UNIVERSITÁ DEGLI STUDI DELLA BASILICATA

SCUOLA DI INGEGNERIA
Dottorato di Ricerca in Ingegneria dell'Innovazione e lo Sviluppo Sostenibile
Settore Scientifico Disciplinare: ING-INF/04

Ph.D. Thesis

Collaborative and Cooperative Robotics Applications using Visual Perception

Thesis Advisors
Prof. Francesco Pierri

Prof. Donato Sorgente

Ph.D. Student
Monica Sileo

Ph.D. Course Coordinator
Prof.ssa Aurelia Sole

XXXV CICLO

Codes are a puzzle. A game, just like any other game.
Alan Turing

Abstract

The objective of this Thesis is to develop novel integrated strategies for collaborative and cooperative robotic applications. Commonly, industrial robots operate in structured environments and in work-cell separated from human operators. Nowadays, collaborative robots have the capacity of sharing the workspace and collaborate with humans or other robots to perform complex tasks. These robots often operate in an unstructured environment, whereby they need of sensors and algorithms to get information about environment changes.

Advanced vision and control techniques have been analyzed to evaluate their performance and their applicability to industrial tasks. Then, some selected techniques have been applied for the first time to an industrial context. A Peg-in-Hole task has been chosen as first case study, since it has been extensively studied but still remains challenging: it requires accuracy both in the determination of the hole poses and in the robot positioning. Two solutions have been developed and tested. Experimental results have been discussed to highlight the advantages and disadvantages of each technique.

Grasping partially known objects in unstructured environments is one of the most challenging issues in robotics. It is a complex task and requires to address multiple subproblems, in order to be accomplished, including object localization and grasp pose detection. Also for this class of issues some vision techniques have been analyzed. One of these has been adapted to be used in industrial scenarios. Moreover, as a second case study, a robot-to-robot object handover task in a partially structured environment and in the absence of explicit communication between the robots has been developed and validated.

Finally, the two case studies have been integrated in two real industrial setups to demonstrate the applicability of the strategies to solving industrial problems.

Acknowledgments

First and foremost I am extremely grateful to my supervisors, Professor Francesco Pierri and Professor Donato Sorgente for their invaluable advice, continuous support, and patience during my Ph.D. Their knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life.

I would like to express my sincere gratitude to Professor Fabrizio Caccavale for his continuous support during my Ph.D. and, also, for his patience, motivation, and immense knowledge. He gave me the possibility to make many beautiful experiences.

My special thanks to Professor Domenico D. Bloisi and Professor Katia Genovese for spending time to teach me the ”secrets” of their research field.

I would also like to thank Michelangelo Nigro for his support and his patience. Thanks to my friends: Gilda, Antonia, Ivan and Pietro for listening to my complaints, and Irene, Mariantonietta, Alessia, Michele, Francesco C., Francesco S. and Marco. Even if from afar we have never stopped sharing moments of joy.

I would like to thank Luisa for sharing with me the apartment in Lund and for all the adventures during the Swedish winter. I was lucky to find her as a roommate and I was fortunate because she became a very good friend.

My special thanks to my parents, my brother Fabio, Incoronata and Erica for always supporting me and helping me during these years. I want to thank my loving boyfriend Mario for his continuous support, especially during the writing of this Thesis. He always believed in me and always encouraged me not to give up.
Thank you.

Contents

List of Figures

List of Tables

Introduction

I.1 State of the art

I.1.1 Collaborative robots

Industrial robotics is the discipline concerning robot design, control and applications in industry. Industrial robots are usually large, heavy and are installed to perform jobs that would be very difficult and dangerous for humans. They operate in a structured environment whose geometrical or physical characteristics are mostly known a priori and that is isolated from human operators.

The presence of such robots produced the continuous automation and remodeling of the industrial processes with the aim of increasing the productivity of the production processes. In 2011, the term *Industry 4.0* is used for the first time with the intent of referring to the fourth industrial revolution, a series of activities to create *Smart Factories* [1] with highly flexible and reconfigurable facilities.

In this context, collaborative solutions, where human workers and robots share the workspace and collaborate in performing complex tasks, are becoming the new frontier of the industrial robotics. Indeed, collaborative robots, also called *cobots*, have started to spread in this new era [2]. They allow the direct interaction with the human operators, thus overcoming the classical division of labour, which requires robots to be confined in safety cages far away from human workers. Moreover, they are designed to work in unstructured environments by leveraging on learning capabilities.

For a complete introduction of collaborative robots in industrial processes, it is necessary to deal with some fundamental technological challenges:

- safe interaction: safety issues are the primary challenge that must be tackled by any approach that requires collaboration between humans and robots;
- intuitive interfaces: it is important that user interfaces are designed so that human operators can easily interact with the robot;

Figure I.1: Levels of Human-Robot Interaction.

- design methods: specific design methods integrating control laws, sensory data and activity planning methods must be developed and applied.

In the Human-Robot Interaction (HRI) there are three levels of interaction [3], which are shown in Fig. I.1. A higher level requires that the features of lower interaction levels are guaranteed. In general, to achieve safety, collisions should be prevented, but if they accidentally occur, e.g., due to the limits of sensors and robot motion capabilities, the robot should be able to react. In this case, the robot could detect the physical collision and immediately remove itself from the collision area [4]. Moreover, in some cases, robots can use appropriate control laws to reduce forces at the impact [5].

The level of coexistence considers that a robot and a human operator safely share the workspace and might also work on the same object, but without any mutual contact or coordination of actions and intentions [3].

Beyond coexistence, collaboration approaches allow the robot and the human operator to perform a complex task together. There are two types of collaboration: the *physical collaboration*, where there is an intentional contact with exchange of forces between human and robot [6], and the *contactless collaboration*, in which actions are coordinated from an exchange of information, like gestures and voice commands [7], without physical interaction.

As a consequence of the introduction of human-robot collaboration technologies, great importance has been attributed to robot safety standards. Indeed, the regulation establishes four collaborative modes [8, 9], which are summarized in Fig. I.2 and described below.

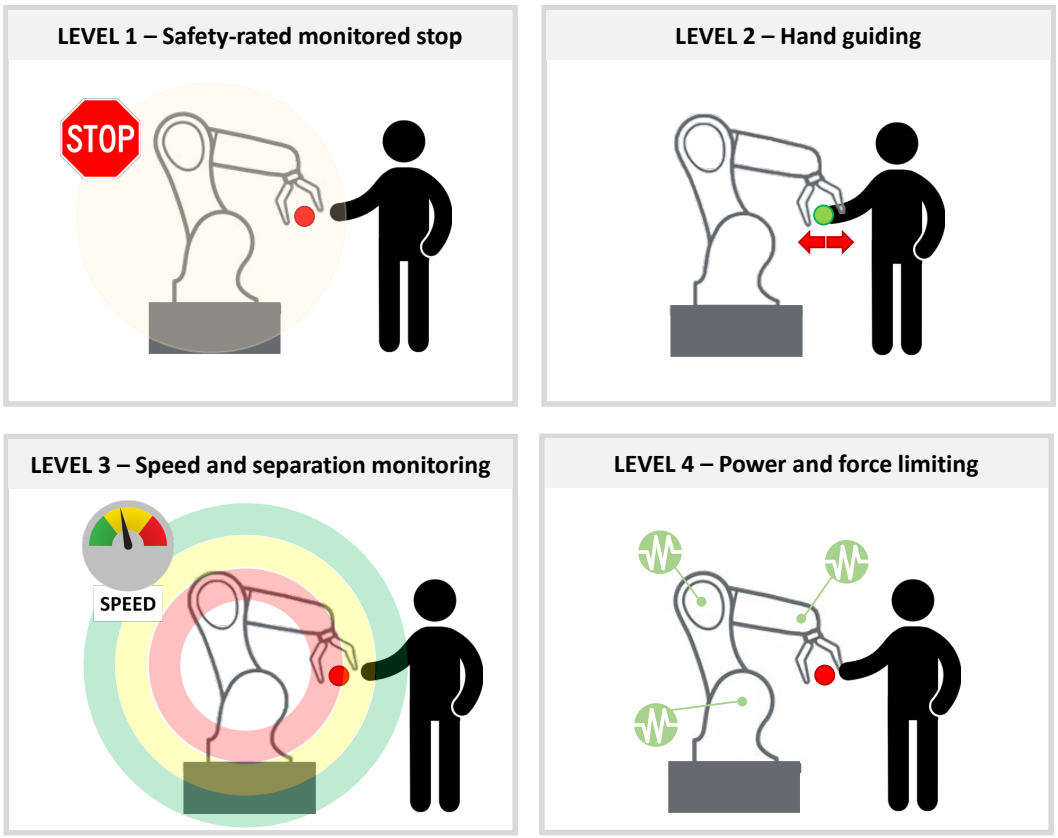


Figure I.2: The four collaborative modes.

Safety-rated Monitored Stop is the simplest type of collaboration. Inside the collaborative area, both the human and the robot can work, but not at the same time since the latter is not allowed to move if the operator occupies the shared space. The robot detects the human presence by using one or more sensors.

Hand Guiding, also known as "direct teach", in which the operator can teach the robot task by moving it without the need of an intermediate interface. The robot executes the program in automatic mode, if the operator approaches the collaborative area, the robot program and movements are interrupted.

Speed and Separation Monitoring allows the human presence within the robot's space through safety-rated monitoring sensors. With reference to Fig. I.2, the robot operates at full speed when the human is in the green zone, at reduced speed in the yellow zone, and it stops when the human moves into the red zone.

Finally, the *Power and Force Limiting* mode prescribes the limitation of motor power and force, so that a human operator can work side-by-side with the robot. This level

requires dedicated equipment and control models for handling collisions between the robot and the human with no harmful consequences for the latter.

Regarding the programming of robots, existing approaches can be traditionally classified into on-line programming and off-line programming. The most novel approaches offer great intuitiveness and ease of use, but, unfortunately, they are still quite limited in terms of possible operations to perform and working scenarios. The main goal of a natural user interface is to offer a reality-based interaction by using actions that correspond to daily practices in the physical world [10].

