POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Master of science in Computer Science and Engineering

Enhancing Visual Competencies for Tool Affordances in a Humanoid Robot

Supervisor:	Prof. Marcello Restelli
Co-supervisor:	Dr. Vadim Tikhanoff
Tutor:	Dr. Alexandre Gomes Pereira Antunes

Master Thesis dissertation of:
Mattia Sanchioni Matr. 918065

Academic year 2019-2020

Acknowledgements

Al termine di questo percorso, non posso non ringraziare alcune persone che mi hanno permesso di arrivare qui questo giorno.

In primo luogo, un grazie sincero va al Professor Restelli, per la disponibilità e la fiducia datami durante questo percorso di tesi.

Un'altra persona senza la quale questo percorso non sarebbe mai iniziato è Vadim. Sei stato per me una guida sicura con i tuoi consigli e le conoscenze che mi hai trasmesso durante questi mesi. Inoltre, voglio ringraziare Alex per avermi supportato ed aiutato nel portare a compimento questa tesi.

Un grazie va anche a tutti i miei amici, quelli di una vita e quelli conosciuti negli ultimi anni a Milano. Grazie per tutte le risate fatte insieme, per i momenti di gioia ma anche di difficoltà, la vostra è stata una presenza sempre costante.

Infine, il grazie più importante va alla mia famiglia, Cri, Chiara, Lele, Mamma e Papà. Voi avete creduto in me si dal giorno zero e ancora oggi continuate ad essere la certezza indissolubile della mia vita.

Abstract

Robots have become an increasingly common presence in our daily life, due to the recent leaps in technology. An important goal of robotics is to endow robots with the ability to operate autonomously and to cooperate with humans. An essential prerequisite to achieve this target is the ability to interact with objects present in the environment. In order to do that, a robot has to be able to recognize and manipulate them.

This work presents a computer vision architecture, which interacts with a pre-existing tool affordances system, in order to provide the robot with the ability to observe a scene, recognize and classify tools through a 2D image segmentation algorithm. Furthermore, by means of an RGB-D camera, the 3D model of the objects is reconstructed which allows to extract various features such as shape, pose and dimensions. The algorithm is also able to find the best part to grasp, select it as the handle, and calculate its position. The extracted features are sent to the affordances system which returns the most suitable tool for the chosen action. This object, thanks to the 3D position of its handle, can be grasped autonomously and the required action can be performed.

The architecture was validated and its effectiveness was confirmed by the results achieved both in the simulator and on the real robot. The analysis encompasses the matches obtained during the computation of the 3D model and the correct identification of the handle in different positions and orientations of the tools. The architecture developed and the results reported in this thesis show

that enhancing visual competencies endows the robot with the ability to perform actions in a more generic and autonomous way and to adapt to increasingly realistic environments.

Sommario

I robot sono diventati una presenza sempre più comune nella nostra vista quotidiana, grazie ai progressi compiuti dalla tecnologia. Un obiettivo importante della robotica è dotare i robot della capacità di operare in modo autonomo e di cooperare con gli esseri umani. Un prerequisito essenziale per raggiungere questo obiettivo è la capacità di interagire con gli oggetti contenuti nell'ambiente. Per fare ciò un robot deve essere in grado di riconoscerli e manipolarli.

Questo lavoro presenta un'architettura di computer vision, che interagisce con un sistema preesistente di affordances, al fine di fornire al robot la capacità di osservare una scena, riconoscere e classificare gli strumenti attraverso un algoritmo di segmentazione di immagini 2D. Inoltre, tramite una telecamera RGB-D, viene ricostruito il modello 3D degli oggetti che permette di estrarre varie caratteristiche, come forma, posa e dimensioni. L'algoritmo è anche in grado di trovare la parte migliore da afferrare, selezionarla come manico e calcolarne la posizione. Le caratteristiche estratte vengono inviate al sistema di affordances che restituisce lo strumento più adatto per l'azione desiderata. Questo oggetto, grazie alla posizione 3D del suo manico, può essere afferrato in modo autonomo e l'azione richiesta può essere eseguita.

L'architettura è stata validata e la sua efficacia è stata confermata dai risultati conseguiti sia nel simulatore che sul robot reale. Le analisi riguardano le corrispondenze ottenute durante il calcolo del modello 3D e la corretta identificazione del manico nelle diverse posizioni e orientamenti degli strumenti. L'architettura

sviluppata e i risultati riportati in questa tesi dimostrano che il potenziamento delle competenze visive conferisce al robot la capacità di compiere azioni in modo più generico e autonomo, e di adattarsi ad ambienti sempre più realistici.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Robots are programmable machines able to perform tasks autonomously or semi-autonomously. They have been designed and developed with several goals, entertainment, research, and also helping humans to perform some tasks, usually the most dangerous and repetitive. Indeed, robots have been placed in settings where their help would reduce the production time, e.g. the production chains in automotive industries. Robots also entered in our houses with service robots that help humans in everyday tasks. In a realistic environment robots should be able to improvise and adapt as people do. In order to do that, robots cannot be programmed for each single scenario but they have to be provided with capabilities to perform in all situations. They have to manage and manipulate tools available at runtime, and for which they were not developed. This thesis proposes an approach with which robots would be able to manage a wide variety of tools and understand how to use them for a task. The goal of this work is to provide not only the theoretical studies, but also to test them on a real humanoid robot.

Before going more in-depth, let us take a step back in the world of robotics. The noun robot was coined in 1920 by the writer Karel Capek in his science fiction drama R.U.R (Rossumovi Univerzální Roboti, Rossum's Universal Robots in English). The word robot derives from the Czech word "robota" that means work. In the play, robots have the same shape as humans and they are also made of organic material, differently from ours that are mechanic. As usually happens in science fiction literary works or films, the end is tragic, robots take over the world and try to replace humans. But already in the vision of Capek, robots are meant to help humanity with physical work. Even nowadays, robots are designed to help with physically demanding or repetitive and tedious tasks, both in simple or complex scenarios. With the same philosophy, this thesis has the aim to equip robots with the ability to perform some human tasks. In order to complete them, they have to be able to recognize and manipulate objects present in everyday scenarios as humans do. The knowledge of objects cannot be limited to restricted sets, but rather based on some properties, intrinsic to the object, with which the objects can be classified.

The technological progress in the last decade, especially in computer vision, guides the development of these capabilities. Nowadays it is possible to reconstruct the 3D model of an object thanks to depth cameras that are able to reconstruct their volumes. The most recent algorithms in deep learning and artificial intelligence are able to recognize objects and to identify the parts that compose them. This is a fundamental step in order to give the robot the ability to distinguish between objects and to allow it to grasp the desired item in order to perform the task. It is easy to imagine that in a real world scenario the input variables are substantial and the amount of possible combinations of values is immense. Objects can have very complex shapes which makes it harder for the robot to grasp. Moreover the object material can increase the complexity of choosing where and how to grasp it. It can be very different grabbing a solid object, like a stone or a wooden stick, from a softer one, such as a sponge. Some objects can even be flexible like a rope or a rubber. All these different aspects are managed daily by us with extreme simplicity. On the other hand some variables are affected by the robot itself, for example the dimension of its hand can influence the range of graspable objects. Also the weight of the object can modify the outcome of an action, since different robots are developed to handle different weights.

This thesis is not limited to a theoretical study but it tries to tackle some real aspects such as those previously cited. From a practical point of view, we tried to overcome some limitations, testing the entire project with all the algorithms beforehand, in a virtual environment and subsequently on a real robot. The simulator is very innovative and it tries to be as close as possible to reality taking into consideration all the properties of the objects, such as friction and weight, and also real capabilities of the robot.

As previously mentioned, robots should to capable of adapting themselves to each scenario. For that reason researchers, from the beginning, tried to develop new algorithms able to learn and modify the robot behaviour depending on the new challenges or tasks. In the beginning, the field of robotic learning was focused on teaching robots specific tasks and improving their skills based on observations. Aboaf and his team [1] were able to program in 1988 a ball-throwing robot system. The ballistic, kinematic and dynamic models of the robot were never improved, whereas the task performance was. The desired outputs were different from the actual ones even if the task was completed correctly. But this difference and the improvement based only on the experience indicated that learning could have continued at task level. The following year, the same team led by Aboaf continued the studies about the task-level robot learning implementing new models for a different scenario [2]. In that case the task had been more difficult than the previous one, since it was a dynamic task, juggling a tennis ball. The algorithms were extended so that they could be applied to a multi-dimensional scenario, and also take into account some of the state variables of system.

Other studies moved beyond the single task model by making robots capable of adapting to dynamic environments and acquiring new skills according to the preferences of the users. Non-expert users would still have to be able to program new robot tasks in as simple a way as possible. For that reason the Programming by Demonstration (PbD) [3] approach took hold, where just by showing the robot the task, it was able to imitate the user's movements instead of programming it through machine commands. This approach, called play-back method, consists in moving the robot through several configurations, relevant for the task completion, where the robot would move from one to the next in a sequential manner. The configuration is composed by the position, orientation and if available the state of the gripper or another end-effector. The PbD approach is often used in environments where tasks are repetitive, such as industry or assembly lines where components are always managed in the same way [4]. Still in industry, in a coworker scenario this approach can be very effective [5]. The PbD approach can also be used in mobile robot programming, teaching the robot the desired path by manually guiding it, enabling the robot to navigate autonomously [6].

Another paradigm for enabling robots to autonomously perform new tasks is learning from demonstration (LfD) [7]. Also in this case the end-user has to move the robot through different configurations, but should also provide more demonstrations rather than a single one. Indeed, LfD is not a record and replay technique, instead, it performs a task by generalizing from observing several demonstrations. When failures occur the user has only to provide more examples, so the robot becomes able to overcome the issues and manage the differences in different scenarios. An example was provided by Ekval et al [8] whose aim was to use a task planner to reach the goal given different initial states of the environment. In that specific case it was necessary to pay attention to some specific tasks that required some constrains that must have been fulfilled. In order to identify the constraints the work proposed two different approaches. In the first case, the teacher could directly instruct the robot about the underlying constraints. In the second case, the constraints were identified by the system performing multiple observations.

A different method was proposed by Leon and his team [9]. As early as 2011 they had guessed that robots would take hold in human activities, but in order to do that, robots had to be able to adapt themselves to the users' needs. Particularly, for robots to be so efficient, a non-expert user should have the possibility and the ability to teach them how to perform new tasks in natural ways. Robots can be guided in completing a task not only by the imitation of an operator but also listening to the human's voice, interpreting the speech as behaviors that are required to be reproduced. This approach was adopted by Rybski [10] that highlighted the utility of that model when the context has to be dealt with depending on the situation, conditioned by contingencies. In addition, the execution of a task can be influenced by the location where it takes place and the people present. Other learning techniques have been developed and combined to increase the ability of the robot to learn new skills. Kormushev [11] combined learning from demonstration and and reinforcement learning where the robot fine-tunes the movement learned previously by imitation. Asfour et al. [12], mixed several techniques, allowing robots to enter human-centred environments and to perform daily tasks like loading dishes into a dishwasher.

In the previous examples the focus has been on the goal achievement and the ability to learn new tasks. However, in order to build a robot as able to adapt as humans, it has to be provided with the ability to grab and manipulate objects. This capability allows robots to complete assignments in multiple situations, such as agriculture, industrial and households. Moreover, it improves the interaction and cooperation with workers or other robots. Grasping an object is not a simple task, the action is influenced by the object constraints, the shape, the size and the weight, but also the material of the object can influence the maximum force that can be applied, for example grasping a glass is not the same as grasping a hammer. Moreover the position of the object with respect to the environment and the robot can affect the possibility to be grasped or not.

The idea to build a robot that can operate at the same level of dexterity as humans in grasping objects remains an unsolved problem. Several proposals tackle the problem with different approaches, for example model-based or task-based approach achieving reasonable results. For instance Cutkosky [13] has focused on tasks performed in small-batch machining operations and job-shops. Along the same line Bormann and his team [14] developed a mobile robot able to pick items in retail store or warehouse. Both works are centered on the action to perform even if the objects handled can be very different one from the other. Cini and her team conducted a study [15] analyzing how a robot has to grasp an object and hand placement during a handover. The final configuration of the hand and the object is very important since they may influence the possibility of the second robot or human to pick the object and perform its task. Du et al [16] provided a survey in which they studied the grasp estimation on a vision based robot. This task can be executed only after localizing the object and estimating its pose, relying on a rgb-d camera to complete the task.

1.1 Motivations and goal

All examples provided in this chapter furnish the awareness that a robot, to be more flexible and be able to enter in human environment, has to be capable of generalising the learned skills and manipulating objects for which it has not been programmed. The achievement of this goal has defined the motivations for this thesis. A more lifelike natural visual system endows a humanoid robot with the ability to complete the tasks through the recognition and manipulation of the available tools. The main hypothesis behind this was originated by observing the shortcomings of the visual analysis of the scene in which the robot acts to discover objects able to help it for achieving its goals. In the experiments, especially with humanoid robots, the tool is pre-positioned in the hand of the robot or the robot knows its position in advance. The goal of this thesis is to build a system, generic enough, to allow the humanoid robot iCub to extract information from the scene that helps the recognition and manipulation of the tools present in the environment.

1.2 Contributions

The potential impact of this research concerns humanoid robots that take advantage of affordances to analyze the effects of their actions given a target object and providing a tool to use during the action. The development of a system able to improve visual abilities of humanoid robots mixing a 3D perception of the objects with a 2D image analysis will provide the ability to detect tools from the environment, retrieve their properties in terms of shape, dimensions, three-dimensional pose and graspability. Robots and agents equipped with this system will be able to extract better information from the surroundings and to analyze the 3D shape of the identified tools. This feature makes the robot capable of identifying the handle of the tool and grasping it autonomously. Current agent models and robotic approaches have limitations, they usually use only 2D image analysis to extract information related to the object. The potential technological and practical advantages of this research, such as systems that are able to acquire more data autonomously act upon it, are numerous for the fields of robotics, artificial intelligence, and cognitive systems design.

This research hands over theoretical as well as technological contributions. Starting with theoretical contributions, the research is the first to use and expand the current affordance learning pipeline for the iCub robot. The extension consists of complementing the learning algorithms providing more visual features used to compute affordances, moreover the extensive analysis of the graspability allows to predict the final pose of the tool and its end effector. A further important theoretical contribution is the new proposed architecture for humanoid robots that integrates 3D vision with model segmentation, point clouds and superquadrics that provide a method to detect the part of the tool that best fits as handle and its position in the space allowing to autonomously grasp the tool. From the technological point of view, the research provides the robotic community with an open source repository and a docker image containing the implementation of the above described system able to be executed in the real iCub robot and its simulator.

In details the system introduces a 2D image segmentation that is able to recognize the tools present in the environment.
Each tool is analyzed in terms of shape and 3D pose by reconstructing its three-dimensional model using point clouds.
In addition, by segmenting the reconstructed model it is possible to study the graspability of each part discovering the one that best fits the robot's hand.
All data gathered by the analyses so far conducted can be collected in a set of both 2D and 3D functionalities that helps and improves computation of the affordances, that are later used to find out which tool is the most suitable for the selected action.
Thanks to the reconstructed model and the graspability analysis, the 3D position of the handle is known and the selected tool can be grasped by the robot.

1.3 Thesis structure

The present thesis is structured as follows:

Chapter 2 contains a detailed description of the background concepts and knowledges that are widely used in this work. It introduces the computer vision field focusing on the techniques used, such as the pointcloud or the superquadrics. It covers also the algorithms used in the code and concludes with a description of the robot used for the experiments.

Chapter 3 collects the description and analysis of the most relevant state-of-the-art. It focuses on the topics that characterizes this thesis, such as the studies on affordances, vision-based robots, object manipulation and grasping. These studies represent the starting point of this work.

Chapter 4 contains a detailed description of the system we developed, following the iterative process we adopted. First of all we prepared the dataset of tool models, then we find each object segmenting the scene. Every tool is analyzed and divided in parts, in order to find the best graspable area. Evaluating the affordances, the best tool is chosen given the desired action.

Chapter 5 presents the experiments performed in the simulator and on the humanoid robot iCub. Moreover it provides an evaluation of the obtained results.

Chapter 6 draws the conclusion of this thesis work and proposes some possible future developments on the remaining issues.

Chapter 2

Historical context

The purpose of this section is to provide the reader with some concepts and tools that will serve as background knowledge for future reasoning in this thesis work. Firstly we will introduce the *concept of affordances*, what that term means in psychology and robotics fields. Afterwards an overview of *computer vision* is provided, going into details on some techniques and algorithms used in this project. Finally a description of the humanoid robot *iCub*, built by Istituto Italiano di Tecnologia (IIT) and an overview of the *Yet Another Robot Platform* (YARP) framework used for the robot control system is provided.

2.1 Affordances

"The affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill. The verb to afford is found in the dictionary, the noun affordance is not. I have made it up. I mean by it something that refers to both the environment and the animal in a way that no existing term does. It implies the complementarity of the animal and the environment." [17]

The term affordance was coined by the psychologist James J. Gibson in his book *The Senses Considered as Perceptual Systems*[18] and was carried on and developed in other books, such as [19] and essays, culminating in his final book *The Ecological Approach to Visual Perception*[17], from where the quote comes. Gibson was one of the most important contributors to the field of visual perception and he promoted ecological psychology, in which the environment and animals are not separable, one cannot survive without the other. Indeed, with the noun "affordance", Gibson wanted to pertain to a specific scenario in which the environment and the animal can work together.

The key point of affordances is that they are relational and characterize the suitability of the environment to the observer, and so, they depend on their current intentions and their capabilities. For this reason the objects in the environment can afford many different behaviours. The ability to predict the interaction between a given object and the body, evaluating the possible outcomes of actions before executing it, is considered the hallmark of cognition. In the brain, this ability is thought to happen through the activation of appropriate sensorimotor schemas[20]. The psychological construct that represents a sensorimotor schema brings together the perceptions and associated actions involved in the habitual behaviours. All the experiences, inferred from these behaviours, are generalized and feed the essence of that schema.

The sensorimotor schema is considered the main unit of knowledge in use during infancy. Since early age, humans start searching and using external objects in order to achieve their goals, learning from the interaction between an action and the resulting effect. The sensorimotor stage was outlined by Piaget in his books on sensorimotor development [21, 22]. It is the earliest of a series of four different stages of cognitive development. Infants gain a basic understanding of the world around them through a trial-and-error approach, and the first stage is called 'sensorimotor' stage exactly because it is through the senses and motor abilities that infants make this discovery. Sensorimotor schemas are considered as a functional explanation of the brain direct perception of object properties and their role in action execution[23].

Affordances are sensorimotor schemas associated to entities in a specific environment, and tool use is a direct consequence of the ability to understand affordances. Neuroscience thus suggests to comprehend the concept of objecthood, involving sensorimotor schemas and experiences, and associating them with symbolic representations. Robotics have followed a similar approach. Indeed, Kruger has formalized in his work [24] an entity that is the basis of symbolic representations of sensory–motor experience. All learnings and refinements are built on the interaction between an agent and the world. Also in other studies, for instance the one conducted by Montesano and his team [25] affordances encode relationships between actions, objects and effects. Indeed, affordances are used to predict the course of a particular action and thus they constitute one of the important ingredients of artificial cognition.

2.2 Computer Vision

An important aspect in robotics development is robotic vision. This field is composed by different sub-fields, like computer vision and image processing, but also incorporates aspects of robots such as their kinematics. An example of robotic vision is to control the motion of a robot by using the feedback of the robot's position as detected by a visions sensor. Indeed the main goal is to generate physical actions starting from images manipulated by image processing and analyzed thanks to computer vision.

2.2.1 3D images

The ability for a robot to move freely inside an environment and interact with objects present in it is essential to place a robot in a real setting. Computer vision algorithms are able to detect objects contained into an image, useful for a static environment or in objects detection scenario. Also in the case of mobile robots, which need to navigate in the real world, avoiding obstacles and interacting with objects, it is fundamental to analyze 3D projections of the environment.

