot for the execution of bi-manual handover tasks. Chapter 7 and 8 reports the proposed object modeling and grasping pose computation respectively for simple and more complex kinds of objects. In Chapter 9, the activity carried out while studying Deep Reinforcement Learning is detailed. Finally, Chapter 10 ends this Thesis work with concluding remarks and more general discussion.

[3]Deep Reinforcement Learning is the combination of Reinforcement Learning together with deep networks. Deep networks are extremely helpful when, for example, the algorithm is asked to learn from raw inputs, such as vision, without any hand-crafted features or domain heuristics.

Chapter 2

What is the stage of autonomous manipulation?

Autonomous manipulation requires the solution of different subproblems, through the combination of techniques belonging to various areas of robotics and the fusion of information provided by different sensors. Considering the intrinsic multidisciplinarity nature of the problem at hand so as the contributions of this Thesis work, we analyze the current stage of autonomous manipulation by focusing on the following topics: *tactile object localization and recognition* (Section 2.1), *bi-manual coordination* (Section 2.2), *object modeling from vision* (Section 2.3) and *autonomous grasping* (Section 2.4).

We complete the overview of the state of the art in autonomous manipulation by reporting on the recent progress of *Deep Reinforcement Learning* in dexterous manipulation and the related challenges (Section 2.5).

2.1 Tactile object localization and recognition

The interest in tactile sensing and perception has considerably increased over the last decade [7], often flanking or even replacing vision information during object manipulation, localization and recognition [8, 9, 10, 11, 12]. This extensive usage of the sense of touch in robotics is put forward in several findings in human physiology that testify how humans jointly exploit vision and touch in order to accomplish manipulation tasks and how humans are even able to explore objects by means of tactile perception solely [13, 14]. Certainly, the surge of interest on this topic is also encouraged by the recent advances in tactile technology [15, 16, 17, 18, 19, 20]

that have made it possible to build tactile systems that are reliable enough and can be deployed on real robots at a reasonable price [21, 22, 23]. Tactile sensors can be categorized in multiple manners, such as according to their sensing principles, fabrication methods or the body parts they are analogous to [7, 24]. The main sensing principles exploited in the current technologies are resistance, capacitance, piezoelectricity, optic component, or magnetics. Another possible classification regards the output nature of touch sensors. The major focus is often to measure the contact force and location over a certain area. Some other sensors, such as the ones exploited in this Thesis work [21], retrieve instead the pressure exerted on the sensor when contact is detected. Very recently, a novel vision-based optical tactile sensor, named GelSight [25, 26, 27], has been developed and commercialized. Unlike the traditional tactile sensors, GelSight basically measures geometry, with very high spatial resolution. The sensor has a contact surface of soft elastomer, and by a conventional camera it captures the deformations of the elastomer when the sensor is pressed against an object. The contact force and slip can be inferred from the sensor deformation itself.

Tactile sensors play a fundamental role in autonomous manipulation since they provide useful information when contacts with the objects are detected. Vision is usually the major source of measurements for perceiving the surrounding environment but it might be occluded while interacting with the object. In such a scenario, the ability of *localizing* and *recognizing* objects by means of solely tactile measurements is crucial for the execution of manipulation tasks. Hereafter, we briefly review the state of the art in *tactile object localization* and *recognition*.

Tactile object localization

The first contributions on tactile object localization (1980s) tackled the problem by using mostly iterative optimization methods and focused on finding a single solution best fitting the set of available measurements [28, 29, 30]. Since these methods tend to be trapped in local minima, *low initial uncertainty* is assumed so as to ensure that the optimization algorithm is initialized near the solution. In order to avoid local minima, the algorithm can be executed multiple times from different starting points.

Over the last years, Bayesian methods have been playing an important role in tactile localization [31, 32, 33, 34]. In particular, these methods are

capable of working with noisy sensors, inaccurate models, moving objects and can give information on where to sense next during tactile exploration. Thus, they can be used not only to localize the object, but also to provide useful information for collecting measurements and for real exploration.

Since the localization problem is intrinsically of multimodal nature (i.e. the probability density exhibits multiple peaks), nonlinear Kalman filtering techniques (such as the extended or unscented Kalman Filter) cannot be satisfactorily used. In this respect, the Bayesian framework (e.g. particle filtering) is more appropriate, since it intrinsically handles multimodal distributions. On the other hand, its main drawback is represented by the computational complexity, which grows exponentially with the number of independent variables (DOFs) and polynomially with the size of the initial region of uncertainty. For example, recalling that the localization of an object involves 6 DOFs, a particle filter should be configured to run with a number of particles in the order of 10^6, which might entail an unaffordable computational burden for real-time operation. In fact, most of the existing work is characterized by assumptions limiting the number of DOFs and the size of initial uncertainty.

In this respect, the first known work traces back to 2001 and is due to Gadeyne et al. [31], who performed 3-DOF localization of a box with initial uncertainty of 300 *mm* in position and 360 degrees in orientation. Measurements were taken by a force-controlled robot and a sampled measurement model, stored in a lookup table and constructed off-line, was used. In 2005, Chhatpar et al. [32] used particle filters to achieve 3-DOF localization with 20 *mm* of initial uncertainty in peg-in-hole assembly tasks. The exploited measurement model was obtained by sampling the object in advance.

An interesting approach to tactile localization makes use of the *Scaling Series* method [34, 35] developed by Petrovskaya et al., by which 6-DOF localization has been achieved with large initial uncertainty of 400 *mm* in position and 360 degrees in orientation. This method, which combines Bayesian Monte Carlo and annealing techniques, exploits measurements of contact points and surface normals. It performs multiple iterations over the data, gradually scaling precision from low to high. For each iteration, the number of particles is automatically selected on the basis of the complexity of the annealed posterior. As it will be shown in Chapter 4 through specific

experiments, the Scaling Series algorithm is however affected by low reliability, that can be attributed to the automatic process of particle generation. Particularly, a rough parameter tuning can easily lead to the generation of an insufficient number of particles or, on the contrary, to their exponential growth, thus precluding the final convergence of the algorithm.

In 2010, Corcoran et al. [33] used an annealed particle filter to estimate a 4-DOF pose and radius of cylindrical objects. The initial uncertainty was of 250 *mm* in position and unrestricted in orientation. The measurement model proposed in [34] was extended by exploiting the concept of "negative information". To this end, a set of "no-contact measurements" is defined to account for regions explored by the robot where it is known or it can be inferred that the object cannot be located, since no contacts are perceived.

In 2013, Chalon et al. [36] presented another particle filter method including information on both object and finger movements. A recent work [37] combines global optimization methods with the Monte Carlo approach in order to provide a batch solution to the global localization problem, either improving the estimate of the object pose obtained by vision or globally estimating pose when vision is not available.

In [38], Koval et al. propose a novel particle filter, named Manifold Particle Filter. This algorithm samples particles directly from a contact manifold guaranteeing the non-penetration constraint. In fact, the real pose of the object in contact with the robot hand belongs to a lower-dimensional manifold determined by non-penetration constraints between the object and the hand itself. If this kind of constraint is not incorporated, the sampled particles might correspond to configurations in which the manipulator and the object are overlapping or separated. The main limitation of this approach is however given by the fact that it relies on explicit analytic and sampled-base representations of the contact manifold. This kind of representation fits well to low-dimensional domains but do not scale properly to more complex scenarios. For this reason in [39], the same authors addressed the problem by proposing an implicit representation of the contact manifold to apply the Manifold Particle Filter to six or more dimensional state spaces.

In 2017, we proposed the Memory Unscented Particle Filter [40] which combines an Unscented Particle Filter with a windowing based memory

strategy to estimate the 6D pose of a stationary object using 3D tactile contact points - and no contact normals - and starting from an initial uncertainty of 400 mm in position and 360 degrees in orientation. This approach is one of the main contributions of the Thesis and is presented in Chapter 4.

Some recent works [41, 42, 43] have focused on the particular problem of tactile in-hand object pose estimation, i.e. the estimation of an object pose when hold in the robot hand. In particular, [43] makes use of a particle filter for processing tactile information and of haptic rendering models of the robot hand. The particle filter at first estimates the grasp pose of the object using the touch measurements. Then, hypotheses of grasp poses are evaluated by comparing tactile measurements and expected tactile information from CAD-based haptic renderings.

Another interesting application, which the attention in the latest years has been turned on, is visuo-tactile localization, i.e. the exploitation and fusion of tactile and vision information with the final goal of estimating the object pose [10, 27, 44]. A detail review of the state of the art of those approaches is hower out of the scope of this work.

Tactile object recognition

Tactile object recognition refers to the problem of discriminating an object with respect to others considering some of its properties measurable with touch sensors. Several features can be taken into account for the solution of the recogntion problem such as the object shape, surface texture or curvature.

Different methods have been proposed in the literature in order to solve tactile object recognition. They can be classified depending on the type of information they use and the object features they recover, namely, material and shape properties. Some researchers have focused on identifying material properties [45, 46, 47]. Decherchi et al. use multiple techniques to classify object materials with tactile data [46]. Liu et al. [47] apply a dynamic friction model to determine physical properties of surfaces while a robotic finger slides along the object with different speeds.

To recognize object shapes, a viable approach is to recover local geometry from each contact point, i.e., surface normal and curvature. By using a cylindrical tactile sensor, Fearing et al. propose a nonlinear, model-based inversion to recover contact surface curvatures [48]. Contact location point-clouds have also been used to reconstruct object shapes with computer graphic techniques [49, 50, 51, 52]. Allen et al. fit points from tactile sensors readings to superquadric surfaces to reconstruct unknown shapes [51]. A similar approach, proposed by Charlebois [53], uses tensor B-spline surfaces instead of superquadratic surfaces. Through these methods, arbitrary object shapes can be identified by estimating surface curvatures. In [54], we perform tactile recogniton by using contact location point-clouds but without reconstructing the object shape or estimating surface normals. We cast tactile recognition into a localization problem wherein multiple models are fit to the available measurements and objects are recognized by selecting the model that minimizes the localization error. More details about this approach are provided in Chapter 5.

Another solution to recognizing object shapes is to use machine learning techniques on the output of tactile sensor arrays. In this case, object features are extracted from the tactile data and/or haptic measurements. A classifier is then trained to predict the shapes of novel objects [18, 55, 56, 57, 58, 59, 60, 61, 62, 63].

The recent development of new sensors, such as GelSight, leads to sensitivity and resolution exceeding that of the human fingertips. This opens the possibility of measuring and recognizing highly detailed surface textures and shapes. The GelSight sensor, when pressed against a surface, delivers a height map. This can be treated as an image, and processed using the tools of visual texture analysis. An example of Gelsight application on texture recognition is shown in [64], where a material can be correctly categorized among a database of 40 classes of tactile textures corresponding to different materials. Other kinds of tactile sensors are also mounted on robot hands for extracting tactile images and applying computer vision technique to process such an information for object recognition [65, 66].

2.2 Bi-manual coordination

Humans take advantage of both their arms for executing plenty of manipulation tasks. Bringing bi-manual coordination skills to humanoid robots, themselves equipped with two arms, is then fundamental within the view of replicating the human manipulation abilities on a robotic platform.

Some examples of manipulation tasks where bi-manual coordination provides the solution are:

- Moving towards a desired position objects too heavy or big to be grasped with a single hand.
- Pick-and-place scenarios where the initial and target position of the object do not belong to the same arm workspace.
- Re-grasping objects whose in-hand pose is not suitable for performing the required task.

In the literature, bi-manual coordination has been addressed focusing on all the manipulation tasks listed above. One of the earliest contributions on bi-manual coordination is [67], in which the authors address the problem of planning the path of two cooperating robot arms to carry an object from an initial configuration to a goal configuration amidst obstacles. The paper compares three 2D planning techniques with different arms – 2-DOFs and 3-DOFs – in different scenarios (without and with obstacles). These results were later extended to 3D planning [68, 69, 70]. A more recent work [71] implements multi-arm handover for object movements towards a final position using the motion planning framework proposed in [70]. In [72], Kromer et al. address bi-manual coordination in a Reinforcement Learning setting. Their method exploits the phase structure in which tasks can be split in order to learn manipulation skills more efficiently. Starting with human demonstrations, the robot learns a probabilistic model of the phases and the phase transitions. Then model-based Reinforcement Learning is used to create motor primitives that well generalize to new situations and tasks. Another work where bi-manual coordination is cast into the learning framework is [73] where the authors combine a dynamical

systems formulation of the demonstrated trajectories and a task- parameterized probabilistic model to extract the relevant features of the demonstrated skill.

Another application of bi-manual coordination is object re-grasping [74, 75]. In [74], bi-manual re-grasping is formulated as an optimization problem, where the objective is to minimize execution time. The optimization problem is supplemented with image processing and a uni-manual grasping algorithm based on machine learning that jointly identifies two good grasping points on the object and the proper orientations for each end-effector. The optimization algorithm exploits this data by finding the proper re-grasp location and orientation to minimize the execution time. The work presented in [75] instead provides an interesting study on when one- or dual-arm re-grasp is to be performed, according to the object properties. Dual-arm re-grasp is more flexible and versatile but if the two hands grasps overlap on the object, an higher success rate is provided by single-arm re-grasp.

Dual-arm re-grasp is a specific case of the more general handover task, i.e. passing the object from one hand on to the other. The handover can be performed for executing pick-and-place tasks when the initial and target position of the object do not belong to the workspace of the same arm [76]. Other works as [54, 77, 78] provide solutions to the general handover problem. In [77], Gasparri et al. cast the handover into a robust optimization problem focusing on the choice of optimal stiffness to accomplish the handover by minimizing the forces involved. Both [54, 78] rank a priori generated poses for the execution of the handover. A detailed description of [54] is provided in Chapter 6, as one of the main contributions of this Thesis work.

2.3 Object modeling from vision

The ability of reconstructing in real-time 3D information about the scene and, particularly, the object to be manipulated is central to autonomous manipulation. Raw 3D information can be obtained differently according to the vision system the robot is equipped with, such as stereo or RGB-D cameras.

Several works in manipulation [79, 80, 81, 82] exploit only partial point clouds collected from a single view of the object to infer how to approach and manipulate the object. However, ignoring a 3D representation of the occluded portions of the object might lead to failures during the execution of the task. Reconstructing instead a full 3D model, capable therefore to represent also the occluded portions, provides much powerful information on the object shape and volume.

Object modeling has been studied since the 1980s mostly in computational geometry and computer graphics as surface reconstruction, i.e. the problem concerned with recreating a surface from scattered data points sampled from an unknown object. Several methods have been developed in last decades [83, 84, 85, 86, 87, 88] mostly returning mesh models as output of the modeling process.

In the last years, a great interest for object modeling arose also in robotics due to the growing availability of depth or stereo cameras [89, 90, 91, 92]. Several recent works reconstruct object mesh models from partial 3D informations using Deep Learning techniques [89, 93, 94, 95, 96, 97]. In [89], for instance, the authors perform shape completion through the use of a 3D convolutional neural network (CNN), trained on a large dataset of mesh models. At runtime, a 2.5D point cloud captured from a single point of view is fed into the CNN that returns a full 3D model of the object. This way, the occluded portion of the object is inferred by the network from the training set, thus providing more realistic models than just using symmetries or minimum volume closures. Another popular kind of object model used in most recent works consists of voxel-based models [98, 99]. They represent in fact the object volume and shape as a binary occupancy grid and therefore provide a representation suitable for being used with CNN, that are increasingly popular in 3D applications.

A different type of 3D model introduced in computer graphics in 1981 by A.H. Barr is the superquadric model [100, 101], a generalization of quadrics that has been well studied in graphics and computer vision [102]. Superquadrics and extensions such as hyperquadrics [103] and deformable superquadrics [101] are a convenient representation for a large class of both convex and non-convex objects. The most popular method to determine superquadric parameters for fitting partial or full object point clouds was proposed by Solina in 1990 [102]. Recently, several works have focused on

speeding up computation [90, 104] and refining the model by extending it to approximate complex shapes with a set of superquadrics [105, 106]. In addition to object approximation, superquadrics have been used for object detection [107], object segmentation [108, 109], collision detection [110, 111] and grasping [90, 112, 113, 114, 115, 116]. The great advantage of superquadrics with respect to mesh or voxel-based models consists of the small number of parameters to be memorized and their closed-form mathematical representation. Nevertheless, the new Deep Learning methods show much lower computation test time, since most of the effort is done during the offline training. In this respect, some preliminary ideas on how to speed up the superquadric computation making use of the Deep Learning framework have been proposed in [117].

2.4 Autonomous grasping

The grasping problem consists of computing a feasible pose of the robot hand, which allows grabbing the object under consideration. While great performance can be achieved if the shape and position of the object are accurately provided, autonomous grasping of unknown objects or whose pose is uncertain is still an open problem. Although the problem has been addressed since the late 1980s [118], recently the robotic community has shown an increasing interest in autonomous grasping [119, 120, 121]. Diverse methodologies have been explored addressing various goals still belonging to the field of grasping. In this respect, the broad field of autonomous grasping can be divided into more specific areas according to several criteria. For instance, grasp actions can be divided into *power* and *precision* grasps [122]. Power grasp involves large areas of contact between the hand and the object, without the adjustment of the fingers after contact [123]. On the contrary, precision grasp provides sensitivity and dexterity, since in this case the object is held with the tips of the fingers [124]. In precision grasp tasks, the hand touches the object at small contact points, therefore the study of grasp stability plays an important role.

Another classification criterion [120] considers how the robot hand pose for approaching the object is computed, grouping the methodologies in *analytic* and *empirical*. The former formulates the grasping problem only in terms of force-closure and form-closure, looking for specific conditions on

the contact wrenches that ensure a certain hand configuration to firmly hold any object. These approaches usually assume that contact point locations were given without explicitly relating the hand configuration to the object geometry. Empirical or *data-driven* approaches instead mimic human grasping in selecting a grasp that best conforms to task requirements and the target object geometry. They often rely on sampling grasp candidates for an object and ranking them according to a specific metric.

Until about twenty years ago, most popular robotic grasping approaches belonged to the analytical class, as reviewed in [119]. Data-driven approaches started to become popular with the availability of the simulator GraspIt! [125]. Several works have been developed [112, 126, 127] using this or analogous simulators and being characterized in how grasp candidates were sampled and ranked. More recently, some studies [128, 129] showed that techniques just evaluated in simulation do not provide good predictors for grasp success in the real world. This motivated several researchers [130, 131, 132] to let the robot learn how to grasp by experience gathered during grasps execution. The problem of transferring from simulated environments to the real world is then removed at the cost of extremely time-consuming collection of examples. Then, a crucial point becomes how to generalize the collected experience to novel objects so to contain the amount of required data. A great improvement in this respect has been provided by Saxena et al. [133] who trained simple logistic regressors on large amounts of synthetic labeled data to predict good grasping points in a monocular image. The authors demonstrated their method in a household scenario in which a robot emptied a dishwasher. Several other works were subsequently proposed addressing the problem of inferring discriminative object features for grasping [134, 135].

The availability of affordable and accurate depth sensing devices starting from the 2010 [136, 137] encouraged grasping research to rely increasingly more on 3D data. Processing depth map or point clouds is nowadays the starting point for the majority of approaches proposed for the grasping problem [90, 113, 123, 124, 138].

A further incentive for the study of grasping has been provided by the Amazon picking challeng [139], whose goal is the development of robotics hardware and software able to identify objects, grasp and move them from

place to place. Since 2015 this challenge has been giving rise to different approaches [140, 141, 142] sharing the same scenario: two-fingered grippers are used to grasp known and novel objects in the clutter.

The methods mentioned thus far place more weight on the object representation and the processing of perceptive data to retrieve grasps from some knowledge base or sample and rank grasps by comparison to existing experience. As a result, a convenient way to group data-driven approaches [121] is based on the prior knowledge assumed on the query object: if it is *known*, *familiar* or *unknown*. The trend of most recent works is in particular to focus on *familiar* or *unkwnon* objects, often generalizing from the available knowledge on how to grasp *known objects*.

One approach towards generating grasp hypotheses for unknown objects is to approximate objects with shape primitives. Marton et al. [91] show how grasp selection can be performed exploiting symmetry by fitting a curve to a cross section of the point cloud of an object. For grasp planning, the reconstructed object is imported in a simulator. Grasp candidates are generated through randomization of grasp parameters on which then the force-closure criteria is evaluated. Rao et al. [143] use segmentation, especially relying on depth information. A supervised localization method is then employed to select graspable segments and plans a grasping strategy, after shape completion from the partial 3D information. Bohg et al. [144] propose a related approach that reconstructs full object shape assuming planar symmetry. Recently, superquadrics functions have become a popular alternative to point clouds or mesh models for representing novel objects. As already mentioned in Section 2.3, they provide a compact mathematical formulation and their precision in modeling the object shape and volume occupancy has been proven to be suitable to grasping tasks. Some works [112, 113, 114] reconstruct the object model using a single or a set of superquadrics and then rely on grasp candidates generators (such as GraspIt!) to select the grasp candidate. In [90] instead, we exploit superquadrics not only for estimating the object 3D model but also for representing the volume graspable by the hand. A single proper grasp candidate is computed by overlapping the superquadric representing the volume graspable by the robot hand onto the object superquadrics while

meeting some orientation and obstacle avoidance constraints (e.g. avoidance of the support on which the object is located). We will provide extensive details on this approach in Chapter 7. Other ideas about how to use the superquadric model to compute directly grasp candidates are shown in [115] and [116]. Makhal et al. in [115] design the grasping pose by maximizing force balance and stability and taking advantage of dimension and surface curvature information obtained from the object superquadric parameters. In [116], the grasp candidates are located in proximity of the superquadric cardinal points of the upper part of the object (for avoiding the support on which the object is located), taking into account proper constraints on orientations. The poses are then ranked according to their reachability and the matching between the object and hand dimensions.

Another approach to deal with unknown objects consists of extracting from 2D or 3D visual information those features able to encode some properties of the object relevant to the grasp. Several heuristics and patterns have been proposed to extract grasp candidates from low-level [79, 80, 138, 145] or more global shape information [146, 147]. Very recently, the most common technique used to extract grasping features from 3D data is Deep Learning [81, 82, 148, 149]. A relevant work in this respect is given by the DexNet project [150, 151, 152] including a growing synthetic dataset of million point clouds, grasps, and analytic grasp metrics generated from thousands of 3D models and a model that rapidly predicts the probability of success of grasps from depth images, where grasps are specified as the gripper planar position, angle and depth. Throughout their papers, Mahler et al. have extended the Dexnet dataset and improved the model predicting grasp success, being able to provide very high success rate on novel and adversarial objects. The large number of samples of the Dexnet dataset is linked to a crucial limitation of Deep Learning techniques: the need of a huge amount of labeled data to enable generalization. Collecting such a number of samples in the real world is not time-wise affordable. The alternative is to generate the data in simulation environments. However, as already mentioned, grasp predictors trained only on simulated data are very likely to behave poorly in the real world. New common strategies for facing this issue are *domain randomization* [153, 154] and *adaptation* [155, 156]. The former consists in training the grasp predictor in simulation using tons of data randomly generated and synthetically

labeled and randomizing over different parameters of the system (e.g. image noise and physical parameters etc.). The latter is a particular case of Transfer Learning that utilizes labeled data in one or more relevant source domains to execute new tasks in a target domain. In grasping applications the source domains are generated and used in simulation and real-world is the target domain.

In the last years, also the Deep Reinforcement Learning framework has been used for addressing the grasping problem. More details about the latest results in this respect are collected in the following Section.

For the sake of comparison, Table 2.1 summarizes the state of the art of autonomous grasping by focusing on data-driven methods for power grasp of unknown objects, including also Deep RL methods that will be described in the following Section.

TABLE 2.1: State of the art comparison among data-driven approaches for power grasp of unknown objects.

Reference	Methodology	Distinctive feature	Input	Robot hand
[123]	pose ranking	geometry-based	3D partial point cloud	multi-fingered
[112]	pose ranking	simulator-based	reconstructed superquadrics	multi-fingered
[126]	pose ranking	geometry-based	reconstructed shapes	multi-fingered
[127]	supervised learn.	SVM	reconstructed superquadrics	multi-fingered
[130]	supervised learn.	automatic data collection	RGB images	gripper
[132]	supervised learn.	predict pose reliability	RGB images	multi-fingered
[133]	supervised learn.	synthetic dataset	RGB images	gripper
[134]	supervised learn.	predict pose stability	RGB-D images	multi-fingered
[90]	optimization	robot agnostic	reconstructed superquadrics	multi-fingered
[138]	pose ranking	local features	3D point cloud	gripper
[91]	pose ranking	simulated grasp planner	reconstructed mesh model	multi-fingered
[143]	supervised learn.	fill missing depth data	reconstructed shape	multi-fingered
[144]	pose ranking	simulated grasp planner	reconstructed mesh model	multi-fingered
[115]	geometry-based	mirror partial point cloud	reconstructed superquadrics	gripper
[116]	pose ranking	geometry-based	reconstructed superquadrics	multi-fingered
[79]	pose ranking	local features	RGB-D images + tactile data	gripper
[80]	pose ranking	local features	RGB-D images	gripper
[145]	local features	active exploration	RGB images	gripper
[146]	pose ranking	global features	RGB images	gripper
[147]	pose ranking	global features	RGB-D images	multi-fingered
[150]	supervised learn.	synthetic dataset	RGB-D images	gripper
[151]	supervised learn.	synthetic dataset	RGB-D images	gripper
[152]	supervised learn.	synthetic dataset	RGB-D images	gripper + suction
[153]	supervised learn.	domain randomization	RGB-D images	gripper
[154]	supervised learn.	domain randomization	RGB-D images	gripper
[155]	supervised learn.	domain adaptation	RGB images	gripper
[156]	supervised learn.	domain adaptation	RGB images	gripper
[4]	supervised learn.	spatial softmax CNN	RGB images	gripper
[5]	deep RL	trained on real-robot	RGB images	gripper
[6]	deep RL	trained on real-robot	RGB images	gripper
[157]	deep RL	trained on real-robot	RGB images	gripper
[158]	deep RL	trained on real-robot	RGB images	gripper

2.5 Deep Reinforcement Learning for autonomous manipulation

One goal of artificial intelligence is to provide fully autonomous agents interacting with the environment and learning optimal behaviors through trial and error. Reinforcement Learning (RL) provides a suitable mathematical framework for experience-driven autonomous learning [159] and has been shown to be successful in several tasks in the past [160, 161, 162] although in the context of low-dimensional domains. RL is in fact affected by the lack of scalability due to its memory requirements and computational complexity. As happened in many areas of machine learning, such as computer vision, speech recognition and language translation, Deep Learning had had a significant impact on RL, improving considerably the state of the art and defining the so-called field of Deep Reinforcement Learning. The main reason of Deep Learning success is that deep neural networks are able to learn compact low-dimensional representations, i.e. *features*, of high dimensional data, such as images. This way, RL has scaled to decision-making problems that were previously intractable, due to their high-dimensional state and action spaces. One example of the successful stories of Deep RL is the development of an algorithm able to play a set of Atari 2600 video games with super-human performance, learning directly from image pixels [163]. Another example is the hybrid DeepRL system, AlphaGo, that defeated a human world champion in Go [164] and brought to another level the AI revolution started two decades earlier with DeepBlue [165] and Watson DeepQA [166] that won respectively chess and quiz competitions against human players. In particular, the novelty of AlphaGo consisted of the usage of neural networks trained with supervised and reinforcement learning together with a traditional heuristic search algorithm.

Deep RL has been applied to a variety of different fields, including also robotics in the very last years. One of the most studied problems is autonomous manipulation [4, 5, 6, 157, 158, 167]. In the popular work described in [6], the authors propose a learning-based approach to learn hand-eye coordination for robotic grasping from monocular images. They train a large CNN to predict the probability that the motion of the gripper

will achieve a successful grasps, using only monocular camera images independent of camera calibration or the current robot pose. For this reason the network is required to learn also the spatial relationship between the gripper and the objects in the scene. The approach was tested on real robots thanks to the collection of large-scale datasets, using up to 14 manipulators and two months of grasp attempts. In [4] Levine et al. propose a method to learn CNN policies that map from raw images to torques at robot motors. The approach is tested on a range of real-world manipulation tasks such as screwing a cap onto a bottle and placing a coat hanger on a clothes rack. Another work where DeepRL is used to learn policies directly from raw pixels is presented in [5]. This method uses a deep spatial autoencoder to acquire a set of feature points that describe the environment for the manipulation task to be solved, such as object position, and learns a motion skill with these feature using RL. This approach is shown with the PR2 robot on task including pushing toy blocks, picking up items with a spatula and hanging a loop of rope of a hook at various positions. Although outstanding, these works still show some limitations. In [6], a huge amount of data is required to be collected on the real robot for achieving good performances and the motions necessary for performing the tasks addressed in [5] are limited to simple actions.

An interesting extension in this respect is the use of multi-fingered manipulators for the accomplishment of dexterous manipulation tasks that cannot be executed with a single gripper, such as in-hand manipulation, complex grasping and tool use. These tasks turn out to be very challenging due to the high dimensional observation and action spaces involved and the difficulties in defining proper reward functions for guiding the agent during the training. Solving tasks with such a level of difficulty often require to face one of the greatest difficulties in RL: the *exploration versus exploitation problem*, i.e. the problem of choosing between non-optimal actions in order to explore the state space and exploiting the optimal action in order to make useful progress. One of the simplest exploration strategy, typical used in off-policy algorithms such as DQN [163], is the ϵ-greedy exploration policy that chooses a random action with probability $\epsilon \in [0, 1]$ and the optimal action otherwise. By decreasing ϵ over time, the agent progresses towards exploitation. As the task becomes more complex or temporally extended however, this kind of exploration strategies

becomes less effective. Several exploration strategies have been proposed in the last years comprising different criteria used for encouraging exploration. In [168, 169], the exploration is based on intrinsic motivation. During the training, the agent learns also a model of the system and an exploration bonus is assigned when novel states with respect to the trained model are encountered. Novel states are identified as those states that create a stronger disagreement with the model trained until that moment. Another group of exploration algorithms are count-based methods that directly count the number of times a certain state has been visited to guide the agent towards states less visited. Obviously, such an approach is infeasible in continuous state space. For this reason, some works such as [170] extend count-based exploration approaches to non-tabular (continous) RL using density models to derive a pseudo-count of the visited states. Another approach to encourage exploration consists of injecting noise to the agent's parameters, leading to richer set of agent behaviors during training [171, 172].

These exploration strategies are task agnostic in that they aim at providing good exploration without exploiting any specific information of the task itself. More recently instead, the exploration problem has been cast into *meta-learning* (or learning to learn), the field of machine learning whose goal is to learn strategies for fast adaptation by using prior tasks [173]. An example of application of meta-learning for the exploration problem is shown in [174], where a novel algorithm is presented to learn exploration strategies from prior experience.

Alternatively, it is possible to get around the exploration problem by providing task demonstrations for guiding and speeding up the training [175, 176, 177, 178, 179]. The work presented in [180] shows how the proper incorporation of human demonstrations into RL methods allows reducing the number of samples required for training an agent to solve complex dexterous manipulation tasks with a multi-fingered hand in simulation. Even if verification on a real robot is still to be verified, the training time thus far obtained is compatible with real-world applications. More details about this work are presented in Chapter 9.

Chapter 3

The iCub humanoid robot and its key components for manipulation

In this Chapter, a brief overview of the iCub humanoid robot is provided. More specifically, the description will cover only those components that are relevant for the manipulation problem (see Fig. 3.1).

The iCub [2] is an open-source robotic platform developed for robotics research. It has the appearance of a child and its design is human-inspired. As mentioned in Section 1.1, the iCub is a proper platform for studying the manipulation problem as it is provided with the following components:

- a human-like upper body including two arms and multi-fingered hands (Section 3.1);
- a proper sensor system including two cameras, tactile sensors and encoders for joint angle sensing (Section 3.2);
- a Cartesian controller for the upper body (Section 3.3).

Section 3.4 ends the platform description introducing the software framework of the iCub.

3.1 iCub upper body

The total number of DOFs of the iCub for the complete body is 53. Focusing on the upper body, the (41) DOFs are distributed as follows:

- 6 in the head: 3 for the neck, providing full movements and 3 for the cameras, to support both tilt, pan and vergence[1] behaviors.

[1]The vergence is the simultaneous movement of both eyes in opposite directions to obtain or maintain single binocular vision.

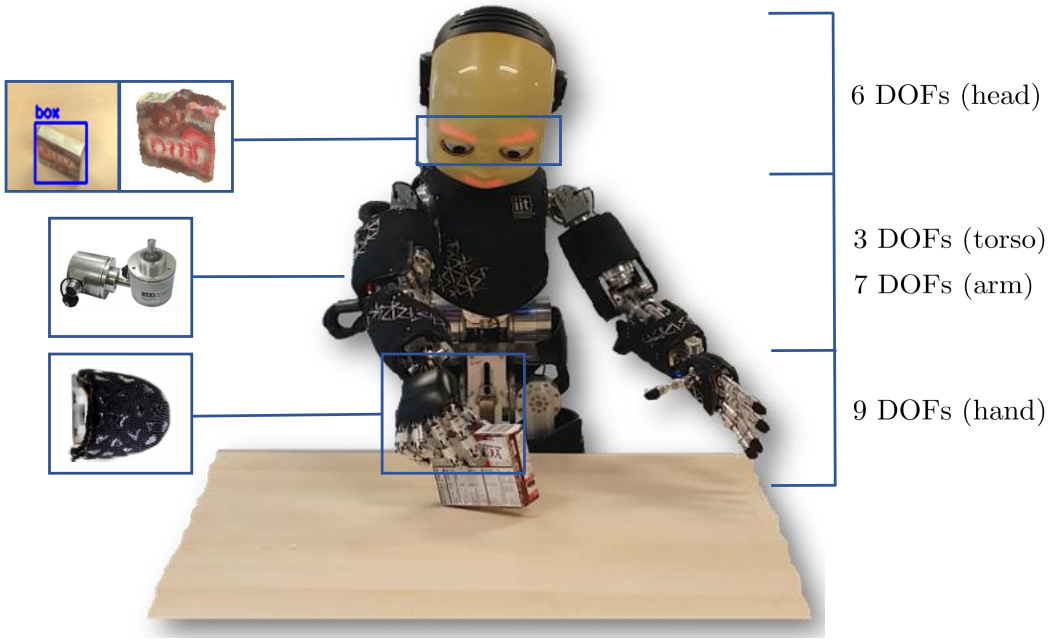


FIGURE 3.1: iCub hardware and perception key components for manipulation. On the left: the sensor system. Stereo vision provides RGB and depth information; encoders are used for proprioception through joints sensing; tactile sensors located on the fingertips detect contacts with the object to manipulate. On the right: how the DOFs are distributed in the upper body.

- 7 for each arm: 3 for the shoulder, 2 for the elbow and 2 for the wrist.
- 9 for each hand: 3 for the thumb, 2 for the index, 2 for the middle finger, 1 for the ring and little fingers and 1 for finger abduction[2]. Consequently, each hand has three independent fingers whereas the fourth and fifth are used for additional stability. Fingers are tendon-driven, with most of the motors located in the forearm. Tendon driven robots (TDR) are robots whose limbs mimic biological musculoskeletal systems, using plastic straps. Such robots are claimed to move in a "more natural" way than traditional robots that use rigid metal or plastic limbs controlled by geared actuators.
- 3 in the waist. A 2 DOF waist/torso would be enough for effective crawling but a 3 DOF waist was incorporated to support manipulation. A 3 DOF waist provides increased range and flexibility of motion for the upper body resulting in a larger workspace for manipulation.

3.2 Perception system

Concerning perception, iCub is equipped different kinds of sensors, including digital cameras (one for each eye), gyroscopes and accelerometers, microphones, encoders, force/torque and tactile sensors. Hereafter, we focus only on those that are mostly relevant for manipulation tasks.

3.2.1 Vision

The RGB cameras mounted are used to perform stereo vision, i.e. the extraction of 3D information from digital images that contain two views of the same scene.

The 3D spatial information of the scene can be obtained by estimating the relative pose of one of the two cameras with respect to the other. This information is coded in the so-called extrinsic parameters of the stereo vision system. Dealing with moving eyes requires the estimation of the extrinsic parameters each time the robot eyes change their relative configuration. This is done on the iCub with a complete Structure From Motion

[2]In physiology, adbuction is the movement which separates a limb or other part from the axis.

pipeline [181]. The extrinsic parameters allows then the rectification of the images and the computation of the disparity map. Finally, the disparity map, the intrinsic parameters of the camera and the forward kinematics of the iCub eyes are used to compute the 3D coordinates of a point in the image expressed with respect to the root frame of the robot.

3.2.2 Tactile sensors

The iCub is equipped with tactile sensors based on capacitive pressure system and covering almost its entire body. The sensors on its arms, torso and legs are mostly used for force estimation and compliant control (see Paragraph 3.3). The tactile sensors on the palm[3] and, in particular, the fingertips of the hands are instead relevant for manipulation. The fingertip structure is the following, outlined in Fig. 3.2.

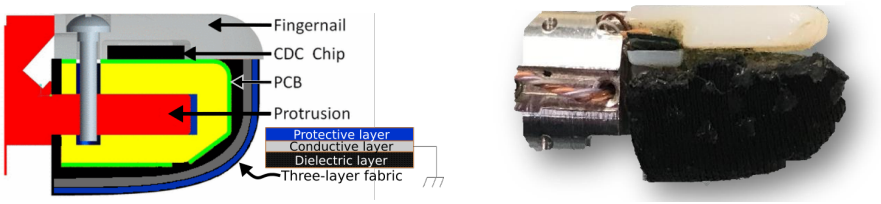


FIGURE 3.2: Cross-section of the fingertip. Yellow: inner support. Green: flexible PCB. Black, grey and blue: the composite three-layer fabric, respectively the dielectric, conductive and protective layers. This structure increases the robustness and repeatability of the fingertip.

A flexible PCB is wrapped around the inner support and hosts 12 sensors called taxels. The PCB also hosts a chip that converts capacitance readings to digital data. A plastic surface of 1mm is used as a mechanical interface to the external environment. It has an inner part that adapts to the shape of the PCB and an external part on which a three-layer fabric is glued. The fabric includes a dieletric layer, a conductive layer, connected to the ground, and a protective textile layer (the black material visible in Fig. 3.3). The conductive lycra is connected to ground. This assembly effectively forms a capacitive pressure sensor.

[3]Due to the hand design, the tactile sensors on the palm are rarely activated during manipulation tasks. The limited hand opening in fact usually prevent the manipulated object to be in contact with the robot palm. For this reason, sometimes we refer to the tactile sensors on the fingertips as the tactile sensors of the hand.

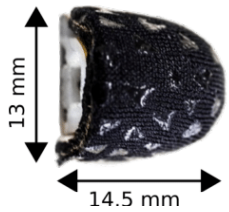


FIGURE 3.3: The protective textile layer on the fingertip.

3.2.3 Proprioception

Proprioceptive inputs in the iCub simply consist of angular position measurements in every joint. For most joints, they are provided by absolute 12bit angular encoders. The joint angles of the hand are sensed using a custom-designed Hall-effect-magnet pair. [182]

3.3 The Cartesian controller

The iCub is provided with a *cartesian controller* [183] for the arms and a *gaze controller* [184] for the head and the vision system. They provide an interface that exploits an inverse kinematics solver in order to control the arms and the head directly in the operational space, by querying 3D points instead of configurations at the joint level and generating trajectories with human-like minimum-jerk velocity profiles [185].

The robot joints can be then controlled with different control modes, including also impedance mode. This mode allows controlling the joint position and its compliance. In particular, both the equilibrium position of a virtual spring and its stiffness/damping are controlled. By tuning the stiffness parameters, the robot joint can behave like a hard or soft spring, while maintaining control on the desired joint position. The torque applied on the robot joints are estimated by combining the force-torque sensors available on the iCub shoulders and the tactile sensors on its upper body. The latter allow better estimating the application point of the torque. The impedance control mode implements a safe way to control robots operating in unstructured environments and interacting with humans.

3.4 Yarp

All the software that runs on the iCub is written using YARP [186]. YARP (Yet Another Robot Platform) is an open-source software framework that supports distributed computation and is compatible with multiple operating systems (Windows, Linux and Mac OS). YARP facilitates the reuse of code by decoupling the user code from the specific hardware (using special device drivers called *PolyDrivers*) and operating system (thanks to OS wrappers). It also enables the development of modular software architectures thanks to an intuitive inter-process communication mechanism based on the concept of Port. A YARP module open ports in order to communicate with other modules and send/receive commands and data. Ports are extremely versatile as they support several types of data (vectors, matrices, images, sounds, point clouds, etc.) and several protocols (e.g. TCP, UDP and many others). YARP also provides several libraries for mathematical computations (vectors, matrices, matrix inversion and singular value decomposition, etc.). Basic image processing is also possible thanks to the integration with the computer vision library OpenCV.

Part II

A novel tactile object localization algorithm: the Memory Unscented Particle Filter

Chapter 4

Memory Unscented Particle Filter for 6-DOF Tactile Object Localization

Accurate object perception is a necessary requirement for the execution of manipulation tasks with autonomous robots in real-world environments. This makes the advances in robotic manipulation and perception strongly related to each other. In particular, the development of new sensors and inference algorithms enhance the robot ability to deal with uncertainties and reduce the cost required to engineer the environment in which the robot will operate. Recently, a great interest arose regarding the use of tactile sensors for manipulation tasks [7, 8, 9, 10, 11, 12]. While the use of vision has been thoroughly investigated [187], recent advances in tactile technology have made it possible to build tactile systems that are reliable enough and can be deployed on real robots at a reasonable price [21, 22, 23]. This recent improvement of tactile sensors is surely one of the reasons for a surge of interest on this topic [15, 16, 17, 19, 20]. But tactile sensing is also fundamental whenever vision is unavailable or imprecise, for example due to occlusions and/or bad lighting conditions. In addition, findings in human physiology testify how humans jointly exploit vision and touch in order to accomplish manipulation tasks and how humans are even able to explore objects by means of tactile perception solely [13], [14].

Due to technological limitations, most tactile systems have low resolution and rarely provide other than estimation of the force normal to the surface. Object localization using tactile feedback is, therefore, challenging and requires the development of filtering techniques that allow appropriate fusion of multiple measurements, taking into account the presence of

noise and the real-time requirements of the task.

In this Chapter, we present a novel algorithm, named *Memory Unscented Particle Filter (MUPF)* [40], designed for addressing global 6-DOF tactile object localization. This algorithm relies on the *Unscented Particle Filter* (UPF) [188] and exploits only the measured position of the contact points obtained from the tactile sensors on the robot. Other works in the literature instead take advantage of other measurements such as the surface normal at the contact points [34, 189, 190] or a 6-dimensional vector including force and torque [31]. The proposed solution is inherently recursive in that the measurements are sequentially processed in real time as they become available, and the algorithm can provide the object's pose estimate after the processing of each measurement. We take into account a recursive approach for several reasons: the algorithm can provide the object's pose estimate after the processing of each measurement, and not only at the final measurement acquisition time as with a batch procedure like the one in [34, 35]; it is compliant with active exploration techniques where the robot decides, at each time t, where to sense next on the basis of the current object's pose estimate; it can allow stopping the object localization procedure at a given time t whenever a suitable stopping criterion is satisfied.

The Chapter is organized as follows. Section 4.1 is a brief introduction to nonlinear filtering techniques useful for the subsequent theoretical developments. Section 4.2 provides a mathematical (Bayesian) formulation of the tactile localization problem. Section 4.3 presents the novel *Memory Unscented Particle Filter* (MUPF) approach to 6-DOF tactile localization. Section 4.4 demonstrate the effectiveness of the proposed approach by means of simulation and experimental tests on the iCub humanoid robot. Finally, Sections 4.5 ends the Chapter with concluding remarks, applications and perspectives for future work.

4.1 Mathematical background

Hereafter, *tactile localization* is cast into the Bayesian framework and addressed as a nonlinear multimodal filtering problem. Recall that filtering is the problem of recursively estimating the state $\boldsymbol{x}_t \in \mathbb{R}^n$ of a dynamical system while acquiring and processing noisy observations on-line. Specifically, from a Bayesian viewpoint, the goal of the filtering problem is to

recursively compute the following conditional PDFs

$$\begin{aligned} p_{t|t}(\boldsymbol{x}) &= p(\boldsymbol{x}_t = \boldsymbol{x}|\boldsymbol{y}^t) \\ p_{t+1|t}(\boldsymbol{x}) &= p(\boldsymbol{x}_{t+1} = \boldsymbol{x}|\boldsymbol{y}^t), \end{aligned} \tag{4.1}$$

given the noisy observations $\boldsymbol{y}^t = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t\}$ with $\boldsymbol{y}_t \in \mathbb{R}^p$.

The solution of the filtering problem is given by the Bayesian recursion, starting from the initial prior $p_{1|0}(\cdot)$ and consisting of two functional equations, i.e. the following Bayes and respectively Chapman-Kolmogorov equations:

$$p_{t|t}(\boldsymbol{x}) = \frac{\ell_t(\boldsymbol{y}_t|\boldsymbol{x})p_{t|t-1}(\boldsymbol{x})}{\int \ell_t(\boldsymbol{y}_t|\boldsymbol{\xi})p_{t|t-1}(\boldsymbol{\xi})d\boldsymbol{\xi}} \tag{4.2}$$

$$p_{t+1|t}(\boldsymbol{x}) = \int \varphi_{t+1|t}(\boldsymbol{x}|\boldsymbol{\xi})p_{t|t}(\boldsymbol{\xi})d\boldsymbol{\xi}, \tag{4.3}$$

where $\varphi_{t+1|t}(\boldsymbol{x}|\boldsymbol{\xi})$ is the *Markov transition density* representing the conditional probability that the state at time $t+1$ will take value $\boldsymbol{x}$ given that the state at time t is equal to $\boldsymbol{\xi}$, and $\ell_t(\boldsymbol{y}|\boldsymbol{x})$ is the *measurement likelihood function* denoting the probability that the measurement at time t will take value $\boldsymbol{y}$ given that the state is equal to $\boldsymbol{x}$.

However, in many practical applications, such as navigation, tracking and localization, the transition and likelihood models are usually affected by nonlinearities and/or non-Gaussian noise distributions, thus precluding analytical solutions of (4.2) and (4.3). In these cases, one must invariably resort to some approximation technique. Most of the existing approximation techniques can be divided in two families: Kalman-filtering-like approaches, and sequential Monte Carlo methods. The algorithms belonging to the former family (like the *Extended Kalman filter* and the *Unscented Kalman filter (UKF)* [191]-[192]) propagate only the first- and second-order moments (i.e., mean and covariance) of the posterior state distribution. Such methods are usually characterized by a low computational cost, but are not appropriate for multimodal distributions like the one arising from the tactile localization problem. On the other hand, sequential Monte Carlo methods, also known as *particle filters* [193], can deal with arbitrary nonlinearities and distributions and supply a complete representation of the posterior state distributions.

Particle filtering techniques stem from the idea of approximating the

posterior density function $p_{t|t}(\boldsymbol{x})$ by means of a finite set of weighted samples (particles) as

$$\hat{p}_{t|t}(\boldsymbol{x}) \approx \sum_{i=1}^{N} \tilde{w}_t^{(i)} \, \delta(\boldsymbol{x} - \boldsymbol{x}_{t|t}^{(i)}), \tag{4.4}$$

where $\delta(\cdot)$ is the Dirac delta function, $\boldsymbol{x}_{t|t}^{(i)}$ is the position of the i-th particle and $\tilde{w}_t^{(i)}$ its normalized importance weight. In this way, the evaluation of the integrals that are necessary for application of the Bayesian filtering equations (4.2) and (4.3) is performed via the Monte Carlo numerical integration method, i.e., by transforming the integrals into discrete sums.

In principle, the particle approximation (4.4) can be computed by drawing a set of independent and identically distributed samples $\{\boldsymbol{x}_{t|t}^{(i)}, i = 1, \ldots, N\}$ from the posterior $p_{t|t}(\boldsymbol{x})$. While such a solution is not feasible because $p_{t|t}(\boldsymbol{x})$ is not known, the difficulty can be circumvented by sampling each particle i from a known, easy-to-sample, *proposal distribution* $q^{(i)}(\boldsymbol{x}_t|\boldsymbol{y}^t)$, and then compute the normalized importance weights as

$$w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \frac{\ell_t(\boldsymbol{y}_t|\boldsymbol{x}_{t|t}^{(i)}) \; \varphi_{t|t-1}(\boldsymbol{x}_{t|t}^{(i)}|\boldsymbol{x}_{t|t-1}^{(i)})}{q^{(i)}(\boldsymbol{x}_{t|t}^{(i)}|\boldsymbol{y}^t)}, \tag{4.5}$$

$$\tilde{w}_t^{(i)} = w_t^{(i)} / \sum_{j=1}^{N} w_t^{(j)}. \tag{4.6}$$

In fact, by comparing (4.5) with (4.2) and (4.3), it is an easy matter to see that the resulting particle-based description approximates the true posterior $p_{t|t}(\boldsymbol{x})$ at time t.

4.1.1 The Unscented Particle Filter

The main drawback of particle filtering techniques is that, unless special care is taken, the number N of particles needed to make the approximation (4.4) sufficiently accurate can increase exponentially with the dimension n of the vector to be estimated (since it is required to sample in a subset of $\mathbb{R}^n$). In this respect, a critical point of particle filtering is how to choose the proposal distribution $q^{(i)}(\boldsymbol{x}_t|\boldsymbol{y}^t)$ so as to approximate the posterior reasonably well with a moderate number of particles. Among the most effective variations, there is the *unscented particle filter* (UPF) which exploits the UKF

in the proposal distribution to improve performance [188]. In the following part of this section, an outline of the UPF algorithm is provided.

The UPF propagates a set of extended particles $\mathcal{P}_t = \{\mathcal{P}_t^{(1)}, \ldots, \mathcal{P}_t^{(N)}\}$, each one comprising a weight $\tilde{w}_t^{(i)}$, a mean $\boldsymbol{x}_{t|t}^{(i)}$, and a covariance $P_{t|t}^{(i)}$, i.e.,

$$\mathcal{P}_t^{(i)} = \{\tilde{w}_t^{(i)}, \boldsymbol{x}_{t|t}^{(i)}, P_{t|t}^{(i)}\}.$$

Given the set of particles at time $t-1$, the UKF prediction and correction steps are applied to each particle mean and covariance so as to move the particle towards the measurements. Then, for each i, a new particle is sampled using $\mathcal{N}(\boldsymbol{x}_t; \bar{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)})$ as proposal distribution where $\bar{\boldsymbol{x}}_t^{(i)}$ is the updated mean after the correction step, $P_{t|t}^{(i)}$ is the updated covariance, and $\mathcal{N}(\boldsymbol{x}; \bar{\boldsymbol{x}}, P)$ denotes the normal distribution with mean $\bar{\boldsymbol{x}}$ and covariance P, thus achieving a more dense sampling in the most relevant areas of the search space.

In order to apply the UKF to each particle, it is necessary to assume that the Markov transition density $\varphi_{t+1|t}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$ and measurement likelihood function $\ell_t(\boldsymbol{y}_t|\boldsymbol{x}_t)$ are generated by a state transition and, respectively, measurement equation, so that the time evolution of $\boldsymbol{x}_t$ and $\boldsymbol{y}_t$ can be described by the discrete-time dynamical system

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}_t(\boldsymbol{x}_t, \boldsymbol{\omega}_t) \tag{4.7}$$

$$\boldsymbol{y}_t = \boldsymbol{h}_t(\boldsymbol{x}_t, \boldsymbol{\nu}_t). \tag{4.8}$$

Notice that in system (4.7)-(4.8) the probabilistic nature of the model is captured by the *process disturbance* $\boldsymbol{\omega}_t$ and *measurement noise* $\boldsymbol{\nu}_t$, which are supposed to be sequences of independent random variables with known probability density functions.

The UKF does not directly approximate the nonlinear process and observation models, but exploits the nonlinear models, approximating the distribution of the state. This is made possible by means of the *scaled unscented transformation (SUT)* [194], which is a tool for computing the statistics of a random variable undergoing a nonlinear transformation. Specifically, the state distribution is specified using a minimal set of deterministically chosen sample points. Such sample points exactly provide the true mean and covariance of such a variable and, when propagated through

the nonlinear transformation, they approximate the posterior mean and covariance accurately to the *2nd order* for any nonlinearity. For the reader's convenience, a brief review of the SUT is provided hereafter.

Let $\boldsymbol{x} \in \mathbb{R}^{n_x}$ be a random variable, with mean $\bar{\boldsymbol{x}}$ and covariance P_x, and $\boldsymbol{g} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ an arbitrary nonlinear function. The goal is to approximate the mean value $\bar{\boldsymbol{y}}$ and covariance P_y of the variable $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$. A set of $2n_x + 1$ weighted samples or *sigma points* $\mathcal{S}_i = \{\mathcal{W}_i, \boldsymbol{\mathcal{X}}_i\}_{i=0}^{2n_x}$ are chosen to completely represent the true mean and covariance of the variable $\boldsymbol{x}$, i.e.

$$
\begin{aligned}
\boldsymbol{\mathcal{X}}_0 &= \bar{\boldsymbol{x}} \\
\boldsymbol{\mathcal{X}}_i &= \bar{\boldsymbol{x}} + (\sqrt{(n_x + k)P_x})_i \qquad i = 1, \ldots, n_x \\
\boldsymbol{\mathcal{X}}_i &= \bar{\boldsymbol{x}} - (\sqrt{(n_x + k)P_x})_i \qquad i = n_x + 1, \ldots, 2n_x \\
\boldsymbol{\mathcal{W}}_0^{(m)} &= \lambda/(n_x + \lambda) \\
\boldsymbol{\mathcal{W}}_0^{(c)} &= \lambda/(n_x + \lambda) + (1 - \alpha^2 + \beta) \\
\boldsymbol{\mathcal{W}}_i^{(m)} &= \boldsymbol{\mathcal{W}}_i^{(c)} = 1/[2(n_x + \lambda)] \qquad i = 1, \ldots, 2n_x,
\end{aligned}
$$

where: $\lambda = \alpha^2(n_x + k) - n_x$; $\alpha > 0$ provides one more degree of freedom to control the scaling of the sigma points and to avoid the possibility of getting a non-positive semi-definite covariance; $k \geq 0$ is another scaling parameter; β affects the weighting of the zero-*th* sigma point.

Each sigma point is then propagated through the function $\boldsymbol{g}(\cdot)$ ($\boldsymbol{\mathcal{Y}}_i = \boldsymbol{g}(\boldsymbol{\mathcal{X}}_i)$, for $i = 0, \ldots, 2n_x$) and the estimated mean and covariance of $\boldsymbol{y}$, as well as the cross-covariance between $\boldsymbol{x}$ and $\boldsymbol{y}$, are computed as follows:

$$
\begin{aligned}
\bar{\boldsymbol{y}} &= \sum_{i=0}^{2n_x} \mathcal{W}_i \boldsymbol{\mathcal{Y}}_i \qquad P_{\boldsymbol{y}} = \sum_{i=0}^{2n_x} \mathcal{W}_i (\boldsymbol{\mathcal{Y}}_i - \bar{\boldsymbol{y}})(\boldsymbol{\mathcal{Y}}_i - \bar{\boldsymbol{y}})^T \\
P_{\boldsymbol{xy}} &= \sum_{i=0}^{2n_x} \mathcal{W}_i (\boldsymbol{\mathcal{X}}_i - \bar{\boldsymbol{x}})(\boldsymbol{\mathcal{Y}}_i - \bar{\boldsymbol{y}})^T.
\end{aligned}
\tag{4.9}
$$

The Unscented Kalman Filter is obtained by applying the SUT to the nonlinear functions f_t and h_t in (4.7)-(4.8).

In practice, in the UPF algorithm, given the mean $\boldsymbol{x}_{t-1|t-1}^{(i)}$ and covariance $P_{t-1|t-1}^{(i)}$ at time $t-1$, as well as the mean and covariance of the process disturbance $\boldsymbol{\omega}_{t-1}$, application of the SUT to the state transition equation (4.7) allows to compute an approximation of the predicted mean $\boldsymbol{x}_{t|t-1}^{(i)}$ and

covariance $P^{(i)}_{t|t-1}$ at time t. In turn, given $\boldsymbol{x}^{(i)}_{t|t-1}$ and $P^{(i)}_{t|t-1}$ as well as the mean and covariance of the measurement noise $\boldsymbol{\nu}_t$, application of the SUT to the measurement equation (4.8) allows to provide an approximation of the predicted measurement mean $\boldsymbol{y}^{(i)}_{t|t-1}$ and covariance $S^{(i)}_t$ as well as of the state-measurement cross-covariance matrix $\Gamma^{(i)}_t$. Then, the updated mean $\bar{\boldsymbol{x}}^{(i)}_t$ and covariance $P^{(i)}_{t|t}$ are obtained by applying the standard Kalman filter correction step.

Since in practice it can happen that, after a few iterations, one of the normalized weights tends to 1, while the remaining weights tend to zero (*weight degeneration*), a selection or *resampling* stage is usually included in the particle filtering algorithm, in order to eliminate samples with low importance weights and replicate samples with high importance weights. Summing up, the resulting algorithm is reported in Algorithm 1.

4.2 Problem formulation

The object to be localized is assumed to be static during the measurement collection. This assumption is common to other works [31, 34, 35, 190] and is realistic, for instance, if the object is very heavy or is stuck on a support preventing any possible movement. Hence, the goal of the 6-DOF object tactile localization problem is to estimate in real-time the pose $\boldsymbol{x} \in \mathbb{R}^6$ of an object $\mathcal{O}$ of known shape, on the basis of the tactile measurements $\boldsymbol{y}^t = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t\}$ collected up to the current time instant t. The minimal pose representation of the object is given by the 6-dimensional state vector $\boldsymbol{x}$, consisting of the coordinates of the center of the reference system attached to the object and the three Euler angles representing the orientation, i.e.

$$\boldsymbol{x} = \begin{bmatrix} x, & y, & z, & \phi, & \theta, & \psi \end{bmatrix}^T. \tag{4.10}$$

The measurements are collected by touching the object with the end effector of the robot. Each measurement $\boldsymbol{y}_t$ consists of the acquired Cartesian position of the contact point, i.e.:

$$\boldsymbol{y}_t = \begin{bmatrix} x_{t,p}, & y_{t,p}, & z_{t,p} \end{bmatrix}^T. \tag{4.11}$$

Algorithm 1 The Unscented Particle Filter

for $i = 1, \ldots, N$ **do**
  draw the state particles $\boldsymbol{x}_{0|0}^{(i)}$ from the prior $p_{0|0}(\boldsymbol{x})$ and set $P_{0|0}^{(i)} = P_0$,
  and $\tilde{w}_0^{(i)} = 1/N$;
end for
for $t = 1, 2, \ldots,$ **do**
  1) UKF prediction and correction
  for $i = 1, \ldots, N$ **do**
    - **Time update**:
    given $\{\boldsymbol{x}_{t-1|t-1}^{(i)}, P_{t-1|t-1}^{(i)}\}$, compute $\{\boldsymbol{x}_{t|t-1}^{(i)}, P_{t|t-1}^{(i)}\}$ by applying
    the SUT to the state transition equation (4.7);
    - **Measurement prediction**:
    given $\{\boldsymbol{x}_{t|t-1}^{(i)}, P_{t|t-1}^{(i)}\}$, compute $\{\boldsymbol{y}_{t|t-1}^{(i)}, S_t^{(i)}, \Gamma_t^{(i)}\}$
    by applying the SUT to the measurement equation (4.8);
    - **Measurement update**: set

$$
\begin{aligned}
K_t^{(i)} &= \Gamma_t^{(i)} \left(S_t^{(i)}\right)^{-1} \\
\bar{\boldsymbol{x}}_t^{(i)} &= \boldsymbol{x}_{t|t-1}^{(i)} + K_t^{(i)} \left(\boldsymbol{y}_t - \boldsymbol{y}_{t|t-1}^{(i)}\right) \\
P_{t|t}^{(i)} &= P_{t|t-1}^{(i)} - K_t^{(i)} S_t^{(i)} \left(K_t^{(i)}\right)^T;
\end{aligned}
$$

  end for
  2) Weight update
  for $i = 1, \ldots, N$ **do**
    sample from the proposal distribution:

$$
\hat{\boldsymbol{x}}_t^{(i)} \sim q^{(i)}(\cdot|\boldsymbol{y}^t) = \mathcal{N}(\cdot; \bar{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)});
$$

    evaluate and normalize the importance weights:

$$
w_t^{(i)} = \tilde{w}_{t-1}^{(i)} \frac{\ell_t(\boldsymbol{y}_t|\hat{\boldsymbol{x}}_t^{(i)})\ \varphi_{t|t-1}(\hat{\boldsymbol{x}}_t^{(i)}|\boldsymbol{x}_{t|t-1}^{(i)})}{\mathcal{N}(\hat{\boldsymbol{x}}_t^{(i)}; \bar{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)})}
$$

$$
\tilde{w}_t^{(i)} = w_t^{(i)} / \sum_{j=1}^{N} w_t^{(j)};
$$

  end for

3) Resampling
for $i = 1, \ldots, N$ **do**
    draw $j \in \{1, \ldots, N\}$ with probability $\tilde{w}_t^{(j)}$ and set:

$$\boldsymbol{x}_{t|t}^{(i)} = \hat{\boldsymbol{x}}_t^{(j)} \quad P_{t|t}^{(i)} = P_{t|t}^{(j)} \quad \tilde{w}_t^{(i)} = \frac{1}{N}.$$

end for
end for

It is worth noticing that the exploited measurements consist only of tridimensional contact point vectors. Notice also that, while for ease of presentation it is assumed that a measurement consists of a single contact point, the proposed approach is well-suited to being extended to measurements consisting of multiple contact points (corresponding to different fingertips touching the object). This would simply amount to processing, at each time t, a measurement vector of size $3n_t$, n_t being the number of fingertips touching the object at that time. Finally, notice that, in the sequel, all measurements and the object pose will be assumed to be expressed in the same, fixed, reference system.

4.2.1 Considerations on the motion model

Since the object is assumed to be static, the 6-DOF object tactile localization problem is basically a static parameter estimation problem. In this respect, it is well known that the use of particle filtering techniques for estimating static parameters requires special care, because a direct application of these techniques to the constant state equation $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t$, corresponding to the Markov transition density $\varphi_{t+1|t}(\boldsymbol{x}|\boldsymbol{\xi}) = \delta(\boldsymbol{x} - \boldsymbol{\xi})$, would incur in the so-called weight-degeneracy phenomenon. Many solutions have been proposed in the literature to circumvent such a problem, see for instance [195] and the references therein. A simple but effective approach consists of adding an artificial dynamic noise on the static parameter by considering a state-transition equation of the form

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{\omega}_t, \tag{4.12}$$

where $\boldsymbol{\omega}_t$ is the artificial dynamic noise which is modeled as a Gaussian random variable with zero mean and suitable covariance Q_t. The idea is

that the artificial evolution provides a mechanism for generating at each time instant new particles with a sufficiently diffuse distribution. In this work, a time-invariant covariance matrix is used, i.e. $Q_t = Q$, since it proves effective in the considered case studies. However more elaborated solutions can be easily incorporated within the proposed algorithm [195].

Some considerations on the possibility of extending the approach to the case of moving object localization are provided in the subsequent Remark 2.

4.2.2 Measurement model

In order to apply the UPF to the tactile localization problem under investigation, it is necessary to define the measurement model both in terms of a likelihood function $\ell_t(\boldsymbol{y}_t|\boldsymbol{x}_t)$ and of a measurement function $\boldsymbol{h}_t(\cdot,\cdot)$. The proposed *likelihood function* is based on the so-called *proximity model*, in which the measurements are considered independent of each other and corrupted by Gaussian noise. For each measurement, the likelihood function depends on the distance between the measurement and the object, hence the name “proximity". This model is the adaptation of the likelihood proposed in [34] to the case of contact point measurements only.

Let the 3D object model be represented by a polygonal mesh consisting of faces $\{f_i\}$. For each face f_i, let $\ell_{t,i}(\boldsymbol{y}_t|\boldsymbol{x}_t)$ be the likelihood of the measurement $\boldsymbol{y}_t$ relative to that face when the object is in the pose $\boldsymbol{x}_t$. Then, the likelihood of the measurement is defined as the maximum likelihood over all faces, i.e.

$$\ell_t(\boldsymbol{y}_t|\boldsymbol{x}_t) \propto \max_i \ell_{t,i}(\boldsymbol{y}_t|\boldsymbol{x}_t), \tag{4.13}$$

apart from a normalizing factor which, however, is independent of the state $\boldsymbol{x}_t$ and needs not necessarily to be computed.

Each likelihood is assumed to be Gaussian, with variance σ_p^2, and can be computed as follows:

$$\ell_{t,i}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \frac{1}{\sqrt{2\pi}\sigma_p} \exp\left(-\frac{1}{2}\,\frac{d_i(\boldsymbol{y}_t,\boldsymbol{x}_t)^2}{\sigma_p^2}\right), \tag{4.14}$$

where the quantity $d_i(\boldsymbol{y}_t,\boldsymbol{x}_t)$ is the shortest Euclidean distance of $\boldsymbol{y}_t$ from the face f_i when the object is in the pose $\boldsymbol{x}_t$. For instance, supposing that f_i is the representation of the i-th face in the object reference system, the

distance $d_i(\boldsymbol{y}_t, \boldsymbol{x}_t)$ can be computed as

$$d_i(\boldsymbol{y}_t, \boldsymbol{x}_t) = \min_{\boldsymbol{p} \in f_i} \|\boldsymbol{y}_t^{\boldsymbol{x}_t} - \boldsymbol{p}\|,$$

where $\|\cdot\|$ is the Euclidean norm and $\boldsymbol{y}_t^{\boldsymbol{x}_t}$ denotes the transformation of the measurement $\boldsymbol{y}_t$ using the roto-translation matrix corresponding to the state $\boldsymbol{x}_t$.

Notice that the considered measurement model does not take *negative information* into account. In other words, the points of the search space exploited to compute the likelihood function are only the ones on the object surface touched during the collection of measurements, while the information provided by the lack of contact in some sub-regions of the search space is not taken into account in the likelihood function. Even if the negative information can also support object localization, it is not exploited in this method in order to keep the computational complexity moderate.

As previously pointed out, the use of the UPF requires also the definition of a *measurement function*, namely a mathematical mapping giving the measurement $\boldsymbol{y}_t$ as a function of the current state $\boldsymbol{x}_t$ and a measurement noise $\boldsymbol{v}_t$, see (4.8). For the sake of simplicity, a measurement equation with additive noise is taken into account, i.e.,

$$\boldsymbol{y}_t = \boldsymbol{h}_t(\boldsymbol{x}_t) + \boldsymbol{v}_t\,. \tag{4.15}$$

In particular, the measurement function is required to compute the Scaled Unscented Tranform (SUT) in the measurement prediction step of the Unscented Kalman Filter.

It is important to highlight how the definition of a measurement equation is different from the one of a likelihood function: given the state and the measurement noise, the measurement equation provides a measurement value - a contact point in the present case - whereas the likelihood function is proportional to the probability of having a certain measurement for a given state.