To achieve this goal, natural user interfaces allow users to directly manipulate and interact with robots rather than instruct them by typing commands. A classification of some particularly important programming methods [11] are briefly reported below:

- **Traditional lead-through programming.** This approach to robot programming relies on the use of a teach pendant for on-line moving the robot through the desired path. Trajectories and endpoints are, then, recorded into the controller memory to allow the execution of the learned task. Although the concept is simple and does not require strong technical expertise, some programming skills are still required and teaching trajectories to the robot in this way turns out to be a time consuming task, requiring production interruptions. For this reason, in industry, this type of robot programming is used only for large production batches. This approach is not suitable for small and medium sized factories, where small production batches require frequent reprogramming of the operations performed by the robot [12].

- **Off-line programming.** This approach uses the simulation of the task in a 3D model of the complete robot workcell. After simulation and testing, the program is loaded into the robot control unit. Programming can still take a long time, but production does not need to be interrupted during this phase. An important step in the off-line programming is the calibration of the robot and the workcell. This step is necessary to compensate for any errors due to a mismatch between the real geometry of the cell and the virtual model used for programming.

- **Walk-through programming.** The basic idea behind this method is that the user can physically move the end-effector of the robot through the desired positions (hand guiding). At the same time the robot's controller records the desired trajectory and the corresponding joints coordinates, and is then able to reproduce the trajectory thereafter. Thus, the robot can be programmed in a very intuitive manner

and no knowledge of the robot programming language is required. For industrial robot this goal is achieved by using a force/torque sensor, typically mounted on the robot wrist, to measure forces and torques during the interaction. Then, these measurements are used to control the robot [13].

- **Programming by demonstration.** In this approach the robot does not repeat the trajectory recorded in the system, but it learns the movements to perform under varying conditions (demonstration phase) and generalizes them in different scenarios. This approach allows an easy and natural interaction, without requiring any experience in robot programming. Regarding the problem of generalizing across demonstration, two approaches have been proposed to extract the relevant features of a given task: the symbolic encoding, in which a task is expressed as a sequence of symbolic primitives [14], and the trajectory encoding, where the demonstrated trajectory is directly transformed into executable robot motion [15].

As mentioned, safety is a priority in HRI because the robots often perform difficult and risky tasks, which present a potential danger to human operators. The robotic systems need to detect human behaviours, motions, environmental changes, and even human intentions, in real time in order to modify their behaviour.

Sensors can help to recognize the environment and the information of robots and humans in real time, which are vital for the safety, but also for using interaction modes that make robots assume a behaviour more similar to the humans' one. The ultimate goal is to help users to control and program a robot by means of behaviours described at a high level of abstraction [16, 17].

In particular, vision systems can be used for safety or for objects and environment recognition and to recognize human body gestures and facial expressions. For safety reasons, the vision-based monitoring systems are mainly used to implement obstacle avoidance. When this research field was born, the proposed approaches mostly used 2D images or videos. The main drawback of using 2D images is that they are sensitive to changes in illumination [18]. With the spreading of sensors that provide also the depth information, new methods have been developed.

Nevertheless, most of these approaches work in domestic scenarios and only a few contributions have dealt with the industrial scenario. Among the latter, in [19] a skeletal joint descriptor is used for action classification in industrial applications. In this scenario, multiple actions can occur simultaneously and the proposed framework is able to detect

multiple actions simultaneously in real-time.

In [20] the Microsoft Kinect sensor (depth camera) is used to implements a real-time framework for collision avoidance. In particular, the human and the robot share the workspace, the Kinect camera monitors the whole scene and provides the depth image. The proposed approach computes distances between robot and workspace obstacles directly from depth data, considering also the human operator as an obstacle to avoid. The distances between the robot and the obstacles are then used to generate repulsive commands for the robot to avoid collisions.

An approach for collision avoidance in an augmented environment, where virtual three-dimensional (3D) models of robots and real images of the human are used for monitoring the working area and avoiding collisions, is proposed in [21]. The depth vision systems proposed consists of Microsoft Kinect sensors, and it provides a point cloud of the human that is put into the 3D model of the workcell so the human-robot distance can be computed.

As mentioned, sensors can be used for recognizing the demonstrator's actions and transferring them to the robot for motion imitation. In [22], a hybrid approach combining visual and force servoing is presented, to automate the deburring process for cast aluminum wheels. In particular, a desired path is marked manually on the object. A camera mounted on the robot end-effector captures the images and identifies the center of the line. The robot is controlled to move in a zig-zag pattern along the line to identify the normal of a local surface while the tool is kept in contact with the surface. To accurately identify the center of the line, the robot is controlled such that the tool is perpendicular to the surface. While the robot moves along the marked path, the position and orientation are recorded to later generate a trajectory to perform the deburring operation.

Regarding the collaborative robots, the Human-Robot Collaboration (HRC) is an interaction, where humans and robots work together without barriers. In this context, also the human needs to know some information about the robot, for example, its state. The ways in which the human can interact with the collaborative robots are influenced by the spread of new technological devices. In [23] the Leap motion and an electromyography sensor (Myo armband) are used to control the movement of a mobile robot.

Also, Mixed Reality can give human operators without particular experience or knowledge of robotics the possibility to easily interact with the robot. An application in which the human operator interacts with a collaborative robot Franka Emika Panda [24] by using the Microsoft HoloLens 2 has been developed in [25]. In this work the state of the

robot and the trajectory that it is going to follow are shown to the worker through the mixed reality device.

In [26] a scalable multi-agent human-robot teaming system for indoor and outdoor exploration is proposed. The system allows multiple users to simultaneously localize, supervise, and receive labeled images from robotic clients by using the Microsoft HoloLens 2 device. In other words, robots are supported in sending text and image alerts about discovered items of interest to human teammates.

Mixed Reality isn't used only to visualize information about the state or messages from the robots. It can also be used to interact with them: [27] proposes the use of mixed reality devices to extend hand-guidance to robots lacking joint-torque sensors. In particular, the in-build hand tracking capabilities of the Microsoft Hololens are used to calculate the position of the hands relative to the robot. By decomposing the hand movements, a completely sensorless hand-guidance is achieved, without any need to build a dynamic model of the robot.

In [28] these two aspects are combined to realize an interface that allows human operators to interact with the robotic system by sending commands and receive instant feedbacks through the Microsoft Hololens in collaborative tasks. The main goal is to help workers to interact with the robot in the most user-friendly way possible: with visual feedbacks, such as a light that indicates the current actions of the robot, or sound feedbacks, to understand what is happening when the robot is out of the human's field of view or in case of interaction with multiple robots. The proposed interface provides different types of user inputs: the gaze, e.g, to give reference for the trajectory, and vocal commands, which guarantee high easy handling. The latter cannot be used in noisy industrial environment, thus also the gesture commands are included in the interface, by exploiting the gesture recognition capabilities of the Microsoft Hololens.

As in HRC humans and robots collaborate with physical interaction between them, Human Factors and Ergonomics [29] can contribute to the success of the collaborative task. The safety in the strict sense, to prevent collisions between robots and operators, has already been mentioned, but the humans' perceptual safety during the task must be considered. Perceived safety is subjective to the individual human worker and is dependent on several factors, like mental stress or anxiety, and this can increase the probability of errors. Moreover, in the same situation each operator reacts differently. Therefore, supporting human operators in collaborative tasks should be a priority in order to minimize the probability of errors.

Thus, it is needed to investigate the causes of stress and error to know how to support correctly human operators, through developing supporting systems and assisting them to increase their perceptual safety. In [30] a preliminary experiment on HRC with the intention of investigating the factors that lead to errors is presented. The study found that the stress of not knowing the position of the robot and the mental ability needed to make the decision on how to proceed during the process are the two main reasons influencing the operators' performance.

I.1.2 Cooperative robots

In many applications, it is useful to consider both the interaction with the human operator (human-robot interaction) and the interaction between several robots (robot-robot interaction). Indeed, many tasks that are difficult or impossible to be performed by a single robot become feasible when two or more manipulators are used in a cooperative manner [31]. Examples of cooperative applications can be the manipulation or transportation of objects with considerable size and/or weight [32].

There are two types of cooperation between robots [33]:

- passive cooperation, in which the robots don't communicate with each other, and the cooperation becomes evident when the whole system is observed. In this type of cooperation, the robots consider the others as obstacles, their trajectories are planned locally and are not shared;
- active cooperation, where the robots can actively coordinate their decisions and actions. It doesn't have to be a direct communication, they can communicate also via the environment. A particular type of active cooperation is the *tight cooperation*, in which the robots need to coordinate their action precisely, for example, cooperative transportation [34] and surgery [35].

In the literature, the first contributions date back to the 1970s [36, 37] and are mainly based on the attempt to confer a suitable mechanical compliance to the manipulators. Also, load distribution among the arms is an important issue, since load sharing may be exploited both for optimal load distribution among arms and for robust holding of the manipulated object. When a cooperative multi-arm system is employed for the manipulation of a common object, it is important to control both the absolute motion of the held object and the stresses applied to it.

When a robot performs a cooperative task, it is necessary that it is able to handle the physical contact with the environment to complete the operations. The term "environment" is generic, it can be referred to the objects in the workcell or to other robots. The dynamic interaction between the robot and the environment that can be inertial (as in pushing a block), dissipative (as in sliding on a surface with friction) or elastic (as in pushing against an elastically compliant wall). In all these cases, a pure motion control strategy, e.g. controlling the position or the velocity of the end-effector, cannot be used when an interaction is involved. Indeed, successful execution of an interaction task with the environment by using pure motion control could be obtained only if an accurate model of both the robot manipulator (kinematics and dynamics) and the environment (geometry and mechanical features) is available. A manipulator model may be known with sufficient accuracy, but a detailed description of the environment is difficult to obtain.

The control that ensures a compliant behaviour during the interaction can be achieved either in a passive or in an active mode [38]. In the passive control the trajectory of the end-effector is modified by the interaction forces due to the intrinsic compliance of the robot, e.g. soft robot arms with elastic joints or links are purposely designed for intrinsically safe interaction with humans [39]. In industrial applications, a Remote Center of Compliance (RCC) [40] device is extremely used. An RCC is a compliant end-effector mounted on a rigid robot, designed and optimized for Peg-in-Hole assembly tasks.

The passive interaction control does not require force/torque sensors; also, the response of a passive compliance mechanism is much faster than active repositioning by a computer control algorithm. On the other hand, the passive compliance is not flexible, and it can only deal with small deviations of the programmed trajectory. Moreover, since forces aren't measured, the passive control cannot guarantee that high contact forces will never occur.

In active interaction control, the compliance of the robotic system is ensured by a purposely designed control system, i.e, the measurement of the contact forces and momenta are fed back to the controller and used to modify or generate online the desired trajectory of the end-effector. Active interaction overcomes the disadvantages of the passive ones, but it is usually slower, more expensive and sophisticated.