There are several techniques to obtain depth data from a scene, one of them with stereo vision. In a traditional version, two cameras are used and placed near each other, obtaining two images of the same scene, similar to human binocular vision [26]. The reconstruction of the world seen through stereo cameras [27] can be divided into two steps: *fusion* and *reconstruction*. The *fusion* problem consists in finding for every point in one image the correspondent point in the other and compute the disparity of these points. This disparity map is used in the *reconstruction* with the focal distance of the cameras and the geometry of the setting, the position and orientation of the cameras, to compute the 3D coordinates (x, y, z) of all points. Stereo vision is widely used in several projects and studies, such as in navigation for mobile robot [28], in obstacle avoidance [29] or for tracking the robot manipulator and its end effector [30].

$$disparity = x_2 - x_1 = \frac{bf}{Z} \tag{2.1}$$

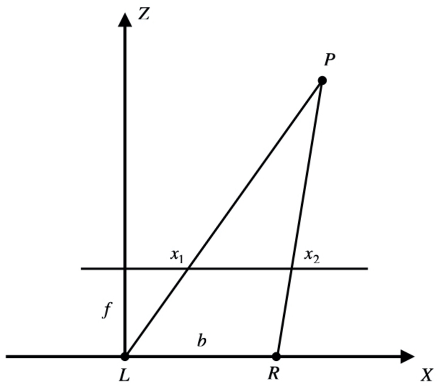
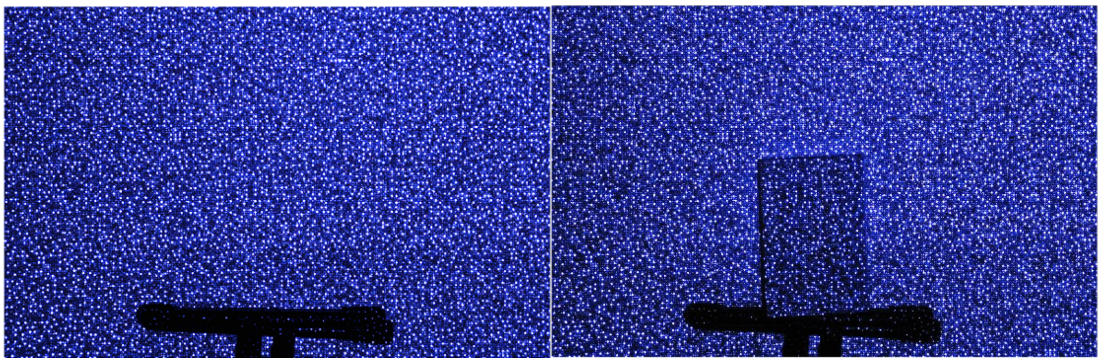


Figure 2.1: Stereo vision geometry, from which it is possible to obtain the disparity equation (2.1).

Another technique used to obtain a depth image is to use a sensor called

RGB-D camera [31]. In order to reconstruct the depth of the scene, the camera uses an infrared projector that emits a structured light with a known pattern of speckles. This grid is acquired by the infrared camera and is correlated with the reference pattern, figure 2.2. The reference pattern is generated during a calibration phase in which a depth plane at a known distance from the sensor is acquired. When an object is positioned closer or further away from the sensor, the projection of its speckle is shifted in a certain direction. Measuring this shift for all the speckles, it is possible to generate a disparity map. For each pixel it is possible to compute the distance to the sensor thanks to the disparity map.



(a) (b)

Figure 2.2: Speckles pattern in rgb-d infrared camera. In the left image it is possible to see the speckled grid of the default scenario. Introducing an object some speckles are shifted, as in the right image, allowing the measurement of the distance to that object.

A very famous product that mounts that sensor is the Microsoft Kinect ® [32] that has been widely used for different purposes, for instance in mapping problem for mobile robot [33] or in object recognition [34].

The second version of the same product is equipped with a different sensor for the depth measurement, a time-of-flight camera [35]. This sensor works by illuminating the environment with a modulated light and observing the reflected one. The shift between the phase of illumination and the reflection is measured and transformed into distance. Figure 2.3 illustrates the basic concept of a time-of-flight camera.

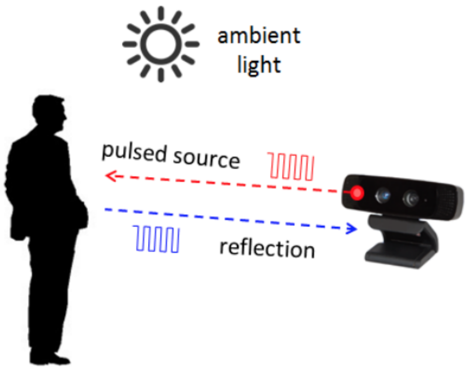


Figure 2.3: Basic concept of a time-of-flight camera.

The illumination is usually generated by a laser or a LED, operating near-infrared range (850nm), invisible to human eyes. The light source illuminates for a short time (Δt), and the reflected beam is sampled with the same Δt. Accumulated charges in the pixel during the samples, Q_1 and Q_2, are measured and used to compute the distance through the formula 2.2. The depth map is computed for each pixel in parallel.

$$d = \frac{1}{2}\, c\, \Delta t \left(\frac{Q_2}{Q_1 + Q_2} \right) \tag{2.2}$$

2.2.2 Point clouds

A depth image, as seen until now, is a collection of 3D points, one for each pixel. These points can be called also voxel, the counterpart of the 2D pixel in a 3D environment, so it represents a parallelepiped with the base as the pixel and the height as the distance. An alternative way to represent a depth map can be representing each points in a three-dimensional space, this collection of points is usually called pointcloud, as we can see in figure 2.4.

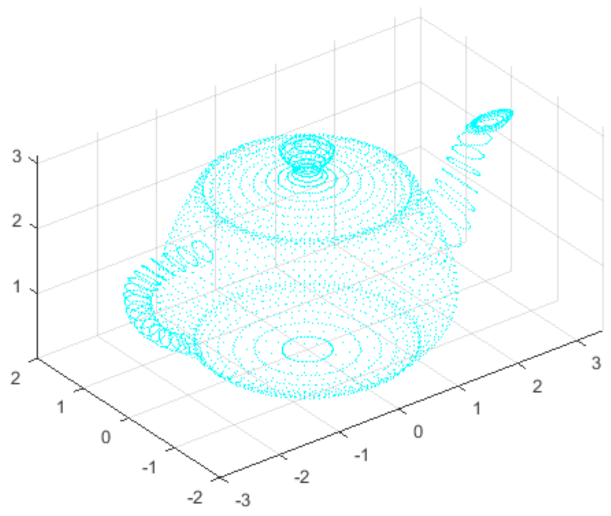


Figure 2.4: An example of point cloud of a tea pot.

Point clouds are used for many purposes, for instance the creation of 3D CAD models. Moreover point clouds are often converted to triangle mesh models or CAD models through surface reconstruction algorithms. There are many techniques for converting a point cloud to a 3D model. Some approaches are explained in paragraph 2.3.1, in general the main goal is to build a series of connected triangles over the vertices that characterize the point cloud. The key limitation during acquisition of point clouds is that only accessible and visible surfaces can be scanned. This means that simply to cover all scanning positions takes time. But at the same time point clouds are a non-intrusive way to measure buildings or objects because no sensors have to be introduced into the environment. Point clouds have become abundantly used in 3D environment, for example in making digital elevation models of the terrain [36]. They are also used to reconstruct the three-dimensional model of a construction, the technique mostly used for that purpose is the photogrammetry [37], a drone takes numerous pictures of the building and a dedicated process overlaps all the images reconstructing the model.

2.2.3 Superquadrics

With the point clouds we are able to scan and represent a three-dimensional model. But in order to manipulate it and study how it interacts with other objects, a compact volumetric model is necessary for shape representation. These models are called superquadrics [38], that extend the classical quadric surfaces and solids and produce a set of useful forms with rounded edges and filleted faces. With these primitives also related operators have been designed, so parametric deformations can be applied, bending, tampering or cavity deformation [39]. The advantage in using superquadrics is to simplify the construction and alteration of complex solids and surfaces using few interactive parameters.

Superquadric can be expressed by the implicit equation 2.3 where parameters a_1, a_2 and a_3 define the size of the superquadric in respectively x, y and z coordinates; instead ϵ_i parameters control the shape of the superquadric, indeed ϵ_1 and ϵ_2 are the squareness parameter respectively in the latitude plane and in the longitude plane.

$$\left(\left(\frac{x}{a_1}\right)^{2/\epsilon_2} + \left(\frac{y}{a_2}\right)^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1} + \left(\frac{z}{a_3}\right)^{2/\epsilon_1} = 1 \tag{2.3}$$

From the equation 2.3 we can obtain a parametric function 2.4, called *inside-outside function* because it provides a way to tell if a point $[x, y, z]^T$ lies relative to the superquadric surface. If $F(x, y, z) < 1$ the point is inside the superquadric. If $F(x, y, z) = 1$ the point is on the surface of the superquadric. If $F(x, y, z) > 1$ the corresponding point lies outside the surface.

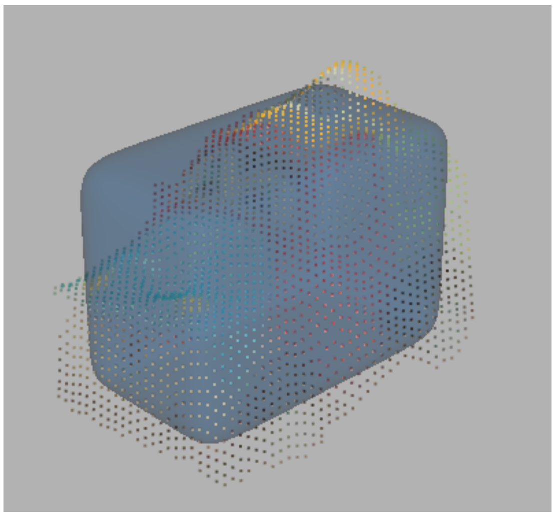
$$F(x,y,z) = \left(\left(\left(\frac{x}{a_1}\right)^{2/\epsilon_2} + \left(\frac{y}{a_2}\right)^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1} + \left(\frac{z}{a_3}\right)^{2/\epsilon_1}\right)^{\epsilon_1} \tag{2.4}$$

It is possible to expand the inside-outside function 2.4 defining a superquadric in general position and orientation in space [40]. Three extra parameters p_x, p_y and p_z are used to express the position of its central point in world coordinate

system. Moreover, three additional parameters θ, ϕ and ψ for the ZYZ Euler angles that define the orientation of the superquadric by three rotations: a rotation of θ around the z axis, followed by a rotation of ϕ around the y axis and a final rotation of ψ around the z axis.
Now the inside-outside function 2.5 has 11 parameters (defined as Λ) and it is able to define a vast variety of models because the function can describe the size and the shape of the object and also the orientation and the position in the space.

$$F(x,y,z) = F(x,y,z,\Lambda) = F(x,y,z,a_1,a_2,a_3,\epsilon_1,\epsilon_2,\theta,\phi,\psi,p_x,p_y,p_z) \tag{2.5}$$

Superquadrics can be recovered from a three-dimensional range points, such as a point cloud, the figure 2.5 shows an example of point cloud retrieved from a car toy and it superquadric representation.



(a)

Figure 2.5: A point cloud recovered from a toy car and its superquadric.

The reconstruction problem can be formulated as a minimization problem [41]. Suppose we have a set of N 3D points (x_i, y_i, z_i) and we want to find the 11 parameters Λ that describe the superquadric so that most of the points lie on the surface. The goal is to find a set of parameters for which the distance from points to the model is minimal, that is a least-squares minimization problem.

Casting that as a minimization problem and multiplying ($\sqrt{a_1\ a_2\ a_3}$) to the minimization terms to enforce the recovery of the smallest superquadric, arriving at:

$$\min \sum_{i=1}^{N} [\sqrt{a_1\ a_2\ a_3}\ F(\Lambda) - 1]^2 \tag{2.6}$$

2.3 Algorithms

This paragraph describes the most important algorithms used in this work. They mainly concern the manipulation of point clouds, first the reconstruction of the surface to generate the model associated with the point cloud, then the segmentation of the model into the parts that make up the object.

2.3.1 Surface reconstruction

Depth cameras have become more and more accurate and cheaper, these factors have allowed the increase in the use of 3D points generated from scans of objects. A traditional problem with these scans is to recover the physical shape of the objects, generating a digital representation. This problem is called *surface reconstruction* and it represents the process by which a 3D object representation is reconstructed or inferred, usually, from a point cloud, see figure 2.6. Since the reconstruction problem is ill-posed, given a set of 3D points it is possible to find an infinite number of surfaces that can pass through them. To deal with it, it must be regularized through prior knowledge [42]. Choosing one algorithm or another is dependent on these priors.

The CGAL Library [43], used in that work, offers three different algorithms: *Poisson Surface Reconstruction*, *Advancing Front Surface Reconstruction* and *Scale Space Surface Reconstruction.*

Poisson reconstruction considers all the points at once, without partitioning

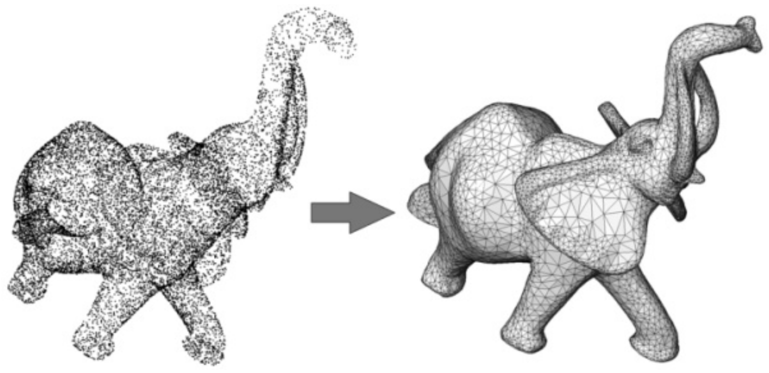


Figure 2.6: The image shows how CGAL can perform the surface reconstruction starting from a point cloud.

or clustering, therefore is highly resilient to data noise and it performs well approximating a point cloud with a smooth surface [44]. The algorithm works by computing an implicit function which is an approximate indicator function of the inferred solid.

Scale space algorithm takes as input a point cloud and computes a triangulated surface interpolating the points. It is also able to handle noise and outliers if in a limited number. The input points are modeled using a scale-space that can be considered as an abstraction of the point set. The algorithm works in two phases, the first one tackles the construction of the scale-space, while the second one computes the triangulated surface mesh of the point set at scale s is computed [45].

Advancing front is a Delaunay-based approach in which the output surface is composed by the union of some triangles. It is the algorithm used in this thesis. The reconstruction is performed sequentially, at each iteration the most plausible triangle is selected using the previously selected triangles [46]. This approach allows to add the most plausible triangles first and to postpone the difficult decisions, since an error can yield terrible results.
The first step of the algorithms is to find a candidate triangle for a boundary edge, in order to do that it defines the radius r_1 of a triangle t as the radius of the

smallest sphere passing through the vertices of t and enclosing no sample point. The triangle with the minimum radius is the starting point of the algorithm. From it, valid triangles incident to the edges are computed, maintained in a priority queue. At each iteration the algorithm pops most plausible candidate from the queue and computes the new candidate triangles and puts them into the queue.

2.3.2 Part segmentation

The image segmentation is a very famous technique used in computer vision that partitions a digital image into multiple segments. This approach is typically used to locate objects. The result of image segmentation is a set of segments that share some characteristic or properties, such as color, texture or shape. Collectively all segments cover the entire image. This technique is widely used, for instance in medical images to locate tumors or for surgery planning, or in object detection like face detection or pedestrian for autonomous cars and many others.
All the algorithms mentioned so far are valid not only for 2D but also for 3D models. Indeed meshes can be segmented into smaller and meaningful sub-meshes. The results can be useful for higher level tasks such as object recognition or scene understanding. Unlike the 2D images, segmentation in a 3D world is more difficult because the properties of the models are more and more complex, for example the segmentation can be based on the volumetric information of the model, or on the surface or on mesh components like vertices, edges and faces. Also the results are not as good for image segmentation that is able to reach very good outcomes in a vast variety of fields. One of the most advanced applications is the human body part segmentation, the figure 2.7 shows results achieved by Chuang et al [47], that have applications for pose estimation, robot-human interaction and gaming. But recently it started also to be developed for object segmentation that allows to distinguish several regions that compose the model, identifying for example in an airplane its wings, stabilizers and fuselage. Service robotics

also adopt these techniques to analyze objects with which humans interact daily, which robots might have to use in the future.
The advances in this area are encouraged also by deep learning with the most advanced and accurate neural networks. Obviously the training of those networks has become possible thanks to the ease with which today it is possible to obtain three-dimensional scans of objects.

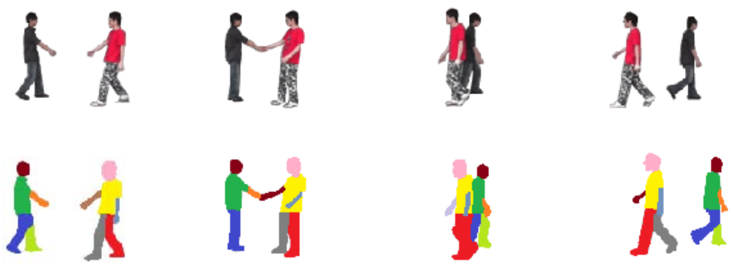


Figure 2.7: The image shows how part segmentation is used in human body to detect the pose of a person.

2.3.3 Triangulated surface mesh segmentation

As mentioned previously, mesh segmentation is the process of separating a 3D model into sub-meshes. The segmentation applied in this work relies on the algorithm provided by CGAL [43] called triangulated surface mesh segmentation. This algorithm uses the *Shape Diameter Function* (SDF) [48]. It is a scalar function that provides an estimate of the local object diameter for each facet of the mesh. The SDF is used to distinguish between thin and thick parts, it is also pose-invariant because the volume of the object remains unaffected by pose changes.

Given an input mesh, the SDF values are computed for each facet. At each iterations, several rays are sampled in a cone constructed using the centroid of the facet as apex and inward-normal of the facet as axis. A segment is calculated from each radius starting from the apex of the cone and ending in the first intersecting

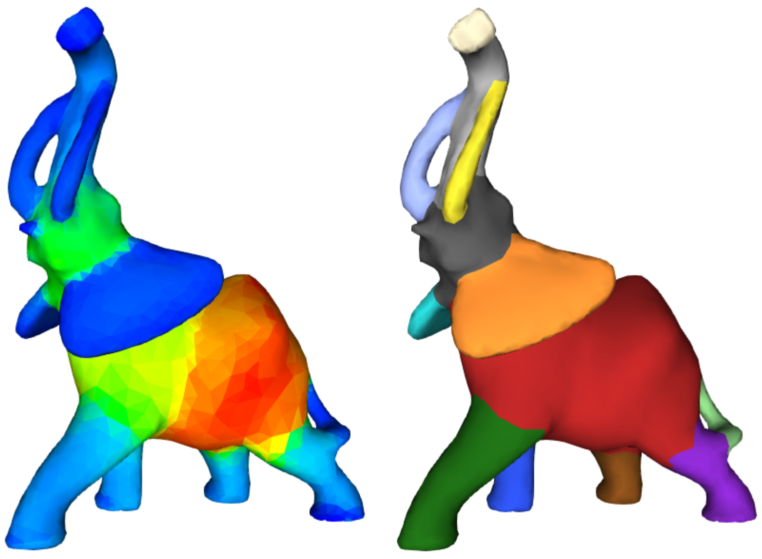


Figure 2.8: The shape diameter function applied to the elephant and the corresponding segmentation.

facet of the radius. Using the lengths of these truncated rays, which intuitively correspond to a local volume sampling, the raw SDF value is calculated by first applying an outlier removal procedure and then taking the average of the lengths. The raw SDF values are post-processed applying a smooth operator that removes the noise but tries to keep unchanged fast changes on values because they can be the boundaries of the surface. Moreover it normalized linearly the values between $[0, 1]$. The algorithm is composed of two steps:

the first one is a soft clustering on the facets using the processed SDF values. It consists in using a Gaussian mixture model fitting k Gaussians to the histogram of SDF values of the faces. The result of this clustering process is that each face has a vector of length k containing the probability to be assigned to one of the clusters.

The final step consists of a graph-cut algorithm that assigns a single partition for each face. This algorithm minimizes an energy function that takes in input the SDF values from soft clustering, geometrical values like concavity and dihedral angle and finally a smoothness parameter.

2.3.4 Point Cloud Upsampling

Point clouds are very useful instruments for manipulating 3D objects, but they usually have the drawback of not being homogeneous in term of number of samples. Indeed, a point cloud generated by scanning an object has a number of points, but repeating the scan this number can change. Furthermore, when scanning another object the number of samples changes in comparison to other objects. In addition when the point cloud is read from a file, like stl model, it usually has the smallest number of points that describe the object, for instance a very regular and squared object can be described with a couple of tens of points. Considering all these disparities it is necessary to make the point clouds homogeneous.

