SAPIENZA
UNIVERSITÀ DI ROMA

Design and Implementation of a Novel Grasping System for a Humanoid Robot

Faculty of Information Engineering, Informatics and Statistics
Master Degree Program in Computer Science

Candidate
Alessio Ferrone
ID number 1751859

Thesis Advisor
Dott. Luigi Cinque

Academic Year 2024/2025

Design and Implementation of a Novel Grasping System for a Humanoid Robot
Master's thesis.. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: ferrone.1751859@studenti.uniroma1.it

"Si dice che il calabrone non potrebbe volare perché il peso del suo corpo è sproporzionato alla portanza delle sue ali. Ma il calabrone non lo sa e vola lo stesso"
- Roberto Saviano

Dedicato al 5 Agosto 2015, quando tutto ancora doveva iniziare, ma già era.

Abstract

In recent years, humanoid robots have become increasingly central to our society thanks to their ability to interact with people in a natural way through voice, gesticulating and trying to show empathy similar to that of a human being. Depending on the social context in which they are placed and the tasks they have to perform, they are designed with specific characteristics and aptitudes rather than others. From assisting the elderly in healthcare facilities to helping with household chores, to entertainment and education of children, these robots are conceived to alleviate or replace human work, allowing individuals to focus on other activities. However, humanoid robots still have many limitations in specific tasks such as autonomous and precise manipulation of physical objects compared to industrial and specialized robots. The objective of this master's thesis is to design and implement an unknown object (of which neither the 3D structure nor its physical properties are known) grasping system for a humanoid robot like Pepper, intended as a companion and entertainment robot, but not designed for specific object manipulation tasks. Pepper is in fact the platform chosen to test the feasibility and robustness of the system in real-life scenarios. The proposed approach combines advanced computer vision techniques like You Only Look Once (YOLO) for object detection with state-of-the-art models for monocular depth estimation like Depth Anything, to calculate the distance of objects from the camera. To plan precise movements to position the hand at the desired location to grasp the target object, in the work are integrated robotic techniques such as inverse kinematics. This method, compared to what is proposed in the literature using Pepper as platform, excludes the use of any additional markers or sensors that could aid in locating objects or determining the position and/or orientation of the end-effector (robot's hand) in space relative to the object to be grasped. The entire task is performed in real-time. The system is implemented in Python, mainly using the Python SDK developed by Aldebaran Robotics and the Robot Operating System (ROS). The system has been validated through a series of experiments conducted in an academic environment (computer labs and offices), representing a typical scenario where the robot could be employed to enhance research and work experiences. The experiments were carried out at different times of the day and in various building environments, using a bottle as the target object (with tests on bottles of different shapes and weights). These experiments were replicated with medications, useful in contexts such as pharmacies or hospitals. The success of these experiments demonstrates that it is possible to implement a reliable object grasping system for Pepper and similar humanoid robots, without the need for additional markers or sensors to complete the task.

Contents

Chapter 1

Introduction

In this chapter, the topics covered in this thesis are introduced. In particular, in Section 1.1, humanoid robots are introduced, highlighting their general characteristics. Section 1.2 illustrates the problem of grasping objects, showing the differences between humans and humanoid robots. Finally, in Section 1.3 and 1.4, the objectives of this thesis are stated and the contribution that this thesis wants to have in the state-of-the-art is highlighted.

1.1 Humanoid robots

Robots have become an unavoidable part of our daily existence, permeating every conceivable context from office spaces to retail environments, museums, hotels, scientific research labs, hospitals, industrial settings, homes, and even agricultural fields. Humanoid robots, specifically designed to emulate human behaviors and actions, generally feature a torso, head, two arms, and two legs, although some models may not include all these elements. In the medical field, these robots are employed in physical rehabilitation programs, aiding patients in recovering mobility through precisely designed exercises. In hospitals and care homes, they serve to entertain patients with the aim of alleviating stress and anxiety. They often participate in therapy sessions where interaction with the robot is leveraged to stimulate positive behavioral or cognitive responses. Robots have taken on significant roles in healthcare, performing tasks like delivering medications, watching over patients' vital signs, and aiding in medical procedures under a doctor's oversight. This assistance helps lessen the burden on medical staff. They also educate patients, explaining their health issues, treatment options, and encouraging healthier living. On the other hand, these robots can have the ability to pick up objects or use tools that allow them to efficiently perform repetitive tasks such as receptionists or workers on a production line. They can also have a companionship and emotional support function for the elderly, children, people with various levels of disabilities or psychological problems and interact with them through group games and conversations. In some contexts, they can also act as teachers, offering educational lessons on various topics. To make all this possible, this type of robot uses sensors of various kinds to measure features of the surrounding world. Through proprioceptive sensors, the robot is able to perceive the position, orientation and speed of its body and joints within the space: among

these we can find accelerometers with which it is possible to measure acceleration, tilt sensors to measure inclination, force sensors positioned in various parts of the robot's body such as the hands, feet, head that can be used to detect the values of the force impressed by external agents and position sensors used to locate the position and orientation of the robot's body. Instead, by using exteroceptive sensors, these robots are able to measure parameters coming from the outside. Among these we find tactile sensors that are installed to collect information on the forces and torques transferred during contact between the robot and objects in the environment; vision sensors that allow the process of capturing data using the electromagnetic spectrum to produce an image from which humanoid robots can extract a lot of important information for recognizing objects, their properties or for mapping the surrounding environment; sound sensors that allow the robot to listen to human speech and sounds in the environment or reproduce them to communicate with the outside world. In addition to many sensors, each humanoid robot is equipped with actuators. Actuators are the motors responsible for the movement of the robot thanks to which it is possible to implement Planning and control for a robot. Planning is the process of planning the different movements and trajectories that the robot must perform. Control, on the other hand, is the process of actually executing what has been planned. Since robots generally operate in complex environments, planning and control may need to take into account self-collision detection, path planning and obstacle avoidance.

1.2 Grasping

Humanoid robots are capable of performing many different tasks very well. However, picking up objects is a particularly arduous task that leads the developer of a possible grasping system for humanoid robots to face numerous challenges. In this regard, a human being is equipped with highly dexterous hands equipped with opposable thumbs that allow it to grasp objects in many different ways, with great precision and from variable angles. On the other hand, a robot, even if with human appearance, is generally equipped with rigid grippers such as simple pincers or in any case with grippers with a limited number of joints that allow it to operate in a narrow range of possible movements, making it difficult to grasp objects of different sizes and shapes. To grasp an object, humans simultaneously use the senses of sight, touch, and proprioception to accurately understand and estimate the size, shape, weight, and texture of the object, while a robot needs cameras and sensors to analyze the surrounding environment and estimate the distance from the object to be grasped, which tend to provide less precise and accurate information. Another aspect to consider is the force needed to hold an object in the hand without dropping it. A person performs this task automatically thanks to the perception sensors present on the skin; for a robot it is much more difficult since it must be equipped with high-precision tactile sensors sensitive to every possible variation in material and shape, which would make it extremely expensive and complex at the hardware level. Therefore, a compromise is needed that reduces costs but at the same time has good level sensors for measurements and that takes into account the different properties of the materials. If we then consider dynamic environments in which objects may be

moving, the robot must be equipped with advanced algorithms to manage situations in which the environment is not static but is continuously evolving, such as algorithms that allow it to avoid obstacles to reach the position of the target object to be grasped. In general, both in static and dynamic environments, it may be necessary to develop complex algorithms to identify objects in three-dimensional space, approach them using computer vision techniques for depth estimation and to manipulate objects in real time using robotic techniques. What essentially differentiates a human being from a humanoid robot in grasping objects is the ability to generalize. A robot, however advanced it may be in terms of hardware and software, has a much less pronounced adaptability to new tasks (such as grasping new objects) than a human being who can easily adapt to new contexts, new requests. A robot needs the integration of new information and new programming, so reaching the levels of flexibility of humans remains a complex task. After having examined the issues and the complexity of the task, it is now possible to start describing the work proposed in this thesis. This work aims to combine computer vision techniques to extract features related to the position, the orientation of the object in space and its distance from the camera with robotics techniques to plan and execute the trajectories and specific movements to reach the target object, exploiting the information extracted and provided by the aforementioned computer vision algorithms. This proposal was born with the idea of making a humanoid robot capable of acting as an assistant within an office that can be of help in carrying out a series of tasks that are often boring, repetitive and time-wasting for those who have to carry them out. The goal is to obtain a robot capable of performing and completing with excellent probability of success an activity, as simple for humans as complex for a robot, such as locating, retrieving and transporting objects within a room. This type of application also fits perfectly with the needs of healthcare environments such as hospitals, pharmacies or nursing homes by assisting nurses or social-health workers in routine tasks such as transporting medicines or essential goods. A robot with these characteristics can be placed in shopping centers for the transport and placement of commercial material in the appropriate spaces of the shelves, inside industries to carry out manual activities that replace human intervention or in risky and dangerous environments for humans such as in nuclear power plants to move radioactive material or to come into contact with harmful substances. In this regard, many humanoid robots have been developed with the primary objective of successfully executing tasks such as object grasping. However, the aim of this thesis is to design and implement a grasping system that achieves good performance on a non-conventional humanoid robot for the specific task of object manipulation. Beyond functionality, this robot must also embody an "aesthetic" quality. For integration into workplace settings, hospitals, or care homes, it should not merely resemble an inanimate machine but rather feature a face that mimics human expressions and a body with soft, rounded contours to convey empathy. This design approach seeks to ensure the robot is perceived not just as a tool, but as a companionable presence in environments where human interaction is paramount. For these reasons, the Pepper robot developed by Aldebaran robotics (now know as Softbank robotics), a leader in humanoid robots, was identified and selected, as it displays all the characteristics sought.

1.3 Aims and scope

The main objective of this thesis is to develop a grasping system for a humanoid robot, designed to operate in indoor environments, with the aim of allowing the robot to grasp objects whose three-dimensional structure and physical properties are unknown. The research aims to identify the most effective techniques to calculate the distance between the robot and the object to be grasped, designing algorithms that allow the robot to approach the target object and to precisely adjust the movement of the arm and hand to the exact point where the end-effector (the robot's hand) must be positioned to successfully complete the grasping and lifting of the object. Once the algorithms necessary for the complete execution of the task have been developed, this thesis aims to illustrate in detail the fundamental steps for the effective implementation of the system, which must work in real time on a physical humanoid robot. The robot chosen as a reference platform to test the proposed approach is Pepper, a humanoid robot developed by Aldebaran Robotics. The decision to use Pepper is motivated by two main reasons. First, Pepper was already available at the Computer Vision Laboratory, the VisionLab[1], of the department, thus representing a stimulating challenge and a concrete opportunity to develop and validate the system in a real-world setting. Second, the development of a grasping system for a robot like Pepper, characterized by arms with only 5 degrees of freedom and designed primarily as a companion and entertainment robot rather than for precise object manipulation, represents a potentially significant contribution to the state-of-the-art. In fact, in the literature there are few studies that address the grasping problem on humanoid robots with characteristics similar to those of Pepper, especially without the aid of additional markers or sensors. Therefore, the success of this project not only demonstrates the feasibility of a reliable grasping system in such conditions, but also paves the way for further developments and applications for similar humanoid robots in real-world settings, such as laboratories, offices or healthcare settings. In particular, this thesis presents and tests the main functions that allow a physical Pepper specimen to move within a computer laboratory and to successfully grasp a detected object, thus helping to fill a gap in the current research landscape.

This work therefore focuses on achieving the following objectives:

- A careful and detailed study on which are the best approaches and methodologies used in literature to solve the grasping problem in humnaoid robots.
- Development and design of a system that allows a humnaoid robot to reach and grasp a target object
- Implementation of the system on the physical robot that moves in an indoor environment that considers different environmental situations in terms of room size, lighting, shape and weight of the object.

1.4 Contribution and thesis outline

The main contribution of this thesis is the development of an innovative single-handed grasping system for humanoid robots. The thesis stands out for having designed and

implemented a system that allows a humanoid robot with only 5 degrees of freedom in its arm to grasp objects in real time, without the aid of additional markers or sensors beyond those already present by default. This approach represents a step forward in the state-of-the-art, since the existing literature on robots with Pepper-like characteristics offers few examples of such advanced autonomous grasping systems, especially in the absence of external tools for object localization or the end-effector. The proposed method integrates cutting-edge computer vision techniques, such as You Only Look Once (YOLO) for object detection and Depth Anything V2 for monocular depth estimation, with advanced robotics algorithms, such as inverse kinematics optimized for a 5-degree-of-freedom arm. This combination allows the humanoid robot to calculate the spatial coordinates of the target objects and plan precise arm movements to perform the grasp, all by processing in real time the visual data from its cameras. The choice to exclude additional sensors not only makes the system lighter and more versatile, but also demonstrates its applicability to general-purpose humanoid robots with limited hardware resources.

The thesis is organized in the following chapters:

- **Chapter 2:** the fundamental notions for the development of a general grasping system are described and analyzed, including forward and inverse kinematics. A digression on the most relevant humanoid robots in the literature and an analysis of the state-of-the-art of grasping approaches for this type of robot are included.
- **Chapter 3:** the reference platforms used are presented, with particular attention to the Pepper robot, used to validate the proposed methodology, and to the different frameworks that contributed to the final implementation of the system.
- **Chapter 4:** the system proposed in this thesis is illustrated, focusing on its structure and describing in detail the artificial vision models used, the coordinate mapping and the adopted kinematics algorithms.
- **Chapter 5:** the experiments conducted to validate the proposed approach are shown and analyzed, with a quantitative and qualitative evaluation of the results obtained.
- **Chapter 6:** the conclusions are drawn, outlining possible future developments of the system and reflections on its potential and limits.

Chapter 2

State of the art

In this chapter, we explore the state-of-the-art developments that contributed to the design and implementation of the grasping system presented in this thesis. Section 2.1 outlines the subsystems that typically constitute a grasping system. Sections 2.2 and 2.3 introduce key robotics concepts—such as transformations, forward kinematics, and inverse kinematics—that are fundamental to many tasks, including grasping. Section 2.4 provides an in-depth study of prominent humanoid robots from the literature, focusing on their characteristics and capabilities. Finally, Section 2.5 examines various approaches and techniques used to design and implement grasping systems, with specific reference to the platform discussed in this state-of-the-art review.

2.1 General Design of a Grasping System

In the paper [2] Dong, Minglun and Zhang, Jian explain how generally an object grasping system for a complete generic robot has a structure composed of 3 subsystems. As shown in Fig. 2.1, this subdivision is tripartite in the following way:

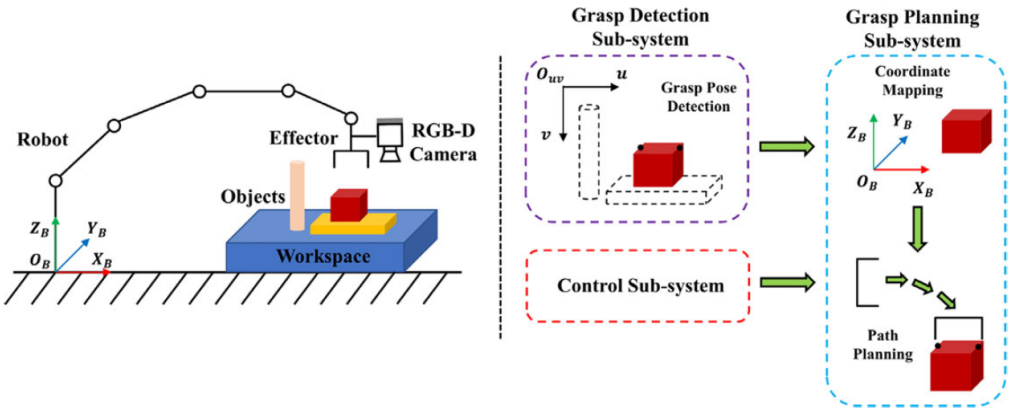


Figure 2.1 A general object grasping system composed of 3 subsystem [2].

- **Grasp detection subsystem:** it is responsible for detecting the target object from the images and establishing the position and pose of the object in the image coordinate system.
- **Grasp planning subsystem:** it is used to map the detected image plane coordinates to the robot base coordinate system and create a consistent path from the manipulator to the object to be grasped.
- **Control subsystem:** its purpose is to calculate the solution of the inverse kinematics of the previous system. Based on the solution obtained, it controls the robot to grasp the object.

After describing how a typical object grasping system is organized, it is necessary to explain three fundamental concepts in robotics: transformations, forward kinematics and inverse kinematics

2.2 Transformations

As a first step, let's define what a frame is. A frame is a coordinate axis attached to a body to describe its position and orientation on a two-dimensional plane or in three-dimensional space. The Fig. 2.2 shows the same object with a 2D and a 3D reference frame.

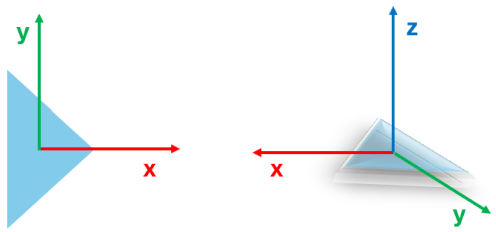


Figure 2.2 The same object in 2D and a 3D.

In a grasping system for robots, we normally move within a three-dimensional space, and that's why we deal with 3D transformations. However, a 2D transformation is nothing more than a subset of the problem to be solved for a 3D transformation [3]. A 3D transformation in three-dimensional space is the change of position and orientation of a frame associated with a rigid body (its size and shape remain unchanged during movement) with respect to another frame attached to another object (an object can be a generic object, a robotic arm, any part of the robot's body, or even a simple point in space). Let us define a reference frame as the frame with respect to which we want to consider the transformation that tells us where a target frame that moves with respect to the reference frame is located [4]. A transformation is simply the combination of a translation and a rotation. A translation is the linear movement of a rigid body along the 3 Cartesian axes from a starting position to a target position. Let us define this translation as a 3 x 1 vector defined as follows:

$$t = (x, y, z)^T \tag{2.1}$$

Let's instead define a rotation R as a 3 x 3 orthogonal matrix that represents the rotation of a rigid body around a Cartesian axis [5]. If we mix the two definitions, it is possible to define a transformation T as a homogeneous matrix defined as follows:

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{2.2}$$

where R represents the rotation matrix, t is the translation vector and the last row [0,0,0,1] represents the row vector that makes the transformation expressible in homogeneous coordinates [6]. In this way, every generic point p_{hom} can be expressed in homogeneous coordinates of the type

$$p_{\text{hom}} = (x, y, z, 1)^T \tag{2.3}$$

Applying the T-transformation on this point, the transformed point p'_{hom} is given by the equation:

$$p'_{\text{hom}} = T p_{\text{hom}} \tag{2.4}$$

2.3 Forward and inverse kinematics

To explain what both forward and inverse kinematics are, it is essential to first define some fundamental components of a robot. An explanation of each follows. Links are rigid parts of a robot's body that essentially give it its structure. Joints are the robot's actuators that assemble and connect the links. They produce relative motion between adjacent links. The most common joints are divided into two categories: a revolute joint is a hinge joint that rotates along its axis taking into account the range within which the value (unit of measurement expressed in radians or degrees) of the rotation must be, while a prismatic joint is a sliding joint that slides with a linear motion along its axis (unit of measurement expressed in meters). Degrees of freedom are the total number of independent joints that influence the motion and pose of a robot. A kinematic chain is a connected series of links and joints with the purpose of generating a specific motion. The end-effector is the terminal part of a kinematic chain that interacts with the environment, which can be a robotic hand, a gripper or any other manipulator. The workspace represents the space in which the robot can perform its activities. In general, it constitutes all the points reachable by the different kinematic chains of the robot. It is now possible to give a definition of forward and inverse kinematics. The aim of the forward kinematics analysis is to determine the cumulative effect of the set of joint variables of the entire kinematic chain considered, that is, to determine the position and orientation of the end-effector given the values of the joints of the chain [7]. In contrast, the aim of the inverse kinematics analysis is to obtain the values of the variables associated with the various joints of the chain given the position and orientation of the end-effector

in question. In this context, we can define a serial kinematic chain with n joints as a vector of joint parameters with the following wording:

$$q = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}^T \tag{2.5}$$

In this expression q_i represents the rotation angle for a revolute joint or the translation if it is a prismatic joint. Each joint is connected to a link. Each link i is in turn attached to a local reference frame that is expressed as a function of the previous frame. Consequently, by expressing the transformation from a reference frame i to its next $(i+1)$, as the following homogeneous transformation matrix,

$$T_i^{(i+1)} = \begin{bmatrix} R_i^{(i+1)} & t_i^{(i+1)} \\ 0^T & 1 \end{bmatrix} \tag{2.6}$$

where $R_i^{(i+1)} \in \mathbb{R}^{3\times3}$ is the rotation matrix present between the reference frame i and $(i+1)$ and where $t_i^{(i+1)} \in \mathbb{R}^{3\times1}$ is the translation vector between the frame i and $(i+1)$, the unique solution for the forward kinematics is found, by iteratively applying all the transformations between the various reference frames of the kinematic chain that determines the position and the final orientation of the end-effector associated with it by solving the following equation.

$$T_0^n = T_{n-1}^n T_{n-2}^{n-1} \dots T_0^1 \tag{2.7}$$

The previous equation can be rewritten by explicitly specifying the joint variable values for each elementary transformation in this way:

$$T_0^n(q_1, q_2, \dots, q_n) = T_{n-1}^n(q_n) T_{n-2}^{n-1}(q_{n-1}) \dots T_0^1(q_0) \tag{2.8}$$

On the other hand, as already anticipated, inverse kinematics implements the reverse procedure. The general problem of inverse kinematics consists in finding the possible solutions of the equation

$$T_0^n(q_1, q_2, \dots, q_n) = H \tag{2.9}$$

In particular, in this equation H is known and represents the desired position and orientation i.e. this indicates where we want the end-effector of the kinematic chain to be while $q_1, q_2, \dots, q_n$ are unknown so our task is to find these values of the joint variables such that $T_0^n(q_1, q_2, \dots, q_n) = H$

However, solving the inverse kinematics problem is much more complex than direct kinematics since there is not always only one solution but there could be many more or even none because an end-effector of a kinematic chain might not be able to reach a specific target position with a specific orientation [8]. When inverse kinematics returns more than one solution, it is the inverse kinematics resolution algorithm that chooses the best solution based on how effective it is in reality and the computational cost that choosing that solution entails. In literature there are many methods that try to solve the inverse kinematics problem, the most common are distinguished in *analytical methods* in which the objective is to find exact solutions applicable only to not very complex robots and in *numerical methods* that exploit iterative algorithms

to find approximate solutions [9]. In the solution of an analytical approach, we are faced with a closed-form expression that expresses the inverse kinematics as a function of the pose of the end-effector of a kinematic chain, that is, an equation is defined for each joint parameter.