Active interaction controls strategies can perform two types of force control:

- indirect, where the force control is achieved via motion control, i.e. without explicit closure of a force feedback loop;

- direct, where the contact force and moment are controlled through a force feedback loop, in order to make them converge to a desired value.

The *impedance* (or *admittance*) *control* [41, 42] is a type of indirect force control, where the deviation of the end-effector motion from the desired trajectory is related to the contact force through a mechanical impedance/admittance. A robot manipulator under impedance (or admittance) control is described by an equivalent mass-spring-damper system with adjustable parameters.

This relationship is an impedance if the robot control reacts to the motion deviation by generating forces, while it corresponds to an admittance if the robot control reacts to interaction forces by imposing a deviation from the desired motion. In the literature, the two terms are often used to refer to the same control scheme.

Admittance control is used to perform compliant motions, in tasks such as excavation and peg-in-hole [43] and, more recently, it has been used in physical HRI [44]. In this context, recent works propose variable admittance structures [45] or even nonlinear admittance filters based on adaptive Dynamic Movement Primitives [46].

I.2 Contribution

In this Thesis, novel integrated strategies for collaborative and cooperative robotics applications are proposed, where the robots interact with the environment and/or with human by using the information provided by its proprioceptive and exteroceptive sensors.

The use of a camera, mounted on the robots, allowed to use different techniques to make the robots more autonomous also in unstructured environment, where the workspace can change and/or is shared with humans and other robots. The use of force/torque sensors allowed to make the robots directly interact with other robots or humans.

The main contributions of the Thesis can be thus summarized as follows:

- various sophisticated visual processing techniques are analyzed to evaluate their performance and their applicability to industrial contexts;
- the visual techniques with the best results are, for the first time, applied to solve a class of manufacturing tasks by adopting low-cost sensors;
- control techniques are integrated with the visual ones to achieve an autonomous execution for industrial tasks;

- the integration of this techniques is used to allow the collaboration and the cooperation between the robots and/or the humans.

Some material and code produced during this Thesis' work are available at `https://sites.google.com/unibas.it/phd-thesis-ms2022`.

I.3 Outline

The Thesis is organized as follows:

Chapter 1 reports the description of advanced vision and control techniques exploited in the execution of complex tasks in partially structured environments. In particular, a Peg-in-Hole task is chosen as case study.

Chapter 2 reports a solution for the execution of a Peg-In-Hole assembly task by means of a collaborative robotic arm, in the presence of large uncertainties on the relative pose of the workpiece with respect to the robot's base. The whole strategy to address this problem is discussed in detail.

Chapter 3 concerns the problem of object grasping and includes some visual techniques to handle it. Moreover, a complete solution for a robot-to-robot object handover application is presented. In this application, the robots do not communicate directly, but use their sensors to understand what is the next step to take. Also in this case, the problem of uncertainties on the object positioning is addressed using advanced visual techniques.

Chapter 1

Advanced vision and control techniques for robotic applications

In this Chapter, the focus is on the execution of complex tasks in partially structured environments, by exploiting the availability of affordable and reliable sensing devices. The goal is to achieve semi-autonomous task execution by resorting to visual sensing for class of assembly tasks involving insertion of mechanical parts. In particular, a Peg-In-Hole operation is chosen as case study.

1.1 The Peg-in-Hole task

The Peg-in-Hole tasks have been extensively studied but remain still challenging: they require accuracy in the determination of the holes pose and also in the robot positioning. The execution of the task can be split in two parts:

- the search phase, in which the localization of the hole and the alignment of the peg to the hole axis are performed;
- the insertion phase, where the peg is inserted into the hole.

In case of perfect knowledge of the hole's position with respect to the robot, the insertion can be performed by using a pure positional control, but this can rarely happen and this approach can be applied only in the presence of generous clearances. Among interaction control schemes, the impedance control paradigm [41] has been extensively adopted, e.g., in [47], [48], [49], for a single-arm robots, or in [50], [51] for multi-arm systems.

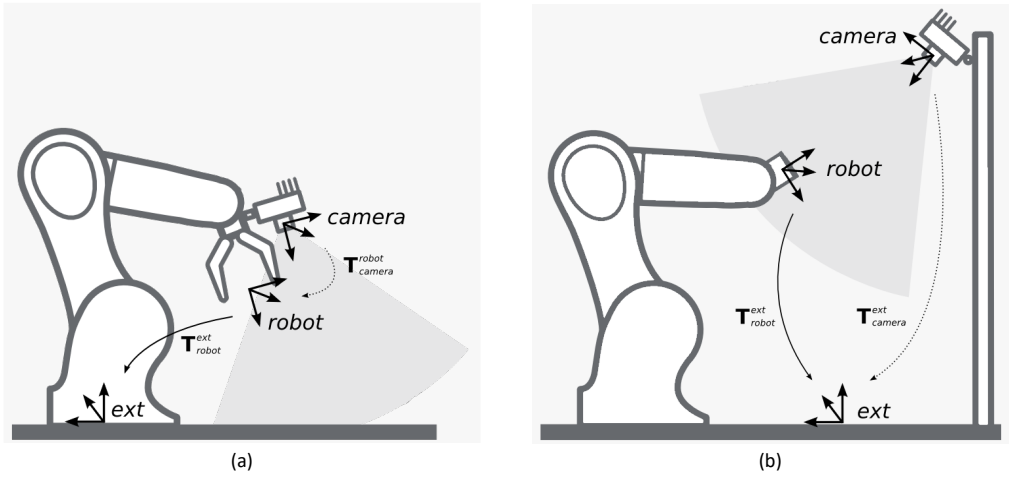


Figure 1.1: Camera in eye-in-hand configuration (a). Camera in eye-to-hand configuration (b).

Regarding the search phase, the approaches in literature can be roughly classified into approaches based on visual sensor feedback and those base on exploration of the hole neighborhood. The first category can lead to satisfactory performance, for example, in [52] a high-speed camera is adopted to align the peg to the hole, or in in [53], where a micro-peg-in-hole tasks is considered, while [54] and [55] propose the adoption of visual coaxial systems.

It is important to notice the visual techniques have some drawbacks: in particular, they are affected by the light conditions of the environment and the texture or reflection of the objects. Also calibration errors and field of view occlusions are problems, especially when an eye-in-hand configuration is adopted.

In the *eye-in-hand* configuration the camera is rigidly mounted on the robot end-effector. This configuration differs from the so-called *eye-to-hand* configuration, where the camera observes the robot within its work space. A scheme of both configurations is shown in Fig. 1.1. A camera in eye-in-hand configuration has a limited, but more precise, view of the scene, whilst a camera in eye-to-hand configuration has a less detailed, but global, sight of the scene.

In other methods, force/torque sensors are used to explore the neighborhood of the hole, as in [48], where a shape recognition algorithm is adopted to extract the outline of the peg from the force/torque sensor data collected during the contact with the object surface. After the recognition of the peg's outline, the hole detection algorithm find the direction of a hole for the insertion phase. In [56], [57] and [58] the common approach used with force/torque sensors can be found: the mapping of the interaction moments

onto the position and tilt of the hole, in order to guide the peg pose during the insertion. Anyway, the presence of this type of sensors mounted at the end-effector of the robot manipulator increases the overall cost and complexity of the system, and, for this reason, the estimation of the contact forces by using information from joint position sensors has started to be adopted [47], [59].

Search methods are often adopted, consisting in tilting and covering the hole neighborhood by moving the peg along an assigned path. In [60], dealing with the problem of inserting a charging plug into its socket, the search path is a Lissajous curve, while the wrist-mounted force/torque sensor is replaced by joint torque sensors, often mounted on collaborative robots. However, it is important to notice that blind search methods are usually time-consuming and require an accurate initial estimate of the hole position.

In order to overcome these drawbacks, visual and force/torque data can be combined, as in [61], where the peg is moved close to the hole and fine alignment is achieved via spiral search. Deep learning has also been adopted in [61]: a deep neural network trained on synthetic data is used to predict the quadrant of the hole in the images, with the aim of moving the peg towards it and then perform the insertion.

In [62], a deep learning approach based on self supervised multi-modal representation of sensory output is proposed, and in [63] a multi-layer perceptron network is trained on a data set, including object position and interaction forces, for a polyhedral pegs in contact with the holes. In [64] a deep neural network, trained via reinforcement learning, is used to find holes with variable shape and surface finish in a concrete wall. The proposed method consists of moving the peg toward the wall and try to insert it. If the hole is not found, the peg is detached from the surface and moved to the next position provided by the neural network. In addition to force and moment, displacement is also used as input of the neural network. Instead of a supervised learning technique, the reinforcement learning was used for training in order to avoid the heavy and difficult dataset labeling process.

In this Chapter, some techniques for the autonomous execution of a Peg-in-Hole assembly task by means of a collaborative robotic arm are investigated. In particular, in the analyzed case study, the holes are placed on the surface of a target object that is roughly positioned in the robot workspace by a human operator. Due to manual positioning, large uncertainties on the relative pose of the holes with respect to the robot's base are present. These uncertainties on both the holes' position and their tilt are far larger than the task tolerance.

Both vision and force/torque sensors are exploit to tackle the described problem and a high-level strategy including four main steps is proposed:

- the data are acquired from the vision sensors mounted on the end-effector of the robot;
- a reconstruction of the object's surface is performed in order to estimate the relative position of the holes with respect to the robot's base;
- the robot moves the end-effector in the neighborhood of the hole;
- the robot inserts the peg into the hole.

In the following, the techniques used to implement the above strategy are detailed, and their advantages and disadvantages are analyzed. In particular, two strategies have been considered: the first is very accurate, but it has the main disadvantage of being time-consuming, while the second one is faster but less accurate. A description of the adopted methods and how they are combined to address the Peg-in-Hole task are presented in the remainder of the Chapter.

1.2 Digital Image Correlation

In Peg-in-Hole applications in partially structured environment, one of the main problems is the detection of the surface position, as well as the holes position, with respect to the robot. In some cases, a CAD model, which can only give information about the shape of the object, is available; in some other cases the object can be completely unknown. Therefore, in both circumstances, the application of techniques to get more information about the surface is necessary. In this Section, a method for a very high accurate object measurement is detailed. This technique, named Digital Image Correlation (DIC) [65], is a non-contact full-field technique for object shape, displacement and deformation measurement.