Tactile sensors are atypical sensors from this standpoint. In fact, typical sensors, e.g. radars, are characterized by a mathematical relationship between the current state of the object and the provided measurement: given

the state of the object, the measurement of the object position and orientation supplied by the sensor remains unchanged (neglecting the measurement noise).

On the other hand, the employment of tactile sensors makes the scenario quite different. The measurement is given by the tactile sensor pose itself, i.e., the forward kinematics of the end effector of the robot, only if the robot actually touches the object. Thus, if the object is in a generic state and the sensor in a specific pose, it cannot be taken for granted that such a configuration provides a contact measurement. Moreover, the sensor moves during the measurement collection, while the object is motionless. It is not possibile to predict unambiguously the measurement value without a model of the sensor motion: given the pose of the sensor and the object distance from it, the predicted measurement is not unique, since the sensor could touch the object in different points.

Nevertheless, in order to compute a predicted measurement for each possible configuration $\boldsymbol{x}$, it is necessary to define a measurement equation capable of handling also the case in which there is no actual contact between the sensor and the object in the considered pose $\boldsymbol{x}$ (in particular, the sigma point of the i-th predicted particle). Further, the predicted measurement should be consistent with the proximity-based likelihood (4.13).

To this end, it is useful to provide an alternative interpretation of the likelihood (4.13). Notice first that, due to the measurement noise, the measurement $\boldsymbol{y}_t$ does not represent the actual contact point between the sensor and the object, which however will be in the neighborhood of $\boldsymbol{y}_t$. The proximity model assumes that the actual contact point is the point on the object surface which is closest to the measurement $\boldsymbol{y}_t$. In fact, equation (4.13) can be rewritten as

$$\ell_t(\boldsymbol{y}_t|\boldsymbol{x}_t) \propto \exp\left(-\frac{1}{2\sigma_p^2} \ \|\boldsymbol{y}_t - \boldsymbol{h}_t(\boldsymbol{x}_t)\|^2\right) \tag{4.16}$$

where

$$\boldsymbol{h}_t\left(\boldsymbol{x}_t\right) = \arg\min_{\boldsymbol{p}\in\partial\mathcal{O}^{\boldsymbol{x}_t}} \|\boldsymbol{y}_t - \boldsymbol{p}\| \tag{4.17}$$

and $\partial\mathcal{O}^{\boldsymbol{x}_t}$ represents the object boundary in the pose $\boldsymbol{x}_t$ with respect to the robot reference system. Then, the likelihood of the measurement $\boldsymbol{y}_t$ depends on its distance from such a hypothetical contact point according to

a Gaussian distribution. Accordingly, given a configuration $\boldsymbol{x}_t$, the corresponding predicted measurement is selected as the point of the object surface which is closest to the measurement $\boldsymbol{y}_t$. Such a choice turns out to be consistent with the proximity likelihood model. In fact, by taking the additive measurement noise $\boldsymbol{\nu}_t$ in (4.15) as a Gaussian random variable with zero-mean and covariance $\sigma_p^2 I$, with I the identity matrix, it is an easy matter to see that (4.15) and (4.17) give rise precisely to a likelihood of the form (4.16).

4.3 The Memory Unscented Particle Filter

The main challenges of the 6-DOF tactile localization problem are its dimension (6-DOFs), its multimodal nature, and the fact that individual measurements are relatively uninformative, since they are tridimensional vectors in a 6-DOF space. In particular, the latter fact implies that the standard UPF algorithm is not well suited to this problem. In fact, Algorithm 1 uses, at each time instant t, only the current measurement $\boldsymbol{y}_t$ in order to compute the importance weights $w_t^{(i)}$. However, since a single contact point measurement is unable to completely characterize the object's pose (lack of observability), the standard weights do not provide enough information to understand which particles must be replicated and which ones must be eliminated in the subsequent resampling step. Thus, performing the standard resampling step - and then discarding some particles - on the basis of such weights is problematic: some potential representative particles could be cut off and the algorithm could limit the search to wrong sub-regions.

In order to overcome such a drawback, we propose a novel variant of the UPF, referred to as Memory UPF (MUPF). The idea is to use also past measurements to update particle weights so as to preserve their ability to characterize particle goodness. Since the object is static, all the measurements refer to the same pose and, in principle, at each time t all the measurements $\boldsymbol{y}^t$ collected up to the current time could be used to compute the importance weights. However, this solution would entail a computational effort growing in time. To avoid such a growth of complexity, the proposed approach follows a *moving window* strategy, i.e., by using, at each time instant, a sliding window consisting of the most recent m measurements. In this way, at each time instant, the weight computation requires

$O(Nm)$ likelihood evaluations, and the size m of the sliding window can be chosen according to the available computational capabilities.

In practice, the particles $\{\hat{\boldsymbol{x}}_t^{(i)}\}_{i=1}^N$ and the set of independent measurements $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t\}$, collected up to the current instant t, are used to compute the weights by:

$$w_t^{(i)} = \frac{\tilde{w}_{t-1}^{(i)} \cdot \prod_{k=\bar{k}(t)}^{t} \ell(\boldsymbol{y}_k | \hat{\boldsymbol{x}}_t^{(i)})}{\mathcal{N}(\hat{\boldsymbol{x}}_t^{(i)}; \bar{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)})}, \tag{4.18}$$

$$\tilde{w}_t^{(i)} = w_t^{(i)} / \sum_{j=1}^{N} w_t^{(j)} \tag{4.19}$$

for $i = 1, \ldots, N$, where

$$\bar{k}(t) = \begin{cases} t - m + 1, & \text{if } t - m + 1 \geq 1 \\ 1, & \text{otherwise.} \end{cases} \tag{4.20}$$

Of course, the reuse of measurements in the update of the particles' weights modifies the nature of the approximation, and hence special care needs to be taken in order to retrieve the pose estimate in a theoretically sound way. To see this, observe preliminarily that the addressed problem is inherently of a multimodal nature, since in the presence of symmetries in the object, there might exist multiple values of $\boldsymbol{x}$ compatible with the measurements. Then, taking the expected value as estimate is not meaningful. Instead, a maximum *a posteriori* probability (MAP) criterion can be followed by taking as pose estimate at time t the corrected particle $\hat{\boldsymbol{x}}_t^{(i)}$ corresponding to the highest value of the estimated posterior distribution [196].

Recalling that each corrected particle after the weight update can be considered corresponding to a Gaussian distribution with mean $\hat{\boldsymbol{x}}_t^{(i)}$ and covariance $P_{t|t}^{(i)}$, one might be tempted to take as estimated posterior $\hat{p}_{t|t}(\cdot)$ the function

$$\hat{p}_{t|t}(\boldsymbol{x}) = \sum_{i=1}^{N} \tilde{w}_t^{(i)} \mathcal{N}(\boldsymbol{x}; \hat{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)}). \tag{4.21}$$

Unfortunately, such a choice would not be theoretically sound due to the multiple use of measurements in the weight computation. In this respect, notice first that, since the object is static, the 6-DOF localization problem

is a parameter estimation problem and, hence, the true posterior $p_{t|t}(\cdot)$ at time t takes the form

$$p_{t|t}(\boldsymbol{x}) \propto \prod_{k=1}^{t} \ell(\boldsymbol{y}_k|\boldsymbol{x})\, p_0(\boldsymbol{x}), \tag{4.22}$$

where $p_0(\cdot)$ is a PDF reflecting the prior knowledge on the object configuration. Since at each time instant multiple measurements are used in the weight computation, the estimated posterior $\hat{p}_{t|t}(\cdot)$ does not approximate the true one $p_{t|t}(\cdot)$ but instead it approximates the PDF

$$\tilde{p}_{t|t}(\boldsymbol{x}) \propto \prod_{k=1}^{\bar{k}(t)-1} \ell(\boldsymbol{y}_k|\boldsymbol{x})^m \prod_{k=\bar{k}(t)}^{t} \ell(\boldsymbol{y}_k|\boldsymbol{x})^{t+1-k}\, p_{0|0}(\boldsymbol{x}), \tag{4.23}$$

where $p_{0|0}(\cdot)$ is the prior density used in the generation of the initial particles, thus introducing an undesired warp in the form of the estimated posterior PDF.

This drawback can be circumvented by computing special weights $\bar{w}_t^{(i)}$, used only for the purpose of pose estimation extraction but not propagated in the recursion. In fact, by setting

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)} \cdot \prod_{k=\bar{k}(t)}^{t} \ell(\boldsymbol{y}_k|\hat{\boldsymbol{x}}_t^{(i)})^{m-t+k-1}}{\mathcal{N}(\hat{\boldsymbol{x}}_t^{(i)}; \bar{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)})}, \tag{4.24}$$

$$\bar{w}_t^{(i)} = w_t^{(i)} \Big/ \sum_{j=1}^{N} w_t^{(j)} \tag{4.25}$$

for $i = 1, \ldots, N$, and using the estimated posterior

$$\hat{p}_{t|t}(\boldsymbol{x}) = \sum_{i=1}^{N} \bar{w}_t^{(i)}\, \mathcal{N}(\boldsymbol{x}; \hat{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)}) \tag{4.26}$$

in place of (4.21), it turns out that such a $\hat{p}_{t|t}(\cdot)$ approximates the PDF

$$\bar{p}_{t|t}(\boldsymbol{x}) \propto \prod_{k=1}^{t} \ell(\boldsymbol{y}_k|\boldsymbol{x})^m\, p_{0|0}(\boldsymbol{x}) \tag{4.27}$$

so that all measurements provide the same contribution to the estimation problem. Then, by choosing $p_{0|0}(\boldsymbol{x}) \propto p(\boldsymbol{x}_0)^m$, we obtain $\bar{p}_{t|t}(\boldsymbol{x}) \propto p_{t|t}^m(\boldsymbol{x})$

which implies that $\bar{p}_{t|t}(\boldsymbol{x})$ and $p_{t|t}(\boldsymbol{x})$ share the same maximum points. In turn, this implies that application of the MAP estimation criterion to $\bar{p}_{t|t}(\boldsymbol{x})$ is equivalent to computing the MAP estimate according to $p_{t|t}(\boldsymbol{x})$. These considerations allow concluding that, with the choice $p_{0|0}(\boldsymbol{x}) \propto p_0(\boldsymbol{x})^m$, the MAP estimate $\hat{\boldsymbol{x}}_t$ corresponding to the particle with the maximum a posteriori probability according to (4.26)

$$\hat{\boldsymbol{x}}_t = \arg\max_j \hat{p}_{t|t}(\hat{\boldsymbol{x}}_t^{(j)}) = \tag{4.28}$$

$$= \arg\max_j \sum_{i=1}^{N} \bar{w}_t^{(i)} \mathcal{N}(\hat{\boldsymbol{x}}_t^{(j)}; \hat{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)}). \tag{4.29}$$

is coherent with the true posterior PDF.

Remark 1 *The fact that (4.26) approximates (4.27) can be shown by noting that $\bar{p}_{t|t}(\boldsymbol{x})$ can be decomposed as follows*

$$\begin{aligned}
\bar{p}_{t|t}(\boldsymbol{x}) &\propto \prod_{k=\bar{k}(t)}^{t} \ell(\boldsymbol{y}_k|\boldsymbol{x})^{m-t+k-1} \\
&\times \prod_{k=\bar{k}(t)}^{t} \ell(\boldsymbol{y}_k|\boldsymbol{x})^{t+1-k} \prod_{k=1}^{\bar{k}(t)-1} \ell(\boldsymbol{y}_k|\boldsymbol{x})^{m}\, p_{0|0}(\boldsymbol{x}) \\
&= \prod_{k=\bar{k}(t)}^{t} \ell(\boldsymbol{y}_k|\boldsymbol{x})^{m-t+k-1}\, \tilde{p}_{t|t}(\boldsymbol{x})
\end{aligned}$$

which precisely corresponds to the weight update in (4.24).

A further modification, as compared to the standard UPF, pertains to the resampling step. Since in the first iterations only few measurements are available (thus providing insufficient information), all the particles are retained so as to account for more possibile solutions, in accordance with the multimodal nature of the problem. This amounts to skipping the standard resampling step for a certain number t_0 of initial time instants (in the experimental results reported in the following sections, for the first two time instants). The degeneration of the weights in the first iterations is avoided by setting the weights of all particles equal to $1/N$.

Summing up, the proposed MUPF algorithm is shown in Algorithm 2. The term *Memory*, in the name of the proposed algorithm, is due to the computation of the weights: at each iteration a non-decreasing number of

Algorithm 2 The Memory Unscented Particle Filter

for $i = 1, \ldots, N$ **do**
  draw the state particles $\boldsymbol{x}_{0|0}^{(i)}$ from the prior $p_{0|0}(\boldsymbol{x})$
  and set $P_{0|0}^{(i)} = P_0$ and $\tilde{w}_{0|0}^{(i)} = 1/N$;
end for
for $t = 1, 2, \ldots$ **do**
  1) UKF prediction and correction
  for $i = 1, \ldots, N$ **do**
    - **Time update**: set $\boldsymbol{x}_{t|t-1}^{(i)} = \boldsymbol{x}_{t-1|t-1}^{(i)}$ and $P_{t|t-1}^{(i)} = P_{t-1|t-1}^{(i)} + Q$;
    - **Measurement prediction**: like in Algorithm 1;
    - **Measurement update**: like in Algorithm 1;
  end for
  2) Weight update
  for $i = 1, \ldots, N$ **do**
    sample from the proposal distribution:

$$\hat{\boldsymbol{x}}_t^{(i)} \sim q^{(i)}(\cdot|\boldsymbol{y}^t) = \mathcal{N}(\cdot; \bar{\boldsymbol{x}}_t^{(i)}, P_{t|t}^{(i)}),$$

    evaluate and normalize the modified importance weights via (4.18) and (4.19);
  end for

  3) Estimated pose extraction (optional)
  for $i = 1, \ldots, N$ **do**
    evaluate and normalize the importance weights via (4.24) and (4.25);
  end for
  compute the estimated pose $\hat{\boldsymbol{x}}_t$ via (4.28);

4) Resampling
for $i = 1, \ldots, N$ **do**
  if $t > t_0$ **then**

    draw $j \in \{1, \ldots, N\}$ with probability $\tilde{w}_t^{(j)}$,
    then set:

$$\boldsymbol{x}_{t|t}^{(i)} = \hat{\boldsymbol{x}}_t^{(j)} \qquad P_{t|t}^{(i)} = P_{t|t}^{(j)} \qquad \tilde{w}_t^{(i)} = \frac{1}{N}.$$

  else
    set :

$$\boldsymbol{x}_{t|t}^{(i)} = \hat{\boldsymbol{x}}_t^{(i)} \qquad \tilde{w}_t^{(i)} = \frac{1}{N}.$$

  end if
end for
end for

measurements is exploited to evaluate the likelihood function. Notice also that the computation of the weights $\bar{w}_t^{(i)}$ is optional (since they are not used in the time propagation from t to $t+1$) and can be limited only to those time instants in which one wants to extract an estimate $\hat{\boldsymbol{x}}_t$ of the object's pose from the approximated posterior.

Remark 2 *While the considered framework deals with static objects, the proposed algorithm is well-suited to being extended to the case of moving objects since it is based on Bayesian filtering and is inherently recursive in nature. When the object is not static, however, the use of a sliding window of the most recent measurements in the weight computation requires some caution because the past measurements do not refer to the current pose. In principle, this problem can be circumvented by considering particle states consisting of the whole object trajectory in the sliding window (similarly to what happens in particle-filtering-based solutions to the SLAM problem) so that the likelihood, with respect to the measurements in the sliding window, can be correctly computed. Further, when the object is static, a simple motion model like (4.12) makes sense only to model small random movements caused by probing. For truly moving objects (for example a rolling ball), more complex motion models are required including also the object velocity. Of course, the main challenge in this case is the increased complexity due to such modifications. Such generalizations are left for future research.*

4.4 Algorithm validation

In order to evaluate the performance of the developed Memory Unscented Particle Filter (MUPF), a C++ implementation of MUPF has been tested via simulations on different objects and collections of measurements. The tests have been run on a Linux platform, with a quadcore 3.40 GHz processor. The developed code, the exploited measurements and the reconstructed object models can be downloaded from *github*[1].

4.4.1 Simulation setup

The simulation setup consists of five objects: a rectangular box, a tetrahedron, a cleaner spray, a robot toy and a safety helmet (Fig. 4.1).

The mesh models of the first two objects, having a simple geometrical shape, are built from ruler measurements whereas the other three more complicated objects are approximated by triangular mesh models, reconstructed via image processing algorithms. In particular, the mesh models of the cleaner spray and safety helmet are obtained from 360 degree point clouds reconstructed with the RTM toolbox[2] [197]. The RTM toolbox merges together several partial 3D models - i.e. different views of the object - captured by rotating the object in front of a RGB-D camera and, in a few seconds, provides a 360 degree point cloud of the object. Conversely, the more complex point cloud of the robot toy is retrieved by making use of the AutoDesk 123d catch application[3] that, in several tens of minutes, processes different object photos taken from different views with a smartphone. Thus, the triangular mesh models of the three objects are extracted by applying the Poisson Surface Reconstruction algorithm [198] to the merged point clouds. The complete pipeline for model reconstruction is outlined in Fig. 4.2.

The contact point measurements exploited in the simulation tests are drawn by non-uniformly sampling random points on a subset of 3D model faces.

The MUPF algorithm requires setting the following parameters: the

[1] DOI:10.5281/zenodo.163860.

[2] Recognition Tracking and Modelling of Objects, by ACIN of Technische Universität Wien, http://www.acin.tuwien.ac.at/forschung/v4r/software-tools/rtm/.

[3] http://www.123dapp.com/catch.

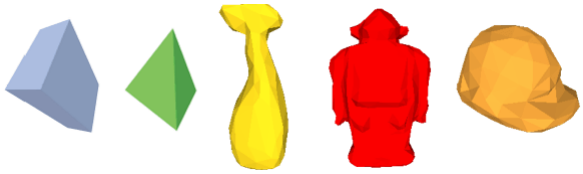


FIGURE 4.1: Simulation setup objects. From left to right: a rectangular box ($0.1 \times 0.3 \times 0.2$ [m]), a tetrahedron (equilateral triangular basis with the side of 0.33 [m] $\times$ height of 0.2 [m]), a cleaner spray (approximately $0.23 \times 0.08 \times 0.05$ [m]), a robot toy ($0.23 \times 0.09 \times 0.06$ [m]), and a safety helmet (nearly a half-sphere with radius 0.1 [m]).

FIGURE 4.2: Pipeline for real object modelling. From left to right: real objects, 360 degree point clouds (obtained with a RGB-D camera and the RTM toolbox for the cleaner spray and the safety helmet, and with 40 photos from different views and the Autodesk 123d catch app for the robot toy), triangular mesh models matching the point clouds, computed by using the Poisson surface reconstruction. On the top: the cleaner spray, whose mesh model consists of 250 faces. In the middle: the safety helmet, featured by a mesh model of 250 faces. On the bottom: the robot toy, whose mesh model is made up of 750 faces.

artificial process noise covariance matrix Q; the measurement noise covariance σ_p^2 characterizing sensor accuracy; the initial covariance matrix P_0 to quantify the initial uncertainty and, hence, the extent of the search region; the parameters of the unscented transformation α, β, k; the number of particles N; the length m of the measurement window for the importance weight update.

As preliminary tests, the parameters are kept constant, as shown in Table 4.1. In particular, the chosen matrix Q is such that the artificial process disturbance spreads the particles with standard deviations of 1 cm in position and about 5 degrees in rotation. Conversely, the covariance σ_p^2 assumes that the measurements of the end-effector position are affected by an error with standard deviation of 1 cm in all Cartesian coordinates. Finally, the initial matrix P_0 indicates an initial uncertainty with standard deviation of 0.2 [m] for the position along the three coordinates and respectively $\pi, \pi/2, \pi$ for the three orientation angles ϕ, θ, ψ. The initial particles $\boldsymbol{x}_{0|0}^{(i)}$ for $i = 1, \ldots, N$ are drawn from the prior distribution $\mathcal{N}(\cdot; \boldsymbol{x}_0, P_0)$, where $\boldsymbol{x}_0$ is arbitrarily chosen (a 6D null vector in our tests). The choices of Table 4.1 have proven effective in all the considered simulations, thus indicating that the proposed algorithm works over a broad range of problems without a case-by-case parameter tuning.

It is worth pointing out how the exploitation of the UKF step in the UPF allows to considerably reduce the number of particles to $N = 700$ (with a standard particle filter it would be in the order of $N = 10^6$ for a 6-DOF problem). Section 4.4.6 provides a detailed analysis about the parameters influence on MUPF performance.

TABLE 4.1: Parameter set for the MUPF.

Q	diag([10^{-5}, 10^{-5}, 10^{-5}, 10^{-4}, 10^{-4}, 10^{-4}]) [m^2], [rad^2]
P_0	diag([0.04, 0.04, 0.04, π^2, $(\pi/2)^2$, π^2]) [m^2], [rad^2]
σ_p^2	10^{-4}[m^2]
α	1
k	2
β	30
N	700

4.4.2 Performance evaluation

The performance of the proposed algorithm is assessed in terms of both effectiveness and execution time, since the ultimate aim of this work is a real-time application of the algorithm.

In this respect, algorithm *reliability* is measured in terms of number of successes among trials, where a trial is considered failed whenever the estimated pose is substantially different from the real one.

In simulation tests, successes and failures can be discriminated by computing the distance between the estimated and the true object poses, since the knowledge of the latter is available. The situation is different in real experiments, wherein the true pose is often difficult (if not impossible) to be measured. In this case, the distinction between a successful or a failed trial is necessarily accomplished by the user by visually inspecting that the solution found by the algorithm is consistent with the real pose of the object. In the successful cases, a numerical evaluation of the localization can be done by relying merely on measurements without the need of the ground truth. This choice is by far preferable (sometimes the only viable solution) for an experimental assessment.

These considerations suggest the definition of the following *performance index*:

$$\mathcal{I}_L = \frac{1}{L}\sum_{i}^{L} d_i, \tag{4.30}$$

where L is the total number of collected measurements and d_i the distance between the i-th measurement and the object in the estimated pose. In other words, given the set of measurements and the estimated pose, the proposed performance index is the average of the distances between each measurement and the object in the estimated pose. It is worth highlighting that the performance index (4.30) is not in the standard least-square fit form in that it is a sum of errors (not of squared errors) and each error term d_i in (4.30) is a complicated nonlinear distance function $G(y, x)$ of two arguments (a contact point measurement y and the object pose x), not expressible in the classical residual form $y - g(x)$ of best-fit problems.

The performance index $\mathcal{I}_L$ has been adopted to evaluate the localization quality in simulation (together with the standard localization error measured as distance of the final estimated pose from the ground truth) and experimental tests, for the reasons listed below. First, the index $\mathcal{I}_L$ is the

FIGURE 4.3: On the left: a robot toy in the real pose. On the right: two different estimated poses, both featured by a performance index of $0.008[m]$ with respect to the set of measurements, coloured in black. The green one corresponds to the correct pose, whereas the red one is a local minimum, representing a completely wrong pose, but anyway consistent with the measurements.

only viable solution for the experimental tests, wherein the real pose cannot typically be known or measured with sufficient accuracy. Secondly, the use of a common error index for both simulation and experimental tests, makes easier the comparison between the two cases. Third, if simulation tests are carried out with noiseless measurements and a sufficient number of informative measurements is collected, then the performance index $\mathcal{I}_L$ can be related to the distance between the estimated and the true object poses, in the sense that $\mathcal{I}_L$ vanishes for large L if and only if the two poses coincide. Finally, the index $\mathcal{I}_t$ is easily computable on-line at each time t and could therefore be monitored in order to understand when to stop localization of the current object. As a further benefit, (4.30) provides a synthetic (scalar) indicator of the pose error, in terms of linear displacement (measured in units of length). Thus, the index computation is not affected by the problems related to the computation of angular displacements.

Nevertheless, it is worth pointing out that when the measurements are too inaccurate, the index (4.30) can be non-informative and the evaluation of the algorithm performance would necessarily require the ground truth object pose. In fact, if measurements are very noisy, the computed performance index might be low even if it is associated to local minima and corresponds to a completely wrong localization (Fig 4.3).

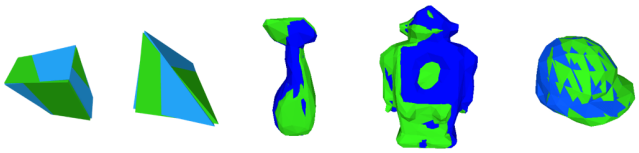


FIGURE 4.4: MUPF simulation results: the real poses are coloured in blue, whereas the estimated ones, featured by an error index of 0.002 [m], are coloured in green.

4.4.3 Simulation results

Table 4.2 provides, for each considered object, the following metrics averaged over 50 independent trials of the MUPF: standard localization error in both position and orientation, performance index $\mathcal{I}_L$ defined in (4.30), execution time and reliability. Table 4.3 reports the total number of measurements L and the MUPF window size m used for each object. The true object poses are fixed over trials and differ from the 6D null vector in translation (from 0.05 up to 0.1 [m] along one axis) and orientation (from 45 up to 90 degrees with respect to one axis).

It is worth underlining how, when an adequate choice of m is adopted (see Fig. 4.5 and 4.6), the localization errors averaged over trials are small (e.g. the index $\mathcal{I}_L$ is less than $2\,[mm]$, see Fig. 4.4), the execution time is acceptable and the reliability is high. In Fig. 4.5, the behavior of the performance index $\mathcal{I}_L$ is shown as a function of the memory m ranging from 1 to L (the total number of available measurements). Such plots highlight how MUPF is capable of solving the problem even with small m ($1 < m \ll L$) whereas the standard UPF (i.e. MUPF with $m = 1$) doe not converge at all. In addition, Fig. 4.6 demonstrates that the algorithm is reliable even with small values of m (provided that $m > 1$).

TABLE 4.2: Simulation results for the MUPF algorithm.

Object	Standard error [deg], [m]	$\mathcal{I}_L$ [m]	Time [s]	Succ./Trials
Box	0.30 - 0.0036	0.0025	1.61	50/50
Tetra.	17.1 - 0.0061	0.0021	3.63	50/50
Cleaner	0.78 - 0.0027	0.0025	7.32	50/50
Robot	19.6 - 0.0072	0.0021	3.95	50/50
Helmet	0.06 - 0.0023	0.0017	4.82	50/50

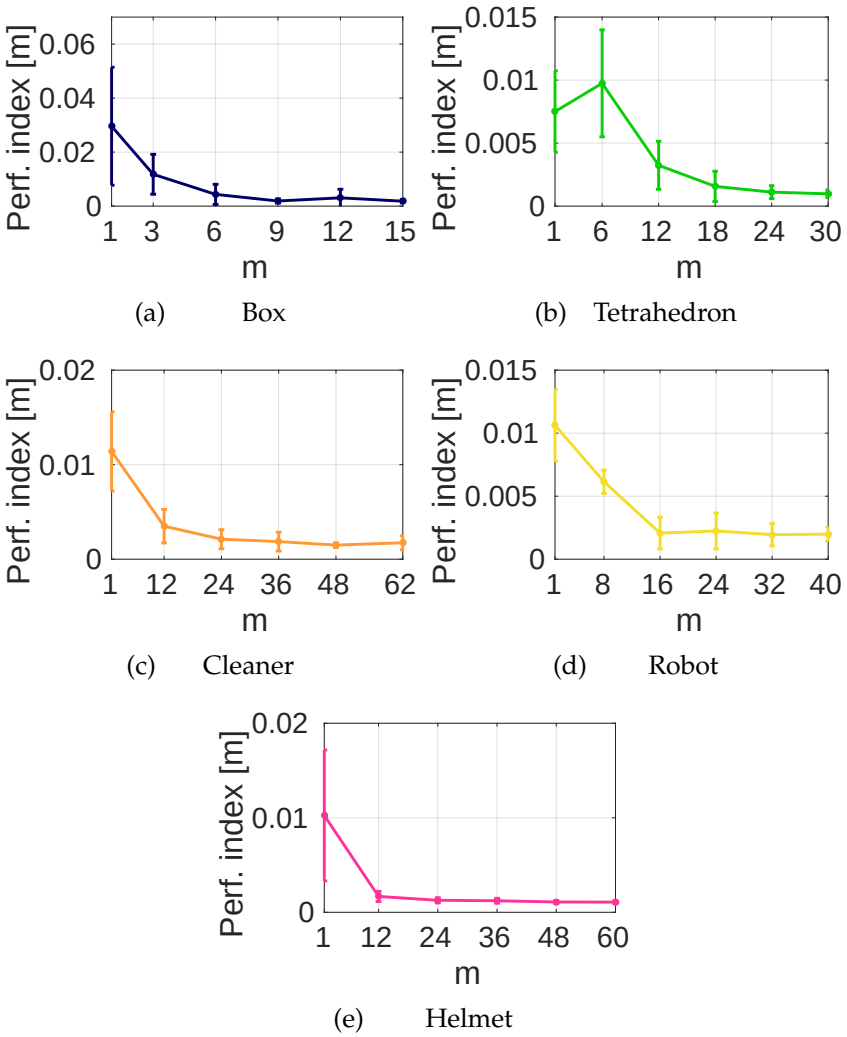


FIGURE 4.5: MUPF simulation results: average performance index on fifty trials by varying m, ranging from 1 up to the total number of measurements L.

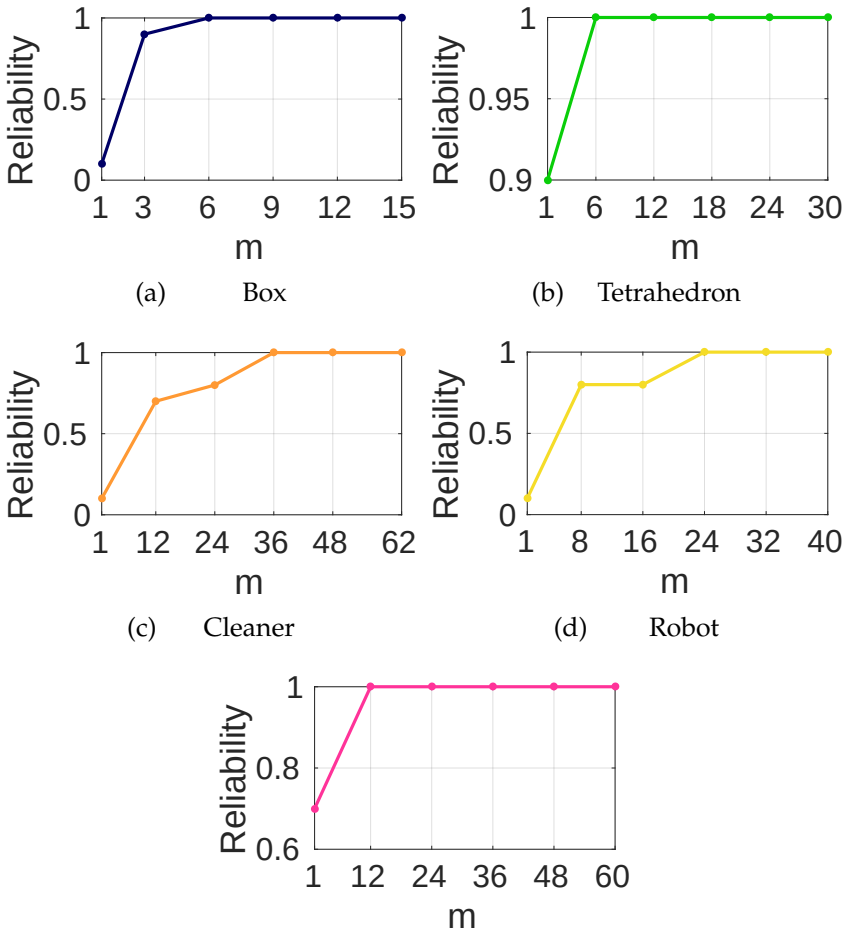


FIGURE 4.6: MUPF simulation results: reliability on fifty trials by varying m, ranging from 1 up to the total number of measurements L.

TABLE 4.3: Simulation results: measurements and m values.

Object	L	m	Object	L	m
Box	15	9	Tetra.	30	12
Cleaner	62	36	Robot	40	24
Helmet	60	36			

For the sake of comparison, a simple batch baseline, the *Iterative Closest Point (ICP)* algorithm [199], and a state-of-art approach, the *Scaling Series* algorithm presented in [34] specifically for tactile localization, have been applied to the same simulation scenario.

In order to adapt ICP, which is originally designed for shape registration, to the tactile localization problem, two point clouds are considered: one consisting of the measurements, and the other representing the object model in the right pose. To this end, suitable models have been obtained by sampling 1000 points on the object mesh models of Fig. 4.1. However, it was found that a standard implementation of ICP does not converge in such a scenario. ICP fails because there is a large uncertainty in the object initial pose. Lacking a good initial guess close to the optimal solution, ICP gets trapped in local minima.

The results obtained with the Scaling Series are reported in Table 4.4 for the same sets of measurements used in the MUPF simulation tests (Table 4.3). For the sake of conciseness, only the values of the performance index $\mathcal{I}_L$ are shown.

TABLE 4.4: Simulation results for the Scaling Series algorithm.

Object	$\mathcal{I}_L$ [m]	Time - Max. Time [s]	Successes/Trials
Box	0.001	3.47 - 13.2	45/50
Tetra.	0.001	0.05 - 1.03	50/50
Cleaner	0.006	0.03 - 5.64	42/50
Robot	0.003	0.02 - 3.64	43/50
Helmet	0.005	0.04 - 4.20	32/50

Notice that the execution time of the Scaling Series algorithm significantly changes over the trials as the algorithm generates quite different

numbers of particles from trial to trial. Hence, Table 4.4 reports both *average* and *maximum* (worst-case) execution times. Nevertheless, the Scaling Series algorithm proves to be relatively faster than MUPF. In terms of localization precision in the successful trials, the MUPF and Scaling Series algorithms exhibit comparable results. It is worth pointing out, however, that in a non negligible number of trials the Scaling Series algorithm diverged and failed to find a solution. The low reliability of the Scaling Series algorithm can be attributed to the automatic process of particle generation. Particularly, a rough parameter tuning can easily lead to the generation of an insufficient number of particles or, on the contrary, to their exponential growth, thus precluding the final convergence of the algorithm. This is somewhat surprising as MUPF has always been executed with the same parameters, whereas the parameters of the Scaling Series algorithm have been specifically tuned to each case in order to achieve better performance. In summary, MUPF turned out to be more reliable than the Scaling Series algorithm.

An extensive evaluation of the MUPF algorithm is performed by tackling the 6-DOF tactile localization problem for real objects via actual tactile measurements. For these experiments, the employed code implementation and hardware computing platform are the same ones exploited for the simulation tests.

4.4.4 Experimental setup

Four everyday objects are considered: two toys, the cleaner spray and the robot toy. The experimental tests on the safety helmet are not shown since many local minima, corresponding to different poses and featured by the same localization error, are wrongly given as possible solutions. The reasons of this behaviour will be explained in detail in Section 4.4.5. The mesh models of the first two objects are reconstructed from ruler measurements (Fig. 4.7), since they are well-represented by geometrical solid figures. The cleaner spray and robot toy mesh models are the same ones exploited for the simulation tests. Note that in order to avoid object's slip caused by the robot's movements, each object was strictly fixed to a support during measurement collection.

FIGURE 4.7: Mesh models of real geometric objects. On the left: cylindrical tube, with a diameter of 0.06 $[m]$ and height of 0.2 $[m]$, 144 triangular faces. On the right: a Lego object, made up of three parallelepipeds (total dimensions of $0.2 \times 0.1 \times 0.2$ $[m^3]$), 36 triangular faces.

The platform used for the collection of tactile measurements is the iCub humanoid robot [2]. Tactile measurements are supplied by fingertips on the iCub hands (see Chapter 3), that are covered with capacitive tactile sensors capable of providing accurate contact point measurements, once contact with the object is detected. Due to the object complexity, tactile measurements are collected through a user-guided strategy, consisting of predefined points approximately located around the objects. This strategy was necessary since a completely blind exploration of the objects turned out to be unfeasible and often caused the robot to hit the object with part of the hand not covered with sensors. It is important to remark that, for this work, the final goal of the experimental tests is the extensive evaluation of the proposed MUPF algorithm through realistic measurements, without focusing on the design of an autonomous measurement collection strategy.

Before providing experimental results, it is worth discussing the main sources of measurement uncertainty, in order to better appreciate the performance of the proposed algorithm and to understand how to set the parameters. In this respect, one relevant source of uncertainty is given by the tactile sensors themselves. The contact point measurement, in fact, is given by the kinematics of one of the fingers and the supplied x, y, z coordinates are affected by calibration offsets. In addition to this, the kinematics provides the x, y, z coordinates of the center of the fingertip. Thus, the retrieved point is always the center of the fingertip even if the tactile taxel activation - and thus the contact detection - has taken place on the

extremity or on the side of the fingertip. Taking into account all these considerations, tactile measurements were empirically estimated to be affected by a noise with standard deviation of 0.015 $[m]$. Such sources of error and uncertainty suggest to choose a slightly larger variance $\sigma_p^2 = 4\ 10^{-4}\ [m^2]$ in order to characterize iCub tactile sensor accuracy. The other MUPF parameters used for the experimental tests are shown in Table 4.1, except for the number of particles N, set equal to 1200 in the experimental tests unless differently specified.

4.4.5 Experimental results

In Tables 4.5 and 4.6, the average performance index, along with the execution time and the algorithm reliability are provided for fifty trials of both the MUPF and Scaling Series algorithms on the four considered objects. The results obtained with the ICP algorithm are not shown due to the lack of convergence. In addition, only the performance index $\mathcal{I}_L$ is computed in the real experiments, where the true pose is difficult to be measured. Figs. 4.8 and 4.9 show the average performance index and the reliability on fifty trials by varying m, ranging from 1 (standard UPF) up to the total number of measurements $m = L$.

TABLE 4.5: Experimental results for the MUPF.

Object	$\mathcal{I}_L$ [m]	Time [s]	Successes/Trials	L	m
Lego toy	0.0090	12.8	46/50	55	55
Cylinder	0.0063	6.71	50/50	30	18
Cleaner	0.0090	13.7	50/50	62	30
Robot	0.0054	12.3	43/50	60	36

The experimental tests confirm the MUPF behavior exhibited in the simulation tests, even if the experimental solutions are unavoidably affected by a slightly worse performance index, due to the high measurement noise (Fig. 4.10). The measurement noise is also responsible for the deterioration of algorithm reliability for the Lego and robot toys. This effect can be ascribed to the fact that the measurement noise is comparable with the dimension of the distinctive details of these two objects. In fact, the distinction between a good or a wrong solution is strongly influenced

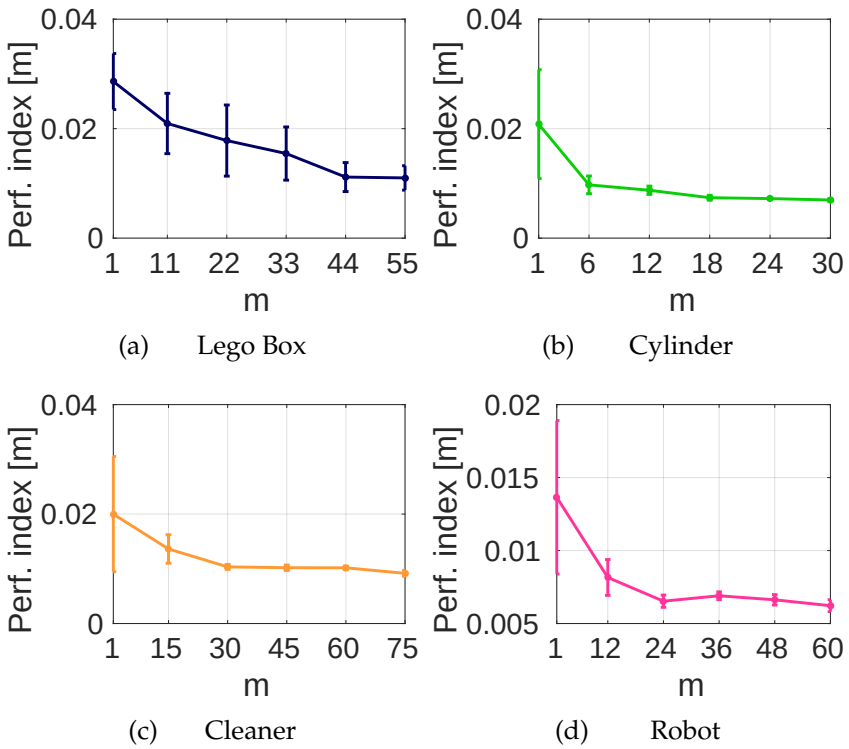


FIGURE 4.8: MUPF experimental results: average performance index on fifty trials by varying m, ranging from 1 up to the total number of measurements L.

TABLE 4.6: Experimental results for the Scaling Series algorithm.

Object	$\mathcal{I}_L$ [m]	Time/ Max. Time [s]	Successes/Trials
Lego toy	0.0073	5.03 - 29.71	40/50
Cylinder	0.0059	4.02 - 13.22	40/50
Cleaner	0.0139	4.02 - 13.22	23/50
Robot	0.0027	0.81 - 8.72	43/50

by the object details, since the only exploited information consists of tridimensional points, without taking advantage of surface normals. In such scenarios, a measurement noise of the same entity of the detail dimensions prevents the user from localizing the object even via visual inspection. As mentioned above, this is also the reason why experimental tests on the safety helmet are not shown: due to the strongly symmetric shape and the measurement noise, the measurements are not informative enough in the sense that there are many different poses compatible with the measurements (i.e. corresponding to local minima).

On the contrary, the Scaling Series performance turns out to be much worse compared to what reported in the simulation tests, particularly in terms of reliability. The failures of the Scaling Series algorithm are mainly caused by the generation of an insufficient number of particles. Often, it is not simple to set the Scaling Series parameters so that the number of generated particles is sufficient to reliably localize the objects. This shows how parameter tuning can actually be a weakness of the Scaling Series approach.

4.4.6 Further analysis

In this section, additional results are provided, with the aim of better analyzing MUPF performance.

First, the algorithm robustness has been tested by varying some algorithm parameters, such as the covariance Q of the artificial process noise and the number of particles N. The box-plots of Figs. 4.11(a) and 4.11(b) point out how the performance index and reliability are not significantly affected by varying the covariance matrix Q. Fifty trials of the MUPF have

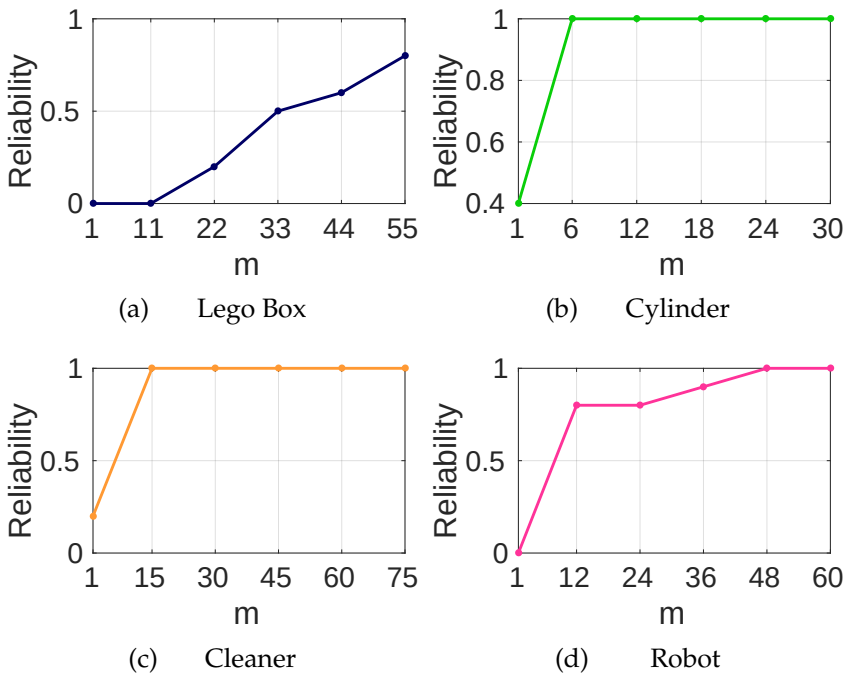


FIGURE 4.9: MUPF experimental results: reliability on fifty trials by varying m, ranging from 1 up to the total number of measurements L.

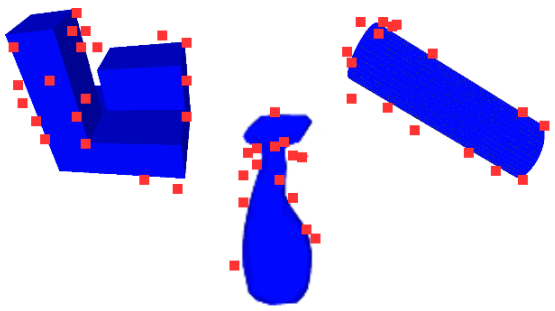


FIGURE 4.10: MUPF experimental results: tactile measurements are coloured in red, the estimated poses (performance index of 0.008 [m]) in blue.

TABLE 4.7: Q matrices used in the tests.

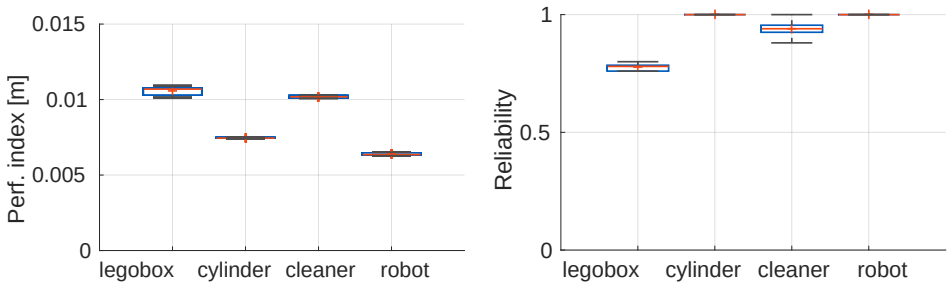
	Simulated tests		
Q_1	diag($[10^{-6}, 10^{-6}, 10^{-6}, 10^{-5}, 10^{-5}, 10^{-5}]$)	[m^2],	[rad^2]
Q_2	diag($5\times[10^{-6}, 10^{-6}, 10^{-6}, 10^{-5}, 10^{-5}, 10^{-5}]$)	[m^2],	[rad^2]
Q_3	diag($[10^{-5}, 10^{-5}, 10^{-5}, 10^{-4}, 10^{-4}, 10^{-4}]$)	[m^2],	[rad^2]
Q_4	diag($5\times[10^{-5}, 10^{-5}, 10^{-5}, 10^{-4}, 10^{-4}, 10^{-4}]$)	[m^2],	[rad^2]
Q_5	diag($[10^{-4}, 10^{-4}, 10^{-4}, 10^{-3}, 10^{-3}, 10^{-3}]$)	[m^2],	[rad^2]
	Experimental tests		
Q_1	diag($[10^{-6}, 10^{-6}, 10^{-6}, 10^{-4}, 10^{-4}, 10^{-4}]$)	[m^2],	[rad^2]
Q_2	diag($5\times[10^{-6}, 10^{-6}, 10^{-6}, 10^{-4}, 10^{-4}, 10^{-4}]$)	[m^2],	[rad^2]
Q_3	diag($[10^{-5}, 10^{-5}, 10^{-5}, 10^{-3}, 10^{-3}, 10^{-3}]$)	[m^2],	[rad^2]
Q_4	diag($5\times[10^{-5}, 10^{-5}, 10^{-5}, 10^{-3}, 10^{-3}, 10^{-3}]$)	[m^2],	[rad^2]
Q_5	diag($[10^{-4}, 10^{-4}, 10^{-4}, 10^{-2}, 10^{-2}, 10^{-2}]$)	[m^2],	[rad^2]

been carried out for five different Q matrices shown in Table 4.7 (a total of 5×50 trials). The performance index $\mathcal{I}_L$ and reliability averaged over the 50 trials - 5 values for each object - are used in building each box. The boxplots of Fig. 4.11 show the performance obtained with real measurements.

Fig. 4.12 shows the influence of the number of particles N on MUPF performance, in terms of localization error and reliability when real measurements are exploited. Performance deteriorates for $N \leq 400$, while slight changes have been found by varying N from 600 up to 1200 (with the exception of the legobox, see Fig. 4.12 (b)).

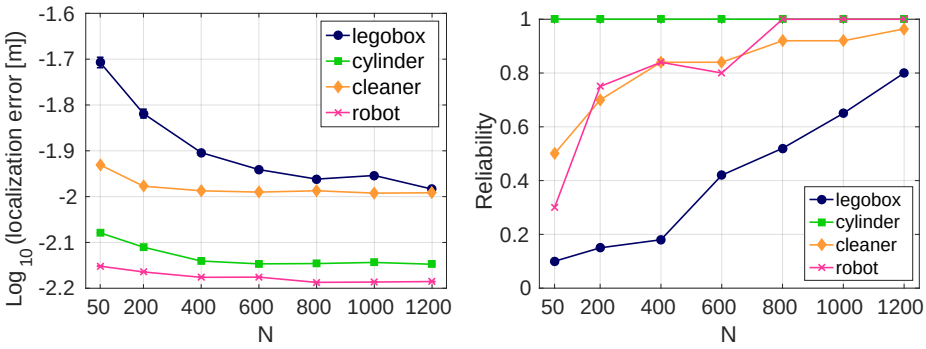
Secondly, MUPF execution time has been studied by varying the number of particles N and the MUPF window size m. Figs. 4.13(a) - 4.13(d) and 4.13(e) show the average execution time over fifty trials versus m (with N=1200) and, respectively, N (with m=L), in the case of real measurements.

Finally, given the recursive nature of the algorithm, it is worth to analyze the evolution of the performance index $\mathcal{I}_t$ during the MUPF iterations in order to check if it could be used as an appropriate stopping criterion for recursive, on-line localization. Fig. 4.14 shows how the index $\mathcal{I}_t$ evolves in time, i.e. while new measurements are being processed. It turns out that, after a burn-in period, $\mathcal{I}_t$ quickly converges to a small value. This suggests that the localization could be terminated whenever the addition of a new



(a) (b)

FIGURE 4.11: MUPF robustness analysis: (a) performance index and (b) reliability (number of successes among trials) on fifty trials for 5 different Q matrices, shown in Table 4.7.



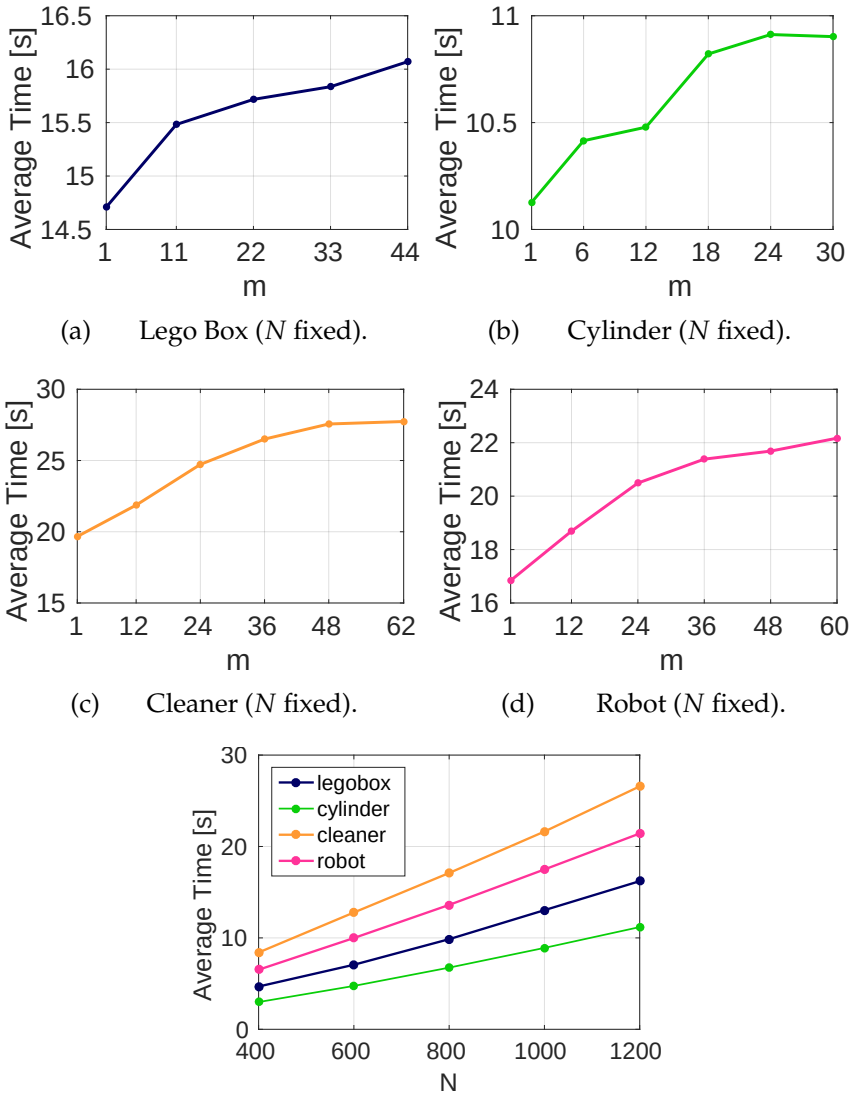
(a) (b)

FIGURE 4.12: MUPF performance analysis on fifty trials by varying N from 50 up to 1200 with real measurements, in terms of localization error (a) and reliability (b).

measurement (or a sequence of measurements) does not corresponds to a significant reduction of $\mathcal{I}_t$.

4.5 Discussion

The proposed solution to the 6-DOF tactile localization is based on a novel recursive Bayesian estimation algorithm, the Memory Unscented Particle Filter (MUPF). In contrast to optimization techniques, Bayesian filtering turns out to be a successful approach to account for noisy sensors and inaccurate models. A further advantage of the Bayesian approach is that it can be naturally extended to consider the case in which the object moves, by introducing a suitable probabilistic model for the object motion. The multimodal nature of the problem makes particle filtering techniques more



(a) Lego Box (N fixed).

(b) Cylinder (N fixed).

(c) Cleaner (N fixed).

(d) Robot (N fixed).

(e) Comparison between all objects (m fixed).

FIGURE 4.13: Average execution time on 50 trials (a) - (d) by varying m (with $N = 1200$) and (e) by varying N (with m equal to the maximum value, i.e. $m = L$).

suitable for tactile localization than nonlinear Kalman filtering approaches. However, the exploitation of standard particle filtering for 6-DOF tactile localization would require a number of particles in the order of 10^6 which, in turn, might entail an unaffordable computational load for real-time operation.

The proposed MUPF algorithm is capable of localizing tridimensional objects through tactile measurements with good overall performance and

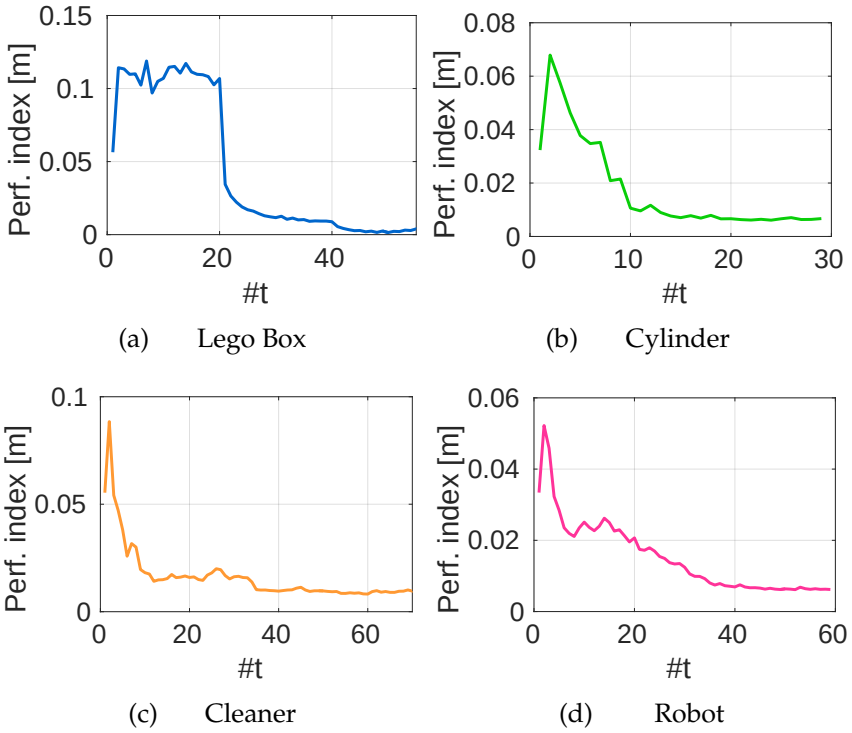


FIGURE 4.14: Performance index trend at each algorithm time step (with real measurements). After a burn-in period, the performance index decreases and converges to an asymptotic value.

by exploiting a reduced number of particles (in the order of hundreds). The MUPF algorithm relies on the Unscented Particle Filter suitably adapted to the localization problem of interest. The Unscented Particle Filter jointly exploits the potentials of the particle filter for approximating multimodal distributions and of the unscented Kalman filter for efficiently generating the proposal distribution. It is worth to point out that, for measurement update purposes, the particle filter requires a probabilistic sensor description in terms of likelihood function while the unscented Kalman filter needs a measurement function allowing to predict the measurement given the estimated state. In the specific problem of interest, it is quite natural to characterize the tactile sensor in terms of likelihood (i.e. probability distribution of the sensed contact point given the object pose) while it is clearly not possible to uniquely predict the sensed contact point given the estimated object pose. To circumvent this difficulty and be able to apply UPF to tactile localization, the following idea has been pursued: for given object pose and measured contact point, define the likelihood in terms of distance

between the object and the measured contact point and take the predicted contact point as the point on the boundary of the object at minimum distance from the measured contact point. As a further contribution, the standard UPF algorithm has been modified by the inclusion of a suitable sliding *memory* (hence the name MUPF) of past measurements in the update of the particle importance weights. In this respect, it was found that the *memory* feature is crucial for a careful exploitation of the available contact point measurements with consequent improvement of localization accuracy.

Furthermore, it is worth underlining how the proposed algorithm succeeds in solving the problem by using only tridimensional contact point measurements, without requiring the knowledge of surface normals.

Performance evaluation, carried out via simulation tests on two geometric objects and three everyday objects by using simulated measurements and tridimensional mesh models reconstructed by vision, demonstrates that the algorithm is reliable and has good performance with an average localization error less than $0.002\,[m]$ and a computing time of a few seconds. Moreover, the algorithm manages to localize real objects with actual tactile measurements collected with the humanoid robot iCub. The results of experimental tests on four real objects confirm the results of the simulation tests, providing localization errors less than $0.01\,[m]$ with a computing time less than $8\,[s]$.

The same simulation and experimental tests have been carried out also with a reference algorithm in the literature, called Scaling Series. The obtained results show how the MUPF is competitive with the state of art for 6-DOF tactile localization, and also exhibits several advantages with respect to the Scaling Series algorithm.

The MUPF described in this Chapter has been successfully applied also on a challenging tactile recognition task (see Chapter 5 for more details). This is in fact a natural extension of the localization problem. A robot able to localize an object using tactile sensors can also recognize it among a finite set of possible objects, using the same information. For example, given an effective localization algorithm, the robot can run it with different known object models and select the one that best matches the observations.

A useful feature of the proposed algorithm consists of the nature of measurements it processes. As we said in Section 4.2, the MUPF take

advantage solely of the 3D contact positions of its fingertip with the object surface, instead of requiring also the surface normal measurements or other kinds of tactile informations, such as exerted force of texture information, as commonly happens in literature. The usage of this limited information can be considered as a limitation, being the responsible of lack of observability. However, this fact has the silver lining to making the algorithm agnostic to the source of measurements: it just needs to be fed with 3D points collected on the object surface, regardless of the exploited sensors. As a result, the MUPF can localize object by processing 3D point clouds. This turns out to be very useful in practical robotic manipulation since visual perception is often the first source of information for manipulating objects. Object point clouds can be used for estimating the initial object pose. After that, since vision occlusion is very likely to happen when interacting with the object, tactile information is exploited to assist or even replace visual feedback. Chapter 6 shows a practical usage of the MUPF to estimate the object in-hand pose using point clouds during the execution of bi-manual handover tasks.

The contributions detailed in this Chapter suggest other perspectives for future work on 6-DOF object tactile localization. First of all, dealing with the localization of objects in presence of slippage or even tracking moving objects is fundamental in real applications. When filtering techniques (e.g. variants of particle filtering) are employed in place of optimization methods, the extension to this case can be achieved by further considering a suitable model for the object motion. Moreover, the nearly recursive nature and the promising computing time of the proposed algorithm would allow reducing localization uncertainty on-line during measurement collection. This partial information could be in fact exploited to guide tactile exploration in order to maximize the information on the object pose.

Chapter 5

Applications of the Memory Unscented Particle Filter to object tactile recognition

This Chapter shows the application of the Memory Unscented Particle Filter, presented in Chapter 4 on tactile object recognition [200]. The robot explores an object using its tactile sensors, registering the 3D coordinates of the finger-object contact locations. The contact locations collected during the exploration are, then, compared with different object models. The solution of the recognition problem is given as the object whose model better fits the measurements, i.e., the object model with the lowest localization error.

As we already stressed out in Chapter 4, also in this application the measurements consisting only of a set of 3D contact point coordinates. Such data provide very basic, and noisy information, making the tactile recognition task more challenging.

The Chapter is organized as follows. Section 5.1 provides our formulation of tactile recognition as a multi-object localization problem. Section 5.2 presents the exploration strategy for acquiring measurements. Section 5.3 demonstrates the effectiveness of the proposed solution by means of simulation and experimental tests on the iCub humanoid robot. In Section 5.4 we suggest possible future directions.

5.1 Methodology

We introduce hereinafter the problem of tactile object recognition. Let k denote the number of objects of interest, each object being represented by

a mesh model consisting of triangular faces $\{f_i\}$. A set of measurements $\{\boldsymbol{y}_t\}_{t=1}^{L}$ is collected using the tactile sensors by detecting contacts on the surface of object k^* (one of the k objects). It is assumed that object is attached to a surface and, thus, does not move during the exploration. Each measurement provides the 3D coordinates of the contact point, i.e. $\{\boldsymbol{y}_t = (x_t, y_t, z_t)\}_{t=1}^{L}$. The goal is to infer on which object the measurements have been collected. In the described scenario, the solution is given by the object model that best fits the available measurements.

5.1.1 Recognition as multi-object localization

We address the tactile object recognition problem as a localization problem applied to multiple objects, where the solution is provided by the object whose localization error is the lowest among all the considered objects.

The localization algorithm we use is the Memory Unscented Particle Filter, described in Chapter 4. Object recognition is achieved by simply running the MUPF for each of the given object models using the same set of measurements. For each possible object $j \in \{1, \ldots, k\}$, the algorithm finds the pose $\hat{\boldsymbol{x}}_l$ that makes the object model representing the j^{th} object best fit the set of measurements. After k executions of the algorithm, k different solutions $\hat{\boldsymbol{x}}_j$, for $j = 1, \ldots, k$, are computed. Once the pose $\hat{\boldsymbol{x}}_j$ is calculated for each object models $j \in \{1, \ldots, k\}$, the corresponding *performance index* $\mathcal{I}_{L,j}$ introduced in 4.4.2 is used in order to measure the fitness of each object model in the estimated pose j. We recall the *performance index* to be defined as:

$$\mathcal{I}_{L,j} = \frac{1}{L} \sum_{t}^{L} d_{t,j}, \tag{5.1}$$

where L is the number of measurements and $d_{t,j}$ is the distance between the t^{th} measurement and the object model j in the estimated pose $\hat{\boldsymbol{x}}_j$. For the sake of clarity, from now on we refer to the performance index by simply using $\mathcal{I}_j$, omitting the number of measurements L in the subscript. In other words, given the set of measurements and the estimated pose, the proposed performance index is the average of the distances between each measurement and the object model in the estimated pose. Finally, after the k executions of the localization algorithm, the quantities $\mathcal{I}_j$ for $j = 1, \ldots, k$, are available and the solution $\hat{k}$ for the tactile recognition problem is given

by:

$$\hat{k} = \arg\min_{j} \mathcal{I}_j. \tag{5.2}$$

Clearly, the recognition is successful when $\hat{k} = k^*$. The steps of our algorithm for the tactile recognition stated as a localization problem are outlined in Algorithm 3.

Algorithm 3 Tactile recognition algorithm

1: **Data:** k object models, a set of tactile measurements $\{\boldsymbol{y}_t\}_{t=1}^{L}$ on object k^*;
2: **for** $j = 1, \ldots, k$ **do**
3: **Localization algorithm:**
4: data: object model j, set of measurements on object k^*;
5: output: $\hat{\boldsymbol{x}}_j$;
6: **end for**
7: Choose $\hat{k}$ as:

$$\hat{k} = \arg\min_{j} \mathcal{I}_j,$$

where $\mathcal{I}_j = \frac{1}{L}\sum_{t}^{L} d_{t,j}$ and $d_{t,j}$ is the distance between the t^{th} measurement and the object j in the estimated pose $\hat{\boldsymbol{x}}_j$.
8: Recognition is successful if $\hat{k} = k^*$.

5.2 Data Acquisition

The experimental setup consists of the iCub robot [201](Fig. 5.1) and six objects of interest (i.e. $k = 6$) for acquiring tactile data in our experiments. The objects, as shown in Fig. 5.2, are made of wooden geometric shapes. The objects are deliberately selected to have overlapping shapes with strong similarities in order to test our method in a challenging setting. For example, objects (a) and (b) have similar geometric configurations: one has a smooth arched surface and the other a saw-tooth surface, respectively. With the same principle we also selected objects (c) and (d), that have same general shape, the only difference being in the smoothness of the surfaces. Objects (e) and (f) can only be discriminated by the bottom edge: one has a straight edge, while the other has a curved edge.