An analytical solution consists in inserting the known values of the end-effector pose and the link lengths into the equations, to obtain the values of the joint parameters that allow the manipulator to move in the required pose. An analytical approach is considered algebraic if it uses the equations derived from the equality between the known transformation matrix (which defines the target pose) and the forward kinematics matrix of the robot. It is instead considered geometric if the equations are derived from the application of trigonometric rules, based on the physical structure of the kinematic chain associated with the end-effector under consideration. An advantage of the analytical solution over the numerical one is that the calculation is much faster, since it does not require successive iterations to reach an optimal solution. However, a numerical method is much more efficient in calculating solutions for complex robots with more than 6 degrees of freedom, unlike an analytical method, for which the problem becomes extremely difficult. Consequently, in a numerical method, it is not sufficient to simply substitute values in a closed expression, but it is necessary to numerically calculate the joint angles based on an iterative optimization that leads the end-effector to progressively approach the target pose. In the literature, several approaches have been proposed to obtain approximate solutions. The most widespread numerical method to solve the inverse kinematics problem exploits the calculation of the Jacobian matrix and its inverse [10]. Given n joints, the values of the joint variables $q = (q_1, q_2, \ldots, q_n)^T$ and the related forward kinematic function $f(q) = T_0^n(q_1, q_2, \ldots, q_n)$, the Jacobian matrix describes a direct relationship between the joint velocities that we define as the column vector $\dot{q} = (\dot{q_1}, \dot{q_2}, \ldots, \dot{q_n})^T$ of size $(n \times 1)$ and the velocities (linear and angular) of the end-effector of a kinematic chain that we define as the column vector $\dot{x} = (\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma})^T$ of size (6×1) for a robot moving in a three-dimensional space. Specifically, this relationship specifies how variations in the velocity of the joint motion influence how fast the end-effector should move [11]. This concept is expressed by the relationship

$$\dot{x} = J(q)\dot{q} \tag{2.10}$$

where $J(q)$ is the $(6 \times n)$ Jacobian matrix obtained by computing the partial derivatives of each component of the forward kinematic function $f(q)$ with respect to each joint variable parameter of q. See the following equation.

$$J(q) = \frac{\partial f(q)}{\partial q} \tag{2.11}$$

Each column of the Jacobian matrix corresponds to a specific joint. In particular, the first column determines the impact of the speed of the first joint $\dot{q_1}$ on the speed of the end-effector $\dot{x}$, similarly the second column represents how much the speed of the second joint $\dot{q_2}$ influences that of the end-effector $\dot{x}$ as will do the n-th column. Consequently, since there is a direct relationship between the velocities of the terminal manipulator and the velocities of the joints that allows us to determine

the final pose of the end-effector in a given time, if we want to find the solution to the inverse kinematics problem it is sufficient to obtain the velocities of the end-effector and calculate the inverse Jacobian to obtain the values of the different velocities of the joints of the kinematic chain that is given by the equation

$$\dot{q} = J^{-1}(q)\dot{x} \tag{2.12}$$

Feed the joint velocity values to the joint actuators, after a certain time, the end-effector will reach the desired target position. However, calculating a solution for this equation is not always possible because the Jacobian matrix may not be invertible. To be invertible a matrix must be square. In this regard, a Jacobian matrix is square only in the case where we are describing robots with a number of joints exactly equal to 6, neither less nor greater than 6 which would produce non-square matrices. Even if initially invertible, a matrix may become non-invertible when the robot assumes some specific pose. This pose is called *singular configuration* and causes the Jacobian matrix to lose its rank, and its determinant to become zero so that the inverse no longer exists [11]. This event is visible, for example, in the case where the end-effector of a robot is at the limits of its workspace causing, due to the excessive extension, the loss of a degree of freedom.
To solve this problem, when the Jacobian is not square or is singular, the Moore-Penrose pseudo-inverse J^+ is computed instead of the simple inverse J^{-1}. Equation 2.12 then becomes

$$\dot{q} = J^{+}(q)\dot{x} \tag{2.13}$$

where $J^+ = (J^T J)^{-1} J^T$ and J^T is the transpose of the Jacobian matrix J [8].

2.4 The history and development of humanoid robots

Grasping is one of the most relevant and at the same time most complex functionalities for a humanoid robot. To understand the state-of-the-art in this field, it is essential to analyze the evolution of humanoid robots that have contributed to defining the current technological and methodological standards.

There is a long history of mechanical systems with human form that perform human-like movements.

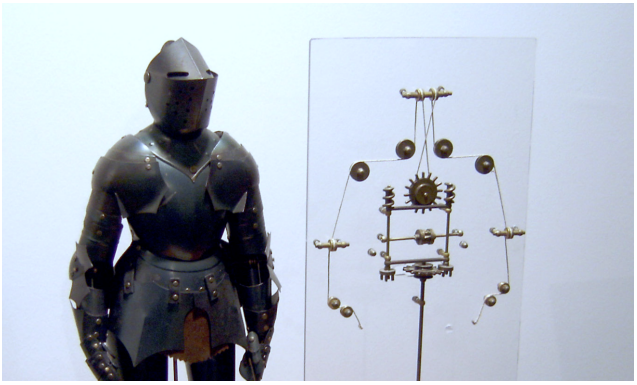


Figure 2.3 Model of Leonardo's robot with inner workings, on display in Berlin [12].

For example, Al-Jazari designed a humanoid automaton in the 13th century [13], Leonardo da Vinci designed a humanoid automaton (see Fig. 2.3) in the late 15th century [12], and in Japan there is a tradition of creating mechanical dolls called Karakuri ningyo that dates back to at least the 18th century [12]. However, it was only in the second half of the 20th century that advancements in digital computing allowed researchers to integrate substantial computational capabilities into their robots, enhancing sensing, intelligence, control, and actuation.

As shown in Fig. 2.4, we can distinguish two types of humanoid robots: *biped humanoids*, which have a full body with two arms and two legs, resembling a human in structure, and *upper-body mobile humanoids*, which are wheeled robots equipped with an upper body that includes a head, torso, and two arms [14].

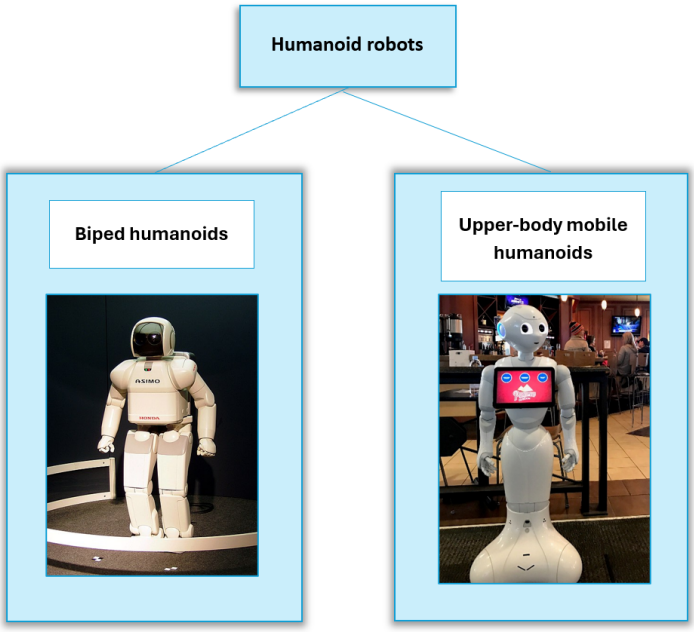
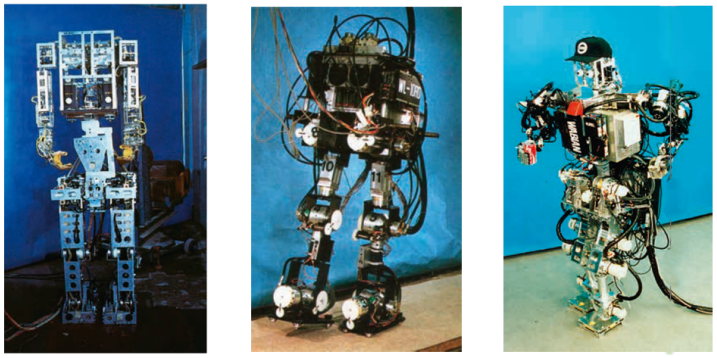


Figure 2.4 Humanoid robots taxonomy.

2.4.1 Biped humanoids

The first humanoid robot that we have traces of in the academic field is the WABOT-1 (Fig. 2.5) that was developed in 1973 by Kato and his team at Waseda University in Japan. This robot represents a milestone in the history of humanoid robotics since it was able to walk, recognize objects and manipulate them with its hands. Its movement was "quasi-static" and rather slow. Nevertheless, the WABOT-1 [15] was a proof that it was possible to realize a robot with a physical structure similar to the human one that paved the way for future developments in robotics. Although very rudimentary, its ability to manipulate objects laid the foundations for research on grasping. Professors Takanishi and Yamaguchi developed the WL (Waseda Leg) (Fig. 2.5) [16] and WABIAN (WAseda BIpedal humANoid) (Fig. 2.5) [17] series which improved dynamic walking thanks to the introduction of the concept of Zero Moment Point (ZMP). Zero Moment Point is a fundamental principle that ensures

the stability of the robot during the fastest movements. These robots had the ability to maintain dynamic balance, even when the center of mass moves outside the support polygon. This innovation represented a significant advancement for locomotion, but also for manipulation activities, since in this way the robot was more stable and could devote more resources to the meticulous control of the hands.



(a) (b) (c)

Figure 2.5 From left to right: **(a)** WABOT (WAseda roBOT); **(b)** Waseda Leg (WL-10RD), and **(c)** WABIAN (WAseda BIpedal HumANoid).

In 1996, P2 (Prototype 2) humanoid robot was unveiled [18]. This robot was the result of a secret work that lasted about 10 years and became famous for its abilities to walk autonomously, go up and down stairs, and even push a cart. It stood out from other humanoid robots for its predisposition to be used in real contexts. ASIMO (Advanced Step in Innovative MObility) [19] was its successor and brought further improvements, such as the ability to run, jump and pour liquids with a certain precision.

In Japan, Honda and other companies developed the Humanoid Research Project (HRP), a project that represented a step forward in the standardization of robotic platforms. In 2003, however, the HRP-2 [20] emerged (see Fig. 2.6), which became a point of reference for international research thanks to its ability to generate dynamic walking patterns through predictive control based on the linear inverted pendulum model [21]. This approach, introduced by Shuuji Kajita and his team, allowed for much more fluid and stable movements. These movements positively affected all those manipulation activities typical in dynamic environments. The HRP-2 was used by numerous research institutes, promoting the sharing of knowledge and the development of advanced grasping algorithms. Since the mid-2000s, several full-scale humanoid robots have emerged, including WABIAN-2 [22], TORO [23], Hubo [24], iCub [25] and Talos [26]. These robots have often been used in research and have helped standardize manipulation and grasping approaches and methodologies. For example, the iCub (see Fig. 2.6), designed by the Italian Institute of Technology, has been used to study learning and human-robot interaction, with a focus on object manipulation and tactile perception.

In parallel with full-scale robots, small-scale humanoid robots have also been developed, such as QRIO [27] and NAO [28] (see Fig. 2.6). These robots have also

been used for research on human-robot interaction and cognitive robotics, not to mention that these robots are used to study grasping in social contexts thanks to their ability for bodily expression and interaction based on sensors present all over the body.

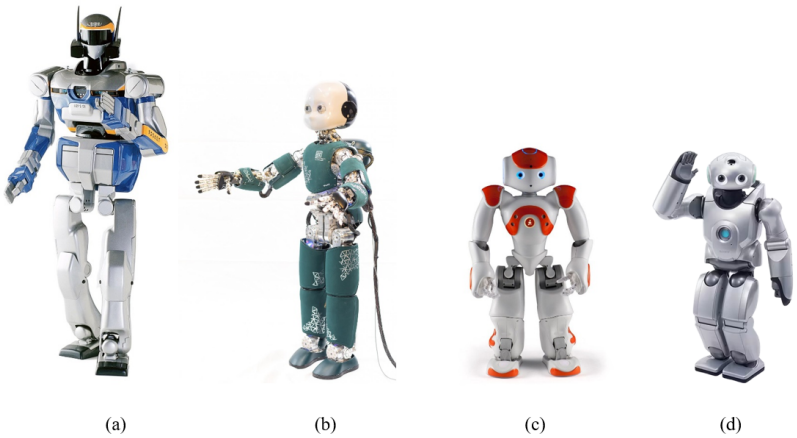


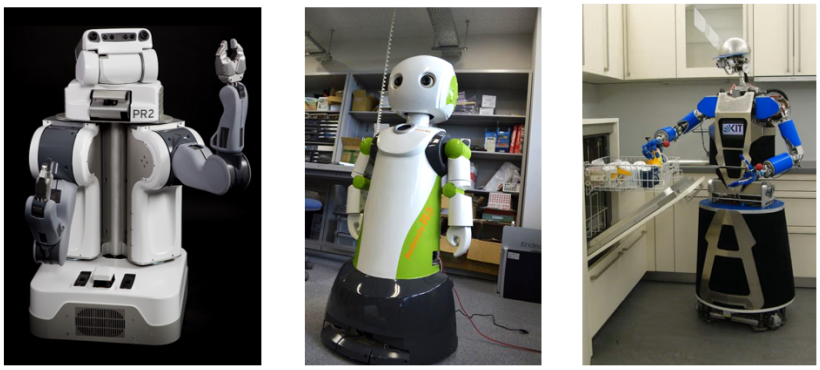
Figure 2.6 From left to right: **(a)** HRP-2, **(b)** iCub, **(c)** NAO and **(d)** QRIO.

2.4.2 Upper-body mobile humanoids

Upper-body mobile humanoids are useful for research on human-robot interaction because of their ability to adopt a wide range of behaviours combining trajectories of mobile base and upper-body motions. One early example of this type of robot was Robovie (Fig. 2.7) [29], which showcased the potential for proactive human interaction. It served as a language tutor for children [30], [31], a museum guide [32], and an emotional companion in shopping malls [33], [34], contributing to the advancement of humanoid robots capable of delivering various services even in crowded space. From 2001 to 2018, the German Research Foundation funded a project called ARMAR [35], aimed at developing humanoid robots capable of performing tasks in human-centered environments, learning from human observation, and interacting naturally with people. The ARMAR series yielded numerous notable robots, but the ARMAR-III [36] platform was the first to revolutionize object manipulation. ARMAR-III (Fig. 2.7) focused on human-robot interaction and independent object manipulation in everyday settings. It features a head, two versatile arms each with 7 degrees of freedom and hands with 8 degrees of freedom, connected to an omni-directional mobile base through a hip joint.

The dual-arm mobile platform PR-2 (Fig. 2.7), developed by Willow Garage, has also been widely used in robotics research, covering areas like advanced task planning, motion planning [37], and object grasping [38].

Pepper, the platform utilized in this work, is an upper-body mobile humanoid robot. In the following Sections, we will explore in detail how it has been employed in research, particularly for manipulation tasks.



(a) (b) (c)

Figure 2.7 From left to right: **(a)** PR-2, **(b)** Robovie, **(c)** ARMAR-III.

2.4.3 Aldebaran Robotics' Humanoid Robots

Returning to what was previously mentioned, Aldebaran robotics (now know as Softbank robotics) has developed several humanoid robots, designed with the intent of performing tasks of interaction with humans rather than strictly manipulation tasks: in addition to NAO and Pepper, Aldebaran has also produced another robot called Romeo.

NAO is a small programmable humanoid robot whose development began in 2004 with the aim of being used in research and education. In particular, it helps teachers in creating educational and fun lessons for children thanks to its interactive way of being, intuition and its friendly appearance. Going into detail in the literature, it is noted how NAO is used in more interaction-oriented jobs. In this regard, for example, in the paper [39] it is shown how NAO is a small-sized humanoid robot that fits well into the research area of human-robot interaction (HRI). This study focuses mainly on speech recognition and gesture recognition. The authors of the paper show that NAO is able to recognize and respond excellently to human language and gestures. This is possible thanks to the use of Generative Pretrained Transformer (GPT) language models as a service in the Internet of Things. In this way, the robot can create a human-sounding text and transform the response obtained into an audio format that is played on NAO using Text-to-Speech services that work on the Cloud. In the paper, they highlight how these results allow NAO to be placed or applied in countless social contexts, such as education, healthcare and entertainment. As evidence of its practicality, they show how NAO is also equipped with a remarkable pelvic kinematic system that allows it to rotate its body at non-trivial angles.

On the other hand, Romeo (see Fig. 2.8) is a humanoid robot whose project was launched in 2009 that, with a height of 1.47 meters, is much more massive than both NAO and Pepper and is designed, according to what was declared by Aldebaran Robotics, with the intent of being of assistance to the elderly and to those who are starting to lose their autonomy. Thanks to its height and 37 joints distributed throughout its body, this humanoid robot is able to walk, climb steps, grab objects and support people in various daily activities. It is also equipped with 4

RGB cameras, 2 of which are located in the eyes and can move separately while the other 2 are positioned at forehead height and are fixed so that they can, theoretically, be used as a stereo system to generate a depth map useful for understanding the distance from objects, people and obstacles nearby. Like Nao, Romeo is equipped with 2 legs.