1.2.1 2D digital image correlation

The *2D digital image correlation* (2D-DIC) was the first version of the algorithm and it used images from one camera placed with its optical axis perpendicular to the flat surface of the object to obtain full-field in-plane displacement measurement.

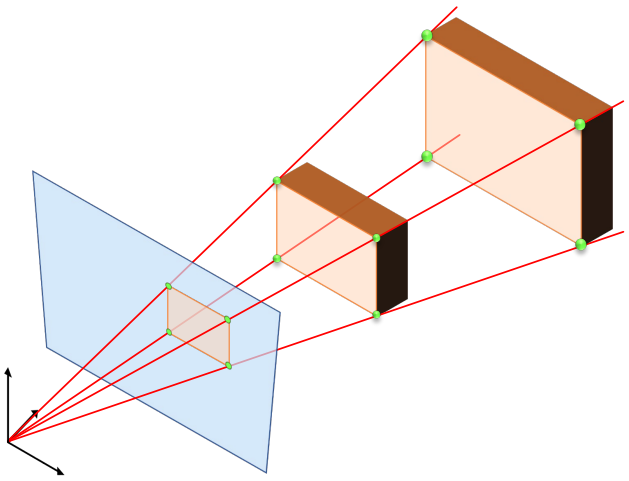


Figure 1.2: Example of transformation (translation and scale) that produces the same projection onto the image plane.

A monocular vision is not sufficient to determine the out-of-plane information and therefore cannot be applied to 3D objects. Indeed, as shown in Fig. 1.2, points placed at different distances from the camera can produce the same projection on the image plane. Thus, in 2D applications, it is assumed that objects are planar, parallel to the vision sensor and at a constant distance from it during the entire process.

The DIC technique is based on the recognition, identification and tracking of corresponding points (or features) into the reference image and in a series of images acquired afterwards, in order to compute the value of the relative displacements of the single points through motion or deformation. The use of the subdivision of the image into pixels is not enough to ensure a unique correspondence between the points in different images. For example, in a gray-scale image each pixel can take a value from 0 to 255, based on the level of diffused light intensity. The same gray value can be found several times within the series of acquired images. Furthermore, this gray level may have variations in the other images, e.g. as a result of the acquisition process.

Other two cases in which a unique correspondence between points in two images cannot be established are the case of a repeating structure, e.g., a grid of small dots and the case of textureless deformable object (Fig. 1.3). In the first case, only when the whole grid is considered, the correspondence becomes unique and a result can be obtained. For a textureless structure undergoing deformation, any motion information can be obtained by considering a region inside the boundaries, since no features are present.

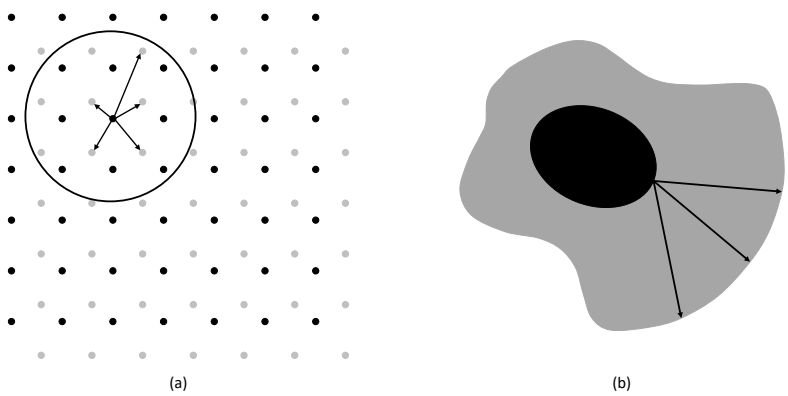


Figure 1.3: Correspondence problem for a grid structure, where the unique correspondence can only be found if the whole grid is considered (a) and a textureless deformable object, where no correspondence can be found without considering the boundary. The Figure is taken from [65].

To overcome this problem, first, a grid of control points on the image, that corresponds to control points on the objects surface, is set. For each control point, the gray scale distribution over a collection of neighboring pixel values, named *subset* (Fig. 1.4), is used to perform the image correlation.

In each subset, the distribution of the gray level of the pixels that compose it is interpolated, in order to have a continuous trend of the gray levels. Thus, it will be characterized by a light intensity I and its distribution, thus, the subset will contain more in-

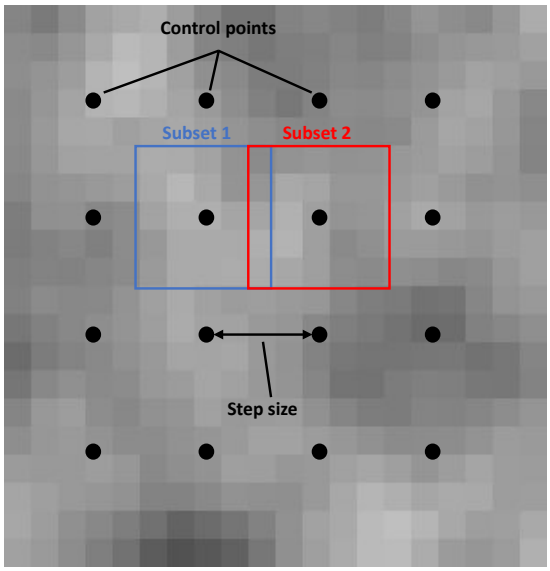


Figure 1.4: Examples of control points, subset and indication of the step size.

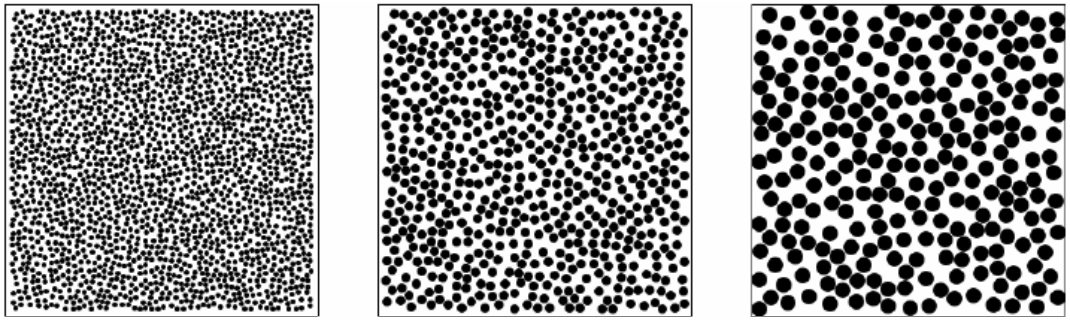


Figure 1.5: Examples of patterns with different speckle size.

formation and can be identified more precisely. The distance between the control points (step size) and subset size are two of the main process parameters, and they have to be set properly.

Unfortunately, the subsets can also generate indeterminacy in the correspondence, if they are repeated in the image (Fig. 1.3). The solution to this problem is represented by using of a stochastic pattern, called *speckle pattern*, that provides subsets a characteristic of uniqueness. Examples of speckle patterns are shown in Fig. 1.5.

The speckle size affects the accuracy of the measurement because a small speckle allows to choose a subset with smaller size and this increases the spatial resolution of the algorithm. A speckle size larger than the subset size could cause the possibility that a subset doesn't contain enough speckle to make the measurement and thus it could yield to erroneous results.

In order to provide unique information and reduce noise and uncertainty in the measurement, a pattern should be non-repetitive, to make each area of the sample surface uniquely identifiable, isotropic, so as not to favor one direction over another, and in high contrast, to allow the image correlation algorithm to work effectively. White on black or black on white speckle patterns respect the specification and in fact, they are the mostly used speckle patterns. The two colors have to be uniformly distributed in the image, in order to avoid regions that cannot be recognized correctly.

After the image acquisition phase, the DIC algorithm compares the initial reference frame and the frames acquired afterwards, to determine the range of displacements and/or deformations. The algorithm considers a reference marker in the initial frame and traces its position in the other images, by searching a subset in the subsequent image that has the same gray distribution of the reference image.

To better understand how a subset is tracked, consider the following simple example.

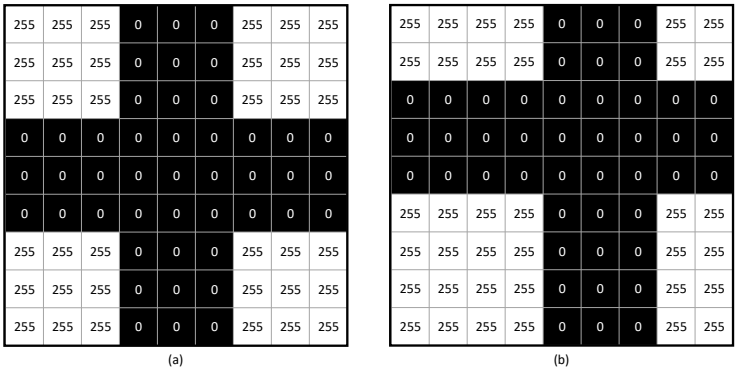


Figure 1.6: Initial gray-scale image (a) and shifted gray-scale image (b) with corresponding pixels values.

A 9×9 pixel gray-scale image and the grid with the relative pixels values in range from 0 to 255 are shown in Fig. 1.6(a). The same image, shifted one pixel up and one to the right and the corresponding pixels values, are displayed in Fig. 1.6(b). A 5×5 pixel subset is assigned on the reference image in order to track where it moved to in the deformed image.

The possible matches are checked by using a classic correlation function, i.e., the sum of squared differences (SSD, eq. (1.1)) of the pixel values, representing the similarity between the two subset. In particular, to smaller values of the function correspond a better similarity. The correlation function is the following:

$$C(x, y, u, v) = \sum_{i,j\,=\,-\frac{n}{2}}^{\frac{n}{2}} (I(x+i, y+j) - I^*(x+u+i, y+v+j))^2 \tag{1.1}$$

where x and y are the pixel coordinates in the reference image, u and v represent the displacements, n is the subset size, I and I^* are the images before and after the shift, respectively.

It is assumed that the subset shifts 5 pixels to the left and 5 pixels down, like in Fig. 1.7(a). The correlation function gives as result a very high error (around 1 million) and this indicates that the match is not found. Then, the algorithm in charge of finding the correlation moves the subset, and shifts it, e.g., 1 pixels up and 1 pixels to the right, as it is shown in Fig. 1.7(b). In this case the error is equal to 0, and the match is found.