An upsampling algorithm was developed to increment the number of points for the models loaded from the stl files. A model loaded from such kind of file is described by two components, the set of points and the set of surfaces. Each surface is described by the indexes of the points that compose the vertices. In our case since the models are triangular meshes the vertices are three for each surface.

The algorithm is composed of three steps. Firstly, for each faces the corresponding area is computed. Since the vertices are described as a tuple (x, y, z), it is possible to compute the sides of the triangle computing the distance between the vertices, as the norm of their difference $||\vec{x_1} - \vec{x_2}||$. Once the area of each face is computed a vector is created with the size equal to the desired number of samples. This vector is populated with the faces of the model, picking them randomly but with a specified weights, the areas of the faces. The weights represent the probability to pick the relative element in the faces list, this means that the larger the face area, the higher the probability of selecting it. Therefore, the largest areas will be those with the most samples.

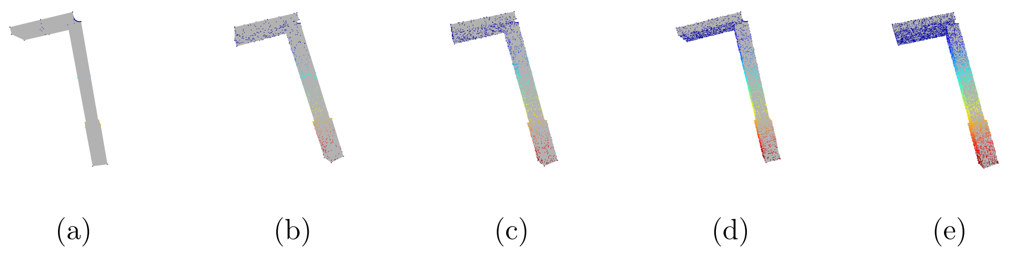


Figure 2.9: The figure shows the upsampling procedure with several number of samples applied on the same model. The images show in grey the 3D model and placed above the colored points to make viewing easier. The figure (a) shows the original model without upsampling, the point clouds is composed by just under a hundred of samples. The figures (b) to (e) show the upsampling process with a gradual increase of final samples, respectively 1000, 2000, 4000 and 8000.

2.3.5 Iterative Closest Point

The point clouds generated from a depth image usually have a limitation: only the visible portion of the surface can be scanned and therefore the reconstructed model is incomplete. There are several techniques to recreate the missing portion of the pointcloud.

The easiest way can be to scan the object from several angles acquiring a point cloud for each iteration. At the end all the partial model has to be joined following a stitching algorithm thanks to the identification of some main points that can be matched between two point clouds. For this method it is necessary that the object is fixed and the largest portion of the surface is visible from several angulation. Therefore, in a real environment it is not always feasible because usually the tools are located on a table or on a rack and for this reason the half in contact with the surface is never visible.

Another way is to infer the missing points from the available pointcloud but, as said before, usually only half of the tool is visible so it means that the missing part has to be completely inferred from the available one. If the tool is not so regular and homogeneous the final point cloud can be very far from the original tool. The last method instead compares the scanned point cloud with another one that is completed and preregistered. The two point clouds, however, can have different scale and pose, moreover it is not always sure that the model selected is the correct one. To be able to understand how correct the selected point cloud is, it is necessary to compute the difference between two point clouds.

The Iterative closest point is an algorithm that is able to minimize the difference between two point clouds [49]. It is often used to compare 3D surfaces from different scans, as in our case. The inputs to the algorithms are two point clouds, one the source and the other the target. The goal is to find the transformation, combination of translation and rotation, that best matches the source to the target, that instead it is kept fixed. The algorithm iteratively revises the transformation in order to minimize the error metric, it is often computed as the sum of squared differences between the coordinates of the matched pairs 2.7. The algorithm is composed by several steps, where initially for each point of the source point cloud the closest point of the target is matched; the transformation is then estimated using a root mean square euclidean distance minimization that best aligns each source point to its match found in the previous step. Continuing, the source point cloud is rototranslated using the computed transformation. These phases are iterated over, associating again the source points to the target ones and computing the new transformation, and so on.

$$e = \frac{1}{N_p} \sum_{i=1}^{N_p} \|\vec{y}_i - \vec{p}_i\|^2 \tag{2.7}$$

2.4 iCub

The experiments carried out for this thesis have been performed using the iCub Humanoid Robot [50, 51, 52]. The iCub is the result of the RobotCub project started in 2010, a European project to create a common platform for researchers interested in embodied artificial cognitive systems. The project aimed to implement a humanoid robot of the size of a little child which would serve as a workbench for research in embodied cognition. The platform had to be developed in terms of hardware and software. The former has to be modular in order to be extended and has the possibility to integrate new components. The latter has to reflect the modularity of the hardware, so it has to be designed with reusable and combinable modules.

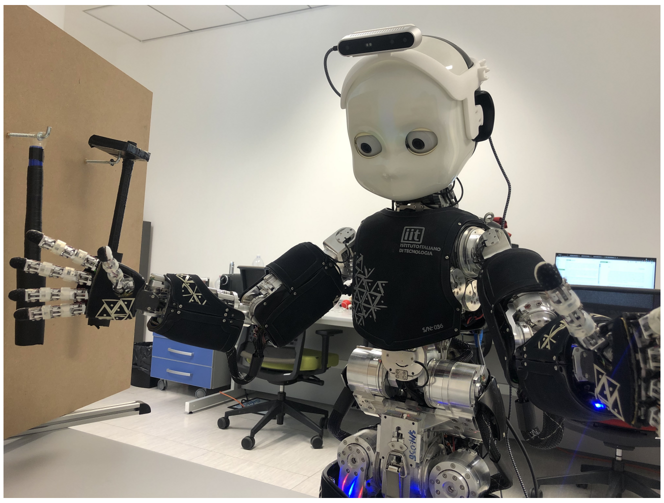


Figure 2.10: The humanoid robot iCub.

iCub is 104 cm high and is the size of a five-year-old child. The robot has 53 degrees of freedom. They are so allocated, 7 DOF for each arm, 9 for each hand and 6 for the head. Continuing, each leg has 6 DOF and 3 DOF waist that increase the range of motion for the body. Most of the joints are tendon-driven, which reduces the size of the robot and introduces elasticity. For example the hand is controlled by 7 motors placed remotely in the forearm which control the motion of the fingers, the flexing is controlled by the tendons while the extension is based on a spring return mechanism. Thumb, index and middle fingers are independent controlled while the last two fingers are coupled together. The thumb can also be controlled for adduction and abduction movements thanks to two additional motors mounted directly inside the hand. This architecture allows iCub to have a very advanced control for grasping and manipulating objects. The robot is equipped with some sensors, for instance cameras for computer vision analysis, microphones and speaker for listening and speaking with humans, force-torque sensors and joint angle sensors for motors and movements control and also full-body skin sensing capabilities. Cameras endow iCub with a binocular vision and they have 3 DOF that support both tracking and vergence behaviors. Thanks to the cameras it is possible to estimate the depth image from the disparity image [53]. Recently iCub has been equipped with a RGB-D camera (Intel RealSense Depth Camera) that provides better depth images thanks to the higher accuracy and precision.

2.4.1 Yet Another Robot Platform

On the software side, the iCub employs *Yet Another Robot Platform* (YARP) a thin middleware that enables inter-process communication across a network of machines [54]. The libraries, indeed, are in charge of real-time communication and hardware interface that is suited for cluster computation. Projects' software is organized in modules that communicate with each other across YARP network, taking advantage of decoupling of tasks in sub modules, reuse of modules and parallelization of algorithms also on several different machines. YARP has been coded to be OS-independent and the communication between modules follows the producer-consumer pattern. So the producer can send a message across the network, using different underlying protocols like UDP or TCP and others, thanks to the concepts of *ports*. On the other side the consumer can connect or disconnect at run time to a port and read the content of the messages. Therefore, two distinct modules can run on two machines with different operative systems and they can communicate effortless.

Chapter 3

State of the Art

In humanoid robotics one of the most important objectives is to enable robots to perform a series of tasks for which they had not been originally developed, by learning new skills and adapting to new environments. A fundamental prerequisite towards that purpose is to be able to recognize and manipulate external elements, such as tools, to carry out tasks for which their own manipulators are not sufficient. On one hand, robots have to learn the tool affordances in order to manipulate the tools, predicting the output of a predetermined action in a specific environment. On the other hand, they have to be able also to recognize the tools in the environment usually done by means of computer vision or deep learning. The ability to autonomously or semi-autonomously learn affordances and discover objects, gives robots the possibility to become more versatile and not only for a purpose specific.

In this chapter we will go into detail on how affordances are designed in humanoid robotics and how computer vision significantly influenced the computation of affordances but also the object recognition and identification. Firstly we will describe how robots using their manipulators interact with objects, and observing the effect of their actions upon them, they learn the related affordance. Later we will go into detail on tool affordances, namely the functionalities that intermediate objects allow the robot to reach through an action on a specific target. Finally, we will highlight the most important aspects of the computer vision and deep learning that have impacted this field improving the outputs of affordances and the capabilities of robots.

3.1 Robot Affordances

The first approach that endows a robot with affordance learning capacity has been conducted by Fitzpatrick et al. [55]. They showed that affordances can be learned by the robot by observing the effects of its actions on the objects. Indeed they adopted a *learn by doing* approach, whose aim was teaching to a robot that, for example, a spherical objects can roll while a box can only slide if pushed or pulled. In that work, affordances are formalized as the relation among the terms of that tuple $\{object, action, effect\}$, that would become a commonplace in future works on that field. Given the object and the action, the object affordances are described as a histogram that represents the probability of displacing the objects in a particular direction given that action.

Another general way to describe and learn affordances was proposed by Krüger et al. In their work [56], they formalize Object-Action Complexes (OACs). OACs are defined as a tuple $\{E, T, M\}$, where E represents the robot action, T identifies the change of state due to the action, and M measures the success rate of the action. OACs are very versatile because they have the ability to represent a low-level process as well as high-level information, which allows them to be used across a wide range of research fields. In addition, OACs can be combined to produce more complex behaviours. The main drawback of that study is that objects are identified by labels, which is why they cannot be easily adopted in learning algorithms that use generic objects.

Subsequent studies tried to overcome this problem substituting the labeled tool with a set of object features in the affordances analysis. For instance, Montensano et al. [57] proposed a model approach for learning affordances that takes in as input simple object features like shape, color. A Bayesian network connects these features to the action and observed effects, as random variables and the relations are dependency links. Another approach was proposed by Ugur [58] in which the raw features of the object are represented as a vector. Features that produce the same effect of others are generalized. This method has been further improved to discover commonalities in action-effect experiences that allow the clustering of outputs. [59]. Once the effect categories are discovered the affordances can be predicted by learning the mapping from the object features to the effect categories.

Another method for learning objects affordances can be mapping object and effect features into separate Self-Organizing Maps (SOMs), as Ridge proposed in his works [60, 61]. The main differences of these works are two. Firstly, the learning algorithm determines for itself the affordance classes given a number of different outputs. Secondly, the affordance classes are based on a set of input features, such as shape.

3.2 Tool Affordances

In this section, we will see how tool affordances are designed and developed. The ability to manipulate an object, as we have already seen, allows robots to perform many more actions. Pioneer work on tool affordances was led by Stoychev and his team [62]. The approach chosen is behavior-based and consists in a phase in which the robot randomly chooses an exploratory behavior, applies it to the tools and observes its effects. In this way the robot learns incrementally affordances by representing them as a list of performed actions and the probability of success with that action. The learned representation allows to infer the similarity between tools based on the outcomes produced.

In Tikhanoff et al [63] the goal is to create a system able to complete a object retrieval task, which requires choosing a tool among a predefined set. The correct tool is chosen after having learned the affordances. The recognition of objects is the first step and the geometric features extracted from 2d images are two of the basic components of the affordances learning. Then the robot is free to explore affordances by tapping the object from any directions. This is necessary since an object can react differently depending on the direction of the action, for example a cylinder can roll easily if pushed from the correct direction. The geometrical features, action and effects are used to learn a map that relates these inputs to effects.

However, these studies have a drawback, the learned affordances are limited to a set of labeled tools, so the knowledge cannot be extended to tools external the initial training set. Subsequent studies tried to overcome this problem substituting, in the affordances analysis, the labeled tool with a set of features extracted directly from objects. Jain and Inamura tackled the problem of applying experience learnt with one tool to another one [64]. The main difference among tools is based on their functionalities. They depend on geometrical features of the tool itself, sharp edge or blade for knife or other cutting tools, bowl and spoon have to have a convex shape. Tool affordances incorporate these geometrical features, so tools having similar features can be used to generate similar effects. A probabilistic model is chosen to implement the learning model in order to deal with uncertainty and redundancy. The discovery of new functionalities is based on geometrical features that are annotated by hand so the process cannot be extended to any new tools. However, Gonçalves et al made functional features automatically detectable [65]. Features are composed of 2D geometrical properties like contour perimeter, the area, size or the minimum-enclosing circle and rectangle. Mar and Tikhanoff in their work [66] extended the set of geometrical features achieving a set of 75 features extracted from the contour of the blob. It is composed by shape descriptors, such as the area, perimeter, length and width, the ratio, the rectangularity, but also some features from convex hull, like the

larger convexity defects, and some properties that consider the centroid.
In [67] a new way to represent object features is presented, the Oriented Multi-Scale Extended Gaussian Image (OMS-EGI). It is a set of 3D geometrical features able to encapsulate in a general and compact way the geometrical properties of a tool on a particular grasp configuration.The figure 3.1 shows the steps for the computation of the OMS- EGI features. The starting point is a 3D model, from which the bounding box is computed and also its normals. Voxels are then created dividing iteratively the volume of the bounding box. For each voxel the histogram of normal orientations is computed. Finally all histograms are concatenated.

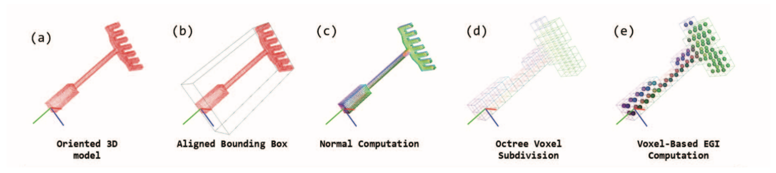


Figure 3.1: OMS-EGI Computation Steps: Starting with a 3D model oriented according to the way in which it is grasped (a), its axis-aligned bounding box (AABB) is computed (b), as well as its normals (c). In the next step (d), the volume enclosed by the AABB is iteratively divided into voxels of different resolutions and for each of them the histogram of normal orientations is computed (e).

Myers proposed a framework to extract 3D geometrical features from RGB-D images [68]. The dataset is composed of 105 kitchen, workshop and garden tools from 17 object categories. For each tool the depth image is extracted. Given an object in a RGB-D image, a superpixel segmentation algorithm divides it into a collection of surfaces. The hypothesis is that there is a deep relationship between effective affordance and geometry of a portion of tool, since the geometric and physical properties of objects are closely tied to the ways they can interact with the environment. The geometrical features extracted are for example color, gray scale, depth, surface normal or curvature. An algorithm aggregates them using max-pooling and the output is a feature vector for each surface.

Deep learning approaches have also been applied for tool affordances training. Roy in [69] used RGB-D data from an affordance database to train multi-scale CNNs with different resolution. Three multi-scale CNNs extract mid-level visual cues, depth map, surface normals and semantic segmentation of surfaces. The outputs are passed as inputs to another multi-scale CNN for predicting affordance map. The affordance types considered in that work were, walkable, sittable, lyable, reachable, movable, and each of them has a particular characteristic such as walkable if any horizontal surface with a similar height has no obstacles above it. Another set of studies focus the attention on an aspect. In order to learn affordances and better generalize to other tools, objects have to be analyzed at the level of their parts to understand object functionalities. In [70], Schoeler created a set of primary tool functionalities. The function of a new tool is found by comparing it to the previously mentioned set. The unknown tool is segmented in its parts and analyzed. For example a helmet can be used to transport water. Abelha and Guerin presented a system [71] that takes in input a tool's point cloud and produce a score for how effective it is for the task and how to use the tool. Indeed, authors considered that completing an action requires several steps, one of them is to know where to grasp the tool, how to orient it and which part to use as end-effector. For example in a human environment like a kitchen is very common to use an object in a creative way. For example a bottle can be used as a rolling pin. In order to use an object differently from the expected one, it has to be segmented in parts and analyzed individually to compute the affordance score. The part that maximizes the score is chosen as end-effector and the other one is selected as graspable part.

The main shortcoming of these works is the absence of tests on a real robot. As long as the architecture has to extract geometrical or functional features from images both bi and three dimensional, it is easier to perform tests that emulate a robot. Nevertheless, the grasp action cannot be tested in a simulator without proper configurations of a physical robot. Each robot has a different hand that allows to grab in a different way, but also the structure and the mobility of the arm effects the result of the action. A more complex arm can be able to grasp objects in several configurations, differently from ones with few degrees of freedom. In this thesis, we tried to overcome some of these limitations thanks to the tests conducted on a physical robot.

3.3 Computer Vision

In the previous paragraph several different works have been presented that take advantage of properties of the tools to improve the computation of the affordances. These properties are mostly extracted from visual features that objects have, and it can be done thanks to computer vision of neural networks. These fields are not only important for the computation of affordances but also for the recognition of the tools themselves and their grasping.

Stoytchev in his work [72] highlighted the importance of the grasping behaviour for the affordances improvement. Although, in almost all the works cited in this chapter this aspect has not been taken into consideration. Often due to lack of a robot that is able to grasp the object without the aid of an operator. Another important step required to endow robots with the ability to grasp autonomously the tools, is the ability to recognize the tools in an environment, even if circumscribed such as a shelf, a drawer or a rack. The first step to recognize objects is to detect them from the environment. This methodology is called image segmentation, in other words, the process that subdivides an image into its constituent parts or objects. Since there is no general solution to this problem, the techniques have been often combined with domain knowledge in order to effectively arrive to results for a specific case. The simplest method to segment an image is the so called *threshold method.* The key is to turn an image, usually gray-scale, into a binary image. The way in which this transformation happens is through a threshold that defines which pixels should become white and which ones black. The threshold can be computed as global or local. Knowing some information about the objects in the scene, like the areas and their predominant colors, it is possible to compute the threshold as a percentage of the number of pixels that allows to map them into the object [73]. Another approach can be applied to images composed by regions with different gray ranges, like medical images. The histogram of the image can be separated into a number of peaks, each corresponding to one region [74]. The algorithms that elaborate colour images have to analyze much more information, the approach proposed in [75] analyzes the values of each pixel in terms of the three components, red, green and blue, finding the maximum and the minimum values used to computed heuristic values to compare a threshold for each channel.

The segmentation problem can be formulated also as a deep learning model. The learning-based methods can be grouped based on several architectural features. The most successful and used architecture is the *Convolutional Neural Networks* [76]. It is characterized by the shared weights that are computed by each neurons applying a specific function to the input values received from the previous layer. The filters, the vector of weights, can be shared among many neurons. Another architecture is the *Fully Convolutional Models* that is vastly used in semantic image segmentation problems. These networks use a particular layer called skip connections, in which feature maps are upsampled and fused with feature from earlier layers. The main limitations are firstly the computational cost that is too expensive, not suitable for real-time inference. Moreover it can not be easily generalized to 3D images. Other examples of neural network to be mentioned are the PointNet [77] and the updated version PointNet++ [78] that are designed to manage point clouds. Indeed, these neural networks respect the permutation invariance of points in the input. The architectures provide a unified structure for three different applications: object classification, part segmentation and scene semantic parting.

The main limitation of a neural network architecture is the dataset. It is required in order to tune the correct weights training the network on the data present in the dataset. In order to achieve good results during the training and later the validation phases, the dataset has to collect a huge amount of data and the classes are to be homogeneous among them.

Chapter 4

Proposed Approach

The following chapter describes extensively the implementation developed in this thesis. Firstly, it is introduced the tool affordances project, developed by IIT contained into the Robotology Github repository [1]. It is composed by several modules the most important are analyzed in term of usefulness features but also limits that represent the starting point for this work. Indeed, this thesis has the main goal of improving and adding some features to the previously cited work. Some improvements are the increment in size of the dataset, usage of a multi model approach, introduction of point clouds and superquadrics for reading, modeling and analyzing the models. Moreover each tool is segmented into parts to choose the best graspable area and finally the affordance of the tool is computed using the tool affordances projects and providing features extracted during the analysis. All these developments are described in details in the following paragraphs.