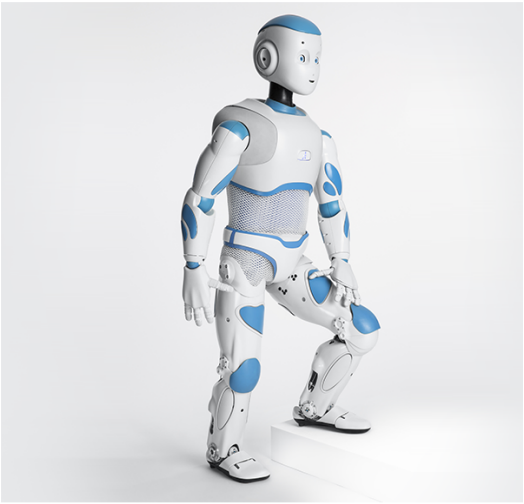


Figure 2.8 Romeo robot.

On the contrary, Pepper, introduced in 2014, is not equipped with legs but with a base with 3 omnidirectional wheels that make it, compared to the other two humanoids, less able to move and manipulate objects given that it also has a low number of degrees of freedom throughout its body and in particular in the arm where there are only 5 degrees of freedom. However, according to Aldebaran Robotics, Pepper is praised for its ability to read emotions, can interact with people through its touch screen, can help people locate exactly where an object or an individual is, can act as a receptionist, attract customers, have a conversation, provide product information in a store, manage routine and repetitive tasks and can learn people's preferences, habits and needs [40], [41], [42], [43], [44], [45].

In this Section, the main humanoid robots in the literature and their characteristics have been introduced and described. In the next Section, an in-depth study of the grasping problem is addressed, how this problem is approached in the literature.

2.5 Related work

In this Section, we will present how grasping systems were designed and implemented for some of the robots introduced in Section 2.4. Specifically, the robots under consideration are HRP-2, iCub, Romeo, PR-2, the ARMAR robots, NAO, and, of course, Pepper.

Despite the HRP-2 robot being equipped with a wide array of sensors, the paper [46] introduced additional sensor patches called HEX-o-SKIN, a self-organizing, multimodal cellular artificial skin. The goal was to develop a tactile-based method for

grasping large, unknown objects using either a single end-effector or both. Low-level controllers, assigned to each HEX-o-SKIN cell, utilize a self-explored sensory-motor map to convert tactile inputs into reactive arm movements, adapting grasping trajectories. A high-level state machine coordinates these controllers through various grasping phases, with desired contact points taught via forceless tactile stimulation, improving adaptability to object variations.

One of the most cited works on grasping for the iCub robot employs a servoing technique [47]. This method precisely determines the position and orientation of the robot's hand using a stereo-vision system. The system relies on a 3D model of the robot's arm and hand, incorporating details about its structure (kinematics) and appearance. It compares real images of the hand with synthetic images generated by the internal 3D model. To assess the alignment between real and synthetic images, the system uses an edge-based distance transform metric, which focuses on the hand's outlines. Accuracy is enhanced through a particle-based optimization method that continuously adjusts the robot's internal 3D model in real-time to better align synthetic images with real ones. This process corrects errors in the robot's kinematic calibration, ensuring the model accurately reflects the real-world position and movement of the hand.

A similar but simpler approach was developed for Romeo [48]. Here, the visual servoing technique enables both one- and two-handed manipulation tasks, such as grasping and solving a ball-in-maze game. It incorporates gaze control to keep the hand and object in view, employs a master/slave control for dual-arm tasks, and introduces methods to avoid joint limits. The 6D pose of the hand and object is estimated using visual markers placed on them.

A highly interesting study was proposed in 2010 with the PR-2 robot [49]. It integrates robust detection algorithms using vision and laser sensors to identify doors, handles and plugs. The system employs compliant control to overcome perception limitations during manipulation. A visual-differencing approach enables precise plug insertion without requiring high-precision calibration. Extensive experiments demonstrate reliable task execution without environmental modifications.

A key aspect of the ARMAR robots' evolution is their grasping and manipulation capabilities. The progression began with ARMAR-I, which had simple grippers, to ARMAR-II, which introduced 5-fingered anthropomorphic hands with fluid actuators—a design adopted in all subsequent models. These hands, combined with human-inspired arm designs, enable both single- and dual-handed grasping tasks in indoor environments like kitchens or industrial settings. As a result, this series of humanoid robots has been the subject of numerous studies. In the 2006 paper [50], Tamim Asfour et al. present a system for grasp planning and visual object localization using a humanoid robot with 5-fingered hands. Their work draws on experience with an ARMAR-II model. The goal is to enable the robot to grasp objects in a kitchen environment using a database of 3D object models. The system integrates an offline grasp analysis module, which simulates optimal grasp configurations using CAD models, and a real-time stereo vision module for object localization. Offline analysis results are stored in the database for grasp execution. The pipeline combines visual perception and planning for manipulation in realistic settings. Similarly, the paper [51] introduces a system for visually recognizing grasping gestures and mapping them onto humanoid robots. Validation tests for this method were conducted on

ARMAR-III. The approach, which operates in real-time without markers, includes human motion capture, hand pose estimation, and gesture mapping/execution on the robot. The aim is to allow a human user to intuitively teach the robot how to grasp objects. Real-time motion capture uses an RGB-D camera to track human hand movements without markers. The system extracts the hand's position and orientation, tracking 3D data to identify grasp configurations. Grasp recognition relies on analyzing 3D hand data from motion capture to classify grasp types, using an algorithm that compares detected poses with a predefined database of standard grasp configurations. Finally, gesture mapping/execution transfers the recognized grasp to the humanoid robot, converting the hand pose into commands for the robot's actuators.

NAO is an affordable robot, making it widely used in research and robotics competitions. Here, we explore three works developed on this platform. In the first [52], the NAO robot uses monocular stereo vision to investigate methods for positioning and grasping objects. The research focuses on how NAO can recognize and locate objects in a home environment to assist individuals with special needs, such as the elderly. The paper proposes a recognition algorithm based on quantitative statistical information to extract the area of interest from images and eliminate environmental interference to identify the target. The second work [53] enhances NAO's object localization and grasping capabilities using the YOLOv8 network for target detection, combined with monocular ranging for distance estimation. It addresses long-distance monocular vision errors by introducing a visual distance error compensation model and incorporates a grasping control strategy based on pose interpolation. The third work [54] develops a grasping system for NAO that employs models to identify known objects and integrates a path planner to generate feasible motion trajectories.

As noted earlier, Pepper is a robot widely used in social environments, but its hand and arm limitations make it less suited for manipulation tasks. Nevertheless, some studies address this. Let's explore them in this state-of-the-art review. In one work [55], Pepper is used for autonomous navigation and grasping. The study relies on a Python script that enables Pepper to explore a room after creating a map and grasp an object marked with Naomarks [56]. Naomarks are special visual markers designed for recognition by SoftBank Robotics' NAO and Pepper robots, similar to QR codes or Aruco markers but tailored to the NAOqi robotics ecosystem. The second work [57] uses a visual servoing approach to implement a reaching and grasping system with Pepper. It leverages the robot's monocular camera and a marker-less, model-based tracker to reach the target. Additionally, markers on the end manipulator improve hand position tracking in space.

Chapter 3

Platform

This Section describes the platform that has been selected and used to test the proposed approach. In Section 3.1 an overview of Pepper is examined, in Sections 3.2 and 3.3 the hardware and software features of Pepper are highlighted while in Section 3.4 the importance of ROS (Robot Operating System) within the work is underlined.

3.1 The humanoid robot Pepper

To design an object grasping system, it is essential to know every aspect of your robot in-depth, in this sense an examination of the features, problems and limitations associated with Pepper is carried out. Pepper is a humanoid robot (see Fig. 3.1) developed by SoftBank Robotics, formerly known as Aldebaran Robotics.

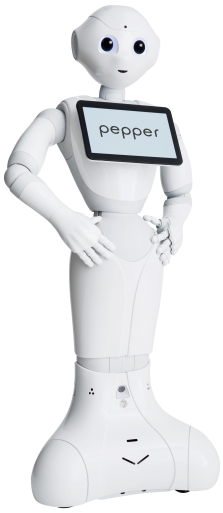


Figure 3.1 Pepper (a humanoid robot).

As anticipated, this robot is designed to assist people, to help social interaction, to be engaging in the activities it proposes and can be inserted in various contexts especially in indoor applications. Pepper is a programmable robot and allows software developers to design and create new applications. However, following the acquisition of Aldebaran Robotics by SoftBank Robotics, the SDKs used to develop applications for Pepper have been modified, leading to more dispersed and fragmented documentation for developers. Unfortunately, Pepper is no longer produced. In 2021, SoftBank Robotics announced that it had discontinued the production of Pepper robots. This uncertainty has made it increasingly difficult to perform careful and specific maintenance of the software, making updates and the production of new applications even rarer. Making the task even more complex is its hardware configuration that includes end-effectors with less than 6 degrees of freedom (DoF), which significantly limits its manipulation capacity compared to other industrial robots. But what is meant by degree of freedom? A crucial aspect that directly affects the ability of a robot to grasp objects is the number of degrees of freedom of the end-effectors, i.e. the end parts of the manipulators, such as hands or grippers. A degree of freedom in robotics refers to the independent movement capacity of a joint. For a robot, the number of DoFs determines the complexity and versatility of the possible movements. In these terms, having 6 DoFs is often considered the minimum necessary to replicate the versatility of the human hand. These 6 degrees of freedom are divided into: three DoFs for spatial positioning that allow the robot to move along the three dimensions of space x, y, z; and 3 DoFs that allow rotation around these 3 axes, providing control over the approach angle and orientation of the object to be grasped. In the case of Pepper, the arm chain, or the set of joints and actuators of the limb, used to perform the grasping of an object, has only 5 degrees of freedom which inevitably offers a limited ability to precisely adapt to the shape and orientation of the objects. Furthermore, each hand has 5 fingers that cannot be moved separately, its hands can either open or close completely, there are no half measures. The fingers are not rigid but tend to be flexible even under stress such as when the hand is closed, and this is why the force that can be impressed by the hands is very limited; even assuming that an object can be grasped and lifted, the weight that the hand joint can support does not exceed 200 grams. Consequently, this robot presents the developer of an object grasping system with countless challenges to address and solve before being able to obtain a reliable grasping system for a robot with the characteristics of Pepper.

3.2 Hardware

Pepper is a humanoid robot with a height of about 1.20 meters and a weight of 29.6 kg, equipped with an omnidirectional base consisting of three wheels that allow the robot to move in all possible directions and to rotate on itself 360 degrees with respect to the two-dimensional plane on which the floor lies. In this context, Pepper is defined as a holonomic robot, which means that the number of controllable degrees of freedom for locomotion corresponds to the number of available control variables. This feature allows Pepper to move in any direction on the horizontal plane without the need to first rotate, as instead happens for example in systems

such as cars in which the wheels are oriented in a specific direction and several maneuvers are necessary to make certain movements. Pepper is equipped with 17 joints distributed throughout the body and has an autonomy of 12 hours. To perform its tasks, Pepper must operate in a temperature range of 5 to 35 degrees Celsius with a humidity of 20% to 80%. The system has a Quad Core CPU module with an Intel ATOM® E3845 Formerly Bay Trail processor with a clock speed of 1.91 GHz and a RAM of 4 GB. For upper body movement, Pepper is provided with 2 degrees of freedom for the head (HeadYaw, HeadPitch) and 5 degrees of freedom for each arm (ShoulderPitch, ShoulderRoll, ElbowYaw, ElbowRoll, WristYaw). On the lower part, there are 2 degrees of freedom for a single hip (HipRoll, HipPitch), 1 degree of freedom (KneePitch) for the single knee and 3 omnidirectional wheels (WheelFL, WheelFR, WheelB) positioned at 120 degrees from each other that can move at a maximum speed of 2 km/h. Each degree of freedom is shown in Fig. 3.2. Pepper

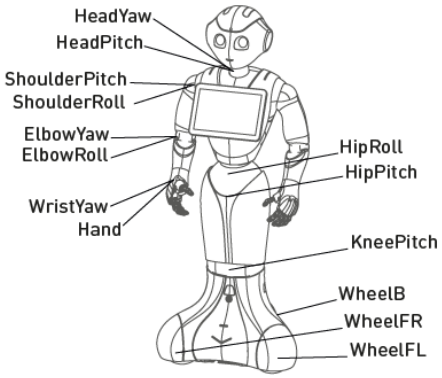


Figure 3.2 Degrees of freedom for the Pepper robot.

is equipped with various tools to interact with humans; it has two loudspeakers positioned in the ears and four microphones on the head (see Fig. 3.3)

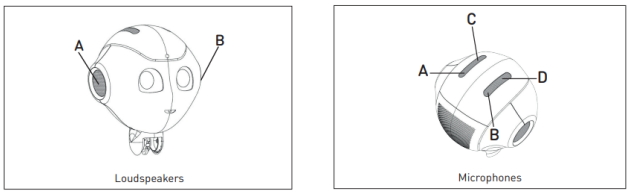


Figure 3.3 Two loudspeakers and four microphones for the Pepper robot.

On the chest there is a tablet with a Multi-Touch LCD screen measuring ($246 \times 175 \times 14.5$) mm (with a resolution of (1280×800) pixels equipped with a quad-core ARM Cortex-A7 processor with a clock speed of 1.3 GHz and a SDRAM of 1 GB. At the top of the screen there is a 2 megapixel front camera. The tablet's operating system is Android, allowing programming to create applications, web pages, images and videos that can be displayed on the screen to allow the user to interact with the robot directly by touching the display. Generally, Pepper is equipped with 2 HD

cameras with a native resolution of (640 × 480) positioned in the mouth and on the forehead, with a resolution of 5 megapixels as shown in figures 3.4 and 3.5. There

Figure 3.4 HFOV.

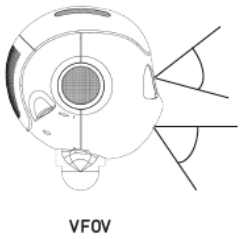


Figure 3.5 VFOV.

are 3 versions of the robot: Pepper 1.8, Pepper 1.8a and Pepper 1.6, and each version may have differences in terms of sensors possessed. To distinguish the models, you can check the number of holes behind Pepper's head: in version 1.8 there are 4 holes while in versions 1.8a and 1.6 only 2. Versions 1.8 and 1.8a have 3 buttons behind the right side of the tablet's display, unlike version 1.6 where there are only 2. The first two versions also have the camera visible on the top edge of the display, while the 1.6 does not. Optionally, depending on the model, Pepper can be equipped with a stereo camera or a 3-D depth sensor, positioned behind the eyes (see Fig. 3.6); the stereo camera is equipped with a pair of 2D cameras that return an image of (2560 × 720) size at 15 fps. The 3-D depth sensor is not present in all versions, having been removed in the most recent one, so the 2D cameras represent the best possible solution as far as visual sensors are concerned; in any case, the 3-D depth sensor was not very efficient. In fact, in the paper [58], the authors highlight how, in version 1.8a of the robot, the ASUS Xtion 3D depth sensor, responsible for depth perception, does not work properly. The point clouds it provides are noisy, distorted and full of artifacts. The scientific community believes that this is due to the lenses that the robot wears right in front of the sensor, but this has not been confirmed or denied by the manufacturer. To improve interaction with humans, Pepper has

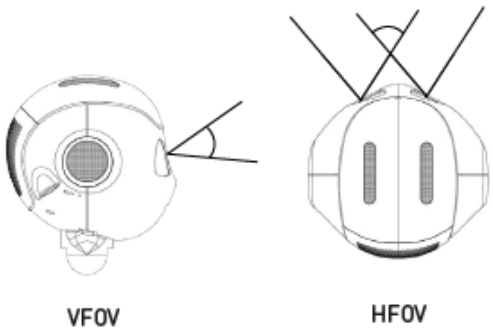


Figure 3.6 The ASUS Xtion 3D depth sensor and the stereo camera for Pepper robot.

touch-sensitive areas as shown in the Fig. 3.7: three on top of its head and two on the back of each hand. It also has buttons that perform specific functions: a button on its chest, located under the tablet, which, if pressed for a long time, turns

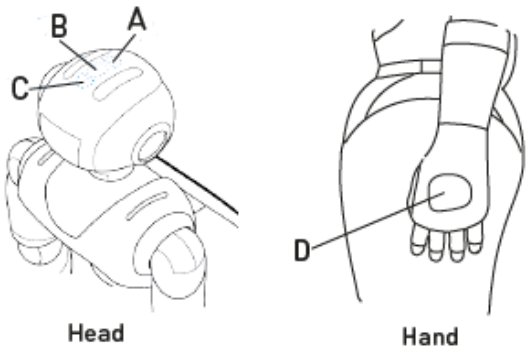


Figure 3.7 The Pepper's touch-sensitive areas: three on top of its head and two on the back of each hand.

the robot on/off; if pressed and released immediately it returns a voice output with Pepper's IP address; if pressed twice, it enters/exits autonomous life mode; a stop button, located behind the neck, which acts as an immediate safety measure. When pressed, this button stops all active operations of the robot, turning off the motors and stopping any movement. Finally, the wheel bumpers, located two in front and one behind the base, are contact sensors that detect when Pepper collides with an object or a person. In order to interact with the environment, Pepper has several environmental sensors: there is a laser positioned on each side of the base and three on the front of the base (see Fig. 3.8 and Fig. 3.9). On each side of the base there

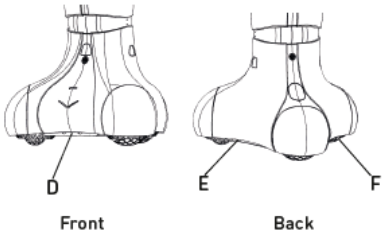


Figure 3.8 A laser positioned on each side of the Pepper's base.

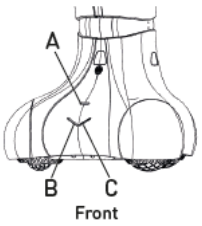


Figure 3.9 Three lasers on the front of the Pepper's base.

are two infrared sensors that detect obstacles at a maximum distance of 50 cm at a height of 27 cm from the ground (Fig. 3.10). Two sonars are instead positioned one in front and one behind the base (see Fig. 3.11). They have a detection range between 0 and 3 meters: objects closer than 30 cm are detected as if they were at 30 cm. Each sonar has an effective cone of 60 degrees. Pepper also has three sets of LEDs that the robot uses to transmit non-verbal information, as feedback to help the user understand the robot's status: there are LEDs positioned in the eye area, in correspondence with the speakers in the ears and on the shoulders. The robot has two gyroscopes located on the torso and at the base of the robot.

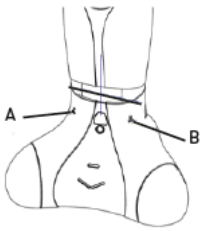


Figure 3.10 Two infrared sensors on each side of the Pepper's base.

Figure 3.11 Two sonars positioned one in front and one behind the Pepper's base.

Finally, it is possible to connect to the robot via Wi-Fi, Ethernet or Bluetooth, which has a connection independent of the tablet.