The robot touches the object at various locations with the tip of its index finger. The fingertip is 14.5 [mm] long, 13 [mm] wide. Each finger is equipped with tactile sensors [22]. A contact location is registered when

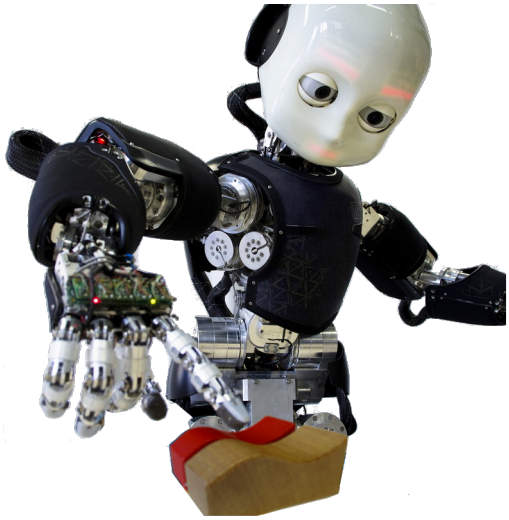
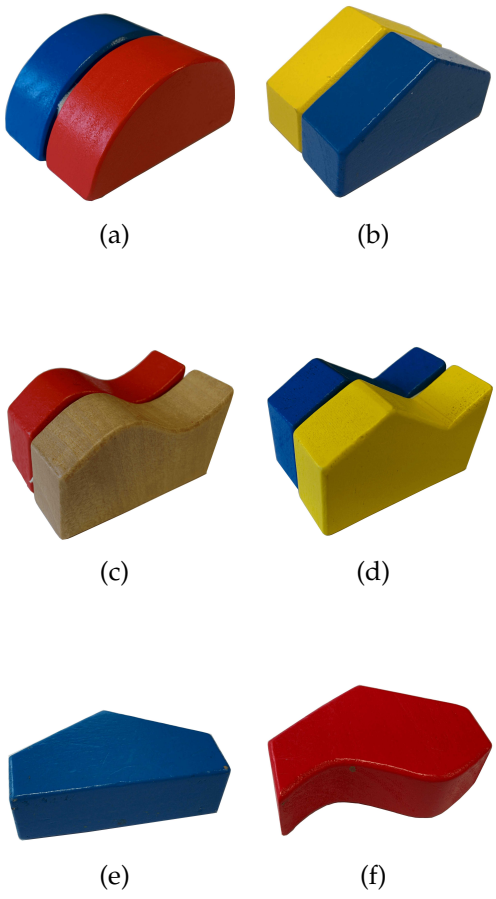


FIGURE 5.1: Experimental setup for data collection: the iCub robot is touching the object with its index fingertip.

the tactile sensors are activated. In our experiments the object is anchored to the surface of a table in front of the robot, hence, it does not move during the exploration. The choice of the exploratory area depends on the the size of the object. We sample an area of 40×50 [mm^2] (Fig. 5.4), using a grid search with a cell size of 2.5×2.5 [mm^2].

At the beginning of the exploration, the robot's index finger is placed at an arbitrary position close to the object. Then, the robot is commanded to sample a location of interest. We will refer to the location of interest as a waypoint. Since we do not have a priori knowledge of the shape of the object, the height of the waypoint is set to an arbitrary value larger than the height of the object. As reported in the flow chart of Fig. 5.3, the robot moves the finger toward the waypoint. After that, the robot extends its finger downward to detect a contact. If no contact is detected when the finger is fully extended, the robot sets the waypoint to the current location of the finger and retracts it. This process is repeated until the finger makes a contact with a surface – either the object or the table. When a contact is detected, the location of the contact is registered and the next waypoint is set to the next point in the grid. This process is repeated until the area is entirely covered. The tactile data collected for each object with this exploration strategy are shown in Fig. 5.4.



(a) (b)

(c) (d)

(e) (f)

FIGURE 5.2: Objects used for experimental evaluation of the method.

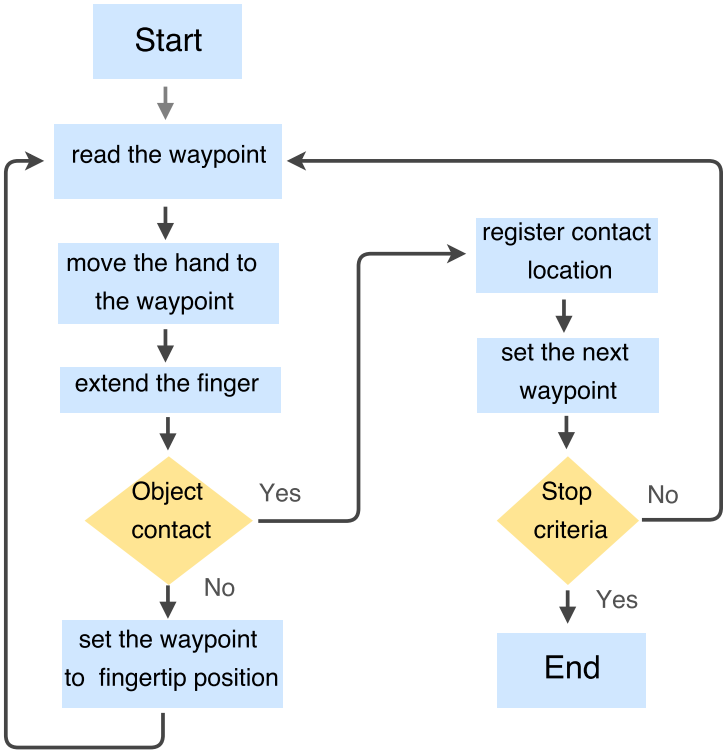


FIGURE 5.3: A flow chart showing the object-surface sampling.

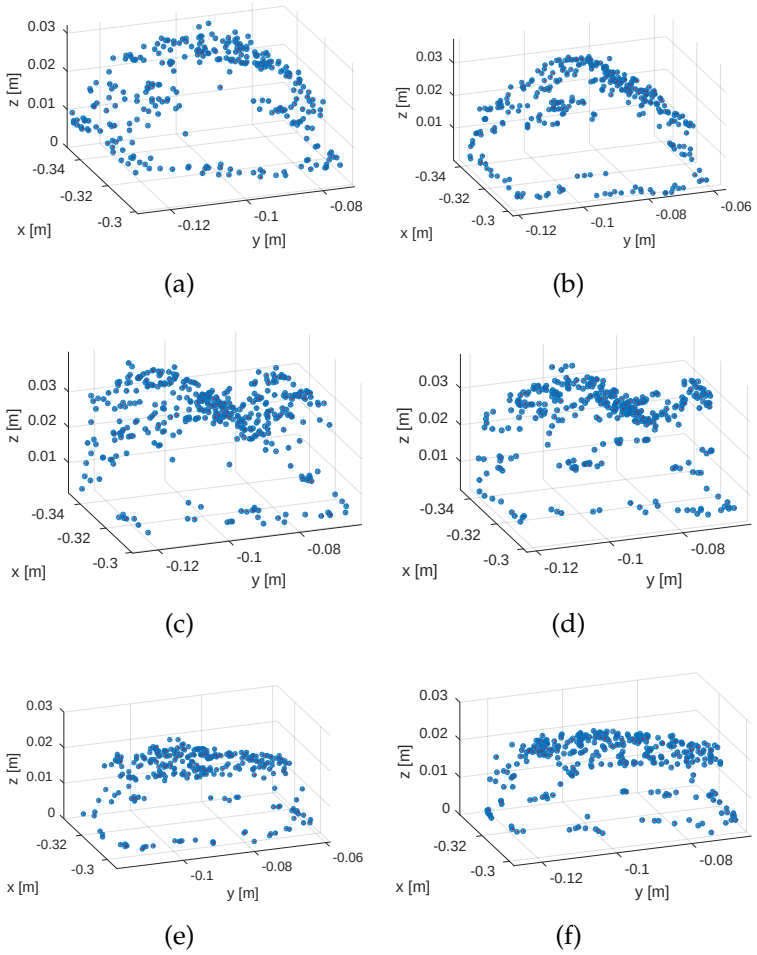


FIGURE 5.4: For each object, the tactile data collected by using the exploration strategy of Section 5.2 are shown. The letters identifying the different plots ((a) - (f)) correspond to the objects according to the notation of Fig. 5.2.

5.3 Results

The algorithm evaluation is performed first with synthetic measurements (Section 5.3.1) and then with real measurements (Section 5.3.2), collected through the exploration strategy described in Section 5.2. In both scenarios, the aim is to recognize the true object labeled as k^*, among the set of six objects shown in Fig. 5.2.

The C++ implementation of the MUPF algorithm used to carry out our experiments is publicly available on GitHub[1].

5.3.1 Simulation results

The synthetic measurements consist of six sets of 3D points (around 170 triplets for each set), each sampled on the surface of one specific model. We refer to the 3D points sampled on object (a) as *set of measurements (a)*. The same notation is used for the other objects. The synthetically-generated data are noiseless.

In Table 5.1, the MUPF parameter set used for running the simulated tests is shown. Matrix Q and σ_p are respectively the covariances of the process noise $\boldsymbol{\omega}_t$ and measurement noise $\boldsymbol{\nu}_t$; P_0 is the covariance matrix representing the initial uncertainty and N is the number of particles. The covariance Q is chosen such that it takes into account the stationarity of the object, similarly, the value of the covariance σ_p models the measurement noise. An arbitrarily large value is instead chosen for P_0 matrix. The selected number of particles N is a trade-off between algorithm execution time and reliability. In order to determine a good value for m, which is the number of most recent measurements used at each time instant, we run the MUPF algorithm for each object. Fig. 5.5 displays how the localization errors vary with different values of m in the range from 1 to L. The figure is for the data collected in the real experiments. The results of the simulated data, which were similar, have been omitted for clarity. Since the localization errors do not decrease significantly for $m > L/2$, $m = L/2$ has been chosen. Such a value leads to accurate performance regardless of the order of the measurements under consideration, thus not requiring the m measurements to be uniformly sampled on the object surface.

[1] DOI:10.5281/zenodo.45493.

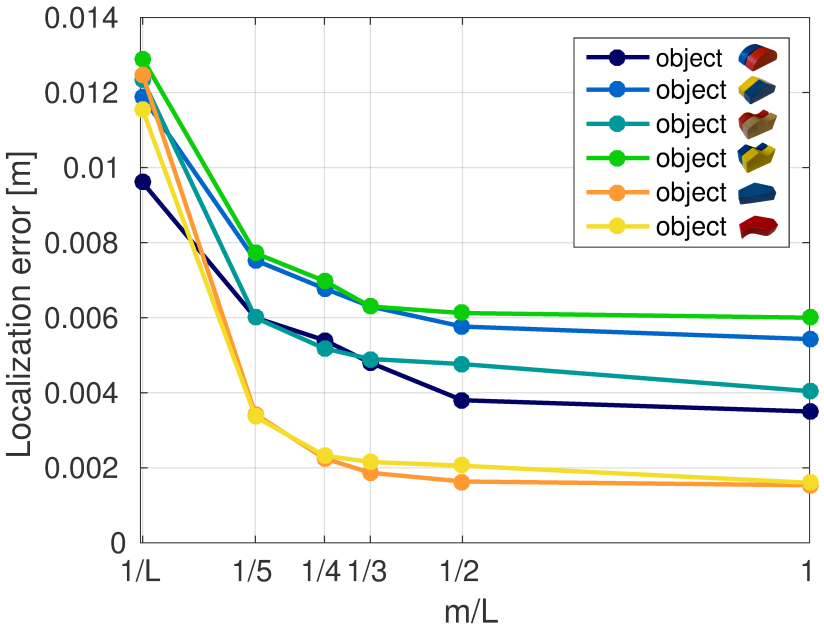


FIGURE 5.5: The localization errors obtained with real measurements with different values of m, from 1 to L. For the sake of clarity, the results from the simulated data have not been plotted as it exhibits a similar trend.

TABLE 5.1: Parameters set for the MUPF in simulation.

Q	diag([10^{-4}, 10^{-4}, 10^{-4}, 10^{-2} 10^{-2}, 10^{-2}]) [m], [rad]
σ_p	10^{-4} [m]
P_0	diag([0.04, 0.04, 0.04, π^2, $(\pi/2)^2$, π^2]) [m], [rad]
N	700
m	$L/2$

Fig. 5.6 shows the performance achieved with the simulated measurements in the shape of confusion matrix. Each row i of the matrix corresponds to a different set of measurements, respectively sampled on each object model surface. The column j instead stands for the j^{th} object model used by the localization algorithm. Each block of the matrix (i, j) contains the average localization errors on 10 trials obtained by running the MUPF with the set of measurements sampled on the object i and the model of object j, i.e. $\mathcal{I}_j^i$. Therefore, the correct behavior of the algorithm can be easily deduced by checking if the localization errors on the diagonal of matrix are the minimum for each row. More tightly, the recognition of object i is

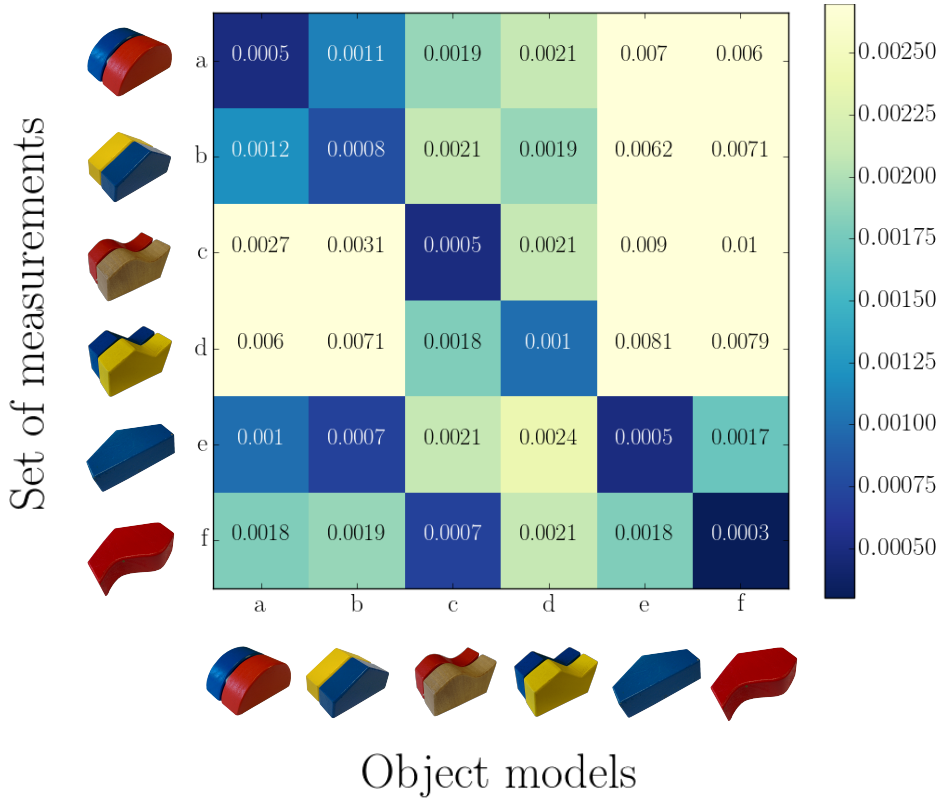


FIGURE 5.6: MUPF performance with simulated measurements. The performance achieved with the simulated measurements are shown in the shape of confusion matrix. Each row i of the matrix corresponds to a different set of measurements, respectively sampled on each object model surface. The column j instead stands for the j^{th} object model used by the localization algorithm. For each experiment, the average localization errors on 10 trials obtained for all the object models are shown.

successful if $\mathcal{I}_i^i = min_j \mathcal{I}_j^i$. Fig. 5.6 confirms that this condition is satisfied when simulation measurements are exploited.

5.3.2 Experimental results

Before showing the performance achieved using the real measurements, we provide a synthetic experiment to point out, from a quantitative viewpoint, that the task at hand is indeed challenging. The results of the experiment are shown in Fig. 5.7. The test consists of calculating the localization error of three different object models: (a), (b) and (c), using the set of *real* measurements (b). More precisely, the three profiles depicted in Fig. 5.7 represent how the localization error varies as the object models slide along the y axis of the frame attached to the object basis. Therefore, Fig. 5.7 reports the localization error versus the y displacement: a displacement equal to 0 represents the correct pose for the object (b), with respect to the set of measurements (b). By observing the trend of the localization errors, we can see how the localization error for object (b) is minimum for a displacement equal to 0, that is in fact the correct pose. However, object (a) and (c) provide an even lower localization error in correspondence of small displacements along y. This fact highlights how the similarity of objects and the noisy nature of the measurements could lead to wrong recognitions.

We discuss hereinafter the performance achieved with real data. The MUPF parameters used for the experimental tests are provided in Table 5.2. The parameters have been chosen by taking into account considerations similar to those explained in Section 5.3.1. In particular, covariances Q and σ_p are tuned differently in order to take into account the measurement noise of the real data. The value of m is determined as described in the previous section, see Fig. 5.5.

TABLE 5.2: Parameters set for the MUPF in real experiments.

Q	diag([$8\ 10^{-6}$, $8\ 10^{-6}$, $8\ 10^{-6}$, $8\ 10^{-4}$ $8\ 10^{-4}$, $8\ 10^{-4}$]) [m,rad]
σ_p	$4\ 10^{-4}$ [m]
P_0	diag([0.04, 0.04, 0.04, π^2, $(\pi/2)^2$, π^2]) [m], [rad]
N	1200
m	$L/2$

Fig. 5.8 shows the results of the real experiments, which can be interpreted similarly to the data of Fig. 5.6. Two main differences can be noticed

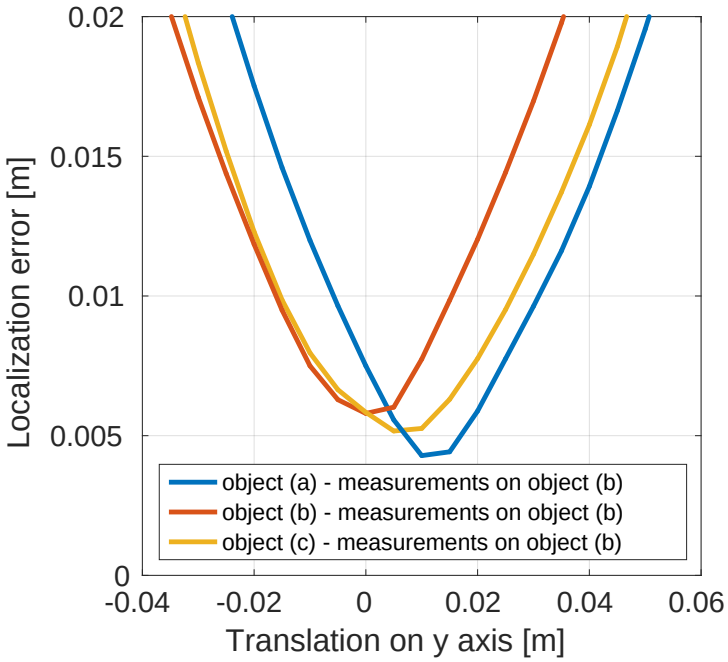


FIGURE 5.7: Synthetic test showing the challenging nature of tactile recognition problem. We compute the localization errors with respect to the set of real measurements (b) and three object models: (a), (b), and (c). Each model is sliding along the y axis of the ground frame. Object (b) results in the lowest error at zero displacement, whereas, notably, object (a) and (c) give lower values for small nonzero displacements.

by comparing Fig. 5.6 and Fig. 5.8, though. First, the measurement noise causes higher average localization errors. Therefore we manage to correctly recognize only 4 objects out of 6 in the real scenario, compared with the 100% overall classification score achieved in simulation. In particular, when the MUPF is executed using set of measurements (b), the solution $\hat{k}$ is given by object (a) and, analogously, when measurements belong to object (d), $\hat{k}$ comes out to be object (c). However, we could reasonably consider these two misclassifications acceptable, considering the high level of similarity between the pairs of objects and the noise in the measurements. In addition, the limited resolution of the tactile sensor and the size of the fingertip (approximately 6×6 [mm^2]) allow only a coarse discrimination of the shape of the object and hide finer details. It is expected that the performance of the recognition would increase using a smaller fingertip or sensors with higher resolution. Given these limitations, however, the carried out experiments demonstrate that the proposed algorithm achieves good performance.

5.4 Discussion

We addressed the problem of tactile recognition as tactile localization on multiple objects using the nonlinear filtering algorithm, Memory Unscented Particle Filter. The algorithm is capable of recognizing objects by exploiting only contact point measurements. The effectiveness of our approach is demonstrated both in simulation and with a real robot.

The promising results presented in this Chapter encourage possible future applications. For example, the model could be extended by including local features, such as surface classification (e.g. local curvature, edge, corners) or material properties (e.g. stiffness, texture). At this aim, the pressure values collected by the robot tactile sensors when contact with the object is detected could be exploited in an extended version of our algorithm. Pressure values provide useful information on stiffness properties, thus facilitating tactile recognition. A further extension of the work we presented consists of taking advantage of a more complex exploration strategy for data collection, by using multiple fingers at the same time. In fact, the exploitation of the knowledge of which finger has caused each

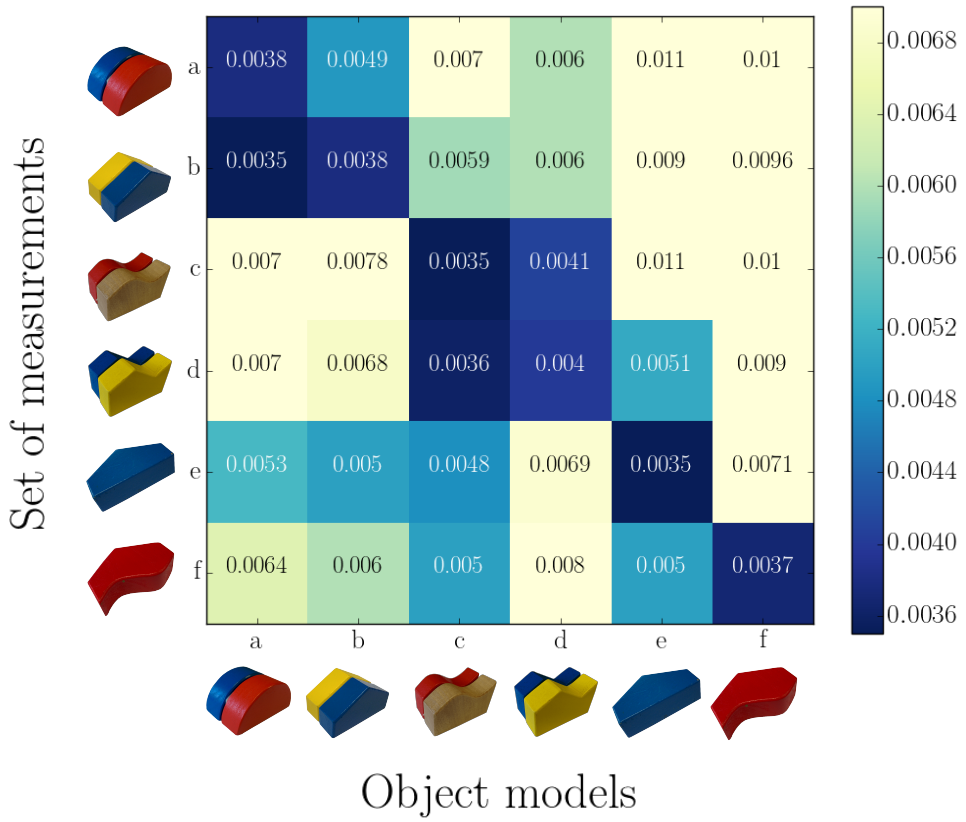


FIGURE 5.8: The performance achieved with the real measurements are shown in the shape of confusion matrix. Each row i of the matrix corresponds to a different set of measurements, respectively collected on each object surface. The column j instead stands for the j^{th} object model used by the localization algorithm. For each experiment, the average localization errors on 10 trials obtained for all the object models are shown.

tactile measurement could be very powerful and considerably improve the performance of our approach.

Part III

Bi-manual coordination: a new pipeline for the execution of handover tasks

Chapter 6

In-hand object localization using vision: bi-manual handover

In the previous Chapters, we presented an algorithm able to localize and recognize objects placed on a support (e.g. a table) by using 3D points sampled on their surface. Both the applications we showed in Chapter 4 and 5 make use of tactile measurements, in terms of the 3D positions of the robot fingertips in contact with the object surface. However, a blind collection of tactile measurements requires accurate planning or, alternatively, needs to be applied on controlled scenarios as shown in Section 5.2. Since our testing platform - the iCub humanoid robot - is provided with stereo vision, a most effective way to collect points belonging to the object surface is by extracting 3D visual point clouds.

In addition, there is no reason to focus only on localizing objects placed on a table. As long as we are able to collect visual and/or tactile measurements, an interesting scenario consists of in-hand object localization, i.e. the goal of estimating the object pose with respect to the hand holding it. In fact, the success, or not, in accomplishing a manipulation task is also determined by how the robot is holding the object. For example, we could image in the future (hopefully not too remote) to give our personal service robot a bottle of wine and ask it to place it on the table in the kitchen. If the robot holds the bottle in the wrong way and *not aware of that*, when the robot will put the bottle on the table, it'll likely fall down, staining the table and the floor and wasting some good wine. This is a simple example showing how in-hand pose localization and re-grasp are fundamental for the achievement even of basic manipulation tasks.

In-hand re-grasp is very challenging and still an open problem. If the robot is provided by more than one arm - as stands for the iCub -, a possible

way to address the problem is by performing *bi-manual handover*. If the object pose in one hand of the robot is not suitable for the current task, the object could be grasped by the other hand in a proper configuration.

Another interesting application of bi-manual handover takes place in a pick-and-place scenario. If the robot is given an object in its left hand and is asked to put it on a target location in right hand workspace, the most reasonable movement in this case requires passing the object from the left to the right hand.

All these considerations encouraged us to study the bi-manual handover problem. The result we achieved is a novel pipeline that allows performing the handover task with the iCub humanoid robot and with different every-day objects. In this work, we did not explicitly take into account handover for re-grasp, but this could be surely a straightforward extension.

The proposed pipeline takes advantage of our previous work on tactile-driven object localization (Chapter 4) as well as other prior works on in-hand tactile manipulation [202] and self-touch [203], conveniently adapted and connected together for tackling the entire handover problem. The core part of the pipeline consists in the pose selection method for selecting the best pose for the handover task among a set of *a-priori* poses. The chosen pose maximizes the distance between the two hands and the manipulability index of a two-arms kinematic chain.

The Chapter is organized as follows. Section 6.1 introduces the pipeline we designed, together with a detailed description of all its steps. Section 6.2 validates our approach by analyzing the results of each pipeline steps and showing a set of successful handovers performed by the robot with different every-day objects. Finally, Section 6.3 ends the Chapter with some discussions about the main limitations of our approach and suggestions for improvements.

6.1 Pipeline

The pipeline for bi-manual object transfer we propose is outlined in Fig. 6.1. In practice, we ask the robot to pass a known object from one hand (that we refer to as *first hand*) to the other hand (named *second hand*). The entire pipeline can be divided in the following steps:

- *Stable grasp with the first hand*: the robot grasps the desired object with the first hand and it controls the grasp using tactile feedback. Grasp stabilization with tactile feedback is active during the entire execution of the task.
- *Point cloud acquisition and filtering:* the robot vision system provides 3D points of the closest blob in the field of view. Then, we properly filter the point cloud in order to extract only points belonging to the object surface, discarding instead those points that belong to the background or the hand.
- *In-hand localization:* a localization algorithm estimates the object in-hand pose by using the filtered 3D points.
- *Grasping pose selection:* the object model is *a-priori* annotated with a set of grasping poses reachable by the second hand. After the object is localized, we rank the candidates according to the distance from the first hand and the manipulability index of the two-arms kinematic chain. We then select the best pose for performing the handover.
- *Approaching strategy:* both arms move until they reach the selected pose.
- *Stable grasp with the second hand*: the robot grasps the object with the second hand and, once the grasp is stabilized, the first hand releases the object. The bi-manual handover is finally achieved.

In the next paragraphs we fully illustrate each step together with the methodology we implemented.

6.1.1 Stable grasp with tactile feedback

A stable grasp of the object is essential throughout the execution of the entire handover task. In fact, the movements of the object can compromise the localization reliability or cause the object to fall while the arms are moving. To this end, we adopted the grasp stabilization strategy described in [202], performing a three-finger precision grasp with the thumb, the index and the middle fingers. In this section we briefly revise this method. Fig. 6.2 shows the overall control schema, which is made of three main components:

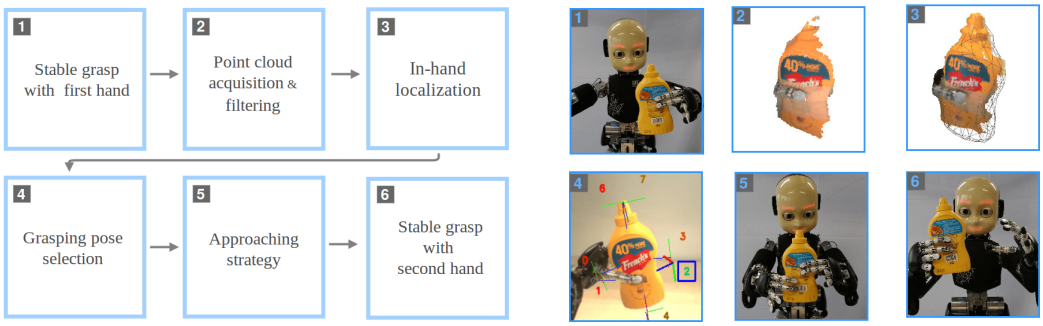


FIGURE 6.1: On the left: a sketch of the proposed pipeline. On the right: some snapshots from the execution of a real handover. 1) The robot grasps the object with the first hand by using tactile stabilization. 2) A set of 3D points of the object is acquired and filtered. 3) The point cloud is used by the localization algorithm for estimating the object pose. 4) The pose for the second hand is selected among a set of previously defined poses. 5) Both arms move so that the second hand achieves the selected pose. 6) Finally, the second hand grasps the object, while the latter is contemporary released by the first hand.

- The *low-level controller* is a set of P.I.D. force controllers, one per finger, which maintain a given force at the fingertips by sending an appropriate voltage signal to the motors actuating the proximal joints Θ_p. We estimate the force at each fingertip by taking the magnitude of the vector obtained by summing up all the normals at the sensor locations weighted by the sensor response.

- The *high-level controller* acts on top of the low-level controller and stabilizes the grasp by regulating the object position while delivering a specified grip strength. The object position α_o, defined in Fig. 6.3, is controlled resorting to a P.I.D. controller that adjusts the set-points of the forces at each finger to reduce the final position error.

  The grip strength is a weighted average between the force applied by the thumb and the forces applied by the index and middle as follows:

$$g = \frac{2}{3} \cdot f_{th} + \frac{1}{3} \cdot (f_{ind} + f_{mid}), \tag{6.1}$$

  where f_{th}, f_{ind} and f_{mid} are the forces at the thumb, the index and the middle fingers, respectively. This is obtained by exploiting some specific assumptions on the force model and the noise distribution,

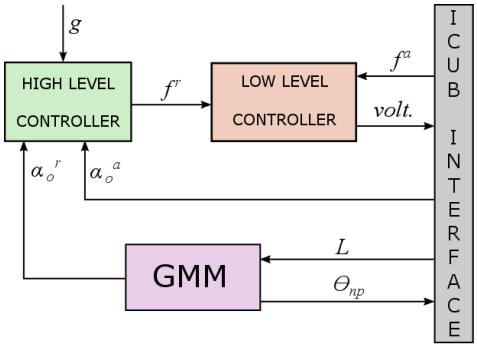


FIGURE 6.2: Grasp stabilizer control schema. While grasping an object, the set $\boldsymbol{L}$ of lengths of the edges of the triangle defined by the points of contact is used by the Gaussian mixture model to compute the reference values of the non-proximal joints $\boldsymbol{\Theta_{np}}$ and the object position α_o^r. In order to reach α_o^r and g, the high-level controller sets the appropriate force references $\boldsymbol{f^r}$ of the low-level controller for each finger. The low-level force controller, in turn, sends voltage to the motors actuating the proximal joints to compensate the force error. The actual object position and the actual forces at the fingertips are represented by, respectively, α_o^a and $\boldsymbol{f^a}$.

whose details can be found in [202]. The target grip strength is kept constant by choosing set-points of the forces that satisfy (Eq. (6.1)).

- The *gaussian mixture model* is a stable grasp model trained by demonstration. We collected several stable grasps using objects of different size and shape. The stability of a grasp was determined by visual inspection, avoiding non-zero momentum, unstable contacts between the object and the fingertips and grasp configurations that are close to joint limits. Given the set $\mathbf{L}$ of lengths of the edges of the triangle defined by the points of contact, which are related to the object shape, the model estimates the target object position α_o^r and the target set of non-proximal joints $\boldsymbol{\Theta_{np}}$ that improve grasp stability as well as the robustness. The target α_o^r is used as the set-point of the high-level controller, while the $\boldsymbol{\Theta_{np}}$ is commanded directly using a position controller.

The grasp stabilizer is triggered when all the fingertips are in contact with the object, which happens by closing all the fingers at constant speed. The fingers stop when they all exceed a given force threshold at the fingertip.

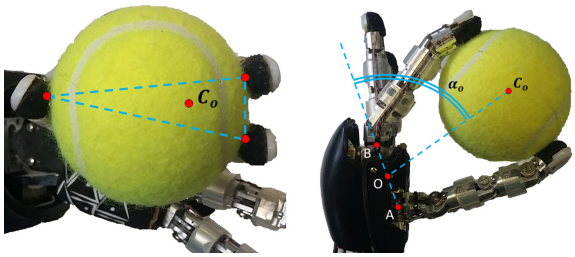


FIGURE 6.3: The object center C_o is defined as the centroid of the triangle identified by the three points of contact (left). The object position α_o is defined as the angle between the vectors $\vec{OC_o}$ and $\vec{OB}$ (right). A and B are set at the base of, respectively, the thumb and the middle finger, while O lies at middle distance between A and B.

(a) (b)

FIGURE 6.4: On the left: Point cloud obtained after the application of coarse filter. The point cloud includes also point belonging to the robot hand. On the right: the blue points represent the selected points after the hand filter. Notice that the points belonging to the hand are removed.

6.1.2 Point cloud acquisition and filtering

Once the object is stably held by the first hand, the nearest blob in the robot visual field is acquired from the stereo vision system. Such a blob contains 3D points belonging both to the visible portion of the object and to the robot hand (see Fig. 6.4(a)). Using point clouds that include parts of the robot's hand would deteriorete the initial information about the object pose. For this reason, a pre-filtering process is required. We implemented two filters that are consequently applied to the point cloud:

a) the *coarse filter* removes possible points outside a volume *a-priori* defined around the robot hand. This filter is necessary in case of noisy initial point clouds, e.g. when the selected blob includes also portions of the background scene.

b) the *hand filter* is applied in order to discard the 3D points belonging to the robot hand. At this aim, the filter removes all points with a specific color property.

An example of filtered point cloud is shown in Fig. 6.4(b). The blue points represent the final point cloud after the filtering process. Hereafter, we describe in the detail the filters we designed.

The coarse filter implementation simply consists of an inside/outside test on the points coordinates. If the 3D point lies outside a 3D box built around the first hand, the point is discarded, otherwise is selected.

A principled way to remove the hand from the point cloud is to use the robot kinematics to project the hand model on the point cloud. In the case of the iCub robot, this approach is unsuitable because the forward kinematics is affected by errors due to the elasticity of the tendons [204]. To overcome this problem we propose to use a color filter. This solution assumes that the hand is gray and it corresponds to pixels with low saturations. Such an approach can be applied to other robotic hands (see Fig. 6.5). The filter selects all points for which a measure of saturation:

$$S = \frac{\sum_{l=1}^{l=L}(|R_l - G_l| + |R_l - B_l| + |B_l - G_l|)/3}{L} > \mu, \tag{6.2}$$

where R_l, G_l and B_l are the RGB values of point $l \in 1, \ldots, L$ and L is the number of points lying in a certain volume of radius r. The value of μ is chosen experimentally to deal with variability in light condition.

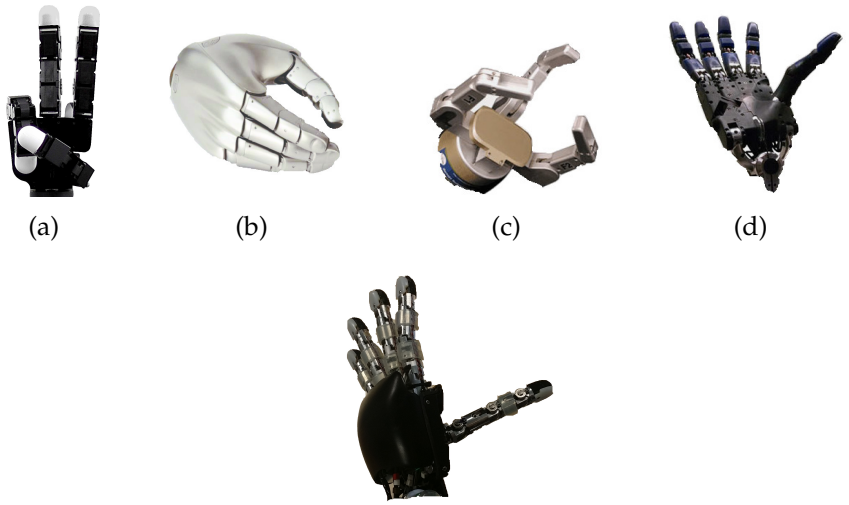
Even though this approach is effective on a large set of objects (see Section 6.2), it may fail when it is applied to grayish objects. The point cloud saturation is in fact not informative enough for distinguishing among the points belonging to a grayish object or to the robot hand. We discuss how we could eventually deal with grayish objects and extend our approach in Section 6.3.

6.1.3 In-hand localization

In order to estimate the pose of the object in the hand, we adapted the Memory Unscented Particle Filter (MUPF) described in Chapter 4.

Recall that the MUPF is a recursive Bayesian estimation algorithm, that is designed for localizing objects given their models and a series of 3D points collected from the object surface.

Even though the MUPF was designed for tactile object localization, we can adopt it for object in-hand localization of the handover pipeline for two reasons. First, the object is stationary (as required by the MUPF) because the grasp approach (Section 6.1.1) stabilizes the object in the robot hand (as validated in Section 6.2). Second, the filter is agnostic about measurements nature, as long as they consists of the Cartesian positions of points lying



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

(e)

FIGURE 6.5: Some examples of grayish robotic hands: (a) Allegro hand, (b) Wessling hand, (c) Barret hand, (d) Shadow hand and (e) the iCub hand, which is the platform we used for testing our approach.

on the object surface. For this reason, we can feed the algorithm with a subset of points belonging to the filtered point cloud (Section 6.1.2). Fig. 6.6 shows an example of localization.

6.1.4 Pose selection

The models of each object are *a-priori* annotated each with N pose candidates for the second hand. These poses represent the minimum set of stable grasps feasible for the object. We compute them by using the Grasp-Studio library provided with the Simox toolbox [205], that offers the possibility to obtain grasping poses for several robotic end-effectors, including the iCub hand. We provide more details on this process in Section 6.2.

Once the object is localized by the MUPF, its model – together with the annotated poses – is considered attached to the first hand according to the estimated pose (Fig. 6.8(a)).

Our approach consists in selecting that pose among the N candidates that allows performing the best handover tasks according to an evaluation criteria and given the estimated object pose and the current robot arms configurations. At this aim, instead of modeling the arms as two kinematic chains with n-DOFs each, we represent them as a single $2n$-DOFs chain,

FIGURE 6.6: An example of object model in the estimated pose, computed via the MUPF algorithm. The algorithm uses the filtered point cloud shown in Fig. 6.4(b).

as explained in[203]. In fact, classical approaches, where the arms are controlled as two separate chains, lead to several difficulties, as remarked in [203]. In particular, they require to identify a specific point $\boldsymbol{x}_d$ reachable by both arms and solve two separate IK problems. In fact to accomplish the task, the solution of the first arm is required to lay in a workspace sufficiently dexterous also for the second arm. The analytical computation of the shared dexterous workspace turns to be quite heavy and, therefore, should be executed off-line. Alternatively, to avoid off-line computations while still complying with real-time requirements, we would need to adopt empirical heuristics in order to come up with a suitable estimation of the dexterous workspace for the task at hand.

The use of a single $2n$-DOF chain, instead, does not require to identify $\boldsymbol{x}_d$ neither to define heuristics, given that it will be implicitly determined in the workspace by the configuration of the $2n$-DOF chain found by the solver. Since the handover task only requires to specify the relative position between the two hands, the use of a $2n$-DOF tends to be more natural and effective, as it does not entail the use of additional heuristics.

For these reasons we make use of a two-arms chain with the *origin* $\mathcal{O}$ located at the end-effector of the first hand and the end-effector $\mathcal{E}$ of the two-arms chain located on the end-effector of the second hand. Specifically for the handover task, we refer to the origin with $\mathcal{O}_j$, since it coincides with the j-th pose candidate, with $j \in 1, \dots N$. This situation is depicted in Fig. 6.7. Under this formulation, the handover task involves moving the

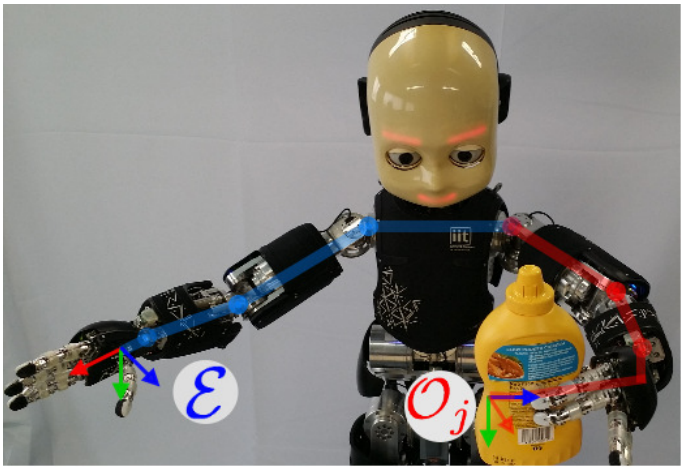


FIGURE 6.7: An outline of the two-arms chain we exploit for the handover task. The new chain origin $\mathcal{O}_j$ is located in correspondence to the j-th pose on the object, held by the left hand. The first part of the chain is reversed with respect to a standard chain for a robotic arm (colored in red). The remaining part (in blue) is a direct chain. The new chain end-effector $\mathcal{E}$ is located on the right palm.

end-effector $\mathcal{E}$ to the origin $\mathcal{O}_j$ of the two-arms kinematic chain. From a mathematical viewpoint, we need to compute the values of joint angles $\boldsymbol{q}_j^*$ such that the end-effector $\mathcal{E}$ reaches the origin $\mathcal{O}_j$ both in orientation and position. Such a problem requires the reversal of the serial chain of the first arm with floating base. In particular, the first part of the kinematic chain is reversed with respect to the standard chains of robot arms, because it is traversed upside down from the origin $\mathcal{O}_j$ to the shoulder. The description of this type of kinematic chain in Denavit Hartenberg (DH) convention [206] is proposed in [203], where the authors provide the algorithm to derive the corresponding DH transformation matrix for each reversed link. We exploit this result for modeling the two-arms kinematic chain.

The joint angles $\boldsymbol{q}_j^*$ for performing the handover task with the j-th pose can be obtained as follows:

$$\begin{aligned} \boldsymbol{q}_j^* &= \arg\min_{\boldsymbol{q}\in\mathbb{R}^{2n}} (\|I - K_\alpha^{\mathcal{O}_j}(\boldsymbol{q})\|^2) \\ &\text{subject to:} \\ &\begin{cases} \|K_x^{\mathcal{O}_j}(\boldsymbol{q})\|^2 < \epsilon \\ \boldsymbol{q}_l < \boldsymbol{q} < \boldsymbol{q}_u \end{cases}, \end{aligned} \tag{6.3}$$

where $I \in \mathbb{R}^{3\times3}$ is the identity matrix, $K_x^{\mathcal{O}_j}(\cdot) \in \mathbb{R}^3$ and $K_\alpha^{\mathcal{O}_j}(\cdot) \in \mathbb{R}^{3\times3}$ are the forward kinematic functions that represents the position and the orientation of the end-effector $\mathcal{E}$ with respect to the origin $\mathcal{O}_j$; $\boldsymbol{q}_l$ and $\boldsymbol{q}_u \in \mathbb{R}^{2n}$ are vectors describing the joints lower and upper limits; ϵ is a parameter for tuning the precision of the reaching movements, typically $\epsilon \in [10^{-5}, 10^{-4}]$. The cost function of Eq. (6.3) imposes the minimization of the error between the orientations of the end-effector reference frame $\mathcal{E}$ and the origin $\mathcal{O}_j$. The constraints take into account the error between the center of the reference frame of the end-effector $\mathcal{E}$ and the origin $\mathcal{O}_j$ and require the solution $\boldsymbol{q}_j^*$ to lie between a set of lower and upper bounds of physically admissible values for the joints. As fully explained in [183], this formulation gives higher priority to the control of the position with respect to the orientation. The former is in fact handled as a nonlinear constraint and is evaluated before the cost function. We require a perfect reaching in position, whereas we can handle small errors in orientations relying on the robustness of our grasp controller.

We solve the problem described in Eq. (6.3) for each pose candidate j, $for\ j = 1, \ldots, N$ using Ipopt [207], thus obtaining the desired joints values to perform the handover with all the possible poses, i.e. $\{\boldsymbol{q}_j^*\}_{j=1}^{j=N}$. The latter N solutions do intrinsically encode suitable configurations for both arms, without the need for dedicated heuristics to specify *a-priori* reachable regions where to perform the handover. Then, we execute two sequential rankings on the N poses in order to select the best one for the handover task.

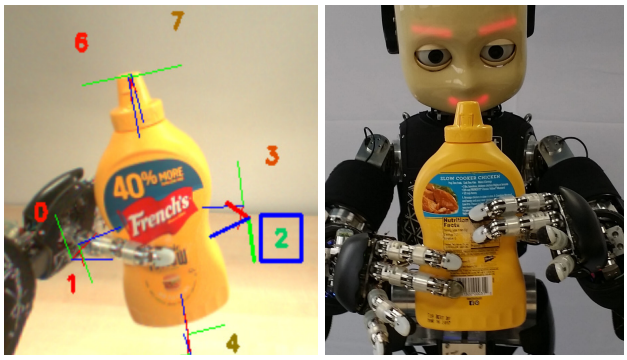
First, the N candidates are ranked according to:

- the **distance d_j of the first hand from the origin** $\mathcal{O}_j$ (which represents the target pose):

$$d_j = \|\boldsymbol{p}_{h,j}\|, \tag{6.4}$$

  where $\boldsymbol{p}_{h,j} \in \mathbb{R}^3$ is the first hand position in the reference frame of the origin $\mathcal{O}_j$. The origin $\mathcal{O}_j$ represents the pose the second hand should reach during the handover. Thus, pose candidates j with larger values of d_j are given a higher score in the ranking, since their probability of collision with the first hand is lower.

Then, we update the first pose ranking by taking into account:



(a) (b)

FIGURE 6.8: On the left: an example of pose ranking. The numbers associated to each pose are colored according to the pose score, ranging from red for the worst pose up to green for the best one. In this case, our method correctly selects pose no. 2, i.e. $j^* = 2$. In particular, the maximum distance criterion discards poses no. 0, 1, 4 and 5. Pose no. 2 is then chosen among the remaining poses due to its higher manipulatibily index.
On the right: the hands holding the object before passing the object from the first to the second hand.

- the **manipulability index** of the **two-arms chain** m_j, in order to favor poses easily reachable by the robot arms:

$$m_j = \sqrt{det(J(\boldsymbol{q}_j^*)J(\boldsymbol{q}_j^*)^T)}, \tag{6.5}$$

  where J is the jacobian of the kinematic chain, $\boldsymbol{q}_j^* \in \mathbb{R}^{2n}$ are the joints values of the two-arms chain which allow performing the handover with the j-th pose and n is the number of DOFs of a single arm.

In summary, the two sequential rankings applied on the N candidates provide as best pose that pose j^* with the maximum distance from the first hand and with the higher manipulability index of the two-arms chain (see Fig. 6.8(a)).

6.1.5 Approach and handover

The robot exploits the joints values $\boldsymbol{q}_{j^*}^*$, computed solving Eq. (6.3) which correspond to the selected pose j^*, to move the arms toward the handover pose. In addition, the second hand passes by an intermediate waypoint so that to avoid its fingers hitting the object during the approach. The

FIGURE 6.9: The iCub performing handover task.

waypoint is simply obtained by shifting the final pose at a fixed distance from the object, along the x and z axis of the hand reference frame.

When the arms reach the final pose, the second hand grasps the object by using the approach described in Section 6.1.1 (Fig. 6.8(b)). The second hand pose j^*, in fact, aims at suitably locate the hand close to the object surface with a proper orientation, leaving the actual grasping task to the grasp controller. The grasp controller of the first hand maintains a stable grasp using tactile feedback. It compensates possible perturbations due to collisions between the object and the second hand, by adjusting the pose of the object. This prevents the object from falling and it improves the robustness of the task. Finally, when the second hand stably grasps the object, the first hand opens and leaves the object in the second hand.

6.2 Results

In order to validate our approach, we tested the pipeline shown in Fig. 6.1 on the iCub humanoid robot (Fig. 6.9). Our implementation of the handover pipeline is availble on GitHub[1].

We carried out our experiments using a set of 5 objects, shown in Fig. 6.10. The objects were deliberately selected among the YCB Object & Model set [208] so as to be different in shape, dimensions and surface texture. We extracted the mesh models of the objects by applying the Poisson

[1]https://github.com/tacman-fp7/handover, DOI:105281/zenodo.437739.

FIGURE 6.10: The set of objects used in the experiments belonging to the YCB Object & Model set. We refer to the objects as: Sugar box, Chocolate box, Mustard box, Chips tube and Little cup.

Surface Reconstruction algorithm [198] to the merged point clouds provided by the YCB dataset.

Without loss of generality, we illustrate the results obtained in case the left hand and the right hand are, respectively, the first and the second hand of the handover task.

We annotated the mesh model by using the Grasp-Studio library. In particular, we used the implemented planner for computing the candidate poses. We selected a subset of poses (Fig. 6.11) among the planner solutions, discarding those that were visibly unstable. Then, we duplicated and rotated the selected poses in order to deal with the object symmetries. This is a crucial point, because multiple, valid solutions of the localization problem are available for symmetric objects. All the models we generated have several symmetries due to their shape. In addition, we only consider the geometric properties of the models, without exploiting information about surfaces color or texture. Our approach is based on geometrical information since we assume that only the object models are available. Fig. 6.12(a) provides an example of the minimum set of poses (for the iCub right hand) we have to consider for box-like objects. Pose annotation shown in Fig. 6.12(a) is in fact invariant with respect to 180-degree rotations of the object along x-, y- or z- axis of its reference frame (located in the object barycenter). Fig. 6.12(b) illustrates how the hand reference

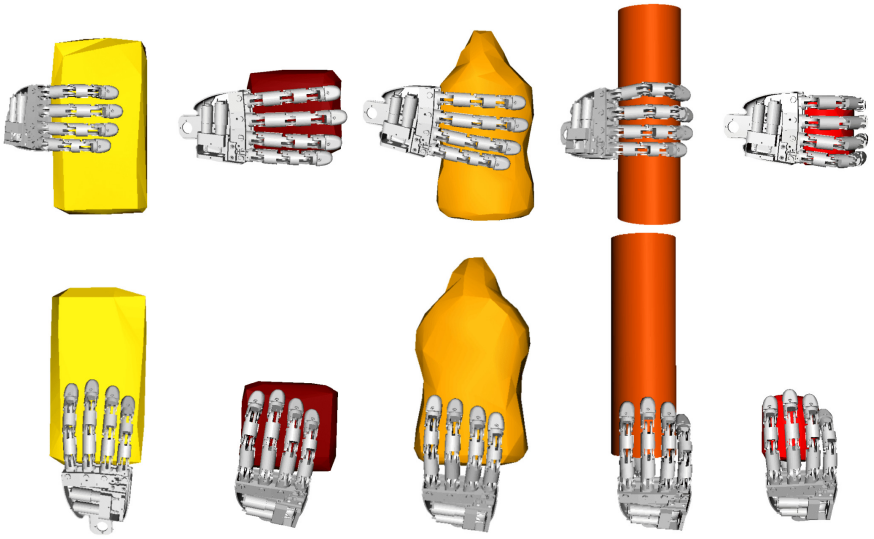
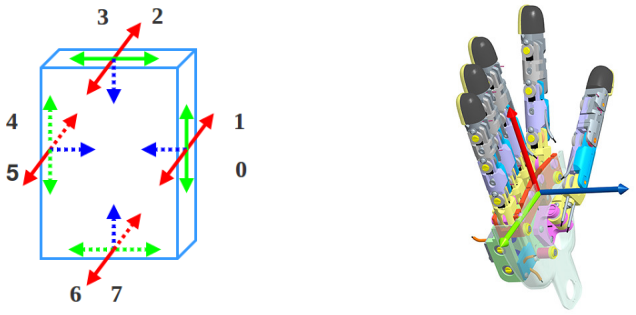


FIGURE 6.11: Some examples of poses generated with GraspStudio. The final set of poses is obtained by multiplying the basic poses according to the symmetry axes of each object.



(a) (b)

FIGURE 6.12: For box-like objects, 8 poses are enough to take into account all the grasping scenarios that might happen, due to the object symmetries (a). Fig. 6.12(b) illustrate the reference frame of the right hand of the robot iCub.

frame is attached to the iCub right hand. More poses are necessary for cylindrical objects, due to their major symmetry. In conclusion, we generated 8 poses for box-like object (Sugar, Chocolate and Mustard box), 24 for the long tube (Chips tube) and 16 for the small cup (Little cup). The Grasp-Studio library generates poses suitable for power grasp tasks. In this work, we make of a precision grip (as shown in Section 6.1.1) in which the fingertips are in contact with the object surface. This is important so that tactile feedback from the fingertip sensors can be used to stabilize the object in the hand. For this reason, the poses computed with Grasp-Studio are shifted far away from the object surface, along the x- and z-axis of the hand reference frame of a fixed amount (for all the objects and poses) which only depends on the fingers lengths.

In the following paragraphs, we at first show the effectiveness of the *stable grasp with tactile feedback*. Afterwards, we show the results obtained with the three main steps of our algorithm: *point cloud filtering*, *in-hand localization* and *pose selection*. Then, we evaluate the reliability of the entire pipeline computing the success rate of our approach for each object and in different poses.

Stable grasp with tactile feedback

The stable grasp we implemented is crucial for the reliability of the handover pipeline. The grasp stabilization is continuosly executed during the entire handover: it avoids object slip while the arm moves and it prevents the object from falling when it touches the other hand. In fact, a change in the in-hand object pose after the localization would affect the handover success. In order to test the effectiveness of our stable grasp, we compared the handover success in presence of stabilization (Table 6.3) with a baseline obtained in absence of stabilization, i.e. only by closing the fingers until contact is detected on the tactile sensors (Table 6.1). This comparison highlights the effectiveness of our stable grasp since it doubles the success rate of the handover test (from 20 - 50% of Table 6.1 up to 50 - 100% of Table 6.3).

Another benefit of the stable grasp is that it is trained by demonstration

TABLE 6.1: Success percentage of the handover task for each object and for different poses, in absence of grasp stabilization. Compare with Table 6.3.

Object	Pose	Success	Pose	Success
Sugar box	Lateral	50%	Top	50%
Chocolate box	Lateral	60%	Top	50%
Mustard box	Lateral	40%	Bottom	30%
Chips tube	Lateral	30%		
Little cup	Lateral	20%		

and, to a certain extent, implicitly takes into account the distribution of the weight in the objects.

Point cloud filtering

During filtering process we make use of the coarse and the hand filter with RGB coding, $r = 0.001$ and $\mu = 25$. Fig. 6.13 shows the point clouds after the coarse filter, on the top, and after the hand filter, on the bottom, for all the objects.

Although we obtain good results with the hand filter, this is a heuristic approach. The main weak point of the method arises when it is applied to grayish objects (see gray portions of Sugar box and Chips tube of Fig. 6.13). In these cases in fact, the point cloud saturation is not informative enough for distinguishing among the points belonging to the object or to the robot hand. A discussion on how we could eventually deal with grayish objects is presented in Section 6.3.

In-hand localization

We select a subset of 100 points (i.e. $L = 100$) of the filtered point clouds for the localization step. The object models in the estimated pose overlapped to the corresponding point clouds are collected in Fig. 6.14. The average MUPF execution time for each object is approximately 45 [s].

FIGURE 6.13: An example of filtered point clouds after the coarse filter, on the top, and after the hand filter, on the bottom. The blue dots represent the final selected points. The hand is correctly removed from the point clouds of all the objects. Sugar box and Chips tube are examples of objects with grayish surface portions. Coherently with the filter behavior, those parts are discarded together with the robot hand. Nevertheless, the filtered point cloud is still representative of the object for the localization algorithm.

FIGURE 6.14: An example of estimated object poses for all the objects. Each mesh model is overlapped to the relative point cloud.

TABLE 6.2: Computation time for pose selection step.

Object	Computation time [s]	Object	Computation time [s]
Sugar box	3.10	Choc. box	3.10
Mustard box	3.10	Chips tube	9.45
Little cup	7.02		

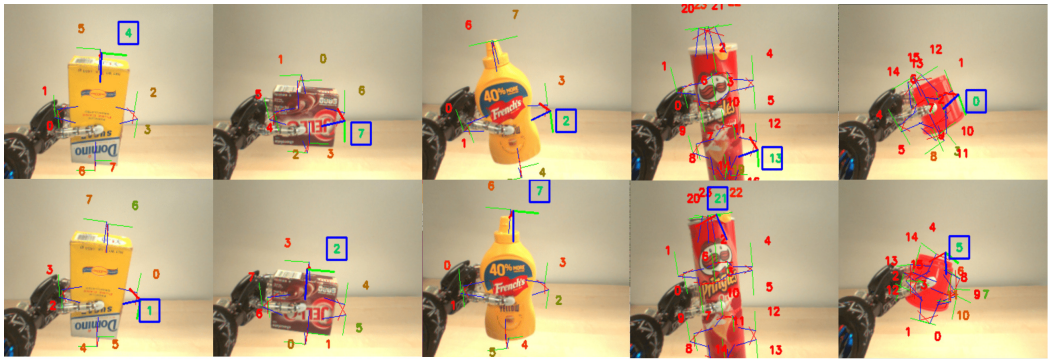


FIGURE 6.15: Some example of grasping pose selection results for the set of objects. Each images shows the pose annotated on the object model in the estimated pose. The poses are labeled with numbers, which are colored according to the pose score, ranging from red for the worst pose up to green for the selected pose.

Pose selection

Fig. 6.15 shows the results obtained with our pose selection approach. For each object, the N candidates are overlapped on the camera image according to the estimated pose. The poses are labeled with numbers. Each number j is colored according to its score in the ranking, ranging from red (worst pose) up to bright green (best pose). The selected pose is indicated with a blue square. Table 6.2 collects the execution time required by the pose selection process.

The tests demonstrate the effectiveness of our approach since the best poses are located on the surfaces farther from the first hand and are better reachable by the second hand with two-arms movements. For example, in the bottom left image of Fig. 6.15, poses no. 0, 1, 6 and 7 have distances larger from the first hand. However, only poses no. 1 and 6 (colored in green) can be selected because they can be reached with a better joints

configuration of the robot arms.

Pipeline reliability

We executed 10 trials for each object and for different poses. Table 6.3 reports the percentage of success of the handovers (greater than 80% for the majority of the experiments). We consider the task successfully achieved if the object is held by the second hand without falling while the arm is moving. Some snapshots of successful handovers are shown in Fig. 6.17. We do not take into account the performance obtained with poses in which the second hand is located on the top of the Mustard box, the Chips tube and the Little cup. Due to the shape of their upper part and their slippery surface, even very small errors in the final reached pose compromise the success/outcome of the handover. These errors are mostly due to the errors in kinematics of the robot and noise in the point cloud.

In Table 6.4, we detail the three main causes of the tests failures. First, uninformative point clouds can be source of errors in object localization. For instance, if only one of the 8 faces of the Sugar box or the Chocolate box is acquired, multiple and wrong solutions of the localization algorithm can properly fit the point clouds. The second critical issue is represented by slippery objects, such as the Mustard box and the Chips tube. In this case, little displacements between the desired and the reached pose can lead to an unstable grasp and to failure. Finally, object size is crucial for the handover success. In case the object size is comparable with the hand dimensions, a fingers avoidance approach is necessary in order to prevent the second hand from blocking the first hand while grasping the object. The lack of such an avoidance approach in our method is the reason of the low reliability of the handover task with the Little cup.

We also validated the robustness of our approach by executing the handover with the object in different initial poses. We focused on the Sugar and Chocolate boxes, because their shape allows significant different grasps by the iCub hand. The other three objects can be instead grasped mostly in the poses shown in Fig. 6.15 due to their slippery surface, their dimensions and their shape. Fig. 6.16 shows the initial poses we took into account for this test. In Table 6.5, we report the success rate among 10 trials

for each different object pose. This test demonstrates that the reliability of the handover is not affected by the initial object pose.

TABLE 6.3: Success percentage of the handover task for each object and for different poses. We consider a handover successfully achieved if the object is held by the second hand without falling while the second arm is moving.

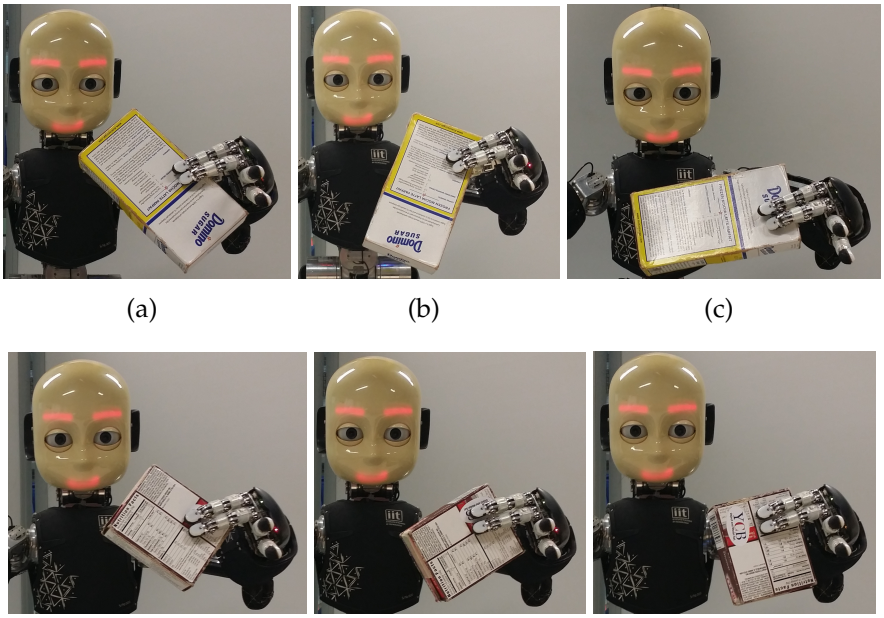
Object	Pose	Success	Pose	Success
Sugar box	Lateral	90%	Top	90%
Chocolate box	Lateral	90%	Top	100%
Mustard box	Lateral	80%	Bottom	80%
Chips tube	Lateral	80%		
Little cup	Lateral	50%		

TABLE 6.4: Main causes of handover failures.

Object	Pose	Failure source
Sugar box	Lateral	Uninformative point cloud
	Top	Uninformative point cloud
Chocolate box	Lateral	Uninformative point cloud
	Top	None
Mustard box	Lateral	Slippery object surface
	Bottom	Slippery object surface
Chips tube	Lateral	Slippery object surface
Little cup	Lateral	Fingers overlapping

6.3 Discussion

Hereafter we summarize the pipeline for the handover task designed by integrating various modules that make use of visual and tactile feedback, namely:



(a) (b) (c)

(d) (e) (f)

FIGURE 6.16: Different initial poses of the object in the first hand used for stressing our approach and testing its robustness.

FIGURE 6.17: Examples of successful handovers.

TABLE 6.5: Success percentage of the handover task for Sugar and Chocolate box for different initial poses, enumerated as shown in Fig. 6.16. We consider a handover successfully achieved if the object is held by the second hand without falling while the second arm is moving.

Object	Initial Pose	Success	Object	Initial Pose	Success
Sugar box	(a)	90%	Choc. box	(d)	90%
Sugar box	(b)	80%	Choc. box	(e)	100%
Sugar box	(c)	100%	Choc. box	(f)	90%

- a grasp controller, stabilizing the object in the robot hand using tactile feedback;
- a point cloud filter, extracting 3D points lying on the object surface from the closest blob in the robot view;
- an object localizer, the Memory Unscented Particle Filter, capable of reliably estimating the object in-hand pose by using the 3D points coming from vision;
- a pose selection strategy, which chooses the best pose for performing the handover by maximizing the distance between the first and the second hand and the manipulability index of a two-arms kinematic chain.

We evaluated our method experimentally with the iCub humanoid robot, showing that it provides a success percentage between 80% to 100% for 4 objects of the YCB Object & Model Set, different in shape, texture and dimensions.

The pipeline design and the experimental evaluation we carried out suggest perspectives for future work. One of the limitations of this work is that it makes use of a saturation filter to separate the hand from the object. This approach may fail when the object and the hand have similar color distributions (specifically, low saturation). To overcome this limitation, we could rely on an accurate kinematic model of the hand or hand tracking techniques [209]. In the particular case of the iCub hand, such

an approach could allow performing the removal of the robot hand also from point clouds of grayish objects. In some cases, the robot had only a partial view of the object, which correspond to an ambiguous point cloud and wrong localizations. If the positions of the hand is known with high accuracy, this can be fixed by considering the localization coming from the fingers in touch with the object. Another extension consists in recognizing the object autonomously. This could be done using techniques from computer vision [210] or by fitting object models on the point cloud acquired from the cameras and the contact points between the fingers and the objects [200]. A more advanced strategy for collisions detection between the first and second hand could help in case of handovers of small objects. For instance, instead of considering only the distance between the positions of the first and the second hand, a better strategy could be to model the hands with enveloping boxes, properly chosen to approximate shape of the robot hand. As we already mentioned at the beginning of this Chapter, the handover could be executed with the aim of re-grasp. Given the desired object in-hand pose $\boldsymbol{p}_d$ for executing a specific task with the second hand, the pose for the second hand could be chosen also taking into account the similarity of the pose candidates with respect to $\boldsymbol{p}_d$. For example, in the scenario shown in Fig. 6.16(c), our pose selection criteria selects the pose candidate located on the shortest side of the object and with maximum distance with respect to the first hand. However, if the desired pose $\boldsymbol{p}_d$ is instead located on the top longest side[2], this information could be added as new constraint in our pose selection approach. The best pose could be selected as the pose candidate more similar to the desired one, but still with maximum manipulability index and the distance with respect to the first hand in order to guarantee the safe execution of the task. Finally, the handover pipeline could be extended to deal with unknown objects by relying on the methodology that will be presented in Chapter 7, in that it provides an approach to reconstruct the object model from a partial point cloud and annotate grasping poses on it.

[2]Grasping the object shown in 6.16(c) on the top longest side is the best choice if the final goal is to place it in vertical pose and not up-side down on a table.

Part IV

Dealing with unknown objects: modeling and grasping with superquadrics

Chapter 7

Superquadric object modeling and grasping

In the previous parts of this Thesis work, we made the assumption to know the model of the object to localize or manipulate. In addition, we a priori annotated the object models with grasp candidates for performing manipulation tasks, such as bi-manual handover. Both these assumptions are often impractical for real-world applications. Moreover, the object models we made use of are CAD[1] models, consisting of triangular faces. CAD models are very expressive but they require proper libraries for efficient computation and the memorization of a large number of parameters, i.e. the mesh vertices and edges.

For these reasons, we decided to address the problem of modeling and grasping *unknown objects*. Even if high performance can be achieved in manipulation in terms of speed, precision and reliability, if a very accurate knowledge of the objects is provided, grasping of unknown objects or whose pose is uncertain is still an open problem.