As mentioned in the paragraph 2.4.1, the project is developed on top of the YARP middleware, taking advantages of the features that it provides. Firstly, the code is structured following the methodology of *divide et impera* with which more complex problems are subdivided into several modules of YARP and C++

[1] `https://github.com/robotology/tool-affordances`

classes. The modules can be easily connected one to each other using the ports that constitute the network provided by the middleware.

All the code developed for this thesis has been inserted into a Docker container. This decision was taken in order to improve the compatibility among different machines and also it increases the portability of the code itself. Indeed, the project takes advantage of some C++ libraries that allows to manipulate and show the point clouds, moreover it uses YARP middleware. All these dependencies are not difficult to install but they are time consuming tasks. With a *Docker container*, the user has only to download the image and automatically he can work in an environment already set, with all the dependencies and also the code installed and ready to run the demos. The image of the Docker container is available with the name *sankios/thesis* [2].

4.1 Initial project

The tool affordances project consists in three different phases. Firstly, during the exploration step, a tool must be provided so that iCub can extract a descriptor vector. After that, a series of actions are performed and the effects obtained are recorded. These data are used during the second phase which learns the relationship among the tool, the action and the effects. The last step consists in providing another tool, similar to those used during the training phase, also in this case the robot extracts the descriptive features and uses them to predict the expected effect of an action thanks to the affordances model trained in the previous step. The goal of the described steps is to achieve the maximum possible displacement on the target object, given a certain tool-pose. So, each exploration and prediction trial begins by placing a tool in the robot's hand. Once detected the tool in its hand, the robot is able to identify the tool-pose. After the training phase, therefore, the robot is able to select the action with the best expected

[2] `https://hub.docker.com/r/sankios/thesis`

effect for any tool-pose.

The main limitation of this approach is the absence of tool selection, indeed every trial starts with a tool already in the hand of the robot. In a real scenario, a robot is expected to be able to analyze the surrounding environment to find the tool that best fits for its goal. And also it should be able to grasp it without human interaction in order to be more autonomous. To that end, the goal of this work is to endow the robot with several capabilities. The first one is the ability to select a tool from a set present in the environment. In addition, for each tool a thorough research is conducted, reconstructing the 3D shape of the object that allows to know the dimension and orientation. With the reconstructed model it is also possible to find which part is the more graspable thanks to segmentation algorithm. The identification of the handle is very important to allow the robot to move the arm and grasp the tool autonomously removing the human assistance. In addition, all the information retrieved during the model analysis are very useful for the affordances computation since the orientation of the tool and the position of the end effector are already known once determined the handle.

4.2 Multi model approach

The first improvement was to equip iCub with the possibility to choose the tool only from the available set, to do that it must be able to recognize all the instruments present in the scene. Therefore the approach with the models was changed and the tool was not provide at the beginning as in the previous approach.

4.2.1 Tools Dataset

In order to use a multi model approach, the system has to be based on a dataset suitable for our actions. The dataset is composed by tools used in the preceding work [79] and series of tools available on *3dwarehouse*. Tool classes present in the models are hoes, linear and bent sticks, rakes, shovels, hammers and spatulas. The dataset from [79] provides tools in a *.x* extension. It is file format introduced by Microsoft in 2008 containing geometry meshes and material information. Libraries that manage 3D model usually use stl or ply files to represent meshes. Thus, the first step was to convert the x files into a stl file. It can be done easily with online tools, in this work we used *ofoct.com service* that can manage a vast variety of file extensions. The *3dwarehouse website* provides a diversified assortment of tool classes. The downloaded ones are from the previously mentioned. In that case the downloaded models use a proprietary extension *.skp* of *SketchUp*. The same software house provides an online editor able to convert the files into stl extension. Files from both dataset, due to the extension conversion or due to an incorrect modelling, can present holes or not well closed surface in the model. The *netfabb service* is able to fix all these problems closing the open surfaces and generating a stl files well formed. The files at the end of all the manipulations constitute the dataset used in this work, some examples of tools used inside the simulator are provided in the figure 4.1.

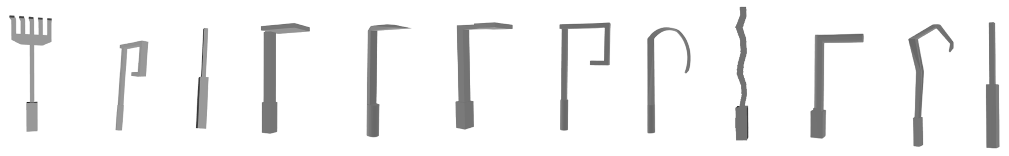


Figure 4.1: The figure shows some tools from the dataset used into the simulator.

4.2.2 Tools Segmentation

In a multi-model environment, the system must be able to first divide the tools from the scene, and then to distinguish one tool from another. The former problem is tackled thanks to an analysis of the scene from a computer vision perspective. From the RGB image of the scene the average color is computed, a range around these colors is considered and the background color is contained into that delta of values. In order to segment the tools from the scene, it is necessary to create a mask to highlight the objects. The mask is created starting from the range of colours, because all the pixels not contained into that range are considered as tools, instead if it belongs to the range it is considered as background. The mask just created can contain noisy pixels or holes inside surface. To remove them the classical morphological transformations are applied, erosion and dilation. Firstly an opening transformation is applied, it consists in an erosion followed by a dilation that deals with removing white pixels from black surface. Later, to fill the holes and join detached areas the closing transformation, that instead is a dilation followed by an erosion. The opening and closing processes are performed using a 3×3 kernel, and the erosion and dilation in the opening are performed 3 times, instead transformations in closing 2 times. For each region remained after transformations it's computed the region of interest (ROI), the minimum rectangle that encloses the region. This is necessary to individuate which tool it is, and also to extract the pointcloud of that tool. The entire pipeline of 2d image segmentation is shown in figure 4.2, furthermore the algorithm 1 represents a pseudo code for the 2d image segmentation.

Algorithm 1: Tools segmentation from 2D image

```
   Input: RGB Image of the scene $\mathcal{I}_{in}$.
   Result: List of bounding boxes $list$<bb>
1  $\mathcal{I}_1 \xleftarrow{2} \mathcal{I}_{in} \ominus \mathbb{R}(3)$;
2  $\mathcal{I}_2 \xleftarrow{2} \mathcal{I}_1 \oplus \mathbb{R}(3)$ ;
3  $\mathcal{I}_3 \xleftarrow{6} \mathcal{I}_2 \oplus \mathbb{R}(3)$;
4  $\mathcal{I}_f \xleftarrow{6} \mathcal{I}_3 \ominus \mathbb{R}(3)$ ;
5  contours $\leftarrow$ findContours($\mathcal{I}_f$) ;
6  $list$<bb> $\leftarrow$ boundingBoxes(contours) ;
```

The input data is a rgb image $\mathcal{I}_{in}$ and return a list of bounding boxes $list$<bb>. $\ominus$ and $\oplus$ represent respectively the erosion and the deletion with a kernel $\mathbb{R}(3)$

a 3×3 rectangular structure. In addition $\overset{i}{\leftarrow}$ represents the number of iterations the morphological transformation is performed.

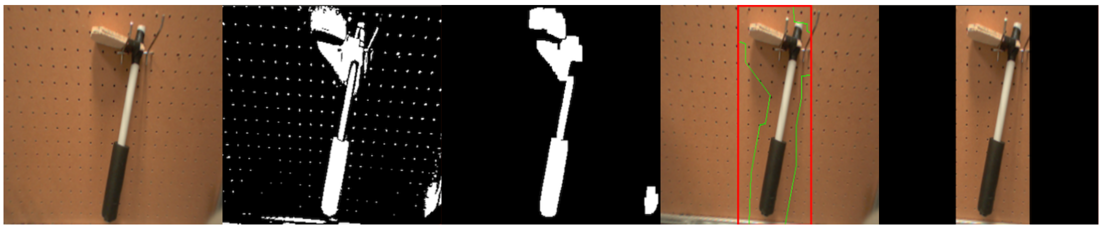


Figure 4.2: The segmentation process takes the image scene as input (a). The mask is calculated using the color information (b), the background color is not very dissimilar to the average color of the scene. The mask can be affected by noise, so morphological transformations, opening and closing are applied, and the denoised image is obtained (c). For each region, the polygon that best matches the region is found (green line in the image). The bounding box (red line in the image) is generated for each polygon calculated in this way (d). Finally, the region of interest of that instrument is shown (e).

4.2.3 Model name matching

In order to continue the analysis of tools is necessary to identify each model correctly. Indeed, in order to use the iterative closest point 2.3.5 it is necessary to have to point clouds. One of the two is loaded from the stl file associated to the model and in order to find the correct one is necessary to classify the tool.
The robotology collection provides a module for this situation. The *Interactive Objects Learning* repository [3] contains the functions necessary to the learning phase in which the selected tools are shown one at time and the model name is provided. Then the module is able to provide a probability distribution of which tool is presented. This is very useful for this work because once the tools are

[3]`https://github.com/robotology/iol`

segmented by 2D analysis, the regions of interest are provided to the module that recognizes the tool contained inside the roi giving back the model name.

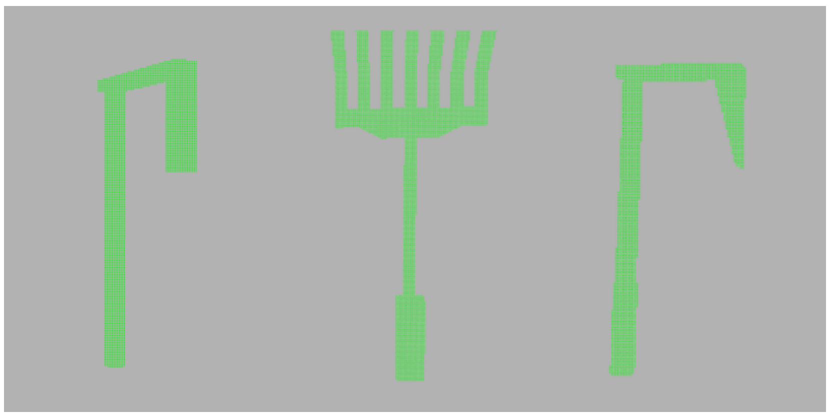
4.3 3D vision analysis

Once the tools are segmented it is possible to continue to the other step. To interact with each tool it is necessary to reconstruct its 3D shape. In order to do that the depth map of the scene has to be reconstructed, this can be done by using the RGB-D RealSense Camera that can be mounted above the iCub head with a specific helmet.

4.3.1 Point Cloud

The first step to reconstruct the 3D shape of the tool is to extract the point cloud from the scene. In the section 4.2.2 we extract a ROI for each tool present in the scene. This is essential to compute the point cloud, because using the entire depth image the computed point cloud would be composed potentially by multiple tools. So the ROI is used as filter to extract the point cloud only inside that rectangle. Some examples of point clouds are presented in the figure 4.3.

Given the depth image, only the pixels contained inside the rectangle are computed. To retrieve the 3D points from the camera coordinate frame, the intrinsic and extrinsic parameters are required. Following the pinhole camera model 4.1, it is possible to retrieve world coordinate with an inverse transformation, called 3D reconstruction. The first matrix represents intrinsic parameters in which f_x and f_y are the focal length, horizontal and vertical and in case of squared pixel they are equal. They are retrieved directly from the camera driver at run-time. Instead c_x and c_y represent the principal point, in our case the image center.



(a) (b) (c)

Figure 4.3: The figure shows the extracted point clouds from the depth image associated to three different models.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R|t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.1}$$

Firstly using intrinsic parameters pixel coordinates are mapped into camera coordinates, and later the extrinsic parameters transform them into world coordinates, the pseudo code 2 shows the algorithm previously described, $focalx$ and $focaly$ represent the focal length and cx and cy the center of the image, instead $Teye$ represents the inverse of full-rank extrinsic matrices. In that way the point cloud of the scene is computed and finally it has to be divided between the object and the background. Since the scene has a homogeneous background it can be easily segmented thanks to the Ransac algorithm that fits a module to the surface and from it extracts the object. The algorithm is presented in 3.

Algorithm 2: 3D reconstruction algorithm

```
Input: Point from RGB-D camera $P(u, v, d)$
Result: The 3D point $B(x, y, z)$
1 B.x = P.d * (P.u - cx ) focalx
2 B.y = P.d * (P.v - cy ) focaly
3 B.z = P.d
4 B = Teye * B
```

4.3.2 Model Fitting

From the previous algorithm it is possible to obtain the point cloud relating to the ROI taken into consideration, so later called *object point cloud*. Moreover the ROI is used also by 4.2.3 to find the model name contained into the region considered. This step is mandatory to retrieve the entire point cloud associated to that model directly from the stl file, the *model point cloud*. But it has a wrong scale factor and pose.

The next step is to fit the model point cloud to the object one, using the Iterative Closest Point well described in the paragraph 2.3.5. Since the depth image can be affected by random noise the algorithm is run several times, each time recomputing the object point cloud in order to find the reading that best approximates the model point cloud. Since the output of the iterative closest point provided by Point Cloud Library [80] is a score that represents the error, the iteration with the minimum value is the best fitting. In the algorithm there are three different parameters that have to be set and they are, *maximum iteration*, *transformation epsilon* and *Euclidean fitness epsilon*. They represent the termination criteria for the algorithm, the first one indicates the maximum number of iterations of the algorithm, the second one the maximum difference between the previous and the current estimated transformation and the last one the maximum of the sum of Euclidean squared errors. The default values are

Algorithm 3: Point clouds Ransac algorithm

```
Input: Scene pointcloud scenePointcloud
Result: Isolate the objectPointcloud from the surfacePointcloud
1  sampleConsensus ← ransac(scenePointcloud) ;
2  inliers ← sampleConsensus.getInliers() ;
3  objectPointcloud ← emptyPointcloud ;
4  surfacePointcloud ← emptyPointcloud ;
5  for point in scenePointcloud do
6      if point in inliers then
7          surfacePointcloud.insert(point)
8      else
9          objectPointcloud.insert(point)
10     end
11 end
```

mentioned in 4 but they can be modified at run time through an RPC port [4] of YARP. Some results from the Iterative Closest Point algorithm can be seen in the figure 4.4 that shows three different models, with the object point cloud and model point cloud.

Algorithm 4: Iterative Closest Point parameters

```
1 MaximumIterations = 500
2 TransformationEpsilon = 1e-9
3 EuclideanFitnessEpsilon =1e-6
```

[4]`http://www.yarp.it/git-master/rpc_ports.html`

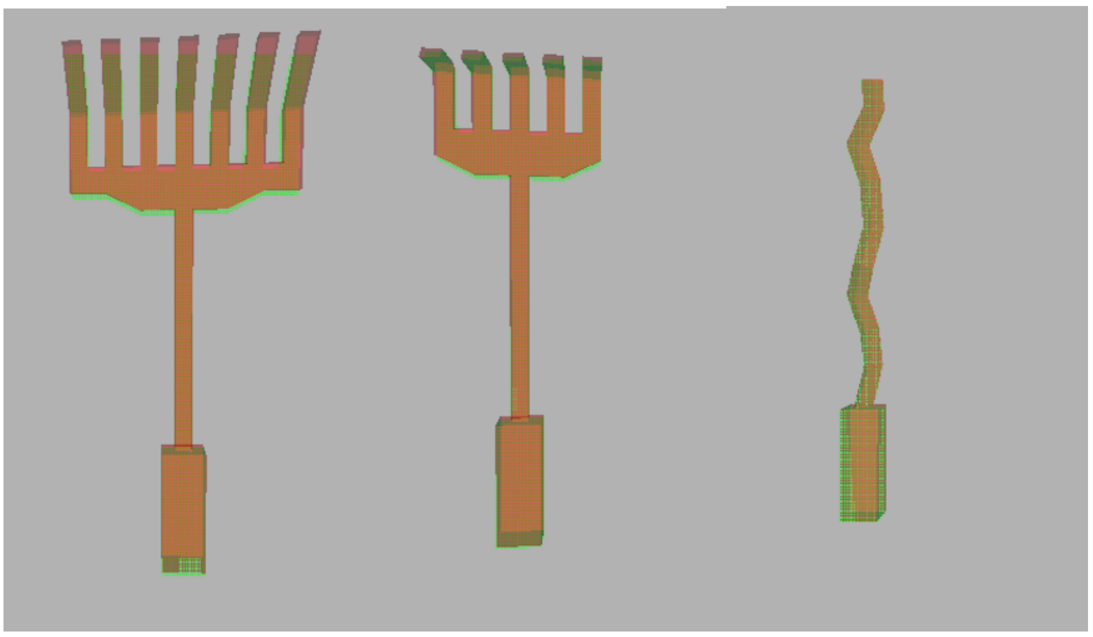

(a) (b) (c)

Figure 4.4: The figure shows three different models after the execution of the *Iterative Closest Point* algorithm. The object point cloud is represented in green, instead the red point cloud is the model one.

4.4 Grasp the tool

Several works mentioned in the chapter 3 compute affordances starting from the end effector of robots, others instead consider also a tool in the affordances analysis. In order to maximize tool affordances, robots have to be able to manipulate a tool, managing how it is grasped and its final pose. What is said highlights the need to endow the robots with the ability to grab and manipulate the tools. iCub has this ability so the model point cloud fitted in the previous paragraph is used to defined the part of the object that can be used to be grasped. The superquadrics are the best geometric model that can be computed over the point cloud. They allow to better analyze if the robot is able to grasp the parts and provide more geometrical features to the tool affordances system.

4.4.1 Mesh reconstruction

The algorithm used to reconstruct the mesh from a point cloud is the *advancing front surface reconstruction* described in the paragraph 2.3.1. This step is necessary because the stl files have a different number of points that describe the model. To level out all models, they have been upsampled to 4000 points in order to have the some numbers of samples, the algorithm used for the upsampling is explained in 2.3.4 and some outputs are shown in figure 4.5. Once upsampled the new shape with all the added points has to be reconstructed and a new stl file is generated. This new model is used to generate the model point cloud.

Once the model point cloud is fitted to the object one, the paragraph *Model Fitting* 4.3.2 describes the procedural steps, the reconstructed mesh can be segmented in several parts in order to analyze them in term of graspability.

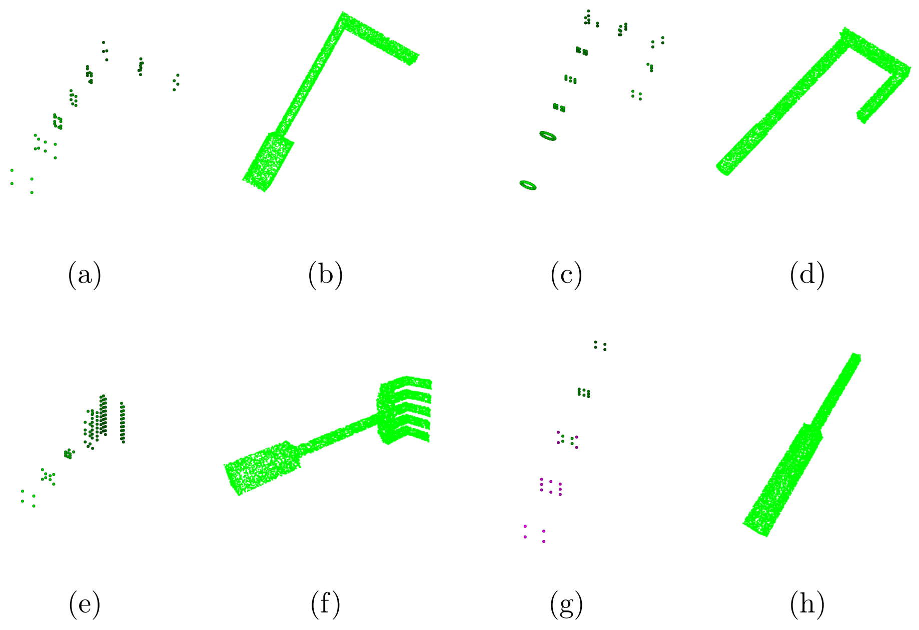
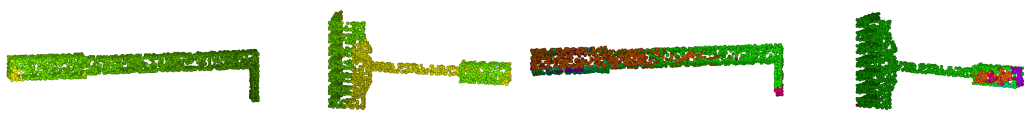


Figure 4.5: The figure shows the original point clouds $(a), (c), (e), (g)$ and the upsampled point cloud with 4000 samples $(b), (d), (e)(h)$.