3.3 Software

Pepper is able to support two types of operating systems: NAOqi 2.5 and NAOqi 2.9. The choice of operating system does not impact the battery life of the robot. The choice should be made based on the different needs and taking into account the differences between the 2 operating systems. NAOqi 2.5 is based on Linux and allows development via Python or C++ SDKs. In particular, NAOqi 2.5 offers a development environment with more than 1000 available APIs, which allow low-level programming and direct access to the raw data of the robot's sensors and actuators. On the other hand, NAOqi 2.9 is based on an Android ecosystem that uses Java and Kotlin as the main programming languages within the QiSDK library. It provides a much more stable and secure development environment with only 20 different APIs that allow high-level development. For each SDK, Aldebaran provides detailed but very scattered documentation: Pepper SDK documentation for the Python SDK [59], C++ SDK documentation for the C++ SDK [60] and QiSDK documentation for the Android QiSDK library [61]. The respective documentation shows the steps for installing the SDKs, for connecting to the robot and for using the APIs. In particular, the Python SDK includes public APIs with modules that can be divided into groups: NAOqi's Motion, Audio or Vision. Typically, a module is a class within a library. These are divided into local modules, when two or more modules launched within the same process can share variables and call each other's methods without serialization or networking, or remote modules, when two or more modules are executed on different processes or on different robots that communicate using the network. More precisely, NAOqi running on the robot acts as a broker. A broker when it starts, it loads a configuration file called autoload.ini, which specifies which libraries it should import. Each library includes one or more modules that leverage the broker to promote their specific methods. A broker is therefore an object that has the purpose of finding modules and methods that can also be called outside the process itself. In this context, an ALProxy is instantiated, which is an object that behaves like the module it represents. For example, if an ALProxy is created for the ALMotion module, an object containing all the ALMotion methods is obtained. So, when

writing the code, the first thing to do is to import ALProxy, then a specific ALProxy must be created for each necessary module, and finally the methods associated with each module must be invoked according to the developer's needs. NAOqi 2.5 is compatible with Choregraphe [62], a multi-platform desktop application that allows you to create animations, behaviors, dialogues and trajectories and then test them on a simulated robot or directly on a real robot. Thanks to this platform it is also possible to monitor, control the robot and create complex behaviors such as dancing, interacting with people without having to write any line of code but simply by combining blocks that already implement the logic of specific behaviors and actions by default. However, it is possible to integrate python scripts into Choregraphe to give programmers an additional possibility of customizing their applications. Choregraphe and NAOqi 2.5 interact very easily, so much so that every possible API of NAOqi 2.5 is accessible through Choregraphe. One aspect to keep in mind is that the behaviors and actions created with this platform are slower at runtime than if they were written directly in Python or C++. Differently, as already anticipated, the Android Studio plugin for NAOqi 2.9 offers a series of graphical tools and a Java library, known as QiSDK. This allows you to make the robot move, talk and interact with people in a simple way through an intuitive API, directly from your Android activity. It also includes a Layout Editor to work directly on the tablet display. The APIs offered by QiSDK cover different areas: activation or deactivation of behaviors that Pepper performs autonomously such as blinking or trying to constantly follow the gaze of the person in front of it; creation of chatbots through the QiChat script language and management of activities such as listening or speaking; management of movements of the robot's limbs and head or creation of predefined trajectories; APIs that allow the robot to perceive and approach a specific human to start an interaction with him. Finally, it is possible to upgrade the operating system from NAOqi 2.5 to NAOqi 2.9. However, it is not possible to downgrade to earlier versions than the one currently installed on the robot.

3.4 ROS (Robot Operating System)

In this thesis, an important role is played by ROS (Robot Operating System). ROS is a free and open source software that defines the components, interfaces, and tools to build advanced robots. ROS helps developers quickly create these components and easily connect them using tools called Topics and Messages. These messages can be stored in log files or ROS bags, simplifying testing, training, and quality assurance. These messages can be sent using various visualization and teleoperation tools. With ROS, you can work with simulated robots instead of physical ones, a practice known as "digital twin," making testing and training more efficient and accessible. ROS supports a wide range of hardware interfaces for robotic components such as cameras, lidars, and motor controllers. Its modular structure allows developers to build robots without being tied to a single vendor or incurring licensing costs. ROS is not only a software, but also a community of highly skilled experts who apply advanced robotics in various fields, sharing their innovations and state-of-the-art algorithms for perception, planning, and localization. Opting for ROS means tapping into a growing ecosystem of ROS-based services that include logging and diagnostics, data

collection for deep learning, testing, quality assurance, etc. Among other things, the ROS community offers a ROS Interface node that allows you to create a bridge between ROS and a NAOqi system running on Desktop. This ROS interface is a program that runs both as a ROS node and as a NAOqi module. It listens for messages from NAOqi and shares the translated ROS messages. There are several tools that allow visualization and management of the robot that use the URDF and meshes provided by Aldebaran to render the data. ROS is compatible with almost any component that has a software interface. However, when it comes to Pepper, only NAOqi 2.5 is directly compatible with ROS.

Chapter 4

Proposed approach

In this chapter, the proposed methodology to solve the grasping problem is described, in particular the following modules are illustrated in detail: in Section 4.1 the constraints of the platform used are presented, in Section 4.2 the architecture of the proposed approach is described with particular attention to the 5 modules that compose it, while in Section 4.3 it is explained why a workaround was necessary to connect to the robot. Section 4.4 presents the NAOqi APIs used. In Sections 4.5 and 4.6 the object detection and depth estimation modules are explored in depth. The two Subsections 4.5.1 and 4.6.1 are associated with them, respectively, which illustrate the architectures of YOLO v10 and Depth Anything V2. Going further, in Sections 4.7 and 4.8 the modules of coordinate mapping and path planning are addressed and analyzed respectively. Section 4.9 is dedicated to the inverse kinematics module. In addition, the related Subsection describing the inverse kinematic Cyclic Coordinate Descent (CCD) approach is attached

4.1 Platform constraints

First of all, before developing and designing an object grasping system for a humanoid robot like Pepper, you need to establish whether the goal is to try to grasp known objects of which you know the three-dimensional structure, material or other specific characteristics of the object that could be useful during the task such as CAD models of the object or you are trying to develop an approach that does not presuppose any a prior knowledge. It is also essential to understand whether you want to use approaches that make use of markers that facilitate and help the grasp or if you want to apply auxiliary sensors to the robot. In addition to all this, you also need to discover and take into account the advantages and disadvantages of the platform you are using to validate and test the proposed approach. In particular, you need to understand what the constraints and limitations may be that could then arise when implementing the system and affect the final product. At the same time, even version 1.8 of Pepper, present in the VisionLab, has constraints and limitations at both hardware and software level that must be taken into account. In particular, one of the most relevant aspects concerns Pepper's inability to perform force-closure single handed grasps because it does not have fingers that cannot be moved individually but rather hands that can be in a state of totally open or totally closed, there is no

middle ground. Furthermore, as already introduced in advance, Pepper is able to lift a maximum weight of 200 grams which limits him in being able to grasp small and light objects. In addition to this limitation in the hands, Pepper has arms with only 5 degrees of freedom which do not allow him to bring the end-effector of this specific kinematic chain into many different positions and orientations. Another negative aspect of Pepper's hands, if used for grasping objects, is that it is only equipped with capacitive sensors that make it able to detect especially the touches of people who interact with it by touching the upper part of the hands and not with advanced tactile sensors on the inside of the hands that would allow him to have sensitive information during the execution of the grasping task. Even from the point of view of vision, this robot has some important limitations. Specifically, Pepper has 2 inefficient 2D cameras (a top camera that points forward and a bottom camera that points downwards) that operate with a frame rate of 15 frames per second (fps) at 10 Hz. Furthermore, these two cameras cannot be used simultaneously. It is necessary to switch from one camera to the other, activating one and turning the other off, depending on the need.
In the version of Pepper present in the VisionLab another quite relevant problem has emerged. In particular, the version at our disposal is equipped with NAOqi 2.9 which exploits the Android ecosystem, but according to what was described by the manufacturer [63], [64], the Python and C++ APIs present in NAOqi 2.5 were more suitable for advanced and low-level robotics applications since they allow direct control of the joints and various sensors including cameras. Furthermore, as highlighted by Aldebaran himself [63], it is not even possible to downgrade from NAOqi 2.9 to NAOqi 2.5, which would allow access again to the specific APIs for controlling the robot's sensors and actuators. Consequently, using the robot to make Pepper perform the grasping task, with the current version, would have led to further and excessive difficulty during the system implementation phase. To try to solve the problem and be able to use the old APIs, a workaround described in Section 4.3 was used.

4.2 System architecture

After illustrating the different constraints of the platform, the architecture structure is presented in the diagram shown in Fig. 4.1.

The system has been divided into 5 modules that are introduced:

- **Object detection:** identifies the target object within the images captured by the robot cameras. In particular, the localization occurs by extracting the bounding box of the object that defines its position within the image. The extraction of the bounding box is essential to subsequently determine the pose of the object.

- **Depth estimation:** given as input an image of size $(h \times w \times 3)$, captured with a robot camera, estimates the depth of each pixel of the image from the camera producing a depth map of size $(h \times w)$. In particular, in this system, the depth estimation algorithm is based on a monocular method rather than a multi-view based one. This choice will be clarified later in Section 4.6. The

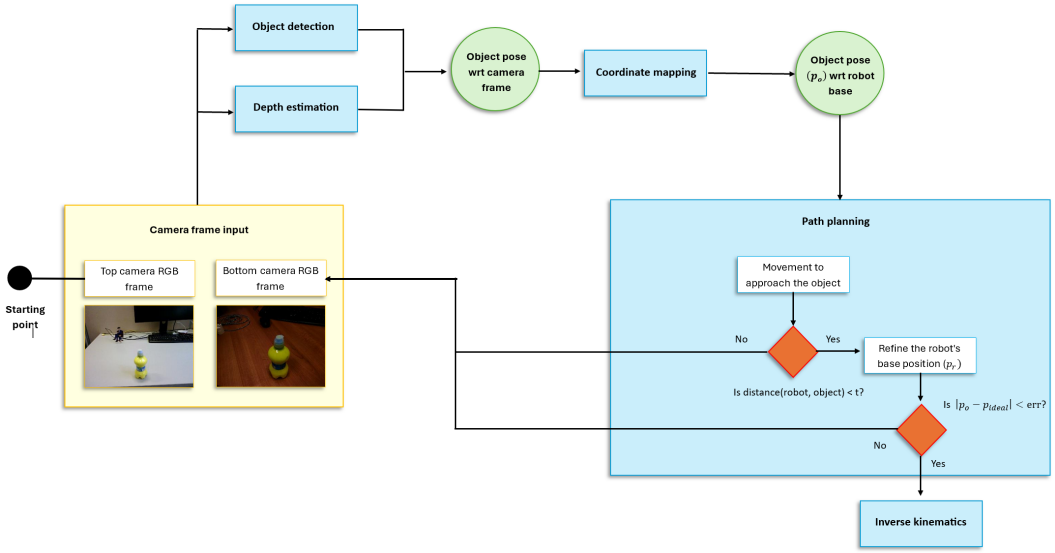


Figure 4.1 System architecture.

bounding box generated by the previous module combined with the depth map, allow to establish the pose of the target object with respect to the camera frame.

- **Coordinate Mapping:** maps the coordinates of the detected planar image in the coordinate system with respect to the robot base.
- **Path planning:** generates a feasible path from the manipulator to the object to be grasped. It performs a set of movements to try to optimize the distance, position and orientation with respect to the desired target to produce the best possible grasping pose.
- **Inverse kinematic:** calculates the angles or configurations needed for each joint of the arm starting from the detected position and orientation of the target object

Before continuing, it is important to define the position and orientation of the reference frames attached to each link of the Pepper body. Fig. 4.2 shows the arrangement of these reference frames according to the conventions implemented by the manufacturer [65].

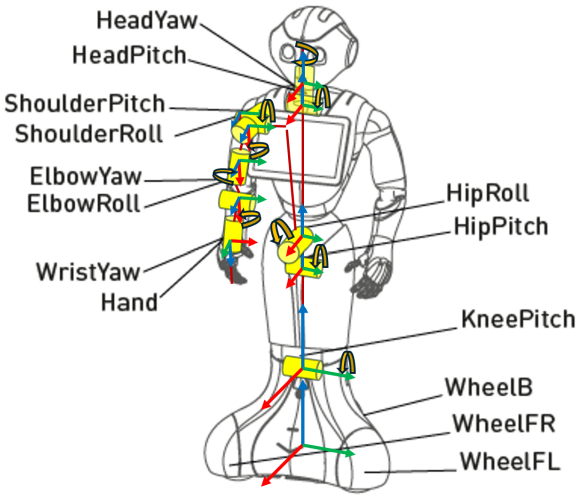


Figure 4.2 Pepper body refrenece frames.

In particular:

- The red axis corresponds to the **x**-axis
- The green axis corresponds to the **y**-axis
- The blue axis corresponds to the **z**-axis

That said, let's now see how the modules that make up the system architecture are connected to each other. The system starts with the capture of frames from the 2D Top camera placed on the forehead in real time. Each frame captured at a frame rate of 15 fps represents an RGB image with a resolution of (320×240) pixels that is fed to the 2 modules of object detection and depth estimation. The object detection module identifies and locates the object to be grasped, in particular it extracts the bounding box associated with it. In this context, a bottle was chosen as the target object, being an object that typically might need to be retrieved and grasped to satisfy some human request and it was chosen to be small in size so that it could be consistently grasped by Pepper's hand, improving stability during the grasp and to respect the constraints and limitations of the robot. At the same time, the depth estimation module estimates the depth map of the entire image. Thanks to a combination of the outputs of the two modules, the pose (position and orientation) of the target object with respect to the camera frame is obtained. The pose is given as input to the coordinate mapping module which transforms the coordinates of the detected planar image into the coordinate system with respect to the robot base. Once the transformation has been performed, the pose p_o of the object with respect to the reference frame of the robot base is forwarded to the path planning module. In this way, the robot roughly approaches the proximity of the target object based

on the obtained coordinates. At this point, if the robot is at a distance from the target lower than a certain proximity threshold, it means that it is positioned at such a proximity that theoretically, considering only the direction along the x-axis coming out of the front part of the pepper base, it would allow it to grasp the object. In this case, before activating the inverse kinematics module, it is necessary to perform an iterative refinement of the robot's base pose p_r until the detected pose p_o of the object (which this time is obtained from the frames of the Bottom camera to have a better view of the target) does not coincide with its ideal pose p_{ideal} with a maximum deviation equal to err that would allow the best grip. When the error is close to zero, the path planning ends and the inverse kinematics module is activated, which allows to adjust the movement of the arm and hand towards the object and then grab it. If, instead, in the path planning module, the distance between the robot and the object was still too large to assume a good grip, the complete process of frame capture, object detection, Depth estimation, coordinate mapping and path planning is repeated this time using the bottom camera since the view with the top camera would no longer be optimal.

These 5 modules are written in Python and run on a laptop with an Intel Core i9 processor and an Nvidia RTX 4060 GPU. The entire system communicates through three main components (see Fig. 4.3) that interact with each other by sending and receiving data:

- **NAOqi APIs**
- A first **Flask Server**
- A second **Flask Server**

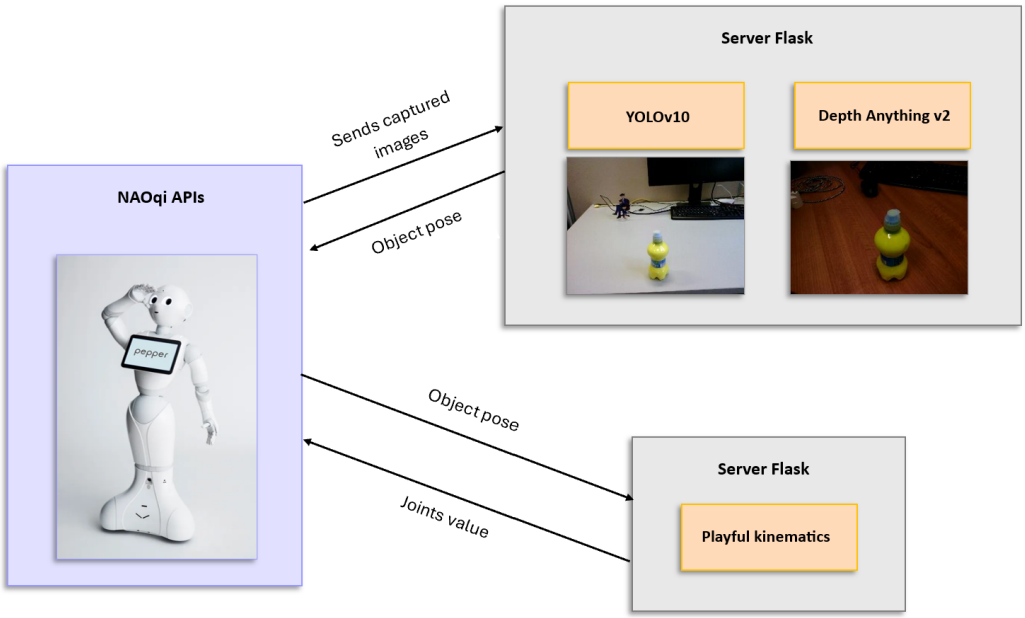


Figure 4.3 System architecture.

The script that invokes the NAOqi APIs is launched on Windows, in particular they are used to capture the frames from the camera and, as a Client, sends them via an HTTP POST request to a first Flask Server. The server, also running locally on Windows, processes the images by applying the object detection and depth estimation modules and generates a response that returns the object's pose to the Client. After obtaining the response, the client invokes the coordinate mapping and path planning modules with which the robot's position is optimized with respect to the target bottle. At this point the Client communicates with a second Server, installed on Ubuntu 18.04.6 LTS, running via Windows Subsystem for Linux (WSL) that offers you a Linux environment running directly on Windows, avoiding having to install a virtual machine or a dual boot. From the python script that interacts with Pepper, another HTTP POST request is sent that triggers the Inverse Kinematic module of Playful (see Section 4.9) that, given the target pose of the object, calculates the inverse kinematics, in particular it returns the values of the joints of the kinematic chain of the arm in the response. Finally, using the NAOqi API, the values of the joints obtained are used to move the actuators that lead the robot's hand to the desired position.

4.3 Communication with robot

As anticipated in 4.1, this Section describes the workaround that allows you to access the old NAOqi 2.5 APIs while having a Pepper version with NAOqi 2.9. Specifically, we know that Pepper and the computer communicate via a TCP/IP connection. Both the robot and the computer need to be connected to the same network so the robot will be able to provide an IP address that grants the ability to control it remotely. On Pepper, external TCP servers expect SSL/TLS connections so that communications are secure. This means that the server socket expects a TLS handshake request from the client. So the URL scheme of the server address typically used becomes tcps instead of tcp (similar to https vs http). In this way, it is possible to use the old APIs associated with NAOqi 2.5 even if the robot is running NAOqi 2.9. By using an API gateway, you can leverage and expose all the services running on the robot and the tablet, in particular it is possible to access the services of the old APIs provided by the Python SDK, even if they are no longer available directly in QiSDK.
But before showing how to exploit this API gateway, you can see how, if the operating system running on Pepper is NAOqi 2.5, connecting to it with the Python SDK is very simple.

```
from naoqi import ALProxy
tts = ALProxy("ALMotion", "<IP of your robot>", 9503)
```

Figure 4.4 Access to the ALMotion service.

For example, if you want to use the ALMotion service, as illustrated in Fig. 4.4, you just need to replace "<IP of your robot>" with the actual IP address of your

robot to point ALProxy to the correct location on the network. Port number 9559 is the default for NAOqi services, so it is usually used unless you have configured the robot to use a different port. If you do not know the IP address of Pepper, you can simply press the chest button of the robot. On the other hand, to access the gateway, you need to authenticate. Connecting to a real robot with QiSDK requires authentication via the robot's password. To authenticate via the old NAOqi 2.5 APIs, it is essential to add to your python script an Authenticator class very similar to the one shown in Fig. 4.5 every time you connect to a real robot that allows you to configure each new session in the correct way.

```
class Authenticator:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def initialAuthData(self):
        return {'user': self.username, 'token': self.password}

class AuthenticatorFactory:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def newAuthenticator(self):
        return Authenticator(self.username, self.password)

session = qi.Session()
logins = ("nao", "<robot password>")
factory = AuthenticatorFactory(*logins)
session.setClientAuthenticatorFactory(factory)
session.connect("tcps://<IP of your robot>:9503")
```

Figure 4.5 Authenticator class.

Going into detail, the `Authenticator class` is responsible for storing and returning authentication data. Its `initialAuthData(self)` method returns a dictionary containing the authentication data with the keys `"user"` and `"token"`. The `AuthenticatorFactory` class is responsible for creating instances of the `Authenticator class`. Its `newAuthenticator(self)` method creates and returns a new Authenticator object using the stored username and password.
Running the line `factory = AuthenticatorFactory(*logins)` creates an instance of `AuthenticatorFactory` using the provided username and password. The *∗logins* syntax unpacks the tuple into two separate arguments (username and password). Finally, the session is configured to use the `AuthenticatorFactory` for authentication. This means that every time authentication is required, the session will request a new Authenticator from the factory.

4.4 NAOqi APIs used

To support the development of the proposed approach on Pepper, as has already been anticipated several times, in particular in Section 4.2, it is necessary to work with the NAOqi API. First of all in this case, you need to register to a service. An instance of a module provided by the NAOqi Service Manager is created that must be associated with a session every time a connection is opened with the robot so as to allow communication. In the developed system, four services were basically used:

- **session_service:** allows the session to be initialized without which no other service can be requested.
- **motion_service:** allows the movement of the robot and is requested with the call to the `session.service("ALMotion")` method associated with the session object.
- **posture_service:** sets the posture of the robot in predefined poses and is requested with the call to the `session.service("ALRobotPosture")` method associated with the session object.
- **video_service:** requires access to Pepper cameras and is requested by calling the `session.service("ALVideoDevice")` method associated with the session object.