In this Chapter we propose a novel method for computing in real-time both object models and grasping pose candidates starting from partial point clouds. Instead of using CAD models, our approach rely on *superquadric functions*. Superquadrics provide a mathematical representation which is compact in the number of parameters and has beneficial properties for efficient computation. Although superquadrics have already been used for this purpose [112, 113, 114, 115, 116], our approach differs from previous works in that it makes use of superquadrics also for representing the volume graspable by the robot hand and relies on this information for the computation of a feasible grasping pose. Very often superquadrics are

[1] Computer-Aided Design.

just used for the modeling and grasp candidates are computed using simulators [112, 113, 114] or simple geometric considerations [116] and then ranked according to their reachability. Our approach instead computes a unique feasible grasping pose for the object of interest.

The Chapter is organized as follows. In Section 7.1 and 7.2 we explain respectively the modeling process and the grasping pose computation. Sections 7.3 and 7.4 show two extra features that complete the proposed modeling and grasping pipeline, summarized in Section 7.5. In Section 7.6 we carefully analyze the performance of the proposed method. Finally, in Section 7.7 we provide some discussion about the method and suggestions for future work.

7.1 Superquadric modeling

Superquadric functions have been introduced in computer graphics by A.H. Barr in 1981 [100]. They are an extension of quadric surfaces and can be distinguished in *supertoroids, superhyperboloids* and *superellipsoids*. Superellipsoids are most commonly used in object modeling because they define closed surfaces. Examples of elementary objects that can be represented with superellipsoids are depicted in Fig 7.1.

The best way to define a superellipsoid – which we will call simply superquadric from now on – in an object-centered coordinate system is the *inside-outside* function:

$$F(x, y, z, \boldsymbol{\lambda}) = \left(\left(\frac{x}{\lambda_1} \right)^{\frac{2}{\lambda_5}} + \left(\frac{y}{\lambda_2} \right)^{\frac{2}{\lambda_5}} \right)^{\frac{\lambda_5}{\lambda_4}} + \left(\frac{z}{\lambda_3} \right)^{\frac{2}{\lambda_4}}. \tag{7.1}$$

As shown in Eq. (7.1) a superquadric is identified by only five parameters $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_5]$, three accounting for dimensions $[\lambda_1, \ldots, \lambda_3]$ and two for the shape $[\lambda_4, \lambda_5]$. Fig. 7.2 shows how the exponents $[\lambda_4, \lambda_5]$ affects the shape of the superquadric. Here are the exponent values for some basic shapes:

- ellipsoids: $\lambda_4 = \lambda_5 = 1$;
- parallelepipeds: $\lambda_4 \to 0$ and $\lambda_5 \to 0$;
- cylinders: $\lambda_4 = 1$ and $\lambda_5 \to 0$.

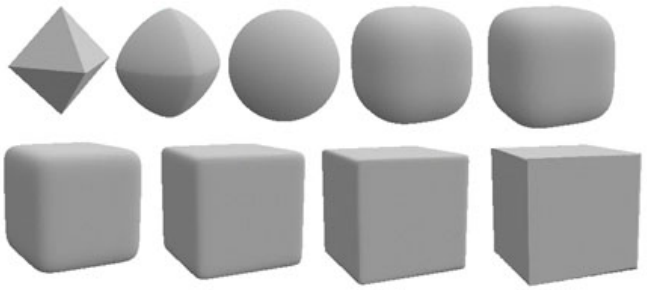


FIGURE 7.1: Superquadric functions can represent several simple objects, ranging from boxes (on the right) to octaedruses (on the left).

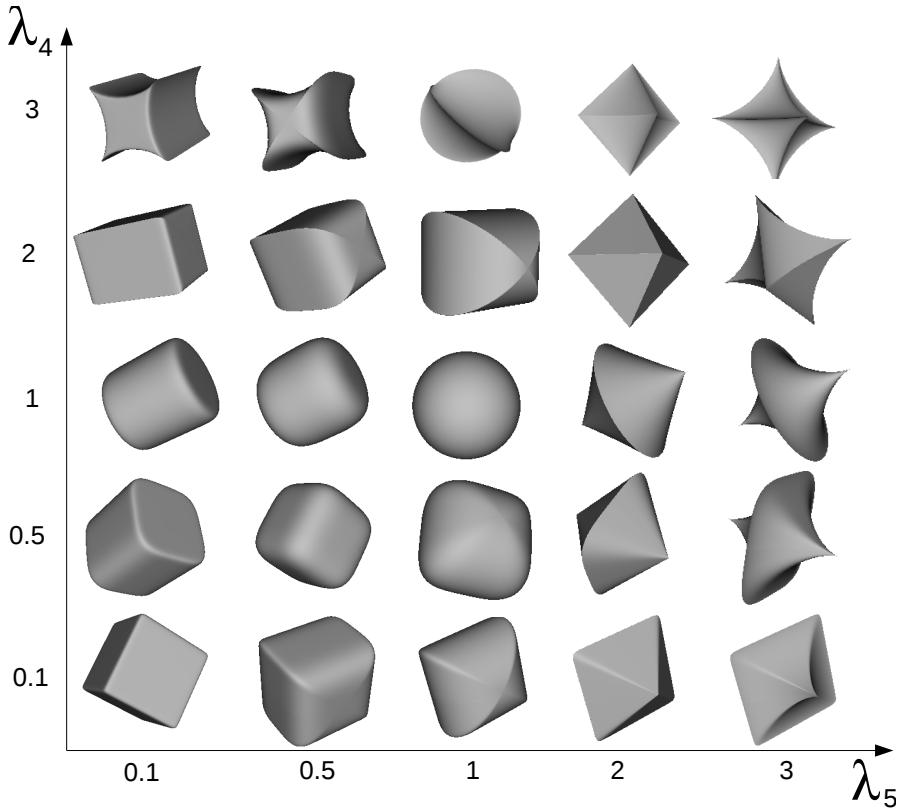


FIGURE 7.2: Examples of superquadrics with different values of the parameters λ_4, λ_5.

In particular, if both λ_4 and λ_5 are smaller than 2, the superquadric has a convex shape. From now on we will always assume $0 < \lambda_4, \lambda_5 < 2$, since convexity is a convenient property for optimization.

Before we referred to Eq. 7.1 as *inside-outside* function. This is because it provides a simple test whether a given point lies inside or outside the superquadric:

- $F < 1$, the given point (x, y, z) is inside the superquadric;
- if $F = 1$ the corresponding point lies on the surface of the superquadric;
- and if $F > 1$ the point lies outside the superquadric.

Furthermore, the inside-outside description can be expressed in a generic coordinate system by adding six further variables, representing the superquadric pose (three for translation and three Euler angles for orientation), with a total of eleven independent variables, i.e. $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_{11}]$. We choose the RPY (roll-pitch-yaw) notation for the Euler angles.

7.1.1 Object modeling

The superquadric $\mathcal{O}$ which best represents the object to be grasped is reconstructed from a single, partial 3D point cloud, acquired by a stereo vision system. The most popular method of superquadric reconstruction from 3D points was formerly proposed by Solina in 1990 [102]. The goal of object modeling via superquadrics consists of finding those values of the parameters vector $\boldsymbol{\lambda} \in \mathbb{R}^{11}$, so that most of the N 3-D points $\boldsymbol{s}_i = [x_i, y_i, z_i]$ for $i = 1, \ldots, N$, acquired by means of the stereo vision system, lie on or close to the superquadric surface. The minimization of the algebraic distance from points to the model can be solved by defining a least-squares minimization problem:

$$\min_{\boldsymbol{\lambda}} \sum_{i=1}^{N} \left(\sqrt{\lambda_1 \lambda_2 \lambda_3} \left(F(\boldsymbol{s}_i, \boldsymbol{\lambda}) - 1 \right) \right)^2, \tag{7.2}$$

where $(F(\boldsymbol{s}_i, \boldsymbol{\lambda}) - 1)^2$ imposes the point-superquadric distance minimization, whereas the term $\lambda_1 \lambda_2 \lambda_3$, which is proportional to the superquadric volume, compensates for the fact that the previous equation is biased towards larger superquadric.

The generic formulation of superquadrics allows for object modeling by solving a single optimization problem (Eq. (7.2)), without requiring *a-priori* information or making assumptions about the object shape.

In the literature Eq. (7.2) is usually solved via Levenberg-Marquardt [211]. We use instead a nonlinear constrained optimization

method implemented through the Ipopt software package [207].

7.1.2 Hand modeling

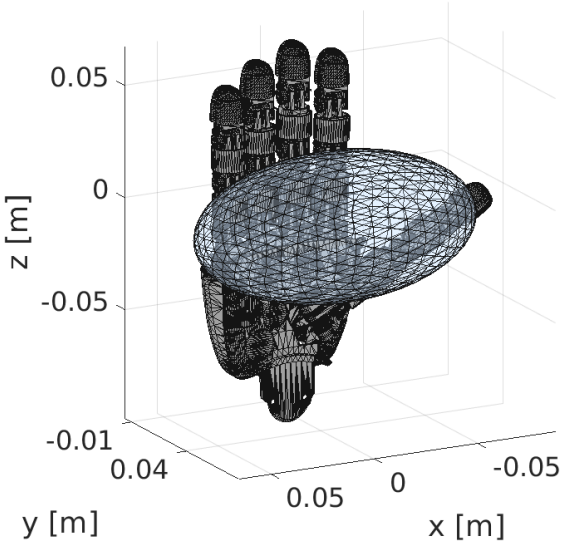


FIGURE 7.3: The volume graspable by the hand is represented as the ellipsoid $\mathcal{H}$ attached to the hand. The right hand of the robot iCub is represented by the CAD model

A fictitious superquadric model is exploited to represent the volume graspable by the hand. The shape and pose of such superquadric are chosen by considering the anthropomorphic shape of the robot hand and its grasping capabilities. A suitable shape for this purpose turns out to be the ellipsoid $\mathcal{H}$ attached to the hand palm, as shown in Fig. 7.3.

7.2 Grasp pose computation

The solution of the grasping problem consists of a feasible pose of the robot hand, which allows grabbing the object under consideration. The hand pose can be represented with a 6D vector:

$$\boldsymbol{x} = [x_h, y_h, z_h, \phi_h, \theta_h, \psi_h], \tag{7.3}$$

where (x_h, y_h, z_h) are the coordinates of the origin of the hand frame and $(\phi_h, \theta_h, \psi_h)$ are the RPY Euler angles, accounting for orientation.

The basic idea of our approach is to compute the solution by looking for a pose $\boldsymbol{x}$ that makes the hand ellipsoid $\mathcal{H}$ overlap with the object superquadric $\mathcal{O}$ while meeting a set of requirements that guarantee $\boldsymbol{x}$ is reachable by the robot hand.

The general formulation we propose can be described by the following nonlinear constrained optimization:

$$\begin{aligned} &\min_{\boldsymbol{x}} \sum_{i=1}^{L} \left(\sqrt{\lambda_1 \lambda_2 \lambda_3}\, (F(\boldsymbol{p}_i^{\boldsymbol{x}}, \boldsymbol{\lambda}) - 1) \right)^2, \\ &\text{subject to:} \\ &\quad h_i(\boldsymbol{a_i}, f_i(\boldsymbol{p}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{p}_L^{\boldsymbol{x}})) > 0, \\ &\quad for\ i = 1, \ldots, M + K. \end{aligned} \tag{7.4}$$

At this regard, it is worth pointing out how such a problem could not be solved with the Levenberg-Marquardt algorithm that is instead designed for dealing with unconstrained optimization problems.

An example of superquadric model and grasping pose computed with our approach is shown in Fig. 7.4. In the next Paragraphs, we report on the meaning of all the mathematical quantities contained in Eq. (7.4).

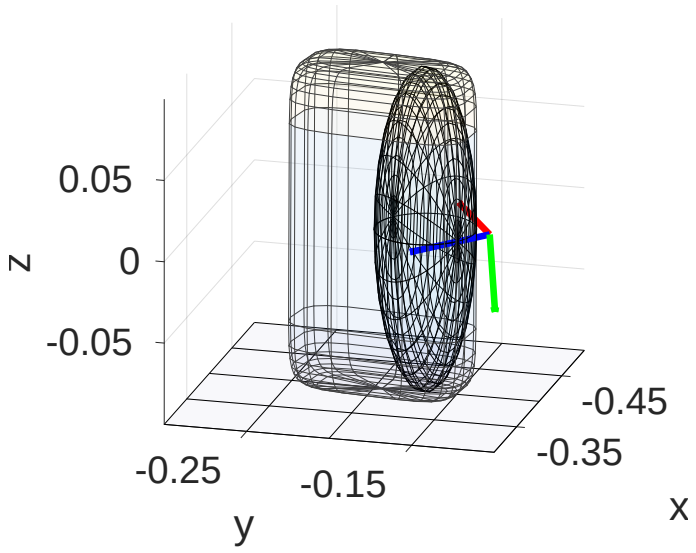
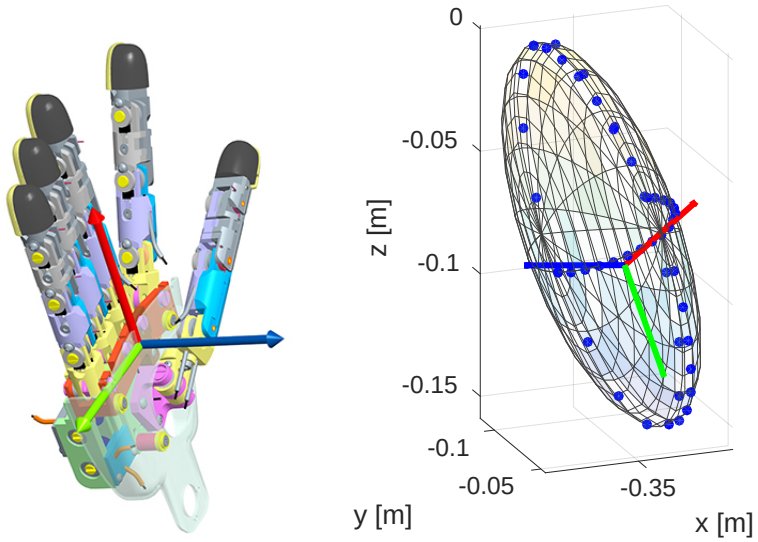


FIGURE 7.4: An example of object modeling and grasp pose computation with our approach for a box located vertically on a table.



(a) (b)

FIGURE 7.5: In Fig. (a), the reference frame (x_h, y_h, z_h) attached to the robot hand in RGB convention (x_h is coloured in red, y_h in green, z_h in blue). In Fig (b): the L points sampled on the closest half of the hand ellipsoid $\mathcal{H}$. The RGB frame represents the hand pose, showing how the ellipsoid $\mathcal{H}$ is attached to the hand.

7.2.1 Grasping avoiding object penetration

The cost function in Eq. (7.4) imposes the minimization of the distance between:

- the object superquadric $\mathcal{O}$, represented by the inside-outside function $(F(\cdot, \boldsymbol{\lambda}) - 1)$,
- and L points $\boldsymbol{p}_i^{\boldsymbol{x}} = \left[p_{x,i}^{\boldsymbol{x}}, p_{y,i}^{\boldsymbol{x}}, p_{z,i}^{\boldsymbol{x}}\right]$ for $i = 1, \ldots, L$, sampled on the surface of the hand ellipsoid $\mathcal{H}$, whose pose is given by vector $\boldsymbol{x}$.

More precisely, the L points lie on the closest half of the ellipsoid $\mathcal{H}$ to the hand (Fig. 7.5(b)). This design choice hinders the robot hand from penetrating the object. In fact, if the points were uniformly sampled on the entire $\mathcal{H}$ surface, the point-superquadric distance minimization could place the ellipsoid $\mathcal{H}$ in the center of the object superquadric $\mathcal{O}$ and, consequently, lead to the object penetration by the hand, in case $\mathcal{O}$ is bigger than $\mathcal{H}$ (Fig. 7.6(a)). By contrast, our choice avoids this scenario. The asymmetric distribution of the L points makes the distance minimization

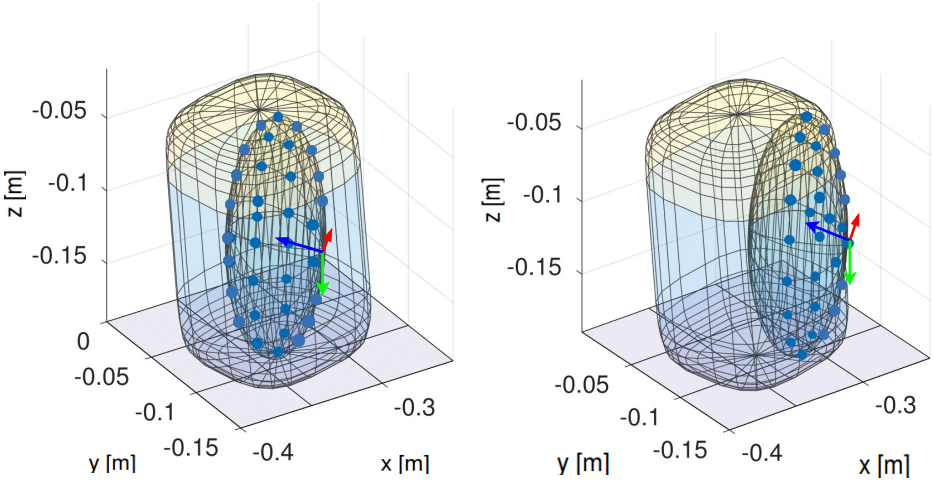


FIGURE 7.6: If the points are uniformly sampled on the ellipsoid $\mathcal{H}$ (i.e. the smallest ellipsoid in the plot), the minimum distance between the points and the object surface can be achieved by placing $\mathcal{H}$ in the centroid of the object superquadric $\mathcal{O}$. In case (a) $\mathcal{O}$ is bigger than $\mathcal{H}$, leading to object penetration (the frame attached to the robot palm is inside the object superquadric $\mathcal{O}$). On the other hand, if the points are sampled only on a portion of $\mathcal{H}$ surface as shown in case (b), the distance minimization is achieved only by placing the $\mathcal{H}$ surface (and then the hand frame) near the surface of $\mathcal{O}$.

possible only if the portion of the $\mathcal{H}$ surface under consideration lies closer to the $\mathcal{O}$ surface, thus avoiding object penetration with the robot hand (Fig. 7.6(b)). The cost function we introduce is similar to the one exploited for object model reconstruction in Eq. (7.2), although the optimization variable in Eq. (7.4) is given by the hand pose $\boldsymbol{x}$ (in the coordinates of the L points $\boldsymbol{p}_i^{\boldsymbol{x}}$), instead of the vector of superquadric parameters $\boldsymbol{\lambda}$, that is given by the object model.

7.2.2 Obstacle avoidance

The use of superquadrics and implicit functions helps us define avoidance tasks. If the implicit functions modeling M obstacles under consideration are given, obstacle avoidance can be taken into account by imposing M constraints in the form of (7.4). Each term $h_i(\boldsymbol{a}_i, \cdot)$, *for* $i = 1, \ldots, M$, is the implicit function representing the i-th obstacle, such as a support on which

the object stands. Each vector $\boldsymbol{a_i}$ consists of the parameters of the i-th implicit function and each $f_i(\boldsymbol{p}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{p}_L^{\boldsymbol{x}})$ accounts for a generic dependency on the L points $\boldsymbol{p}_i^{\boldsymbol{x}}$. The formulation displayed in Eq. (7.4) is general and can be modified according to the problem we aim to address.

In our case, the only obstacle is the table on which the object is located, hence $M = 1$ in Eq. (7.4). For the sake of simplicity, we refer to $h_1(\boldsymbol{a}_1, f_1(\cdot))$ as $h(\boldsymbol{a}, f(\cdot))$. The table is modeled as a plane, whose implicit function is thus linear and given by:

$$h(\boldsymbol{a}, x, y, z) = \; a_1\, x + a_2\, y + a_3\, z + a_4, \tag{7.5}$$

with (x, y, z) a generic point.

We then define the function $f(\boldsymbol{p}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{p}_L^{\boldsymbol{x}})$ as follows. Let $\left(\boldsymbol{x_p}, \boldsymbol{y_p}, \boldsymbol{z_p}\right)$ be

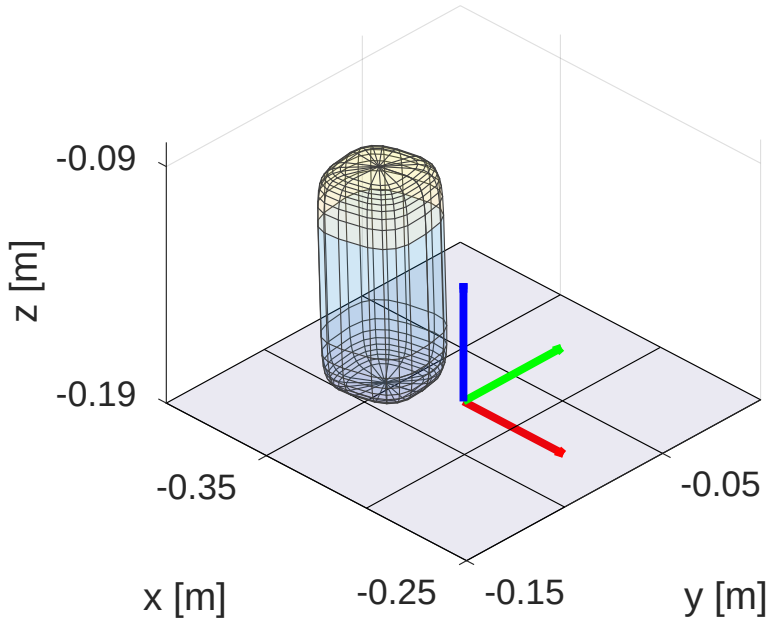


FIGURE 7.7: The frame in RGB convention represents the plane frame $(\boldsymbol{x_p}, \boldsymbol{y_p}, \boldsymbol{z_p})$. The z_p axis is parallel to the plane normal and it is positive in the space region where the object stands.

the reference system anchored to the plane to be avoided. Let $\boldsymbol{z_p}$ be aligned with the plane normal and positive in the space region where the object lies (Fig. 7.7). We call $\boldsymbol{p}_{i,p}^{\boldsymbol{x}} = [p_{x_p,i}^{\boldsymbol{x}}, p_{y_p,i}^{\boldsymbol{x}}, p_{z_p,i}^{\boldsymbol{x}}]$ for $i = 1, \ldots, L$ the points expressed in the plane reference system $\left(\boldsymbol{x_p}, \boldsymbol{y_p}, \boldsymbol{z_p}\right)$. Table avoidance can be achieved by forcing $\bar{p}_p^{\boldsymbol{x}}$, the point with the smallest z_p-coordinate in the plane frame, to lie in the region above the plane representing the table.

Thus, the constraint of Eq. (7.4) can be expressed as follows:

$$\begin{aligned} &a\ \bar{p}^{\boldsymbol{x}}_{x_p} + b\ \bar{p}^{\boldsymbol{x}}_{y_p} + c\ \bar{p}^{\boldsymbol{x}}_{z_p} + d > 0, \\ &\text{with } (\bar{p}^{\boldsymbol{x}}_{x_p}, \bar{p}^{\boldsymbol{x}}_{y_p}, \bar{p}^{\boldsymbol{x}}_{z_p}) = \arg\min_{p^{\boldsymbol{x}}_{z_p,i}} \boldsymbol{p}^{\boldsymbol{x}}_{i,p}. \end{aligned} \tag{7.6}$$

7.2.3 Specifications on pose reachability

An additional advantage of our formulation is the possibility of imposing specifications on the robot pose, by adding further constraints to the optimization problem. This is fundamental for the computation of poses that can be actually reached by the robot. In fact, our formulation does not take into account the robot kinematic and, thus, requires to explicitly impose the final hand pose to belong to the robot workspace.

The coordinates of the hand position (x_h, y_h, z_h) can be bounded with proper lower and upper values representative of the robot workspace:

$$\begin{aligned} x_{h,l} < x_h < x_{h,u} \\ y_{h,l} < y_h < y_{h,u} \\ z_{h,l} < z_h < z_{h,u}. \end{aligned} \tag{7.7}$$

Specifications on the orientation can be instead formulated with additional K constraints, by defining suitable $h_i(\boldsymbol{a}_i, \cdot)$ and $f_i(\boldsymbol{p}^{\boldsymbol{x}}_1, \ldots, \boldsymbol{p}^{\boldsymbol{x}}_L)$ functions for $i = 1, \cdots, K$ and increasing the total number of constraints of Eq. (7.4) up to $M + K$.

At this aim, the orientation of the current hand pose $\boldsymbol{x}$ is expressed by means of the $x-$, $y-$, $z-$axes $(\boldsymbol{x_h}^{\boldsymbol{x}}, \boldsymbol{y_h}^{\boldsymbol{x}}, \boldsymbol{z_h}^{\boldsymbol{x}})$. We formulate $K = 3$ constraints, one for each axis[2]:

$$\begin{aligned} h_x(\boldsymbol{a}_x, \boldsymbol{x}^{\boldsymbol{x}}_{\boldsymbol{h}}) > 0, \\ h_y(\boldsymbol{a}_y, \boldsymbol{y}^{\boldsymbol{x}}_{\boldsymbol{h}}) > 0, \\ h_z(\boldsymbol{a}_z, \boldsymbol{z}^{\boldsymbol{x}}_{\boldsymbol{h}}) > 0. \end{aligned} \tag{7.8}$$

[2]In this formulation, instead of using a function of the points sampled on the hand ellipsoid $(\boldsymbol{p}^{\boldsymbol{x}}_1, \ldots, \boldsymbol{p}^{\boldsymbol{x}}_L)$, we directly exploit the hand reference frame axis $(\boldsymbol{x_h}^{\boldsymbol{x}}, \boldsymbol{y_h}^{\boldsymbol{x}}, \boldsymbol{z_h}^{\boldsymbol{x}})$. Therefore, it is more accurate to say that the orientation constraints are expressed in terms of $h_i(\boldsymbol{a}_i, f(\boldsymbol{x}))$, instead of $h_i(\boldsymbol{a}_i, f(\boldsymbol{p}^{\boldsymbol{x}}_1, \ldots, \boldsymbol{p}^{\boldsymbol{x}}_L))$, since the axes $(\boldsymbol{x_h}^{\boldsymbol{x}}, \boldsymbol{y_h}^{\boldsymbol{x}}, \boldsymbol{z_h}^{\boldsymbol{x}})$ are directly computed from the hand pose $\boldsymbol{x}$.

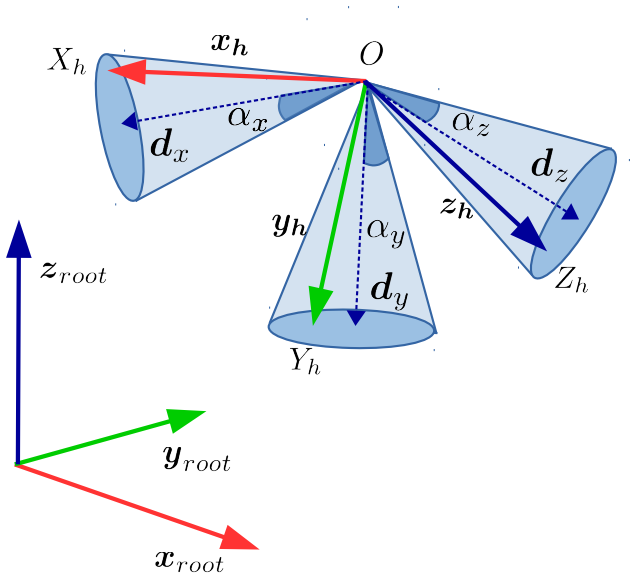


FIGURE 7.8: Cones identifying subregions of the space where hand reference frame axes $(\boldsymbol{x_h}, \boldsymbol{y_h}, \boldsymbol{z_h})$ can range to identify feasible orientations. Each cone is featured by an axis $\boldsymbol{d}$ and an aperture α. The axes $\boldsymbol{d}_x, \boldsymbol{d}_y, \boldsymbol{d}_z$ are expressed in the root reference frame $(\boldsymbol{x_{root}}, \boldsymbol{y_{root}}, \boldsymbol{z_{root}})$. The cone apertures and axes of this image are just an example and do not represent the real quantities.

$h_x(\boldsymbol{a}_x, \cdot)$, $h_y(\boldsymbol{a}_y, \cdot)$ and $h_z(\boldsymbol{a}_z, \cdot)$ are the implicit functions of three different cones representing subregions of the space where each axis $(\boldsymbol{x_h^x}, \boldsymbol{y_h^x}, \boldsymbol{z_h^x})$ can range to identify feasible orientations (see Fig. 7.8). The parameters $\boldsymbol{a}_x$,$\boldsymbol{a}_y$ and $\boldsymbol{a}_z$ include the axis $\boldsymbol{d}$ and aperture α of each cone.

We recall that the implicit function of a cone evaluated in a *generic point* P (Fig. 7.9) is given by:

$$h(\boldsymbol{d}, \alpha, \boldsymbol{p}) = \boldsymbol{d} \cdot \boldsymbol{p} - \|\boldsymbol{d}\| \, \|\boldsymbol{p}\| \cos\alpha, \tag{7.9}$$

where $\boldsymbol{p} = \vec{OP}$ is the vector connecting the vertex of the cone O with the generic point P and the operator $\cdot$ stands for the dot product between vectors. According to the value of $h(\boldsymbol{d}, \alpha, \boldsymbol{p})$ we can assert that:

- the point P lays inside the cone if $h(\boldsymbol{d}, \alpha, \boldsymbol{p}) > 0$ (Fig. 7.9(a));
- the point P lays on the cone surface if $h(\boldsymbol{d}, \alpha, \boldsymbol{p}) = 0$ (Fig. 7.9(b));
- the point P lays outside the cone if $h(\boldsymbol{d}, \alpha, \boldsymbol{p}) < 0$ (Fig. 7.9(c)).

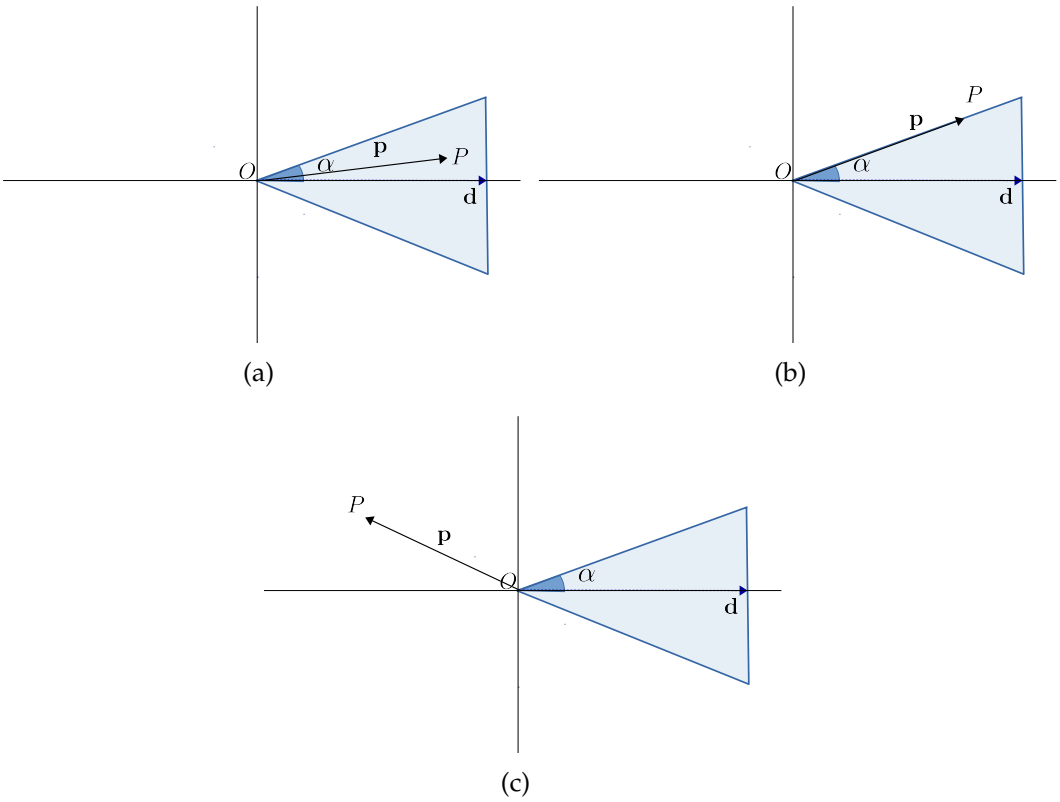


FIGURE 7.9: Examples of possible positions of a point P with respect to a cone with axis $\boldsymbol{d}$ and aperture α.

Combining Eq. (7.8) ad (7.9), we force the axes $(\boldsymbol{x_h}, \boldsymbol{y_h}, \boldsymbol{z_h})$ to belong to the corresponding cones by imposing:

$$\begin{aligned} h(\boldsymbol{d}_x, \alpha_x, \boldsymbol{x}_h^x) &= \boldsymbol{d}_{x_h} \cdot \boldsymbol{x}_h^x - \cos(\alpha_{x_h}) > 0, \\ h(\boldsymbol{d}_y, \alpha_y, \boldsymbol{y}_h^x) &= \boldsymbol{d}_{y_h} \cdot \boldsymbol{y}_h^x - \cos(\alpha_{y_h}) > 0, \\ h(\boldsymbol{d}_z, \alpha_z, \boldsymbol{z}_h^x) &= \boldsymbol{d}_{z_h} \cdot \boldsymbol{z}_h^x - \cos(\alpha_{z_h}) > 0, \end{aligned} \tag{7.10}$$

where both the cone axes $(\boldsymbol{d}_{x_h}, \boldsymbol{d}_{y_h}, \boldsymbol{d}_{z_h})$ and hand reference frame axes $(\boldsymbol{x_h}, \boldsymbol{y_h}, \boldsymbol{z_h})$ have unitary norm. In practice, Eq. (7.10) imposes the extremes of the hand axes X_h, Y_h, Z_h - such that $\boldsymbol{x_h} = O\vec{X}_h$, $\boldsymbol{y_h} = O\vec{Y}_h$, $\boldsymbol{z_h} = O\vec{Z}_h$ - to lay inside the desired cones (Fig. 7.8).

The orientation constraints are general and can be adapted for the computation of grasping poses for different robots by specifying proper cone axes and apertures $(\boldsymbol{d}, \alpha)$. The cone parameters can also be modified according to the problem we aim to address. For instance, they could be used to impose the grasp type, e.g. side or top grasp, for accomplishing specific manipulation tasks.

7.2.4 Lifting objects

The theoretical formulation we presented thus far does not take into account dynamic constraints to let the robot actually *lift* the object. As an initial approximation, the robot can physically lift an object if it places its hand in proximity of the object geometric centroid (the object is assumed to have a uniform density). In fact, in case the hand is located on an extremity of the object, as in the examples illustrated in Fig. 7.10(c) and 7.10(d), the probability that the object may fall while it is lifted is large. A possible solution could be adding a constraint to Eq. (7.4) to require the minimization of the distance between the centroid of $\mathcal{O}$ and the ellipsoid $\mathcal{H}$. However, instead of a further constraint that might cause the overall execution time to eventually increase, we can alternatively vary the dimensions of $\mathcal{H}$. Specifically, when the largest dimension of the object superquadric $\mathcal{O}$ (say λ_3) is greater than the corresponding dimension of the ellipsoid $\mathcal{H}$, (say $\lambda_{h,3}$, thus $\lambda_3 > \lambda_{h,3}$), then we resize $\mathcal{H}$, so that $\lambda_{h,3} = \lambda_3$. In this way, the dimensions of $\mathcal{H}$ and $\mathcal{O}$ implicitly impose a constraint on the reciprocal position of the centroids of the two entities. A practical proof of the effectiveness of this approach is provided in Fig. 7.10. If the ellipsoid $\mathcal{H}$ is stretched so as to

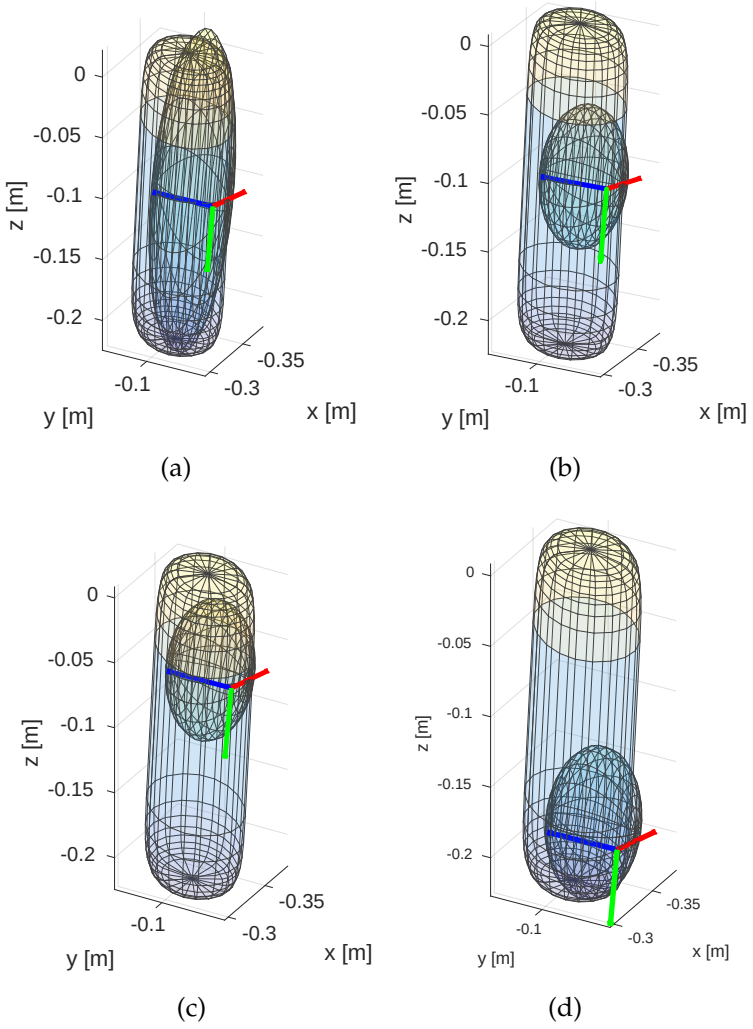


FIGURE 7.10: Stretching the ellipsoid $\mathcal{H}$ so as to amount the longest dimension of the object superquadric $\mathcal{O}$ leads to a grasping pose that eventually enables lifting the object, case (a), without imposing additional constraints in Eq. (7.4). If a smaller $\mathcal{H}$ was exploited, more solutions were acceptable for the optimization problem, including some poses that do not allow the robot to lift the object properly, such as case (c) and (d).

amount the longest dimension of the object superquadric $\mathcal{O}$ (Fig. 7.10(a)), the centroid of $\mathcal{H}$ in the computed pose is close to the centroid of $\mathcal{O}$.

7.2.5 Real-time computation and execution

The optimization problem we propose for the computation of proper grasping pose is solved by the Ipopt package efficiently and with execution

times compatible with the requirements of real-time applications (see Table 7.1 in Section 7.6.1). Even if the global solution of a non-linear constrained optimization problem is not generally guaranteed, Ipopt package ensures that a local minimizer is provided to the user.

Finally, the Cartesian controller available on the iCub [185] is responsible for providing suitable joint trajectories to reach for the grasping pose found by our method.

7.3 Using prior on object shape for modeling

Our superquadric modeling approach processes a 3D partial point cloud of the object. In order to extract the point cloud of the object of interest, we make use of the recognition system described in [212], a segmentation module [213, 214] and module reconstructing depth information [204] (see Appendix A for more details).

As we use a recognition system for extracting the point cloud, we therefore are automatically provided with some information on the object shape that can be useful for the modeling step. Even if our approach *does not require any prior on object shape*, we decided to encode this information to assist the superquadric modeling process, with the final goal of making the model more accurate (as it will be shown in Section 7.6.3).

Recall that object modeling consists in estimating the parameters vector $\boldsymbol{\lambda} \in \mathbb{R}^{11}$ of a superquadric function through the optimization problem of Eq. (7.1). A useful information to facilitate the optimization is to properly set the lower $\boldsymbol{\lambda}_l \in \mathbb{R}^{11}$ and upper bounds $\boldsymbol{\lambda}_u \in \mathbb{R}^{11}$ of the variables to be estimated $\boldsymbol{\lambda} \in \mathbb{R}^{11}$. Reasonable bounds for the object dimensions $(\lambda_1, \lambda_2, \lambda_3)$ and position $(\lambda_6, \lambda_7, \lambda_8)$ can be extracted respectively from the volume occupancy and the centroid of the 3D point cloud.

We can instead obtain proper bounds on the superquadric exponents (λ_4, λ_5), if the object is similar to shape primitives, such as cylinders, parallelepipeds and spheres. Each shape is in fact identified by a specific couple (λ_4, λ_5) in the superquadric representation: $(\lambda_4, \lambda_5)_{cyl}$ = (1.0, 0.1) for cylinders, $(\lambda_4, \lambda_5)_{par}$ = (0.1, 0.1) for parallelepipeds and $(\lambda_4, \lambda_5)_{sph}$ = (1.0, 1.0) for spheres (Fig. 7.11). Thus, we can use different lower and upper bounds for the superquadric exponents according to the shape primitive

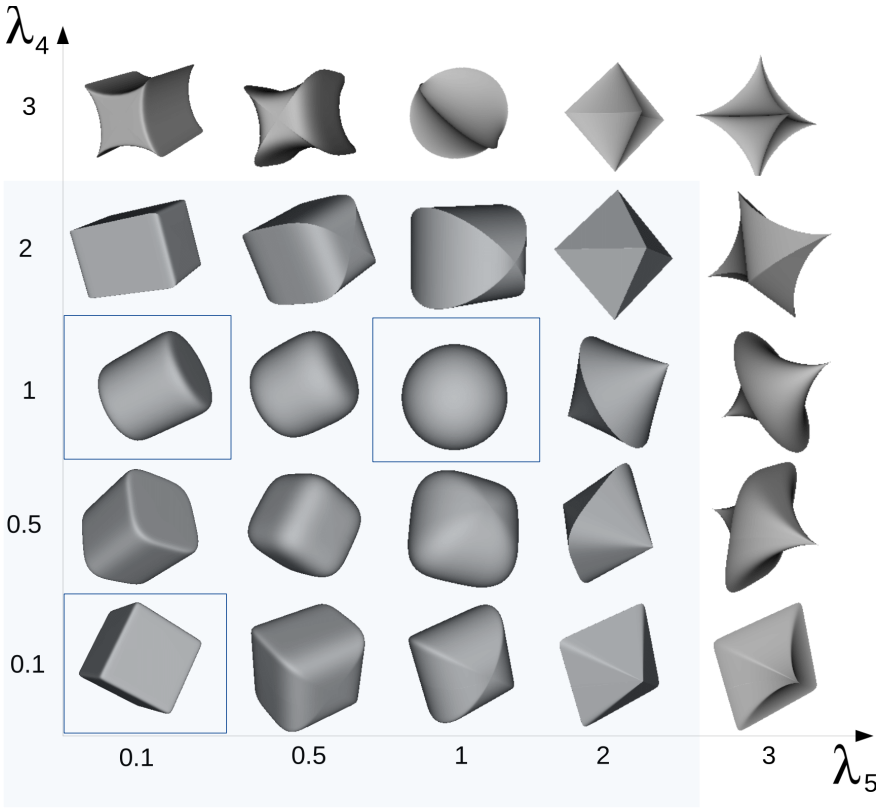


FIGURE 7.11: How superquadric shapes change according to λ_4 and λ_5 values. We are interested only in convex objects, thus $\lambda_{4,min} = \lambda_{5,min} > 0.0$ $\lambda_{4,max} = \lambda_{5,max} < 2.0$. For avoiding difficulties with singularities we use the further bounds $\lambda_{4,min} = \lambda_{5,min} = 0.1$ [102]. In this work we take into account the object shapes highlighted with blue frames. The sharp-cornered shape of parallelepiped and cylinder shapes are caused by $\lambda_4 = 0.1$.

of the object. The bounds can be expressed as:

$$\begin{aligned}(\lambda_4, \lambda_5)_{l,shape} &= (\lambda_4, \lambda_5)_{shape} - (\Delta_{l,4}, \Delta_{l,5}) \\ (\lambda_4, \lambda_5)_{u,shape} &= (\lambda_4, \lambda_5)_{shape} + (\Delta_{u,4}, \Delta_{u,5}),\end{aligned} \tag{7.11}$$

where the label *shape* stands for one of the shape primitives. The bounds shown in Eq. (7.11) force the superquadric shape to be similar to one of the shape primitives.

The $(\Delta_{l,4}, \Delta_{l,5})$ and $(\Delta_{u,4}, \Delta_{u,5})$ values are positive numbers introduced in order to deal with the noise affecting the point cloud. In fact, an object point cloud might be better represented by a superquadric with softer

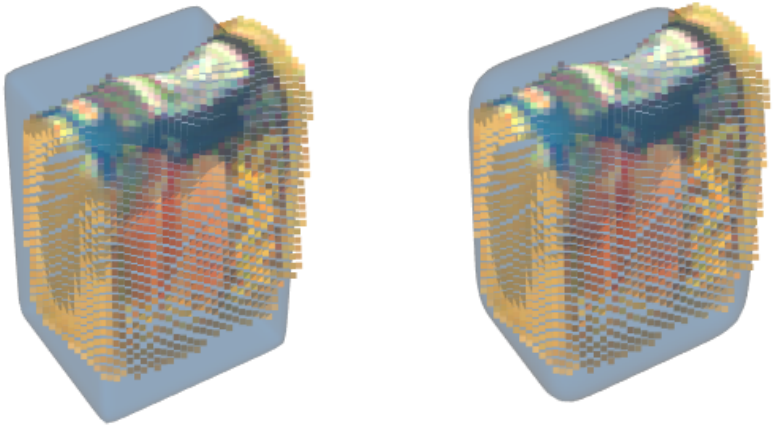
edges due to its noise. Fig. 7.12 shows an example of this phenomenon while modeling a box. The noisy point of the box cloud is represented with dots with the original color of the object and we provide two examples of reconstructed superquadrics. In Fig. 7.12(a), we force the superquadric to be a sharp-cornered parallelepiped, i.e. $\Delta_{l,4} = \Delta_{u,4} = \Delta_{l,5} = \Delta_{u,5} = 0.0$ and, thus, $(\lambda_4, \lambda_5) = (0.1, 0.1)$. In this case, the optimization problem of Eq. (7.2) estimates only 9 parameters (instead of 11), since λ_4 and λ_5 are fixed. However, a sharp-cornered shape is not the best one for the point cloud of interest. For this reason, the solution of the optimization problem does not correctly fit the point cloud and provides wrong dimensions for the object. If we instead set $\Delta_{u,4} = \Delta_{u,5} > 0$ and $\Delta_{l,4} = \Delta_{l,5} > 0$, the optimization problem has to estimate also λ_4 and λ_5, since they can range in a non-zero interval. This allows properly fitting the point cloud with a superquadric with softer edges, i.e. $(\lambda_4, \lambda_5) > (0.1, 0.1)$, as result of the optimization problem (Fig. 7.12(b)). It's important to highlight how actually also the solution $\lambda_4 \simeq \lambda_5 \simeq 0.1$ can be provided by $\Delta_{u,4} = \Delta_{u,5} > 0$ and $\Delta_{l,4} = \Delta_{l,5} > 0$ if this is the shape best fitting the object point cloud (Fig. 7.13).

Next Paragraph reports how we properly trained the system so as to be able to *classify* the object to be grasped according to its similarity to primary shapes.

7.3.1 Object classifier

Object classification is formulated as a categorization problem and is achieved by taking advantage of the recognition system described in [212]. We employ the implementation currently in use on the iCub robot[3] and, specifically, we adopt the ResNet-50 Convolutional Neural Network model [215], trained on the ImageNet Large-Scale Visual Recognition Challenge [216] and available in the Caffe framework [217][4], as a feature extractor. A rectangular region around the object of interest is cropped from the image by using an object segmentation method based on RGB information. Each cropped image is fed to the network and encoded into a 1024-dimensional vector composed of the activations of the 'pool5' layer. In the considered recognition pipeline, the extracted vector representations

[3]https://github.com/robotology/himrep.
[4]https://github.com/KaimingHe/deep-residual-networks.



(a) (b)

FIGURE 7.12: Two examples of superquadric models overlapped to the acquired object point cloud. Fig. (a) shows the superquadric modeling the object obtained by setting $\Delta_{l,4} = \Delta_{u,4} = \Delta_{l,5} = \Delta_{u,5} = 0.0$. Consequentely, (λ_4, λ_5) are fixed equal to (0.1, 0.1) and they are not estimated by the optimization problem. Fig. (b) shows the superquadric modeling the object by setting $\Delta_{l,4} = \Delta_{l,5} = 0.0$ and $\Delta_{u,4} = \Delta_{u,5} = 0.4$. In this case, λ_4 and λ_5 are computed by solving the optimization problem and corresponds to (0.25, 0.25). The superquadric with softer edges shown in Fig. (b) better fits the noisy object point cloud.

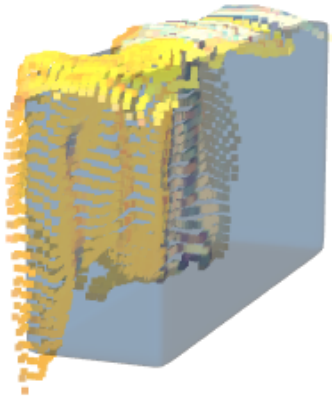


FIGURE 7.13: An example of superquadric modeling an object and reconstructed by using $(\lambda_4, \lambda_5) = (0.1, 0.1)$, $\Delta_{u,4} = \Delta_{u,5} > 0$ and $\Delta_{l,4} = \Delta_{l,5} > 0$. The estimated value of λ_4, λ_5=(0.15, 0.13).

are fed to a multiclass Support Vector Machine (SVM), which is trained to categorize the objects according to their shape.

The training is performed on the fly in a supervised manner: a human teacher shows a number of example objects to the robot, whose labels represent their shape categories. Fig. 7.14 depicts the objects used to train the system on the three shapes under consideration. At test time, the object is assigned to the class with maximum score produced by the SVM classifier, if this is above a certain threshold (set empirically), otherwise it is considered a generic object. Importantly, the objects used for the graping experiments are not part of the training set, i.e., they are fully novel when presented to the robot.

FIGURE 7.14: Training set: 8 parallelepipeds, 8 cylinders and 8 spheres belonging to the YCB dataset and the iCub world dataset [210].

7.4 Best hand selection

In case of robots equipped with two hands - as it holds for the iCub- computing grasping poses for both the end-effectors helps enlarge the dexterous workspace and, ultimately, the grasping capabilities. However, the increased redundancy brings about a further complexity due to the need for conceiving a principled way to determine which is the best hand to be used for accomplishing the grasp. Depending on the object, in fact, the robot can perform better grasps either with the right or with the left hand, alternatively.

The method described in Section 7.2 can be applied both on the right and left hand. What is still missing thus far is an automatic way for selecting the hand to be used given pose candidates for the right and the left hand. At this aim, we propose to rely on the following cost:

$$\mathcal{C}_{hand} = w_1 F_{hand} + w_2 \, E_{p,hand} + w_3 E_{o,hand}, \tag{7.12}$$

that is inversely proportional to the pose quality. In other words, a lower cost C_{hand} stands for a better pose. The cost of Eq. (7.12) takes into account:

- The cost function of Eq. (7.4) evaluated in the computed grasping pose $\boldsymbol{x}_{hand}$,

$$F_{hand} = \sum_{i=1}^{L} \left(\sqrt{\lambda_1 \lambda_2 \lambda_3} \left(F(\boldsymbol{p}_i^{\boldsymbol{x}}, \boldsymbol{\lambda}) - 1 \right) \right)^2 \Big|_{\boldsymbol{x}=\boldsymbol{x}_{hand}}. \tag{7.13}$$

  The higher F_{hand}, the worse the overlapping between the hand ellipsoid and the object superquadric and, thus, the pose are. An example is shown in Fig. 7.15.

- $E_{p,hand}$, the accuracy in reaching the desired position $\boldsymbol{p}_d = [x_h, y_h, z_h]_{hand}$[5]. Let $\hat{\boldsymbol{p}}$ be the position actually reachable by the robot, the accuracy in position is obtained as follows:

$$E_{p,hand} = \| \boldsymbol{p}_d - \hat{\boldsymbol{p}} \| . \tag{7.14}$$

- $E_{o,hand}$, that analogously represents the accuracy in reaching the desired orientation $\boldsymbol{o}_d = [\theta_h, \phi_h, \psi_h]_{hand}$. The orientation error is computed according to the procedure explained in [218], that is hereafter summarized. Let R_d and $\hat{R}$ respectively be the rotation matrices representing the desired orientation of the hand and the one actually reachable by the robot. The matrix representing the error in orientation is given by:

$$R_{error} = R_d \cdot \hat{R}^T. \tag{7.15}$$

  If we use the equivalent axis-angle representation $(\boldsymbol{a}_{error}, \alpha_{error})$ to express the information encoded in R_{error}, the accuracy in reaching the

[5]Recall that $\boldsymbol{x}_{hand} = [x_h, y_h, z_h, \theta_h, \phi_h, \psi_h]_{hand}$.

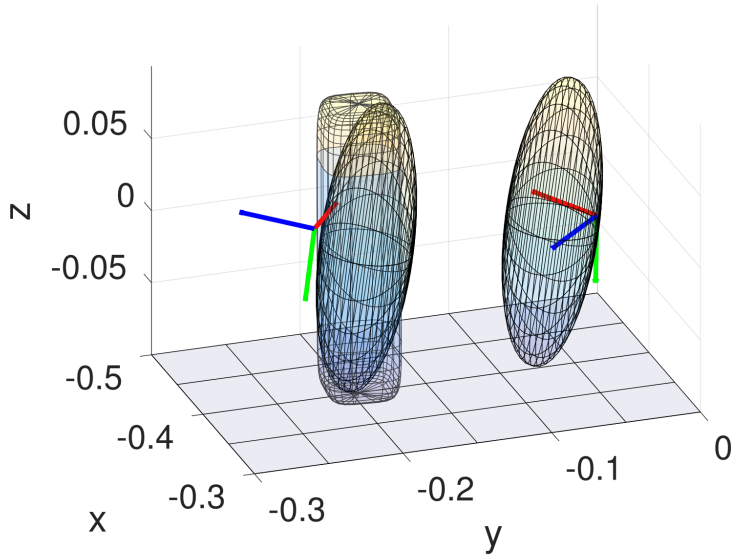


FIGURE 7.15: Example of object graspable only by the left hand. The poses and the ellipsoids shown on the left and on the right are respectively for the left and the right hand. The object is located out of the right hand reachable workspace. For this reason, the optimization problem is not able to find a solution where the right ellipsoid is overlapped on the object model. In this case we obtain $F_{f,left} < F_{f,right}$.

desired orientation can be computed as:

$$E_{o,hand} = \|(\boldsymbol{a}_{error})\| \cdot \sin(\alpha_{error}). \tag{7.16}$$

The three terms of the cost C_{hand} are combined together with proper positive weights w_1, w_2 and w_3 to make the quantities comparable.

In summary, once pose candidates are computed for the right and the left hand, we evaluate the costs $\mathcal{C}_{right}$ and $\mathcal{C}_{left}$. The hand chosen for grasping the object is the one providing the minimum cost, i.e. $\arg\min(\mathcal{C}_{right}, \mathcal{C}_{left})$. In case both the pose candidates are both suitable for grasping the object, the costs $\mathcal{C}_{right}$ and $\mathcal{C}_{left}$ might be very similar to each other. Even in this case, our approach chooses the hand whose pose cost is minimum, as shown in Fig 7.16.

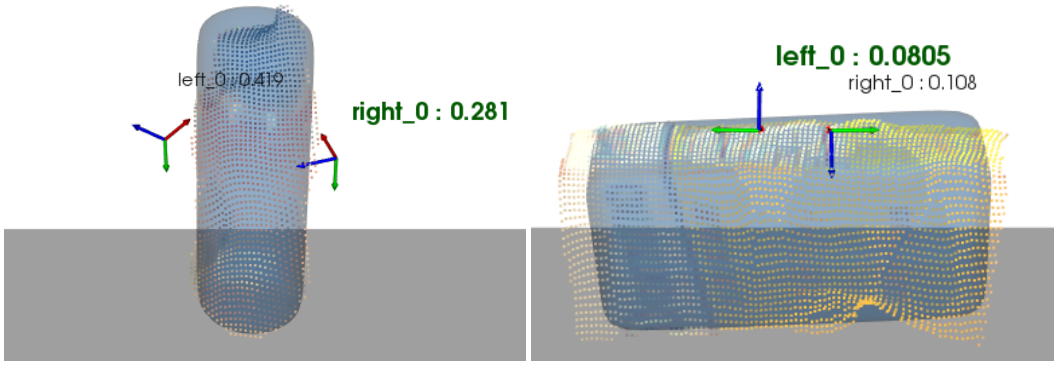


FIGURE 7.16: Example of grasping candidates with similar costs $\mathcal{C}_{right}$ and $\mathcal{C}_{left}$.

7.5 Final modeling and grasping pipeline

The modeling and grasping methods described respectively in Section 7.1 and 7.2 are the core part of a complete pipeline for executing the grasping task on the iCub humanoid robot (Fig. 7.17). The pipeline steps are the following:

1. A rectangular crop of the image is extracted from the camera images. The categorization system classifies the cropped image according to its similarity to shape primitives. We take into account three possible shapes: cylinder, parallelepiped and sphere. All those objects that cannot be well represented with a shape primitive are threated as generic objects.

2. The object segmentation is used for extracting the relative 3D point cloud from stereo vision.

3. The modeling approach computes the superquadric that better fits into the object 3D points.

4. Our grasping approach finds a pose candidate for the right and left hand by solving Eq. (7.4) and the pose costs $\mathcal{C}_{right}$ and $\mathcal{C}_{left}$ are evaluated.

5. The hand with minimum cost value is chosen for grasping the object

$$hand = \arg\min(\mathcal{C}_{right}, \mathcal{C}_{left}). \tag{7.17}$$

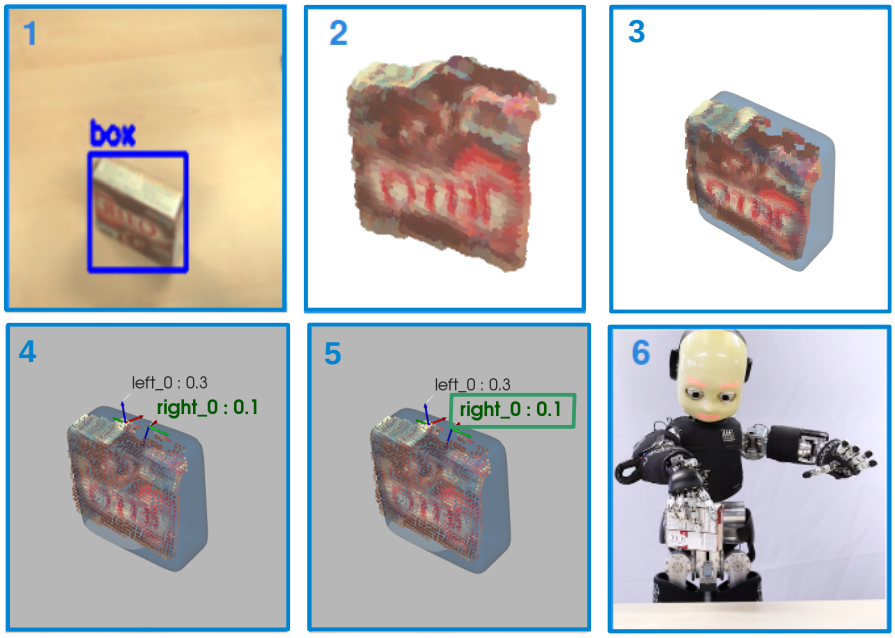


FIGURE 7.17: The complete modeling and grasping pipeline. The gray background of steps 4 and 5 represents the table on which the object is located, that is modeled as a plane.

6. The robot uses the selected hand to reach for the grasping pose. Once the final pose is reached, the robot close the fingers until a contact is detected by the tactile sensors mounted on the fingertips. Then, the tactile feedback on the fingertips is used continuosly to achieve a stable grasp of the object [219] and the robot lifts the object.

7.6 Evaluation

In this Section, we report the analysis we performed in order to evaluate the superquadric modeling and grasping pipeline proposed in this Chapter.

We implemented on the iCub humanoid robot the pipeline shown in Fig. 7.17. Our implementation of superquadric modeling[6] so as grasp pose computation[7] is publicly available on GitHub and detailed in Appendix A.

[6] *https://github.com/robotology/superquadric-model*, DOI:10.5281/zenodo.262995.
[7] *https://github.com/robotology/superquadric-grasp*, DOI:10.5281/zenodo.263015.

The dimensions of the ellipsoid we use for representing the volume graspable by the iCub hand are related to the fingers lengths and the palm dimensions. The ellipsoid pose in the hand frame is instead determined by considering the fingers workspace.

The executed experiments can be summarized as follows:

- At first, we report for qualitative analysis some examples of object models and grasping poses computed for a large set of objects graspable by the iCub, together with statistics on the computation time (Paragraph 7.6.1).
- We then test the reproducibility and robustness of the modeling and grasping steps by performing different trials for some objects and analyzing the similarity of the outcomes (Paragraph 7.6.2).
- Then, we show how the use of prior helps improve the model accuracy (Paragraph 7.6.3).
- Finally, we demonstrate how the use of two hands together with our best selection approach leads to higher reliability in grasping objects in generic poses (Paragraph 7.6.4).

7.6.1 Evaluation on multiple objects

The proposed approach has been tested on 18 objects (Fig. 7.18), including objects of the YCB dataset [208]. The objects have been selected so as to be graspable by the iCub: we discarded objects with slippery surfaces, that are too large or too heavy for the robot hand and too small for being grasped with a power grasp technique.

The quality of the estimated models and, therefore, of the pose candidates strongly depend on the point cloud noise. In order to reduce such a influence, we implemented the following expedients. First, the least square formulation itself of Eq. (7.2) makes the model reconstruction approach immune to white noise. Second, we reduce outliers effect on object modeling by pre-filtering the point clouds with the clustering algorithm Density-Based Spatial Clustering of Applications with Noise (DBSCAN [220])[8]. Of course, the use of single-view point clouds leads to rough models whose

[8]The C++ implementation we use is derived from the C implementation available at https://github.com/gyaikhom/dbscan.

FIGURE 7.18: Object set used for testing our modeling and grasping pipeline.

quality highly depends on the view angle during data acquisition. Fig. from 7.19 to 7.22 show some examples of extracted point clouds, together with the estimated superquadric models and grasping poses. In Table 7.1, we report the average time among 10 trials required for reconstructing the object model and computing the pose candidates. The average time required by the modeling is larger than the one reported in [221] since in the current experiments we set the desired tolerance of the final modeling error to be an order of magnitude bigger with respect to [221]. Instead, the average time required for the grasping poses computation is smaller than the one shown in [90] thanks to code improvements.

TABLE 7.1: Execution time for model reconstruction.

Object	Modeling time [s]	Grasp computation time [s]
Cylinder	0.15	0.09
Pig	0.31	0.06
Cat	0.16	0.12
Lettuce	0.18	0.26
Bear	0.12	0.12
Mustard box	0.24	0.23
Juice bottle	0.16	0.08
Sugar box	0.18	0.08
Jello box 1	0.23	0.16
Turtle	0.27	0.17
Meat can	0.18	0.05
Lego brick	0.21	0.05
Carrots	0.16	0.10
Cereal box	0.19	0.08
Jello box 2	0.24	0.06
Octopus	0.14	0.14
Dog	0.19	0.42
Ladybug	0.32	0.05

Table 7.1 indicates the average execution time across 10 trials for model reconstruction process and grasping pose computation of each object. Superquadric modeling is performed including prior information on object shape and by using only 50 points uniformly sampled from the object point clouds.

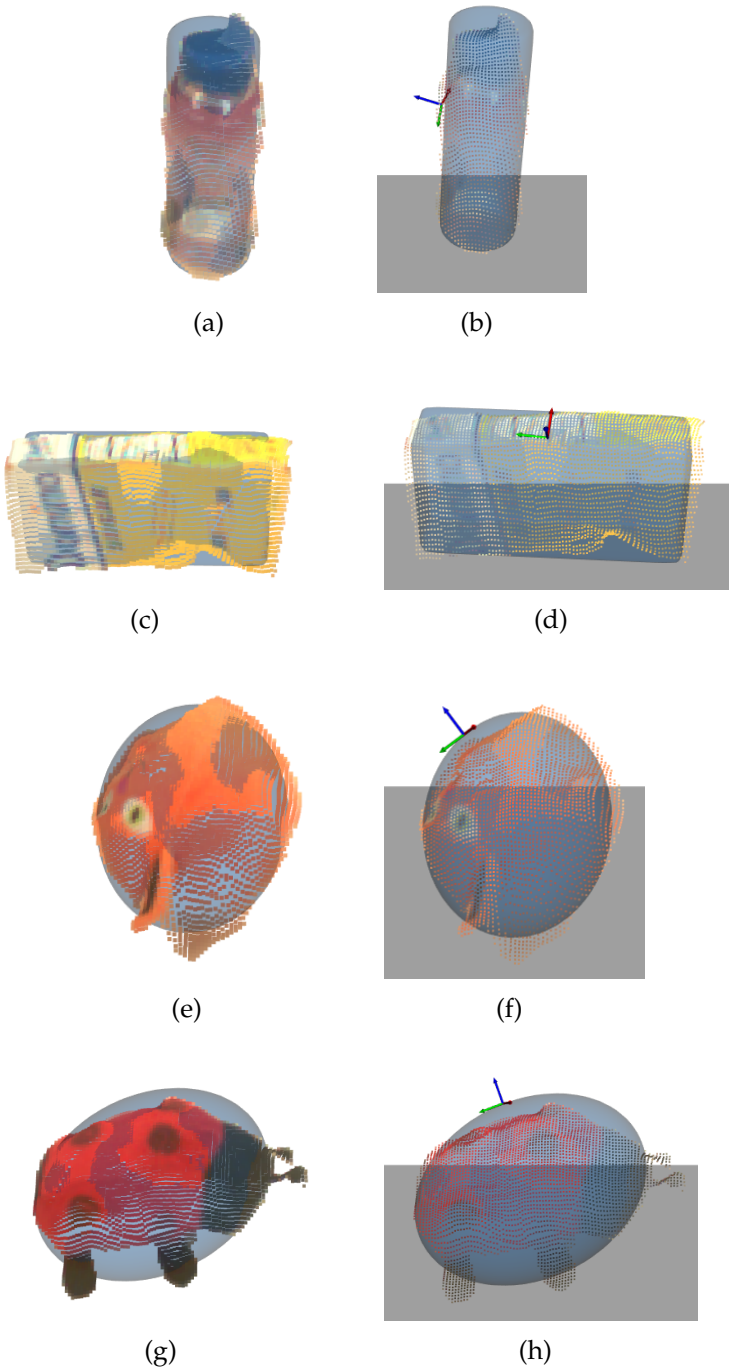
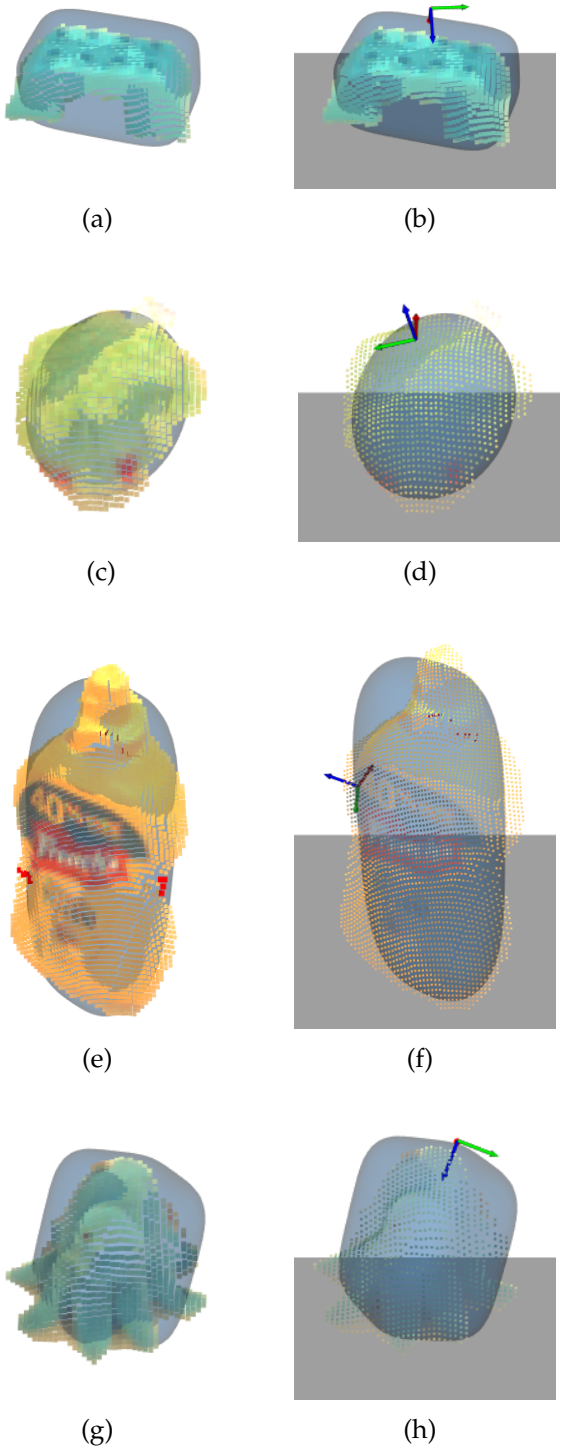


FIGURE 7.19: Some examples of superquadric models and grasping pose candidates for the 18 objects of the test set.