4.4.2 Mesh segmentation

In order to perform the segmentation of the point cloud we have tried several possible solutions. Firstly we have approached to the problem with a deep learning solution. The first neural network tested was the PointNet, the most famous architecture to analyze point clouds, and also the PointNet++ which contains some improvements over its predecessor. Finally we have tried to segment point clouds also with the DGCNN, a much more recent architecture if compared to the other two mentioned.

The tests with the neural network don't have provided the expected results because doesn't exist a dataset that contains objects similar to the used in this work already segmented in parts. The most famous dataset is the *ModelNet* that provides up to 40 classes of objects very different one to each others. Instead our dataset contains a large number of tools but all of them can be organized into a small number of different classes, and also from a class to another the differences are not so evident.



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

Figure 4.6: The figure shows the different outputs produced by two different neural network, the images (a) and (b) from the *PointNet* instead (c) and (d) from the *DGCNN*.

Since segmenting the point cloud doesn't provide the desired results, as shown by the figure 4.6, we have changed our approach and considered the mesh reconstructed from the point cloud. The approach used to segment the mesh is the *Triangulated surface mesh segmentation* that uses a scalar function to estimate the diameter for each facet of the mesh. The observation behind the algorithm is that the volume is unaffected by pose changes, and the steps are widely described in the paragraph 2.3.3. The model is now subdivided into several point clouds each one associated to a different segment that can be analyzed in the following paragraphs. The figure 4.7 shows the same models of figure 4.6 but segmented with the *Triangulated surface mesh segmentation* algorithm and the results are are visibly similar to how humans would do it.

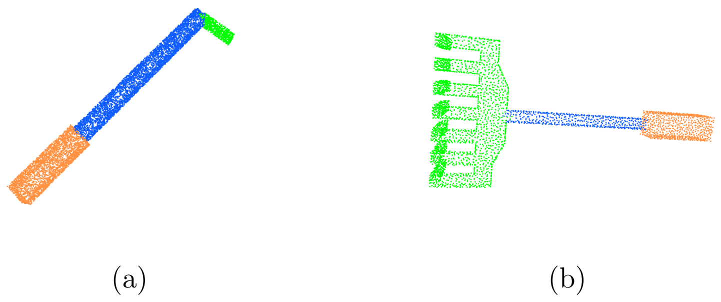
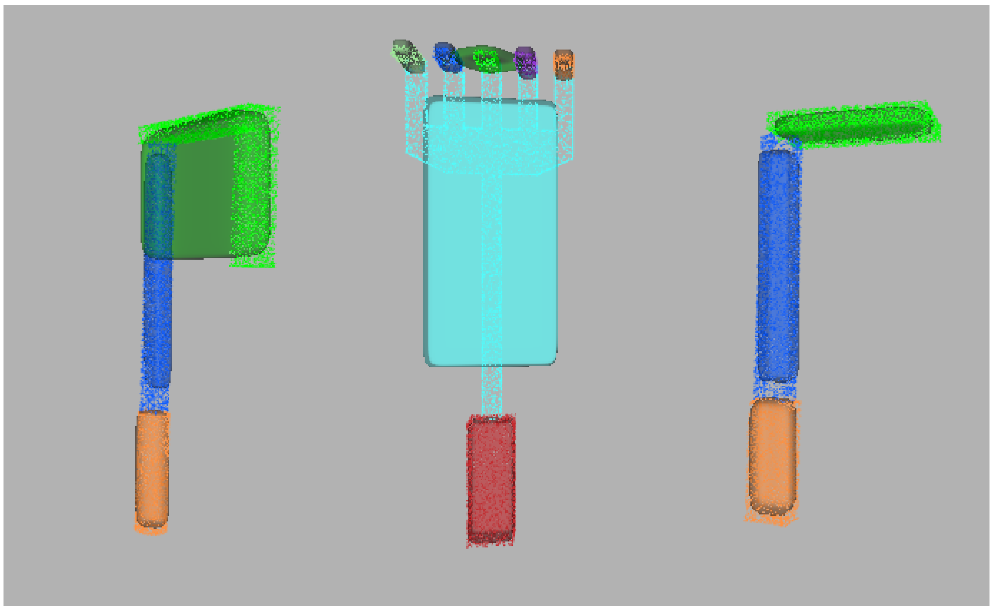


Figure 4.7: The figure shows two models segmented with the Trinagulated surface mesh segmentation algorithm.

4.4.3 Superquadrics

Superquadrics are volumetric models for shape representation. They are appropriate for point clouds because are able to express them as simple and linear surfaces and solids. Moreover they can be modelled with the classic deformations lie bending and tampering. The reconstruction of a superquadrics from a point cloud is a minimization problem as described in 4.4.3 because it tries to tune the parameters of the superquadrics that better fit the point cloud. Some examples of superquadrics associated to each segment can be found in figure 4.8.

The superquadric is expressed in terms of its center (x, y, z), the angle around the z-axis, the size the three direction (s_x, s_y, s_z) and the squareness parameters epsilon (ϵ_1, ϵ_2). All these parameters are used to represent the superquadrics, manipulate and also visualize them.



(a) (b) (c)

Figure 4.8: The figure shows the superquadrics of three different models. Each superquadric associated to a segment is colored with a different color. The superquadrics are opaque to make point clouds visible

4.4.4 Graspable Area

Once the superquadrics are computed, the module can easily identify the graspable part. Of course, the graspability depends on the robot and its hand for this reason a tool can be grab by a robot and not by another. Our goal is to find out which tool is graspable by iCub robot and also which segment in the tool is the most suitable for the handle.

To solve this problem the idea is to compare the object that can easily grasped by the iCub hand with each segment of the tool. Analysing the grasp area from the CAD drawing of the hand, we defined the object that best matches the space contained in the iCub hand, a cylinder of 32 mm of diameter and 80 mm of height.

So the point cloud of each part segmented is compared with point cloud of the cylinder. This approach allows to identify the part of the tool that best fits as handle. The pseudo code 5 provides an idea of how the comparison is performed and the figure 4.9 shows some outputs from the function.

Algorithm 5: Handle fitting

```
Input: Parts of the model segmented: segments
Result: The segment that best fits the cylinder
1  minScore ← ∞ ;
2  minI ← -1 ;
3  cylinderPointcloud ← getPointcloudCylinder() ;
4  for (i, segment) in segments do
5      score ← computeICP(segment, cylinderPointcloud) ;
6      if score < minScore then
7          minScore ← score ;
8          minI ← i ;
9      end
10 end
```

In order to grasp the tool the robot has to know the position of the handle respect its frame. Since all the information about the cameras are provided it is possible to compute the transformation from the pixel frame to the camera frame following the pinhole camera model. Once transformed and given the rototranslation between the center of the camera and the root of the robot it is possible to compute the point cloud respect to the robot frame. In algorithm 2 it is provided a pseudo code that formalized the above-mentioned steps.

Since the superquadric is computed in the same frame of the related point cloud it will have the same measurement units and also the position is maintained. So the superquadric that fits the handle is described by its center, the parameters of the shape and its size. With this information it becomes easier for the robot

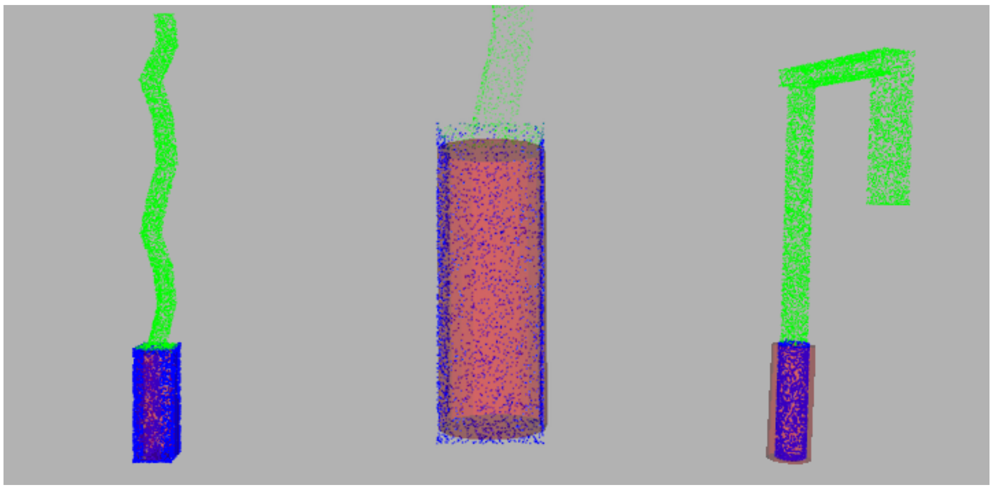


Figure 4.9: The figure shows the segment that best fits as a handle in blue, all other parts are in green. Behind the blue point cloud is also the cylinder model representing the volume which iCub can easily grasp, in red. The image (b) shows the detail of the handle of the same object represented in the figure (a), instead in the image (c) the cylinder is more visible since the tools has a thinner handle.

to know where and how to grasp since with the center information it knows approximately the area that is has to reach, and knowing also the size of the superquadric it is possible to compute the correct position that allows to put the tool between the hands.

4.5 Overall algorithm

The pseudo code 6 provides an overview of how all the sections previously explained are implemented and used in order to run the complete analysis. Following the pipeline, first of all at line *(3)* from the scene are extracted the bounding boxes containing the tools 4.2.2. For each tool so found, the point cloud from the depth image is computed 4.3.1 and the correct classification is performed in order to find out its model name 4.2.3, at line respectively *(6)* and *(7)*. With the object point cloud and once loaded the model point cloud, at line *(9)*, it fits the two point clouds 4.3.2.

Given the model point cloud it is possible to segment it in order to retrieve the parts that compose the object 4.4.2, line *(10)*, and for each part, line *(11)*, it computes the related superquadric 4.4.3. In addition the handle part is computed 5.2.

Finally all the features retrieved during the whole analysis is used to compute the affordances of the tool. Once the iteration on all the tools is completed, it possible to find the best tool given all the affordances, line *(16)* and the selected tool can be grasped.

Algorithm 6: Overall algorithm

```
   // Move the robot to the correct position
1  reachPosition()
2  ack ← waitMotionDone()
   // Multi model approach
3  boundingBoxes ← exloreScene()
4  toolsAffordances ← emptyList()
5  for boundingBox in boundingBoxes do
       // 3D vision analysis
6      objectPointcloud ← readPointcloud()
7      modelName ← getModelName(boundingBox)
8      modelPointcloud ← loadPointcloud(modelName)
9      toolPointcloud ← performIcp(objectPointcloud, modelPointcloud)
       // Grasp the tool
10     segments ← segmentModel(toolPointcloud)
11     superquadrics ← computeSuperquadrics(segments)
12     handle ← fitHandle(segments)
       // Affordances analysis
13     affordances ← computeAffordances()
14     toolAffordances ← insert(affordances, toolPointcloud)
15 end
16 bestTool ← findBestTool(toolAffordances)
17 graspTool(bestTool)
```

Chapter 5

Experiments

In this chapter we enter in the detail of the experiments conducted using the code that implements all the improvements and reasoning expressed in the previous chapters. Firstly, the tests have been performed with the iCub simulator to verify the whole pipeline and the correctness of the actions. Later, the experiments have been moved on the physical iCub robot. Several improvements and parameters tuning have been made in the code once the tests on the real robot have been carried out, the changes will be described in detail in the paragraph 5.2.

Before starting the description of the experiments it is necessary to introduce some tools used in the experiments both in the simulator and on the robot. Since the projects widely uses images captured at run time and analyzed by computer vision algorithms, it is needed to show them. This is possible thanks to the *yarpview* tool provided by YARP, that creates a window and exposes an input port that, once connected to an output stream, can show the images that flow in the connection. For example it is possible to see what the robot visualizes in the simulator and also the real robot, because both provide two output ports that stream images of the scene that are shows with two separate viewers, as visible in the figure 5.1. Also other viewers are used in this project in order to visualize the output of several algorithm steps such as the morphological transformation applied to the scene image, or the tools segmentation from the rack with their bounding boxes.

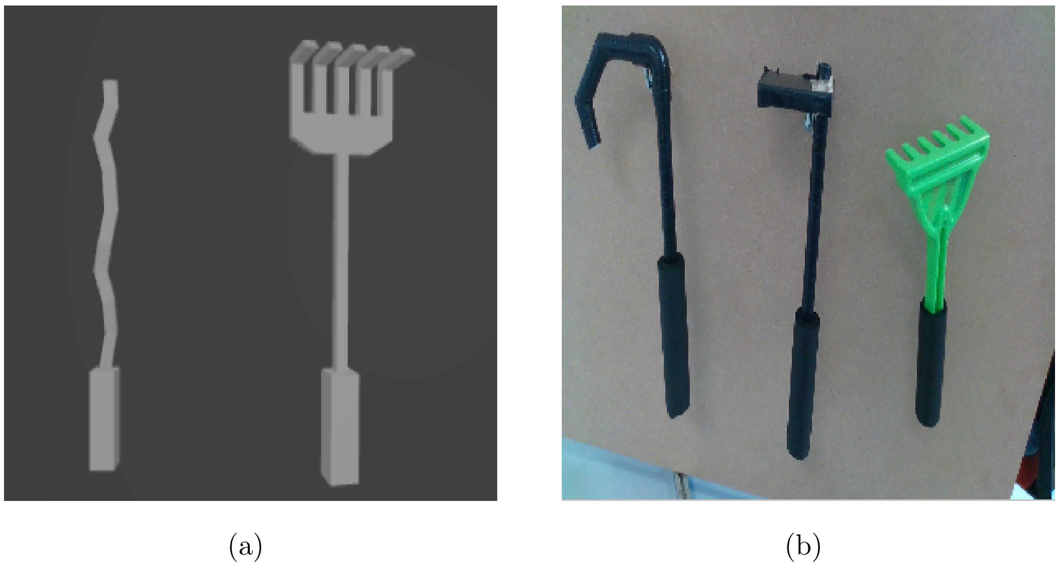


Figure 5.1: The figures show what iCub sees directly from its eyes. The images (a) shows the reconstructed images from the Gazebo simulators. The image (b) streams the outputs of the robot left camera.

Moreover the project takes extensively advantages of point clouds for manipulating tools. For this reason it is also necessary a system able to visualize them and also superquadrics. This viewer is developed thanks to the open-source framework, Visualization Toolkit (VTK) [81], with which is possible to create a window and shows point clouds and superquadrics, as shown in figure 5.2.

The experiments environment consists in the iCub robot, a table on which an object is positioned, the goal of the action is to grasp that object. The robot is positioned in front of the table so, if the object is close enough the robot is able to grab it directly, instead if it is not reachable by the hand it uses a tool to complete its task. The tools are laid on a rack placed on the right side of the robot. This must be near since iCub has to be able to grasp the tools when they are required for task completion. To perform the tests, the described scenario is

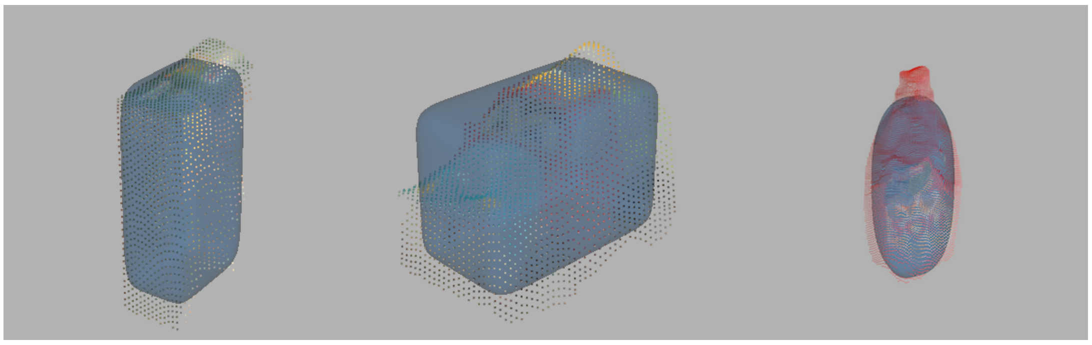


Figure 5.2: VTK viewer. In blue the superquadrics and the color points represent the objects point clouds

reconstructed in the Gazebo simulator and also in a real environment.

5.1 Gazebo

The first approach to experiments has been done by means of a simulator, Gazebo [82], that allows to test algorithms on a robot in a realistic scenario. Indeed, iCub robot model is available and can be used inside Gazebo simulator to emulate the robot with all its joints, links and degrees of freedom. It is possible to command the robot's movements to be performed thanks to several drivers and plugins that emulate the real experience with the robot. It is also possible to insert several objects inside the simulator in order to reconstruct a real environment in which the robot should operate. In our case we have reproduced the scenario previously described with the table and the rack. The target object on the table and the tools in front of the rack.

The tests execution follows the implementation of the improvements so first of all the correct loading of the tools inside the simulator is checked. Also the position is very important because in order to emulate correctly a real scenario they must be placed in front of the rack and the rack itself has to be located not too far so the robot is able to reach and grasp all the tools. At the same time, the rack cannot be placed in front of the robot but on the side because on the table in front of it, where it has to perform the action to displace the target object. Considering all these constrains the starting scenario is provided by the figure 5.3.

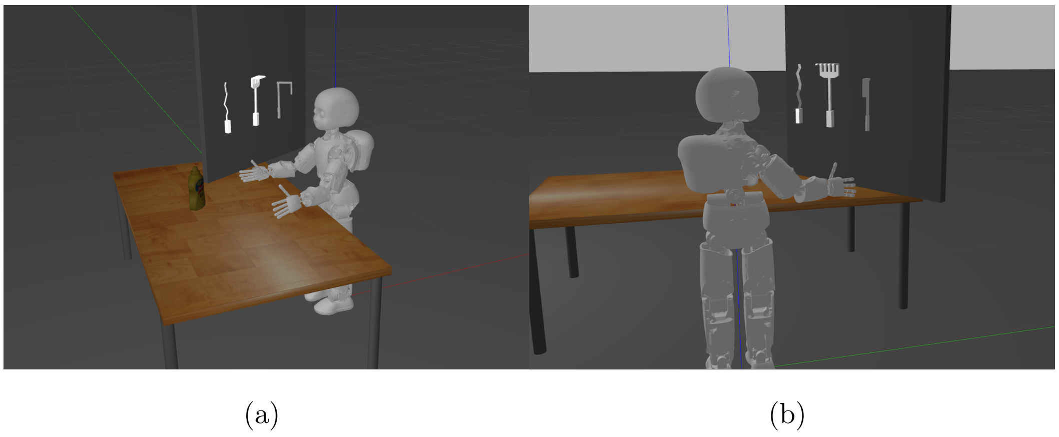

(a) (b)

Figure 5.3: The image shows the simulator containing several objects including the iCub robot, the table, the object and the rack with several tools.

It should be highlighted that the tools are randomly loaded into the simulator at run time and they can be removed to make room for others. So the first check is to verify the correct loading and positioning of the three tools in front of the scene. Since the rack is placed to the right of the robot, it has to firstly rotate the torso and then the head in order to correctly visualize tools putting them into its visual field. The figure 5.4 shows the movements that the robot has to perform in order to arrive to the initial position.

Once the correct position is reached, the tools analysis can start. As mentioned in the section 4.2.2 tools have to be firstly segmented from the surface in background and then they have to be isolated in order to find the region of interest for each tool. The tools segmentation takes in advantage of the differences in colors between the homogeneous background and the colors of the tools. The figure 5.5 shows the binary mask that distinguishes between the tools

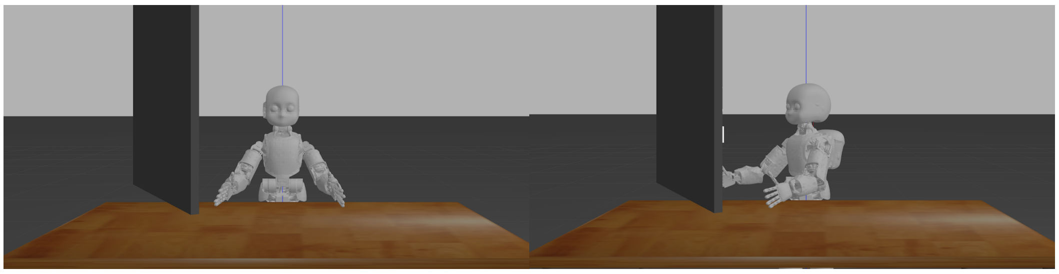


Figure 5.4: The images show the two positions, (a) the initial one with when iCub is loaded that is the *home* position with enlarged arms, and the second one (b) represents the robot after the rotation to reach the correct position to start the analysis.

and the background. From the regions of interest that contains the tools the corresponding bounding boxes are computed.