Table 4.1 summarizes the main methods implemented in the system with their explanations.

NAOqi API	Description
`qi.Session()`	Creates a session to connect to the Pepper robot.
`session.connect(robot_url)`	Connects the session to the specified robot via URL.
`session.service("ALMotion")`	Retrieves the `ALMotion` service to control robot movements.
`session.service("ALRobotPosture")`	Retrieves the `ALRobotPosture` service to manage robot postures.
`session.service("ALVideoDevice")`	Retrieves the `ALVideoDevice` service to access camera data.
`motion_service.wakeUp()`	Wakes the robot from rest mode.
`motion_service.rest()`	Puts the robot into rest mode.
`motion_service.setAngles()`	Sets the angles of specific robot joints.
`motion_service.getAngles()`	Gets the current angles of specific robot joints.
`motion_service.moveTo()`	Moves the robot to a specific position (x, y, theta).
`motion_service.angleInterpolationWithSpeed()`	Performs joint angle interpolation at a specific speed.
`motion_service.getPosition()`	Retrieves the position of a frame relative to another frame.
`motion_service.setMotionConfig()`	Configures robot motion parameters (e.g., max speed).
`posture_service.goToPosture()`	Moves the robot to a predefined posture (e.g., "Stand").
`video_service.subscribeCamera()`	Subscribes a client to receive images from a specific camera.
`video_service.getImageRemote()`	Retrieves an image from the subscribed camera.
`video_service.unsubscribe()`	Unsubscribes a client from the camera.

Table 4.1 NAOqi APIs used in the Python file.

In particular, among the methods offered by NAOqi, we find the `moveTo` method that allows the robot to move in the surrounding environment by passing as parameters the movement along the X axis (meters that the robot travels forward or backward), the movement along the Y axis (meters that the robot travels moving laterally from left to right and vice versa) and the rotation around the Z axis (expressed in radians) that allows it to rotate on itself.

Another fundamental method is the `setAngles` method offered by the motion service, which allows you to control the angles of the joints or chains of joints. The method takes as input the name of the joints to move, the values of the desired angles (in radians) and a fraction of the maximum speed to use. Similarly, the `angleInterpolationWithSpeed` method is adopted if you need to make more fluid movements by simultaneously considering the movement of multiple joints. This method receives as input the same parameters as the previous method, but is different in that it generates a movement of the joints that gradually passes from the current pose to the target one.

Associated with the posture service, there is the `goToPosture` method that allows you to specify which default pose the robot must assume simply by passing as an argument the name of the pose ("Stand", "StandZero", "Crouch") and the speed with which it must reach it.

These APIs also allow you to manage the cameras thanks to the video service. In particular, the `subscribeCamera` method was used to subscribe to a client that allows you to receive a video stream from a specific camera. Based on the camera you want to subscribe to, this method receives as a parameter the index 0 if you want to access the upper camera, 1 for the lower one that points towards the floor, and 3 for the stereo cameras placed behind the eyes. The image resolution, the color space, and the number of frames per second are passed as additional parameters.

Another method of the motion service widely used within the system is certainly the `getPosition` method that returns the 6D position (position and orientation) of an item with respect to a particular frame given the name of the item (which could be any joint, chain, or sensor of Pepper) and the index of a task frame (`FRAME_TORSO = 0`, `FRAME_WORLD = 1`, `FRAME_ROBOT = 2`) that represents the reference frame with respect to which we want to calculate the position of an item.

These are the working methods that have been used within the work. Therefore, unfortunately, there are several APIs that cannot be reached through the aforementioned gateway, including the important one that implements the `setPosition` method of the motion service. This method would allow you to move an end-effector to the given position and orientation with a simple command. This is not possible, therefore it was necessary to implement a separate module that implements inverse kinematics (see Section 4.9).

Finally, it is important to take into account that in the version of Pepper available in VisionLab, the `setExternalCollisionProtectionEnabled` method of the motion service is not reachable, and consequently, it is impossible to disable the protections for external collisions. This causes strong limitations when you want to bring the base of the robot very close to obstacles placed on the floor, such as the legs of a table, as these protections cause the movement to block beyond a certain proximity.

4.5 Object detection module

In this Section, the object detection module is described in detail. But let's start from the beginning. A critical aspect of the implementation of the proposed approach concerns the management of Pepper's operating mode. During development, it was noticed that by disabling the robot's Autonomous Life (the default mode that allows it to interact autonomously with the environment) Pepper becomes less responsive to external stimuli but more controllable through programmatic commands. Although this choice reduces the integrated safety features, it facilitates precise control of movements, an essential requirement for a grasping system. In fact, by leaving it in this mode, Pepper tends to move by rotating on itself and to orient its head in various directions to study the environment or to follow the faces of people who are looking at it to create greater empathy with its audience. Therefore, Autonomous Life and manual control are not fully compatible: to move the robot, you have to

give up its interactive capabilities and vice versa. To overcome this limitation, a hybrid solution has been designed that emulates some features of Autonomous Life (such as the ability to "look around") while this mode is deactivated. In practice, the system includes a search routine that allows Pepper to incrementally scan the environment by rotating on itself by an angle of $\pi/3.0$, to maximize the probability of detecting the target object.

After clarifying the problem of the Autonomous Life mode, it is necessary to describe in detail the object detection module. This module, as previously introduced, runs on a Flask server. A script in Python 2.7 communicates with the robot to give it the commands to execute. For the script that communicates directly with Pepper, a version of Python so old is used because the Python SDK necessarily requires that version to be installed as described in the documentation [66]. On the other hand, having no constraints, the Flask server present here is written in the Python version `3.11.10.` which is much more recent and compatible with many more frameworks. The use of a server was necessary to allow powerful Computer Vision models to be run efficiently remotely by sending only the output values to the robot, as Pepper is equipped with a processor that is not particularly high-performance, which, not being fully able to support them, would cause a significant slowdown of the system and a decline in performance.
Closing this parenthesis, the script that runs the NAOqi API, continuously sends HTTP requests to the server containing the images extracted from the frames captured by Pepper's camera. The shooting and sending of the photos occurs in synergy with the wheel that searches for the object in the environment. Then, Pepper rotates on itself by a variable angle depending on how many steps you want the robot to take to complete a lap. In this system the parameter has been set to $\pi/3.0$ which allows to complete a lap in 6 iterations. In each iteration, each RGB frame (see Fig. 4.6) is therefore sent to the server at a resolution of (320×240) to avoid further slowing down the data transfer between the robot and the server.

Figure 4.6 A RGB frame captured by the top RGB camera.

Here the heart of the object detection module is executed which consists of the following. The model that implements object detection is YOLOv10 in its small version optimized for use with a GPU since it offers a good balance between speed

and accuracy in predictions ideal for tasks such as robotics. In each phase of the approach, as in this first phase, YOLOv10 processes each frame that is forwarded to it. It identifies and localizes the objects in the image including the target object, localizes its position with respect to the reference frame of the image and returns the related bounding boxes (Fig. 4.7) that are drawn on the frame thanks to the famous OpenCV library.

Figure 4.7 A RGB frame in which the bounding box of the target object is drawn.

When used in conjunction with the search routine in the environment, if our target object is present among the various objects identified, the robot perceives that the target object has been identified in the frame just sent and blocks its search by stopping with the orientation of its base in the direction of the target object. From here on, the robot will always be oriented in the direction of the object to be grasped.

This object detection model was trained with the intent of identifying many classes of objects. Table 4.2 shows all the objects detectable by the YOLO model used. Among these, there is also the class of our target object identified with the ID 39, so it was not necessary to retrain YOLO on a new class of objects.

The approach described in this thesis can be generalized to different objects as long as they are light and small enough to be graspable by Pepper's hand according to the constraints mentioned in Section 4.1. Taking into account what has been said, as already anticipated in the previous chapters, a small bottle was chosen as the target object that was filled and colored with a light yellow acrylic tempera with a pasty consistency.

This is to prevent a possible future problem in the Depth Estimation module (see Section 4.6) due to the possible transparency of the bottle that could lead to an increased and incorrect distance estimate that in this way does not exist.

ID	Class	ID	Class	ID	Class	ID	Class
0	person	20	elephant	40	wine glass	60	dining table
1	bicycle	21	bear	41	cup	61	toilet
2	car	22	zebra	42	fork	62	tv
3	motorcycle	23	giraffe	43	knife	63	laptop
4	airplane	24	backpack	44	spoon	64	mouse
5	bus	25	umbrella	45	bowl	65	remote
6	train	26	handbag	46	banana	66	keyboard
7	truck	27	tie	47	apple	67	cell phone
8	boat	28	suitcase	48	sandwich	68	microwave
9	traffic light	29	frisbee	49	orange	69	oven
10	fire hydrant	30	skis	50	broccoli	70	toaster
11	stop sign	31	snowboard	51	carrot	71	sink
12	parking meter	32	sports ball	52	hot dog	72	refrigerator
13	bench	33	kite	53	pizza	73	book
14	bird	34	baseball bat	54	donut	74	clock
15	cat	35	baseball glove	55	cake	75	vase
16	dog	36	skateboard	56	chair	76	scissors
17	horse	37	surfboard	57	couch	77	teddy bear
18	sheep	38	tennis racket	58	potted plant	78	hair drier
19	cow	39	**bottle**	59	bed	79	toothbrush

Table 4.2 List of YOLOv10 Object Classes.

In Session 4.5.1 the architecture of YOLO is described in detail.

4.5.1 YOLO (You Only Look Once)

The YOLO (You Only Look Once) [67] family of models is based on the use of Convolutional Neural Networks (CNN) [68] which are an evolution of simple Neural Networks (NN) [69]. While neural networks are made up of fully connected layers where each neuron is connected to each neuron of the next layer, CNNs use convolutional layers in which filters (kernels) of different sizes and shapes are passed over the image to analyze and extract local patterns, such as edges or textures through a process called convolution. Specifically, during convolution, a Kernel (with associated weights) passes over the image multiplying at each iteration the weights of the kernel with the respective pixel values of the image on which it is located and adding them together to produce an output that makes up part of the filtered image. The new image filtered in this way represents a feature map and is obtained after the Kernel has passed over the entire image (see Fig. 4.8).

CNNs are particularly suitable for tasks where a lot of work is done with images, especially object detection.

An evolution of CNN is R-CNN [70], a pioneering model for object detection and semantic segmentation. In this approach, region proposals are combined with Convolutional Neural Networks to accurately localize and classify objects. R-CNN extracts features from candidate regions using a CNN, and then uses an SVM

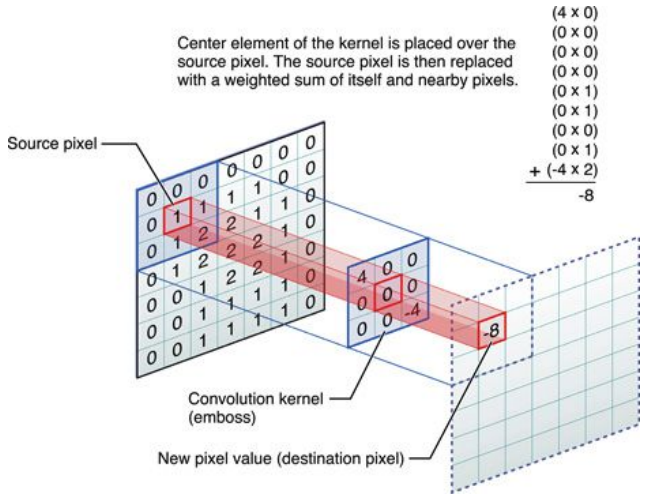


Figure 4.8 A filter (kernel) sliding over the image, applying the convolution operation to extract features.

classifier to identify classes within the regions. This model was a precursor to the later famous YOLO. In this context, YOLO is a better version of traditional CNNs, thanks to its ability to detect multiple objects in real time from an entire image in a unified detection [67] in which all the separate components that serve to detect objects are merged into a single neural network. In particular, in the basic YOLO model, all the bounding boxes of the objects are detected simultaneously taking into account the features of the entire image. The image is divided into a grid of size $(S \times S)$. Each grid cell is responsible for detecting the object whose center falls in that cell and predicts B bounding boxes, each of which has associated confidence scores that specify how confident the model is that a particular bounding box contains an object and how accurate it believes the predicted bounding box is with respect to its respective ground truth bounding box. The computed bounding boxes and confidence scores are then combined with the class probability map of the entire image that represents the probability of having a certain class of objects in a cell conditioned on whether a cell contains an object. This works through the following architecture (Fig. 4.9).

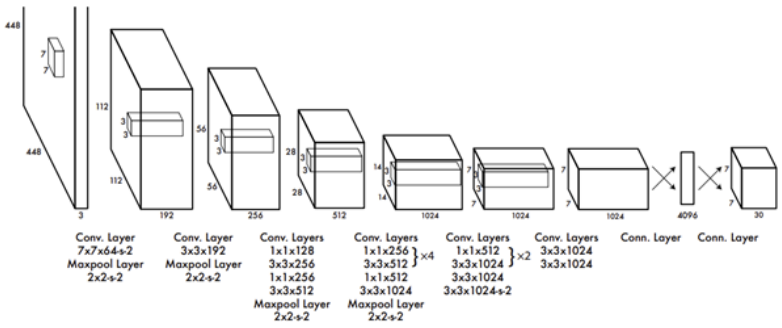


Figure 4.9 The YOLO Architecture [67].

The basic YOLO model has 24 initial convolutional layers that allow to extract the image features and 2 fully connected layers to generate the class probabilities and the corresponding bounding box coordinates. (1×1) reduction layers followed by (3×3) convolutional layers are used.

The version used in the grasping approach proposed in this thesis is YOLOv10 [71] which is found to have significant improvements over previous versions due to its training strategy and architectural improvements. The architecture of YOLOv10 is shown in Fig. 4.10.

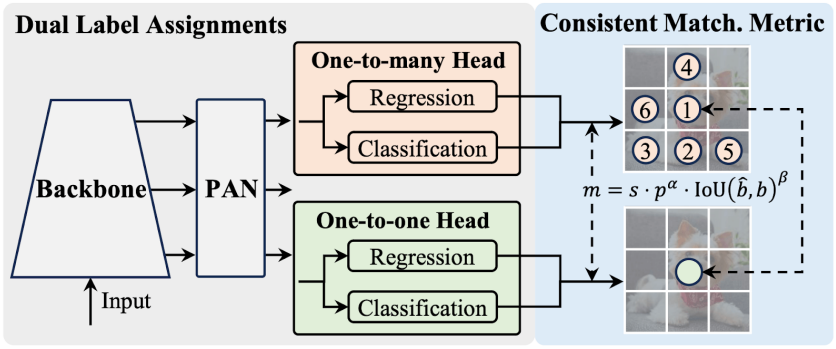


Figure 4.10 The YOLOv10 Architecture [71].

The architecture is structured in:

- **Backbone**
- **Neck**
- **Head**

The Backbone is responsible for extracting features from images using an improved version of CSPNet (Cross Stage Partial Network) [72] that guarantees a better gradient flow and reduces computational redundancy. The Nech allows to aggregate the features obtained from different scales thanks to the application of particular layers of PAN (Path Aggregation Network) [73] that allow to efficiently fuse multiscale features. The Neck is then responsible for feeding the features obtained to the head.
The head is divided into two:

- **One-to-Many Head:** allows to generate multiple predictions for each object during training in order to provide rich supervisory signals and improve the learning accuracy.
- **One-to-One Head:** traditional object detection models often use non-maximum suppression (NMS) [74] to remove duplicate bounding boxes. This head instead generates a single best prediction (a single high-quality bounding box is generated) for each object during inference to eliminate the need for the NMS post-processing step, thus reducing latency and improving efficiency and accuracy compared to previous versions of YOLO.

Furthermore, there are several innovative components in this architecture:

- **Lightweight Classification Head:** to assign labels to identified objects, YOLOv10 uses a Lightweight Classification Head consisting of two depthwise separable convolutions with a kernel size of (3×3), followed by a (1×1) convolution. This head is designed to be efficient without compromising accuracy. This is achieved by reducing the number of parameters and operations in the classification head.
- **Spatial-Channel Decoupled Downsampling:** Standard YOLO models typically use (3×3) convolutions with a stride of 2 for simultaneous spatial downsampling and channel transformation. To reduce costs YOLOv10 decouples these operations by first using pointwise convolution to adjust the channel dimension, followed by depthwise convolution for spatial downsampling.
- **Large-Kernel Convolution:** a large-kernel deepwise convolution is used which increases the kernel size of the second (3×3) deepwise convolution to (7×7) to broaden the receptive field of features that the model is able to capture.
- **Partial Self-Attention** Self-Attention excels at modeling global features but comes with high computational costs. YOLOv10 optimizes this through its Partial Self-Attention (PSA) mechanism. It starts by dividing channels after applying a (1×1) convolution, splitting the features into distinct parts. Then, self-attention is selectively applied to only one portion of these split features, reducing processing demands. The features are recombined efficiently using another (1×1) convolution. To further enhance efficiency, PSA reduces the dimensions of the query and key to half the size of the value in the Multi-Head Self-Attention (MHSA) setup.

4.6 Depth estimation module

The version of Pepper present in the VisionLab is equipped with stereo cameras. Therefore, a first approach to try to solve the problem of understanding the distance of an object from the robot was to try to estimate it through stereo vision. This technique allows to estimate the depth of a point-like object 'P' from the camera using two cameras. The depth is estimated by exploiting the difference in position (disparity) of a point between two images captured from slightly different angles. Two cameras, placed at a known distance (baseline), observe the same scene: the closer an object is, the greater the apparent displacement between the two. This approach was therefore tried. However, implementing the depth estimation module in this way did not lead to good results. This solution was immediately abandoned because there were problems in calibrating the cameras producing distorted reconstructed images. But it does not end here. In our version of the robot there is the possibility to reconstruct a depth map from stereoscopic images directly using the NAOqi API, an attempt was made but the result of the depth map was very poor. Given these problems, we opted for the two 2D RGB cameras, abandoning the idea of using both the reconstructed depth of the API and the stereo cameras, because we do not know

if the lack of effectiveness of the calibration is due to a software or hardware problem, since even the depth maps obtained from them were not accurate. Observing more carefully the depth map obtained from the NAOqi API, and comparing it with the corresponding original images (see Fig. 4.11 and Fig. 4.12), it is possible to notice a low accuracy and significantly marked inaccuracies.

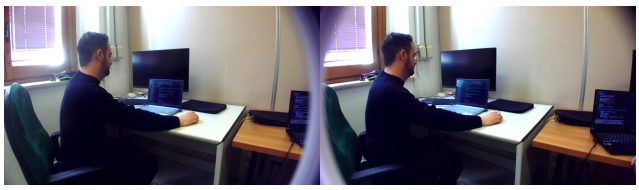


Figure 4.11 View from two stereo cameras.

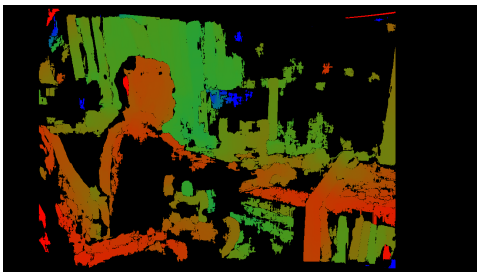
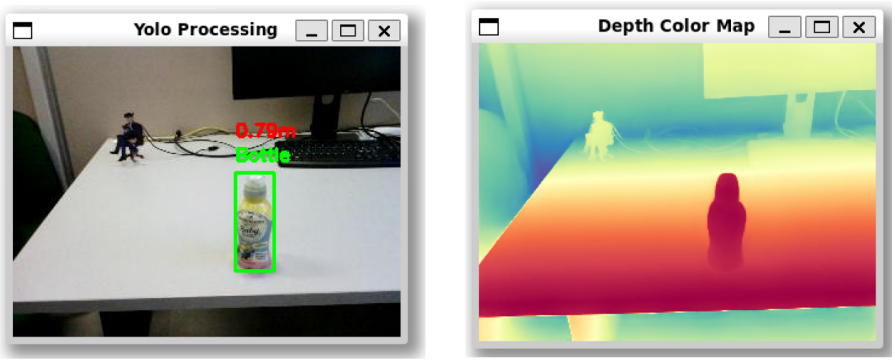


Figure 4.12 Depth map generated using stereo cameras and NAOqi APIs.