In examples closer to reality, images are corrupted by some noise, so a minimization of the correlation function is performed to find the corresponding subsets.

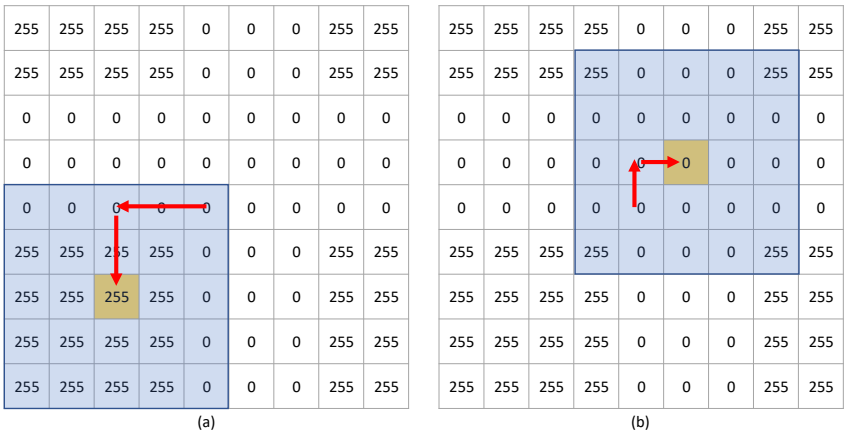


Figure 1.7: First attempt to find the match: the subset (light blue window) is shifted of 5 pixels to the left and 5 pixels down (a). Second attempt to find the match: the subset is shifted of 1 pixels up and 1 pixels to the right. In this case the match is found (b).

1.2.2 3D digital image correlation

The 3D Digital Image Correlation (3D-DIC) was introduced to overcome the limit of 2D-DIC. In fact, the 3D technique can also be applied to non-flat objects that undergo displacements even outside the plane.

This technique gives information regarding both the shape of the object's surface, obtained by comparing the images taken at the same instant from two angled views and the range of displacements and deformations, obtained by comparing the reference image with the images acquired afterwards.

To get 3D information with two sensors, the stereo triangulation technique is used. This method uses known information, like the locations of the sensors, to compute the intersections of optical rays, in order to locate features in the three-dimensional space. To this aim, the optical rays needs to be expressed in a common coordinate system and, to do this, it is necessary to build the vision system calibration model.

The calibration model includes both extrinsic geometric parameters, such as the angles and distances between the camera sensors, and intrinsic parameters specific to each camera lens setup, like the focal length, pixel dimensions, distortions, the principal point (where the optical axis intersects with the image plane) and the skew between the axes of the image plane. The calibration model is built by acquiring a series of images of a calibration target, in arbitrary positions. A calibration target is a panel, with a predefined pattern, of which the calibration software knows exactly the dimensions, the color tone and the surface roughness. Different types of calibration targets are shown in Fig. 1.8. A

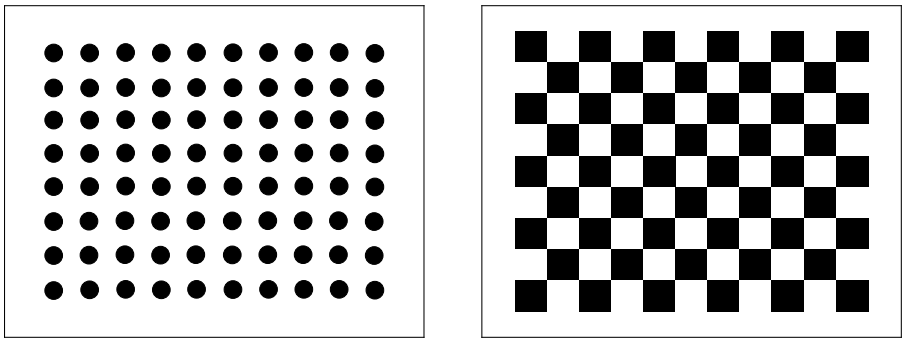


Figure 1.8: Examples of calibration targets.

good calibration allows to partially correct the distortion introduced by the lenses and this is a proof of the importance and delicacy that this phase of the measurement process. At the end of the calibration process, it is possible to know the position of the point P in 3D space, from its pair of image points in the two stereoscopic images acquired.

After the calibration process, it is possible to express the optical rays in a common coordinate frame and, then, the stereo triangulation can be applied to recover the 3D structure of the object using two image sensors.

To summarize, in 3D-DIC, the subsets in the right and left image are matched through correlation, next, they are tracked throughout time to see their deformations and then a 3D reconstruction is built by using the stereo triangulation after camera calibration.

The 3D-DIC technique is affected by two main types of measurement errors:

- Correlation errors, that express the uncertainties that occur in the recognition and tracking of the subsets between the various frames. They are related to the accuracy of the instrumentation and software.

- Calibration errors, that lead to errors in the 3D reconstruction. They occur systematically according to the position of the subsets in the frames and are mainly due to the inaccurate correction of the distortion introduced by the lenses.

Since in one of the presented Peg-in-Hole strategies a blind insertion is performed, it is necessary to get a very accurate reconstruction, hence the 3D-DIC has been used to generate the reconstruction of the whole surface with a very low error.

1.3 Iterative Closest Point

In the other Peg-in-Hole strategy presented in this Thesis, the insertion doesn't happen to be blind and, for this reason, another technique to generate the object reconstruction is exploited. This technique is faster than 3D-DIC, even if it is less accurate. The Iterative Closest Point (ICP) [66] is an algorithm for geometric registration. In this section the the point-to-point ICP algorithm [67] is described, and the approach in [68] is detailed.

The inputs of the algorithm are two point clouds $\mathcal{S}$ and $\mathcal{Q}$, obtained by the same surface in two different views. They are in registration if, for any pair of corresponding points $(s_i \in \mathcal{S}, q_j \in \mathcal{Q})$ in the two clouds representing the same point on the surface, there exists a unique homogeneous transformation matrix $\boldsymbol{T} \in \mathbb{R}^{4\times4}$ such that

$$\forall s_i \in \mathcal{S}, \exists q_j \in \mathcal{Q} \;\mid\; \|\boldsymbol{T}\tilde{s}_i - \tilde{q}_j\| = 0, \tag{1.2}$$

where the symbol $\tilde{}$ denotes the homogeneous representation of the coordinate vectors [69].

For reconstructing a surface, a set of n point cloud needs to be acquired in n different measurement point. Every point cloud is composed by N_i points, $\mathcal{P}_i = \left\{\boldsymbol{\pi}^i_{i,l}\right\}_{l=1,\dots,N_i}$, expressed in the camera coordinate frame, $\mathcal{F}^i_c$. The superscript i corresponds to the measurement point in which the point cloud is acquired. A set of homogeneous transformation matrices, $\boldsymbol{T}_i$ $(i = 1,\dots,n)$, needs to be determined in order to align the point clouds in a global coordinate frame.

Let assume that the global coordinate frame is the camera coordinate frame, $\mathcal{F}^1_c$, in which the first point cloud $\mathcal{P}_1$ is acquired. Thus, the transformation matrix $\boldsymbol{T}_1$ is the identity matrix $\boldsymbol{I}_{4\times4}$, while each $\boldsymbol{T}_i$ $(i = 2,\dots,n)$ is in charge of aligning the point cloud $\mathcal{P}_i$ to $\mathcal{P}_1$.

In order to compute the matrices $\boldsymbol{T}_i$ $(i = 2,\dots,n)$, the procedure described in [68] can be adopted. In particular, for each $\mathcal{P}_i$, the transformation matrix $\boldsymbol{T}^i_j$ $(j = i+1,\dots,n)$ that aligns $\mathcal{P}_j$ to $\mathcal{P}_i$ is computed. To this aim, first, the point clouds are down-sampled by using a volume element (voxel) grid filter [70], and then the ICP algorithm is adopted, where the following cost function has to be minimized with respect to $\boldsymbol{T}^i_j$.

$$\mathcal{C}(\boldsymbol{T}^i_j) = \sum_{\boldsymbol{\pi}_{j,l}\in\mathcal{P}_j,\, \boldsymbol{\pi}_{i,l}\in\mathcal{P}_i} \|\boldsymbol{T}^i_j\tilde{\boldsymbol{\pi}}_{j,l} - \tilde{\boldsymbol{\pi}}_{i,l}\|^2. \tag{1.3}$$

Since each $\boldsymbol{T}^i_j$ has 12 unknown components, by considering at least 4 pair of corresponding points it is possible to find the matrix $\boldsymbol{T}^i_j$ that minimizes the cost function (1.3) by resorting to a least-squares estimation.

In the *point-to-plane* ICP, the surface, represented by the point cloud $\mathcal{P}_i$, is locally approximated with its tangent plane and the cost function (1.3) becomes

$$\mathcal{C}(\boldsymbol{T}_j^i) = \sum_{\boldsymbol{\pi}_{j,l} \in \mathcal{P}_j,\, \boldsymbol{\pi}_{i,l} \in \mathcal{P}_i} \left((\boldsymbol{T}_j^i \tilde{\boldsymbol{\pi}}_{j,l} - \tilde{\boldsymbol{\pi}}_{i,l})^{\mathrm{T}}\, \tilde{\boldsymbol{n}}_{j,l}^i \right)^2, \tag{1.4}$$

where $\tilde{\boldsymbol{n}}_{j,l}^i = \boldsymbol{T}_j^i \tilde{\boldsymbol{n}}_{j,l}$ is the homogeneous representation of the unit vector normal to the surface represented by the point cloud $\mathcal{P}_j$ in the point $\boldsymbol{\pi}_{j,l}$, expressed in the reference frame of $\mathcal{P}_i$.

The ICP algorithm, for estimating the matrix $\boldsymbol{T}_j^i$, consists of the following steps:

1. A set of $m \geq 4$ control points, with corresponding normals $\boldsymbol{n}_{j,u}$ $(u = 1, \ldots, m)$, is selected in the point cloud $\mathcal{P}_j$. Such points can be chosen on a regular grid.

2. $\boldsymbol{T}_j^i$ is initialized to the identity matrix, i.e., $\boldsymbol{T}_j^i(0) = \boldsymbol{I}_{4\times 4}$.