Once the segmentation is completed, the region of interest related to a tool is used to find the model name associated to that tool. Indeed thanks to the ROI is possible to retrieve the probability distribution of the possible names learned by the robot, and that with the highest score is the candidate model name.

For each tool with a model name associated, it is possible to continue the analysis loading the model point cloud from the correlated stl file. This together with the object point cloud, reconstructed from the depth image, is given as input in the iterative closest point algorithm 4.3.2 that computes the best transformation between the two. The fitted point cloud, then, is segmented into parts 4.4.1 and the superquadric of each segment is computed 4.4.3. The images 5.6 show several iteration of these three steps for different tools.

For each tool segmented its graspability has to be computed. As described in the paragraph 5.2 the part that matches better the cylinder that the hand can contain, is chosen as the handle of the tool. The figure 5.7 shows the selected handle for several tools.

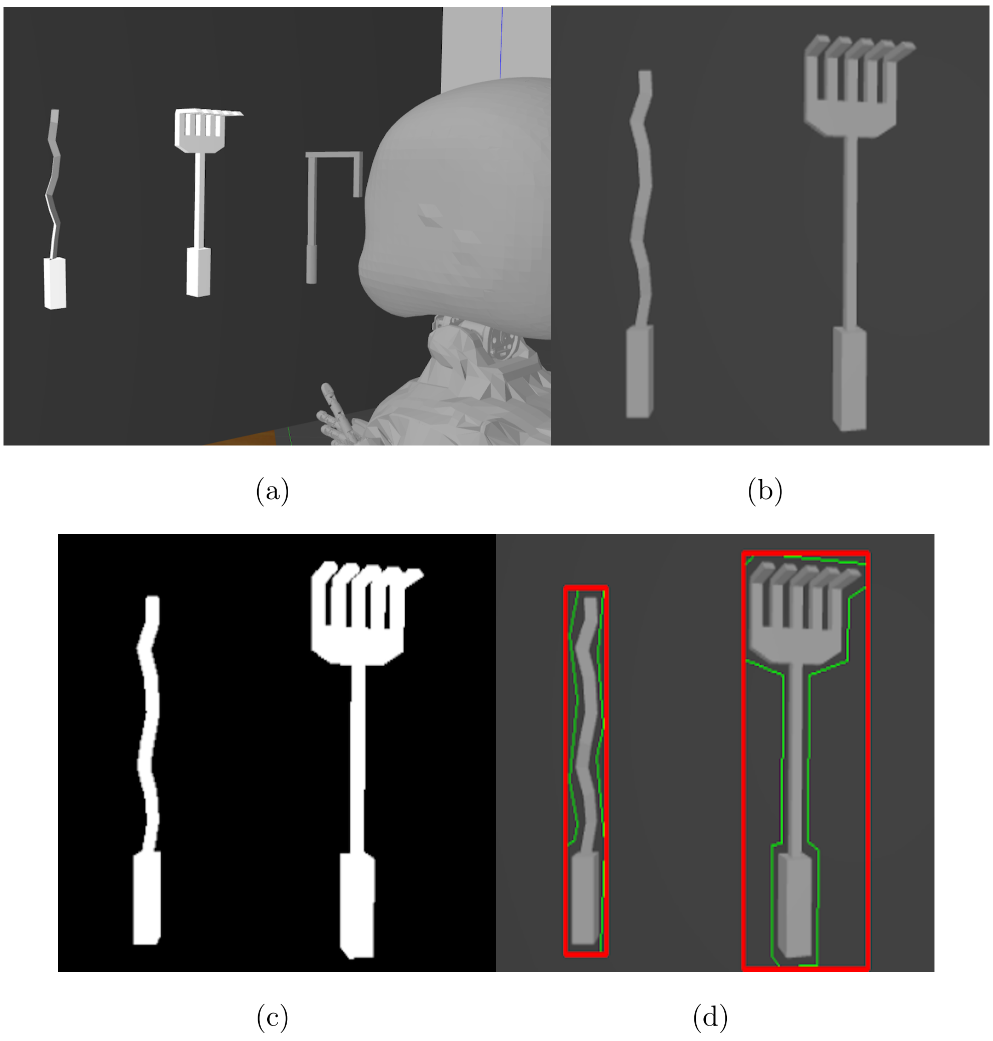


Figure 5.5: The image shows the tools segmentation. From the image (a) it is possible to see the entire scene from which the analysis starts and the image (b) shows the scene from the point of view of the robot. Then the image (c) provides an image of the binary mask that represents the tools in the scene. Finally, the image (d) shows in green the contours of the white region of the mask that represent the tools figured out, and in red the related bounding boxes that are used in the following phases.

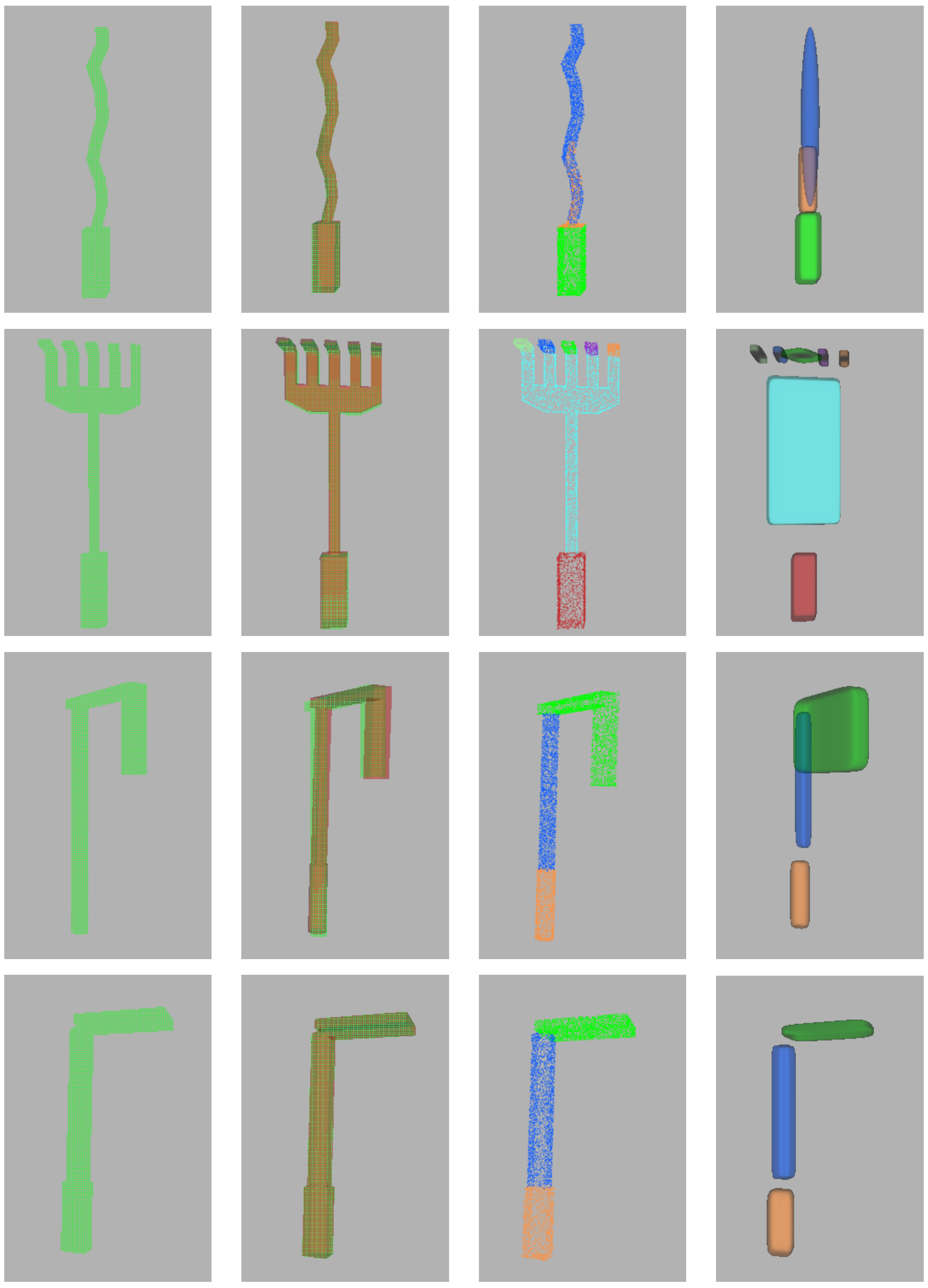


Figure 5.6: The first column contains the point clouds reconstructed from the depth maps, the following one shows the model matching, in green the object point clouds, in red the model ones. The next column shows in different colors the distinct parts in which they are segmented. The last column shows the superquadrics computed from each segment, the relation between the segment and the superquadric is displayed through the color.

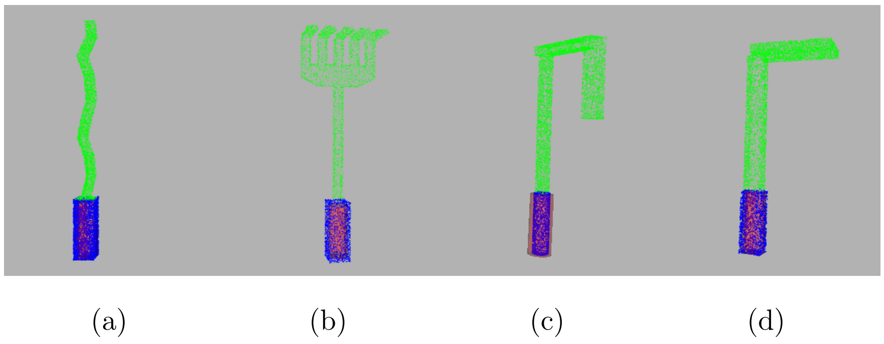


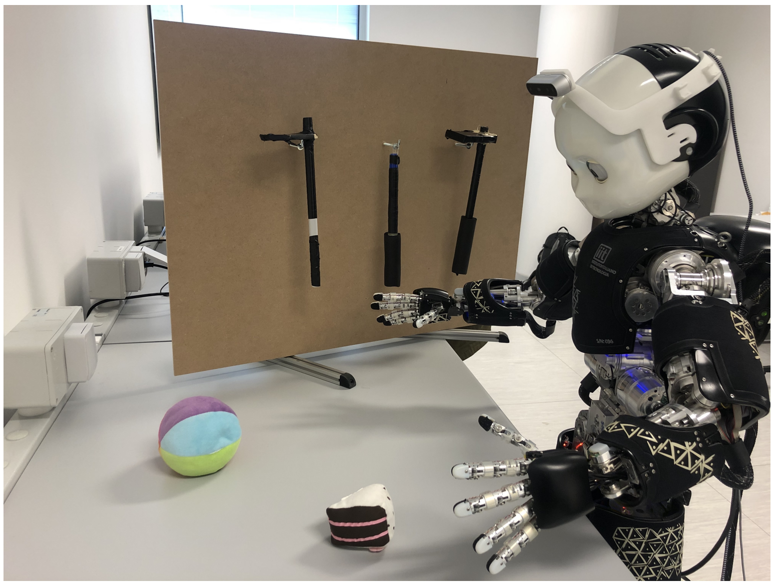
Figure 5.7: The images show the handle selection for the four tools analyzed in figure 5.6. It is possible to see in blue the segment selected as the handle, and in green the other segments. In addition inside the handle it is also present, in red, the model of the cylinder that the iCub had can easily grasp.

Finally affordances of the tools can be calculated, given a desired action. For each tool analyzed until now the affordance is computed as also the effect of the action in term of displacement of the object respect the original position. The tool with the highest value is chosen, it is grasped and the required action can be performed.

5.2 iCub

The experiments on the humanoid robot iCub have followed the same steps through the pipeline of the project. The first phase is to prepare the setup with a table and near the robot. The rack has to be placed to the right of the robot. And finally the object and tools are put respectively on the table and on the rack, figure 5.8.

Subsequently, the robot starts the action of removing from the table the target object. If that is not reachable, it looks at the rack for a tool suitable for the action. Successively, the positions of the torso and the head are adjusted in order to contain in the field of view the rack with the tools, at this point, the tool analysis can start. Firstly, the tools are segmented from the background. Initially, the segmentation was developed considering only the gray scale of the scene image.

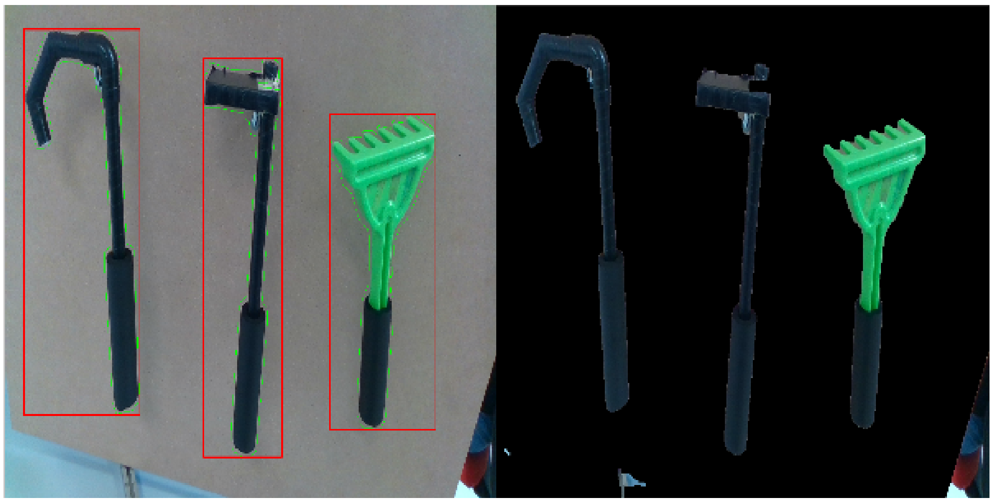


(a)

Figure 5.8: The image shows the setup configured for executing the experiments.

The segmentation worked well in the simulator thanks to the homogeneous colors of the tools and the background. In the real scenario, instead, there are a lot of colors, in addition the shadows can create some noises for the tools segmentation. All these observations led us to reconsider the segmentation algorithm and using the rgb image instead of a grey scale. Using the three channels the precision and the robustness of the algorithm have increased significantly. In the new version of the algorithm the changes in colors and the shadows have been tackled adjusting the thresholds that distinguish the background color from all the other colors. For each tool the related region of interest is computed, figure 5.9.

Afterwards, the tools have to be recognized. The classification is performed thanks to the just computed bounding boxes of the tools, that circumscribe them. From each of them, the associated model name is returned by *IOL* module 4.2.3,



(a) (b)

Figure 5.9: The images show outputs of the segmentation algorithm. The image (a) contains the bounding boxes of the segmented tools in red and a green line defines the contour of the objects. The image (b), instead, shows the tools segmented from the scene image.

that provides the probabilities of matches above all the learnt classes, it is picked the one with the highest value. This matching is necessary for the continuation of the analysis.

The next algorithm to perform is the *iterative closest point* that finds the transformation that best suits a point cloud to another one thanks to a series of translation and rotation. The point clouds are two, one is loaded from the model file thanks to the name previously found. The other point cloud instead is computed from the depth image, if it is not well formed the algorithm is not able to adjust the point clouds.

Firstly we have tried to use the stereo vision to compute a depth image but the results have not been adequate to proceed. For this reason we have decided to use a rgb-d camera that computes directly the depth image thanks to the tof sensor. The cameras used in these experiments is the Intel Real Sense D435. The results with this camera were very satisfactory, because we have also relied on a better resolution respect to the one of cameras incorporated into the eyes, the comparison between the depth images are visible in figure 5.10.

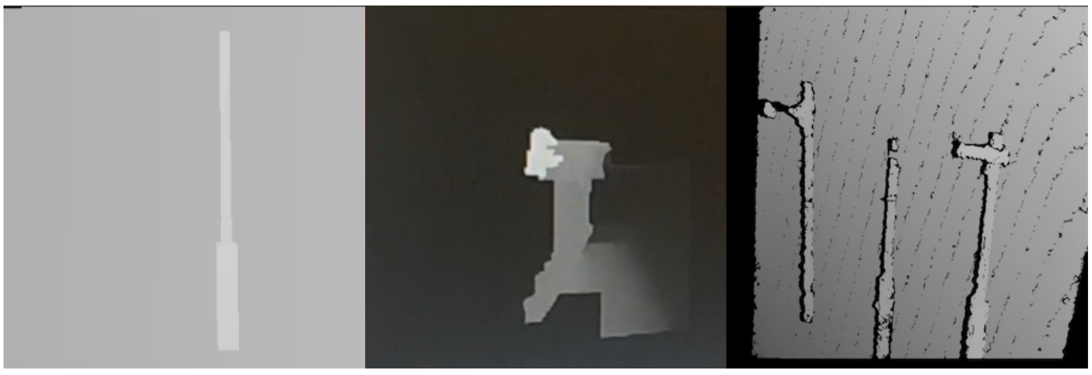

(a) (b) (c)

Figure 5.10: The images show the disparity images. The (a) is the depth image reconstructed in gazebo, instead (b) is the depth image computed by the stereo vision. Finally the image (c) shows the depth image from the depth camera.

Another issue that occurred during the test is related to the scaling factor between the model loaded from the stl file and the point cloud from the depth image. The two point clouds could have different scales, for this reason they are compared in terms of dimension. Once the scaling factor is computed, the model can be scaled in order to have the same dimension of point cloud read. Several executions of the matching between the model and the point cloud can be visualized in the picture 5.11.

Once, the point cloud related to each tool is computed, the algorithm can continue in the same way of the simulator: segmenting each tool and computing the related superquadrics. The following step is the identification of the segment in the object that best performs as handle. As explained in the paragraph , the choice of the handle is taken comparing each segment with the cylinder that can be easily grasped by iCub hand. Successively, the point cloud of the segment is used to find the 3D position of the handle. Indeed, the transformations from the

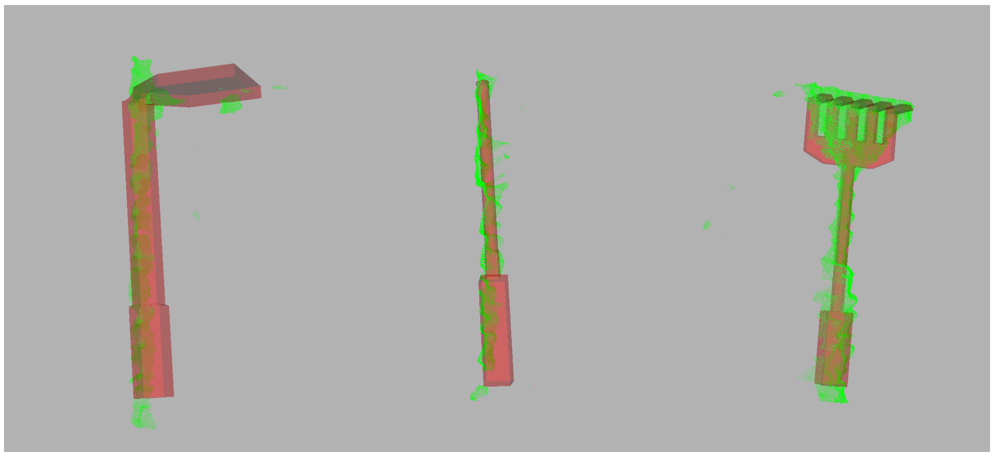


Figure 5.11: The images show different executions of ICP algorithm for three models. In green it is represented the object point cloud, reconstructed from the depth image. In red the model point cloud.

pixel reference to the camera reference is computed thanks to the pinhole camera model. This transformation makes use of the field of view of the camera *fov* to compute the coordinates of each point of the point cloud.

Finally, to obtain the set of points in the robot coordinates the transformation from the camera frame to the robot frame has to be applied. It must be taken into consideration that in the real robot we used the rgb-d camera that is mounted on top of the head of the robot with a helmet. For this reason the camera frame in the real robot is translated respect to the camera frame in the simulator that is attached to the left eye of the robot, the pseudo code 7 shows the steps used to obtain the point cloud in the robot frame, the values of the translation from the depth camera to the left eye are retrieved from the model drawing of the helmet. Taking into account all these differences it is possible to compute the coordinates of the point cloud in the robot frame. This allows to know where the robot has to move the right arm in order to grasp the tool, the figure 5.12 shows the handle of the tool and the movements of the robot done in order to reach the position of the handle and grasp it.