(a) (b) (c) (d) (e) (f) (g) (h)

FIGURE 7.20: Some examples of superquadric models and grasping pose candidates for the 18 objects of the test set.

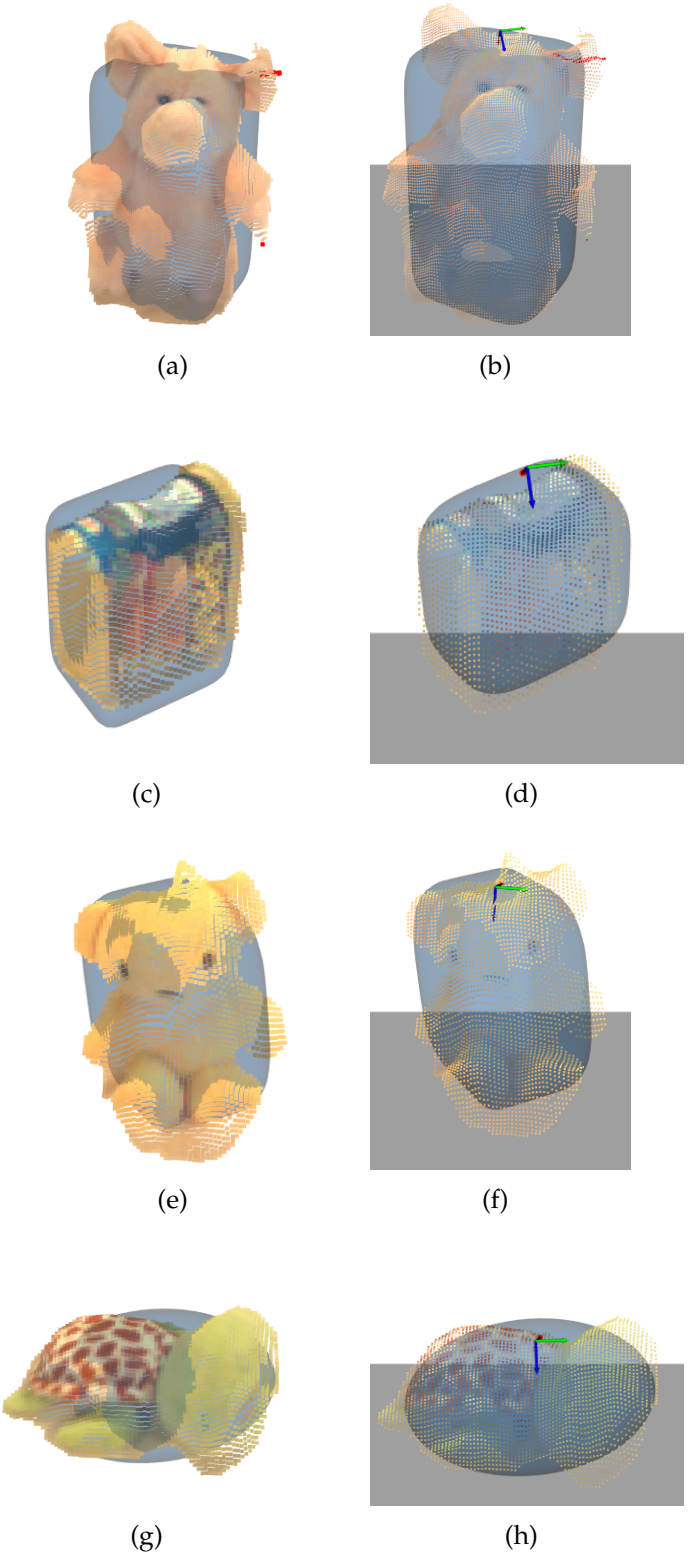


FIGURE 7.21: Some examples of superquadric models and grasping pose candidates for the 18 objects of the test set.

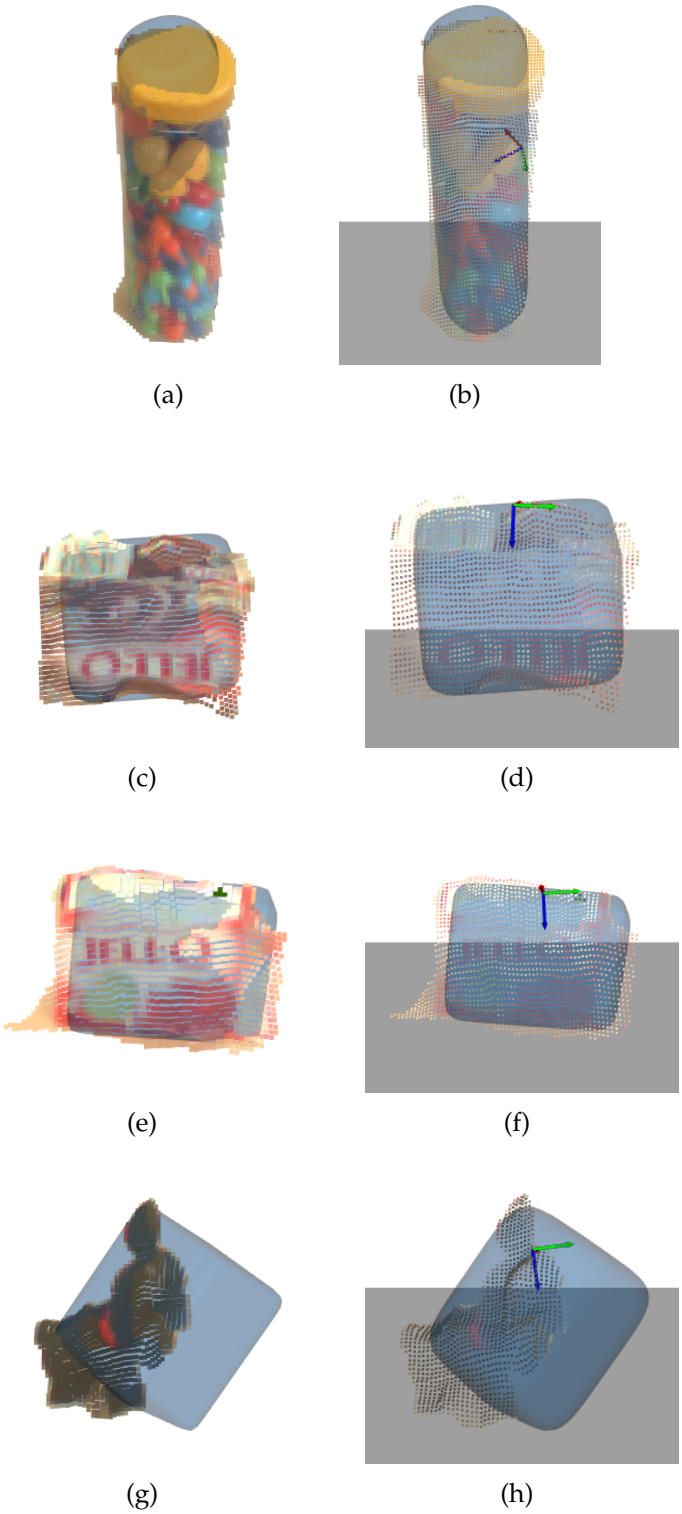


FIGURE 7.22: Some examples of superquadric models and grasping pose candidates for the 18 objects of the test set.

It is important to remark that the exploitation of 3D object models in pose computation allows considering even those portions of the object occluded by vision, as illustrated in Fig 7.19(b) and 7.20(f), where the computed poses bring the hand to touch the objects on the side and thus to place the fingers on the back. This is a remarkable advantage, because using only the visible portion of the object may lead to hand poses that appear not *natural* from the standpoint of human-like reaching or even not easily attainable for a humanoid robot, whose dexterity is often limited compared with that of humans.

Another advantage of our method is that we model the base on which the object is placed (e.g. the table), and impose constraints to avoid it in the optimization problem (the constraint in Eq. (7.4)). This feature allows the robot to grasp even small objects, without hitting the table with the fingers, as it is the case of the objects in Fig. 7.20(b), 7.20(d) and 7.20(h) (these objects are approximately 6 [cm] tall).

7.6.2 Robustness of the method

We select the subset of 6 objects shown in Fig. 7.23 in order to perform experiments on the method robustness. The objects were deliberately selected among the entire test set so as to be different in shape and dimensions (Table 7.2).

TABLE 7.2: Object Dimensions.

Object	Volume [m^3]	Object	Volume [m^3]
Cylinder	$0.06 \times 0.06 \times 0.20$	Ladybug	$0.16 \times 0.08 \times 0.08$
Cat	$0.10 \times 0.10 \times 0.10$	Lettuce	$0.07 \times 0.07 \times 0.07$
Bear	$0.09 \times 0.09 \times 0.12$	Turtle	$0.16 \times 0.10 \times 0.06$

Table 7.2 shows the dimensions of the object used in the experiments.

We compute the object models and grasping poses 10 times. The grasping poses computed for each object are compared in Fig. 7.24, where for the sake of clarity we show only one reconstructed superquadric $\mathcal{O}$ for each object, without showing the (overlapping) ellipsoid $\mathcal{H}$ that represents the hand and the object point cloud. The poses computed in each trial differ because the superquadrics that model the objects vary as a result of variations in the object segmentation and point cloud. Nevertheless, Fig.

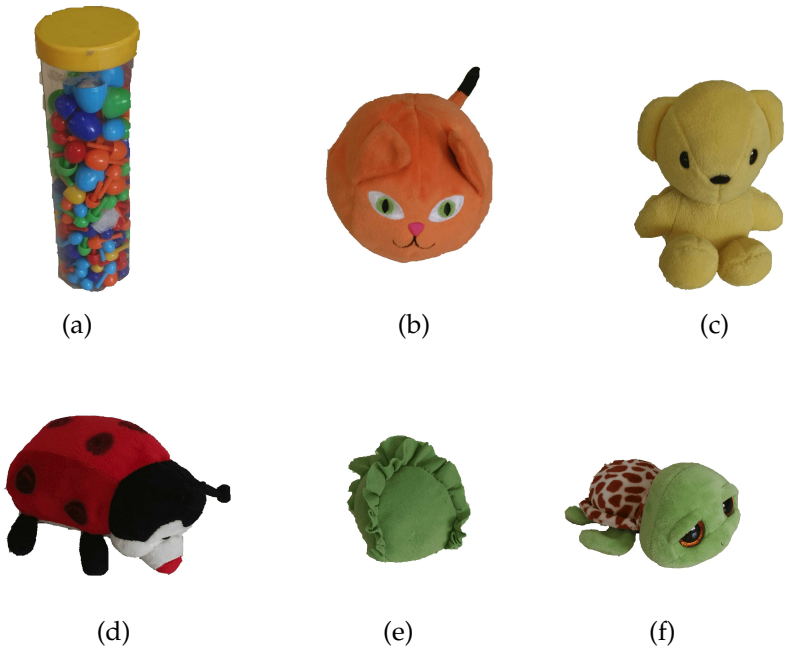


FIGURE 7.23: Objects used for robustness evaluation.

7.24 demonstrates that the desired poses computed with different models of the same object are affected by a small variability, thus guaranteeing a high grade of similarity and therefore underpinning the robustness of our method. For instance, the poses computed for object (a) are all very similar, representing the best grasp for a cylinder shape, that is located on the side at the middle of its height (Fig. 7.24(a)).

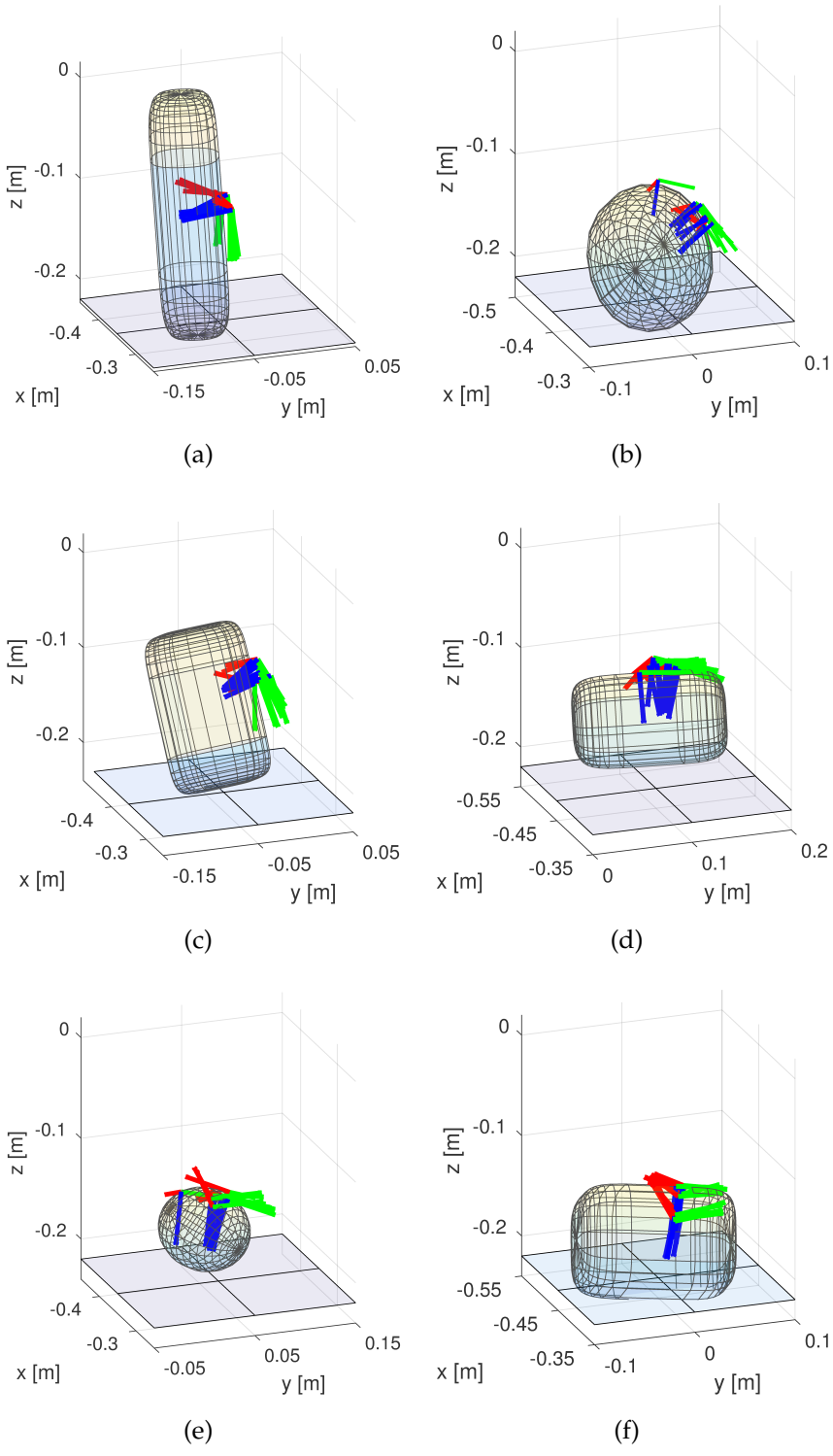


FIGURE 7.24: For each object 10 poses are shown. The letters identifying the different plots ((a) - (f)) correspond to different objects, according to the notation of Fig. 7.23.

7.6.3 The effect of prior on object shapes

In this Paragraph, our goal is to show the effect of prior information of the object shape on the modeling output both in terms of final model quality.

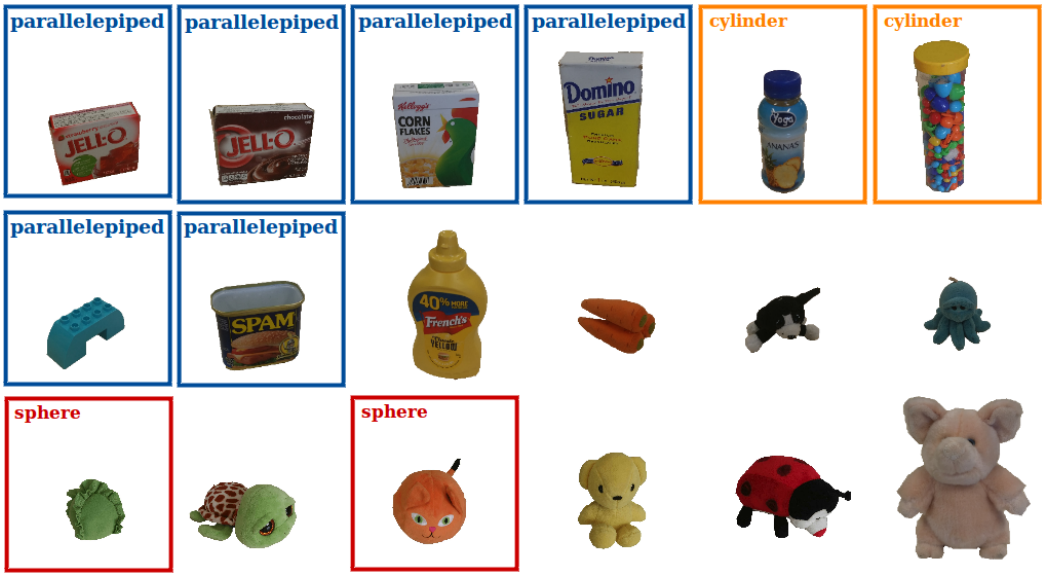


FIGURE 7.25: Classification results on the test set. The objects whose confidence is lower than a threshold for all the shape primitives are not classified and are considered as generic objects from the superquadric modeling process.

Fig. 7.25 shows how the objects of the test set are classified using the categorization system described in Paragraph 7.3.1. The objects that cannot be modeled with a shape primitive are considered as generic objects in the superquadric reconstruction step.

We noticed that prior information on object shapes helps obtain a finer and more sharp-cornered model that is crucial for computing better grasping poses for parallelepipeds, cylinders and spheres. Fig. 7.26 highlights how the use of prior on the object shape improves the model reconstructed for a box (i.e. a parallelepiped). In particular, the box model shown in Fig. 7.26 (c) leads to pose candidates on the top or on the lateral surfaces of the box, since the hand ellipsoid better overlaps on those portions of the object superquadric. If, instead, we use the model of Fig. 7.26 (a), the model made of rounded corners lets the final pose lie also on the box top corners (Fig. 7.26 (b)).

Other examples of the improvements obtained with using prior on object shapes are shown in Fig. 7.27.

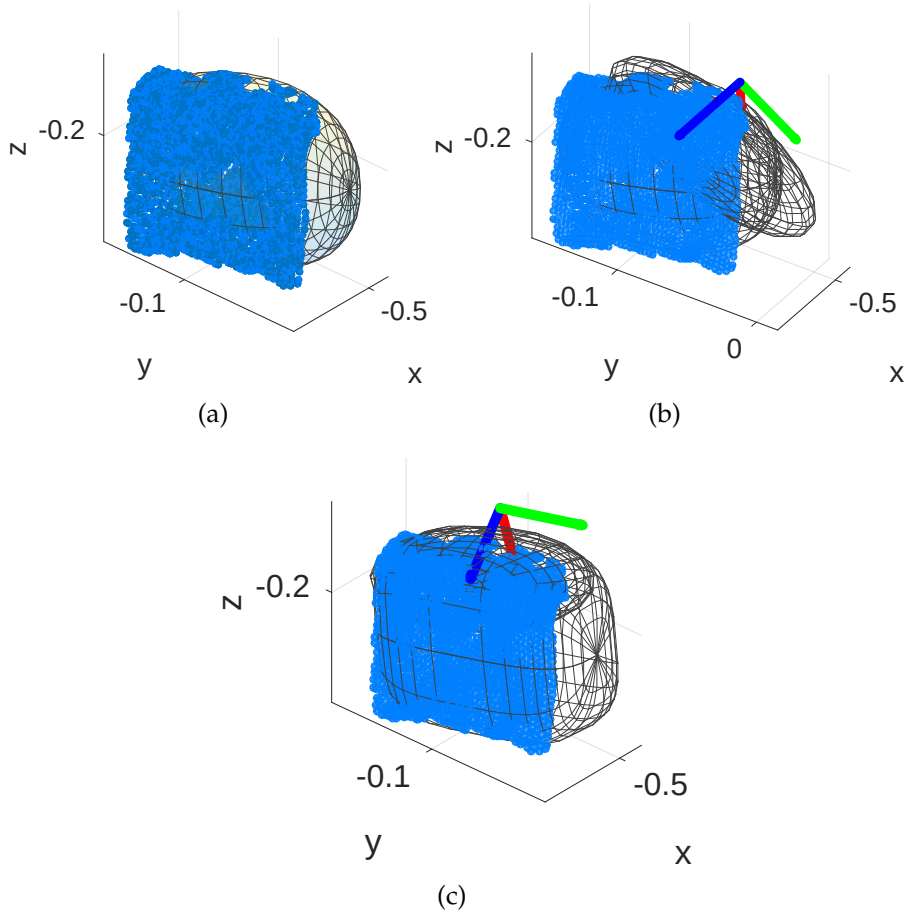
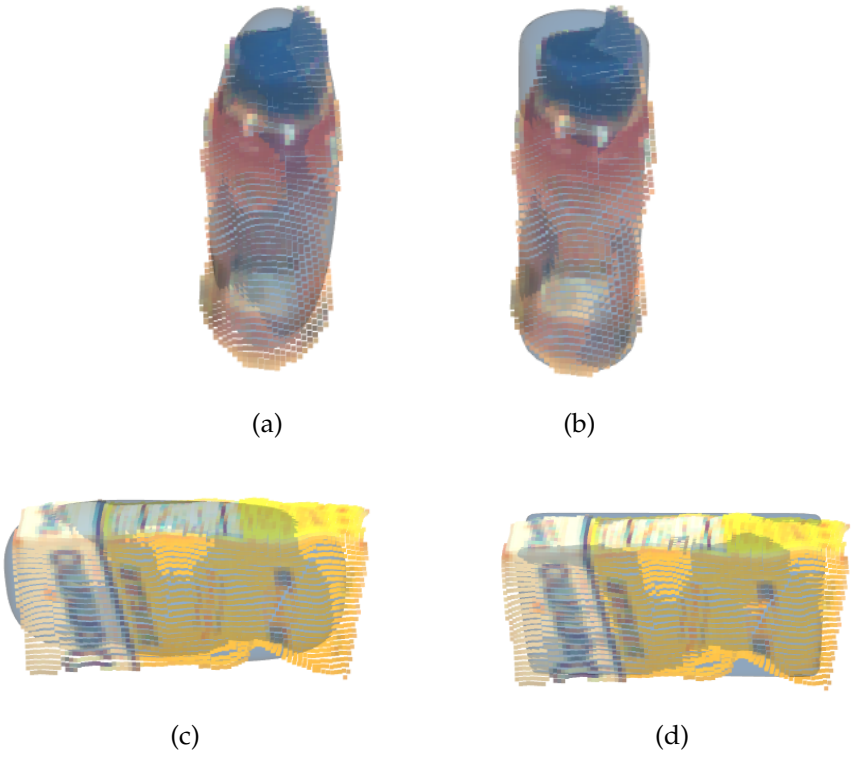


FIGURE 7.26: Superquadric models of the jello box 1 overlapped on the complete object point clouds (represented with blue dots). We show the superquadric obtained without any prior information on the object shape (Fig. (a)) and the relative grasping pose (Fig. (b)) and the same quantities obtained with the prior information (Fig. (c)). The model obtained with prior information has sharp-cornered shapes. The use of prior information enable to significantly downsample the object point cloud used for superquadric estimation (number of points used=50) and to obtain better grasping poses, i.e. located on the top surface of the box (Fig (c)) instead of on the box corners (Fig. (b)).



(a) (b)

(c) (d)

FIGURE 7.27: Some examples of the improvements obtained in terms of model accuracy by using prior on object shape (Fig.7.27(b) - 7.27(d)).

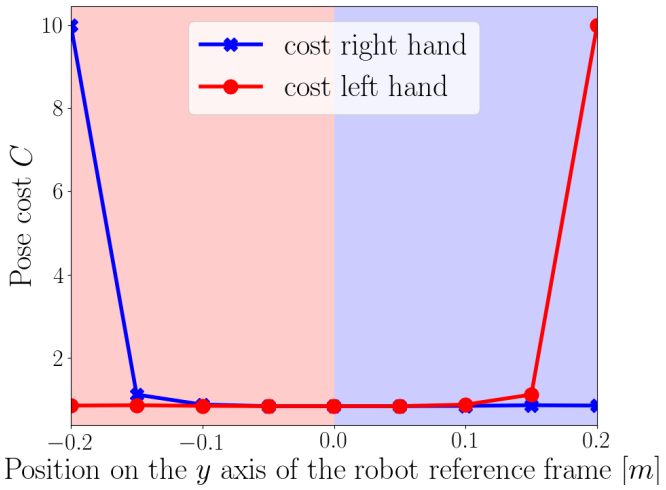


FIGURE 7.28: Pose costs for right hand and left hand of the jello box 1 in different positions. The costs have been computed sliding the object along the y axis of the robot reference frame from the left (in red) to the right (in blue) workspace.

7.6.4 Enlarging the workspace using two hands

In order to evaluate the effectiveness of our automatic approach for selecting the hand for grasping the object, we evaluate the pose costs $\mathcal{C}_{right}$ and $\mathcal{C}_{left}$ by varying the object position (with the same orientation) from the left hand to the right hand workspace. The trend of the costs obtained with the jello box 1 is shown in Fig. 7.28. As expected, the cost for each hand is lowerer in the hand workspace and increase while the object position goes towards the other hand workspace.

In addition, we executed the following experiment to show how the pose computation for both the hands and the selection of the best hand for grasping the object increases the number of successful grasps. We put the object of interest in a fixed position reachable by both the hands and we change only its orientation during each trial. Table 7.3 compares the success rate if respectively, only one hand or two hands are used for grasping the object. Even if the object is in a workspace sufficiently dexterous for the both hands, its orientation and reconstructed model can favor one hand with respect to the other, increasing the success percentage when the best hand is automatically selected for grasping the object.

TABLE 7.3: Experiment on best pose selection: Percentage of successful grasps.

Object	Success on Trials [%] One hand approach	Success on Trials [%] **Automatic hand selection**
Jello box 1	90%	**100**%
Jello box 2	80%	**90**%
Cereal box	70%	**90**%
Sugar box	70%	**90**%
Juice bottle	80%	**90**%
Cylinder	70%	**100**%
Lego brick	80%	**90**%
Meat box	60%	**80**%
Mustard box	70%	**90**%
Carrots	60%	**80**%
Dog	80%	**90**%
Octopus	90%	**100**%
Lettuce	70%	**90**%
Turtle	60%	**80**%
Cat	70%	**80**%
Bear	60%	**100**%
Ladybug	70%	**90**%
Pig	50%	**70**%

Table 7.3 shows the percentage of successful grasps, in case only one hand is used for grasping the hand and the automatic selection of the hand is implemented.

7.7 Discussion

In this Chapter, we proposed a novel approach for solving the grasping problem of unknown objects. In short, the idea of our approach is to use superquadrics to model the graspable volume of the hand and the objects from vision. These models are then used to compute a proper grasping pose solving a nonlinear constrained optimization problem. We showed how to add constraints so that the set of possible poses is limited to those that do not cause collisions with obstacles (e.g. the base on which the object stands) and do not lead to object penetration. Our approach is sufficiently generic to deal with objects and obstacles of different shape and size, and enables to specify further requirements on the robot pose by adding new constraints. In addition, we refined the superquadric modeling technique by using prior information on the object shape. The prior information

is provided by a visual object classifier we trained and integrated in the pipeline employing the recognition system of [212]. We finally proposed a pose cost for automatically selecting the best hand for grasping the object among two pose candidates for the right and the left hand.

We evaluated the improved pipeline on 18 real objects with the iCub humanoid robot. The experiments highlight how the overall success rate of the entire pipeline is nearly 85%. The main source of failures is represented by the uncalibrated eye-hand system of the robot that entails non-negligible misplacements of the robot hand when reaching for the target pose. This problem is peculiar of humanoid robots in that elastic elements lead to errors in the direct kinematics computation. Moreover, robots with moving cameras, such as the iCub platform, need to deal with errors in the visual estimation of the object pose due to imprecise knowledge of the cameras extrinsic parameters. These errors can be compensated by closed loop control techniques of the end-effector resorting to a visual feedback. At this regard, we improved the grasping reliability by integrating the visual servoing technique shown in [222], where the desired poses computed from stereo vision are accurately reached by the robot end-effector thanks to the use of a precise end-effector pose estimate over time (see Appendix B).

This Chapter reported exhaustive tests of the proposed techniques on the iCub humanoid robot. However, the approach does not depend on the platform and is portable to other humanoid robots. Good evidence of this is given by the implementation and testing of the same approach for the R1 robot (Fig. 7.29). Porting the modeling and grasping approach to this new robot just required the proper sizing of the hand ellipsoid $\mathcal{H}$. We plan to execute more modeling and grasping tests on R1 in the next future.

The pipeline we propose in this Chapter can be extended in several ways. At first, we are aware of the fact that a trajectory plan for reaching the final pose is still missing in our work. A viable solution is to use our approach also for computing a set of waypoints, together with the final grasping pose. At this aim, superquadrics could be used to model obstacles, and their avoidance added as optimization constraints as shown in Section 7.2. Another extension is the formulation of a supervised learning method for automatically discriminate good grasping poses. Our approach in fact only selects the best pose between two candidates, even if neither of them is suitable for grasping the object.

FIGURE 7.29: R1 grasping a wine paper bottle in a kitchen.

A strong limitation of the approach described in this section is that it works under the assumption that the object of interest can be well represented by a single superquadric. This does not hold generally for object with non-convex shapes, such as tools. As we will show in Chapter 8 the object model can be refined by using a set of superquadrics in place of only a single superquadric [104, 105]. This way, we can accurately model more complex and concave objects and, thus, compute proper grasping pose also on specific object portions.

Chapter 8

Modeling and grasping more complex objects using multiple superquadrics

The modeling and grasping pipeline described in Chapter 7 has been shown to be effective on a large number of simple objects. In summary, the method consists of estimating a *single* superquadric function representing the object and using such a model for computing a grasping pose for the robot hand.

The effectiveness of the approach relies on the assumption that the estimated superquadric is representative enough of the object volume and shape. As already mentioned in Section 7.1, our modeling process makes use only of convex superquadrics. This choice is prompted by the fact that convexity is a desirable property in optimization. In addition, considering also concave superquadrics does not increase their representative power in practical applications since non-convex superquadrics mostly represent shapes that are very uncommon in reality (Fig. 8.1). As a result, the modeling and grasping approaches described in the previous Chapter work effectively when applied on objects whose overall volumes and shapes can be properly represented or approximated with a single convex superquadric. Unfortunately, many everyday objects do not satisfy this hypothesis.

In order to remedy such a limitation, this Chapter proposes a novel extension of the modeling and grasping pipeline described in Chapter 7 to deal with more complex objects, i.e. objects that cannot be properly represented with a single superquadric. The leading idea of this contribution is to model objects with *multiple superquadrics* and use them as finer models to compute grasping pose candidates for specific portions of the

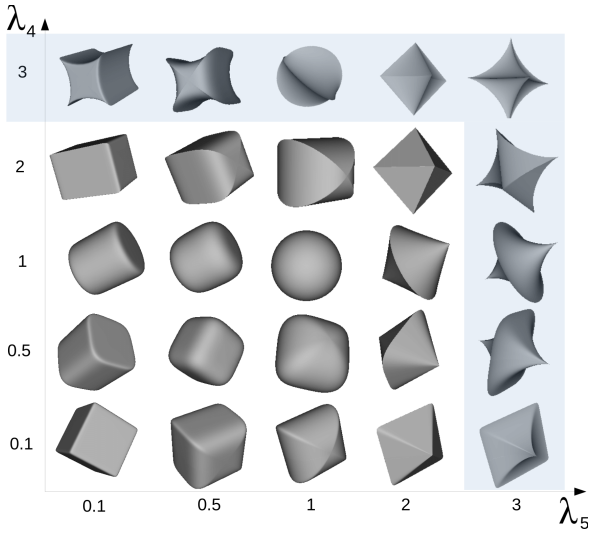


FIGURE 8.1: Non-convex superquadrics, i.e. with $\lambda_4, \lambda_5 > 2.0$, represent shapes that are very uncommon in everyday objects.

object. While the usage of multiple superquadrics is common to other works [105, 106, 112, 113], the modeling and grasping algorithms we design are novel in various respects, as it will be discussed in Section 8.4. Unlike the contributions presented thus far, the content of this Chapter is current under development. Our intention is to show the status of the approach and the interesting results that have been achieved.

The Chapter is organized as follows. Section 8.1 describes the new modeling process, that entails the estimation of multiple superquadrics for representing the object. In Section 8.2 we show how the grasping method described in Chapter 7 can be adapted to deal with multi-superquadrics object models. Section 8.3 reports a careful evaluation of the proposed approaches and Section 8.4 ends the Chapter discussing the current limitations and some ideas for improvements.

8.1 Multi-superquadrics modeling

The goal of multi-superquadric modeling is to estimate the minimum number S and the parameters of the superquadrics necessary for properly representing an object. The algorithm we propose in this Chapter consists of three steps that are detailed in the following Paragraphs:

1. Creating the *superquadric-tree* $\mathcal{ST}$ (Paragraph 8.1.1). The object point cloud is iteratively split in smaller portions using dividing planes and the corresponding superquadrics are estimated (Algorithm 4).

2. Inferring which of the dividing planes actually separate object portions that should be represented with different superquadrics, due to specific geometric properties (Paragraph 8.1.2, Algorithm 5).

3. Generating the final S superquadrics (Paragraph 8.1.3). The original object point cloud is split using only the selected planes and a superquadric for each portion is reconstructed (Algorithm 6).

The entire modeling process is obtained by executing consecutively Algorithms 4 - 5 - 6.

8.1.1 Creating the superquadric-tree

The superquadric tree $\mathcal{ST}$ is a binary tree with given height H containing $\sum_{h=0}^{H} 2^h$ nodes. The node with height $h = 0$, named $\mathcal{N}_0$, is the *root* of the tree. Every node, except for the leaves (i.e. nodes with height $h = H$), has a *left* and *right* child. We refer to nodes with height $h > 0$ as

$$\mathcal{N}_{side,h}, \tag{8.1}$$

where $side \in \{l, r\}$ indicates whether the node is a *left* or *right* child with respect to its father and h is the height of the node. Fig. 8.2 shows an example of superquadric-tree with $H = 3$.

Each node $\mathcal{N}_{side,h}$ contains the following quantities:

- A portion of the object point cloud $pc_{side,h}$;
- The relative superquadric $\mathcal{S}_{side,h}$;
- The plane $p_{side,h}$ passing through the point cloud barycenter and perpendicular to the axis with maximum inertia.

The root $\mathcal{N}_0$ instead contains the entire object point cloud $p_0 = pc$.
The superquadric-tree is generated according to the following procedure, outlined in Algorithm 4.

For each node $\mathcal{N}_{side,h}$ for $h = 0, \ldots, H-1$ and for $side \in \{left, right\}$:

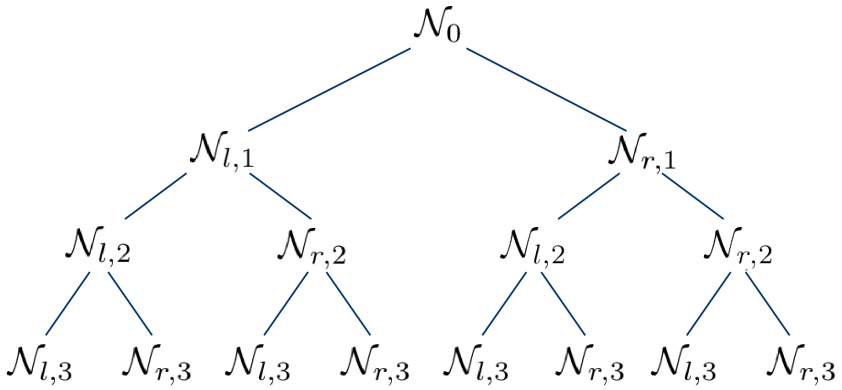


FIGURE 8.2: An example of superquadric-tree with height $H = 3$.

1. We compute the plane $p_{side,h}$ passing through the barycenter of the point cloud $pc_{side,h}$ and perpendicular to its axis with maximum inertia.

2. The point cloud $pc_{side,h}$ is then split in two portions $pc_{l,h+1}$ and $pc_{r,h+1}$ using the plane computed in step 1 and they are stored respectively in the node children $\mathcal{N}_{l,h+1}$ and $\mathcal{N}_{r,h+1}$.

3. Finally, the superquadrics $\mathcal{S}_{l,h+1}$ and $\mathcal{S}_{r,h+1}$ are estimated by fitting respectively $pc_{l,h+1}$ and $pc_{r,h+1}$ and are stored in the nodes $\mathcal{N}_{l,h+1}$ and $\mathcal{N}_{r,h+1}$.

Fig. 8.3 reports the superquadric-tree computed for a tool.

8.1.2 Inferring the splitting planes

This step is the core part of the proposed multi-superquadric modeling method. The basic idea is to identify among the superquadric-tree planes $\{\{p_{side,h}\}_{h=0}^{H-1}\}_{side \in l,r}$ those planes that divide the object point cloud passing through *relevant* regions. Our intuition suggests to consider as relevant those regions of the point clouds that correspond to a change of concavity and/or dimensions. We refer to the corresponding planes as *splitting planes*. Fig. 8.4 shows some examples of desired splitting planes.

Instead of relying on point cloud properties, we find the splitting planes using the superquadric-tree introduced in the previous Paragraph. This way, the approach is more robust to possible point cloud noise in that the

Algorithm 4 Generating the superquadric-tree $\mathcal{ST}$

Data: Object point cloud pc, desired final tree height H;
Initialize the root point cloud with the object point cloud, i.e. $p_0 = pc$ and compute the superquadric S_0 fitting the entire point cloud.
Initialize height of the tree $h = 0$.
while $h < H$ **do**:
    for $side \in \{l, r\}$ **do**:
        if $h = 0$ **then**:
            Compute the plane p_h;
        else
            Compute the plane $p_{side,h}$;
        end if
        Split the point cloud $pc_{side,h}$ into $pc_{l,h+1}$ and $pc_{r,h+1}$;
        Compute a superquadric for each portion $\mathcal{S}_{l,h+1}$ and $\mathcal{S}_{r,h+1}$;
        Store both the point clouds $pc_{l,h+1}$ and $pc_{r,h+1}$ and the superquadrics $\mathcal{S}_{l,h+1}$ and $\mathcal{S}_{r,h+1}$ in the children nodes $\mathcal{N}_{l,h+1}$ and $\mathcal{N}_{r,h+1}$.
        $h = h + 1$;
    end for
end while
Output: the superquadric-tree $\mathcal{ST}$ with height H.

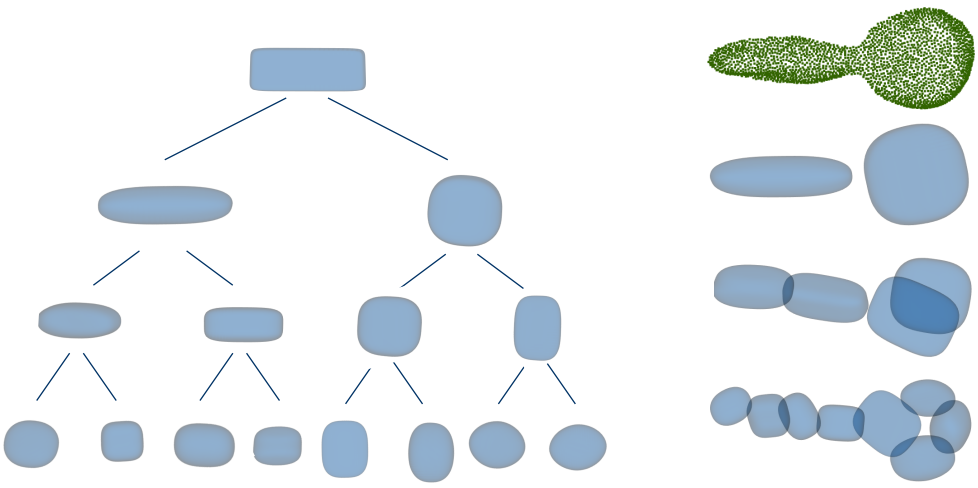


FIGURE 8.3: An example of superquadric tree computed for a tool (on the left). The corresponding object representation using different numbers of superquadrics is shown on the right. The object point cloud is represented with green dots.

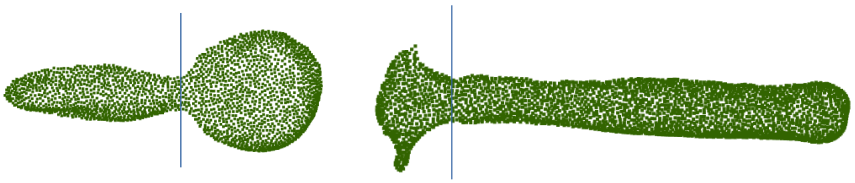


FIGURE 8.4: Some examples of desired splitting planes represented with blue lines.

reconstructed superquadrics are able to filter zero-mean noise of the point clouds. This kind of noise in fact generates a rippling effect on the point cloud. Since the superquadric is estimated by solving a least square problem (see Eq. (7.2)), the superquadric surface turns out to approximate the mean of the noisy point cloud.

Our basic idea consists of marking as splitting planes those dividing pairs of superquadrics having some of their dimensions different to each other, avoiding separating point cloud regions that can actually be well represented with a single superquadric.

Hereafter we describe in detail the methodology adopted to infer if the plane $p_{side,h-1}$ is a splitting plane by analyzing geometric properties of the superquadrics $\mathcal{S}_{l,h}$ and $\mathcal{S}_{r,h}$ of the children nodes. In the final description of the algorithm we will refer to this methodology as the *differentDimensions(·, ·)* function. The procedure consists of the following steps:

- Given the superquadrics $\mathcal{S}_{l,h}$ and $\mathcal{S}_{r,h}$ with orientations $R_{l,h} = [\boldsymbol{x}_{l,h}, \boldsymbol{y}_{l,h}, \boldsymbol{z}_{l,h}]$ and $R_{r,h} = [\boldsymbol{x}_{r,h}, \boldsymbol{y}_{r,h}, \boldsymbol{z}_{r,h}]$, we pair each axis of $\mathcal{S}_{l,h}$ with the axis of $\mathcal{S}_{r,h}$ that is parallel to. In case none of the axes of $\mathcal{S}_{r,h}$ is parallel to the axes of $\mathcal{S}_{l,h}$, we pair the axes providing the maximum dot product. This information is encoded in a rotation matrix R_r^l, i.e. $R_{l,h} = R_r^l R_{r,h}$ (Fig. 8.5).

- Let's refer to the dimensions of $\mathcal{S}_{l,h}$ and $\mathcal{S}_{r,h}$ respectively as $dim_l = (\lambda_{l,1}, \lambda_{l,2}, \lambda_{l,3})$ and $dim_r = (\lambda_{r,1}, \lambda_{r,2}, \lambda_{r,3})$. According to its definition, the matrix R_r^l also encodes the information to reorder the vector dim_r in $\overline{dim}_r = R_r^l \cdot dim_r$ so that $dim_l[i]$ and $\overline{dim}_r[i]$ for $i = 1, \ldots, 3$ correspond to the dimensions along the axes of the two superquadrics that are parallel among each other (Fig. 8.6).

- We then check the similarity between dim_l and $\overline{dim}_r$. Superquadrics with similar dimensions should be merged and, therefore, the plane

that led to their generation is not a splitting plane. However, it is important to stress out that not all the superquadric dimensions should be compared. In particular, if the parallel axes of the i-th pair are contiguous[1], the respective dimensions $dim_l[i]$ and $\overline{dim}_r[i]$ can be arbitrary (Fig. 8.6). In summary, we compare the dimensions $dim_l[i]$ and $\overline{dim}_r[i]$ for each pair i of *not contiguous parallel axes*. If those dimensions are comparable (within a given tolerance μ), i.e. $|dim_l[i] - \overline{dim}_r[i]| < \mu$, the plane $p_{side,h-1}$ is not considered a splitting plane.

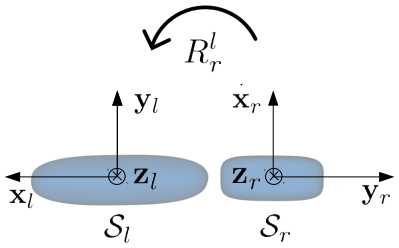


FIGURE 8.5: Matrix R_r^l encodes the information of which axes of superquadric S_r are parallel to axes of S_l. In this particular example $R_r^l = [0, 1, 0; -1, 0, 0; 0, 0, 1]$.

The *differentDimensions(·, ·)* function is applied to each node $\mathcal{N}_{side,h}$ for $h = H - 1, \ldots, 0$. If the application of this function returns that the plane of node $\mathcal{N}_{side,h}$ is a splitting plane, a further step is required, as shown in the example of Fig. 8.7. The plane $p_{l,1}$ of node $\mathcal{N}_{l,1}$ is a splitting plane since the children superquadrics $\mathcal{S}_{l,2}$ and $\mathcal{S}_{r,2}$ do not satisfy the conditions on their dimensions to be mergeable. In this case, it is important to check the relationship between the dimensions of one of the children superquadrics and its *uncle* superquadric. The uncle of a node is defined as the brother of its father. More formally, if $\mathcal{N}_{side,h}$ is the father of $\mathcal{N}_{l,h+1}$ and $\mathcal{N}_{r,h+1}$, their uncle is $N_{\overline{side},h}$, where $\overline{side} = right$ if $side = left$ and viceversa. Still referring to the example of Fig. 8.7, the uncle of $\mathcal{N}_{l,2}$ and $\mathcal{N}_{r,2}$ is $\mathcal{N}_{r,1}$. Due to the process of generation of the superquadric-tree, one of the nephew is contiguous to the uncle, $\mathcal{N}_{r,2}$ in this case. We apply then the *differentDimensions(·, ·)* function to the uncle $\mathcal{N}_{r,1}$ and its closest nephew $\mathcal{N}_{r,2}$. If their dimensions satisfy the similarity condition we explained before, this means that the plane p_0 of the grandfather (and father) $\mathcal{N}_0$ of $\mathcal{N}_{r,2}$ (and $\mathcal{N}_{r,1}$) is not a splitting

[1]In other words, if they lie approximately on the same line, i.e. besides being parallel two of their extremes are close to each other.

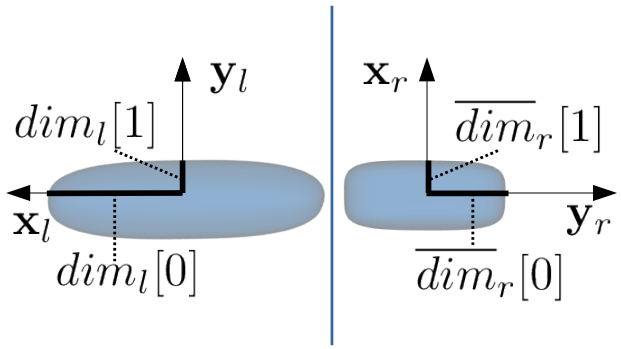


FIGURE 8.6: Example showing why the dimensions along parallel and contiguous axes can be arbitrary. In this example, the *left* superquadric $\mathcal{S}_l$ have the x-axis parallel to y-axis of the *right* superquadric $\mathcal{S}_r$, i.e. $\boldsymbol{x}_l$ parallel to $\boldsymbol{y}_r$, and $\boldsymbol{y}_l$ parallel to $\boldsymbol{x}_r$. The axes $-\boldsymbol{x}_l$ and $-\boldsymbol{y}_r$ are also contiguous. The dimensions to be compared are just $dim_l[1]$ with $\overline{dim_r}[1]$, since they are the dimensions along parallel but not contiguous axes. Their dimensions are similar, therefore the plane is not relevant for splitting the point cloud. If we instead compared also $dim_l[0]$ with $\overline{dim_r}[0]$ our algorithm would say that the plane is a splitting plane, since $|dim_l[0] - \overline{dim_r}[0]| > \mu$.

plane, as happens in the case of Fig. 8.7. In the opposite case instead also the plane p_0 of the grandfather (and father) $\mathcal{N}_0$ is considered a splitting plane.

Fig. 8.8 explains the reason why comparing the dimensions of one child with its uncle is required in case the plane of the father is a splitting plane. Checking the plane p_0 with *differentDimensions*$(\cdot,\cdot)$ function applied to $\mathcal{S}_{l,1}$ and $\mathcal{S}_{r,1}$ would return a positive answer. The superquadric $\mathcal{S}_{l,1}$ is in fact quite big in order to cover the entire point cloud portion $pc_{l,1}$ and very different in dimensions[2] with respect to the brother superquadric $\mathcal{S}_{r,1}$. If we go further in the approximation however, we obtain that a finer representation of point cloud $pc_{l,1}$ generates two superquadrics $\mathcal{S}_{l,2}$ and $\mathcal{S}_{r,2}$, one of which very similar in dimensions to its uncle $\mathcal{S}_{r,1}$. This happens because the plane $p_{l,1}$ cuts the point cloud in a relevant part - corresponding of a change of dimensions - while plane p_0 is applied on a point cloud portion without any change of dimensions or concavity.

[2]For the sake of simplicity, from now on we will simply say that two superquadrics have similar or different dimensions if they satisfy or not the conditions previously explained.

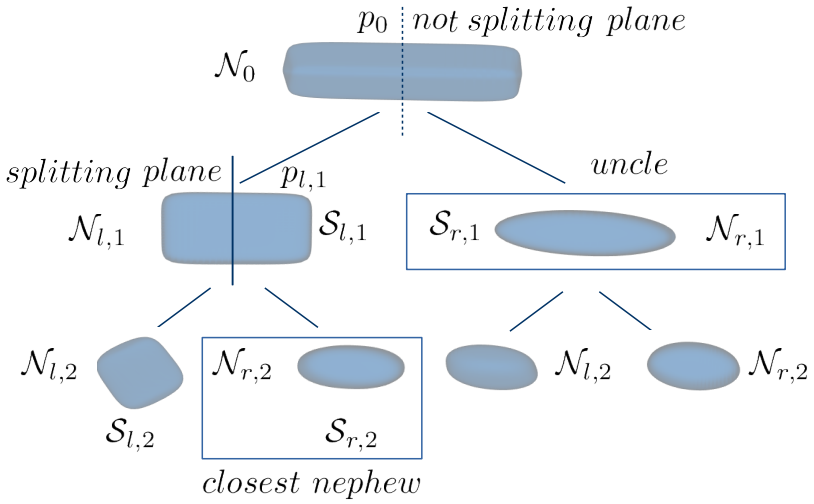


FIGURE 8.7: Example of comparison between nephew and uncle superquadric for a hammer (point cloud in Fig. 8.8). The superquadrics $\mathcal{S}_{l,2}$ and $\mathcal{S}_{r,2}$ do not satisfy the dimensions criteria, therefore the plane of $\mathcal{N}_{l,1}$ is a splitting plane. After comparing the superquadric of the node uncle $\mathcal{N}_{r,1}$ with the closest nephew $\mathcal{S}_{r,2}$, it turns out that the plane of node $\mathcal{N}_0$ is not important for splitting the point cloud, in that the two superquadrics do satisfy the dimensions criteria.

The final Algorithm for finding the splitting planes is detailed in Algorithm 5. In summary, we proceed bottom-up, starting from the tree level with height $h = H - 1$.

Then, for each node $\mathcal{N}_{side,h}$ for $side = \{l, r\}$ and $h = H - 1, \ldots, 0$:

1. We evaluate the function *differentDimensions*$(\cdot, \cdot)$ on $\mathcal{S}_{l,h+1}$ and $\mathcal{S}_{r,h+1}$.

2. If the superquadrics are different in dimensions, the plane of the father $p_{side,h}$ is a splitting plane. We then evaluate the function *differentDimensions*$(\cdot, \cdot)$ on $\mathcal{S}_{nephew,h+1}$ and $\mathcal{S}_{uncle,h}$. In case the test highlights that also $\mathcal{S}_{nephew,h+1}$ and $\mathcal{S}_{uncle,h}$ are different in dimensions, also the plane $p_{side,h-1}$ is a splitting plane.

3. If instead the superquadrics are similar in dimensions, the plane of the father $p_{side,h}$ is not important for splitting the point cloud.

Algorithm 5 Inferring splitting planes

Data: Superquadric-tree $\mathcal{ST}$, with height H generated with Alg. 4;
Threshold μ for *differentDimensions*$(\cdot,\cdot)$ function;
Initially all the planes $\{\{p_{side,h}\}_{h=H-1}^{0}\}_{side\in\{l,r\}}\}$ are considered not important for splitting the point cloud;
for $h = H-1,\dots,0$ and $side \in \{l,r\}$ **do**:
  if $p_{side,h}$ is not a splitting plane **then**:
    Apply *differentDimensions*$(\mathcal{S}_{l,h+1}, \mathcal{S}_{r,h+1})$;
    if *differentDimensions*$(\mathcal{S}_{l,h+1}, \mathcal{S}_{r,h+1})$ returns false **then**:
      Plane $p_{side,h}$ is not a splitting plane;
    else
      Plane $p_{side,h}$ is a splitting plane;
      Compute *differentDimensions*$(\mathcal{S}_{nephew,h+1}, \mathcal{S}_{\overline{side},h})$,
      where $\mathcal{S}_{nephew,h+1}$ is the closest superquadric among
      $\mathcal{S}_{l,h+1}, \mathcal{S}_{r,h+1}$ with respect to their uncle $\mathcal{S}_{\overline{side},h}$;
      if *differentDimensions*$(\mathcal{S}_{nephew,h+1}, \mathcal{S}_{\overline{side},h})$ returns false **then**:
        Plane $p_{side,h-1}$ is not a splitting plane;
      else
        Plane $p_{side,h-1}$ is a splitting plane;
      end if
    end if
  end if
end for
Output: Splitting planes for cutting the point cloud.

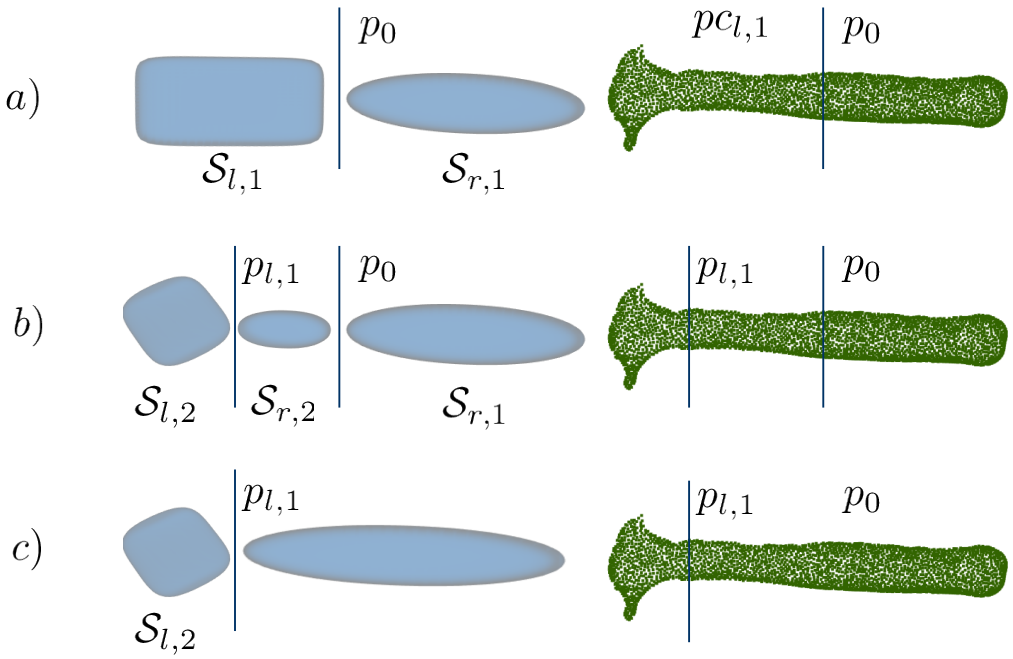


FIGURE 8.8: Different multi-superquadric models obtained using different planes of the superquadric-tree. Case a) shows the multi-superquadric model when only plane p_0 is considered a splitting plane. The model of case b) instead is obtained by cutting the point cloud both with p_0 and $p_{l,1}$ planes. Case c) can be instead obtained after comparing the closest nephew with its uncle superquadrics (see Fig. 8.7) and concluding that only $p_{l,1}$ is important for splitting the point cloud.

8.1.3 Generating the final superquadrics

Once the splitting planes for cutting the object point cloud has been computed, we use them to divide the point cloud into $\mathcal{S}$ regions and we fit a superquadric for each region (Algorithm 6).

8.2 Multi-superquadrics grasping

Modeling a single object with S superquadrics allows distinguishing portions with different volumes, shapes and, therefore, grasping properties. In order to fully exploit the information encoded in the object model, the

Algorithm 6 Computation of the final $\mathcal{S}$ superquadrics

Input: Superquadric-tree $\mathcal{ST}$ obtained from Algorithm 4 and splitting planes computed with Algorithm 5;
Cut the object point cloud using the splitting planes and generating $\mathcal{S}$ point clouds $pc_{new,i}$ for $i = 1, \ldots, \mathcal{S}$;
for $i = 1, \ldots, \mathcal{S}$ **do**:
    if $pc_{new,i} \in \{\{pc_{side,h}\}_{h=0}^{H}\}_{side \in \{l,r\}}$ and thus it belongs to $\mathcal{ST}$ **then**:
        Set the superquadric $\mathcal{S}_{side,h}$ as one of the final $\mathcal{S}$ superquadrics;
    else
        Compute the superquadric fitting the new point cloud $pc_{new,i}$;
    end if
end for
Output: Final S superquadrics modeling the object.

robot hand pose $\boldsymbol{x} = [x_h, y_h, z_h, \phi_h, \theta_h, \psi_h]$[3] for grasping the object is obtained by executing the following two steps:

- a grasping pose candidate is computed for each object region p represented by the superquadric $\mathcal{S}_p$;
- the best grasping pose among the S candidates is selected according to a given proper criterion.

In Paragraph 8.2.1 we show how to compute each grasping candidate by extending the optimization problem presented in Section 7.2. Then, Paragraph 8.2.2 reports on the criteria used for selecting the best pose among the grasping candidates.

8.2.1 Grasping pose candidates computation using multi-superquadric models

The result of the modeling process described in Section 8.1 is an object model consisting of S superquadrics. The first step of the new grasping pose computation approach consists of computing a pose candidate $\boldsymbol{x}_p \in R^6$ for each object portion represented by the superquadric $\mathcal{S}_p$, while avoiding the other superquadrics S_k for $k = 1, \ldots, \mathcal{S}$ and $k \neq p$. Thereby, the S grasp candidates $\boldsymbol{x}_p$ for $p = 1, \ldots, S$ are computed by solving S optimization problems:

[3]The orientation of the robot hand pose is expressed with Euler angles.

$$
\begin{aligned}
&\text{for } p = 1, \ldots, S \\
&\qquad \text{Solve :} \\
&\qquad\quad \min_{\boldsymbol{x}} \sum_{i=1}^{L} \left(\sqrt{\lambda_{1,p}\lambda_{2,p}\lambda_{3,p}} \left(F_p(\boldsymbol{p}_i^{\boldsymbol{x}}, \boldsymbol{\lambda_p}) - 1 \right) \right)^2, \\
&\qquad\quad \text{s. t.:} \\
&\qquad\qquad h_i(\boldsymbol{a_i}, f_i(\boldsymbol{p}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{p}_L^{\boldsymbol{x}})) > 0 \\
&\qquad\qquad \text{for } i = 1, \ldots, M + K, \\
&\qquad\qquad \sqrt{\lambda_{1,k}\lambda_{2,k}\lambda_{3,k}} \sum_{i=1}^{V} \left(F_k \left(\boldsymbol{v}_i^{\boldsymbol{x}}, \boldsymbol{\lambda_k} \right) - 1 \right) > 0 \\
&\qquad\qquad \text{for } k = 1, \ldots, S \text{ and } k \neq p.
\end{aligned}
\tag{8.2}
$$

The structure of each optimization problem p is very similar to the original one proposed in Eq. (7.4). Hereafter, we explain the mathematical quantities more relevant for dealing with a multi-superquadric object model. The extensive explanation of the other components is available in Section 7.2.

- The cost function $\left(\sqrt{\lambda_{1,p}\lambda_{2,p}\lambda_{3,p}} \left(F_p(\boldsymbol{p}_i^{\boldsymbol{x}}, \boldsymbol{\lambda_p}) - 1 \right) \right)^2$ imposes the overlapping between the hand ellipsoid $\mathcal{H}$ (introduced in Section 7.1.2) and the superquadric $\mathcal{S}_p$, named **target superquadric** $\mathcal{T}_p$. $\mathcal{T}_p$ is the superquadric - and thus the object region - where we want the grasping pose $\boldsymbol{x}_p$ to be located. The target superquadric $\mathcal{T}_p$ is represented by the inside-outside function $F_p(\boldsymbol{\cdot}, \boldsymbol{\lambda}_p)$, where $\boldsymbol{\lambda}_p$ are the 11 parameters representing the superquadric shape, dimensions and pose.

- The constraints $h_i(\boldsymbol{a_i}, f_i(\boldsymbol{p}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{p}_L^{\boldsymbol{x}})) > 0$ for $i = 1, \ldots, M + K$ represent with a unique generic formulation the $K = 3$ constraints on the orientation (Section 7.2.3) and the unique constraint ($M = 1$) for the avoidance of the support on which the object is located (Section 7.2.2)[4]. These constraints does not depend on the superquadrics involved and thus are the same in the S optimization problems.

[4]For the sake of clarity, in this formulation we only report the constraints that we used for the evaluation of our approach, three for the orientation and one for avoiding the support on which the object is located. Our formulation however could deal also with additional constraints.

- The constraints $\sqrt{\lambda_{1,k}\lambda_{2,k}\lambda_{3,k}}\sum_{i=1}^{V}\left(F_k\left(\boldsymbol{v}_i^{\boldsymbol{x}},\boldsymbol{\lambda_k}\right)-1\right)\right) > 0$ for $k = 1,\ldots,S$ and $k \neq p$ form the core part of Eq. (8.2). For each optimization problem p with target superquadric $\mathcal{T}_p$, they impose the avoidance of the other $S-1$ superquadrics S_k for $k = 1,\ldots,S$ and $k \neq p$. These superquadrics are named **obstacle superquadrics** $\mathcal{OS}_k$ and are represented by the inside-outside function $F_k(\cdot,\boldsymbol{\lambda}_k)$ with parameters $\boldsymbol{\lambda}_k$. The left side of these constraints is very similar to the cost function since they both rely on the inside-outside function of superquadrics. The *cost function* imposes *the hand ellipsoid* $\mathcal{H}$ to be *overlapped* onto the target superquadric $\mathcal{T}_p$ by minimizing the distance between the points $\boldsymbol{p}_1^{\boldsymbol{x}},\ldots,\boldsymbol{p}_L^{\boldsymbol{x}}$ sampled on $\mathcal{H}$ and the superquadric $\mathcal{T}_p$ itself (see Section 7.2 for more details). The *constraints* instead are used to locate the *hand palm and fingers outside* each obstacle superquadric $\mathcal{OS}_k$ for $k = 1,\ldots,S$ and $k \neq p$, by imposing the average value of inside-outside function of the superquadric evaluated in the points $\boldsymbol{v}_1^{\boldsymbol{x}},\ldots,\boldsymbol{v}_V^{\boldsymbol{x}}$ to be positive[5]. The quantities $\boldsymbol{v}_i^{\boldsymbol{x}}$ for $i = 1,\ldots V$ are used for representing in a compact way the robot hand and fingers occupancy in pose $\boldsymbol{x}$. In particular, we use the following $V = 5$ points (Fig. 8.9):

  - The palm center $\boldsymbol{v}_1 = [x_h, y_h, z_h]$.
  - The palm edges, given by:

$$\begin{aligned}\boldsymbol{v}_2 &= \boldsymbol{v}_1 + \frac{w_p}{2}\boldsymbol{y}_h,\\ \boldsymbol{v}_3 &= \boldsymbol{v}_1 - \frac{w_p}{2}\boldsymbol{y}_h,\end{aligned} \tag{8.3}$$

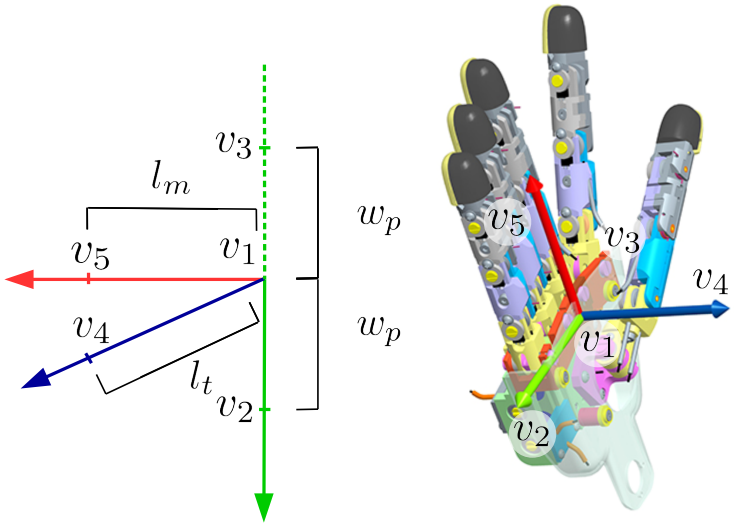
    where w_p is the palm width and $\boldsymbol{y}_h$ is the $y-$axis of the hand reference frame in pose $\boldsymbol{x}$.