At k-th iteration, the following steps are executed:

3. For each control point, $\boldsymbol{\pi}_{j,u}$, the corresponding point in $\mathcal{P}_i$ is determined as follows:

   - the homogeneous transformation matrix $\boldsymbol{T}_j^i(k-1)$ is applied to the control point $\boldsymbol{\pi}_{j,u}$ and its normal, $\boldsymbol{n}_{j,u}$, to obtain $\tilde{\boldsymbol{\pi}}_{j,u}^i(k-1) = \boldsymbol{T}_j^i(k-1)\tilde{\boldsymbol{\pi}}_{j,u}$ and $\tilde{\boldsymbol{n}}_{j,u}^i(k-1) = \boldsymbol{T}_j^i(k-1)\tilde{\boldsymbol{n}}_{j,u}$.
   - the intersection $\boldsymbol{\pi}_{i,u}$ of the surface defined by the point cloud $\mathcal{P}_i$ with the line defined by $\boldsymbol{\pi}_{j,u}^i(k-1)$ and $\boldsymbol{n}_{j,u}^i(k-1)$ is computed.

4. The transformation matrix, $\bar{\boldsymbol{T}}$, that minimizes the cost function

$$\mathcal{C}_m(\bar{\boldsymbol{T}}, \boldsymbol{T}_j^i(k-1)) = \sum_{u=1}^{m} \left((\bar{\boldsymbol{T}} \tilde{\boldsymbol{\pi}}_{j,u}^i(k-1) - \tilde{\boldsymbol{\pi}}_{i,u})^{\mathrm{T}}\, \tilde{\boldsymbol{n}}_{j,u}^i(k-1) \right)^2, \tag{1.5}$$

   is computed.

5. The transformation matrix is then updated as $\boldsymbol{T}_j^i(k) = \bar{\boldsymbol{T}} \boldsymbol{T}_j^i(k-1)$.

The iterative procedure is stopped when the convergence error is below a tolerance threshold $\bar{e}$, i.e.,

$$e = \frac{\mathcal{C}_m(\boldsymbol{I}_{4\times 4}, \boldsymbol{T}_j^i(k)) - \mathcal{C}_m(\boldsymbol{I}_{4\times 4}, \boldsymbol{T}_j^i(k-1))}{m^*} \leq \bar{e}, \tag{1.6}$$

where $m^* \leq m$ is the number of control points for which a correspondence is found.

In [68], it has been shown that most of the identified transformation matrices $\boldsymbol{T}_j^i$ are false positives that lead to misalignments. Thus, a further global optimization must be carried out. To the aim, a new cost function is defined

$$\mathcal{E}(\boldsymbol{T}_1, \dots, \boldsymbol{T}_n) = \sum_{\substack{i,j=1 \\ i \neq j}}^{n} \sum_{u=1}^{m} (\boldsymbol{T}_i \tilde{\boldsymbol{\pi}}_{i,u} - \boldsymbol{T}_j \tilde{\boldsymbol{\pi}}_{j,u}) \,, \tag{1.7}$$

where $\boldsymbol{T}_i = \boldsymbol{T}_1 \boldsymbol{T}_2^1 \cdots \boldsymbol{T}_i^{i-1}$ that needs to be minimized with respect to $\boldsymbol{T}_1, \dots, \boldsymbol{T}_n$.

Once the n point clouds are aligned, it is possible to merge them in a single point cloud to have the reconstructed surface, i.e.

$$\mathcal{P}_r = \bigcup_{i=1}^{n} \bigcup_{j=1}^{N_i} \boldsymbol{T}_i \tilde{\boldsymbol{\pi}}_{i,j}. \tag{1.8}$$

1.4 Neural Networks

Neural Networks (NNs) are computing systems which are heavily inspired by the biological nervous systems and the way they operate. NNs are composed by a set of algorithms that permit learning information from available data and predict new information in light of the learned information.

The simplest neural network has only one output to which all inputs are connected and is called *single layer perceptron*. Fig. 1.9 shows the perceptron structure and its four main parts: input values, weights and a bias, a weighted sum and the activation function.

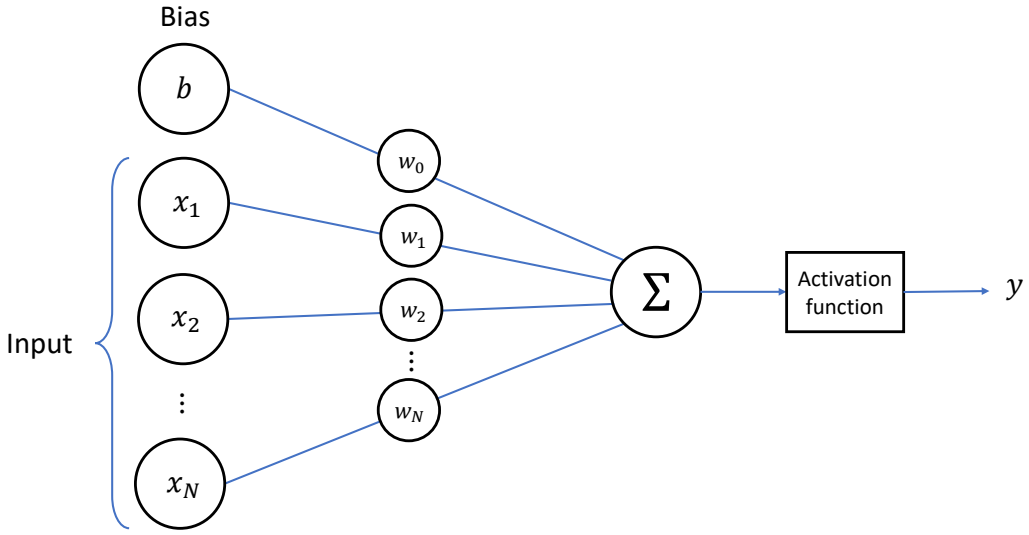
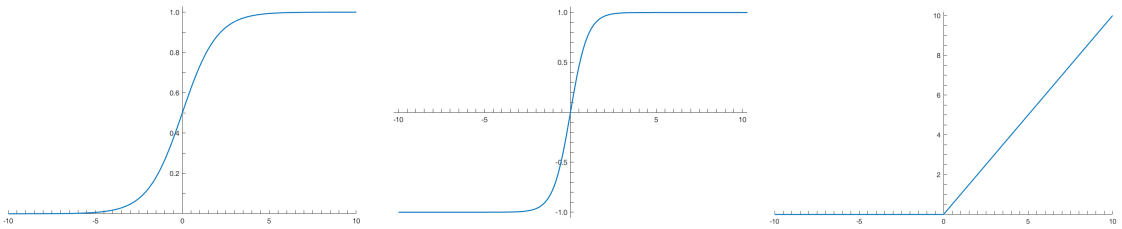


Figure 1.9: Perceptron structure.



(a) (b) (c)

Figure 1.10: Graphics of activation functions: sigmoid (a), tanh (b), ReLU(c).

The idea is simple: given the numerical value of the inputs and the associated weights, there is a function inside the neuron that produces the output

$$y = \sum_{n=0}^{N-1} x_n w_n, \tag{1.9}$$

i.e., the weighted sum of the N inputs. Then, the activation function is used to map y into desired ranges, usually nonlinear, in order to introduce non-linearity into the mapping.

There are different kinds of activation functions, according to the needed range for the output. The most used are:

- logistic function (sigmoid), used to have an output from 0 to 1;
- hyperbolic tangent (tanh), when the output needs to be a number from -1 to 1;
- rectified linear unit (ReLU), used to take the positive numbers as they are and replace the negative ones by 0.

The equations and the graphics for this three types of activation function are shown in Fig. 1.10.

Another important activation function mostly used is the Softmax, which takes the inputs and outputs a vector of values between 0 and 1, whose sum is 1. The output of a Softmax layer depends on the outputs of all other perceptrons in its layer.

Finally, the bias is a value that allows to translate the activation function, so that the output of the perceptron is in the desired range. Thus, (1.9) becomes

$$y = b + \sum_{n=0}^{N-1} x_n w_n \tag{1.10}$$

where, b is the bias.

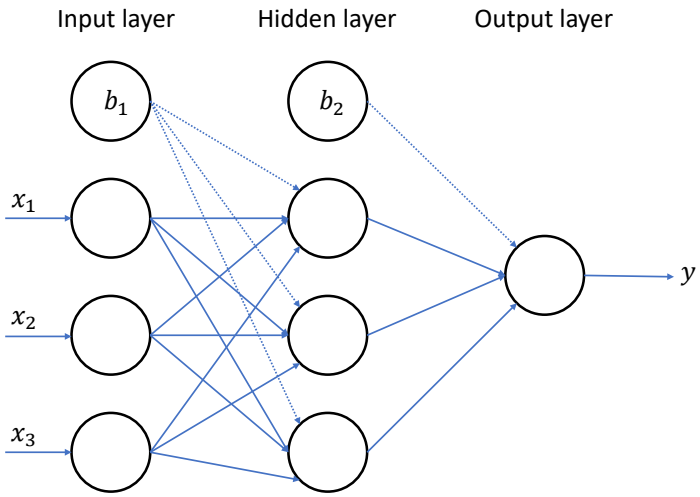


Figure 1.11: Neural Network structure.

A set of perceptrons connected to each other and operating in parallel is a neural network. This set of perceptrons can be divided in layers, where the inputs of one layer consists of the output of the previous one, as shown in Fig. 1.11. The input layer is the layer where the values that will be used for the prediction are brought in; the hidden layer is where the activation values of the inputs are computed and the output layer is where the activation function is used to produce the prediction.

To produce predictions, the hidden layer evaluates whether a certain change in its weights makes the final output improve or worsen and this assessment constitutes the learning process. The two key learning paradigms in neural networks are supervised and unsupervised learning.

In supervised learning methods, for each training example, a set of input values and one or more associated designated output values (or labels) are provided to the algorithm, that tries to learn the rule from the data and generalize it correctly. In this type of learning, it is possible to distinguish two categories of problems:

- regression problems, when the algorithm needs to learn a continuous mapping function.
- classification problems, when the algorithms learns how to assign a class label to the input data.

In unsupervised learning models, the data are provided without an associated label. The success of the learning process is usually determined by whether the network is able

to reduce or increase an associated cost function. Unsupervised learning models are used for clustering problems, where unlabeled data are grouped based on their similarities or differences, for association problems, when different rules to find relationships between variables in the dataset are used, and for dimensionality reduction, to reduce the number of features (or dimensions) of the data inputs to a manageable size while preserving data integrity.

In a neural network, the training phase consists of determining the weights in order to minimize a cost function that represents the error between the expected output values and the values predicted by the network. Weights can be computed with mathematical optimization techniques. The technique typically used is the *gradient descent* through backpropagation.