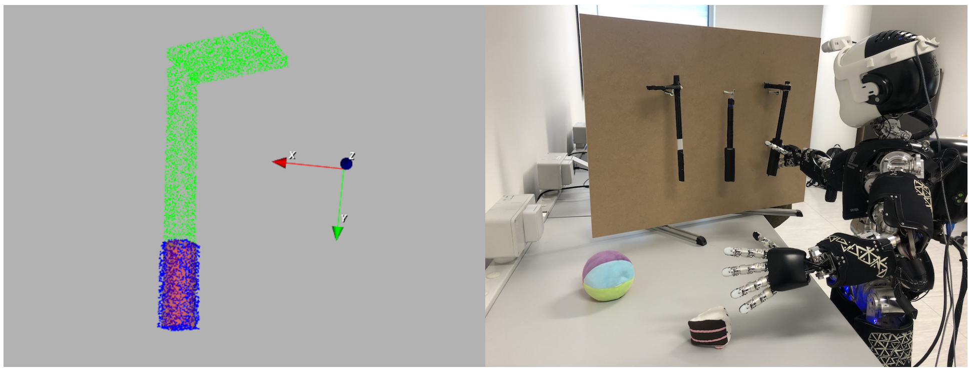
Algorithm 7: Compute point cloud from camera frame to robot frame

```
  Input: Point cloud in camera frame $P_c$
  Result: Point cloud in robot frame $P_r$
1 for point in $P_c$ do
      // Translation from the depth camera to the left eye
2     point.x ← point.x + 0.034 - 0.0175;
3     point.y ← point.y - 0.11564;
4     point.z ← point.z + 0.02673;
      // Transformation from camera frame to robot frame
      // $T_{eye}$ represents the inverse of full-rank extrinsic
         matrix
5     point ← $T_{eye}$ * point
6 end
```

5.3 Results

In this paragraph we exhibit the results obtain from the execution of our code both in the simulated and the real environment. The iterations have the aim to test the code in terms of robustness and the correctness.

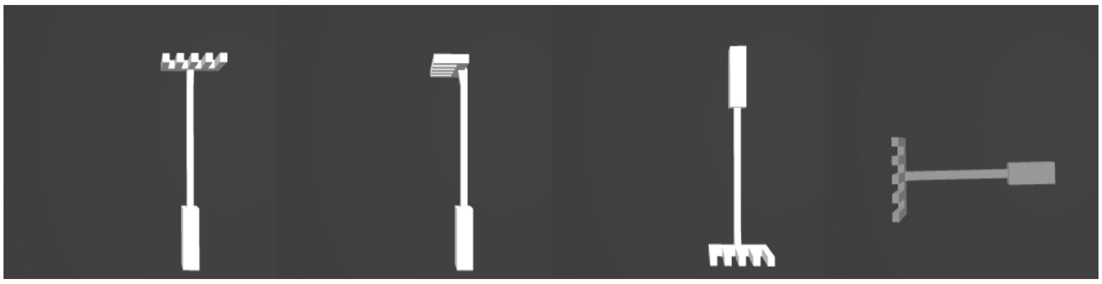
The first test is performed in the simulator and wants to verify the precision of the algorithm, analyzing all the tools in all the available positions in front of the rack. Staying in the simulator, we ran another series of tests. Each tool is loaded in the central position and 3 different rotations are applied, one around the vertical axis of 90 degrees to change the orientation, the other two are around the normal axis, one of 180 degrees to flip the tool the other of 90 degrees to change the main axis. All the transformation are counterclockwise and they are shown



(a) (b)

Figure 5.12: The images show on the left the handle matching of the tool and on the right there is the grasping final action.

in the figure 5.13. Some of them are ineffective on some tools, like the rotation of the stick along the vertical axis.



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

Figure 5.13: The figure (a) shows the original tool, the figure (b) the rotation around the vertical axis of 90 degrees counterclockwise. The images (c) and (d) represent the rotations around the normal axis respectively of 180 and 90 degrees, both counterclockwise.

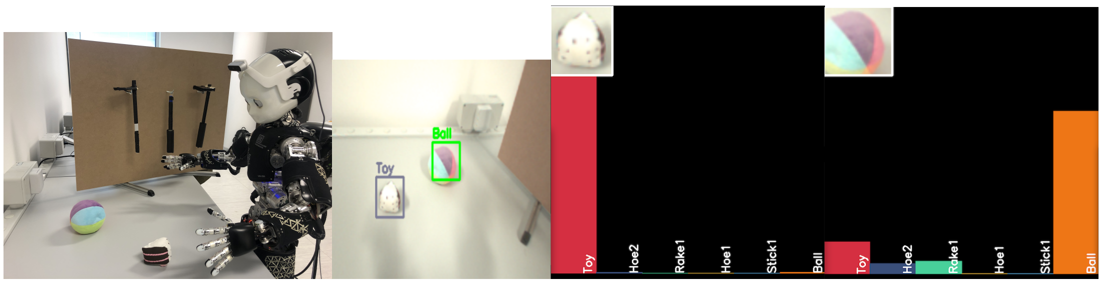
The results take into accounts the final position of the fitted model. If the model is perfectly matched the score is 1 on the other hand if it is totally incorrect the score is 0, but we introduced also the 0.5 value because sometimes the algorithm puts the model in the correct main direction but oriented in the wrong direction, this can happen because the algorithm doesn't converge to the correct model but can also caused by the point cloud read from the depth image, because the orientation of the end effector can be not so visible to make the matching mostly impossible. The table 5.3 shows the scores for each tool and for each experiment. The tools marked with the * symbol have not been used in rotations tests because the models do not have the same center as the others, so applying the same rotations the object is moved outside the scene.

tool name	pos1	pos2	pos3	rot0	rot1	rot2	rot3	overall
hoe	0.5	0.5	1.0	1.0	1.0	1.0	0.0	0.71
hoe1	1.0	1.0	0.5	1.0	0.5	1.0	1.0	0.86
hoe2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
hoe3	1.0	1.0	0.0	0.0	0.5	0.5	1.0	0.57
hoe4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
hoe5	1.0	0.0	0.5	1.0	1.0	0.5	1.0	0.71
hoe6*	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.14
hoe7	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.57
hok1	1.0	0.5	0.0	1.0	1.0	1.0	1.0	0.79
hok2	1.0	1.0	1.0	0.5	1.0	0.5	1.0	0.86
hok3	1.0	1.0	0.5	0.5	1.0	0.5	1.0	0.79
hok4	1.0	1.0	0.0	0.5	1.0	0.5	0.5	0.64
hok5	0.5	0.5	1.0	1.0	1.0	0.5	1.0	0.79
hok6*	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.14
hok7	1.0	0.5	0.0	0.0	1.0	0.5	1.0	0.57
lstick*	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.29
lstick1	0.5	1.0	1.0	1.0	1.0	0.5	1.0	0.86
rak	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
rak1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
rak2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
rak3	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.93
rak4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
rak5	1.0	0.5	1.0	1.0	1.0	1.0	1.0	0.93
stick	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
stick1	1.0	0.0	1.0	1.0	1.0	1.0	0.5	0.79
stick2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
stick3	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.71
stick4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
stick5	1.0	1.0	0.0	0.5	0	1.0	1.0	0.64

5.4 Experiments on the Real Robot

In this section we reports several execution performed on the iCub robot, showing pictures taken to the robot or captured from the software, in order to show the image transformations, point clouds and superquadrics.

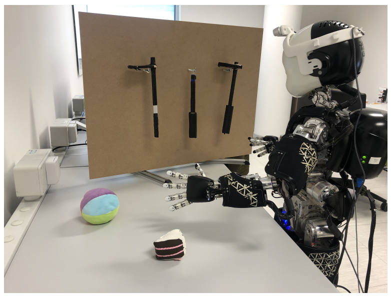
The first action performed by the robot is to look the table and recognizes the toys, figure 5.14, and the ball present in the scene. These two objects are the targets of its actions and the goal is to reach and grasp them.



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

Figure 5.14: The image shows the recognition of the objects above the table.

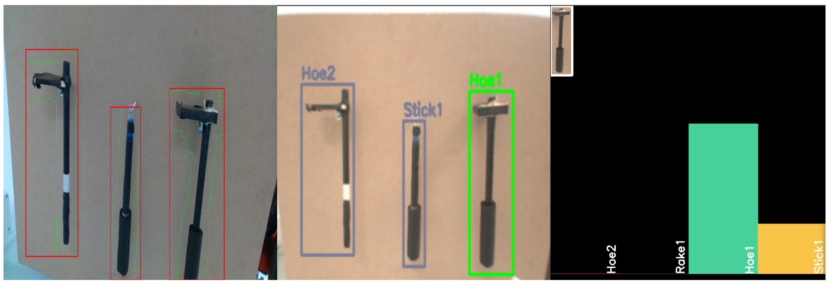
Once the robot visualizes that the *ball* is too far, in order to grasp it, iCub has to use a tool in order to increase the range of its arm. To do that, it rotates the torso and the head to visualize the tools on the rack, figure 5.15.



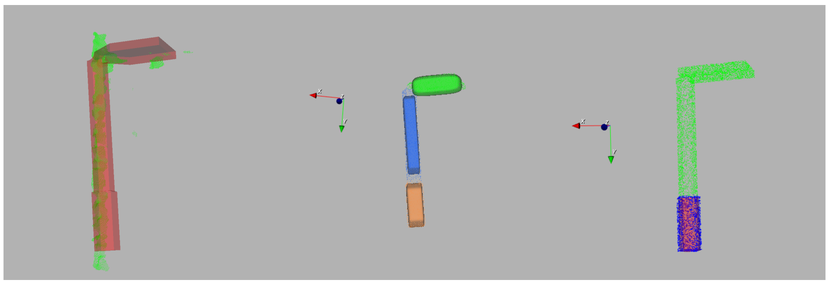
(a)

Figure 5.15: The image shows iCub that looks the rack in which three different tools are available.

Now the analysis, developed in the code 6, can start. Once the image is segmented and the bounding boxes are found, it is possible for each tool to match the correct model retrieving from the classifier the correct model name. Furthermore, for each fitted tool, the segmentation algorithm divides the point clouds into parts from which a superquadric is computed. From all the segments, the most graspable is chosen as handle and the related superquadric is used to compute the 3D position. Finally also the affordances of that tool are computed. All these steps are shown in figure 5.16 for the first tool and in figure 5.17 for the other two tools.



(a) (b) (c)



(d) (e) (f)

Figure 5.16: The images show all the steps performed in the entire pipeline. First of all in image (a) it is possible to visualize the bounding boxes with the object. By using the roi of each tool is possible to retrieve the name of each tool (b). The object on the right is the first one to be analyzed, and the classifier returns the *Hoe1* name as visible in (c). The images (d, e, f) show the tool analysis, model matching, superquadrics computation and handle identification.

The hoe on the right has the highest affordances value so it is selected as the tool used for the action. By knowing the position of the handle, the tool can be grasped and used to push closer the ball, in an area in which the robot is able to grab the object, as shown in figure 5.18. Once the action is performed the tool is left on the table, figure 5.19.

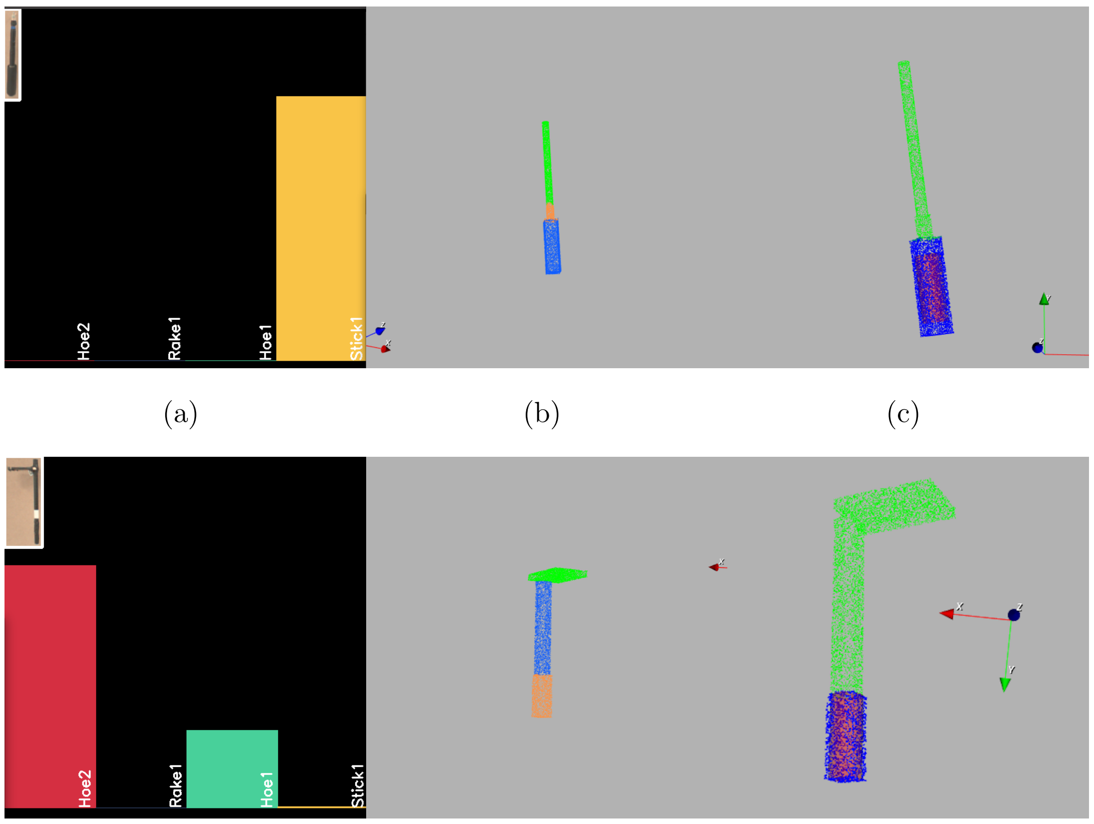


Figure 5.17: The images represent the same analysis computed for the first tool of figure 5.16 also for the other two tools, the stick in the center and the second hoe on the left.

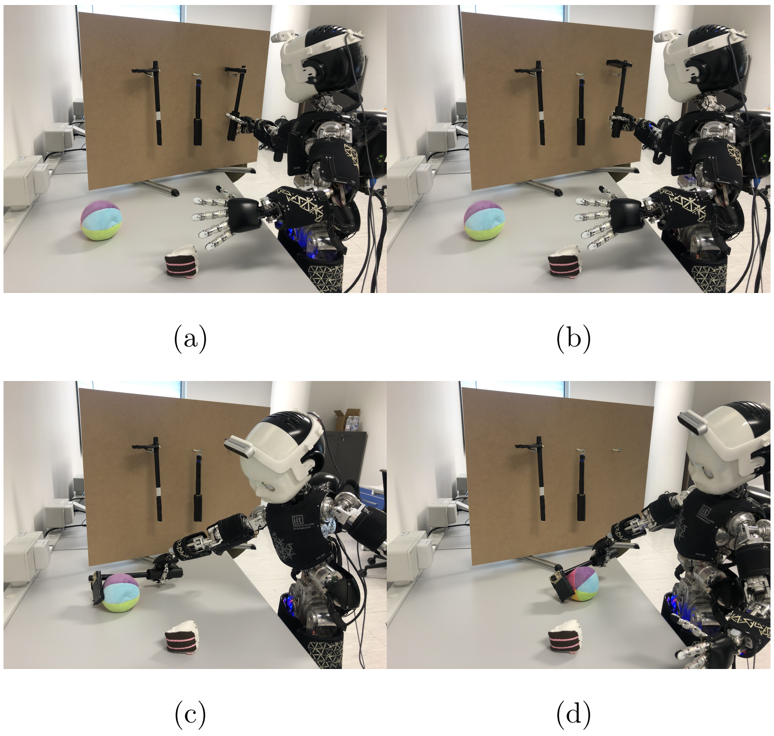


Figure 5.18: The image shows iCub that move the right arm in order to grasp the tool, figure (a). Once done it leaves it a little bit in order to move the object figure (b) and then it performs the action, figures (c) and (d).

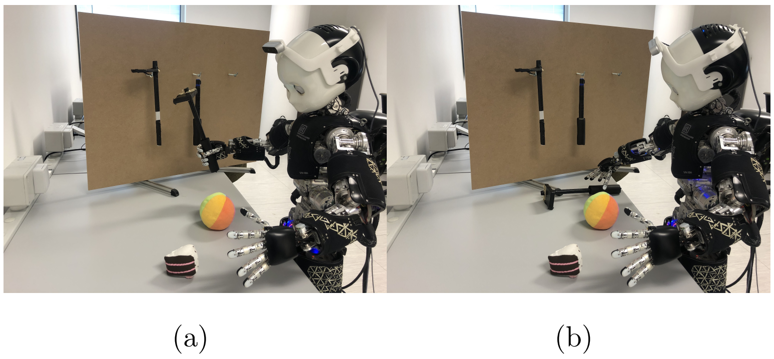
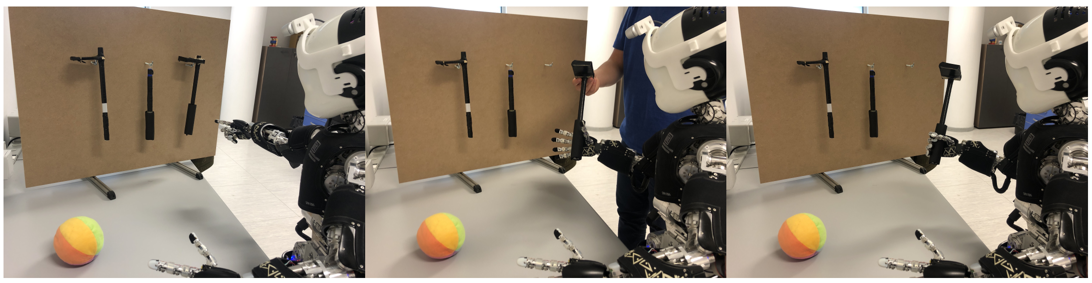


Figure 5.19: In image (a) iCub moves the arm in the home position locating the hand over the table, and in the image (b) it opens the hand in order to leave the object.

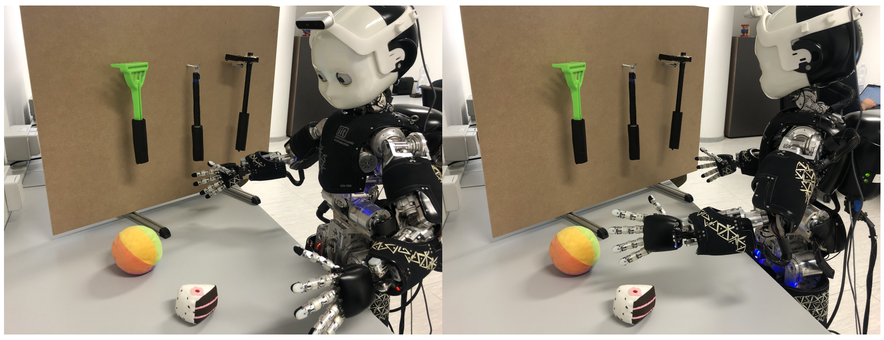
If iCub is not able to reach the tool it asks for an external help, from the human user which has to take the tool from the rack and puts on the iCub hand and wait until it doesn't close completely the hand, figure 5.20.



(a) (b) (c)

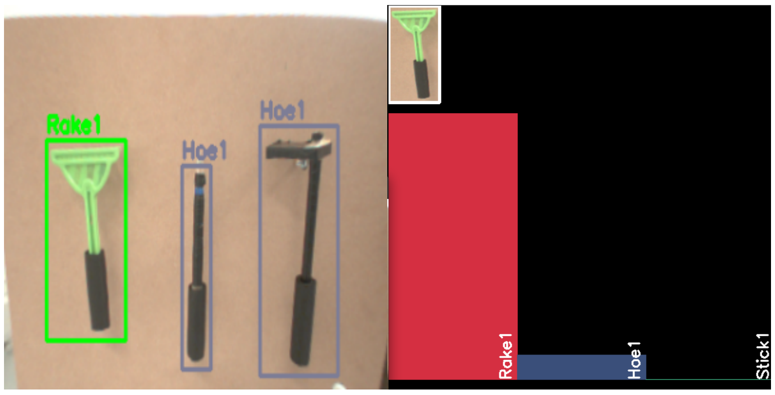
Figure 5.20: The images show the sequence of actions performed if the tool is not reachable. (a) it points the desired tool. (b) it asks and waits for the tool with the hand opened. (c) grasps the tool.

The same pipeline can be executed putting a tool with an higher affordances value. Again iCub after looking the table, moves the head in order to see the rack, figure 5.21. The tools analysis is performed computing, for each tool retrieved by the 2d image segmentation, the model matching, the point cloud segmentation and also the handle fitting, figure 5.22.