  - The thumb fingertip position:

$$\boldsymbol{v}_4 = \boldsymbol{v}_1 \pm l_t\boldsymbol{z}_h, \tag{8.4}$$

    where l_t is the thumb length and $\boldsymbol{z}_h$ is the $z-$axis of the hand reference frame in pose $\boldsymbol{x}$. The sign $\pm$ varies according to the

[5] As said in Chapter 7, the inside-outside function $F(\lambda,\cdot)$ representing a superquadric provides a way to check if a given point $\boldsymbol{p} \in R^3$ lays inside ($F(\lambda,\boldsymbol{p}) < 0$), outside ($F(\lambda,\boldsymbol{p}) > 0$) or on ($F(\lambda,\boldsymbol{p}) = 0$) the surface of the superquadric.



(a) (b)

FIGURE 8.9: Points used for representing right hand and fingers occupancy. The hand reference frame is represented with the RGB convention (x_h colored in red, y_h in green, z_h in blue) Fig 8.9(a) shows how the edges v_i for $i = 1, \ldots, 5$ are computed. Fig 8.9(b) helps to understand how these quantities are representative of the hand and fingers volume occupancy. During the execution of the grasp the thumb is aligned with z_h of the hand reference frame, unlike the configuration shown in Fig. 8.9(b)

hand taken into account, since z-axes of the two hand reference frames have opposite directions.

– The middle fingertip position:

$$\boldsymbol{v}_5 = \boldsymbol{v}_1 + l_m \boldsymbol{x}_h, \tag{8.5}$$

where l_m is the middle finger length and $\boldsymbol{x}_h$ is the $x-$axis of the hand reference frame in pose $\boldsymbol{x}$.

Thank to this simple representation we impose the robot hand and fingers in the grasp candidate $\boldsymbol{x}_p$ not to lay inside the other obstacle superquadrics $\mathcal{OS}_k$ and thus, other object portions.

In summary, each solution $\boldsymbol{x}_p$ for $p = 1, \ldots, S$ is computed in order to :

- make the hand ellipsoid $\mathcal{H}$ overlapped on the target superquadric $\mathcal{T}_p$;
- avoid the support on which the object is located;
- provides a reachable orientation;
- avoid the other $S-1$ obstacle superquadrics $\mathcal{OS}_k$ for $k = 1, \ldots, S$ and $k \neq p$.

Fig. 8.10 shows an example of the grasp candidates computation for a tool represented with two superquadrics ($S = 2$). Two optimization problems are formulated and solved. First, i.e. $p = 1$ (Fig. 8.10(a)), the handle of the tool is the target superquadric $\mathcal{T}_1$ and the head of the tool is the only obstacle superquadric $\mathcal{OS}_2$. The solution $\boldsymbol{x}_1$ is located on the tool handle such that to avoid the head of the tool (Fig. 8.10(b)). In the second optimization problem ($p = 2$) instead, the head of the tool is the target superquadrics $\mathcal{T}_2$ and the handle is considered the obstacle $\mathcal{OS}_1$ (Fig. 8.10(c)). In this case the final solution $\boldsymbol{x}_2$ is located on the head of the tool (see Fig 8.10(d)). In both the optimization problems, the solutions are computed by avoiding the support on which the object is located and providing an orientation reachable by the robot hand.

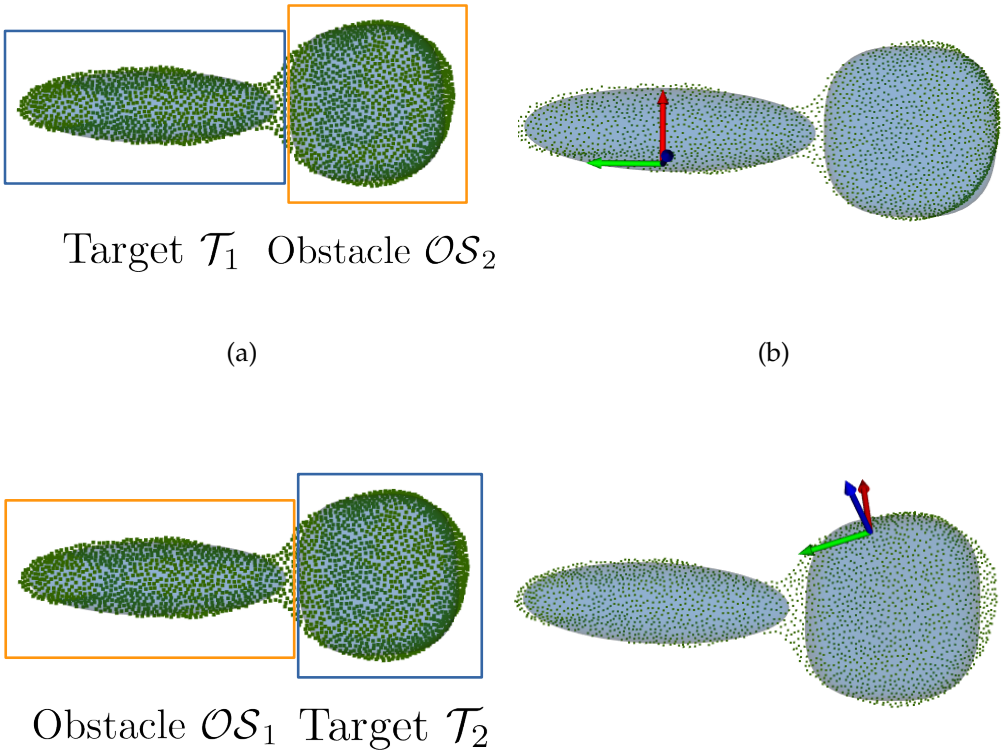


FIGURE 8.10: Example of left hand grasp candidates computed for an object represented with two superquadrics. The poses are represented with the RGB convention ($\boldsymbol{x}_h$ colored in red, $\boldsymbol{y}_h$ in green, $\boldsymbol{z}_h$ in blue). Fig. 8.10(a) shows the target superquadric $\mathcal{T}_1$ and the obstacle superquadrics $\mathcal{OS}_2$ used in the first optimization problem, i.e. $p = 1$. Fig. 8.10(b) reports the relative solution $\boldsymbol{x}_1$. In the second optimization problem instead ($p = 2$), the handle of the tool is the obstacle $\mathcal{OS}_1$ and the head is the target $\mathcal{T}_2$ (Fig. 8.10(c)). The solution $\boldsymbol{x}_2$ is therefore located on the head of the tool (Fig. 8.10(d)).

8.2.2 Best pose selection with multi-superquadric models

The best grasping pose is selected among the S candidates $\boldsymbol{x}_p$ for $p = 1, \ldots, S$ as the pose $\boldsymbol{x}_p*$ that minimizes the cost $\bar{\mathcal{C}}_p$:

$$\boldsymbol{x}_p* = \arg\min_p \bar{\mathcal{C}}_p, \tag{8.6}$$

with

$$\bar{\mathcal{C}}_p = w_1 F_{p,\boldsymbol{x}_p} + w_2\, E_{pos,p} + w_3 E_{o,p} + w4\, \frac{S-1}{\sum_k F_{k,\boldsymbol{x}_p}}. \tag{8.7}$$

The cost $\bar{C}_p$ of Eq. (8.7) is a generalization of the cost $\mathcal{C}$ proposed in Eq. (7.12) for selecting the best hand and can be expressed as:

$$\bar{\mathcal{C}}_p = \mathcal{C} + w4\, \frac{S-1}{\sum_k F_{k,\boldsymbol{x}_p}}, \tag{8.8}$$

where, for the sake of clarity, we remove the subscript *hand* from the cost $\mathcal{C}$ (originally defined as $\mathcal{C}_{hand}$) since thus far we just want to select the best pose among the candidates computed for the *same hand*. After, we explain how to use this cost to select the *best hand* for grasping an object modeled with multiple superquadrics.

The new cost $\bar{\mathcal{C}}_p$ takes into account:

- The cost function of Eq. (8.2) evaluated in the solution $\boldsymbol{x}_p$,

$$F_{p,\boldsymbol{x}_p} = \sum_{i=1}^{L} \left(\sqrt{\lambda_{1,p}\lambda_{2,p}\lambda_{3,p}} \left(F_p(\boldsymbol{p}_i^{\boldsymbol{x}}, \boldsymbol{\lambda_p}) - 1 \right) \right)^2 \Big|_{\boldsymbol{x}=\boldsymbol{x_p}}. \tag{8.9}$$

  The higher $F_{p,\boldsymbol{x}_p}$, the worse the overlapping between the hand ellipsoid $\mathcal{H}$ and the target superquadric $\mathcal{T}_p$ and, thus, the pose are.

- $E_{pos,p}$, the accuracy in reaching the desired position $\boldsymbol{pos}_{p,d} = [x_h, y_h, z_h]_p$ (see Section 7.4 for more details).

- $E_{o,p}$, the accuracy in reaching the desired orientation $\boldsymbol{o}_{d,p} = [\theta_h, \phi_h, \psi_h]_p$, as explained in Section 7.4.

- The average distance between the $S-1$ obstacle superquadrics $\mathcal{OS}_k$, for $k = 1, \ldots S$ and $k \neq p$ and the robot hand in the pose $\boldsymbol{x}_p$, defined

as follow:

$$\frac{1}{S-1}\sum_{k} F_{k,\boldsymbol{x}_p} \text{ with } k = 1,\dots,S \text{ and } k \neq p, \tag{8.10}$$

where

$$F_{k,\boldsymbol{x}_p} = \sum_{i=1}^{5} \left(\sqrt{\lambda_{1,k}\lambda_{2,k}\lambda_{3,k}}\,(F_k(\boldsymbol{v}_i^{\boldsymbol{x}}, \boldsymbol{\lambda_k}) - 1) \right) \Big|_{\boldsymbol{x}=\boldsymbol{x_p}}. \tag{8.11}$$

The larger the value of Eq. (8.11), the further the robot hand and fingers with pose $\boldsymbol{x}_p$ are from the obstacle superquadrics and, therefore, the better $\boldsymbol{x}_p$ is. Consequently, in the cost expression $\bar{\mathcal{C}}_p$ of Eq. (8.6) we use the reciprocal of (8.10). This way, for larger values of Eq. (8.11), we obtain lower costs $\bar{\mathcal{C}}_p$.

This term is the core part of the cost $\bar{\mathcal{C}}_p$. It is in fact fundamental to select the best grasping pose $\boldsymbol{x}_p*$ such that there are no collisions between the robot hand and other portions of the object, represented by the obstacle superquadrics.

The weights $w_1, w_2, w_3, w_4 > 0$ are then properly chosen in order to make the cost terms comparable among each other.

The optimization problems proposed in Eq. (8.2) can be used for computing grasping poses both for the right and left hand for a total of $2S$ candidates. The best grasping pose $\boldsymbol{x}_{p,hand}*$ can be then selected as:

$$\boldsymbol{x}_{p,hand}* = \arg\min_{hand} \min_{p} \bar{\mathcal{C}}_{p,hand}. \tag{8.12}$$

with

$$\bar{\mathcal{C}}_{p,hand} = w_1 F_{p,\boldsymbol{x}_{p,hand}} + w_2\, E_{pos,p,hand} + w_3 E_{o,p,hand} + w4\,\frac{S-1}{\sum_k F_{k,\boldsymbol{x}_{p,hand}}}, \tag{8.13}$$

$$p = 1,\dots,S,$$

and

$$hand = \{right, left\}.$$

8.3 Evaluation

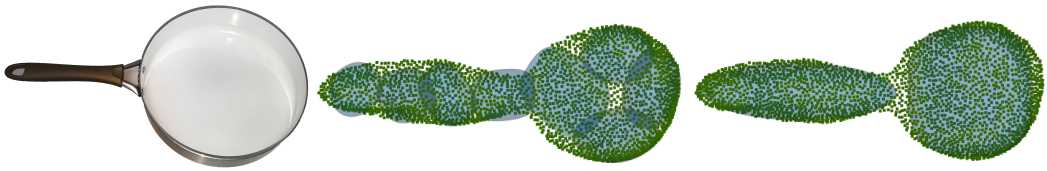
This Section reports the experiments we performed for evaluating the multi-superquadric modeling and grasping approaches proposed in this Chapter. The next two Paragraphs group respectively the results on the modeling process (Paragraph 8.3.1) and the grasping pose computation (Paragraph 8.3.2).

8.3.1 Multi-superquadric models

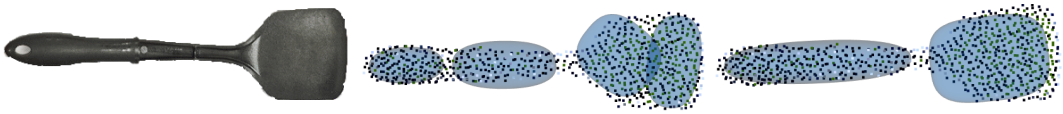
The multi-superquadric modeling has been tested on 23 objects also including objects that cannot be grasped by the iCub robot due to their dimensions or weight. The goal of this analysis is in fact to check the effectiveness of the modeling technique regardless of the grasping application. Some objects point cloud are noiseless and have been downloaded from the YCB [208] and ShapeNet [223] datasets. Other point clouds have been instead acquired with the stereo vision system of the iCub and are, therefore, quite noisy.

Figs. 8.11 - 8.15 collect the outcomes of the multi-superquadric modeling, in particular showing the leaves of the superquadric-tree $\mathcal{ST}$ - i.e. all the superquadrics corresponding to nodes with height H - and the final $\mathcal{S}$ superquadrics. In Table 8.1, we report the times required for the computation of the final model and of each intermediate step (i.e. Algorithms 4 - 5 - 6) with tree height $H = 3$. The modeling times are higher than the one obtainable with the single-superquadric modeling process (Table 7.1), since the new method entails the computation of a large number of superquadrics. In particular, Table 8.1 points out how actually Algorithm 4 is the most expensive step of the modeling process from a computational viewpoint. In fact, especially if no new superquadrics need to be estimated in Algorithm 6, the total computation time coincides with the time required for executing Algorithm 4.

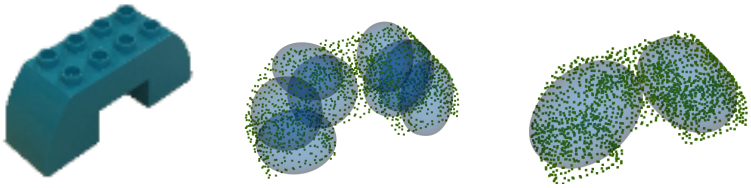
The multi-superquadric modeling algorithm relies on some parameters, such as the tree height H of the superquadric-tree $\mathcal{ST}$ (Paragraph 8.1.1) and the threshold μ used in the *differentDimensions*$(\cdot,\cdot)$ function (Paragraph 8.1.2). The tree height H is indirectly set by the user via the specification of the maximum number of superquadrics desired for the object model $\mathcal{S}_{max}$. The tree height is then obtained as $H = \lfloor \log_2 \mathcal{S}_{max} \rfloor$. The



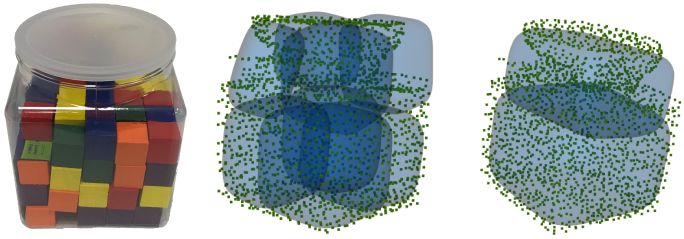
(a) Tool 1, $\mu = 0.03$ [m].



(b) Tool 2, $\mu = 0.03$ [m].



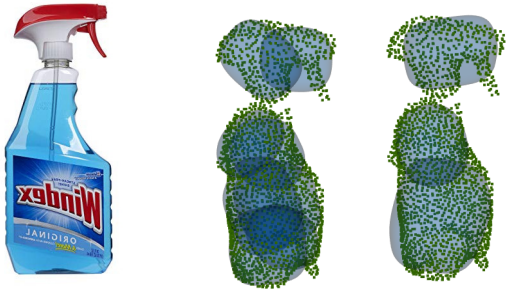
(c) Lego brick, $\mu = 0.03$ [m].



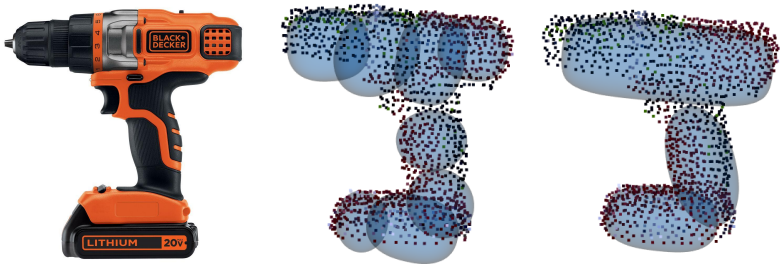
(d) Big box, $\mu = 0.03$ [m].

(e) Soap bottle, $\mu = 0.03$ [m].

FIGURE 8.11: The superquadric leaves of the superquadric-tree $\mathcal{ST}$ (in the center) and the final multi-superquadric model (on the right) obtained for noiseless point clouds from YCB dataset with height $H = 3$.



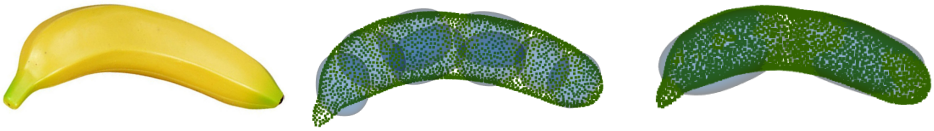
(a) Spray cleaner, $\mu = 0.015$ [m].



(b) Drill, $\mu = 0.015$ [m].

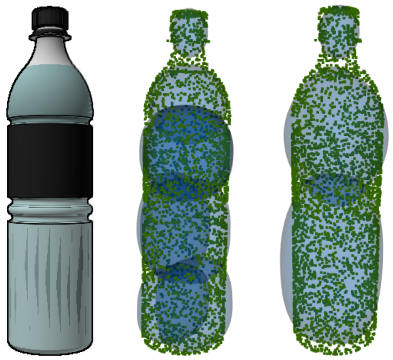
(c) Mustard bottle, $\mu = 0.015$ [m].

(d) Hammer , $\mu = 0.015$ [m].

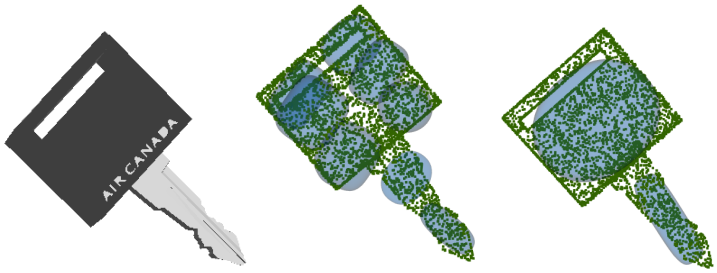


(e) Banana , $\mu = 0.015$ [m].

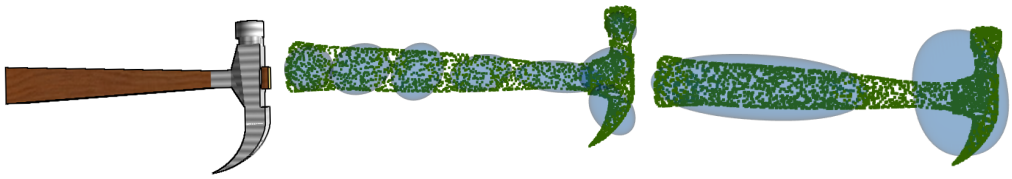
FIGURE 8.12: The superquadric leaves of the superquadric-tree $\mathcal{ST}$ (in the center) and the final multi-superquadric model (on the right) for noiseless point clouds from YCB datasets with height $H = 3$.



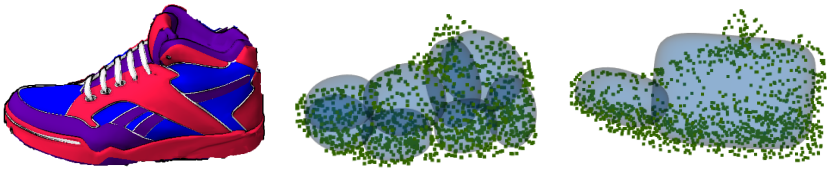
(a) Bottle , $\mu = 0.015$ [m].



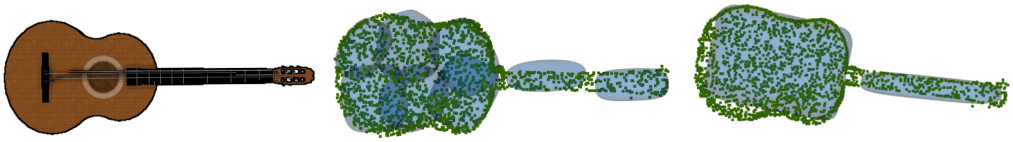
(b) Key, $\mu = 0.03$ [m].



(c) Hammer 2, $\mu = 0.015$ [m].



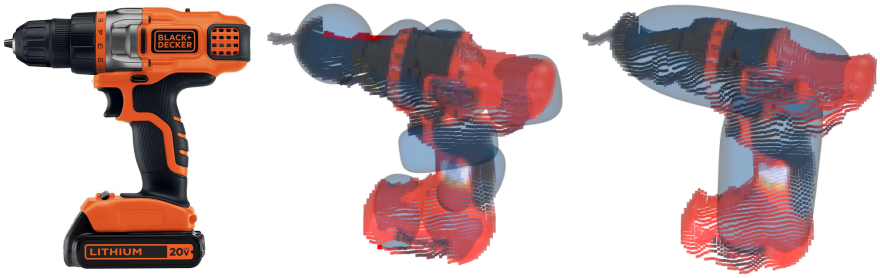
(d) Shoe, $\mu = 0.015$ [m].



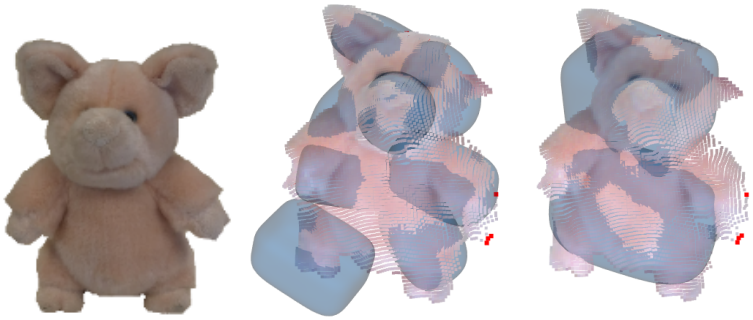
(e) Guitar, $\mu = 0.015$ [m].

FIGURE 8.13: The superquadric leaves of the superquadric-tree $\mathcal{ST}$ (in the center) and the final multi-superquadric model (on the right) for noiseless point clouds from ShapeNet and YCB datasets with height $H = 3$.

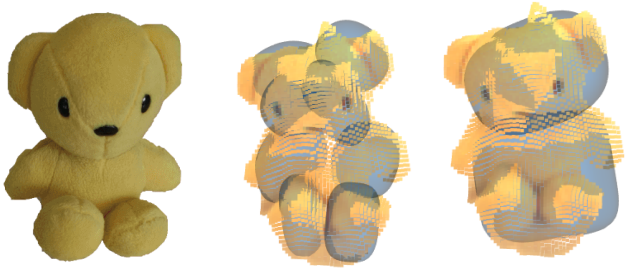
(a) Mustard bottle (noisy), $\mu = 0.03$ [m].



(b) Drill (noisy), $\mu = 0.03$ [m].

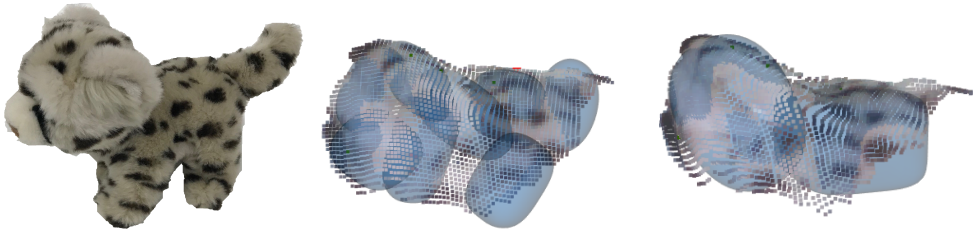


(c) Pig plush, $\mu = 0.03$ [m].

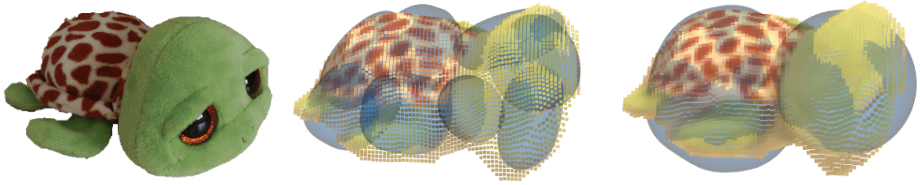


(d) Teddy bear, $\mu = 0.03$ [m].

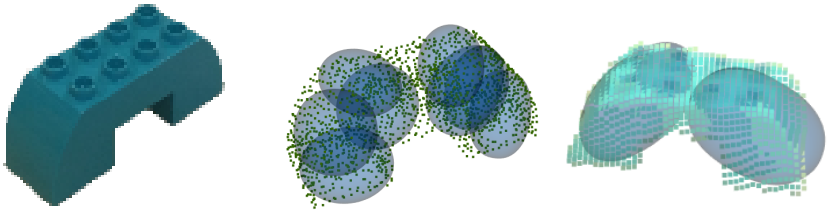
FIGURE 8.14: The superquadric leaves of the superquadric-tree $\mathcal{ST}$ (in the center) and the final multi-superquadric model (on the right) with height $H = 3$ for noisy point clouds, i.e. acquired from the iCub stereo system.



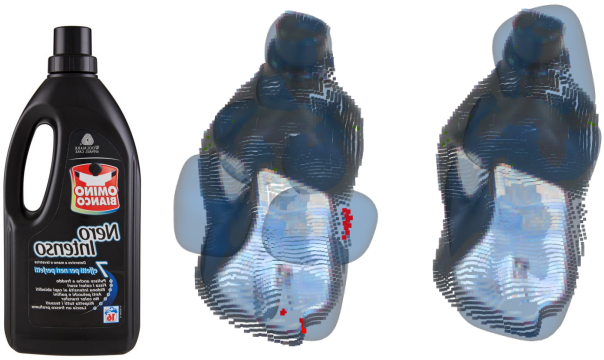
(a) Tiger plush, $\mu = 0.03$ [m].



(b) Turtle plush, $\mu = 0.03$ [m].



(c) Lego brick (noisy), $\mu = 0.03$ [m].



(d) Soap bottle (noisy), $\mu = 0.03$ [m].

FIGURE 8.15: The superquadric leaves of the superquadric-tree $\mathcal{ST}$ (in the center) and the final multi-superquadric model (on the right) with height $H = 3$ for noisy point clouds, i.e. acquired from the iCub stereo system.

TABLE 8.1: Execution time for multi-superquadric modeling.

Object	Total time [s]	Alg. 4 [s]	Alg. 5 [s]	Alg. 6 [s]
Tool 1	1.04	1.03	$8 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Tool 2	0.51	0.50	$4 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Lego brick	0.86	0.85	$7 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Big box	1.59	1.58	$8 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Soap bottle	1.23	1.22	$8 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Spray cleaner	1.24	1.23	$1 \cdot 10^{-3}$	$6 \cdot 10^{-5}$
Drill	1.10	1.09	$1 \cdot 10^{-3}$	$6 \cdot 10^{-5}$
Mustard bottle	1.28	1.28	$9 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Hammer	1.15	1.14	$1 \cdot 10^{-3}$	0.27
Banana	0.96	0.95	$9 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Bottle	1.48	1.47	$1 \cdot 10^{-3}$	0.17
Key	1.29	1.28	$1 \cdot 10^{-3}$	0.18
Hammer 2	0.63	0.62	$1 \cdot 10^{-3}$	$3 \cdot 10^{-5}$
Shoe	1.08	1.07	$1 \cdot 10^{-3}$	0.11
Guitar	1.43	1.42	$1 \cdot 10^{-3}$	0.23
Mustard b. (noisy)	1.18	1.17	$8 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Drill (noisy)	1.28	1.27	$1 \cdot 10^{-3}$	$3 \cdot 10^{-5}$
Pig plush	1.21	1.20	$7 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Teddy bear	1.17	1.17	$7 \cdot 10^{-3}$	$1 \cdot 10^{-4}$
Tiger plush	1.12	1.11	$9 \cdot 10^{-4}$	$3 \cdot 10^{-5}$
Turtle plush	1.07	1.06	$1 \cdot 10^{-3}$	$4 \cdot 10^{-5}$
Lego brick (noisy)	0.48	0.47	$1 \cdot 10^{-3}$	$2 \cdot 10^{-3}$
Soap bottle (noisy)	1.38	1.38	$1 \cdot 10^{-3}$	0.33

Table 8.1, we report the time for the computation of the final model and of each intermediate step of the multi-superquadric modeling, with tree height $H = 3$.

value H is however computed by taking into account a lower bound for the maximum number of superquadrics $\mathcal{S}_{max}$ in order to guarantee that a significant number of points is contained in each point cloud regions used in the superquadric fitting. The larger H is and the finer the object model can potentially be. Nonetheless, the threshold μ turns to be the parameter mostly responsible of the final accuracy of the reconstructed model. This threshold is used to discriminate whether superquadrics have similar dimensions and, thus, if the plane that led to their generation is relevant or not for splitting the point cloud. Using smaller thresholds leads to finer

models made of a larger number of superquadrics. Fig. 8.16 shows the effect of varying the threshold μ. In general, we noticed that noiseless point clouds of objects featured by small details require a smaller threshold in order to obtain fine models. Instead, when the point cloud is noisy, larger thresholds are necessary to remove unnecessary superquadrics from the final model. In Section 8.4 we discuss a possible approach to automatically infer a correct threshold μ.

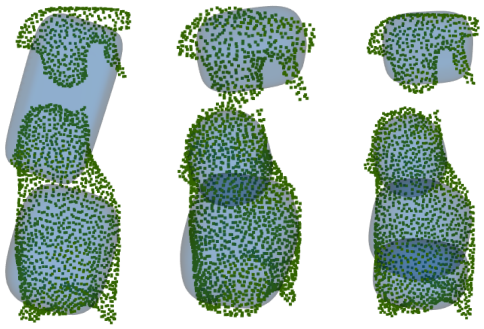
8.3.2 Multi-superquadric grasping poses

Grasping pose computation with multi-superquadric object models has been tested on 16 objects, selected among the 23 used for the modeling evaluation by discarding those objects too big with respect to the iCub hand. Even if all the objects under consideration have proper dimensions for being grasped by the iCub, they might be too heavy for real tests. However, this first simulative analysis aims at evaluating the effectiveness of the proposed method on a larger set of objects. In the next future, we are planning to perform experiments on the iCub only using objects actually graspable by the robot.

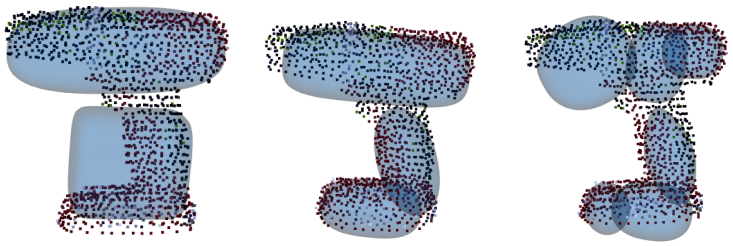
Figs. 8.17 - 8.18 - 8.19 collect the object models and grasping poses computed with the proposed approach. For each object, the S grasp candidates, their costs $\bar{\mathcal{C}}_p$ and the best selected pose $\boldsymbol{x}_{p*}$ (highlighted in green) are shown[6]. Table 8.2 reports the times for obtaining a single candidate and for the entire grasping pose computation. The total time is nearly S times the time required for obtaining a single grasp candidate. The total time could be improved by computing the S grasp candidates in parallel.

A key point of the grasping pose computation presented in this Chapter is the introduction in the optimization problem of suitable constraints for taking into account all the superquadrics of the object model. We recall that the solution $\boldsymbol{x}_p$ of the p-th optimization problem is computed by imposing the overlapping between the hand ellipsoid $\mathcal{H}$ and the target superquadric $\mathcal{T}_p$ and the avoidance of the other superquadrics $\mathcal{OS}_k$ for $k = 1, \ldots, S$ and

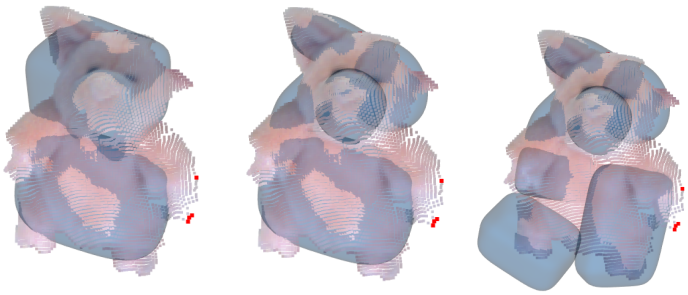
[6] For the sake of clarity of representation, the pose candidates have been computed only for one hand, according whether the object is better graspable with the left or right hand.



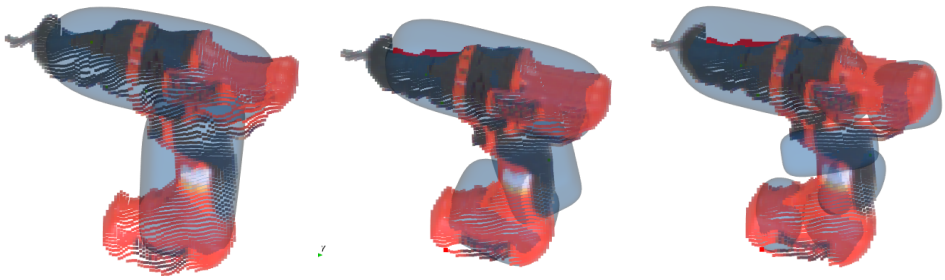
(a) $\mu = 0.02$ [m]. (b) $\mu = 0.015$ [m]. (c) $\mu = 0.01$ [m].



(d) $\mu = 0.02$ [m]. (e) $\mu = 0.015$ [m]. (f) $\mu = 0.005$ [m].

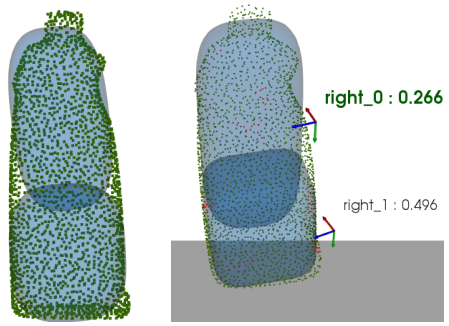


(g) $\mu = 0.03$ [m]. (h) $\mu = 0.02$ [m]. (i) $\mu = 0.01$ [m].

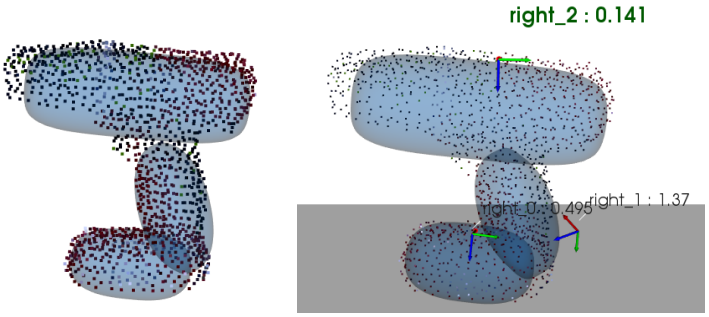


(j) $\mu = 0.03$ [m]. (k) $\mu = 0.02$ [m]. (l) $\mu = 0.01$ [m].

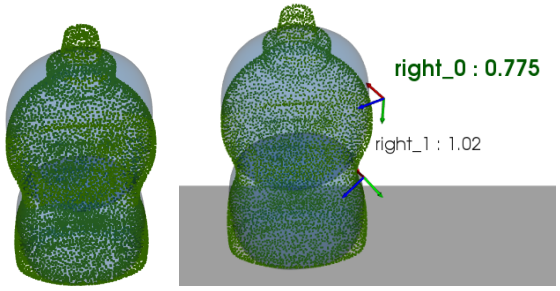
FIGURE 8.16: Examples of models obtained with different μ values. In each example the superquadric-tree is computed with height H=3 and the threshold μ decreases from left to right.



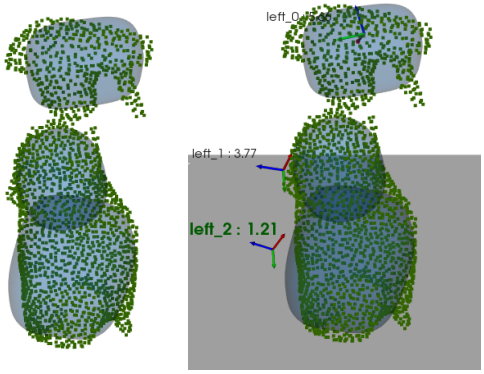
(a) Soap bottle.



(b) Drill.

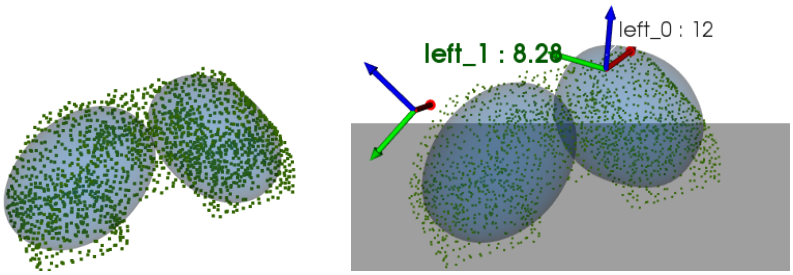


(c) Mustard bottle.

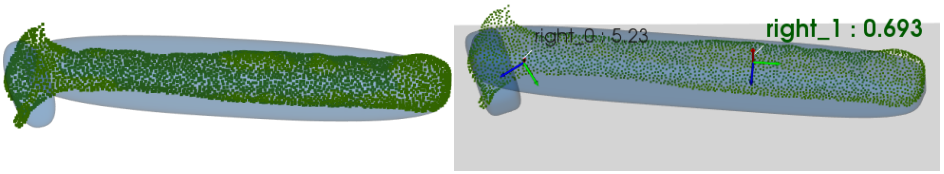


(d) Spray cleaner.

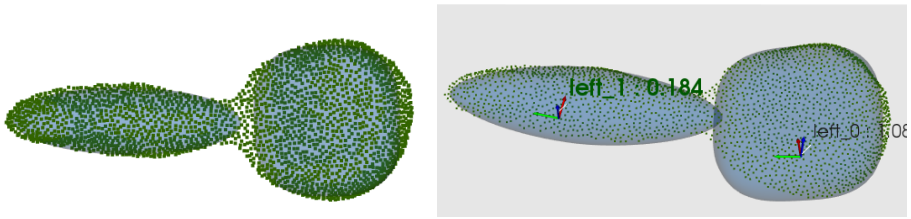
FIGURE 8.17: Multi-superquadric models and grasping candidates computed with noiseless point clouds of the YCB datasets.



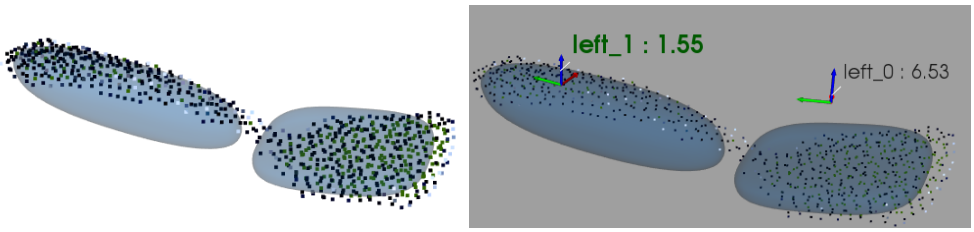
(a) Lego brick.



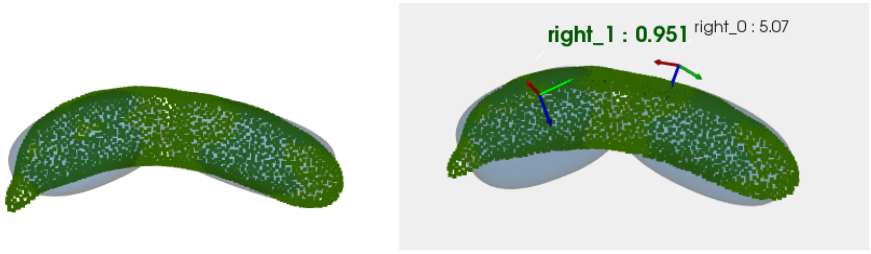
(b) Hammer.



(c) Tool 1.

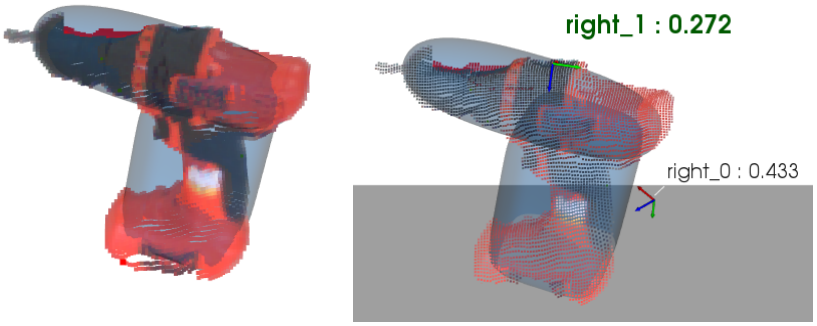


(d) Tool 2.

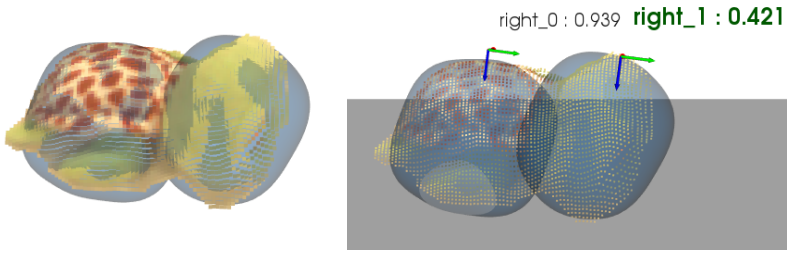


(e) Banana.

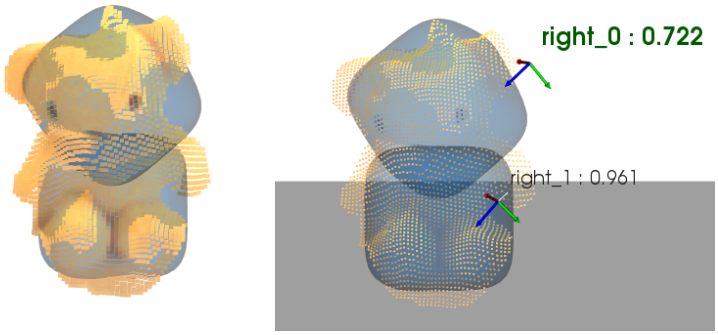
FIGURE 8.18: Multi-superquadric models and grasping candidates computed with noiseless point clouds of the YCB datasets.



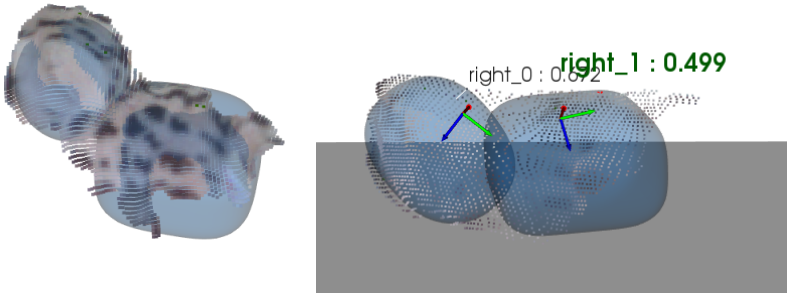
(a) Drill (noisy).



(b) Turtle plush.

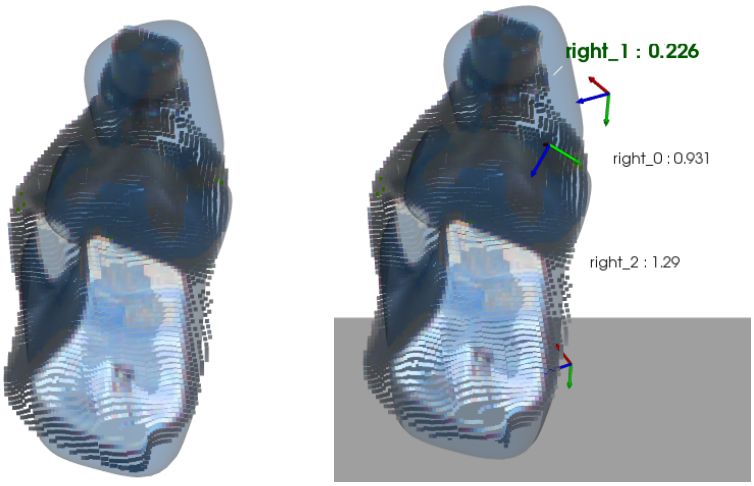


(c) Teddy bear.

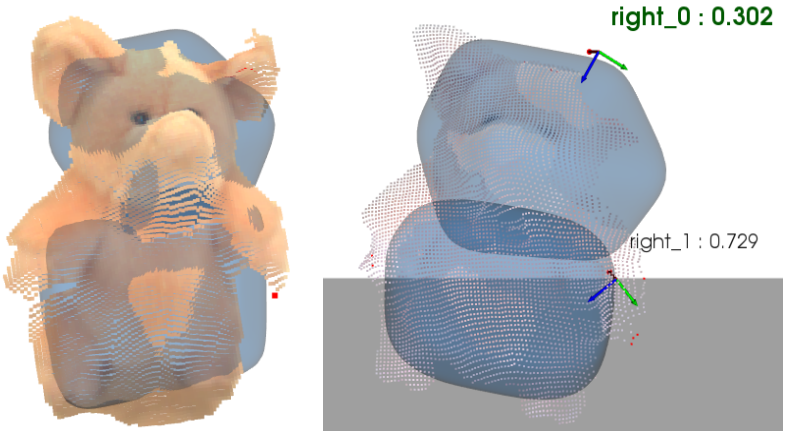


(d) Tiger plush.

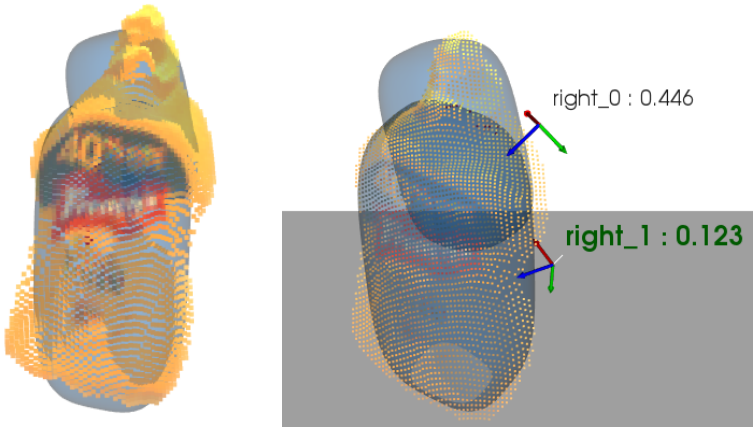
FIGURE 8.19: Multi-superquadric models and grasping candidates computed with noisy point clouds.



(a) Soap bottle (noisy).



(b) Pig plush.



(c) Mustart bottle noisy.

FIGURE 8.20: Multi-superquadric models and grasping candidates computed with noisy point clouds.

TABLE 8.2: Execution time for grasping pose computation.

Object	Single comput. time [s]	Total comput. time [s]
Soap bottle	0.25	0.51
Drill	0.27	0.97
Mustard bottle	0.03	0.06
Spray cleaner	0.08	0.28
Lego brick	0.25	0.60
Hammer	0.13	0.25
Tool 1	0.26	0.65
Tool 2	0.15	0.33
Banana	0.18	0.40
Drill (noisy)	0.12	0.22
Turtle plush	0.11	0.25
Teddy bear	0.13	0.29
Tiger plush	0.06	0.15
Soap bottle (noisy)	0.15	0.49
Pig plush	0.20	0.42
Mustard b. (noisy)	0.17	0.36

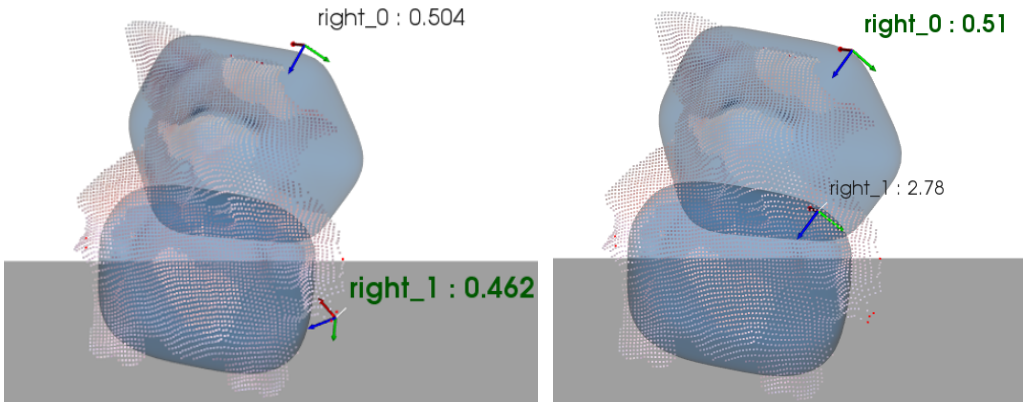
Table 8.2 indicates the time for computing a single grasp candidate $\boldsymbol{x}_p$ (central column) and the final solution $\boldsymbol{x}_p*$.

$k \neq p$ (obstacle superquadrics) with the robot palm and fingers. To support this choice, in Fig. 8.21, we report an example comparison between:

- The grasp candidates computed by solving the S optimization problems of (8.2) (Fig 8.21(a));
- The poses obtained by simply applying the single-superquadric grasping pose computation of Eq. (7.4) on each superquadric $\mathcal{S}_p$ of the model and thus ignoring the presence of the other $S-1$ superquadrics during the computation (Fig 8.21(b)).

Ignoring the presence of the other $S-1$ superquadrics does not guarantee the grasp candidates to be located outside the obstacle superquadrics and indeed they might lay inside portions of the object, as happens in the example shown in Fig 8.21(b).

Taking into account all the superquadrics belonging to the object model is crucial also during the selection of the final pose $\boldsymbol{x}_p*$. Fig. 8.22 shows an example of outputs of the best pose selection (highlighted in green) when using:

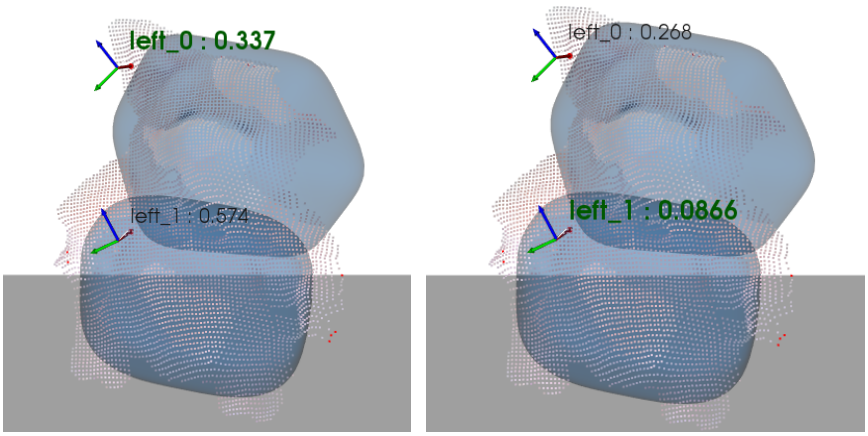


(a) (b)

FIGURE 8.21: Comparison between grasping candidates computed: 1) by solving the optimization problems of (8.2) and, therefore, imposing the avoidance of the obstacle superquadrics (Fig 8.21(a)) and 2) by applying the single-superquadric grasping pose computation of Eq. (7.4) on each superquadric $\mathcal{S}_p$ of the model (Fig 8.21(b)). In case 2) , we ignore the presence of the other $S-1$ superquadrics during the computation.

- the cost $\bar{\mathcal{C}}$ defined in Eq. (8.7), that includes the average distance between the hand palm and fingers and the obstacle superquadrics (Fig 8.22(a));
- and the cost $\mathcal{C} = \bar{\mathcal{C}} - w4\ \frac{S-1}{\sum_k F_{k,x_p}}$, that ignores any spatial relationships between the robot hand and the obstacle superquadrics (Fig 8.22(b)).

The best poses selected using $\mathcal{C}$ are often worse since they might be located quite close to the obstacle superquadrics, as shown in Fig 8.22(b) where the pose on the bottom superquadric is selected. The solutions of the optimization problems of (8.2) are necessary located *outside* the obstacle superquadrics - thanks to the constraints - but they still might be quite close to their surfaces (this might happen due to the presence of the other constraints of the optimization problem, such as the avoidance of the support on which the object is located, as in Fig 8.22(b)). In practical grasping applications, it is better to avoid such a poses since they are more likely to lead to a collision between the robot hand and the object while reaching for the desired pose. If the average distance with respect to the obstacle superquadrics is not taken into account there is no guarantee that those poses are low-ranked during the final



(a) (b)

FIGURE 8.22: Comparison between pose candidates cost when cost $\bar{\mathcal{C}}$ (on the left) and $\mathcal{C}$ (on the right) are used. When cost $\mathcal{C}$ is used grasp candidates close to the surface of obstacle superquadrics might be selected as best pose if they provide good overlapping and reachability, as happens in the case on the right where the grasping pose computed for the bottom superquadric is selected (highlighted in gree). However, these poses are likely to lead to collision between the hand and the object. Using instead cost $\bar{\mathcal{C}}$ for selecting poses more distance from the obstacle superquadrics leads to safer poses, such as the one selected in the left image, which is located on the top superquadric.

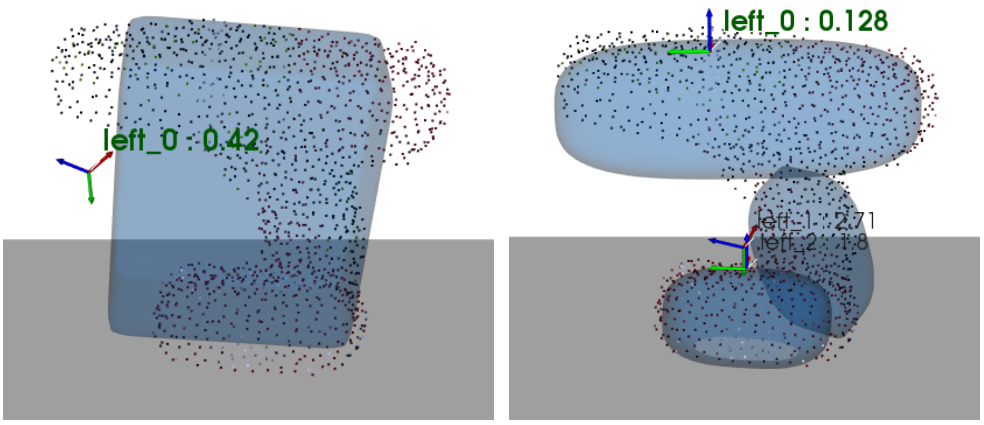
selection, since they could anyway provide a good overlapping of the hand ellipsoid $\mathcal{H}$ and target superquadric $\mathcal{T}_p$ and be reachable by the robot.

It is also interesting to analyze the benefits that the multi-superquadric grasping approach of this Chapter brings with respect to the method based on single-superquadric models described in Chapter 7. Figs. 8.23 and 8.24 collect and compare the grasping poses computed when using the *single-* and *multiple-* superquadric approaches. When the objects are poorly represented with a single superquadric (Fig 8.23), the grasping pose computed using such a model is likely to be completely wrong. Some other objects instead (Fig 8.24), even if better represented with multiple superquadrics, can be approximated also with a single superquadric. In these cases, the grasping poses computed with the multi-superquadric approach better fit the object surface but those computed using single superquadric models can be effective for the grasping task as well. This depends mostly on the object softness and texture: plushes for example can also be grasped using poses computed with a single superquadric model because they can deform under fingers pressure. However, using finer object models for computing grasp candidates also for this kind of objects allows for a better control on the object portion on which the hand should be located.

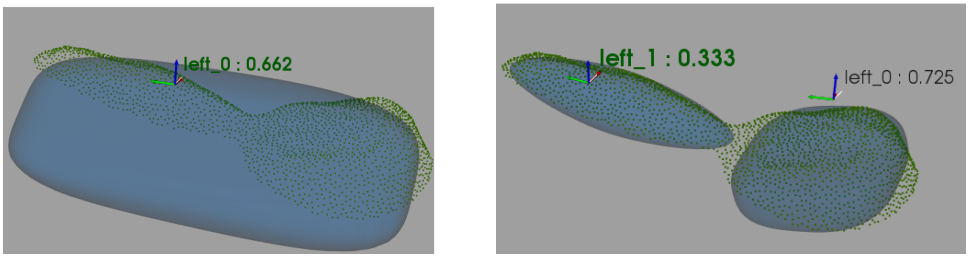
8.4 Discussion

In this Chapter, we presented an extension of the single superquadric modeling and grasping techniques able to deal with more complex objects. The main limitation of the approach described in Chapter 7 is the assumption that the object can be properly represented with a single superquadric, hypothesis that unfortunately does not hold for several everyday objects. This encouraged us to design a new algorithm capable of generating finer object models, made of multiple superquadrics.

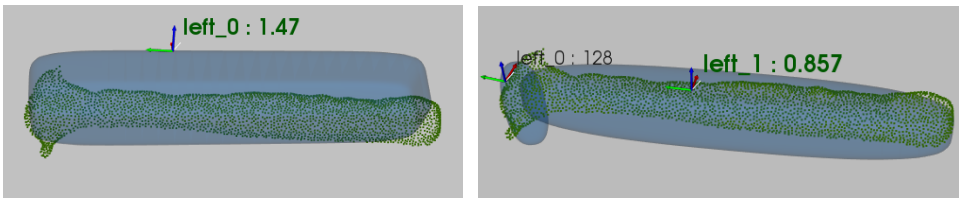
The modeling method proposed in this Chapter is able to infer automatically the minimum number and the parameters of the superquadrics required for modeling the object with higher accuracy than using a single superquadric. This is achieved by iteratively splitting the object point cloud with simple geometrical criteria, fitting a superquadric for each portion and building the so-called *superquadric-tree* $\mathcal{ST}$ (Paragraph



(a) (b)

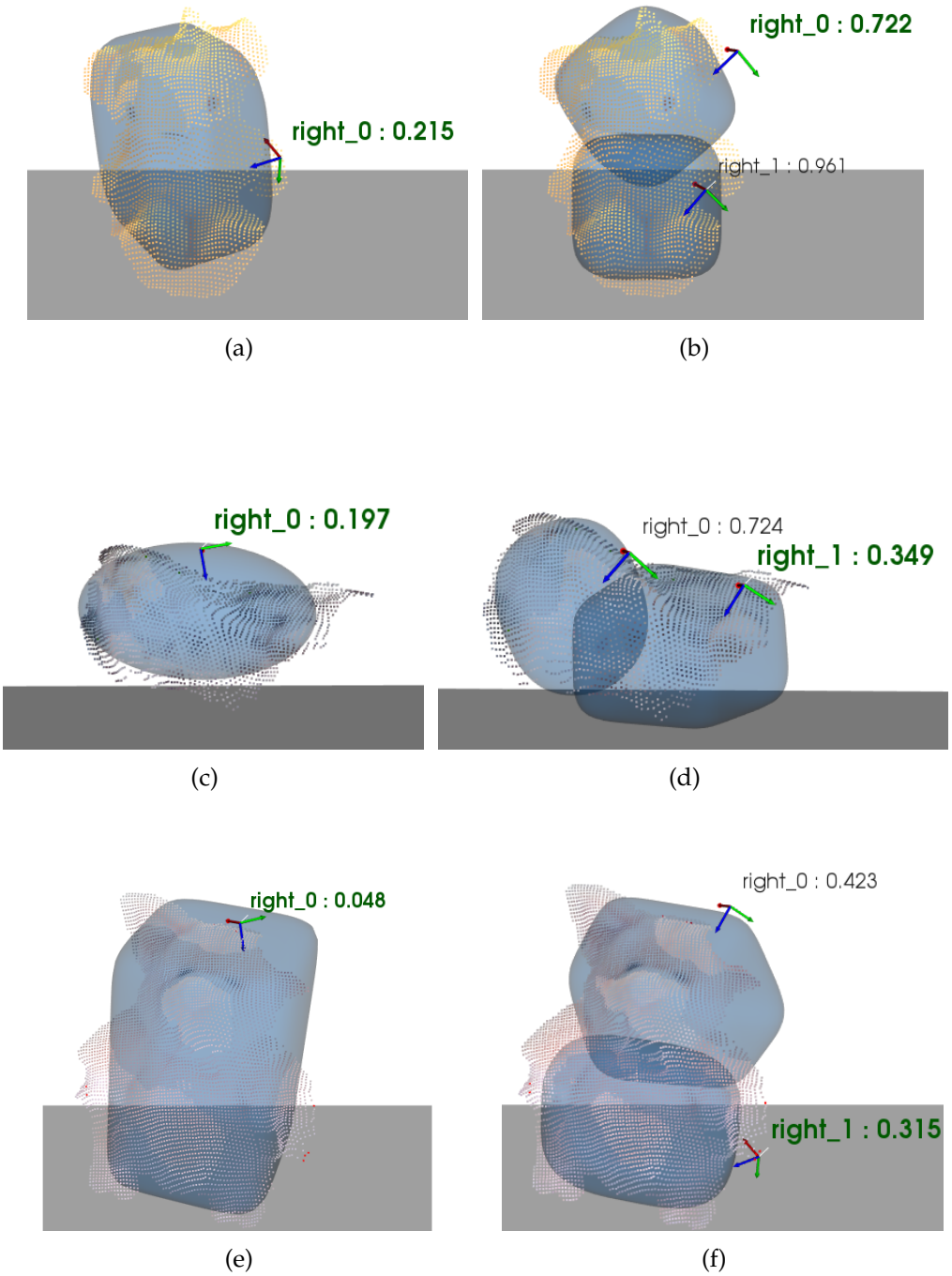


(c) (d)



(e) (f)

FIGURE 8.23: Comparison between grasping poses computed by modeling tools with single- and multiple-superquadric models.



(a) (b) (c) (d) (e) (f)

FIGURE 8.24: Comparison between grasping poses computed by modeling soft objects with single- and multiple-superquadric models.

8.1.1). The structure of the superquadric-tree is then used to infer whether the portions of the point cloud need to be represented with different superquadrics or can be merged together.

Once a multi-superquadric model is provided, a grasp candidate for each superquadric is computed and the best one for performing the task is selected. Both the multi-superquadric grasp poses computation and selection extend the approaches described in Section 7.2 and 7.4 in order to deal with multi-superquadric models.

The problem of modeling an object with multiple superquadrics has been addressed in some previous works [105, 106, 112, 113]. In particular, in [105] the authors proposed a *splitting and merging approach*, which has been exploited, with slight modifications, also in [106, 112, 113]. The basic idea of the algorithm consists of proceeding at first top-down by splitting the point cloud and fitting a superquadric for each resulting portion. During the merging step, the algorithm merges each superquadric with its neighbors and the merged superquadric that provides the best improvement in representing that object portion is selected. This step is repeated going bottom-up until a final model accuracy is reached. The algorithm presented in [112] implements the splitting and merging procedure of [105] using a binary tree to limit the possible merging combinations. In [106] instead the number of neighbors taken into account during each merging trial are reduced using k-nearest neighbor algorithms. Even though every work presents some different features, they all rely on checking if each merging step produces an improvement in the model. This kind of condition is hard to verify, as it requires the definition of a reconstruction error with respect to the point cloud, independent from the superquadric size and point cloud noise. This is the reason why our approach deviates from the standard merging approach proposed in [105] and aims instead at finding the splitting planes for cutting the object point cloud.

To the best of our knowledge, this is the first work which proposes an approach for computing grasp candidates using multiple superquadrics. The works using multiple superquadrics for grasping the object make use of grasp candidates generator such as GraspIt! [112]. Other works that instead compute grasping candidates by exploiting superquadric properties focus only on single superquadric models [115, 116].

We tested our approach on a large number of objects, including also

objects from the YCB and ShapeNet datasets. Even if exhaustive experiments on the real robot to test the effectiveness of the models and grasping poses are still missing, the methods provide promising results, as shown in Section 8.4.

We are however aware of some of the limitations the method presented in this Chapter is affected by, especially regarding the modeling process. The final model accuracy in fact strongly depends on the setting of the threshold μ (Paragraph 8.1.2) used to discriminate whether a pair of superquadrics have similar dimensions and, consequently, can be merged in the final model. Even though inferring the proper μ value to obtain an accurate model for a specific object is fast and intuitive, it is instead very hard to find a unique value μ suitable for different kind of objects (Fig. 8.13) and point cloud qualities (Fig. 8.12 vs Figs. 8.14 and 8.15). However, it is worth pointing out how setting a quite small threshold μ leads to over-segmentation of the objects and thus to models still effective for the grasping task. A possible way to avoid a manual setting of the threshold is to rely on a classifier properly trained to predict the optimal value μ for different object classes, following a similar approach to the one proposed in Section 7.3.1 to improve and speed up the single superquadric estimation using prior on object shapes.