In particular, dark objects or shapes that tend to black, have values for the Depth Map that are completely wrong. The pixels corresponding to these areas have values that locate them much deeper than their actual position in space as if they were entities that are at the same level as the background when in fact they are objects in the foreground as in the case of the monitor or especially the dark blue sweater that should be closer to the camera. Another example of error are the blue spots located in the image that do not seem to have a real correspondence in reality but rather seem apparently positioned randomly. In fact, these areas should correspond to areas that are further away than the orange ones but in many cases this does not happen. Consequently, the aforementioned depth map turns out to be too poor to contribute to solving the grasping problem for robots that instead require high precision and coherence in every point of the generated depth map. That said, the two 2D cameras used, not being stereo cameras and being positioned in distant and non-aligned points, do not allow the use of classic techniques such as the triangulation described above but there is the need to use more complex and modern Deep Learning approaches. It was chosen to exploit the computational power of the State of Art model for depth estimation Depth Anything V2 [75] whose algorithm is based on a monocular method. Monocular depth estimation methods try to calculate the distance of objects using a single image, without the multiple viewpoints of stereo techniques. This type of algorithm analyzes cues such as texture gradients, object size, shadows, and perspective, addressing inherent ambiguity to generate depth maps, enhanced by deep learning. It is inspired by human single-eye depth perception, based on linear perspective, texture, and motion parallax. The

choice of version 2 of the model is due to experiments carried out on a simulator for Pepper and other robots that is part of a parallel work to this one in which the models of versions 1 and 2 were compared and the superiority of the second model emerged. It was therefore a choice dictated by preliminary experimentation.

As anticipated in Chapter 4.5, our target object is a small plastic bottle that has been filled with acrylic paint to make it non-transparent. This choice was necessary to try to overcome the challenges that depth estimation models face when detecting the depth of transparent objects. These systems look for visual cues within the image such as texture, edges and color variations to estimate the distance of an object. However, transparent surfaces often do not have distinct textures, which prevents accurate detection. In particular, light refraction and reflection can create problems: when light passes through the object, its path tends to distort the perceived image, while reflections from the environment can be mistaken for the background even if they are part of an object at a different depth. Depth Anything V2 can also suffer from object transparency especially when there is not good lighting in the room. Consequently, to make the results more accurate and stable, this choice was made. Let's now go into detail on how Depth Anything V2 was used inside the depth estimation module. In general in the system, every time the bottle is located with the object detection module inside the system, the depth estimation module is also triggered to estimate its position from the camera. Depth Anything V2 runs on the Flask server together with YOLOv10. After receiving the RGB image (of size $(h \times w \times 3)$) captured by Pepper, Depth Anything V2 computes the depth map for each pixel of the original image, that is, it generates an image of size $(h \times w)$ that will have in correspondence with each pixel of the original image the value of the distance of each point of the image from the camera (see Fig. 4.13 generated with OpenCV).



(a) (b)

Figure 4.13 From right to left: **(a)** bounding box generated by YOLOv10 applied to a frame from the top 2D-camera, **(b)** depth map generated using the top 2D-camera

At this point it communicates with the object detection module from which it receives the coordinates of the bounding box of the bottle detected in the foreground as shown in Fig. 4.13. These coordinates allow us to obtain the central position of the object from the original image thanks to which the depth estimation module

extracts from the depth map the distance corresponding to this central point which is certainly a point that belongs to the surface indicative of the entire surface of the bottle since we assume that the object to be grasped has a convex structure, in the sense that only objects that are not pierced in the center are considered. This allows us to estimate the distance along the x coordinate (remember that the depth is expressed on the x-axis in the convention used for Pepper) in meters of the bottle with respect to the reference system of the camera. The coordinates along the y and z axes are determined in the coordinate mapping module illustrated in the chapter. In implementing Depth Anything V2 it was necessary to do a fine-tuning of the parameters. First, a pre-trained Depth Anything V2 model was engaged for depth estimation. In this system, a pre-trained encoder on the synthetic Hypersim dataset [76] was used for indoor depth estimation rather than trained on the Virtual KITTI dataset [77] for outdoor depth estimation since Pepper is typically intended, as in our case, to move in indoor environments. In this context, there is a parameter called `max_depth` that plays an essential role in determining the numerical distance values, allowing the model to align with the desired metric scale and adapt to specific environments, such as indoor or outdoor settings. For indoor scenarios, the developers of Depth Anything [78] suggest setting `max_depth` to 20, capping depth estimates at 20 meters. This ensures the output aligns with the typical dimensions of indoor spaces. In contrast, setting `max_depth` to 80 adjusts the model for broader outdoor environments. It is also possible to configure the model to the Depth Anything model that has been trained on a number of different parameters. With the encoder configuration `"vits"`, the small model with 24.8 M parameters is used, with `"vitb"` the base model with 97.5 M parameters and with `"vitl"` the Large model that uses 335.3 M. In this system the large model was used which allows to work with much more accurate depth maps. In section 4.6.1 the architecture of Depth Anything V2 is explained in detail.

4.6.1 Depth Anything V2

The Depth Anything V2 model [75] stands out for its innovative approach to monocular depth estimation, integrating advanced methodologies to overcome the limitations of traditional real-world systems. One of its peculiarities is the use of synthetic images in combination with labeled real data that ensure greater accuracy and detail in depth annotations. Its architecture, based on DINOv2, is designed to be scalable in order to offer models that adapt to the different computational needs of developers. To bridge the gap between synthetic and real data, the model introduces a pseudo-labeling process, in which an extended set of unannotated real images is enriched with automatically generated depth estimates. In addition, Depth Anything V2 introduces DA-2K, a new benchmark characterized by diverse scenes and precise annotations, useful for evaluating performance in heterogeneous contexts.

The model works in three main phases (see Fig. 4.14). Initially, a large `"teacher"` model is trained exclusively on high-quality synthetic datasets to acquire a solid knowledge base on depth estimation. Subsequently, this model generates pseudo-realistic depth labels for a large set of real unannotated images. This way, the diversity of the training set is increased. Finally, smaller `"student"` models are trained on these pseudo-labeled data that allow to effectively generalize to real

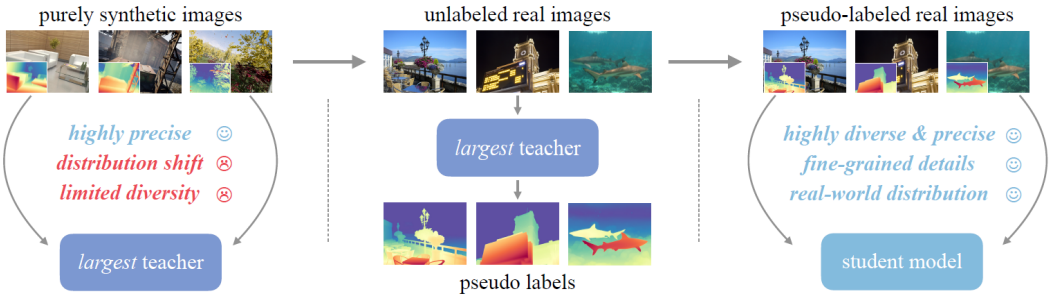


Figure 4.14 The three phases of The Depth Anything V2 [75].

scenarios. The main component of the architecture is the DINOv2 encoder [79] which, as already mentioned in the previous Section 4.6 is available in different scales: small, basic, large and giant. The latter, although mentioned in the paper [75], has not yet been released, and anyway, with 1.3 Billion parameters, it would be intractable for our computational resources. In any case, the encoder extracts the most important features from the image while a DPT decoder [80] is in charge of transforming these features into detailed depth maps. The training process is developed in two distinct phases. In the first, the teacher model, based on a DINOv2-G (giant) encoder, is optimized on synthetic datasets using images scaled to (518×518) pixels. In the second phase, the small student models refine their skills on real pseudo-labeled data, using the Adam optimizer with differentiated learning rates (5×10^{-6} for the encoder and 5×10^{-5} for the decoder). The loss function combines a scale term, to ensure consistency in the estimates, and a gradient matching term, to improve the sharpness of the predictions matching gradients between predicted and ground truth depth maps. Depth estimation is performed by converting a 2D image into a depth map, representing the distance of each pixel from the camera. The encoder extracts multi-scale features, while the decoder processes them to generate a high-resolution map, further refined by post-processing techniques such as bilateral filtering. Thanks to this ability to combine information from synthetic and real data, Depth Anything V2 is therefore a very robust model that allows to approach very complex challenges such as making depth predictions on transparent or reflective surfaces. However, even if much improved compared to the previous version Depth Anything V1, the predictions on this type of surfaces still cause several inaccuracies, for this reason it was chosen to make the bottle non-transparent.

4.7 Coordinate mapping module

In this phase, the goal is to express the coordinates of the center of the bottle's bounding box with respect to the reference frame of the robot's base to understand how far the robot must travel to reach the object and grab it later. Thanks to the collaboration of the object detection and depth estimation modules, we know the coordinate along z (which in the robot's convention is x) with respect to the camera that captured the frame that is being processed and the coordinates along x (which in the robot's convention corresponds to the y-axis) and y (which in the robot's convention corresponds to the x-axis) expressed instead in the reference frame of

the 2D image (we know the position of the center in terms of row and column of the image). Therefore, first of all, we need to convert the 2D coordinates x and y into the corresponding 3D coordinates expressed with respect to the camera's reference frame. In this context, we consider the classic convention to express the concept in a more fluid way. Let x be the horizontal axis, y the vertical axis and z the axis indicating the depth perpendicular to the plane formed by the x and y axes and let f be the focal length of the cameras (obtainable from Pepper's documentation [81]), it is possible to convert the 2D coordinates (x, y) expressed in the image reference system into 3D coordinates (X, Y, Z) expressed with respect to the camera reference frame in the following manner:

$$x = f \cdot \frac{X}{Z} \tag{4.1}$$

$$y = f \cdot \frac{Y}{Z} \tag{4.2}$$

In this equation (X, Y, Z) are the 3D coordinates of the center of the object we want to calculate. The value of Z has already been computed using Depth Anything V2. Our unknowns are instead X and Y, which we can calculate using the inverse formulas of the following equations (4.3), (4.4).

$$X = \frac{x \cdot Z}{f} \tag{4.3}$$

$$Y = \frac{y \cdot Z}{f} \tag{4.4}$$

As you can see, the orientation of the bottle is not expressed here because it would require information that takes into account the possible rotations around the 3 Cartesian axes which would add a further difficulty to the problem. Therefore in this approach we assume that the bottle is placed perfectly vertical with respect to the horizontal plane.
Once we have computed the position of the center of the bottle in 3D space coordinates, we come into focus of the coordinate mapping module. Remembering the focus of this Section, we need to find a way to express the found 3D coordinates of the object, which at the moment are expressed with respect to the reference frame of the camera, in coordinates expressed with respect to the reference frame of the robot base. As already described in detail in Chapter 2 we need to compute transformations between reference frames.

Let's define the different reference frames mathematically to give a more logical explanation to what happens.

- **Reference frame 0:** represents the 6D coordinates (position and orientation) of the bottle and are expressed with respect to reference frame 1.
- **Reference frame 1:** represents the 6D coordinates of the camera and are expressed with respect to reference frame 2.
- **Reference frame 2:** represents the reference system of the Pepper basis which is the final reference frame with respect to which we want to express the pose of reference frame 0.

The goal is to find the transformation T_0^2, which represents the pose of reference frame 0 with respect to reference frame 2.

Let us now ascertain what transformations we must take into consideration:

- **Transformation from 0 to 1:** we denote this transformation as T_0^1, which represents the pose of reference frame 0 with respect to reference frame 1.
- **Transformation from 1 to 2:** we denote this transformation as T_1^2, which represents the pose of reference frame 1 with respect to reference frame 2.

To find the transformation that goes from 0 to 2, T_0^2, it is necessary to compose the two transformations T_0^1 and T_1^2. The composition of transformations in homogeneous coordinates is obtained by multiplying the transformation matrices:

$$T_0^2 = T_1^2 \cdot T_0^1 \tag{4.5}$$

Remembering what was written in Section 2.2, the transformation matrices can be represented in the following way:

$$T_0^1 = \begin{bmatrix} R_0^1 & t_0^1 \\ 0^T & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} R_1^2 & t_1^2 \\ 0^T & 1 \end{bmatrix}$$

where:

- R_0^1 and R_1^2 are (3×3) rotation matrices.
- t_0^1 and t_1^2 are (3×1) translation vectors.
- 0^T is a row vector of zeros $[0, 0, 0]$.

The multiplication of the two matrices results in:

$$T_0^2 = \begin{bmatrix} R_1^2 & t_1^2 \\ 0^T & 1 \end{bmatrix} \cdot \begin{bmatrix} R_0^1 & t_0^1 \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} R_1^2 \cdot R_0^1 & R_1^2 \cdot t_0^1 + t_1^2 \\ 0^T & 1 \end{bmatrix} \tag{4.6}$$

The resulting matrix T_0^2 represents the homogeneous transformation that describes the pose of reference frame 0 with respect to reference frame 2. In particular the resulting rotation matrix is:

$$R_0^2 = R_1^2 \cdot R_0^1 \tag{4.7}$$

Similarly resulting translation vector is obtained as follows:

$$t_0^2 = R_1^2 \cdot t_0^1 + t_1^2 \tag{4.8}$$

So, starting from the 6D pose of the bottle p_0 expressed in reference frame 0, we can transform it into reference frame 2 (i.e. the reference frame of the robot base) using the transformation T_0^2:

$$p_2 = T_0^2 \cdot p_0 \tag{4.9}$$

Where p_0 and p_2 are vectors in homogeneous coordinates:

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}, \quad p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

As a preliminary aspect to be able to carry out the transformations, the pose of the camera reference frame with respect to the position of the robot base was calculated through the APIqi in particular thanks to the `getPosition` method of the `motion_service` service described in the table 4.1.

4.8 Path planning module

The path planning module represents the heart of the system, coordinating the other four modules of the proposed approach. It allows to plan all the movements necessary for the robot to move from an initial point A to a final point B, executing them in such a way as to optimize distance, position and orientation with respect to the desired target. The goal is to obtain the best possible position for the grasp. This result is achieved thanks to a continuous interaction with the other modules. In this system, a two-phase algorithm is used that operates in real time. In the first phase, the movement of the robot is guided by the features extracted from the top camera and consists of a preliminary and approximate approach towards the bottle. In the second phase, instead, the bottom camera comes into play, which offers a more detailed view of the scene, in particular of the bottle placed on the table. For this phase, the robot must be sufficiently close to the table for the bottle to appear in the frames captured by the camera, since it is oriented downwards. The steps of this method are summarized in the three following algorithms, which offer an explanatory overview of how path planning works.

Algorithm 1 Path Planning algorithm for grasping

```
1: Input: id_camera_top, id_camera_bottom, proximity_threshold, max_error
2: Output: None
3: procedure PathPlanning(id_camera_top, id_camera_bottom)
4:     distance ← PathPlanning1(id_camera_top)
5:     PathPlanning2(distance, proximity_threshold, max_error,
6:         id_camera_bottom)
7: end procedure
```

Algorithm 1, illustrated above, is the routine that coordinates the other two algorithms. In sequence, it first calls Algorithm 2, which summarizes the main steps of the first phase, and then activates Algorithm 3, which uses the output of Algorithm 2 as input. In the pseudocodes presented, the implementation details of the other modules are not explored in depth; instead, attention is placed on the critical points of the process.

Algorithm 2 Path Planning 1

```
1: Input: id_camera
2: Output: distance
3: procedure PathPlanning1(id_camera)
4:     robot_position ← GetCurrentPosition()
5:     object_position_top ← GetObjectPositionTopCamera(id_camera)
6:     distance ← calculate_distance(robot_position, object_position_top)
7:     MoveRobot(distance)
8:     robot_position ← GetCurrentPosition()
9:     object_position_top ← GetObjectPositionTopCamera(id_camera)
10:    return distance
11: end procedure
```

In Algorithm 2, which describes the first phase, the robot determines its position in space and the pose of the bottle, information provided by the previous modules. At this stage, the position of the bottle is expressed with respect to the reference system of the base of the Pepper robot, allowing the calculation of the distance to travel to reach the target position. The robot then approaches the bottle and stops near it. Before concluding, Algorithm 2 recalculates the position of the bottle and the robot, returning the residual distance from the object at the point where it stopped.

At this point, Algorithm 3 is activated, which receives as input the distance calculated by Algorithm 2, along with a `proximity_threshold`, a `max_error` and the `bottom camera ID`. Algorithm 3 provides two possible scenarios: If the distance from the bottle is less than or equal to the `proximity_threshold`, the robot is close enough to attempt to grab the bottle.

Otherwise, the robot is not yet close enough to the target and requires further approach. In this situation, the robot recalculates its position and that of the object, using the frames of the bottom camera, which offers a clearer view thanks to the greater proximity compared to the initial position. The distance between the two is recalculated and the algorithm is called recursively, updating the distance parameter by subtracting the `proximity_threshold`.

If, on the other hand, the distance is less than the `proximity_threshold`, the robot is almost ready to grab. However, before the last step, the algorithm performs a refinement of the position through an iterative cycle. In this cycle, Pepper calculates the position of the bottle from the bottom camera frames and compares it with the ideal grasp position, which is the ideal position to successfully grasp the object. The error is then determined, which is the difference between the ideal grasp position and the current position of the bottle. If this error is less than or equal to `max_error`, the discrepancy is small enough to enable the activation of the inverse kinematics module, responsible for grasping. This module calculates the values to assign to the joints to move the robot arm and hand and complete the action. If the error exceeds `max_error`, the robot is not yet in the optimal position; therefore, an adjustment is made with small movements to reduce the difference between the ideal grasp position and the current position of the bottle. The cycle repeats until the error drops below the `max_error` threshold. The final module involved is described in Section 4.9.

Algorithm 3 Path Planning 2

```
1: Input: distance, proximity_threshold, max_error, id_camera
2: Output: None
3: procedure PathPlanning2(distance, proximity_threshold, max_error,
   id_camera)
4:     if distance ≤ proximity_threshold then
5:         repeat
6:             object_position_bottom ←
7:                 GetObjectPositionBottomCamera(id_camera)
8:             ideal_object_grasp_position ← GetIdealPosition()
9:             error ← CalculateError(robot_position, ideal_object_grasp_position)
10:            if error ≤ max_error then
11:                joint_values ←
12:                    CalculateInverseKinematics(object_position_bottom)
13:                ExecuteArmAndHandMovement(joint_values)
14:                break
15:            else
16:                robot_position ←
17:                    AdjustPosition(object_position_bottom, ideal_object_grasp_position)
18:            end if
19:        until error > max_error
20:    else
21:        robot_position ← GetCurrentPosition()
22:        object_position_bottom ←
23:            GetObjectPositionBottomCamera(id_camera)
24:        distance ← calculate_distance(robot_position, object_position_bottom)
25:        PathPlanning2(
26:            distance−proximity_threshold, proximity_threshold, max_error,
27:            id_camera)
28:    end if
29: end procedure
```

4.9 Inverse kinematics module

The final phase of this grasping system for humanoid robots consists of the inverse kinematics module. As it has been described in detail in Section 2.3, the inverse kinematics allows the robot to move the joints of the arm kinematic chain to the target position that localizes the object to be grasped, which in the specific case of this thesis is a bottle. Given the desired pose of the end-effector, the inverse kinematics returns the set of joint values that allow the chain to bring the end-effector to the desired pose. In this approach, the inverse kinematics has been implemented using Playful Kinematics [82] which is based on the Cyclic Coordinate Descent (CCD) [83] which is an iterative method for implementing the inverse kinematics that aims to minimize the error between the current pose of the end-effector and the target one by cyclically updating the angles of each joint involved. In Section 4.9.1 CCD is described in detail. Playful kinematics allows us to calculate inverse kinematics for robots that have a number of degrees of freedom between 5 and 8 (inclusive). In the case of robots with fewer or more degrees of freedom in its effectors it is necessary to modify the source code to adapt it to the specific case. This framework calculates inverse kinematics in python and uses C++ for the backend. It runs exclusively on Ubuntu and communicates with ROS in order to use KDL (Kinematics and Dynamics Library) [84] that performs the kinematics calculations. It is therefore necessary to install several ROS packages. Furthermore, the library requires that a URDF file be provided, which consists of an XML file that can contain the specifications of the robot, sensors, environments and so on. Thanks to the use of CCDs, Playful kinematics is designed to favor convenience and reactivity over precision.