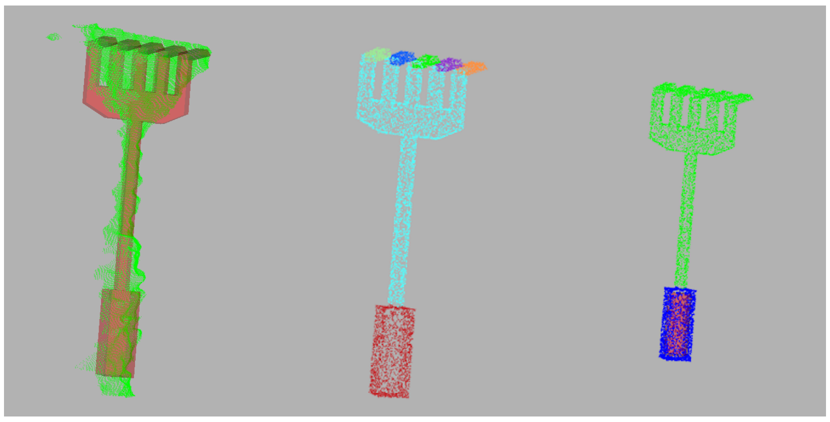


(a) (b)

Figure 5.21: iCub looks before the table and after the rack.



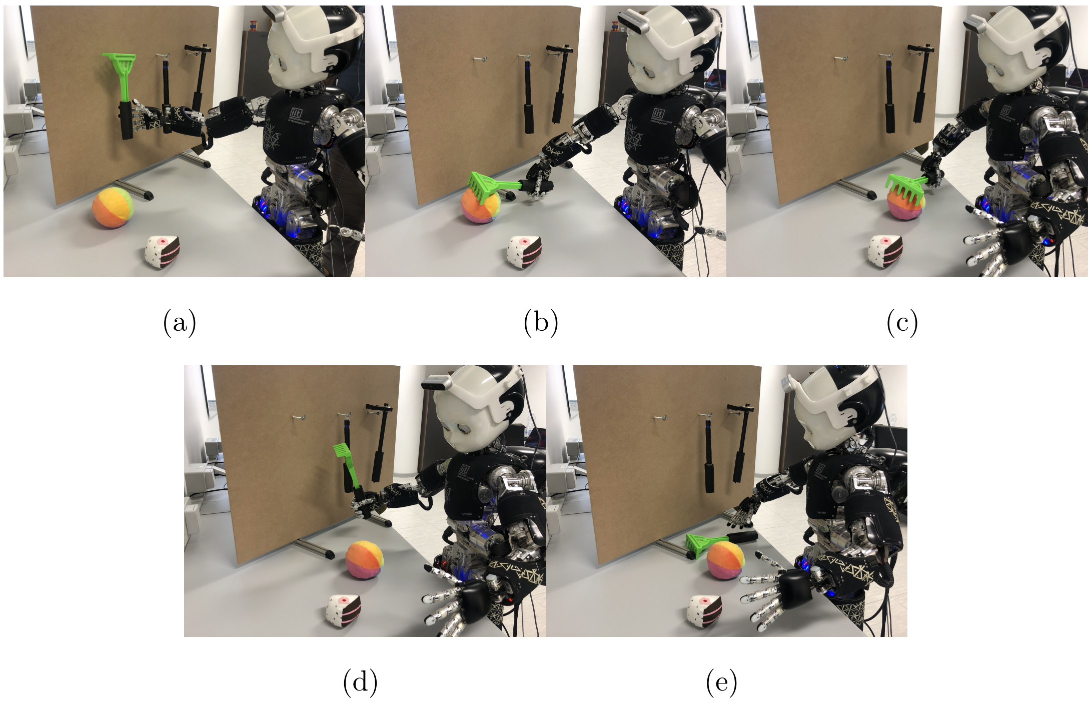
(a) (b)



(c) (d) (e)

Figure 5.22: Image (a) and (b) show the name matching performed by the classifier, and the selected tool is the new one, the *rake*. Images (c), (d) and (e) show the model matching, the segmentation and the handle fitting of the selected tool.

Once the tool is selected, it is grasped and then the action can be performed. At the completion, the tool is left on the table, these steps are shown in figure 5.23.



(a) (b) (c)

(d) (e)

Figure 5.23: The image (a) show the frames in which iCub grasp the tool. Later images (b) and (c) visualize the action performing of pushing the object closer, in a graspable area. And finally, in images (d) and (e) the tool is left on the table.

Chapter 6

Conclusion

With this thesis it has been shown that it is possible to enhance visual competencies of iCub robot for a better computation of tool affordances. For this purpose we started from an IIT's project which dealt with the computation of tool affordances and we have increased and improved the initial analysis that were made on the tool, endowing the robot with the ability to manage a multi model environment, to choose and grasp autonomously the tool that best fits for its goal.

With this work, the robot instead of asking for a specific tool from a predefined set, now is able to observe the scene and detect the tools available in that specific scenario. In addition the analyses conducted on the tool allow to define where and how to grasp the object, predicting its final pose and also the final pose of the end effector that will be used in the execution of the action. These capabilities allow the robot to become more autonomous since it no longer depends on a human. Furthermore it becomes able to decide the final position of the end effector instead of observing how the tool is placed on the robot hand by the user. Indeed, knowing the position of the tool, its shape and the orientation, the robot can adjust the grasping movements in order to grab the tool with the desired position. Another useful aspect to be noted is the information produced by the analyses conducted on the tools to distinguish and recognize them. The extracted data can be provided to the affordances studies to improve the results. Indeed, a large quantities of descriptors, related for instance to the shape of the tool, but also the end effector position compared to handle position, can be provided to the features extractor contained in the affordances algorithms, improving the results and reducing the amount of data to compute. Finally, the proposed method was tested in the Gazebo simulator, but also on a real robot, the humanoid robot iCub. The results confirm that provide the robot with better visual abilities it becomes able to perform actions in a more generic and autonomous way.

6.1 Future Works

The work presented in this thesis can be further extended in the future following these two main limitations.

The idea presented and the related code developed use the dataset described in the section 4.2.1. The execution of the ICP algorithm can be executed only if the point cloud of the model is present in the database. The module employed to classify and identify the tool, is able to provide the name of the object that is more like to the tool segmented from the scene. If the shapes of the two models are similar the ICP algorithm can be able to converge to a solution otherwise it fails. In order to overcome this limitation can be possible to design and develop a module able to learn the point cloud of a model not present in the database.

In addition, it is possible to replace the 2d image segmentation and also the tool classification with a deep learning algorithm. Indeed, neural network are able to takes as input an image and perform the segmentation even if the background is not homogeneous or the tools share the same color with the context. The proposed approach has as assumptions that these two conditions do not occur otherwise the segmentation could be wrong.

Bibliography

[1] E. W. Aboaf, C. G. Atkeson, and D. J. Reinkensmeyer, "Task-level robot learning," in *1988 IEEE International Conference on Robotics and Automation Proceedings*, pp. 1309–1310 vol.2, Apr. 1988.

[2] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson, "Task-level robot learning: juggling a tennis ball more accurately," in *1989 International Conference on Robotics and Automation Proceedings*, pp. 1290–1295 vol.3, May 1989.

[3] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, *Robot Programming by Demonstration.* Springrer, 2008.

[4] A. Skoglund, B. Iliev, B. Kadmiry, and R. Palm, "Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators using Task Primitives," in *2007 International Symposium on Computational Intelligence in Robotics and Automation*, pp. 368–373, June 2007.

[5] J. Norberto Pires, G. Veiga, and R. Araújo, "Programming-by-demonstration in the coworker scenario for SMEs," *Industrial Robot: An International Journal*, vol. 36, pp. 73–83, Jan. 2009. Publisher: Emerald Group Publishing Limited.

[6] A. A. N. Kumaar and S. TSB, "Mobile Robot Programming by Demonstration," in *2011 Fourth International Conference on Emerging Trends in Engineering Technology*, pp. 206–209, Nov. 2011. ISSN: 2157-0485.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, pp. 469–483, May 2009.

[8] S. Ekvall and D. Kragic, "Robot Learning from Demonstration: A Task-level Planning Approach," *International Journal of Advanced Robotic Systems*, vol. 5, p. 33, Sept. 2008. Publisher: SAGE Publications.

[9] A. León, E. F. Morales, L. Altamirano, and J. R. Ruiz, "Teaching a Robot to Perform Task through Imitation and On-line Feedback," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications* (C. San Martin and S.-W. Kim, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 549–556, Springer, 2011.

[10] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, HRI '07, (New York, NY, USA), pp. 49–56, Association for Computing Machinery, Mar. 2007.

[11] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with EM-based Reinforcement Learning," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3232–3237, Oct. 2010. ISSN: 2153-0866.

[12] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schröder, and R. Dillmann, "Toward humanoid manipulation in human-centred environments," *Robotics and Autonomous Systems*, vol. 56, pp. 54–65, Jan. 2008.

[13] M. Cutkosky, "On grasp choice, grasp models, and the design of hands

for manufacturing tasks," *Robotics and Automation, IEEE Transactions on*, vol. 5, pp. 269–279, July 1989.

[14] R. Bormann, B. F. de Brito, J. Lindermayr, M. Omainska, and M. Patel, "Towards Automated Order Picking Robots for Warehouses and Retail," in *Computer Vision Systems* (D. Tzovaras, D. Giakoumis, M. Vincze, and A. Argyros, eds.), Lecture Notes in Computer Science, (Cham), pp. 185–198, Springer International Publishing, 2019.

[15] F. Cini, V. Ortenzi, P. Corke, and M. Controzzi, "On the choice of grasp type and location when handing over an object," *Science Robotics*, vol. 4, Feb. 2019. Publisher: Science Robotics Section: Research Article.

[16] G. Du, K. Wang, S. Lian, and K. Zhao, "Vision-based Robotic Grasping From Object Localization, Object Pose Estimation to Grasp Estimation for Parallel Grippers: A Review," *Artificial Intelligence Review*, Aug. 2020.

[17] J. J. Gibson, *The Ecological Approach to Visual Perception.* Psychology Press, 1° edizione ed., 1979.

[18] J. J. Gibson, *The Senses Considered as Perceptual Systems.* Houghton Mifflin, 1966.

[19] J. J. Gibson, R. Shaw, and J. Bransford, *Perceiving, Acting and Knowing: Toward an Ecological Psychology.* Hillsdale, N.J. : New York: Routledge, 1977.

[20] V. Gallese, L. Fadiga, L. Fogassi, and G. Rizzolatti, "Action recognition in the premotor cortex," *Brain*, vol. 119, no. 2, pp. 593–609, 1996.

[21] J. Piaget, "La naissance de l'intelligence chez l'enfant," 1936.

[22] J. Piaget, "La construction du réel chez l'enfant.," 1937.

[23] R. Shaw and J. Bransford, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology.* Lawrence Erlbaum Associates, 1977.

[24] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrčen, A. Agostini, and R. Dillmann, "Object–Action Complexes: Grounded abstractions of sensory–motor processes," *Robotics and Autonomous Systems*, vol. 59, pp. 740–757, Oct. 2011.

[25] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning Object Affordances: From Sensory–Motor Coordination to Imitation," *IEEE Transactions on Robotics*, vol. 24, pp. 15–26, Feb. 2008. Conference Name: IEEE Transactions on Robotics.

[26] J. Turski, "A Geometric Theory Integrating Human Binocular Vision with Eye movement," *bioRxiv*, p. 2020.09.03.280248, Sept. 2020. Publisher: Cold Spring Harbor Laboratory Section: New Results.

[27] D. Forsyth and J. Ponce, *Computer vision: a modern approach.* Boston: Pearson, 2nd ed ed., 2012.

[28] D. Murray and J. J. Little, "Using Real-Time Stereo Vision for Mobile Robot Navigation," *Autonomous Robots*, vol. 8, pp. 161–171, Apr. 2000.

[29] R. Lagisetty, P. N K, R. Padhi, and M. Bhat, "Object detection and obstacle avoidance for mobile robot using stereo camera," pp. 605–610, Aug. 2013.

[30] G. D. Hager, Wen-Chung Chang, and A. S. Morse, "Robot hand-eye coordination based on stereo vision," *IEEE Control Systems Magazine*, vol. 15, pp. 30–39, Feb. 1995. Conference Name: IEEE Control Systems Magazine.

[31] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D Mapping With an RGB-D Camera," *IEEE Transactions on Robotics*, vol. 30, pp. 177–187, Feb. 2014. Conference Name: IEEE Transactions on Robotics.

[32] Z. Zhang, “Microsoft Kinect Sensor and Its Effect,” *IEEE MultiMedia*, vol. 19, pp. 4–10, Feb. 2012. Conference Name: IEEE MultiMedia.

[33] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments,” *The International Journal of Robotics Research*, vol. 31, pp. 647–663, Apr. 2012. Publisher: SAGE Publications Ltd STM.

[34] A. Garcia-Garcia, *Towards a real-time 3D object recognition pipeline on mobile GPGPU computing platforms using low-cost RGB-D sensors*. PhD thesis, Sept. 2015.

[35] L. Li, “Time-of-Flight Camera - An Introduction,” 2014.

[36] “From Point Cloud to Grid DEM: A Scalable Approach | SpringerLink.”

[37] T. Schenk, “Introduction to Photogrammetry,” p. 100.

[38] Barr, “Superquadrics and Angle-Preserving Transformations,” *IEEE Computer Graphics and Applications*, vol. 1, pp. 11–23, Jan. 1981. Conference Name: IEEE Computer Graphics and Applications.

[39] P. Ferreira, “Sampling Superquadric Point Clouds with Normals,” *arXiv:1802.05176 [cs]*, Feb. 2018.

[40] A. Jaklic, A. Leonardis, F. Solina, and F. Solina, *Segmentation and Recovery of Superquadrics*. Springer Science & Business Media, Sept. 2000.

[41] F. Solina and R. Bajcsy, “Recovery of parametric models from range images: the case for superquadrics with global deformations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 131–147, Feb. 1990. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[42] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A Survey of Surface Reconstruction from Point Clouds," *Computer Graphics Forum*, vol. 36, no. 1, pp. 301–329, 2017.

[43] The CGAL Project, *CGAL User and Reference Manual.* CGAL Editorial Board, 5.2 ed., 2020.

[44] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," p. 10.

[45] J. Digne, J.-M. Morel, C.-M. Souzani, and C. Lartigue, "Scale Space Meshing of Raw Data Point Sets," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1630–1642, 2011.

[46] D. Cohen-Steiner and F. Da, "A greedy Delaunay-based surface reconstruction algorithm," *The Visual Computer*, vol. 20, pp. 4–16, Apr. 2004.

[47] C.-H. Chuang, J.-W. Hsieh, C.-C. Lee, Y.-N. Chen, and L.-W. Tsai, "Human Body Part Segmentation of Interacting People by Learning Blob Models," *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2012.

[48] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *The Visual Computer*, vol. 24, p. 249, Jan. 2008.

[49] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, Feb. 1992. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[50] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The iCub humanoid robot: an open platform for research in embodied cognition," in

Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems - PerMIS '08, (Gaithersburg, Maryland), p. 50, ACM Press, 2008.

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

[52] D. Vernon, C. Hofsten, and L. Fadiga, *A Roadmap for Cognitive Development in Humanoid Robots*, vol. 11. Jan. 2011.

[53] S. R. Fanello, U. Pattacini, I. Gori, V. Tikhanoff, M. Randazzo, A. Roncone, F. Odone, and G. Metta, "3D stereo estimation and fully automated learning of eye-hand coordination in humanoid robots," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 1028–1035, Nov. 2014.

[54] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, Mar. 2006.

[55] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini, "Learning about objects through action - initial steps towards artificial cognition," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, pp. 3140–3145 vol.3, Sept. 2003.

[56] C. Geib, K. Mourao, R. Petrick, N. Pugeault, M. Steedman, N. Krueger, and F. Worgotter, "Object Action Complexes as an Interface for Planning and Robot Control," p. 6.

[57] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Modeling affordances using Bayesian networks," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4102–4107, Oct. 2007. ISSN: 2153-0866.

[58] E. Ugur, E. Sahin, and E. Oztop, "Predicting future object states using learned affordances," pp. 415–419, Sept. 2009.

[59] E. Ugur, E. Oztop, and E. Sahin, "Goal emulation and planning in perceptual space using learned affordances," *Robotics and Autonomous Systems*, vol. 59, pp. 580–595, July 2011.

[60] B. Ridge, D. Skočaj, and A. Leonardis, "Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems," in *2010 IEEE International Conference on Robotics and Automation*, pp. 5047–5054, May 2010.

[61] B. Ridge, A. Leonardis, A. Ude, M. Deniša, and D. Skočaj, "Self-Supervised Online Learning of Basic Object Push Affordances," *International Journal of Advanced Robotic Systems*, vol. 12, p. 24, Mar. 2015. Publisher: SAGE Publications.

[62] J. Sinapov and A. Stoytchev, "Detecting the functional similarities between tools using a hierarchical representation of outcomes," in *2008 7th IEEE International Conference on Development and Learning*, pp. 91–96, Aug. 2008.

[63] V. Tikhanoff, U. Pattacini, L. Natale, and G. Metta, "Exploring affordances and tool use on the iCub," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 130–137, Oct. 2013.

[64] R. Jain and T. Inamura, "Learning of Tool Affordances for autonomous tool manipulation," in *2011 IEEE/SICE International Symposium on System Integration (SII)*, pp. 814–819, Dec. 2011.

[65] A. Gonçalves, G. Saponaro, L. Jamone, and A. Bernardino, "Learning visual affordances of objects and tools through autonomous robot exploration,"

in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 128–133, May 2014.

[66] T. Mar, V. Tikhanoff, G. Metta, and L. Natale, "Self-supervised learning of grasp dependent tool affordances on the iCub Humanoid robot," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3200–3206, May 2015.

[67] T. Mar, V. Tikhanoff, G. Metta, and L. Natale, "Multi-model approach based on 3D functional features for tool affordance learning in robotics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 482–489, Nov. 2015.

[68] A. Myers, A. Kanazawa, C. Fermuller, and Y. Aloimonos, "Affordance of Object Parts from Geometric Features," p. 3, 2014.

[69] A. Roy and S. Todorovic, "A Multi-scale CNN for Affordance Segmentation in RGB Images," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), Lecture Notes in Computer Science, (Cham), pp. 186–201, Springer International Publishing, 2016.

[70] M. Schoeler and F. Wörgötter, "Bootstrapping the Semantics of Tools: Affordance Analysis of Real World Objects on a Per-part Basis," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 8, pp. 84–98, June 2016. Conference Name: IEEE Transactions on Cognitive and Developmental Systems.

[71] P. Abelha and F. Guerin, "Transfer of Tool Affordance and Manipulation Cues with 3D Vision Data," *arXiv:1710.04970 [cs]*, Oct. 2017.

[72] A. Stoytchev, "Behavior-Grounded Representation of Tool Affordances," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 3060–3065, Apr. 2005.

[73] B. Bhanu and O. D. Faugeras, "Segmentation of Images Having Unimodal Distributions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, pp. 408–419, July 1982.

[74] D. S and G. Shinde, "An Adaptive Color Image Segmentation," *ELCVIA. Electronic letters on computer vision and image analysis, ISSN 1577-5097, Vol. 5, Nº. 4, 2005, pags. 12-23*, vol. 5, Jan. 2005.

[75] N. Kulkarni, "Color Thresholding Method for Image Segmentation of Natural Images," *International Journal of Image, Graphics and Signal Processing*, vol. 4, Feb. 2012.

[76] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, pp. 328–339, Apr. 1989.

[77] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *arXiv:1612.00593 [cs]*, Apr. 2017.

[78] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," *arXiv:1706.02413 [cs]*, June 2017.

[79] T. Mar, V. Tikhanoff, and L. Natale, "What Can I Do With This Tool? Self-Supervised Learning of Tool Affordances From Their 3-D Geometry," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, pp. 595–610, Sept. 2018. Conference Name: IEEE Transactions on Cognitive and Developmental Systems.

[80] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[81] W. Schroeder, K. Martin, and B. Lorensen, *The visualization toolkit: an object-oriented approach to 3D graphics ; [visualize data in 3D - medical, engineering or scientific ; build your own applications with C++, Tcl, Java or Python ; includes source code for VTK (supports Unix, Windows and Mac).* Clifton Park, NY: Kitware, Inc, 4. ed ed., 2006.

[82] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Sendai, Japan), pp. 2149–2154, Sep 2004.