Gradient descent is a way to minimize a cost function by updating weights in the opposite direction of the gradient of the cost function with respect to the weights [71].

In the gradient descent algorithm, an important parameter is the *learning rate*, that determines the size of the steps to reach the minimum. If the learning rate is chosen too low the algorithm convergence could be very slow (Fig. 1.12(a)), whilst if it is chosen too high the convergence would be fast but there is the possibility that the minimum will be lost (Fig. 1.12(b)). The learning rate is usually chosen variable and generally has a decreasing trend so that the algorithm can make larger steps at the beginning of the training and reduce them when it begins to approach to the minimum (Fig. 1.12(c)). The biggest problem with this technique is the presence of local minima, since they prevent from reaching the global minimum if the step is not large enough.

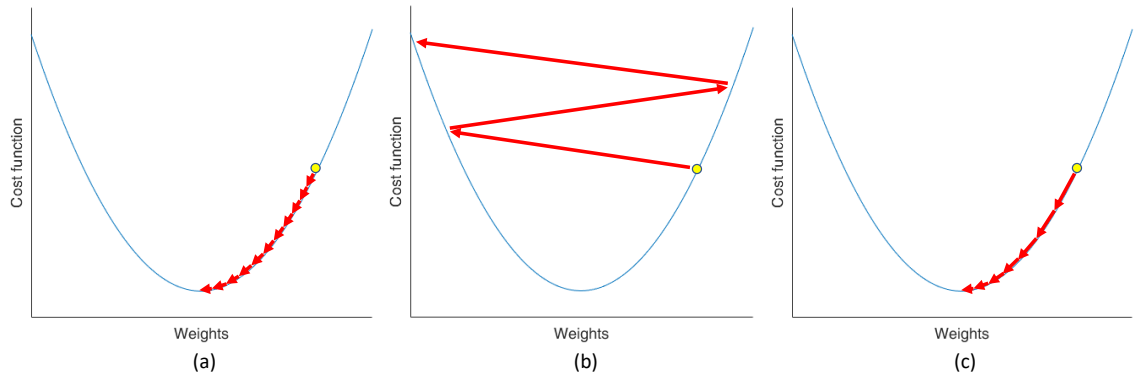


Figure 1.12: Examples of learning rate. When it is chosen too low, the convergence is slow (a). When it is chosen too high, the minimum may be lost (b). When it is variable, the steps are larger at the beginning and they are reduced when the minimum is nearby (c).

There are four variants of gradient descent. They differ in the amount of data that is used to compute the gradient of the cost function:

1. Batch gradient descent (BGD). In this variant, the gradient of the cost function is computed with respect to all the data. Then, the average of the gradients is used to update the weights. This computation is repeated for each step of the training phase (*epoch*). Since the gradients need to be calculated for the whole dataset, batch gradient descent can be very slow and it is not exploitable for datasets that do not fit in memory. On the other hand, it is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces;

2. Stochastic gradient descent (SGD). In this case, the weights are updated after the computation of the gradient with respect to a single example. The term stochastic refers to the fact that the single example is randomly taken. Since only one sample from the dataset is chosen for each iteration, the SGD is faster than the BGD, but the path taken by the algorithm to reach the minimum is usually noisier. However, when the epoch number is high, the cost function decreases with fluctuations, whereby this algorithm is used for large datasets;

3. Stochastic gradient descent with Momentum (SGDM). This variant was created to address the optimization problems of the SGD and it is faster than the previous one. A new hyperparameter is introduced, named *momentum* which determines the contribution to the current iteration of the gradient of the previous step. SGDM is able to reach global minima whereas SGD is stuck in local minima. A disadvantage of this variant is that the momentum could still fluctuate after reaching global minima and take some time to get stable. For this reason, SGDM is slower than other optimization but still faster than SGD.

4. Mini-batch gradient descent. It is mixture of batch gradient descent and SGD. In particular, a batch of a fixed number of examples, smaller than the actual dataset, is used to compute the gradient of the cost function.

Learning algorithms aim to identify a general model, by considering a finite set of data. In some cases the resulting model is unable to learn due to the use of little amount of data, or it is unable to generalize. The first problem is usually called *underfitting*, while the second one is called *overfitting*. To detect underfitting or overfitting phenomena, the initial dataset is usually splitted into three subsets:

Table 1.1: An example of confusion matrix.

	Predicted as Positive	Predicted as Negative
Labeled as Positive	A	B
Labeled as Negative	C	D

- The training set, a dataset of examples used during the learning process.
- The test set, a dataset that is independent of the training dataset, used only to assess the performance at the end of the learning phase.
- The validation set, used to evaluate how well the model makes predictions based on new data and prevent from overfitting on the test set. This happens because the tests on the validation set are made many times during the learning process. In this way, it is possible to realize that the model doesn't generalize enough and intervene to avoid the overfitting.

Another method to evidence the presence of underfitting or overfitting phenomena consists of using metrics for the analysis of the errors. Most of this metrics can be defined using the confusion matrix [72], which shows the ways in which a classification model is "confused" when makes predictions. Table 1.1 contains a confusion matrix, where the rows represent the ground-truth labels and the columns represent the labels predicted by the neural network.

Typically, there are as many matrices as there are classes. In the simple case of two classes, in the confusion matrix, e.g., for class 1, the term "positive" indicates that the object belongs to class 1 and the term "negative" that the object does not belong to that class. In case of multi-class problem, "positive" has the same meaning than before, while "negative" indicates that the object does not belong to any other classes.

Referring to Table 1.1 there are $A + B$ objects labeled as positive, i.e., belonging to a class, but only A of them are correctly recognized as positive by the model, while B of them are incorrectly recognized as negative. At the same time, there are $C + D$ objects labeled as negative: the model correctly recognizes only D as negative, while C objects are incorrectly recognized as positive. The set A of objects represents the *true positive* predictions of the network, B are the *false negative* predictions, C are the *false positive* predictions and D represents the *true negative* predictions.

The most popular metrics for the analysis of the errors are:

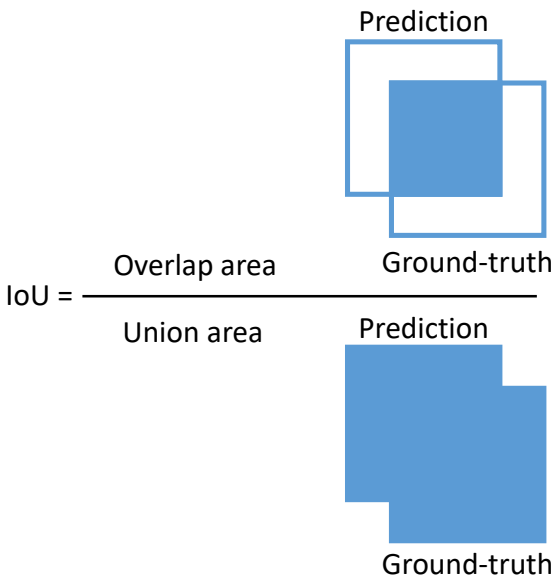


Figure 1.13: Intersection over Union.

- Precision, is the ratio between the true positive and all the positive predictions of the model, i.e. $P = \frac{A}{A+C}$;
- Recall, is the ratio between the true positive and all the actual positive classifications, i.e. $R = \frac{A}{A+B}$;
- F-measure, is a measure of the test's accuracy computed as $F_{measure} = 2\frac{P \times R}{P+R}$;
- Intersection over Union (IoU), that measures the overlap between the predicted bounding box and the labeled one, as shown in Fig. 1.13.

1.5 Deep Neural Networks

The term Deep Learning or Deep Neural Network (DNN) refers to Neural Networks with a very large number of hidden layers. Over the last decades, it has become very popular in the literature as it is able to handle a huge amount of data. The principle of operation is the same as classical neural networks. Indeed, it is still possible to use the classic algorithm of backpropagation for training.

Among DNNs, in the field of computer vision and of speech recognition, the Convolutional Neural Network (CNN) have found great success. CNNs are composed by stacking four types of layers, shown in Fig. 1.14, and the perceptrons in any layer are only connected to a small region of the previous layer [73].

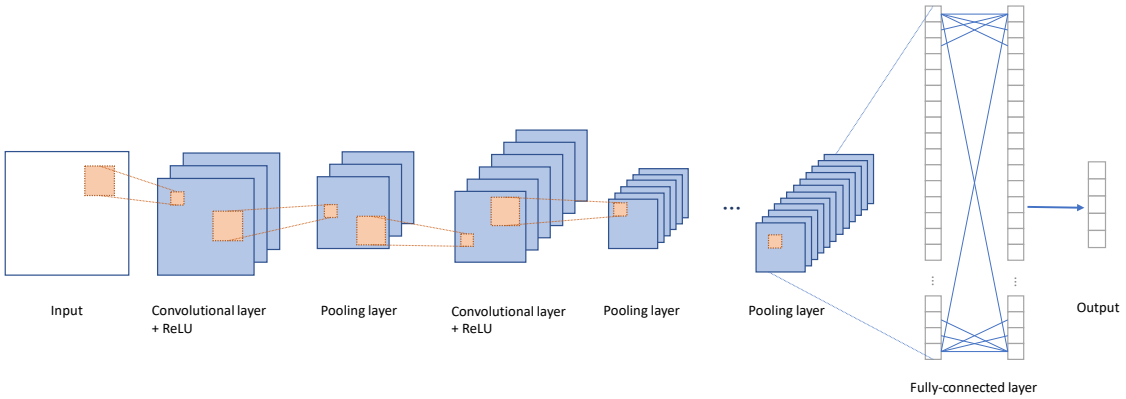


Figure 1.14: Convolutional Neural Network structure.

The input layer holds the pixel values of the input image. Unlike other classical neural networks (where the input is in a vector format), in CNNs the input is a multi-channeled image, e.g., for RGB image it has 3 channels and for a gray-scale image it has a single channel.

The convolutional layer is the most important component of the architecture and contains a set of multiple convolution kernels (also called filters), which gets convoluted with the input image in order to generate the output feature maps and extract different types of features. A kernel is a grid of numbers, where each value is known as the weight of this kernel. The initial steps of a convolution operation between a gray-scale image of dimension 4×4 ad a kernel of size 2×2 are shown in Fig. 1.15. The kernel is slid over all the complete input image (horizontally and vertically), and one scalar value is computed by multiplying the corresponding values of the kernel and the image and sum up all values.