Playful Kinematics already supports inverse kinematics for Pepper by default, which has also been exploited in this work. Specifically, as already highlighted several times, especially in Section 3.2, Pepper has an arm with only 5 degrees of freedom (ShoulderPitch, ShoulderRoll, ElbowYaw, ElbowRoll, WristYaw) and hands with no degrees of freedom since the fingers cannot move separately, but can only open or close completely all together. However, with only 5 DoF, Pepper could only move in a narrow and very limited workspace. Consequently, the authors of Playful, to calculate the inverse kinematics for the kinematic chain of the arm that ends with the hand effector, also take into account 3 other joints in addition to the 5 of the arm. These 3 auxiliary joints are HipPitch, HipRoll and KneePitch and give Pepper greater flexibility by increasing the width of its workspace. In fact, this gives the robot the ability to bend its torso forward or backward, to the right or left to find the best position to grasp the object. Also in our IK module the inverse kinematics is obtained starting from these 8 degrees of freedom.

In our approach, before calculating the possible solutions for the inverse kinematics equations, a preliminary step is performed in which, after the robot has refined its position to allow for the most optimal possible grasp of the object, Pepper's arm is brought to a predefined position in which the hand is positioned at a height just below the shoulder with the elbow contracted to increase the possibility that the inverse kinematics can find a solution. At this point Pepper opens the hand and sends via an HTTP POST request to the Flask server, which manages the Playful

Kinematic, the values of the current reference posture that the inverse kinematics needs to understand the reference pose with respect to which the inverse kinematics must calculate a solution. At this point the actual kinematics is calculated using the IK method by Playful `get_posture(left_hand, grasp_xyz, hand_orientation, blocked_joints)` which takes as parameters the parameter `left_hand` to indicate with which hand (boolean False for right hand or True for left hand) you want to perform the grasping, the parameter `grasp_xyz` which indicates the desired position in which you want to bring the end-effector of the kinematic chain (which in the grasping system proposed in this thesis represent the 3D coordinates calculated in the coordinate mapping module), the parameter `hand_orientation` to indicate with which orientation of the hand you approach the grasp and a final parameter `blocked_joints` which specifies the joints you want to keep blocked when calculating the solutions for the inverse kinematics equations.

Figure 4.15 Snapshot of the phase where Pepper is computing the inverse kinematics to perform the grasping of the bottle.

Figure 4.16 Snapshot of the moment when Pepper successfully grasped the bottle after computing the inverse kinematics.

If you keep some joints blocked, finding a solution for the inverse kinematics becomes increasingly difficult because the number of degrees of freedom that the robot can use to grasp the bottle is reduced. The inverse kinematics calculation performed in this way returns 3 results as output: the first is a boolean value that is equal to True if the inverse kinematics was successful in finding at least one solution to reach the target; the second output specifies how close the calculated solution is to the ideal one while the third result returns the values of the joints calculated by the inverse kinematics to be assigned to the respective actuators that make the robot hand move to the target position. If no solution is found (an exact solution is not found if the grasp score is above a certain very low threshold), the returned posture is still meaningful, representing the robot's best effort to approach the target which in many cases is sufficient since the tolerated error to be considered an exact solution is really low.

4.9.1 Cyclic Coordinate Descent (CCD)

Although there are different methods for solving the inverse kinematics problem, the cyclic coordinate descent (CCD) method is one of the most computationally fast and least complex to put into practice. It is an iterative numerical algorithm that is straightforward and intuitive to implement while also boasting the added advantage of not requiring any complex matrix math decomposition. Cyclic Coordinate Descent is a heuristic iterative approach that is ideal for real-time applications such as games because it has a very low computational overhead per joint iteration. For the same reasons, CCD also finds applications in fields like robotics, digital animation, and physical simulation, where rapid computation of joint configurations for manipulators or virtual characters is essential. Despite its strengths, the CCD method has some limitations. For instance, it may not always yield the most natural joint configuration, as it relies on a heuristic approach. Additionally, in the presence of complex kinematic chains, it might require further adjustments to handle conflicts or specific constraints.

Fundamentally, when solving the inverse kinematics problem, we aim to determine a comprehensive set of joint angles that positions the end-effectors at intended locations. At any given moment, we can use this complete set of joint angles to compute the distance error between the end-effectors and their targets. This is achieved by propagating each link's angle and length forward from the base of the connected hierarchy to the end-effectors. Once calculated, we can assess the discrepancy between the end-effectors' positions and the target locations, as illustrated, for example, in Fig. 4.17.

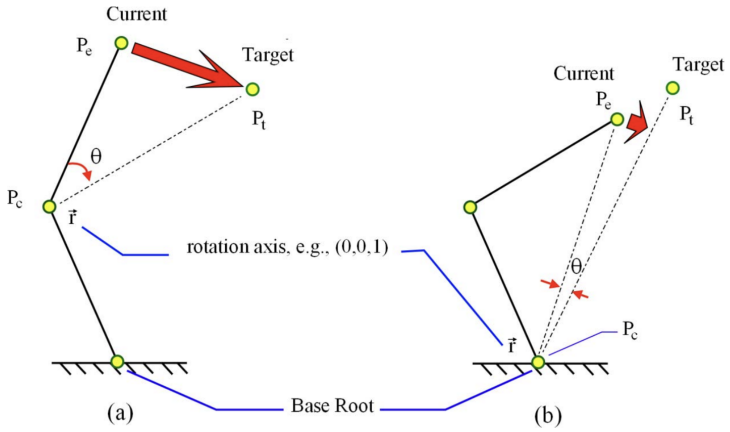


Figure 4.17 Cyclyc coordinate descent on a simple two-link chain with a single end-effector [83].

After each iterative update, the algorithm measures the distance between the end-effector and the target to decide if it is close enough and should exit. Additionally, to prevent infinite recursive loops caused by unreachable or conflicting goals, the algorithm establishes a maximum iteration limit. Fundamentally, the base of the inverse kinematics (IK) chain remains fixed, while each subsequent link is iteratively rotated around a specified axis and angle. This process aims to position the end-effector as near as possible to the target. The axis and angle for these rotations are determined using the following equation:

$$\cos(\theta) = \frac{P_e - P_c}{\|P_e - P_c\|} \cdot \frac{P_t - P_c}{\|P_t - P_c\|} \tag{4.10}$$

$$\vec{r} = \frac{P_e - P_c}{\|P_e - P_c\|} \times \frac{P_t - P_c}{\|P_t - P_c\|} \tag{4.11}$$

where the values P_e, P_c, and P_t refer to the positions in Fig. 4.17.

Chapter 5

Experimental results

In this chapter, we will conduct an analysis of the experiments performed to evaluate the system. In Section 5.1, the ability of YOLOv10 to detect bottles in photos taken with Pepper's cameras will be tested. Section 5.2 will focus on the application of Depth Anything v2 to the same photos considered earlier, to assess the accuracy of measuring the distance between the bottle and the robot's camera. Finally, in Section 5.3, we will measure the robustness of the developed system by evaluating the success rate of grasping the target object.

5.1 Object detection module evaluation

To carry out the experiments aimed at evaluating the object detection and depth estimation modules, a dataset of images captured with the two Pepper cameras was created. The dataset includes 150 photos taken with the top camera and 150 with the bottom camera. Going into more detail, the 150 photos for each camera were obtained by immortalizing three different bottles (Fig. 5.1), for each of which 50 images were taken.

Figure 5.1 The three different bottles used for the experiments.

To allow the evaluation of the depth estimation module, each image was labeled with the value of the real distance between the bottle and the robot's camera, which acts as ground truth. This value was obtained by manually measuring the distance between the target and the camera with a tape measure. The images that make up the dataset were captured not only at different distances, but also with various inclinations. Furthermore, they were acquired in two environments chosen to conduct

all the experiments: a room with mainly natural light and another with exclusively artificial light.

As discussed in detail in Section 4.5, in the object detection module, we decided to use YOLOv10 in its small version. In this evaluation, we will assess how well the model can identify the chosen target object (a bottle) across different scenes. Since a bottle is always present in the captured photos, we manually created a new dataset consisting of images organized into two folders: "bottle" and "not bottle". The images in the "bottle" folder represent the positive class (class 1), while those in the "not bottle" folder represent the negative class (class 0). We populated the "not bottle" folder with images of plates, vases, cups, mugs, and elongated fruits to introduce elements that could confuse the model, simulating structures similar to those of a bottle (Fig. 5.2).

Figure 5.2 Images from the "not bottle" class.

First of all, the following were calculated:

- **False negative**: occurs when the model fails to detect a bottle that is actually present in the photo.
- **True negative**: occurs when there is no bottle in the photo, and the model correctly detects its absence.
- **False positive**: occurs if there is no bottle in the photo, but the model erroneously detects one.
- **True positive**: the model detects the presence of a bottle, and there is indeed a bottle in the photo.

We also calculated the confusion matrix, shown in Fig. 5.3 of the obtained results:

- 300 images of the "not bottle" class were correctly classified as not containing bottles.
- 0 images of the "bottle" class were erroneously classified as "not bottle."
- 3 images of the "not bottle" class were erroneously classified as containing a bottle.
- 297 images of the "bottle" class were correctly classified as containing a bottle.

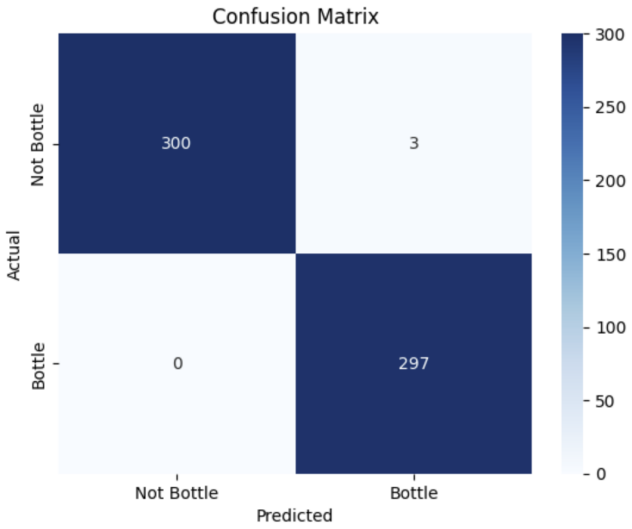


Figure 5.3 Confusion matrix.

The following metrics were also calculated:

- **Accuracy**: the proportion of correct predictions out of the total predictions. Its calculated value here is 0.995.
- **Recall**: measures the proportion of correctly identified positive instances out of the total actual positive instances. It has a value of 1.0 in this evaluation.
- **Precision**: measures the proportion of correct positive predictions out of the total positive predictions made by the model. It has a value of 0.99.

These data confirm that the model exhibits high accuracy in detecting bottles and a low percentage of false positives. In conclusion, YOLOv10 has proven to be a robust and effective model for this application scenario.

5.2 Depth estimation module evaluation

In this Section, we reuse part of the dataset utilized in Section 5.1, which consists of 300 images captured by Pepper's two cameras. As previously mentioned, the distance between the robot's camera and the target object has been manually annotated for each image. The goal of the experiments in this Section is to perform fine-tuning of the `max_depth` parameter in Depth Anything v2, which implements the depth estimation module used in this system, and to assess how robust this method is for metric depth estimation in a system where precision is critical. The `max_depth` parameter, as noted in Section 4.6, is highly significant because it allows the model to align with the desired metric scale and adapt to specific environments, such as indoor or outdoor settings. The dataset has been divided into two parts: images captured with the bottom camera and images captured with the top camera. In evaluating the depth estimation module, the results will be reported while taking into account this division and the inherently different characteristics of the two

cameras. The bottom camera has a narrower field of view, though more specific and close-range, whereas the top camera provides a longer-range view. Consequently, the `max_depth` value for metric depth estimation must differ for each camera type. For both cameras, we will analyze the variation in the root mean square error (RMSE) with respect to changes in `max_depth` and the distance between the robot's camera and the object. RMSE is defined as the measure of the differences between the values predicted by a model and the values actually observed. In this formula, N denotes the number of observations:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}} \tag{5.1}$$

Since the data were acquired manually, for the top camera, which has a very wide field of view, very different distance values were calculated. For this reason, these values were grouped into ranges of similar values; in this case, for each cell of the table the average of the RMSE of the individual distances belonging to each group will be reported. The ranges considered are: from 40 cm to 60 cm, from 61 cm to 1 meter, and from 1 meter and 1 cm to 1 meter and 50 cm. As for the images captured with the bottom camera, however, since they all have a rather similar distance from the robot, they were grouped based on the distance value without calculating further additional averages.

In Table 5.1 are shown the results for bottom camera.

Target distance (m)	**max_depth**								
	5	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**20**
0.30	0.187	0.142	0.097	0.055	0.028	0.052	0.094	0.138	0.160
0.31	0.195	0.150	0.104	0.060	0.021	0.040	0.083	0.129	0.151
0.32	0.205	0.160	0.115	0.072	0.037	0.043	0.081	0.125	0.148
0.33	0.155	0.155	0.108	0.070	0.058	0.085	0.128	0.176	0.200
0.34	0.217	0.168	0.120	0.073	0.035	0.046	0.089	0.136	0.160
0.35	0.222	0.171	0.120	0.070	0.025	0.041	0.089	0.140	0.165
0.36	0.219	0.164	0.110	0.063	0.050	0.087	0.138	0.193	0.221
0.37	0.237	0.184	0.131	0.079	0.033	0.039	0.088	0.140	0.167
0.38	0.245	0.191	0.138	0.087	0.044	0.047	0.092	0.144	0.170
0.39	0.251	0.196	0.142	0.091	0.050	0.055	0.099	0.152	0.179
0.40	0.254	0.197	0.142	0.091	0.060	0.077	0.124	0.178	0.207
0.41	0.277	0.224	0.171	0.118	0.065	0.015	0.043	0.095	0.122
0.42	0.270	0.211	0.153	0.097	0.055	0.063	0.110	0.167	0.196
0.43	0.275	0.213	0.154	0.098	0.058	0.072	0.122	0.181	0.211
0.44	0.299	0.243	0.187	0.132	0.080	0.042	0.060	0.109	0.136
0.45	0.297	0.236	0.176	0.117	0.061	0.039	0.082	0.140	0.170
0.46	0.318	0.261	0.205	0.149	0.093	0.042	0.036	0.085	0.113
0.47	0.313	0.251	0.191	0.135	0.090	0.080	0.113	0.166	0.195

Table 5.1 Results of Depth Anything v2 with images captured from bottom camera expressed in RMSE for various target distances and values of the `max_depth` parameter.

We can easily notice that the best values of `max_depth` for the bottom camera are 13 and 15, as can also be observed in Fig. 5.4.

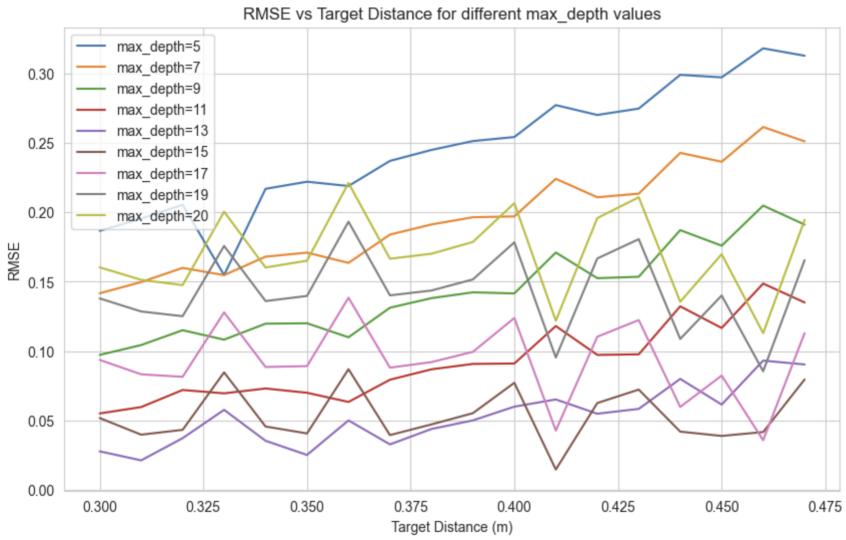


Figure 5.4 `max_depth` comparison for bottom camera images.

If we consider the average RMSE, the best overall value is 13, which performs better up to a distance of 0.40. Beyond that point, however, its performance declines, and `max_depth`=15 starts to become more consistent. The choice between the two values largely depends on the specific environment in which they are being applied, which may have characteristics that are more or less ideal for leveraging one value or the other, favoring either greater proximity or distance. That said, since the top camera is often used above 40 cm, we can generally state that 13 is the most effective value. Typically, by analyzing the other data, it can be observed that as the distance increases, the RMSE value rises accordingly.

In Table 5.2 are shown the results for top camera.

Target distance	**max_depth**								
	5	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**20**
40 cm - 60 cm	0.339	0.259	0.184	0.122	0.103	0.145	0.214	0.291	0.332
61 cm - 1 m	0.506	0.423	0.348	0.289	0.257	0.261	0.300	0.363	0.400
1.01 m - 1.5 m	0.971	0.874	0.780	0.690	0.608	0.535	0.477	0.439	0.429
1.51 m - 2 m	1.575	1.496	1.417	1.338	1.259	1.180	1.102	1.025	0.986

Table 5.2 RMSE values for Depth Anything v2 at different target distance ranges and max_depth values.

Regarding the variation of the average RMSE values as the distance ranges change, we can observe a clear trend, particularly evident in the figure: as the distance increases, the `max_depth` value must also increase to achieve an accurate evaluation of the metric depth estimation. The worst values overall are obtained when very low `max_depth` values are used for large distances, likely because this overly restricts the estimation in a sharp and premature manner, thus altering the

resulting metric values. Furthermore, it can be noted also from Fig. 5.5, that the optimal `max_depth` value is 20, as it never yields an average RMSE value greater than or equal to one for any distance range.

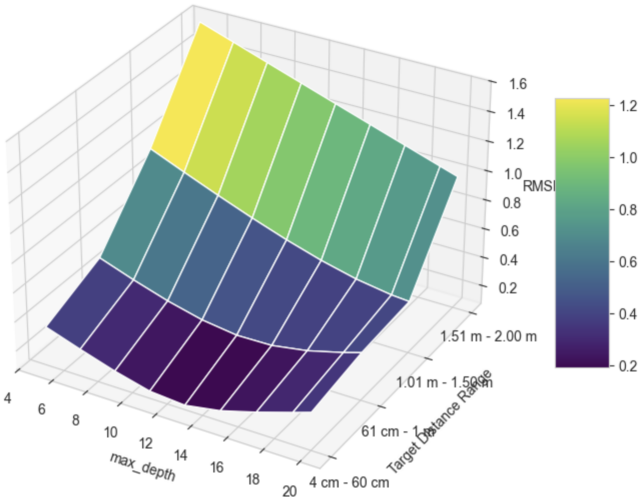


Figure 5.5 `max_depth` comparison for bottom camera images.

In light of the results of these experiments, it can be stated that 13 is the best value for the `max_depth` parameter for the bottom camera, while 20 is the best value for the `max_depth` parameter for the top camera.

5.3 Results of grasping attempts with Pepper

To evaluate the performance of the grasping system, a series of experiments were carried out with the robot. The aim of these tests was to verify the applicability of the proposed system to real and common contexts in which the robot must deal with typical problems, such as variations in light, temperature, humidity and other conditions that can affect its operation. Simulating everyday contexts is not particularly easy, also due to some limitations of the robot, such as the impossibility of deactivating the protections for external collisions. This issue prevents the robot from placing the object to be grasped on chairs or small furniture, as their legs would cause the robot's movement to stop before it could reach a certain proximity. Furthermore, the type of floor also affects its movement, since the robot is equipped with wheels and not real legs.