Another limitation of the modeling process, affecting also the grasp pose computation, is the computation time. Together with code improvements and parallelization, a considerable advancement in this respect could be provided by the adaption of learning framework. A CNN could learn single superquadrics to speed up their reconstruction and another network could be trained to identify where to split the object point cloud for making the multiple superquadric modeling more robust. At this regard, some works proposing 3D and 2D object portions segmentation for the affordance problem [224, 225] could be adapted to identify how to split object portions in volumetric and shape proprierties relevant for the more general grasping task.

Part V

Deep Reinforcement Learning for manipulation

Chapter 9

The exploration problem in Deep Reinforcement Learning and its relevance in manipulation

The research activity described thus far in this Thesis addresses the problem of autonomous manipulation by relying on model-based approaches where the problem is partitioned into simpler sub-problems. This is the standard way-to-go in robotics but it is actually very different from how humans learn to manipulate objects. Humans do not learn separately how to segment objects, reconstruct their models and compute poses for grasping them. Instead, the solution of each problem is achieved at the same time with experience, learning from trials and errors. For this reason, the interest in alternative approaches based on learning has been increasingly growing lately in robotic community, in particular in context of manipulation [4, 5, 6, 157, 158, 167][1]. However, the application of learning to robotics has been encouraged also by its potentials. First of all, such a framework can simplify robot actions planning by designing general purpose algorithms with proper cost functions. It also allows better exploiting the sensors and robots DOFs especially when not easy to be modeled. Finally, learning algorithms can potentially adapt constantly to the environment where the robot operates.

Deep Reinforcement Learning (RL) provides a mathematical formulation that well lends itself to model the properties of humans of learning from theirs mistakes and encodes the potentials listed above. This is the

[1] The interest in learning is much wider in robotics. Deep Learning is nowadays the state of the art in computer vision and Deep Reinforcement Learning is applied for solving other classic robotics problems, such as locomotion.

reason of my interest in Deep RL, with the aim to cast the model-based techniques described thus far in this Thesis into a more general learning framework in the next future.

This Chapter collects the activities I carried out at Berkeley Artificial Intelligence Research Lab (BAIR), UC Berkeley, under the supervision of Prof. Pieter Abbeel. During my internship I studied and practiced with Deep Reinforcement Learning with the high-level goal to understand its effectiveness and potential for improving robots manipulation skills.

The Chapter is organized as follows. Section 9.1 introduces the Reinforcement Learning setting[2] focusing on the exploration problem, which is the topic I worked on during my internship. Section 9.2 summarizes a project I was involved in which addresses explicitly robotic manipulation and solves the exploration problem by exploiting human demonstrations. Section 9.3 describes the main project I worked on during my internship: how to use information properly learned from the solution of prior tasks for encouraging exploration in a new task. Finally, Section 9.4 discusses the current limitations of these works and ideas for future work on these topics.

9.1 Problem formulation

Reinforcement Learning is well-suited for robotic applications as it is concerned with learning the actions an agent has to execute for performing a desired task. Unlike *supervised learning* where the labels of the training set provide a supervision to the learning algorithm, in the Reinforcement Learning framework the algorithms are provided only with a *reward function R*, which indicates to the learning agent when it is acting well, and when it is acting poorly. In grasping applications for instance, the reward function might give the robot positive rewards when moving the end-effector towards the object to grasp, and negative rewards for moving in the opposite direction. The reinforcement learning algorithm is supposed to figure out how to choose actions over time so as to obtain large rewards.

[2] In this Chapter we refer to Deep RL and RL interchangeably. To be precise, Deep RL is the combination of RL together with deep networks. Deep networks are extremely helpful when, for example, the algorithm is asked to learn from raw inputs, such as vision, without any hand-engineered features or domain heuristics.

Reinforcement Learning is usually posed in the formalism of *Markov Decision Processes (MDPs)*. A Markov Decision Process is a tuple:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}_{sa}, R, \gamma, \rho_0\}, \tag{9.1}$$

where:

- $\mathcal{S} \subseteq \mathbb{R}^n$ is the set of states the system can experience. For example, in manipulation tasks this could include all the possible end-effector and fingers poses together with the object position. In this Chapter, we assume the states to be known and available. The more general scenario where only observations are available is modeled with the *Partially Observed MDPs*, and, even if more realistic, is out of the scope of this Chapter.

- $\mathcal{A} \subseteq \mathbb{R}^m$ is the set of actions the agent can execute, e.g. the joint angles velocities of the robot arm and hand.

- $\mathcal{P}_{sa} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition dynamics, which can be stochastic, i.e. the distribution over what states the system transitions to if the action a is taken in state s. The transition dynamics can be known or not, leading to two different families of RL algorithms, respectively named *model-based* and *model-free* approaches. From now on we always assume this distribution not to be provided and, thus, it can be accessed only by sampling.

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which evaluates the effectiveness of action a taken in state s with respect to the task to accomplish.

- $\gamma \in [0, 1]$ is the so-called discount factor, that is required in infinite horizon tasks whereas it can be omitted if the task is executed in a finite number of steps[3].

- ρ_0 is the probability distribution over initial states.

The MDP is used to model systems where the execution of an action a_t at time t leads to the transition from the state s_t to the state s_{t+1} starting

[3] As shown in Eq. (9.2), in infinite-horizon tasks the discount factor < 1 is necessary to make the sum $\sum_{t=0}^{\infty} \gamma^t R_t$ finite. In case of finite horizon tasks instead, that sum is automatically finite: $\sum_{t=0}^{T} \gamma^t R_t$.

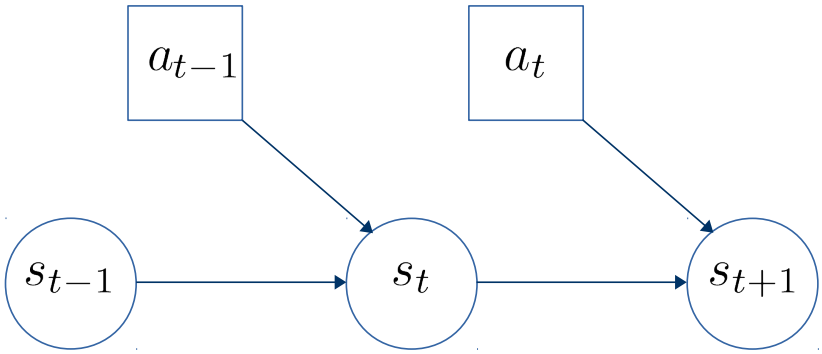


FIGURE 9.1: Markov Decision Process graphical model.

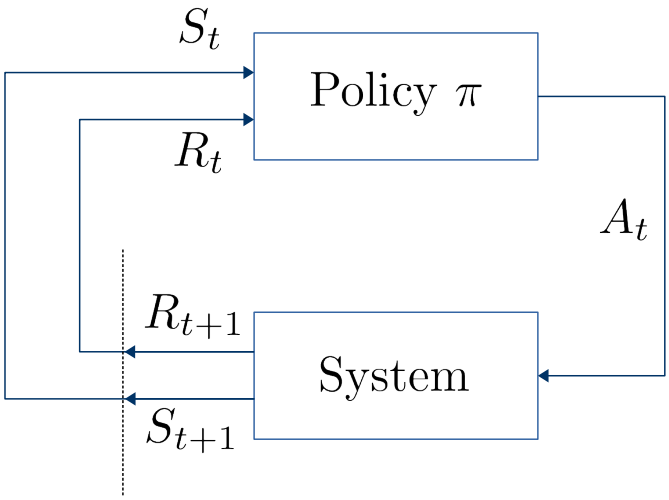


FIGURE 9.2: Reinforcement Learning scheme. The policy π performs an action A_t so as to maximize the reward R_t, being in state S_t. The execution of A_t makes the system transition to the state S_{t+1} and a new reward R_{t+1} is provided.

from an initial condition s_0 and according to the transition distribution $\mathcal{P}_{sa}$, i.e. $s_{t+1} \sim \mathcal{P}_{sa}$ (Fig. 9.1).

The goal of RL is to estimate a *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$, i.e. a mapping from the states to the actions (Fig. 9.2), so as to maximize the expected *return* during the execution of the task:

$$\eta(\pi) = E_{\pi,\mathcal{M}}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]. \tag{9.2}$$

For a complete formulation of the Reinforcement Learning setting we also define:

- The value function:

$$V_\pi(s) = E_{\pi,\mathcal{M}}\left[\sum_{t=0}^{\infty}\gamma^t R_t \mid s_0 = s\right] \tag{9.3}$$

  i.e. the expected sum of discounted rewards upon starting in state s, and taking actions according to π.

- The Q-function:

$$\begin{aligned} Q_\pi(s,a) =& E_{\pi,\mathcal{M}}\left[\sum_{t=0}^{\infty}\gamma^t R_t \mid s_0 = s, a_0 = a\right] = \\ & E_{\mathcal{M}}\Big[R(s,a)\Big] + E_{s'\sim\mathcal{P}_{sa}}\Big[V_\pi(s')\Big] \end{aligned} \tag{9.4}$$

  that provides the expected total reward of the agent when starting in state s and picking action a.

- The advantage function:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{9.5}$$

The three functions above introduced are fundamental for the design of the main RL algorithms and we will refer to them in the next Sections. A careful description of the most popular RL algorithm is however out of the scope of this Thesis work. We refer to [159] for a complete discussion of the topic.

The problem we focus on in this Chapter is the so-called *exploration problem* or, in other words, *how to efficiently explore the state space in order to fast experience the rewards*.

The effectiveness of RL algorithms strongly depends on the design of the reward functions. This is in fact the only information our agent receives regarding the quality of the actions it performs. Designing the reward function is not trivial in real world applications. While the reward in game playing tasks is constantly available during their execution being the game score, in practical applications the rewards that can be easily obtained are mostly indicator functions for task completion. In the grasping task for example, such a indicator can provide 1 when the robot is holding

the object and 0 otherwise. Unfortunately, this kind of rewards, also called *sparse task completion rewards*, provide poor information during the training of the policy. How can the policy learn the actions to perform the task, if it receives non-zero rewards only when the task is accomplished? For this reason, in practice, various forms of *shaped rewards* that incorporate human priors on how to accomplish the task might be required to make progress on the learning problem. Even if very helpful in many applications, reward shaping is often quite difficult also requiring to deal with a well structured environment. Shaping the rewards for a pick-and-place task for example is not trivial (as shown in Section 9.2). The rewards should take into account the distance between the hand palm, fingertips and the object and from the object itself to its final desired position. Even if this might look easy, it actually requires some tuning of parameters for weighting differently each reward term. In addition, we need to track the position of the object, that might be complicated due to possible vision occlusions during the execution of the task[4]. It also might happen that the shaped reward we manage to define actually depends only on a portion of the state. When the state dimensionality is large ($n >> 10$), instead of searching the solution by exploring the entire state, it would be more efficient to focus only on that portion of the state responsible for the rewards following proper exploration strategies. For example, if a robot is asked to move a specific object in a cluttered scenario towards a goal position, moving all the objects is not convenient. It would instead be more efficient to play only with the object of interest. The case when learning is performed from raw images is even more explanatory. In this case the state is very large and redundant: all the pixels representing the background are in fact very likely to provide useless information for the solution of the task.

In the next Sections, we describe two different strategies for addressing the problem of exploration in presence of sparse rewards and high-dimensional state space: using human demonstrations (Section 9.2) and learning the latent representation of the state that mostly affects the reward function from prior tasks (Section 9.3). The work presented in Section 9.2 particularly focuses on the exploration problem for dexterous

[4]It is important to clarify that this is actually a problem in case of POMDPs, i.e. when the entire state of the system is not available. As we said before, this scenario is out of the scope of our work, but it is actually what we encounter in real world applications. It is therefore relevant taking into account these issues during the motivation of our study.

manipulation tasks.

9.2 Learning complex dexterous manipulation with Deep Reinforcement Learning and demonstrations

Multi-fingered dexterous manipulators are crucial for robots to function in human-centric environments, due to their versatility and potential to enable a large variety of contact-rich tasks, such as in-hand manipulation, complex grasping, and tool use. However, this versatility comes at the price of high dimensional observation and action spaces, complex and discontinuous contact patterns, and under-actuation during non-prehensile manipulation. This makes dexterous manipulation with multi-fingered hands a challenging problem.

Dexterous manipulation behaviors with multi-fingered hands have previously been obtained using model-based trajectory optimization methods [226, 227]. However, these methods typically rely on accurate dynamics models and state estimates, which are often difficult to obtain for contact rich manipulation tasks, especially in the real world. Reinforcement learning provides a model agnostic approach that circumvents these issues. Indeed, model-free methods have been used for acquiring manipulation skills [228, 229], but thus far have been limited to simpler behaviors with 2-3 finger hands or whole-arm manipulators, which do not capture the challenges of high-dimensional multi-fingered hands.

We find that existing RL algorithms can indeed solve these dexterous manipulation tasks, but require significant manual effort in reward shaping. In addition, the sample complexity of these methods is very poor, thus making real world training infeasible. To overcome this challenge, we propose to augment the policy search process with a small number of human demonstrations collected in virtual reality (VR). In particular, we find that pre-training a policy with behavior cloning, and subsequent fine-tuning with policy gradient along with an augmented loss to stay close to the demonstrations, dramatically reduces the sample complexity, enabling training within the equivalent of a few real-world robot hours. Although success remains to be demonstrated on hardware, our results in this work

indicate that DRL methods when augmented with demonstrations are a viable option for real-world learning of dexterous manipulation skills.

Our main contributions are:

- We demonstrate, in simulation, dexterous manipulation with high-dimensional human-like five-finger hands using model-free DRL. To our knowledge, this is the first empirical result that demonstrates model-free learning of tasks of this complexity.
- We show that with a small number of human demonstrations, the sample complexity can be reduced dramatically and brought to levels which can be executed on physical systems.

In the next Paragraphs, we briefly report the main contributions and results of this work. An exhaustive explanation and evaluation of the contributions is provided in [180].

9.2.1 Dexterous manipulation tasks

The real world presents a plethora of interesting and important manipulation tasks. While solving individual tasks via custom manipulators in a controlled setting has led to success in industrial automation, this is less feasible in an unstructured settings like the home. Our goal is to pick a minimal task-set that captures the technical challenges representative of the real world. We present four classes of tasks - object relocation, in-hand manipulation, tool use, and manipulating environmental props (such as doors). Each class exhibits distinctive technical challenges, and represent a large fraction of tasks required for proliferation of robot assistance in daily activities – thus being potentially interesting to researchers at the intersection of robotics and machine learning. All our task environments expose hand (joint angles), object (position and orientation), and target (position and orientation) details as observations, expect desired position of hand joints as actions, and provides an oracle to evaluate success. We now describe the four classes in light of the technical challenges they present.

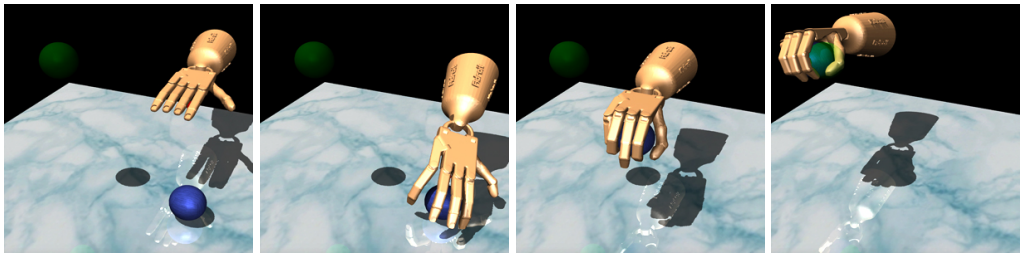


FIGURE 9.3: Object relocation – move the blue ball to the green target. Task is considered successful when the object is within epsilon-ball of the target.

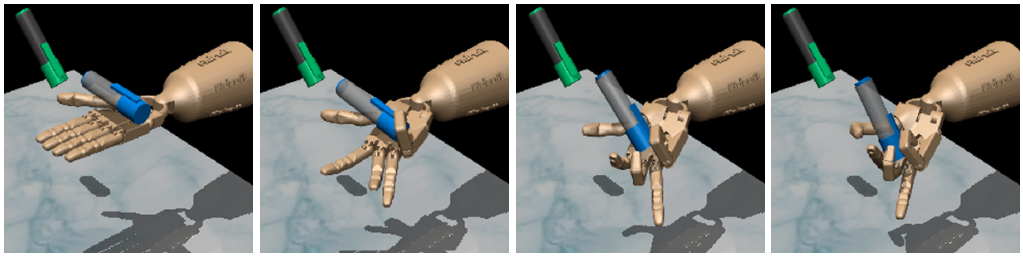


FIGURE 9.4: In-hand manipulation – reposition the blue pen to match the orientation of the green target. Task is considered successful when the orientations match within tolerance.

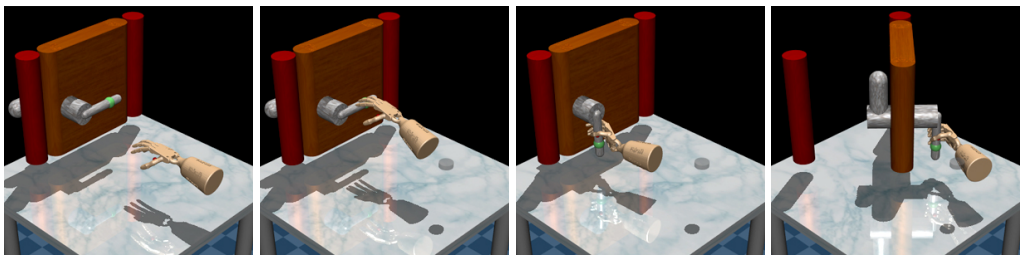


FIGURE 9.5: Door opening – undo the latch and swing the door open. Task is considered complete when the door touches the door stopper at the other end.

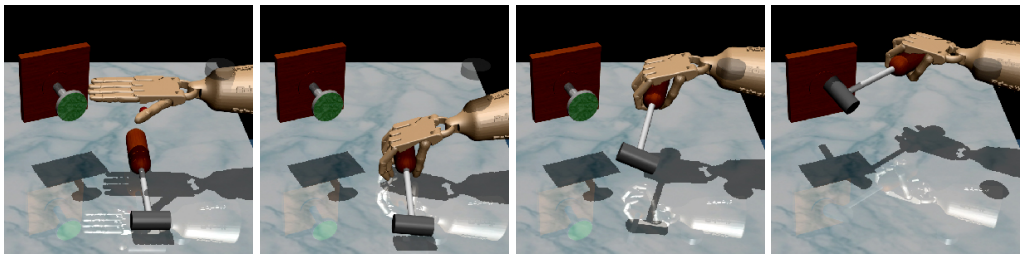


FIGURE 9.6: Tool use – pick up and hammer with significant force to drive the nail into the board. Task is successful when the entire length of the nail is inside the board.

Object relocation (Fig. 9.3)

Object relocation is a major class of problems in dexterous manipulation, where an object is picked up and moved to a target location. The principal challenge here from an RL perspective is exploration, since in order to achieve success, the hand has to reach the object, grasp it, and take it to the target position – a feat that is very hard to accomplish without priors in the form of shaped rewards or demonstrations.

In-hand Manipulation – Repositioning a pen (Fig. 9.4)

In hand-manipulation maneuvers like re-grasping and re-positioning objects involve leveraging the dexterity of a high DOF manipulator to effectively navigate a difficult landscape filled with constraints and discontinuities imposed by joint limits and frequently changing contacts. Due to the large number of contacts, conventional model-based approaches which rely on accurate estimates of gradients through the dynamics model struggle in these problem settings. The major challenge in these tasks represents the complex solutions needed for different maneuvers. For these reason, sampling based DRL methods with rich neural network function approximators are particularly well suited for this class of problems.

Manipulating Environmental Props (Fig. 9.5)

Real-world robotic agents will require constant interaction and manipulation in human-centric environments. Tasks in this class involve modification of the environment itself - opening drawers for fetching, moving furniture for cleaning, etc. The solution is often multi-step with hidden subgoals (e.g undo latch before opening doors), and lies on a narrow constrained manifold shaped primarily by the inertial properties and the under actuated dynamics of the environment.

Tool Use – Hammer (Fig. 9.6)

Humans use tools such as hammers, levers, etc. to augment their capabilities. These tasks involve co-ordination between the fingers and the arm to apply the tool correctly. Unlike object relocation, the goal in this class of tasks is to use the tool as opposed to just relocating it. Not all successful

grasp leads to effective tool use. Effective tool use requires multiple steps involving grasp reconfiguration and careful motor co-ordination in order to impart the required forces.

9.2.2 Demo Augmented Policy Gradient (DAPG)

In this work, we use a combination of RL and imitation learning to solve complex dexterous manipulation problems. To reduce sample complexity and help with exploration, we collect a few expert demonstrations using a VR system, and incorporate these into the RL process. We first present the base RL algorithm we use for learning and then describe our procedure to incorporate demonstrations.

Natural Policy Gradient

In this work, we primarily consider *policy gradient methods*, which are a class of *model-free* RL methods. In policy gradient methods, the parameters θ of the policy π_θ are directly optimized to maximize the objective $\eta(\theta)$ - defined in Eq. (9.2) - using local search methods such as gradient ascent. In particular, for this work we consider the NPG algorithm [230, 231, 232]. First, NPG computes the vanilla policy gradient, or REINFORCE [233] gradient:

$$g = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \hat{A}_\pi(s_t^i, a_t^i, t). \tag{9.6}$$

Secondly, it pre-conditions this gradient with the (inverse of) Fisher Information Matrix [230, 234] computed as:

$$F_\theta = \frac{1}{NT} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)^T, \tag{9.7}$$

and finally makes the following normalized gradient ascent update [231, 232, 235]:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{g^T F_{\theta_k}^{-1} g}} F_{\theta_k}^{-1} g, \tag{9.8}$$

where δ is the step size choice. A number of pre-conditioned policy gradient methods have been developed in literature [230, 231, 232, 235, 236, 237, 238] and in principle any of them could be used.

Augmenting RL with demonstrations

Although NPG with an appropriately shaped reward can somewhat solve the tasks we consider, there are several challenges which necessitate the incorporations of demonstrations to improve RL:

1. RL is only able to solve the tasks we consider with careful, laborious task reward shaping.
2. While RL eventually solves the task with appropriate shaping, it requires an impractical number of samples to learn - in the order of a 100 hours for some tasks.

Combining demonstrations with RL can help combat both these issues. Demonstrations help alleviate the need for laborious reward shaping, help guide exploration and decrease sample complexity of RL. We propose the demonstration augmented policy gradient (DAPG) method which incorporates demonstrations into policy gradients in two ways.

Pretraining with behavior cloning

Policy gradient methods typically perform exploration by utilizing the stochasticity of the action distribution defined by the policy itself. If the policy is not initialized well, the learning process could be very slow with the algorithm exploring state-action spaces that are not task relevant. To combat this, we use behavior cloning (BC) [239, 240] to provide an informed policy initialization that efficiently guides exploration. Use of demonstrations circumvents the need for reward shaping often used to guide exploration. This idea of pretraining with demonstrations has been used successfully in prior work [241], and we show that this can dramatically reduce the sample complexity for dexterous manipulation tasks as well. BC corresponds to solving the following maximum-likelihood problem:

$$\underset{\theta}{\text{maximize}} \sum_{(s,a)\in\rho_D} \ln \pi_\theta(a|s), \tag{9.9}$$

where ρ_D denotes the demonstrations data $\left\{s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}, r_t^{(i)}\right\}$ (t indexes time and i indexes different trajectories). The optimizer of the above objective, called the behavior cloned policy, attempts to mimic the actions taken

in the demonstrations at states visited in the demonstrations. In practice, behavior cloning does not guarantee that the cloned policy will be effective, due to the distributional shift between the demonstrated states and the policy's own states [242]. Indeed, we observed experimentally that the cloned policies themselves were usually not successful.

RL fine-tuning with augmented loss

Though behavior cloning provides a good initialization for RL, it does not optimally use the information present in the demonstration data. Different parts of the demonstration data are useful in different stages of learning, especially for tasks involving a sequence of behaviors. For example, the hammering task requires behaviors such as reaching, grasping, and hammering. Behavior cloning by itself cannot learn a policy that exhibits all these behaviors in the correct sequence with limited data. The result is that behavior cloning produces a policy that can often pick up the hammer but seldom swing it close to the nail. The demonstration data contains valuable information on how to hit the nail, but is lost when the data is used only for initialization. Once RL has learned to pick up the hammer properly, we should use the demonstration data to provide guidance on how to hit the nail. To capture all information present in the demonstration data, we add an additional term to the gradient:

$$g_{aug} = \sum_{(s,a)\in\rho_\pi} \nabla_\theta \ln \pi_\theta(a|s) A_\pi(s,a) + \sum_{(s,a)\in\rho_D} \nabla_\theta \ln \pi_\theta(a|s) w(s,a). \tag{9.10}$$

Here ρ_π represents the dataset obtained by executing policy π on the MDP, and $w(s,a)$ is a weighting function. This augmented gradient is then used in Eq. (9.8) to perform a co-variant update. If $w(s,a) = 0\ \forall(s,a)$, then we recover the policy gradient in Eq. (9.6). If $w(s,a) = c\ \forall(s,a)$, with sufficiently large c, it reduces to behavior cloning, as in Eq. (9.9). However, we wish to use both imitation and Reinforcement Learning, so we require an alternate weighting function. The analysis in [243] suggests that Eq. (9.6) is also valid for mixture trajectory distributions of the form $\rho = \alpha\rho_\pi + (1-\alpha)\rho_D$. Thus, a natural choice for the weighting function would be $w(s,a) = A_\pi(s,a)\ \forall(s,a) \in \rho_D$. However, it is not possible to

compute this quantity without additional rollouts or assumptions [244]. Thus, we use the heuristic weighting scheme:

$$w(s,a) = \lambda_0 \lambda_1^k \max_{(s',a') \in \rho_\pi} A_\pi(s',a') \;\; \forall (s,a) \in \rho_D,$$

where λ_0 and λ_1 are hyperparameters, and k is the iteration counter. The decay of the weighting term via λ_1^k is motivated by the premise that initially the actions suggested by the demonstrations are at least as good as the actions produced by the policy. However, towards the end when the policy is comparable in performance to the demonstrations, we do not wish to bias the gradient. Thus, we asymptotically decay the auxiliary objective. We empirically find that the performance of the algorithm is not very sensitive to the choice of these hyperparameters. For all the experiments, $\lambda_0 = 0.1$ and $\lambda_1 = 0.95$ was used.

9.2.3 Results

Our results study how RL methods can learn dexterous manipulation skills, comparing several recent algorithms and reward conditions. First, we evaluate the capabilities of RL algorithms to learn dexterous manipulation behaviors from scratch on the tasks outlined in Section 9.2.1. Subsequently, we demonstrate the benefits of incorporating human demonstrations with regard to faster learning and ability to cope with sparse task completion rewards. A more detailed analysis and further experiments are reported in [180].

Experimental setup

To test our algorithm, we use a high degree of freedom dexterous manipulator and a virtual reality demonstration system which we describe below.

We use a simulated analogue of a highly dexterous manipulator – ADROIT [245], which is a 24-DoF anthropomorphic platform designed for addressing challenges in dynamic and dexterous manipulation [227, 246]. The first, middle, and ring fingers have 4 DoF. Little finger and thumb have 5 DoF, while the wrist has 2 DoF. Each DoF is actuated using position control and is equipped with a joint angle sensor.

Our experimental setup uses the MuJoCo physics simulator [247]. The stable contact dynamics of MuJoCo [248] makes it well suited for contact rich hand manipulation tasks. The kinematics, the dynamics, and the sensing details of the physical hardware were carefully modeled to encourage physical realism. In addition to dry friction in the joints, all hand-object contacts have planar friction. Object-fingertip contacts support torsion and rolling friction. Though the simulation supports tactile feedback, we do not use it in this work for simplicity, but expect that its use will likely improve the performance.

Reinforcement Learning from Scratch

We aim to address the following question in this experimental evaluation:

- Can existing RL methods cope with the challenges presented by the high dimensional dexterous manipulation tasks?

In order to benchmark the capabilities of DRL with regard to the dexterous manipulation tasks outlined in Section 9.2.1, we evaluate the NPG algorithm described briefly in Section V, and the DDPG algorithm [249], which has recently been used in a number of robotic manipulation scenarios [229, 250, 251]. Both of these methods have demonstrated state of the art results in popular DRL continuous control benchmarks, and hence serve as a good representative set. We score the different methods based on the percentage of successful trajectories the trained policies can generate, using a sample size of 100 trajectories. We find that with sparse task completion reward signals, the policies with random exploration never experience success (except in the in-hand task) and hence do not learn.

In order to enable these algorithms to learn, we incorporate human priors on how to accomplish the task through careful *reward shaping*. With the shaped rewards, we find that NPG is indeed able to achieve high success percentage on these tasks (Fig. 9.7), while DDPG was unable to learn successful policies despite considerable hyperparameter tuning. DDPG can be very sample efficient, but is known to be very sensitive to hyperparameters and random seeds [252], which may explain the difficulty of scaling it to complex, high-dimensional tasks like dexterous manipulation.

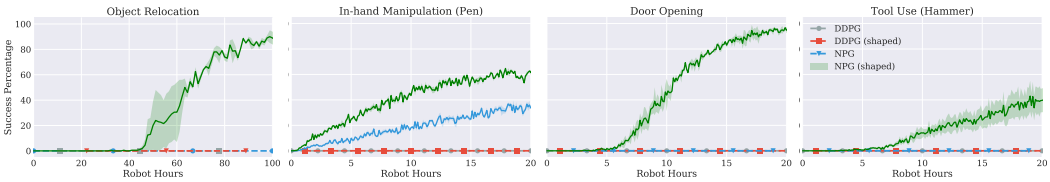


FIGURE 9.7: Performance of pure RL methods – NPG and DDPG, with sparse task completion reward and shaped reward. Sparse reward setting is primarily ineffective in solving our task set (expect in-hand manipulation). Incorporating human priors using reward shaping helps NPG get off the ground, but DDPG sill struggles to find success.

Although incorporation of human knowledge via reward shaping is helpful, the resulting policies are too sample inefficient to be useful for training on the physical hardware.

Reinforcement Learning with Demonstrations

In this section we aim to study the following questions:

1. Does incorporating demonstrations via DAPG reduce the learning time to practical timescales?

2. How does DAPG compare to other model-free methods that incorporate demonstrations, such as DDPGfD [250]?

We employ the DAPG algorithm in Section 9.2.2 on the set of hand tasks and compare with the recently proposed DDPGfD method [250]. DDPGfD builds on top of the DDPG algorithm, and incorporates demonstrations to bootstrap learning by: (1) Adding demonstrations to the replay buffer; (2) Using prioritzed experience replay; (3) Using n-step returns; (4) Adding regularizations to the policy and critic networks. Overall, DDPGfD has proven effective on arm manipulation tasks with sparse rewards in prior work, and we compare performance of DAPG against DDPGfD on our dexterous manipulation tasks.

For this Section of the evaluation we use only sparse task completion rewards, since we are using demonstrations. With the use of demonstrations, we expect the algorithms to implicitly learn the human priors on how to accomplish the task. Fig. 9.8 presents the comparison between the different algorithms. DAPG convincingly outperforms DDPGfD in all the tasks well before DDPGfD even starts showing signs of progress. Furthermore,

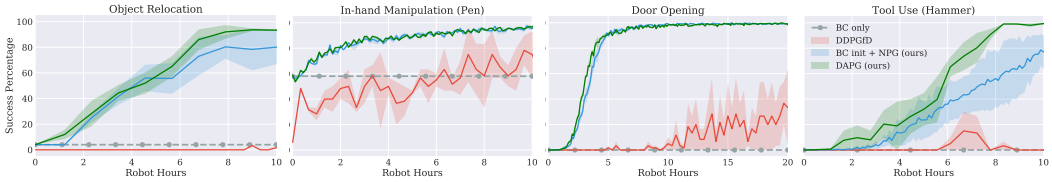


FIGURE 9.8: Performance of RL with demonstrations methods – DAPG(ours) and DDPGfD. DAPG significantly outperforms DDPGfD. For DAPG, we plot the performance of the stochastic policy used for exploration. At any iteration, the performance of the underlying deterministic policy will be better.

TABLE 9.1: Sample and robot time complexity of DAPG (ours) compared to RL (Natural Policy Gradient) from scratch with shaped (sh) and sparse task completion reward (sp). N is the number of RL iterations needed to achieve 90% success rate, Hours represent the robot hours needed to learn the task. Each iteration is 200 trajectories of length 2 seconds each.

Method	DAPG	(sp)	RL	(sh)	RL	(sp)
Task	N	Hours	N	Hours	N	Hours
Relocation	52	5.77	880	98	∞	∞
Hammer	55	6.1	448	50	∞	∞
Door	42	4.67	146	16.2	∞	∞
Pen	30	3.33	864	96	2900	322

DAPG is able to train policies for these complex tasks in under a few robot hours Table 9.1. In particular, for the object relocation task, DAPG is able to train policies almost 30 times faster compared to learning from scratch. This indicates that RL methods in conjunction with demonstrations, and in particular DAPG, are viable approaches for real world training of dexterous manipulation tasks.

9.3 Learning state representations for improving exploration

When no task demonstrations are available, exploration of the state space is a crucial component in RL algorithms [169, 170, 253, 254, 255, 255, 256,

257, 258], and remains notoriously difficult for problems with high dimensionality or sparse reward functions.

Most prior works assume that exploration is performed with no prior knowledge about the solution of the task. This assumption is not necessarily realistic and surely does not hold for humans that constantly use their knowledge and past experience to solve new tasks.

The idea of exploiting knowledge from prior tasks to fast adapt to the solution of a new task is a new promising approach already widely used in Deep RL [173, 259], specifically for addressing the exploration problem [174].

Driven by this motivation, the scenario we take into account in this Chapter is the following:

- We consider a family of tasks sampled from a distribution P_T.
- We assume N tasks $T_1, \dots, T_N \sim P_T$ to be already solved, and to have access to the states visited during each training $\{S^{(i)}\}_{i=1}^{N}$ together with the experienced rewards signals $\{R^{(i)}\}_{i=1}^{N}$. The set of states $S^{(i)}$ of the i-th task includes all the trajectories $\tau_t^{(i)}$ experienced during each training step t:

$$S^{(i)} = \{\tau_t^{(i)}\}_{t=1}^{t_{max}}, \tag{9.11}$$

  with t_{max} the number of training steps, $\tau = \{s_0 \in \mathbb{R}^n, \dots, s_l \in \mathbb{R}^n\}$ and l the length of each trajectory. Analogously for the rewards $R^{(i)}$:

$$R^{(i)} = \{r_t^{(i)}\}_{t=1}^{t_{max}}, \tag{9.12}$$

  with t_{max} the number of training steps, $r = \{R_0 \in \mathbb{R}, \dots, R_l \in \mathbb{R}\}$ and l the length of each trajectory.
- The goal is to efficiently use the information that can be extracted from the N tasks for fast exploration during the solution of a new task $T_{N+1} \sim P_T$.

The key idea of this work consists of learning from the N prior tasks a latent representation $z \in \mathbb{R}^p$ ($p < n$) of that portion of the state mostly affecting the experience of rewards. During the solution of the new task T_{N+1}, the latent representation z is then considered the subregion of the state on which to focus the exploration.

At this aim, we design:

- A multi-headed framework for reward regression on the tasks $T_1, \ldots, T_N \sim P_T$ to encode in the shared layers of the network the latent representation z.
- A novel exploration strategy consisting of the maximization of the entropy over the latent representation z rather than over the entire state space.

In the next Paragraphs the multi-headed regression (9.3.1) and the exploration strategy (9.3.2) are presented. Some preliminary experiments supporting our intuitions are shown in Paragraph 9.3.3. Even if the reported results are not enough for a complete evaluation of the method, we consider the contributions presented in this work to be promising and thus inspiring for new projects.

9.3.1 Learning the latent representation z

We assume N tasks to be sampled from the same distribution P_T. An example of task distribution is given by the object-pusher task (Fig. 9.9): a manipulator is required to push one specific object (among other objects) towards a target position. Each task can differ in the initial objects and goal positions. Our intuition suggests that tasks sampled from the same distribution P_T must share some common information useful for the solution of a new task $T_{N+1} \sim P_T$. In our example, we can easily notice that moving the object of interest is what really matters for the task solution, regardless of the other objects positions. Althought it might be easy sometimes to derive similar arguments, inferring a proper latent representation is not straightforward in general, especially for harder tasks. A possible way to extract this information is to make use of the reward signals collected during the solution of N past tasks. Proper reward functions in fact incorporate the information leading to the solution of the tasks.

At this aim, we design a multi-headed network (Fig. 9.10) where:

- The states $\{S^{(i)}\}_{i=1}^{N}$ collected during the training of the N tasks are the network input.
- The network has some shared layers (with parameters α_{shared}) followed by N separate heads (with parameters $\alpha_1, \ldots \alpha_N$).

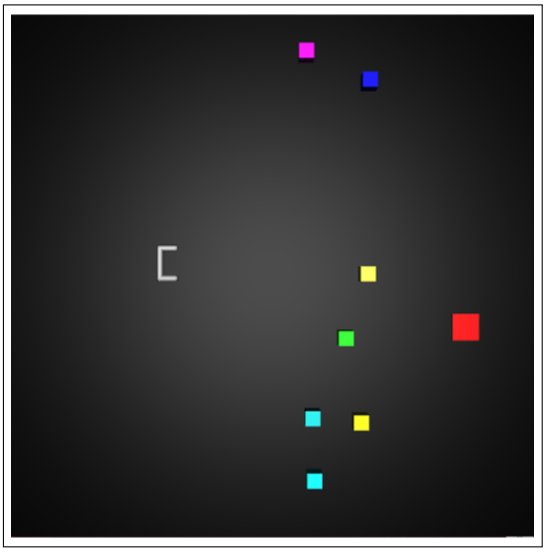


FIGURE 9.9: An example of 2D object-pusher environment. The task consists of moving the green object towards the target, represented with a red square. Only the gray pusher is actuated and it is responsible for the movements of the other objects.

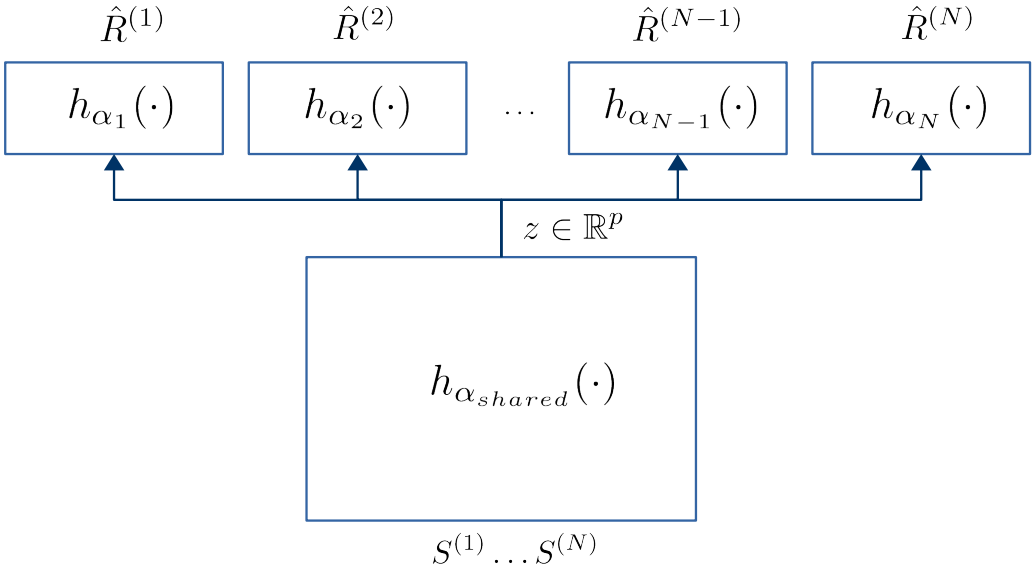


FIGURE 9.10: Multi-headed network for reward regression on N tasks.

- Each head outputs $\hat{R}^{(i)} = h_{\alpha_i}(h_{\alpha_{shared}}(S^{(i)}))$ are the estimate of the reward signals $R^{(i)}$ of the i-th task.

- The output of the shared layers is the latent variable $z = h_{\alpha_{shared}}(s)$, with $z \in \mathbb{R}^p$ and $s \in \{S^{(i)}\}_{i=1}^N \subseteq \mathbb{R}^n$. The network is designed so as to bottleneck the output of the shared layers and obtain a latent variable with lower dimension with respect to the states ($p < n$).

The shared layers and the heads of the network can be convolutional or shallow neural networks, according to whether the inputs $\{S^{(i)}\}_{i=1}^N$ are images or not, as it will be shown in Section 9.3.3.

The idea underlying the structure of such a network is that the shared layers should be able to learn what is important in the reward function, regardless of the specific task. The latent variable $z = h_{\alpha_{shared}}(s)$ should then represent that portion of the state responsible for experiencing the rewards. Moreover, the multi-headed structure prevents overfitting and allows better generalization.

The network training is formulated as a regression problem on each head:

$$\min_{\alpha} \sum_{i=1}^{N} \sum_{\substack{R \in R^{(i)} \\ \hat{R} \in \hat{R}^{(i)}}} \left\| R - \hat{R} \right\|^2, \tag{9.13}$$

where $\alpha = \{\alpha_{shared}, \alpha_1, \ldots, \alpha_N\}$.

9.3.2 Exploration via maximum-entropy bonus on the latent variable z

A family of strategies for speeding up state space exploration consist of adding a bonus $\mathcal{B}(s)$[170, 255, 256, 257, 258] to the reward function $R(s)$[5]:

$$R(s)_{new} = R(s) + \gamma \mathcal{B}(s), \tag{9.14}$$

where γ is a scaling factor for properly weighting the bonus with respect to the original reward. The goal of the bonus is to drive the learning algorithm when no or poor rewards $R(s)$ are provided. A possible choice for

[5]In this Chapter, we only consider rewards depending on the state, but this could be extended to the case of rewards depending also on the actions.

$\mathcal{B}(s)$ is the entropy $H(\cdot)$ over the state distribution $p(s)$ [256]:

$$R(s)_{new} = R(s) + \gamma H(p(s)), \tag{9.15}$$

with $H(p(s)) = E_{p(s)}[-log(p(s))]$. The policy π resulting from training with $R(s)_{new}$ maximizes both the original reward and the entropy over the state space density distribution $p(s)$. Maximizing the entropy of a distribution entails making the distribution as uniform as possible. The uniform distribution on a finite space is in fact the maximum entropy distribution among all continuous distributions that are supposed to be in the same space. This results in encouraging the policy to explore states not visited yet.

Even if reasonable, the choice of visiting all the possible states is not efficient when dealing with state space with high dimensionality or rewards that can be experienced only in a small subregion of the space. Maximizing the entropy over the portion of space responsible for the rewards is more efficient. For this reason, the augmented reward function we use for exploration is given by:

$$R(s)_{new} = R(s) + \gamma H(p(z)), \tag{9.16}$$

where $z = h_{\alpha_{shared}}(s) \in \mathbb{R}^p$ $(p < n)$ is the latent representation learned from prior tasks (Paragraph 9.3.1). Maximizing the entropy only over z is much more efficient because it represents the only portion of the state responsible for the rewards and has lower dimensionality with respect to the states $s \in \mathbb{R}^n$.

In order to estimate the density entropy $H(p(z)) = E_{p(z)}[-log(p(z))]$, we use a Variational AutoEncoder (VAE) [260] (Fig. 9.11). The loss used to train a VAE provides in fact a good estimation of $log(p(z))$ at the end of the training. Hereafter, we explain how it is possible to obtain such an approximation and how to use it for computing the entropy $H(p(z))$.

The structure of the VAE is the following:

- The input of the encoder $q_\psi(v|\cdot)$ is the latent representation $z = h_{\alpha_{shared}}(s)$. The procedure to learn $h_{\alpha_{shared}}(\cdot)$ has been presented in Paragraph 9.3.1.

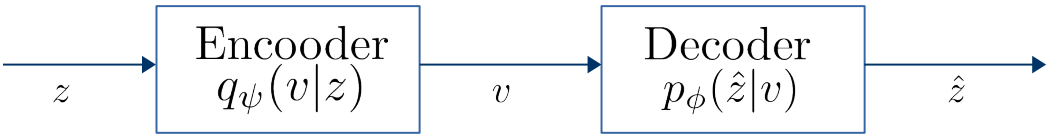


FIGURE 9.11: Variational Autoencoder.

- v is latent variable reconstructed by the VAE, i.e. the output of the encoder and input of the decoder . This quantity is not relevant for our formulation since we are not interested in dimensionality reduction. We just mention it for the sake of completeness.
- The output $\hat{z}$ of the decoder $p_\phi(\cdot|v)$ is the reconstruction of the input z.

The loss minimized during VAE training [261] is given by:

$$L(\psi, \phi) = -E_{q_\psi(v|z)}[log(p_\phi(\hat{z}|v)] + D_{KL}(q_\phi(v|z)||p(v)), \tag{9.17}$$

where D_{KL} is the Kullback-Leibler divergence between the encoder distribution $q_\phi(v|z)$ and the distribution $p(v)$. The first term of Eq. (9.17) is the *reconstruction loss*, or expected negative log-likelihood. This term encourages the decoder to learn to reconstruct the data. The second term is a regularizer that measures the information lost when using $q_\phi(v|z)$ to represent $p(v)$. In variational autoencoders $p(v)$ is chosen to be a standard Normal distribution $p(v) = N(0, 1)$.

The VAE loss function in Eq. (9.17) can be proved to be equal to the negative evidence lower bound (ELBO) [260], that is defined as:

$$-\text{ELBO} = -log(p(z)) + D_{KL}(q_\phi(v|z)||p(v|z)). \tag{9.18}$$

$p(v|z)$ cannot be computed analytically, because it describes the values of v that are likely to provide a sample similar to z using the decoder. The KL divergence imposes the distribution $q_\phi(v|z)$ to be close to $p(v|z)$. If we use an arbitrarily high-capacity model for $q_\phi(v|z)$, we can assume that - at the end of the training - $q_\phi(v|z)$ actually match $p(v|z)$ and the KL-divergence term is close to zero [260]. As a result, the final loss function of the VAE can be expressed as:

$$L(\psi, \phi) = -\text{ELBO} \simeq -log(p(z)), \tag{9.19}$$

therefore providing a good approximation of $-log(p(z))$. Recalling that the entropy $H(\cdot)$ of a distribution $p(z)$ is given by:

$$H(p(z)) = E_{p(z)}[-log(p(z))], \tag{9.20}$$

we can therefore augment our reward function as follows:

$$R_{new}(s) = R(s) - ELBO = R(s) - log(p(z)). \tag{9.21}$$

Using Policy Gradient[6] for training the policy π parametrized in θ, the maximization problem to be solved can be formulated as:

$$\max_{\theta} E_{p_\theta(a,s)}[R(s) - log(p_\theta(z))] = \max_{\theta} E_{(p_\theta(a,s)}[R(s)] + H(p_\theta(z)), \tag{9.22}$$

where $H(p_\theta(z)) = E_{p_\theta(a,s)}[-log(p_\theta(h_{\alpha_{shared}}(s)))]$. The expectation is computed with respect to the trajectories $(a, s) \sim p_\theta(a, s)$ obtained by running the current policy π_θ. For the sake of completeness, we should remark that our VAE reconstructs $p_\theta(z)$, the distribution of the *trajectories in* z obtained by executing the current policy π_θ, instead of $p(z)$ the distribution of the variable z. As shown in Algorithm 7, the VAE is in fact trained at each training step of Policy Gradient with the data collected by running the current policy π_θ and is therefore fed with trajectories $z \sim p_\theta(z)$, depending on the current policy, instead of samples $z \sim p(z)$[7]. Consequently, our approach maximize $H(p_\theta(z))$ instead of $H(p(z))$, encouraging the policy to generate *different trajectories in* z instead of visiting new z values.

The final exploration algorithm is summarized in Algorithm 7 and graphically represented in Fig. 9.12.

[6]Policy gradient methods are a family of reinforcement learning techniques that rely upon optimizing parametrized policies with respect to the expected return (long-term cumulative reward) by gradient descent.

[7]To be exact, the data we collect are in terms of the state s and the latent variable z are computed as $z = h_{\alpha_{shared}}(s)$.

[8]Trust Region Policy Optimizer is an algorithm similar to natural Policy Gradient methods that is effective for optimizing large nonlinear policies such as neural networks.

On policy methods evaluate or improve the same policy that is used to make decisions, whereas *off policy* methods evaluate or improve a policy different from one that was used to generate the data.

Algorithm 7 Maximum-entropy bonus exploration

Initialize the policy π_{θ_0};
Initialize the VAE encoder and decoder $q_\phi(v|\cdot)$ and $p_\psi(\cdot|v)$;
Let's define $r_t = \{R_0, \ldots, R_l\}$ and $\tau_t = \{s_0, \ldots, s_l\}$, with l the trajectory length (in case f rollouts are executed l is f times the trajectory length);
Use one *on policy* PG algorithm (TRPO [235] in our case [8]);
for $t = 1, \ldots, t_{max}$ **do**
    Collect data (τ_t, r_t) running π_{θ_t} ;
    Compute $r_{t,new} = r_t - ELBO(z_{t-1})$;
    Update policy parameters according the algorithm in use:
    $\theta_t \rightarrow \theta_{t+1}$
    Compute the latent representation $z_t = h_{\alpha_{shared}}(\tau_t)$;
    Train VAE on z_t.
end for

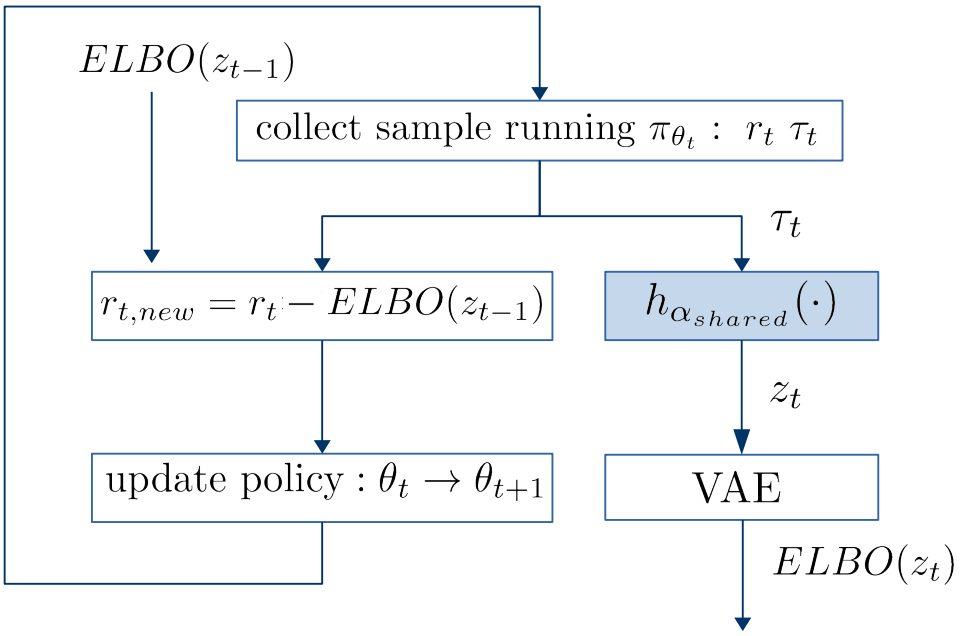


FIGURE 9.12: Maximum-entropy bonus exploration on a learned latent representation z. Note that $h_{\alpha_{shared}}(\cdot)$ was trained offline, using the data collected during the solution of prior tasks. During the maximum-entropy bonus exploration algorithm its parameters are kept fixed.

9.3.3 Preliminary results

The method proposed in the previous Paragraphs has been tested on the object-pusher environment (Fig. 9.9), where:

- The goal is to push the green object (identified as object no. 0) towards the red target.
- The environment state includes the objects ($o_0, \dots, o_6$), pusher (p) and target[9] (g) 2D positions:

$$s = [o_0, o_1, \dots, o_6, p, g]. \tag{9.23}$$

  The 2D space of the environment is finite and continuous.

- The action space is 2D and continuous, allowing the pusher to move forward - backward and laterally.
- The reward function is given by:

$$R = R_{pusher}(o_0) = \mathbb{1}\{d(o_0, g) < \delta\} d(o_0, g)^2, \tag{9.24}$$

  where $d(o_0, g)$ is the Euclidean distance between the green object and the target. The reward is then > 0 only when the object is sufficiently close to the target, i.e. in the circle with center g and radius δ. The value δ regulates the sparsity of the rewards.

- Different tasks of this environment differ in the initial object positions and target position.

In the next Paragraphs, we first show the multi-headed framework presented in Paragraph 9.3.1 to be able to learn a latent representation z for the position of the object of interest o_0, responsible for the reward function R_{pusher}. Then, we experimentally find that the exploration method proposed in Paragraph 9.3.2 is more efficient than other exploration strategies in our testing environment.

[9] The target position is constant during time.

Learning the latent representation z for the object-pusher environment

In order to learn the latent representation z, we use the data $(S^{(i)}, R^{(i)})_{i=1}^{N}$ collected during the training of $N = 30$ different object-pusher tasks[10]. Since the rewards $(R^{(i)} = R^{(i)}_{pusher})_{i=1}^{N}$ are sparse (as shown in Eq. (9.24)), the N tasks are sampled so that the initial object of interest and target position of all the tasks cover as much as possible the 2D environment. This is necessary to ensure that the sparse rewards are experienced *across the N tasks* in the entire 2D space for proper generalizing during the training of a new task.

The multi-headed neural network trained for learning z has:

- Two shared hidden layers with dimensions $[M, M/4]$, with $M = 16$ and ReLu activations, for representing $h_{\alpha_{shared}}(\cdot)$.
- Heads with two hidden layers each with dimensions $[M/4, M]$ and ReLu activations, representing each $h_{\alpha_i}(\cdot)$.

We recall that the network input is given by the collected states $\{S^{(i)}\}_{i=1}^{N}$ and we perform regression on the rewards $\{R^{(i)}\}_{i=1}^{N}$ using the different heads for estimating $\{\hat{R}^{(i)}\}_{i=1}^{N}$ (see Fig. 9.10). The training is performed using Adam optimizer and batch size 100.

At the end of the training, i.e. once each head is able to properly predict the rewards of the i-th task given the states, we can experimentally demonstrate that the output of the shared layers $z = h_{\alpha_{shared}}(s)$ represents the portion of the state responsible for the rewards. In this simple environment, it is clear that the reward function is directly affected only by the position of the object of interest o_0. We therefore expect our latent representation $z = h_{\alpha_{shared}}(s)$ to encode this information. In order to check the correct behavior of $h_{\alpha_{shared}}(\cdot)$ we perform the following tests, that we refer to as *relative distances analysis*. We feed the learned shared layers $h_{\alpha_{shared}}(\cdot)$ with three different sets of data:

- S_1: m states obtained by varying only the object of interest position o_0:

$$S_1 = \{s_1, \dots, s_m\}, \tag{9.25}$$

[10]To fasten the training, we use dense rewards during the N training, including also the distance between the pusher p and the object of interest o_0 and setting δ to be very large. We then compute the sparse rewards $R^{(i)}$ from the dense rewards, removing the extra information.

with $s_i = \{o_{0,i}, o_1 \ldots, o_7, p, g\}$ for $i = 1, \ldots, m$.

- S_2: m states obtained by varying only the position of another object o_1:

$$S_1 = \{s_1, \ldots, s_m\}, \tag{9.26}$$

with $s_i = \{o_0, o_{1,i} \ldots, o_7, p, g\}$ for $i = 1, \ldots, m$.

- S_3: m states obtained by varying only the manipulator position p:

$$S_1 = \{s_1, \ldots, s_m\}, \tag{9.27}$$

with $s_i = \{o_0, o_1 \ldots, o_7, p_i, g\}$ for $i = 1, \ldots, m$.

The entity of each variation ($s_i - s_{i-1}$ for $i = 2, \ldots, m$) is the same in the three data sets S_1, S_2, S_3.

We call the output of the shared layers respectively:

$$\begin{aligned} Z_1 &= h_{\alpha_{shared}}(S_1), \\ Z_2 &= h_{\alpha_{shared}}(S_2), \\ Z_3 &= h_{\alpha_{shared}}(S_3), \end{aligned} \tag{9.28}$$

with $Z_i = \{z_1, \ldots, z_m\}$ for $i = 1, 2, 3$.

For each set Z_i we compute the following quantities:

$$D_i = \{d_{l,j}\}_{l,j=1}^{m,m}, \text{ for } i = 1, 2, 3 \tag{9.29}$$

where

$$d_{l,j} = z_l - z_j. \tag{9.30}$$

D_i therefore contains the distance of each output collected in Z_i with respect to the other outputs still belonging to Z_i, for total of $m \times m$ values for each D_i. We refer to these quantities as *relative distances*. Larger relative distances correspond to outputs $z_1, \ldots, z_m$ very different among each other and then, able to well represents variations in the corresponding inputs $s_1, \ldots, s_m$.

Fig. 9.13 shows the relative distances D_1, D_2, D_3 respectively in green, blue and magenta. Each radial line l for $l = 1, \ldots, m$ corresponds to $\{d_{l,j}\}_{j=1}^{m}$ and each dot of each line to the j-the distance $d_{l,j}$. We can infer from the plot that the learned latent representation z is more sensitive to

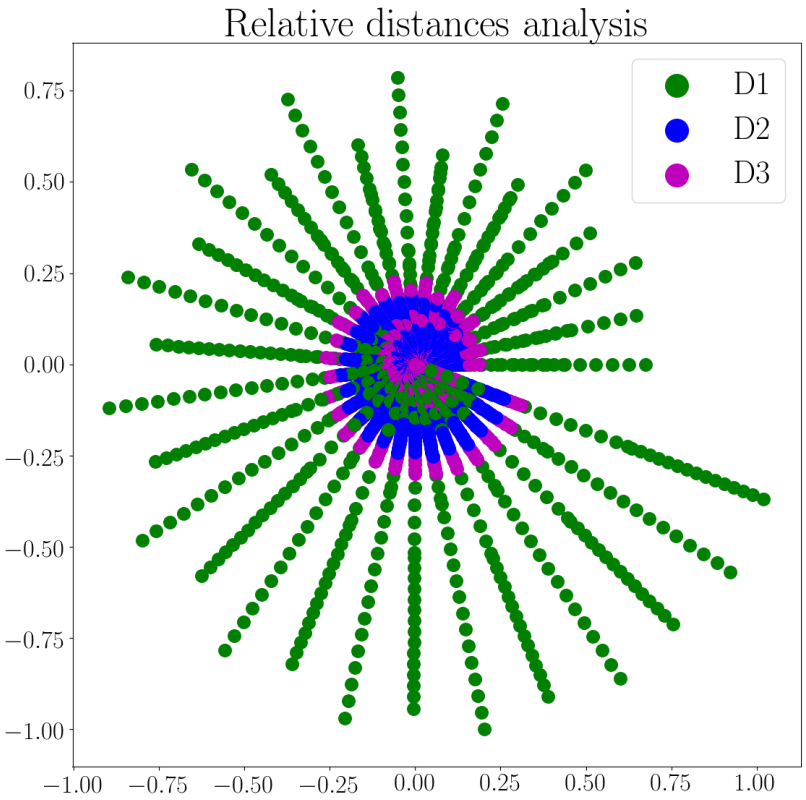


FIGURE 9.13: Relative distances analysis of the learned latent representation z when using $(S^{(i)}, R^{(i)})_{i=1}^{N}$. D1 (green) contains the relative distances when feeding $h_{\alpha_{shared}}(\cdot)$ with different position of the object of interest o_0, D2 (blue) with different positions of another object and D3 (magenta) with different pusher positions.

the variation of the object of interest position (S_1), since the green relative distances D_1 are larger than the others. This is evident in the plot because the distances $d_{l,j}$ between the dots of each radial green line are larger, making the green circle have a larger radius. The radius of the green, blue and magenta circles therefore represent the sensitivity of the latent representation $z = h_{\alpha_{shared}}(s)$ with respect to the three set of data S_1, S_2, S_3. In summary, this test proves that the latent variable z actually represents the portion of the state responsible for the rewards, as it is more sensitive to variations of the object of interest position.

Remarkably, our framework is suitable to perform multi-headed regression also on images. This is very relevant because it allows obtaining a latent representation z with a much lower dimension with respect to the entire image. In fact, the variation of only a few pixels (the green block) is actually responsible for variations of the rewards. Factoring out, information leads to considerably reduce the space where to perform exploration when, for instance, the state is not completely accessible and we are therefore forced to learn from raw images (that is closer to practical applications).

We train a multi-headed *convolutional* neural network with the data set $(I^{(i)}, R^{(i)})_{i=1}^{N}$, including, instead of the states visited, the images collected during the training of the N past tasks. The images size is 84×84. The CNN structure used in this case is the following:

- two convolutional (conv) layers[11] with leaky Relu activations followed by two fully connected layers (FC) of dimensions (192,48) with leaky Relu activations for the shared part $h_{\alpha_{shared}}(\cdot)$;
- two FC layers of dimensions (48, 48) with leaky Relu activations for each head $h_{\alpha_i}(\cdot)$.

We perform the training using Adam optimizer and batch size 50. In order to perform the relative distances analysis, the set S_1, S_2, S_3 now contains the *images* when varying respectively, only the object of interest, another object or the pusher. The outputs Z_1, Z_2, Z_3 and the distances D_1, D_2, D_3 can be computed as shown in Eqs. (9.28) - (9.30), just using the new $h_{\alpha_{shared}}(\cdot)$ function. Fig. 9.14 confirms that the multi-headed framework is able to learn a proper latent variable z also regressing on images.

Maximum-entropy bonus exploration

As motivated in Paragraph 9.3.2, we propose to encourage exploration by maximizing the entropy over the latent representation $H(p_\theta(z))$. In order to support our intuition, we perform the following analysis:

[11] The conv layers parameters are: filter size: 3, depth of the output volume: 16, stride to slide the filter: 2, padding:"same", i.e. the output size is the same of the input.

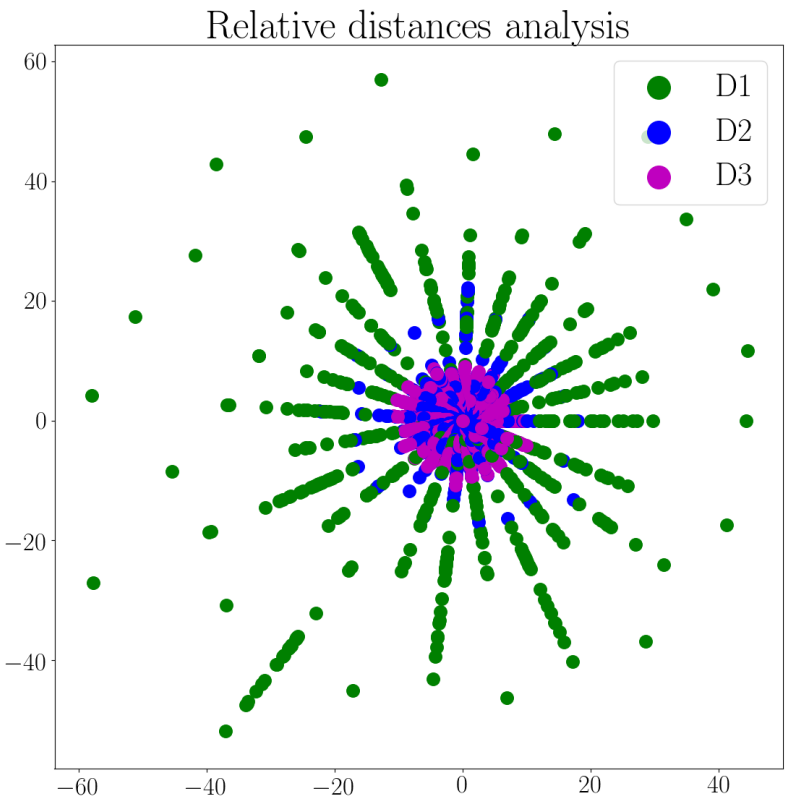


FIGURE 9.14: Relative distances analysis of the learned latent representation z when using $(I^{(i)}, R^{(i)})_{i=1}^{N}$. D1 (green) contains the relative distances when feeding $h_{\alpha_{shared}}(\cdot)$ with different position of the object of interest o_0, D2 (blue) with different positions of another object and D3 (magenta) with different pusher positions.

- We train three policies π_1, π_2, π_3 in order to maximize the rewards:

$$\begin{aligned} R_1 &= H(p_\theta(o_0)), \\ R_2 &= H(p_\theta(z)), \\ R_3 &= H(p_\theta(s)). \end{aligned} \tag{9.31}$$

The three policies are therefore trained in order to make respectively the distribution of the trajectories of the 1) *object of interest position* o_0, 2) the *latent representation* z p and 3) the *entire state* s as uniform as possible. The policies are trained by using Algorithm 7, with $R_{new} = R_1, R_2$ and R_3 and, when rewards R_1 and R_3 are used, without using

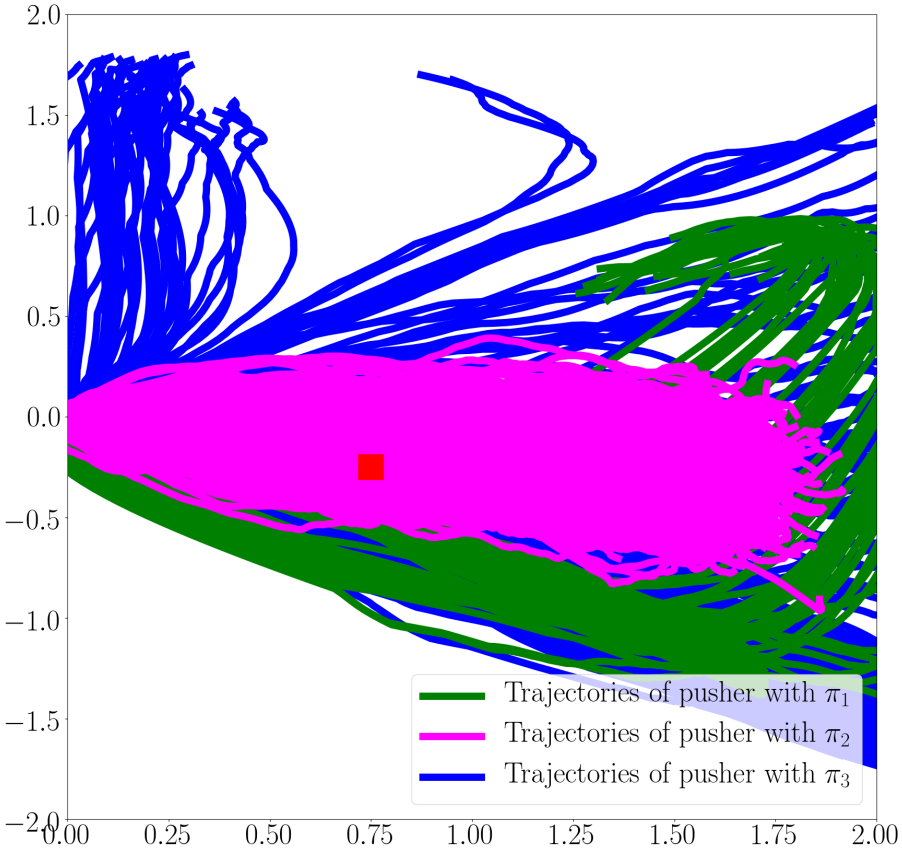


FIGURE 9.15: 100 pusher trajectories obtained when running the trained policy π, π_2, π_3. The initial position object the object of interest o_0 is represented with a red square.

the learned latent representation z but directly feeding the VAE with, respectively, o_0 and s to estimate $-log(p_\theta(o_0))$ and $-log(p_\theta(s))$. Figs. 9.15 and 9.16 reports the trajectories followed by the *pusher* and the *object of interest* o_0 when running the three trained policies. When the policy is asked to maximize only the object of interest trajectories, the pusher (green trajectories in Fig. 9.15) focuses the efforts in moving towards the object of interest (whose initial position is represented with a red square). The consequent object trajectories are shown in green in Fig. 9.16. Analogous trajectories both for the pusher and the object are generated by π_2, obtained by maximizing $R_2 = H(p_\theta(z))$

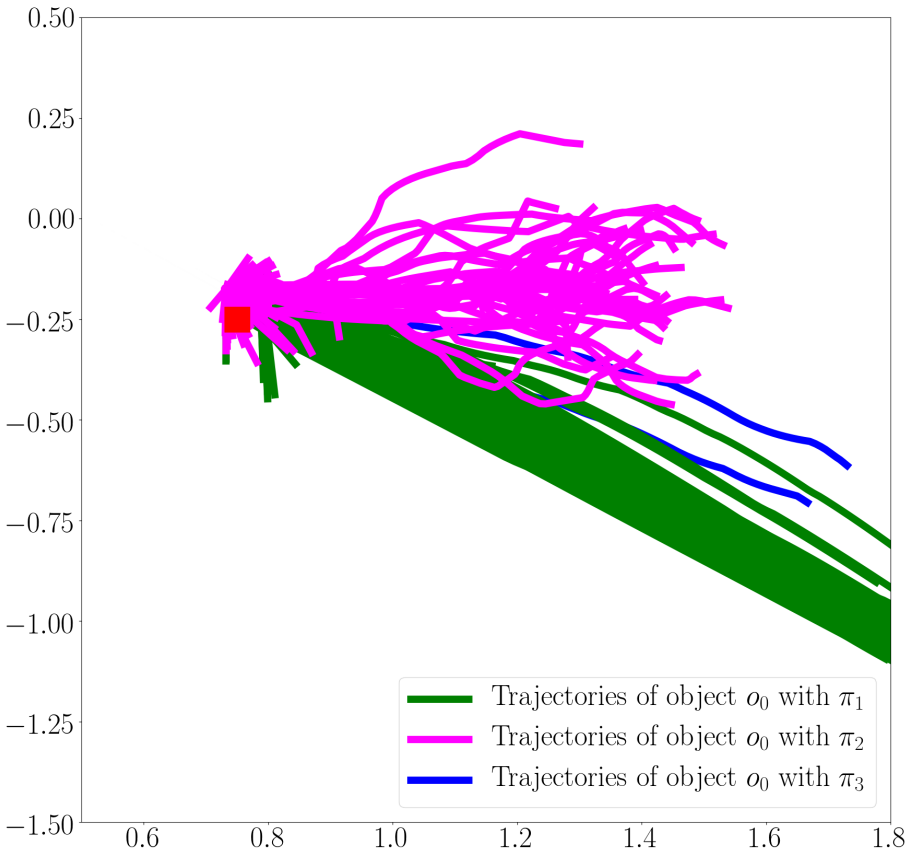


FIGURE 9.16: Object trajectories obtained when running 100 times each trained policy π, π_2, π_3. The number of trajectories shown is less than 100 for each policy because not all the policy executions lead to movements of the object of interest.

(in magenta in 9.15 and 9.16), meaning that our latent representation correctly encodes the position of the object of interest. The pusher in the other cases instead explores the entire state and the object is rarely pushed (blue trajectorie in 9.15 and 9.16).

- We train three policies π_1, π_2, π_3 in order to maximize the following rewards:

$$\begin{aligned} R_1 &= R_{pusher}(o_0) + H(p_\theta(o_0)), \\ R_2 &= R_{pusher}(o_0) + H(p_\theta(z)), \\ R_3 &= R_{pusher}(o_0) + H(p_\theta(s)), \end{aligned} \tag{9.32}$$

with radius of the circle in which the reward is different from zero

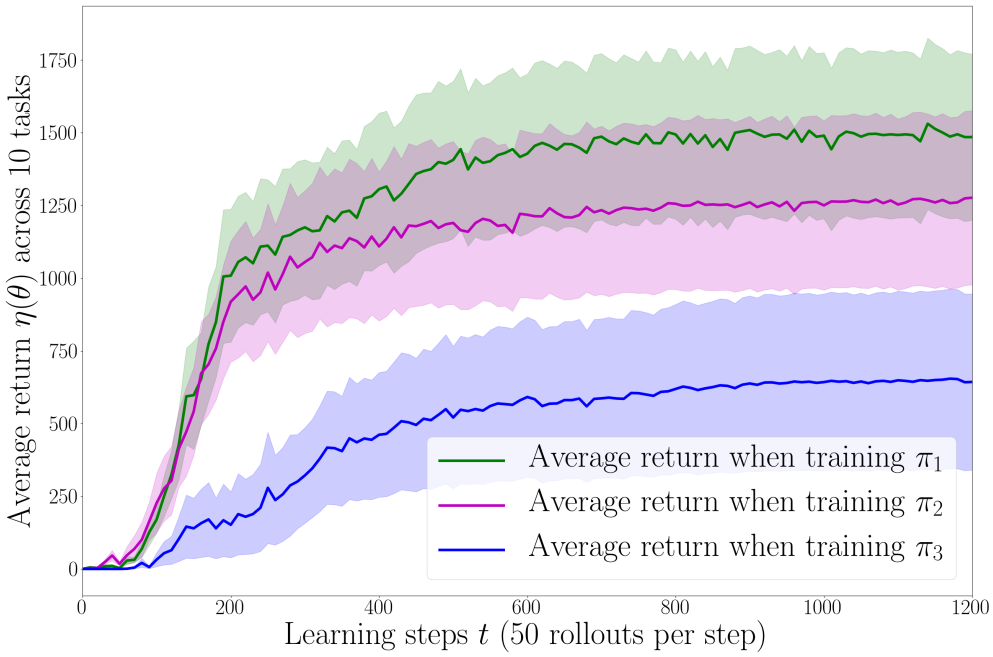


FIGURE 9.17: Average returns $\eta(\theta)$ with different reward bonus.

$\delta = 0.1$ (see Eq. (9.24)). Fig. 9.17 reports the average returns $\eta(\theta)$ (defined in Eq. (9.2)) obtained when training π_1, π_2 and π_3. We consider the training curve obtained with reward R_1 as an oracle, because in this case the task is solved by maximizing the entropy of the *exact portion* o_0 responsible for the rewards. The training curve obtained when training π_3 is instead the baseline, since the policy is required to maximize the entropy over the entire state s. The plot shows that maximizing $R_2 = R_{pusher}(o_0) + H(p_\theta(z))$ (i.e. our approach, summarized in Algorithm 7) leads to performance almost as good as using

the oracle and considerably better than running the baseline. This is at the same time a proof of the effectiveness of our learned latent representation z and of our maximum-entropy bonus exploration strategy.

- We finally compare our approach (Algorithm 7) with some basic baselines in Fig. 9.18. We respectively report the average return during training of:

  1. Our approach, shown in Algorithm 7.
  2. TRPO with Maximum-entropy bonus over the *entire state* s.
  3. TRPO with Maximum-entropy bonus over the *action state*. In this case the policy is encouraged to perform all the possible actions, making the actions distribution as uniform as possible.
  4. TRPO with no exploration bonus in the reward function.

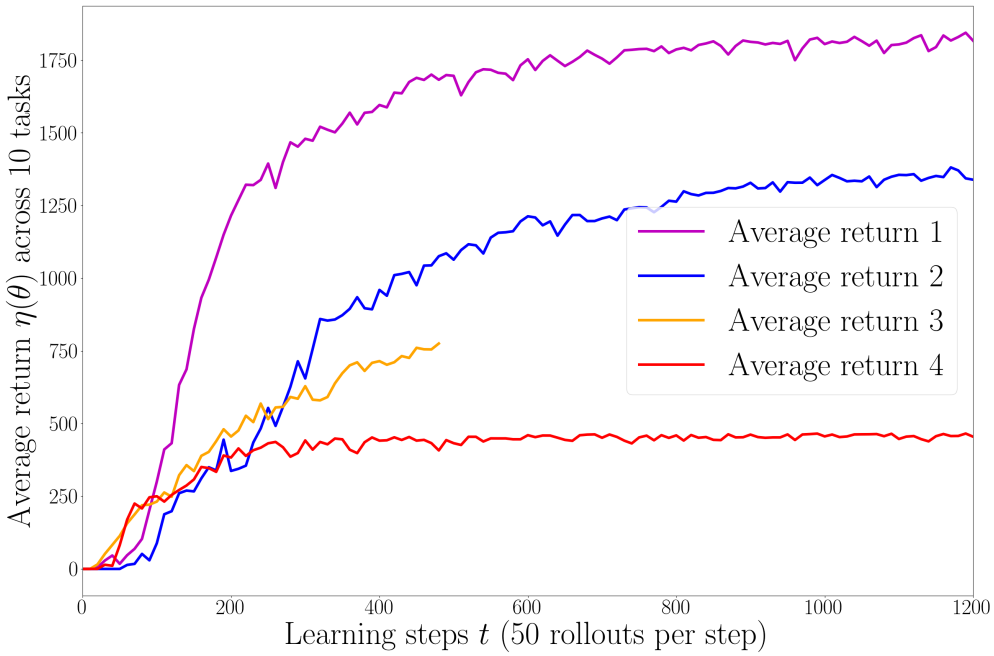


FIGURE 9.18: Average return $\eta(\theta)$ of our approach and some basic baselines. We do not show the standard deviation to facilitate the reading of the plot.

Even if a more intensive evaluation of the approach is required to ensure its effectiveness (e.g. on different environments and training from raw images), the results shown thus far prove that, at least for the considered environment, we are able to learn the latent representation z and that the maximum-entropy bonus on this variable benefits for exploration, making our intuitions promising for future applications.

A possible future application regards the more realistic task of grasping objects standing on a table. Training a RL algorithm to solve this kind of task with sparse rewards is pretty challenging, due to the difficulty in efficiently exploring high-dimensional state spaces. Exploration could be improved by focusing only on a subspace of the values allowed for the robot arm joints. Intuitively, this subspace could include those joints values that lead to end-effector positions close to the table and make the fingers move in a coordinated manner. In fact, exploring regions of the space far away from the table are useless for the solution of the task, such as moving randomly the fingers. While it is very hard to hand-craft a mathematical formulation for such a state subspace, our approach provides a possible solution by directly learning from the solution of analogous tasks a latent representation that encodes the relevant subspace.

9.4 Discussion

This Chapter collects the description of two different projects addressing the problem of exploration in RL, i.e. how to efficiently explore the state space to speed up the experience of rewards and, consequently, the solution of the task. The method reported in Section 9.2 faces the exploration problem relying on human demonstrations. In the detail, the proposed method, DAPG, incorporates demonstrations into policy gradient methods. The empirical results we carried out highlight that our DAPG algorithm acquires policies that are substantially more robust than those obtained with other state of the art algorithms. Furthermore, we found that DAPG can be up to 30x more sample efficient than RL from scratch with shaped rewards. DAPG is able to train policies for the tasks we considered in under 5 hours, which is likely practical to run on real systems. Although success remains to be demonstrated on real hardware, the complexity of the tasks in our evaluation and the sample-efficiency of our DAPG method

makes this work a significant step toward practical real-world learning of complex dexterous manipulation. In future work, we aim at learning policies on real hardware systems, further reducing sample complexity by using novelty based exploration methods, and processing only raw visual inputs and tactile sensing.

In Section 9.3, we proposed a novel exploration algorithm that, instead of relying on human demonstrations, leads effectively the training of the desired task by using information learned from the solution of similar prior tasks. In particular, we proposed a multi-headed network (Section 9.3.1) capable of learning in its shared layers the portion of state space responsible for experiencing the reward in the task family of interest. This information was then encoded into an exploration algorithm based on entropy maximization (Section 9.3.2). Preliminary tests (Section 9.3.3) showed that the proposed method leads to a faster solution of new tasks sampled from the task distribution under consideration. The promising results encourage us to extend this work in the next future by including processing from raw pixels and tests on more complex tasks, such as robotic manipulation.

Part VI

Conclusions

Chapter 10

Conclusions and future work

In this work of Thesis, we proposed possible solutions to some key sub-problems of autonomous manipulation, focusing on how to process perceptive information for interacting with objects.

The first contribution is the design of a novel *6-DOF tactile object localization algorithm* based on recursive Bayesian estimation, named the *Memory Unscented Particle Filter (MUPF)* (Chapter 4). The proposed algorithm is capable of localizing tridimensional objects through 3D contact points collected on the object surface with good overall performance and by exploiting a reduced number of particles. The MUPF algorithm relies on the Unscented Particle Filter suitably adapted to the localization problem of interest. We implemented a probabilistic description of tactile sensors in terms of likelihood and measurement function allowing respectively to evaluate the quality of the current estimate and predict the measurement given the estimated state. In addition, the standard UPF algorithm has been modified by the inclusion of a suitable sliding memory (hence the name MUPF) of past measurements in the update of the particle importance weights. In this respect, it was found that the memory feature is crucial for a careful exploitation of the available contact point measurements with consequent improvement of localization accuracy. Performance evaluation, carried out via simulation and experimental tests, demonstrated that the algorithm is reliable and has good performance also in presence of measurement noise.

The MUPF has been successfully applied also on a challenging tactile recognition task (Chapter 5). *Tactile object recognition* has been formulated as a localization problem applied to multiple objects, where the solution is provided by the object whose localization error is the lowest among all the considered objects. This approach has been tested on a set of objects very

similar in shape and dimensions both using simulated and real measurements.

The second main contribution of this work is given by the implementation of a *pipeline for the execution of the handover task* on the iCub humanoid robot (Chapter 6). In practice, the robot is asked to pass an object (whose model is known) from one hand on to the other. The pipeline can be summarized as follows. The robot grasps the desired object with the first hand and it controls the grasp using tactile feedback. The MUPF then estimates the object in-hand pose using a partial 3D point cloud of the object extracted with stereo vision. The MUPF in fact just requires to be fed with 3D points collected on the object surface, including also point clouds. The object model is a-priori annotated with a set of grasping poses reachable by the second hand. After the object is localized, the candidates are ranked according to proper criteria and the best pose for performing the handover is then selected. The method has been experimentally evaluated with the iCub humanoid robot, showing that it provides a high success rate for some objects of the YCB dataset different in shape, texture and dimensions.

As further contribution, we designed and implemented on the robot iCub an *object modeling and grasping pipeline* based on superquadric functions (Chapter 7). The leading idea of the approach is to use superquadrics to represent both the volume graspable by the robot hand and the object of interest. While the former is given a priori as it depends only on the hand geometry, the latter is estimated from an object partial point cloud. A proper grasping pose is then obtained by overlapping the superquadric representing the volume graspable by the robot hand onto the object superquadrics while meeting some orientation and obstacle avoidance constraints (e.g. avoidance of the support on which the object is located). The pipeline has been tested on a large number of objects, including also YCB objects, computing a grasp candidate for both the left and right hand and then choosing the best one according to a proper cost function.

The first implementation of the modeling and grasping pipeline envisages the use of a single superquadric for modeling the object. Since representing some objects with a single superquadric leads to too rough or even wrong models, we extended the modeling and grasping methods to

deal with *multi-superquadric object representation* (Chapter 8). The multi-superquadric model is obtained by iteratively splitting the object point cloud with simple geometrical criteria, fitting a superquadric for each portion and building the so-called superquadric-tree $\mathcal{ST}$. The structure of the superquadric-tree is then used to infer whether the portions of the point cloud need to be represented with different superquadrics or can be merged together. The computation of the grasping pose can be instead performed by extending the technique proposed for single-superquadric models to deal with more superquadrics. A grasp candidate is computed for each superquadric of the model and the best one according to proper criteria is then selected for performing the grasp. The approaches have been tested on a larger number of objects in simulation.

The conclusive Chapter 9 collected the activities carried out at Berkeley Artificial Intelligence Research Lab (BAIR). In particular, we reported on the work carried out during a joint Deep RL project on robotic manipulation and the development of a new algorithm addressing the exploration problem in Reinforcement Learning.

The contributions detailed in this work of Thesis suggest several perspectives for future work. As mentioned in Chapter 4, the Memory Unscented Particle Filter is agnostic to the source of measurement, taking advantage of the 3D points collected on the object surface. It has been used in fact for localizing objects using tactile measurements (Chapter 4) or 3D point clouds acquired by means of stereo vision (Chapter 6). A possible extension which we are currently working on is to *combine visual and tactile information to estimate the object pose*. Object point clouds can be for example used for localizing initially the object. While interacting with the objects instead, tactile information is exploited to assist or even replace visual feedback in order to cope with possible occlusions.

Another possible future application of the methods developed in this work of Thesis is the combination of the handover pipeline with the superquadric modeling and grasping approach in order to perform *handover of unknown objects*. The object in-hand point cloud could be in fact used for reconstructing the superquadric object model. The grasping pose computation could be then modified so as to adapt to the handover requirements.

A limitation of the superquadric modeling and grasping pipeline presented in Chapters 7 and 8 is given by the fact that no supervision on the quality of the object model and the grasping pose is provided. In particular, we showed how to select the best hand for grasping an object but we make no previsions for deciding whether both poses are unsatisfactory and should be discarded. A viable solution to *quantify the quality of object models and grasping pose*s is to rely on *supervised learning* techniques after proper data collection and labeling. Another benefit the learning framework could introduce is in terms of computation time for multi-superquadric models. A *3D CNN could be trained with a proper dataset to generate complex multi-superquadric models* processing the object point cloud and comply with real-time requirements.

An alternative approach to *deal with ineffective grasping poses* is to cast the problem of grasp candidates improvement in the *Deep RL framework*. The reward to be maximize could be formulated as the quality of the grasping pose, in terms of grasp stability and the ability of lifting the object (information retrievable via touch and vision). The reward should guide towards the modifications of the grasping pose components allowing only small deviations from the initial pose retrieved with the superquadric method. This way, the training on the real robot should be safe, since no random poses or movements could be executed by the robot.

Concluding, the study presented in this work of Thesis suggests how remarkable improvements in autonomous manipulation could be attained by means of *combining* classical or *model-based methods* with the recent *data-driven approaches* developed into *learning framework*.

Part VII

Appendix

Appendix A

Superquadric modeling and grasping pipeline: implementation

Even though the object modeling and grasping approach described in Chapter 7 is designed for a generic humanoid robot, we developed a complete software architecture for executing this approach on the iCub humanoid robot. We provide some details about the implementation in this Appendix.

We designed two modules, namely *superquadric-model*[1] and *superquadric-grasp*[2], which implement, respectively, the modeling and the grasping approached described in Chapter 7. Our leading idea is to develop a *self-contained* code that provides *query services* to the user. In this respect, our code handles only the information strictly necessary for the superquadric modeling and grasping approach and minimizes the dependencies from external modules. The user is asked to write a wrapper code, that communicates with the two modules and makes them properly interact. In this respect, we provide a tutorial code[3], implementing a possible use case of our modules, that can be adapted by the user to fit in his own pipeline.

The implementation of a complete modeling and grasping pipeline requires the use of external modules for point cloud computation. Several modules developed by the iCub community implement the image processing necessary for point cloud extraction. Hereafter, we report the main

[1] https://github.com/robotology/superquadric-model.
[2] https://github.com/robotology/superquadric-grasp.
[3] https://github.com/robotology/superquadric-grasp-example.

steps of the complete pipeline. For the sake of clarity, we recall a graphical example of the pipeline execution in Fig. A.1. The entire commented code is available on Github[4], together with a detail description on how to run the code in the *README.md* file.

- The object is classified according to its similarity to a primary shape through the classifier system[5] properly trained as shown in Section 7.3.1. The object label, together with information on its 2D bounding box, are stored by the *Object Property Collector*[6] ([262]). The wrapper code is given the object class by the user and uses it for asking the object property collector for the relative 2D bounding box.

- The 2D blob of the object is computed by the *lbpExtract module*, once it is provided with the bounding box information. This uses Local Binary Pattern (LBP) ([213]) in order to analyze the texture of what is in the robot view (a table in our experimental scenario). Then, the general blob information allow using grabCut algorithm ([214]) to properly segment all the objects on the table.

- Given the 2D blob, the wrapper code reconstructs the 3D point cloud by querying the *Structure from Motion module* ([181]). This module uses a complete Structure From Motion (SFM) pipeline for the computation of the extrinsics parameters between two different views. These parameters are then used to rectify the images and to compute a depth map.

- Then, the wrapper code asks the *superquadric-model* to estimate the superquadric modeling the object by sending the acquired point cloud and the object label to the module.

- Once the superquadric is estimated, the user code asks the *superquadric-grasp* module to compute pose candidates for grasping the object and the corresponding costs.

- Finally, the user can ask the *superquadric-grasp* to perform the grasping task by selecting the best hand.

[4]https://github.com/robotology/superquadric-grasp-demo.
[5]https://github.com/robotology/iol/tree/master/src/himrepClassifier.
[6]https://github.com/robotology/icub-main/tree/master/src/modules/objectsPropertiesCollector.

Fig. A.2 outlines the structure of the entire pipeline, following the steps described in this Section.

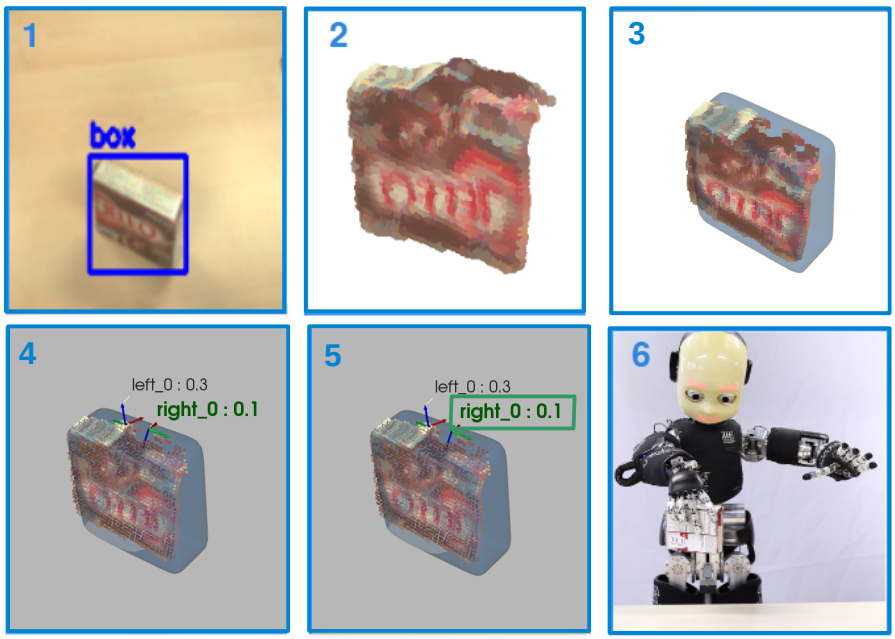


FIGURE A.1: Outcomes of the modeling and grasping pipeline. 1) The object is stored by the object property collector with the label *box*. LbpExtract provides the 2D blob of the object. 2) The 3D point cloud is extracted from the disparity map, by querying the Structure From Motion module. 3) The superquadric modeling the object is reconstructed. 4) The grasping pose for the right and left hand are computed. 5) The best hand for grasping the object is selected. 6) The robot grasps the object.

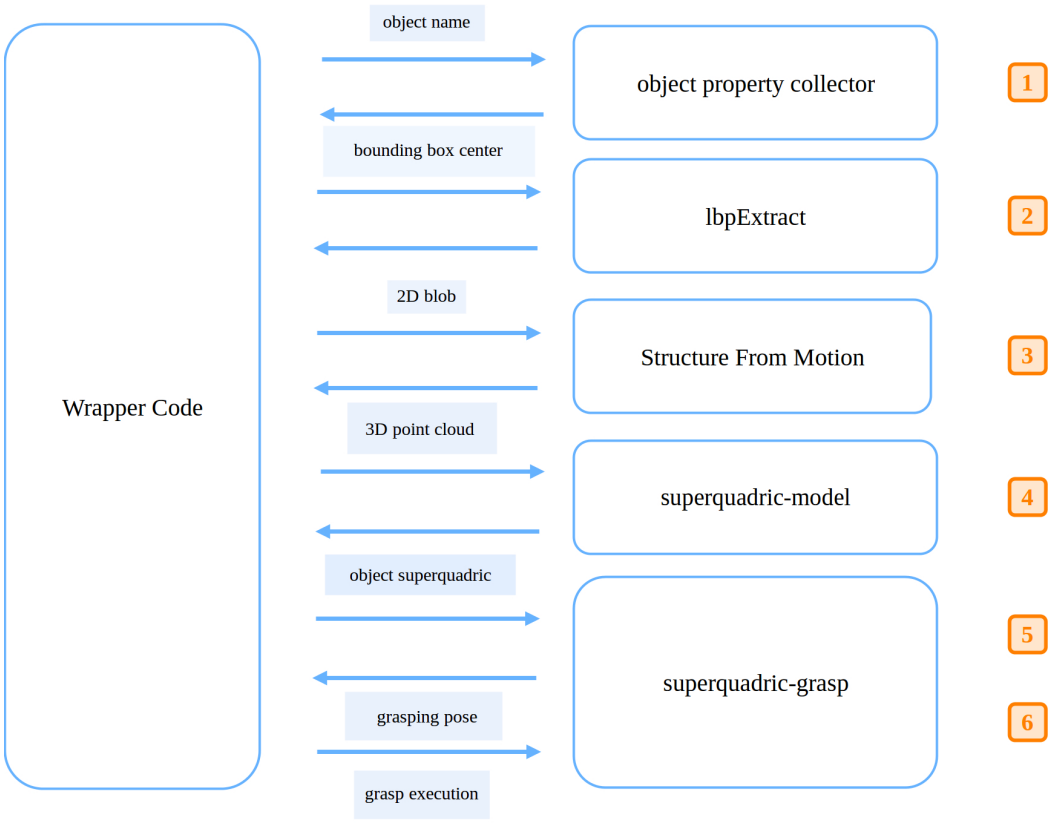


FIGURE A.2: Modules communication for the implementation of the modeling and grasping pipeline.

Appendix B

Grasping pose computation with superquadrics for markerless visual servoing on unknown objects

The superquadric modeling and grasping approach described in Chapter 7 has been intensively tested on the iCub humanoid robot providing an average success percentage of 85% on 18 objects. Although competitive with other grasping state of the art techniques, the pipeline reliability can still be improved. As already mentioned in Section 7.7, we noticed that the main source of failures is not given by the modeling and grasping outcomes but instead is represented by the uncalibrated eye-hand system of the robot that entails non-negligible misplacements of the robot hand when reaching for the target pose. This problem is peculiar of humanoid robots in that elastic elements lead to errors in the direct kinematics computation. Moreover, robots with moving cameras, such as the iCub platform, need to deal with errors in the visual estimation of the object pose due to imprecise knowledge of the cameras extrinsic parameters.

This Appendix briefly shows how these errors can be compensated by closed loop control techniques of the end-effector resorting to a visual feedback. In particular, a recursive Bayesian filtering technique, based on Sequential Monte Carlo (SMC) filtering, estimates the 6D pose of the robot end-effector without the use of markers [209]. Then, an image-based visual servo control commands the robot end-effector precisely toward the grasping pose computed using superquadrics [222].

Going into the details both of the robot end-effector pose estimator and

image-based visual servo control does not fall within the scope of this Thesis. For the full explanation of the methods we refer to [209] and [222]. Hereafter we report the complete modeling and grasping pipeline embedding markerless visual servoing for precise pose reaching. The pipeline is outlined in Fig. B.1 and consists of the following steps:

S1. Our modeling approach reconstructs a superquadric representing the object by using a 3D partial point cloud acquired from stereo vision.

S2. The estimated model is exploited by the pose computation method for providing a grasping pose[1].

S3. An open loop phase brings the robot's end-effector in the proximity of the object and in the cameras field-of-views.

S4. The 3D model-aided particle filter of [209] estimates the end-effector pose using RGB images.

S5. Visual servoing [222] uses the particle filter output of **S4.** in order to reach for the pose computed in **S2.**.

S6. Reaching completes and the robot grasps the object.

[1]The current implementation can only deal with the right robot hand but the method is general and it is straightforward the extension to two hands.

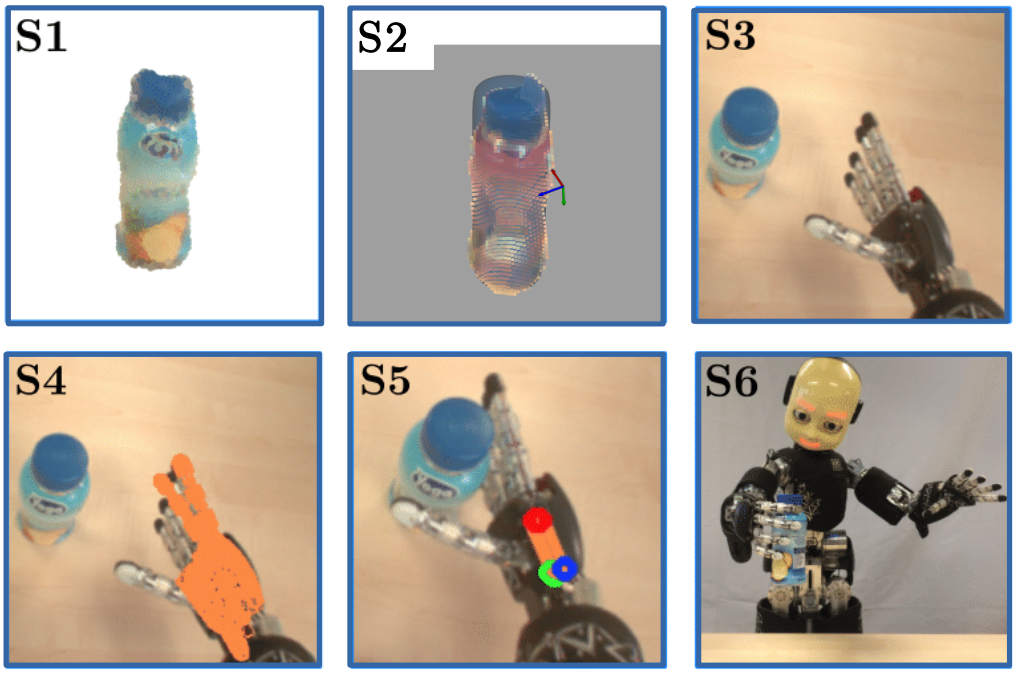


FIGURE B.1: Block representation of the proposed markerless visual servoing framework on unknown objects.

Bibliography

[1] Aristotle. In *De partibus animalium*, volume 687a 7, ca. 340 BC.

[2] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes Von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, et al. The iCub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125 – 1134, 2010.

[3] Richard S Sutton, Andrew G Barto, Francis Bach, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

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

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

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

[7] Shan Luo, Joao Bimbo, Ravinder Dahiya, and Hongbin Liu. Robotic tactile perception of object properties: A review. *Mechatronics*, 48: 54–67, 2017.

[8] Qiang Li, Robert Haschke, and Helge Ritter. A visuo-tactile control framework for manipulation and exploration of unknown objects. 2015.

[9] Chuanyu Yang and Nathan F Lepora. Object exploration using vision and active touch. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 6363–6370. IEEE, 2017.

[10] Kuan-Ting Yu and Alberto Rodriguez. Realtime state estimation with tactile and visual sensing application to planar manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7778–7785. IEEE, 2018.

[11] Adithyavairavan Murali, Yin Li, Dhiraj Gandhi, and Abhinav Gupta. Learning to grasp without seeing. *arXiv preprint arXiv:1805.04201*, 2018.

[12] Huaping Liu, Yuanlong Yu, Fuchun Sun, and Jason Gu. Visual–tactile fusion for object recognition. *IEEE Transactions on Automation Science and Engineering*, 14(2):996–1008, 2017.

[13] Per Jenmalm, Seth Dahlstedt, and Roland S Johansson. Visual and tactile information about object-curvature control fingertip forces and grasp kinematics in human dexterous manipulation. *Journal of Neurophysiology*, 84(6):2984 – 2997, 2000.

[14] Ardesheer Talati, Francisco J Valero-Cuevas, and Joy Hirsch. Visual and tactile guidance of dexterous manipulation tasks: an FMRI study 1, 2. *Perceptual and motor skills*, 101(1):317 – 334, 2005.

[15] Ravinder S Dahiya, Giorgio Metta, Maurizio Valle, and Giulio Sandini. Tactile sensing - from humans to humanoids. *IEEE Transactions on Robotics*, 26(1):1–20, 2010.

[16] Hanna Yousef, Mehdi Boukallel, and Kaspar Althoefer. Tactile sensing for dexterous in-hand manipulation in robotics - A review. *Sensors and Actuators A: physical*, 167(2):171–187, 2011.

[17] Mark R Cutkosky, Robert D Howe, and William R Provancher. Force and tactile sensors. In *Springer Handbook of Robotics*, pages 455–476. Springer, 2008.

[18] Alin Drimus, Gert Kootstra, Arne Bilberg, and Danica Kragic. Design of a flexible tactile sensor for classification of rigid and deformable objects. *Robotics and Autonomous Systems*, 62(1):3–15, 2014.

[19] Zhanat Kappassov, Juan-Antonio Corrales, and Véronique Perdereau. Tactile sensing in dexterous robot hands. *Robotics and Autonomous Systems*, 74:195–220, 2015.

[20] Liang Zou, Chang Ge, Z Wang, Edmond Cretu, and Xiaoou Li. Novel tactile sensor technology and smart tactile sensing systems: A review. *Sensors*, 17(11):2653, 2017.

[21] Nawid Jamali, Marco Maggiali, Francesco Giovannini, Giorgio Metta, and Lorenzo Natale. A new design of a fingertip for the iCub hand. In *28th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2705 – 2710, Hamburg, Germany, 2015.

[22] Alexander Schmitz, Perla Maiolino, Marco Maggiali, Lorenzo Natale, Giorgio Cannata, and Giorgio Metta. Methods and technologies for the implementation of large-scale robot tactile sensors. *IEEE Transactions on Robotics*, 27(3):389 – 400, 2011.

[23] Nicholas Wettels, Veronica J Santos, Roland S Johansson, and Gerald E Loeb. Biomimetic tactile sensor array. *Advanced Robotics*, 22(8): 829 – 849, 2008.

[24] Gereon H Büscher, Risto Kõiva, Carsten Schürmann, Robert Haschke, and Helge J Ritter. Flexible and stretchable fabric-based tactile sensor. *Robotics and Autonomous Systems*, 63:244–252, 2015.

[25] Micah K Johnson and Edward H Adelson. Retrographic sensing for the measurement of surface texture and shape. In *IEEE Conference onComputer Vision and Pattern Recognition (CVPR), 2009*, pages 1070–1077. IEEE, 2009.

[26] Micah K Johnson, Forrester Cole, Alvin Raj, and Edward H Adelson. Microgeometry capture using an elastomeric sensor. In *ACM Transactions on Graphics (TOG)*, volume 30, page 46. ACM, 2011.

[27] Wenzhen Yuan, Siyuan Dong, and Edward H Adelson. Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors*, 17(12):2762, 2017.

[28] PC Gaston and T Lozano-Perez. Tactile recognition and localization using object models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):257 – 265, 1983.

[29] W Eric L Grimson and Tomas Lozano-Perez. Model-based recognition and localization from sparse range or tactile data. *The International Journal of Robotics Research*, 3(3):3 – 35, 1984.

[30] Olivier D Faugeras and Martial Hebert. A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces. In *8th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 996 – 1002, Karlsruhe, Germany, 1983.

[31] Klaas Gadeyne and Herman Bruyninckx. Markov techniques for object localization with force-controlled robots. In *10th International Conference on Advanced Robotics (ICAR)*, pages 91 – 96, Budapest, Hungary, 2001.

[32] Siddharth R Chhatpar and Michael S Branicky. Particle filtering for localization in robotic assemblies with position uncertainty. In *18th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3610 – 3617, Edmonton, Canada, 2005.

[33] Craig Corcoran and Robert Platt Jr. A measurement model for tracking hand-object state during dexterous manipulation. In *27th IEEE International Conference on Robotics and Automation (ICRA)*, pages 4302 – 4308, Anchorage, Alaska, 2010.

[34] Anna Petrovskaya and Oussama Khatib. Global localization of objects via touch. *IEEE Transactions on Robotics*, 27(3):569 – 585, 2011.

[35] Anna Petrovskaya, Oussama Khatib, Sebastian Thrun, and Andrew Y Ng. Bayesian estimation for autonomous object manipulation based on tactile sensors. In *23rd IEEE International Conference on Robotics and Automation (ICRA)*, pages 707 – 714, Orlando, Florida, 2006.

[36] Maxime Chalon, Jens Reinecke, and Martin Pfanne. Online in-hand object localization. In *26th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2977 – 2984, Tokyo, Japan, 2013.

[37] Joao Bimbo, Petar Kormushev, Kaspar Althoefer, and Hongbin Liu. Global estimation of an object's pose using tactile sensing. *Advanced Robotics*, 29(5):363 – 374, 2015.

[38] Michael C Koval, Nancy S Pollard, and Siddhartha S Srinivasa. Pose estimation for planar contact manipulation with manifold particle filters. *The International Journal of Robotics Research*, 34(7):922–945, 2015.

[39] Michael C Koval, Matthew Klingensmith, Siddhartha S Srinivasa, Nancy S Pollard, and Michael Kaess. The manifold particle filter for state estimation on high-dimensional implicit manifolds. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on. IEEE*, pages 4673–4680, 2017.

[40] Giulia Vezzani, Ugo Pattacini, Giorgio Battistelli, Luigi Chisci, and Lorenzo Natale. Memory unscented particle filter for 6-DOF tactile localization. *IEEE Transactions on Robotics*, 33(5):1139–1155, 2017.

[41] Joao Bimbo, Shan Luo, Kaspar Althoefer, and Hongbin Liu. In-hand object pose estimation using covariance-based tactile to geometry matching. *IEEE Robotics and Automation Letters*, 1(1):570–577, 2016.

[42] Artem Molchanov, Oliver Kroemer, Zhe Su, and Gaurav S Sukhatme. Contact localization on grasped objects using tactile sensing. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 216–222. IEEE, 2016.

[43] Yitao Ding, Julian Bonse, Robert Andre, and Ulrike Thomas. In-hand grasping pose estimation using particle filters in combination with haptic rendering models. *International Journal of Humanoid Robotics*, 15(01):1850002, 2018.

[44] Gregory Izatt, Geronimo Mirano, Edward Adelson, and Russ Tedrake. Tracking objects with point clouds from vision and touch. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4000–4007. IEEE, 2017.

[45] Nawid Jamali and C Sammut. Majority voting: Material classification by tactile sensing using surface texture. *IEEE Transaction on Robotics*, 27(3):508 – 521, 2011.

[46] Sergio Decherchi, Paolo Gastaldo, Ravinder S. Dahiya, Maurizio Valle, and Rodolfo Zunino. Tactile-data classification of contact materials using computational intelligence. In *25th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 635 —- 639, 2012.

[47] Hongbin Liu, X. Song, J. Bimbo, L. Seneviratne, and K. Althoefer. Surface material recognition through haptic exploration using an intelligent contact sensing finger. In *25th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 52 – 57, 2012.

[48] R. S. Fearing and T. O. Binford. Using a cylindrical tactile sensor for determining curvature. In *IEEE Transaction Robotics and Automation*, volume 7, pages 806 – 817, 1991.

[49] R Andrew Russell and Simon Parkinson. Sensing surface shape by touch. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 423–428. IEEE, 1993.

[50] M Charlebois, K Gupta, and Shahram Payandeh. Shape description of curved surfaces from contact sensing using surface normals. volume 18, pages 779–787. SAGE Publications, 1999.

[51] Peter K Allen and Kenneth S Roberts. Haptic object recognition using a multi-fingered dextrous hand. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 342–347. IEEE, 1989.

[52] Peter K Allen and Paul Michelman. Acquisition and interpretation of 3-d sensor data from touch. volume 6, pages 397–404. IEEE, 1990.

[53] M Charlebois, K Gupta, and Shahram Payandeh. Shape description of general, curved surfaces using tactile sensing and surface normal information. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 2819–2824. IEEE, 1997.

[54] Giulia Vezzani, Massimo Regoli, Ugo Pattacini, and Lorenzo Natale. A novel pipeline for bi-manual handover task. *Advanced Robotics*, 31 (23-24):1267–1280, 2017.

[55] Hongbin Liu, Xiaojing Song, Thrishantha Nanayakkara, Lakmal D Seneviratne, and Kaspar Althoefer. A computationally fast algorithm for local contact shape and pose classification using a tactile array sensor. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1410–1415. IEEE, 2012.

[56] AR Jimenez, AS Soembagijo, Dominiek Reynaerts, Hendrik Van Brussel, R Ceres, and JL Pons. Featureless classification of tactile contacts in a gripper using neural networks. volume 62, pages 488–491. Elsevier, 1997.

[57] Shan Luo, Wenxuan Mou, Min Li, Kaspar Althoefer, and Hongbin Liu. Rotation and translation invariant object recognition with a tactile sensor. In *IEEE Sensors 2014*, pages 1030–1033. IEEE, 2014.

[58] Stefan Escaida Navarro, Nicolas Gorges, Heinz Wörn, Julian Schill, Tamim Asfour, and Rüdiger Dillmann. Haptic object recognition for multi-fingered robot hands. In *Haptics Symposium (HAPTICS), 2012 IEEE*, pages 497–502. IEEE, 2012.

[59] Massimo Regoli, Nawid Jamali, Giorgio Metta, and Lorenzo Natale. Controlled tactile exploration and haptic object recognition. In *Advanced Robotics (ICAR), 2017 18th International Conference on*, pages 47–54. IEEE, 2017.

[60] Huaping Liu, Di Guo, and Fuchun Sun. Object recognition using tactile measurements: Kernel sparse coding methods. *IEEE Transactions on Instrumentation and Measurement*, 65(3):656–665, 2016.

[61] Alexander Schmitz, Yusuke Bansho, Kuniaki Noda, Hiroyasu Iwata, Tetsuya Ogata, and Shigeki Sugano. Tactile object recognition using deep learning and dropout. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 1044–1050. IEEE, 2014.

[62] Shan Luo, Wenxuan Mou, Kaspar Althoefer, and Hongbin Liu. Iterative closest labeled point for tactile object shape recognition. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3137–3142. IEEE, 2016.

[63] Marianna Madry, Liefeng Bo, Danica Kragic, and Dieter Fox. St-hmp: Unsupervised spatio-temporal feature learning for tactile data. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2262–2269. IEEE, 2014.

[64] Rui Li and Edward H Adelson. Sensing and recognizing surface textures using a gelsight sensor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1241–1247, 2013.

[65] Zachary Pezzementi, Erion Plaku, Caitlin Reyda, and Gregory D Hager. Tactile-object recognition from appearance information. *IEEE Transactions on Robotics*, 27(3):473–487, 2011.

[66] Hongbin Liu, Juan Greco, Xiaojing Song, Joao Bimbo, Lakmal Seneviratne, and Kaspar Althoefer. Tactile image based contact shape recognition using neural network. In *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, pages 138–143. IEEE, 2012.

[67] Yoshihito Koga and Jean-Claude Latombe. Experiments in dual-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 2238–2245. IEEE, 1992.

[68] Yoshihito Koga and Jean-Claude Latombe. On multi-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 945–952. IEEE, 1994.

[69] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning motions with intentions. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 395–408. ACM, 1994.

[70] Kris Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*, 29(7):897–915, 2010.

[71] Andrew Dobson and Kostas E Bekris. Planning representations and algorithms for prehensile multi-arm manipulation. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6381–6386. IEEE, 2015.

[72] Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1503–1510. IEEE, 2015.

[73] Joao Silvério, Leonel Rozo, Sylvain Calinon, and Darwin G Caldwell. Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 464–470. IEEE, 2015.

[74] Benjamin Balaguer and Stefano Carpin. Bimanual regrasping from unimanual machine learning. In *IEEE International Conference on Robotics and Automation*, pages 3264–3270. IEEE, 2012.

[75] Weiwei Wan and Kensuke Harada. Developing and comparing single-arm and dual-arm regrasp. *IEEE Robotics and Automation Letters*, 1(1):243–250, 2016.

[76] Jean-Philippe Saut, Mokhtar Gharbi, Juan Cortés, Daniel Sidobre, and Thierry Siméon. Planning pick-and-place tasks with two-hand regrasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4528–4533. IEEE, 2010.

[77] G Maria Gasparri, Filippo Fabiani, Manolo Garabini, Lucia Pallottino, M Catalano, Giorgio Grioli, R Persichin, and Antonio Bicchi. Robust optimization of system compliance for physical interaction in uncertain scenarios. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 911–918. IEEE, 2016.

[78] Weiwei Wan and Kensuke Harada. Achieving high success rate in dual-arm handover using large number of candidate grasps, handover heuristics, and hierarchical search. *Advanced Robotics*, 30 (17-18):1111–1125, 2016.

[79] Kaijen Hsiao, Sachin Chitta, Matei Ciocarlie, and E Gil Jones. Contact-reactive grasping of objects with partial shape information. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1228–1235. IEEE, 2010.

[80] Ellen Klingbeil, Deepak Rao, Blake Carpenter, Varun Ganapathi, Andrew Y Ng, and Oussama Khatib. Grasping with application to an autonomous checkout robot. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2837–2844. IEEE, 2011.

[81] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

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

[83] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics (TOG)*, 3(4):266–286, 1984.

[84] Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994.

[85] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421. ACM, 1998.

[86] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

[87] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.

[88] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):29, 2013.

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

[90] G. Vezzani, U. Pattacini, and L. Natale. A grasping approach based on superquadric models. In *34th IEEE International Conference on Robotics and Automation (ICRA)*, pages 1579–1586, 2017.

[91] Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, Jonathan Kleinehellefort, and Michael Beetz. General 3d modelling of novel objects from a single view. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3700–3705. IEEE, 2010.

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

[93] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems*, pages 1696–1704, 2016.

[94] Jason Rock, Tanmay Gupta, Justin Thorsen, JunYoung Gwak, Daeyun Shin, and Derek Hoiem. Completing 3d object shape from one depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2484–2493, 2015.

[95] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. Marrnet: 3d shape reconstruction via 2.5 d sketches. In *Advances in neural information processing systems*, pages 540–550, 2017.

[96] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2017.

[97] Minhyuk Sung, Vladimir G Kim, Roland Angst, and Leonidas Guibas. Data-driven structural priors for shape completion. *ACM Transactions on Graphics (TOG)*, 34(6):175, 2015.

[98] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.

[99] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *3D Vision (3DV), 2017 International Conference on*, pages 412–420. IEEE, 2017.

[100] Alan H Barr. Superquadrics and angle-preserving transformations. *IEEE Computer graphics and Applications*, 1(1):11–23, 1981.

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

[102] Ales Jaklic, A. Leonardis, and F. Solina. Segmentation and recovery of superquadrics. *Computational imaging and vision*, 20, 2000.

[103] Andrew J Hanson. Hyperquadrics: smoothly deformable shapes with convex polyhedral bounds. *Computer vision, graphics, and image processing*, 44(2):191–210, 1988.

[104] Kester Duncan, Sudeep Sarkar, Redwan Alqasemi, and Rajiv Dubey. Multi-scale superquadric fitting for efficient shape and pose recovery of unknown objects. In *IEEE International Conference om Robotics and Automation (ICRA)*, pages 4238 – 4243, 2013.

[105] Laurent Chevalier, Fabrice Jaillet, and Atilla Baskurt. Segmentation and superquadric modeling of 3d objects. 2003.

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

[107] Georg Biegelbauer and Markus Vincze. Efficient 3d object detection by fitting superquadrics to range image data for robot's object manipulation. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1086–1091. IEEE, 2007.

[108] Ales Leonardis, Ales Jaklic, and Franc Solina. Superquadrics for segmenting and modeling range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1289–1295, 1997.

[109] Dimitrios Katsoulas, Christian Cea Bastidas, and Dimitrios Kosmopoulos. Superquadric segmentation in range images via fusion of region and boundary information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):781–795, 2008.

[110] Konstantinos Moustakas, Dimitrios Tzovaras, and Michael Gerassimos Strintzis. Sq-map: Efficient layered collision detection and haptic rendering. *IEEE Transactions on Visualization & Computer Graphics*, (1):80–93, 2007.

[111] Nilanjan Chakraborty, Jufeng Peng, Srinivas Akella, and John E. Mitchell. Proximity queries between convex objects: an interior point approach for implicit surfaces. *IEEE Transactions on Robotics*, 24(1): 211– 220, 2008.

[112] Corey Goldfeder, Peter K. Allen, Claire Lackner, and Raphael Pelossof. Grasp planning via decomposition trees. In *IEEE International Conference om Robotics and Automation (ICRA)*, pages 4679 – 4684, 2007.

[113] Tiberiu T. Cocias, Sorin M. Grigorescu, and Florin Moldoveanu. Multiple-superquadrics based object surface estimation for grasping in service robotics. In *13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, pages 1471 – 1477, 2012.

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

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

[116] PD Nguyen, Fabrizio Bottarel, Ugo Pattacini, Matej Hoffmann, Lorenzo Natale, and Giorgio Metta. Merging physical and social interaction for effective human-robot collaboration. *Humanoid Robots (Humanoids)*, 2018.

[117] Jurij Slabanja, Blaž Meden, Peter Peer, Aleš Jaklič, and Franc Solina. Segmentation and reconstruction of 3d models from a point cloud with deep neural networks. 2018.

[118] Karun B Shimoga. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230–266, 1996.

[119] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *ICRA*, volume 348, page 353. Citeseer, 2000.

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

[121] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-Driven Grasp Synthesis - A Survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2015.

[122] John R Napier. The prehensile movements of the human hand. *The Journal of bone and joint surgery. British volume*, 38(4):902–913, 1956.

[123] Ilaria Gori, Ugo Pattacini, Vadim Tikhanoff, and Giorgio Metta. Ranking the good points: A comprehensive method for humanoid robots to grasp unknown objects. In *Advanced Robotics (ICAR), 2013 16th International Conference on*, pages 1–7. IEEE, 2013.

[124] Ilaria Gori, Ugo Pattacini, Vadim Tikhanoff, and Giorgio Metta. Three-finger precision grasp on incomplete 3d point clouds. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5366–5373. IEEE, 2014.

[125] Andrew Miller and Peter K. Allen. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics and Automation Magazine*, 11(4): 110– 122, 2004.

[126] Andrew T Miller, Steffen Knoop, Henrik I Christensen, and Peter K Allen. Automatic grasp planning using shape primitives. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1824–1829. IEEE, 2003.

[127] Raphael Pelossof, Andrew Miller, Peter Allen, and Tony Jebara. An svm learning approach to robotic grasping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3512–3518. IEEE, 2004.

[128] Jonathan Weisz and Peter K Allen. Pose error robust grasping from contact wrench space metrics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 557–562. IEEE, 2012.

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

[130] Renaud Detry, Emre Baseski, Mila Popovic, Younes Touati, N Kruger, Oliver Kroemer, Jan Peters, and Justus Piater. Learning object-specific grasp affordance densities. In *Development and Learning, 2009. ICDL 2009. IEEE 8th International Conference on*, pages 1–7. IEEE, 2009.

[131] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and José Santos-Victor. Learning object affordances: from sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

[132] Antonio Morales, Eris Chinellato, Andrew H Fagg, and Angel P Del Pobil. Using experience for assessing grasp reliability. *International Journal of Humanoid Robotics*, 1(04):671–691, 2004.

[133] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[134] Ashutosh Saxena, Lawson LS Wong, and Andrew Y Ng. Learning grasp strategies with partial shape information. In *AAAI*, volume 3, pages 1491–1494, 2008.

[135] Jeannette Bohg and Danica Kragic. Learning grasping points with shape context. *Robotics and Autonomous Systems*, 58(4):362–377, 2010.

[136] Microsoft, "Kinect- xbox.com". `www.xbox.com/en-US/KINECT`.

[137] Primesense. `www.primesense.com`.

[138] Maxime Adjigble, Naresh Marturi, Valerio Ortenzi, Vijaykumar Rajasekaran, Peter Corke, and Rustam Stolkin. Model-free and learning-free grasping by local contact moment matching. In *IEEE-RSJ International Conference on Intelligent Robots and Systems, IEEE*, 2018.

[139] Gabriel Thomas Bell and Theodore Creighton Armstrong. Amazon picking challenge. 2015.

[140] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano, and Peter R Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188, 2018.

[141] Carlos Hernandez, Mukunda Bharatheesha, Jeff van Egmond, Jihong Ju, and Martijn Wisse. Integrating different levels of automation: Lessons from winning the amazon robotics challenge 2016. *IEEE Transactions on Industrial Informatics*, pages 1–11, 2018.

[142] Douglas Morrison, Adam W Tow, M McTaggart, R Smith, N Kelly-Boxall, S Wade-McCue, J Erskine, R Grinover, A Gurman, T Hunn, et al. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7757–7764. IEEE, 2018.

[143] Deepak Rao, Quoc V Le, Thanathorn Phoka, Morgan Quigley, Attawith Sudsang, and Andrew Y Ng. Grasping novel objects with depth segmentation. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2578–2585. IEEE, 2010.

[144] Jeannette Bohg, Matthew Johnson-Roberson, Beatriz León, Javier Felip, Xavi Gratal, Niklas Bergström, Danica Kragic, and Antonio

Morales. Mind the gap-robotic grasping under incomplete observation. In *IEEE International Conference on Robotics and Automation (ICRA) Location: Shanghai, Date: May 09-13, 2011*, pages 686–693. IEEE, 2011.

[145] Dirk Kraft, Nicolas Pugeault, Emre Başeski, Mila Popović, Danica Kragić, Sinan Kalkan, Florentin Wörgötter, and Norbert Krüger. Birth of the object: detection of objectness and extraction of object shape through object–action complexes. *International Journal of Humanoid Robotics*, 5(02):247–265, 2008.

[146] Antonio Morales, Pedro J Sanz, Angel P Del Pobil, and Andrew H Fagg. Vision-based three-finger grasp synthesis constrained by hand geometry. *Robotics and Autonomous Systems*, 54(6):496–512, 2006.

[147] Mario Richtsfeld and Markus Vincze. Grasping of unknown objects from a table top. In *Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments*, 2008.

[148] Philipp Schmidt, Nikolaus Vahrenkamp, Mirko Wächter, and Tamim Asfour. Grasping of unknown objects using deep convolutional neural networks based on depth images. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6831–6838. IEEE, 2018.

[149] Joseph Redmon and Anelia Angelova. Real-time grasp detection using convolutional neural networks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1316–1322. IEEE, 2015.

[150] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1957–1964. IEEE, 2016.

[151] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net

2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *Robotics: Science and Systems (RSS)*, 2017.

[152] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-net 3.0: Computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning. *arXiv preprint arXiv:1709.06670*, 2017.

[153] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[154] Josh Tobin, Lukas Biewald, Rocky Duan, Marcin Andrychowicz, Ankur Handa, Vikash Kumar, Bob McGrew, Alex Ray, Jonas Schneider, Peter Welinder, et al. Domain randomization and generative models for robotic grasping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3482–3489. IEEE, 2018.

[155] Kuan Fang, Yunfei Bai, Stefan Hinterstoisser, Silvio Savarese, and Mrinal Kalakrishnan. Multi-task domain adaptation for deep learning of instance grasping from simulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3516–3523. IEEE, 2018.

[156] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[157] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[158] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[159] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[160] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[161] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2619–2624. IEEE, 2004.

[162] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. Springer, 2006.

[163] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[164] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[165] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[166] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

[167] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[168] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

[169] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[170] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[171] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[172] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[173] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[174] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *arXiv preprint arXiv:1802.07245*, 2018.

[175] Herke Van Hoof, Tucker Hermans, Gerhard Neumann, and Jan Peters. Learning robot in-hand manipulation with tactile features. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 121–127. IEEE, 2015.

[176] Andrew Sendonaris and COM Gabriel Dulac-Arnold. Learning from demonstrations for real world reinforcement learning. *arXiv preprint arXiv:1704.03732*, 2017.

[177] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR, abs/1707.08817*, 2017.

[178] Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[179] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[180] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *Robotics: Science and Systems (RSS)*, 2018.

[181] Sean Ryan Fanello, Ugo Pattacini, Ilaria Gori, Vadim Tikhanoff, Marco Randazzo, Alessandro Roncone, Francesca Odone, and Giorgio Metta. 3D stereo estimation and fully automated learning of eye-hand coordination in humanoid robots. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 1028–1035, Madrid, Spain, 2014. IEEE.

[182] Alberto Parmiggiani, Marco Maggiali, Lorenzo Natale, Francesco Nori, Alexander Schmitz, Nikos Tsagarakis, Jose Santos Victor, Francesco Becchi, Giulio Sandini, and Giorgio Metta. The design of the icub humanoid robot. *International journal of humanoid robotics*, 9 (04):1250027, 2012.

[183] Ugo Pattacini, Francesco Nori, Lorenzo Natale, Giorgio Metta, and Giulio Sandini. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1668–1674. IEEE, 2010.

[184] Alessandro Roncone, Ugo Pattacini, Giorgio Metta, and Lorenzo Natale. A cartesian 6-dof gaze controller for humanoid robots. In *Robotics: Science and Systems*, volume 2016, 2016.

[185] The YARP cartesian interface. *description available at Cartesian Interface*.

[186] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. Yarp: yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1):8, 2006.

[187] S.Y. Chen. Kalman filter for robot vision: A survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409 – 4420, 2012. ISSN 0278-0046. doi: 10.1109/TIE.2011.2162714.

[188] Rudolph Van Der Merwe, Arnaud Doucet, Nando De Freitas, and Eric Wan. The unscented particle filter. In *Advances in Neural Information Processing Systems 13*, pages 584 – 590. 2001.

[189] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Task-driven tactile exploration. In *6th Conference on Robotics: Science and Systems*, pages 66 – 72, Zaragoza, Spain, 2010.

[190] Niccoló Tosi, Olivier David, and Herman Bruyninckx. Action selection for touch-based localisation trading off information gain and execution time. In *31st IEEE International Conference on Robotics and Automation (ICRA)*, pages 2270 – 2275, Hong Kong, China, 2014.

[191] Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. ISSN 0018-9219. doi: 10.1109/JPROC.2003.823141.

[192] Simon J Julier and Jeffrey K Uhlmann. New extension of the Kalman filter to nonlinear systems. In *AeroSense '97*, pages 182 – 193, Orlando, Florida, 1997.

[193] Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential Monte Carlo methods. In Springer, editor, *Sequential Monte Carlo methods in practice*, pages 3 – 14. 2001.

[194] Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. ISSN 0018 – 9219. doi: 10.1109/JPROC.2003.823141.

[195] Jane Liu and Mike West. *Combined Parameter and State Estimation in Simulation-Based Filtering*, pages 197–223. Springer New York, New York, NY, 2001.

[196] Saikat Saha, Yvo Boers, Hans Driessen, Pranab Kumar Mandal, and Arunabha Bagchi. Particle based MAP state estimation: A comparison. In *12th International Conference on Information Fusion (FUSION)*, pages 278 – 283, Seattle, USA, 2009.

[197] Johann Prankl, Aitor Aldoma, Alexander Svejda, and Markus Vincze. RGB-D object modelling for object recognition and tracking. In *28th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 96 – 103, Hamburg, Germany, 2015.

[198] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *4th Eurographics Symposium on Geometry Processing (SGP)*, volume 7, pages 61 – 70, Cagliari, Italy, 2006.

[199] Yang Chen and Gerard Medion. Object modelling by registration of multiple range imagesy. *Image Vision Computing*, pages 145 – 155, 1991.

[200] Giulia Vezzani, Nawid Jamali, Ugo Pattacini, Giorgio Battistelli, Luigi Chisci, and Lorenzo Natale. A novel Bayesian filtering approach to tactile object recognition. In *16th IEEE International Conference on Humanoid Robotics*, pages 256 – 263, Cancun, Mexico, 2016.

[201] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The iCub humanoid robot: an open platform for research in embodied cognition. *Proc. 8th Work. Perform. Metrics Intell. Syst.*, pages 50–56, 2008. ISSN 00426989. doi: http://dx.doi.org/10.1145/1774674.1774683.

[202] Massimo Regoli, Ugo Pattacini, Giorgio Metta, and Lorenzo Natale. Hierarchical grasp controller using tactile feedback. In *IEEE-RAS 16th International Conference on Humanoid Robots*, pages 387–394. IEEE, 2016.

[203] Alessandro Roncone, Matej Hoffmann, Ugo Pattacini, and Giorgio Metta. Automatic kinematic chain calibration using artificial skin: self-touch in the icub humanoid robot. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2305–2312. IEEE, 2014.

[204] Sean Ryan Fanello, Ugo Pattacini, Ilaria Gori, Vadim Tikhanoff, Marco Randazzo, Alessandro Roncone, Francesca Odone, and Giorgio Metta. 3d stereo estimation and fully automated learning of eye-hand coordination in humanoid robots. In *14th IEEE-RAS International Conference on Humanoid Robots*, pages 1028–1035. IEEE, 2014.

[205] Nikolaus Vahrenkamp, Manfred Kröhnert, Stefan Ulbrich, Tamim Asfour, Giorgio Metta, Rüdiger Dillmann, and Giulio Sandini. Simox: A Robotics Toolbox for Simulation, Motion and Grasp Planning. In *International Conference on Intelligent Autonomous Systems (IAS)*, pages 585–594, 2012.

[206] Jacques Denavit. A kinematic notation for lower-pair mechanisms based on matrices. *Transactions of the ASME. Journal of Applied Mechanics*, 22:215 – 221, 1955.

[207] Andreas Wächter and Lorenz-T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 2006. ISSN 0025-5610.

[208] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. In *preprint available athttps://arxiv.org/abs/1502.03143*, 2015.

[209] Claudio Fantacci, Ugo Pattacini, Vadim Tikhanoff, and Lorenzo Natale. Visual end-effector tracking using a 3d model-aided particle filter for humanoid robot platforms. In *Intelligent Robots and Systems*

(IROS), 2017 IEEE/RSJ International Conference on, pages 1411–1418. IEEE, 2017.

[210] Giulia Pasquale, Carlo Ciliberto, Lorenzo Rosasco, and Lorenzo Natale. Object identification from few examples by improving the invariance of a deep convolutional neural network. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4904–4911. IEEE, 2016.

[211] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[212] Giulia Pasquale, Carlo Ciliberto, Francesca Odone, Lorenzo Rosasco, and Lorenzo Natale. Teaching icub to recognize objects using deep convolutional neural networks. volume 43, pages 21–25, 2015.

[213] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.

[214] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM transactions on graphics (TOG)*, volume 23, pages 309–314. ACM, 2004.

[215] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[216] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015. ISSN 1573-1405.

[217] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[218] Lorenzo Sciavicco and Bruno Siciliano. *Modelling and control of robot manipulators*. Springer Science & Business Media, 2012.

[219] Massimo Regoli, Ugo Pattacini, Giorgio Metta, and Lorenzo Natale. Hierarchical grasp controller using tactile feedback. In *IEEE-RAS 16th International Conference on Humanoid Robots*, pages 387–394. IEEE, 2016.

[220] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise.

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

[222] Claudio Fantacci, Giulia Vezzani, Ugo Pattacini, Vadim Tikhanoff, and Lorenzo Natale. Markerless visual servoing on unknown objects for humanoid robot platforms. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3099–3106. IEEE, 2018.

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

[224] Mia Kokic, Johannes A Stork, Joshua A Haustein, and Danica Kragic. Affordance detection for task-specific grasping using deep learning. In *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*, pages 91–98. IEEE, 2017.

[225] Thanh-Toan Do, Anh Nguyen, and Ian Reid. Affordancenet: An end-to-end deep learning approach for object affordance detection. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5. IEEE, 2018.

[226] Igor Mordatch, Zoran Popović, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In *ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144. Eurographics Association, 2012.

[227] Vikash Kumar, Yuval Tassa, Tom Erez, and Emanuel Todorov. Real-time behaviour synthesis for dynamic hand-manipulation. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6808–6815. IEEE, 2014.

[228] Herke Van Hoof, Tucker Hermans, Gerhard Neumann, and Jan Peters. Learning robot in-hand manipulation with tactile features. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 121–127. IEEE, 2015.

[229] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*, 2017.

[230] S. Kakade. A natural policy gradient. In *NIPS*, 2001.

[231] Jan Peters. Machine learning of motor skills for robotics. *PhD Dissertation, University of Southern California*, 2007.

[232] Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6550–6561, 2017.

[233] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[234] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[235] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[236] J. Andrew Bagnell and Jeff G. Schneider. Covariant policy search. In *IJCAI*, 2003.

[237] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[238] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[239] Dean Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *NIPS 1988]*, pages 305–313, 1988.

[240] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL `http://arxiv.org/abs/1604.07316`.

[241] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008. doi: 10.1016/j.neunet.2008.02.003. URL `https://doi.org/10.1016/j.neunet.2008.02.003`.

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

[243] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[244] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. 2017.

[245] Vikash Kumar, Zhe Xu, and Emanuel Todorov. Fast, strong and compliant pneumatic actuation for dexterous tendon-driven hands. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1512–1519. IEEE, 2013.

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

[247] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[248] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4397–4404. IEEE, 2015.

[249] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL `http://arxiv.org/abs/1509.02971`.

[250] Matej Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR*, abs/1707.08817, 2017. URL `http://arxiv.org/abs/1707.08817`.

[251] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. pages 6292–6299, 2018.

[252] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL `http://arxiv.org/abs/1709.06560`.

[253] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*, 2017.

[254] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *arXiv preprint arxiv.1605.09674*, 2016.

[255] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017.

[256] Justin Fu, John Co-Reyes, and Sergey Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2577–2587, 2017.

[257] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[258] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[259] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214*, 2018.

[260] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[262] Clément Moulin-Frier, Tobias Fischer, Maxime Petit, Grégoire Pointeau, Jordi-Ysard Puigbo, Ugo Pattacini, Sock Ching Low, Daniel Camilleri, Phuong D. H. Nguyen, Matej Hoffmann, Hyung Jin Chang, Martina Zambelli, Anne-Laure Mealier, Andreas C. Damianou, Giorgio Metta, Tony J. Prescott, Yiannis Demiris,

Peter Ford Dominey, and Paul F. M. J. Verschure. Dac-h3: A proactive robot cognitive architecture to acquire and express knowledge about the world and the self. *IEEE Transactions on Cognitive and Developmental Systems*, 2017. URL `http://arxiv.org/abs/1706.03661`.