Taking these reasons into account, it was decided to test the robot in two specific environments. The first is a laboratory room equipped with computers, large desks and a large window that allows natural light to be used for most of the day. The second room is a windowless room, lit with artificial light.

Before testing the system, experiments were conducted to determine which type of bottle was most suitable for a robot with very small hands, such as those of Pepper. The three bottles shown in Fig. 5.1, based on their size and shape, proved to be the most appropriate for this task and for the structure of the robot's hands. These were bottles of approximately 0.25 liters, as larger half-liter bottles were too

big for Pepper's hand. In particular, the bottle also shown in Fig. 5.6 was the one most frequently used in our experiments because its shape best fits Pepper's hand.

Figure 5.6 The two environments where the grasping system was tested.

In our tests, a grasp is considered successful when the robot is able to reach and lift the bottle. Conversely, a grasp is deemed unsuccessful if the robot cannot grasp and lift the bottle. Clearly, for the bottle to be reachable, it must be placed at the edge of the table, as Pepper's arms do not allow it to reach points further inward. A total of 30 trials were conducted—15 in the first room and 15 in the second—with the robot starting from different positions, yielding the following results:

	Successful grasp	Unsuccessful grasp
Room 1	8	7
Room 2	10	5

Overall, the success rate, i.e., the percentage of successful grasps relative to the total number of trials, is 60%. However, it is interesting to note that the presence of artificial light still improves the system's performance: if it were used solely in this context, the success rate would rise to 66%. This observation is further supported by the fact that, even in Room 1, when tests are conducted in the dark with artificial light, the values increase, approaching those obtained in the second room. Nevertheless, the achieved result is highly satisfactory, especially considering that the proposed approach did not use markers or 3D models and employed technologies like Depth Anything, which had not previously been utilized in the literature for this type of task.

A demonstration video of a successful grasp can be found at:
`https://drive.google.com/file/d/1VGXv6XXMQ5bpULFn8LILfGQ4_c1tmW4y/view`

Chapter 6

Conclusions

In this work, a novel approach has been developed that allows Humanoid Robots with the characteristics of Pepper to grasp unknown objects (whose dimensional structure, shape or material are unknown) without the use of markers. This approach is innovative precisely because it is able to complete a complex task such as grasping without the need to rely on external sensors, Depth cameras or other devices that can help to complete the task. It is structured around five key modules, described as follows. The Object Detection module identifies the target object within frames captured by two 2D cameras, which operate independently rather than simultaneously, and outputs the corresponding bounding box. The Depth Estimation module collaborates with the Object Detection module to estimate the object's distance from the camera providing the frames. Specifically, it extracts a depth value at the center of the bounding box, which serves as a representative point reflecting the general trend of depth values across the object's surface. Next, the Coordinate Mapping module converts the object's coordinates from the camera's reference frame to the robot's base reference frame. These processes are orchestrated by the Path Planning module, which guides the robot toward the target object in two phases: an initial coarse approach to bring the humanoid near the object, followed by a refinement phase to adjust the robot's base position for optimal grasping. Finally, the Inverse Kinematics module executes the grasping action. An extensive experimental evaluation was conducted, and the results demonstrated that, under controlled environmental conditions, this solution is a competitive alternative within the state-of-the-art for marker-free grasping approaches for humanoid robots like Pepper. Notably, it performs effectively without relying on auxiliary devices such as depth cameras or sensors. The tests further confirmed that satisfactory performance is achievable without detailed prior knowledge of the object's three-dimensional structure, CAD models, or geometry. Instead, the approach leverages widely available models, including YOLOv10 for object detection, Depth Anything V2 for depth estimation, and Playful Kinematics for inverse kinematics calculations. Looking ahead, future developments could integrate this work with advanced robotics techniques to enhance its capabilities. Furthermore, it would be interesting to extend this work by giving the robot the ability to grasp new target objects even with a non-convex shape in which the surface is holed in the center. A further future development could consist in the introduction of a Collision Avoidance or Simultaneous localization

and mapping (SLAM) module that allows to build or update in real time the map of an unknown environment.

Acknowledgments

First of all I would like to thank Prof. Luigi Cinque who with his great kindness and professionalism has been able to make working inside the VisionLab particularly pleasant. A special thanks goes to the guys at the VisonLab, where there were many moments of fun like in the wonderful days when we played videogames together after hard days of work but with the spirit of those who never want to give up. After moments of discouragement, you can always play, and come back stronger than before. A special thanks goes to Marco who with his advice and his way of encouraging me, has pushed me to always give that little bit more. I also say thank you to Omar, who between one break and another, with his fantastic pseudo-robotics lessons, has given me a huge help with an availability that I have rarely encountered. I thank my uncles, my grandparents and my cousins with whom we always spend fantastic Christmases every year. It is not the number of times we get together each year that is important but the quality of the time we spend together between many laughs and harmony, all seasoned with a nice card game. I thank my mom and dad. Thank you mom, thank you dad, for all the sacrifices you have made for me and my brothers throughout our lives. Thank you for the love you have given us, thank you for the love I feel for you. I hope one day to be with my children at least half of what you have been for me. Thank you Matteo and Deborah for being the best siblings I could ever wish for. Thank you Matteo for the childhood we spent together. To the beautiful days spent together with Mauro and Massimo. The dolls, the spinning tops, the Play Station, the whole days playing soccer with destroyed knees, to the jumps on the ramps set up at the last moment that also here shattered our knees, to the Yu-Gi-Oh cards, to the blows and so much more. It was an honor to grow up by your side. I also thank my big sister. With you when we were little we had a relationship that led me to have legs, arms, back full of pinches, bruises and so on and so forth..AHAHAHAHAH. All this, to have the TV remote control. How much I love you. This is what I love about you, wanting to always be what you are, in the most instinctive and true way possible. Thanks to Gaia, who is now part of the family, who has been able and continues to make Matteo happy. Thanks for the industrial amount of cheese you put in the pasta AHAHAH. A big thank you is reserved for Milika and Giuseppe who accepted me into their home as if I were their son. I also thank Noemi, Dalila and Gabriele, who have been wonderful traveling companions. We have known each other for many years now, we have been through so many beautiful situations together. We all graduated with many difficulties but also just as many joys. You have been a very important block in my life. I also thank the JABA brothers. Who are much more than true friends, they are brothers who have been with me for a lifetime now. I thank Joel who when you get to know him, he captures you with his way of being. It seems like nothing touches him, from the outside he may seem almost superficial, but in reality he is much deeper than what he shows on the outside and he is a loyal friend. Brenda knows this very well, and I thank her for it, since she became part of our circle of friends, she has proven to be a precious friend. I thank Andrea, who now lives his life in Bologna, studies there, but who has been and always will be the greatest example of lifestyle I have ever known. He is not Andrea, he is the kind of person who wakes up in the morning and decides to go to Sri Lanka to volunteer.

It is a privilege to keep up with you, my friend. We certainly do not forget Balraj. It is difficult to tell you how much I love you because for us it is normal, it is like something so obvious and true, that shaking your hand to say goodbye seems reductive to me. I do not know why but I can tell you that in life, wherever you are in the world, if you look for me, I will be there. Finally, I want to thank you Ciambi, who since that August 5, 2015, when we did not know each other yet, you have changed my life. From that moment, I have learned to live life as a couple. In every form that this can take, both in the beautiful and fantastic things and in the less beautiful things. You have given me the opportunity to grow in my way of being responsible. You have done something that few in life do, you have given the opportunity to a humble human being to be able to choose for a second time. I am grateful to you for having, one day, taken a step towards my heart. I will thank you again and again. Forever.

Bibliography

[1] VisionLab, Research laboratory of the Department of Computer Science at the Sapienza University of Rome, Italy.

[2] Minglun Dong and Jian Zhang. A review of robotic grasp detection technology. *Robotica*, 2023.

[3] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2016.

[4] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 3rd edition, 2005.

[5] Richard M. Murray, Zexiang Li, and Shankar S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[6] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2nd edition, 2020.

[7] Bruno Siciliano, L. Sciavicco, Villani Luigi, and G. Oriolo. Robotics: Modelling, planning and control. *Advanced Textbooks in Control and Signal Processing*, 2009.

[8] Samuel Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE*, 2004.

[9] Lorenzo Sciavicco and Bruno Siciliano. Modelling and control of robot manipulators. *Measurement Science and Technology*, 2000.

[10] D. E. Whitney. The mathematics of coordinated control of prosthetic arms and manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 1972.

[11] Yoshihiko Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[12] Mark Elling Rosheim. *Leonardo's Lost Robots*. Springer, 2006.

[13] Mark E. Rosheim. *Robot Evolution: The Development of Anthrobotics*. John Wiley & Sons, Inc., 1994.

[14] Eiichi Yoshida. Robots that look like humans: A brief look into humanoid robotics. *Metode*, 2018.

[15] Ichiro Kato, Sadamu Ohteru, Hiroshi Kobayashi, Katsuhiko Shirai, and Akihiko Uchiyama. Information-power machine with senses and limbs. *On Theory and Practice of Robots and Manipulators: Volume I*, 1974.

[16] S.A. Setiawan, J. Yamaguchi, Sang Ho Hyon, and A. Takanishi. Physical interaction between human and a bipedal humanoid robot-realization of human-follow walking. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 1999.

[17] Md Akhtaruzzaman. Advancement of android and contribution of various countries in the research and development of the humanoid platform. *cscjournals.org*, 2010.

[18] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of honda humanoid robot. *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, 1998.

[19] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka. Development of the honda humanoid robot asimo. *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[20] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot hrp-2. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, 2004.

[21] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 2003.

[22] Yu Ogura, H. Aikawa, K. Shimomura, H. Kondo, A. Morishima, Hun ok Lim, and A. Takanishi. Development of a new humanoid robot wabian-2. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006.

[23] Johannes Englsberger, Alexander Werner, Christian Ott, Bernd Henze, Maximo A. Roa, Gianluca Garofalo, Robert Burger, Alexander Beyer, Oliver Eiberger, Korbinian Schmid, and Alin Albu-Schäeffer. Overview of the torque-controlled humanoid robot toro. *IEEE-RAS Int. Conf. Humanoid Robots*, 2014.

[24] Ill-Woo Park, Jung-Yup Kim, Jungho Lee, and Jun-Ho Oh. Mechanical design of the humanoid robot platform, hubo. *Advanced Robotics*, 2007.

[25] Giorgio Metta, Giulio Sandini, D. Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: An open platform for research in embodied cognition. *Performance Metrics for Intelligent Systems (PerMIS) Workshop*, 2008.

[26] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux, J.-P. Laumond, L. Marchionni, H. Tome, and F. Ferro. Talos: A new humanoid research platform targeted for industrial applications. *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017.

[27] T. Ishida. Development of a small biped entertainment robot qrio. *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[28] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. The nao humanoid: a combination of performance and affordability. *CoRR*, 2008.

[29] T. Kanda, H. Ishiguro, T. Ono, M. Imai, and R. Nakatsu. Development and evaluation of an interactive humanoid robot "robovie". *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, 2002.

[30] Takayuki Kanda, Takayuki Hirano, Daniel Eaton, and Hiroshi Ishiguro. Interactive robots as social partners and peer tutors for children: A field trial. *Human Computer Interaction (Special issues on human-robot interaction)*, 2004.

[31] Takayuki Kanda, Michihiro Shimada, and Satoshi Koizumi. Children learning with a social robot. *HRI'12 - Proceedings of the 7th Annual ACM/IEEE International Conference on Human-Robot Interaction*, 2012.

[32] Masahiro Shiomi, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Interactive humanoid robots for a science museum. *Intelligent Systems, IEEE*, 2007.

[33] Takayuki Kanda, Masahiro Shiomi, Zenta Miyashita, Hiroshi Ishiguro, and Norihiro Hagita. An affective guide robot in a shopping mall. *2009 4th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2009.

[34] Alessandra Sabelli and Takayuki Kanda. Robovie as a mascot: A qualitative study for long-term presence of robots in a shopping mall. *International Journal of Social Robotics*, 2016.

[35] Tamim Asfour, Karsten Berns, Rüdiger Dillmann, and Forschungszentrum Karlsruhe. The humanoid robot armar: Design and control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[36] Tamim Asfour, K. Regenstein, Pedram Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, and Rüdiger Dillmann. Armar-iii: An integrated humanoid platform for sensory-motor control. *Proceedings of the 2006 6th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*, 2007.

[37] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. *2010 IEEE International Conference on Robotics and Automation*, 2010.

[38] Jur van den Berg, Stephen Miller, Kenneth Goldberg, and Pieter Abbeel. Gravity-based robotic cloth folding. *Springer Tracts in Advanced Robotics*, 2010.

[39] Christopher Vijay R, Dakkshesh G, Santhanu K J, Rangesh Sriram B S, and Ramkumar A. Human robot interaction with nao. *2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, 2023.

[40] Fumihide Tanaka, Kyosuke Isshiki, Fumiki Takahashi, Manabu Uekusa, Rumiko Sei, and Kaname Hayashi. Pepper learns together with children: Development of an educational application. *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015.

[41] Arkadiusz Gardecki, Michal Podpora, Ryszard Beniak, and Bartlomiej Klin. The pepper humanoid robot in front desk application. *2018 Progress in Applied Electrical Engineering (PAEE)*, 2018.

[42] Deepti Mishra, Guillermo Arroyo Romero, Akshara Pande, Bhavana Nachenahalli Bhuthegowda, Dimitrios Chaskopoulos, and Bhanu Shrestha. An exploration of the pepper robot's capabilities: Unveiling its potential. *Applied Sciences*, 2024.

[43] Masaya Iwasaki, Akiko Yamazaki, Keiichi Yamazaki, Yuji Miyazaki, Tatsuyuki Kawamura, and Hideyuki Nakanishi. Perceptive recommendation robot: Enhancing receptivity of product suggestions based on customers' nonverbal cues. *Biomimetics*, 2024.

[44] Marius Misaroș, Ovidiu Petru Stan, Szilárd Enyedi, Anca Stan, Ionuț Donca, and Liviu Cristian Miclea. A method for assessing the reliability of the pepper robot in handling office documents: A case study. *Biomimetics*, 2024.

[45] Alexandra Ștefania Ghiță, Alexandru Florin Gavril, Mihai Nan, Bilal Hoteit, Imad Alex Awada, Alexandru Sorici, Irina Georgiana Mocanu, and Adina Magda Florea. The amiro social robotics framework: Deployment and evaluation on the pepper robot. *Sensors*, 2020.

[46] Philipp Mittendorfer, Eichii Yoshida, Thomas Moulard, and Gordon Cheng. A general tactile approach for grasping unknown objects with a humanoid robot. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[47] Pedro Vicente, Lorenzo Jamone, and Alexandre Bernardino. Towards markerless visual servoing of grasping tasks for humanoid robots. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[48] Giovanni Claudio, Fabien Spindler, and François Chaumette. Vision-based manipulation with the humanoid robot romeo. *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016.

[49] Wim Meeussen, Melonee Wise, Stuart Glaser, Sachin Chitta, Conor McGann, Patrick Mihelich, Eitan Marder-Eppstein, Marius Muja, Victor Eruhimov, Tully Foote, John Hsu, Radu Bogdan Rusu, Bhaskara Marthi, Gary Bradski, Kurt Konolige, Brian Gerkey, and Eric Berger. Autonomous door opening and plugging in with a personal robot. *2010 IEEE International Conference on Robotics and Automation*, 2010.

[50] Antonio Morales, Tamim Asfour, Pedram Azad, Steffen Knoop, and Rudiger Dillmann. Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[51] Martin Do, Javier Romero, Hedvig Kjellström, Pedram Azad, Tamim Asfour, Danica Kragic, and Rüdiger Dillmann. Grasp recognition and mapping on humanoid robots. *2009 9th IEEE-RAS International Conference on Humanoid Robots*, 2009.

[52] Hongbin Chen. Target positioning and grasping of nao robot based on monocular stereo vision. *Mobile Information Systems*, 2022.

[53] Yingrui Jin, Zhaoyuan Shi, Xinlong Xu, Guang Wu, Hengyi Li, and Shengjun Wen. Target localization and grasping of nao robot based on yolov8 network and monocular ranging. *Electronics*, 2023.

[54] Judith Müller, Udo Frese, Thomas Röfer, Rodolphe Gelin, and Alexandre Mazel. Graspy - object manipulation with nao. *Gearing Up and Accelerating Cross-fertilization between Academic and Industrial Robotics Research in Europe*, 2014.

[55] João Silva, Miguel Simão, Nuno Mendes, and Pedro Neto. Navigation and obstacle avoidance: a case study using pepper robot. *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, 2019.

[56] Naomarks, Aldebaran Robotics, `http://doc.aldebaran.com/2-5/naoqi/vision/allandmarkdetection.html#allandmarkdetection`.

[57] Paola Ardón, Mauro Dragone, and Mustafa Suphi Erden. Reaching and grasping of objects by humanoid robots through visual servoing. *EuroHaptics*, 2018.

[58] Zuria Bauer, Felix Escalona, Edmanuel Cruz, Miguel Cazorla, and Francisco Gomez-Donoso. Refining the fusion of pepper robot and estimated depth maps method for improved 3d perception. *IEEE Access*, 2019.

[59] Python SDK, Aldebaran Robotics, `http://doc.aldebaran.com/2-5/dev/python/index.html`.

[60] C++ SDK, Aldebaran Robotics, `http://doc.aldebaran.com/2-5/dev/cpp/index.html`.

[61] Pepper SDK for Android, Aldebaran Robotics, (`https://qisdk.softbankrobotics.com/sdk/doc/pepper-sdk/index.html`.

[62] Choregraphe, Aldebaran Robotics, `http://doc.aldebaran.com/2-4/software/choregraphe/index.html`.

[63] Comparison of Pepper's OS versions FAQ, Aldebaran Robotics, `https://drive.google.com/file/d/13JDe-LtH_Li7dNHRYeMLd5elBQI1E1gC/view?usp=drive_link`.

[64] Comparison of Pepper's OS versions datasheet, Aldebaran Robotics, `https://drive.google.com/file/d/1gfD6moNim2uNtO3JZmEJKVhnvPtKA4s3/view?usp=drive_link`.

[65] Pepper body refrenece frames, Aldebaran Robotics, `http://doc.aldebaran.com/2-5/family/pepper_technical/masses_pep.html`.

[66] Python 2.7 required to communicate with Pepper, Aldebaran Robotics, `http://doc.aldebaran.com/2-5/dev/python/install_guide.html`.

[67] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[68] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. Convolutional neural network (cnn) for image detection and recognition. *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 2018.

[69] M.S. Kamel. Neural networks: the state of the art. *ICM'99. Proceedings. Eleventh International Conference on Microelectronics (IEEE Cat. No.99EX388)*, 1999.

[70] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[71] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection. *Advances in Neural Information Processing Systems*, 2024.

[72] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[73] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[74] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. *18th International Conference on Pattern Recognition (ICPR'06)*, 2006.

[75] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv:2406.09414*, 2024.

[76] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M. Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. *International Conference on Computer Vision (ICCV) 2021*, 2021.

[77] A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual worlds as proxy for multi-object tracking analysis. *CVPR*, 2016.

[78] Yang, Lihe and Kang, Bingyi and Huang, Zilong and Xu, Xiaogang and Feng, Jiashi and Zhao, Hengshuang. Depth Anything V2 for Metric Depth Estimation, `https://github.com/DepthAnything/Depth-Anything-V2/tree/main/metric_depth`.

[79] Théo Moutakanni Huy Vo Marc Szafraniec Vasil Khalidov Pierre Fernandez Daniel Haziza Francisco Massa Alaaeldin El-Nouby et al. Oquab, Timothée Darcet. Learning robust visual features without supervision. *TMLR*, 2023.

[80] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[81] 2D Cameras, Aldebaran Robotics, `http://doc.aldebaran.com/2-5/family/pepper_technical/video_2D_pep.html`.

[82] Playful Kinematics, Vincent Berenz, Github `https://github.com/vincentberenz/playful_kinematics`.

[83] Ben Kenwright. Inverse kinematics – cyclic coordinate descent (ccd). *Journal of Graphics Tools*, 2012.

[84] Kinematics and Dynamics Library (KDL), Orocos project, `https://www.orocos.org/kdl.html`.