The final output of the convolution operation is shown in Fig. 1.16. In general, the output size shrinks every time a convolutional operation is completed. This represents a problem, since once the convolutional neural network goes deeper, the output of each layer becomes smaller and smaller. In addition, the corner pixels of the image are convoluted much less times than the others and this can lead to a loss of information. Usually, the problem is solved by adding one pixel with intensity value of zero in each border of the input, called *padding*. It is also possible to choose the step size along the horizontal or vertical direction during the convolution. This step size is called *stride*.

In the example of Fig. 1.15, the convolution operation was performed with padding equal to 0 and stride equal to 1. The output feature map size after convolution can be

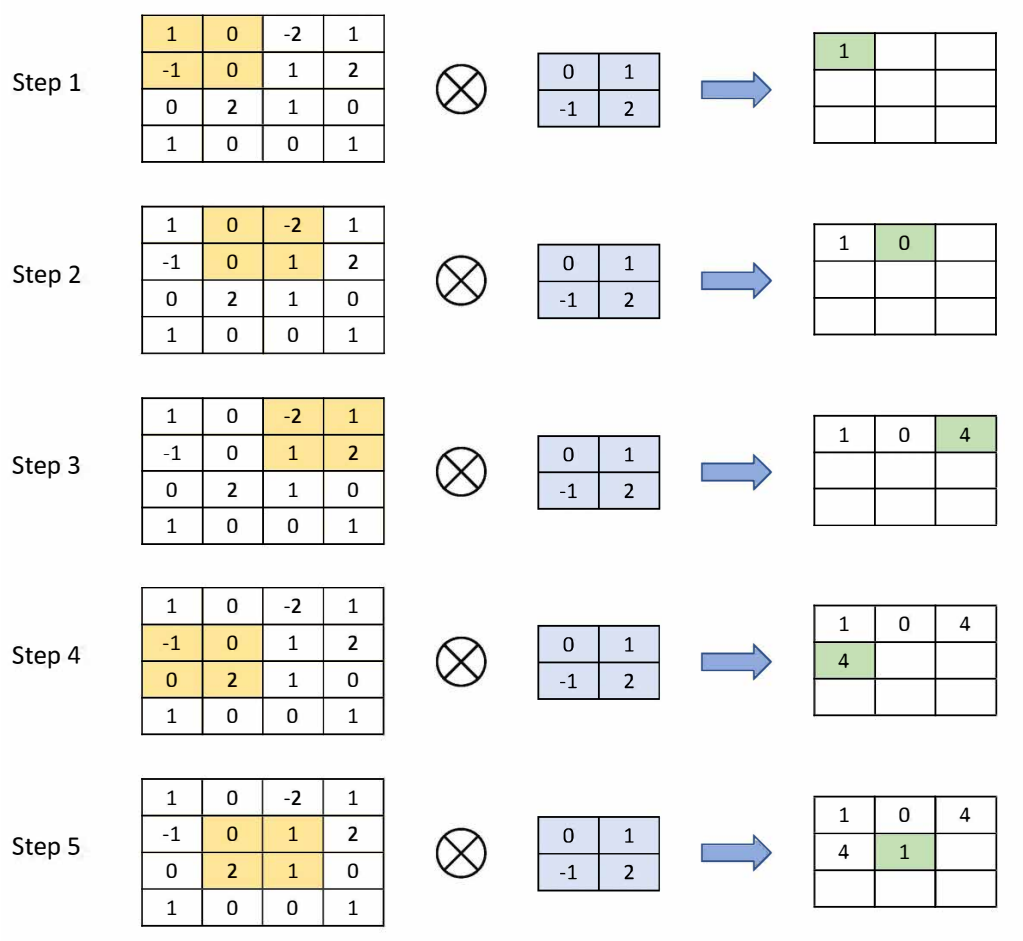


Figure 1.15: First 5 steps of a convolution operation. The kernel (in light blue color) is multiplied with the same sized region of the input image (in yellow color) and values are summed to obtain a corresponding value in the output feature map at each convolution step (in light green).

computed as:

$$h' = \left\lfloor \frac{h - f + p}{s} + 1 \right\rfloor,$$
$$w' = \left\lfloor \frac{w - f + p}{s} + 1 \right\rfloor,$$

where h' and w' are the height and the width of the output feature map, h and w are the height and the width of the input image, f is the kernel size, p and s denote the padding and the stride, respectively, of the convolution operation. The operator $\lfloor \cdot \rfloor$ is the floor function, that takes as input a real number and gives as output the greatest integer less than or equal to that number.

The pooling layers are used to perform a downsampling of the feature maps produced

1	0	4
4	1	1
1	1	2

Figure 1.16: The feature map after the complete convolution operation shown in Fig. 1.15

by convolution operations. In other words, the pooling layer takes a large feature map and reduces it to a smaller one, while preserving the most dominant features in each pool steps. The pooling operation is performed by specifying the pooled region size and the stride of the operation, similar to convolution operation.

There are different types of pooling techniques such as max pooling (the most used), min pooling, average pooling etc. [74]. The initial steps of a max pooling operation and the relative final result are shown in Fig. 1.17. The main drawback of the pooling is that it sometimes decreases the overall performance of CNN. A simple example of information loss using max pooling [75] is shown in Fig. 1.18. In particular, an example of input feature map, in which most of the elements in the pooling region are of high magnitudes, and the output after max pooling is shown in Fig. 1.18(a), while in Fig. 1.18(b), the digits before and after the max pooling operation are shown. As it can be seen, the information present in the input feature map, corresponding to the black line (or the zero pixels in the matrix) is completely lost.

The formulas to find the output feature map size after pooling operation are reported below:

$$h' = \left\lfloor \frac{h-f}{s} \right\rfloor,$$
$$w' = \left\lfloor \frac{w-f}{s} \right\rfloor,$$

where h and w are the height and the width of the input feature map, f and s denote the pooling region size and the stride of the pooling operation, respectively.

The fully-connected layer performs the same operations of standard neural network in order to make a classification. These layers take input from the last convolutional or pooling layer, which is in the form of a set of feature maps. First, this maps are flattened to create a vector and, then, this vector is fed into the fully-connected layer to generate the final output of the CNN.

In a CNN, the performance is proportional to the amount of data used to train it. Usually, a technique of *data augmentation* is used to artificially increase or expand the size

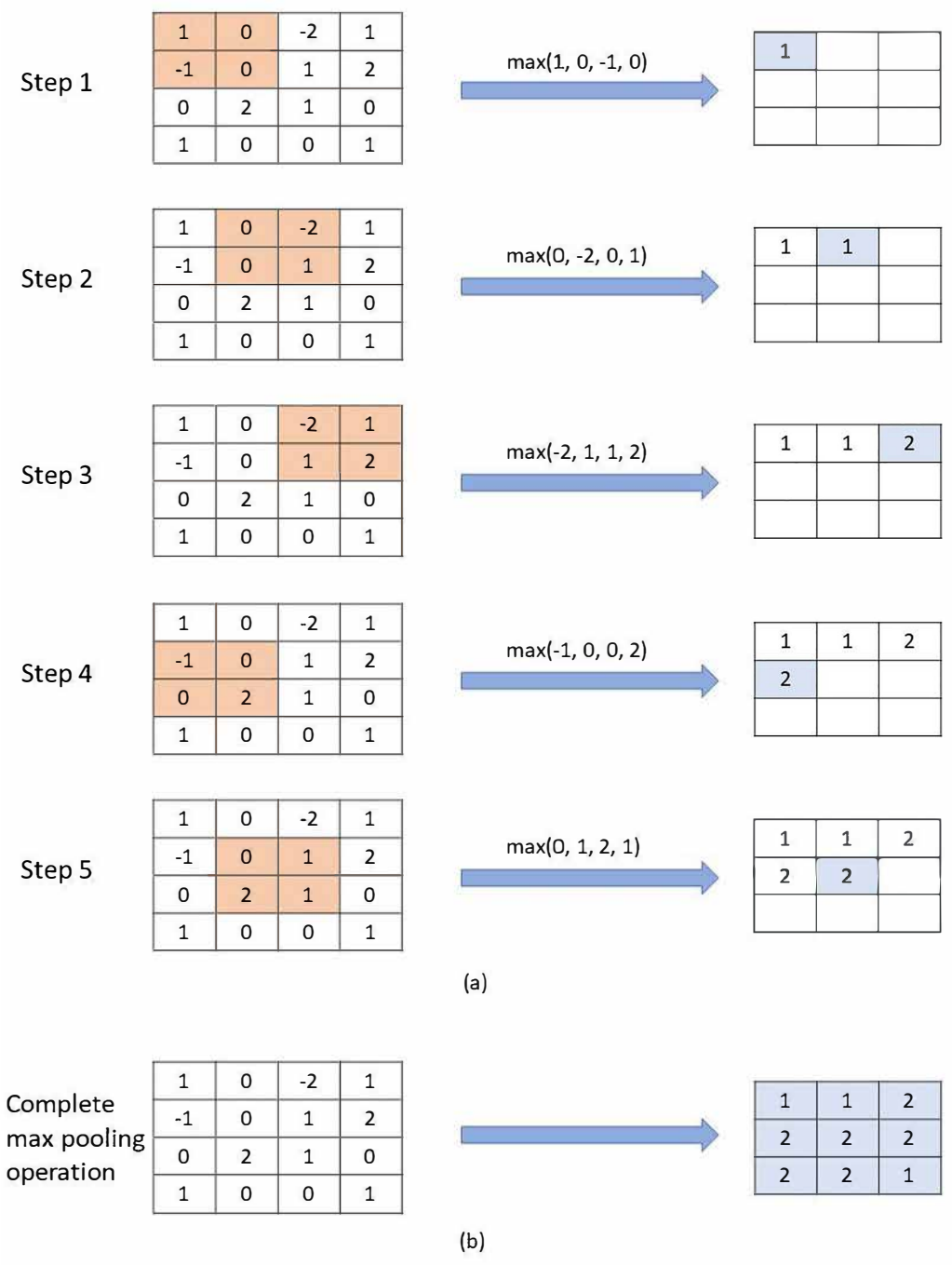


Figure 1.17: First 5 steps of a max pooling operation. The size of the pooling region is 2×2 (in light orange in the input feature map) and the stride is equal to 1. In light blue, the corresponding computed values in the output feature map are shown (a). The input feature map on the left and the result of the complete max pooling operation on the right (b).

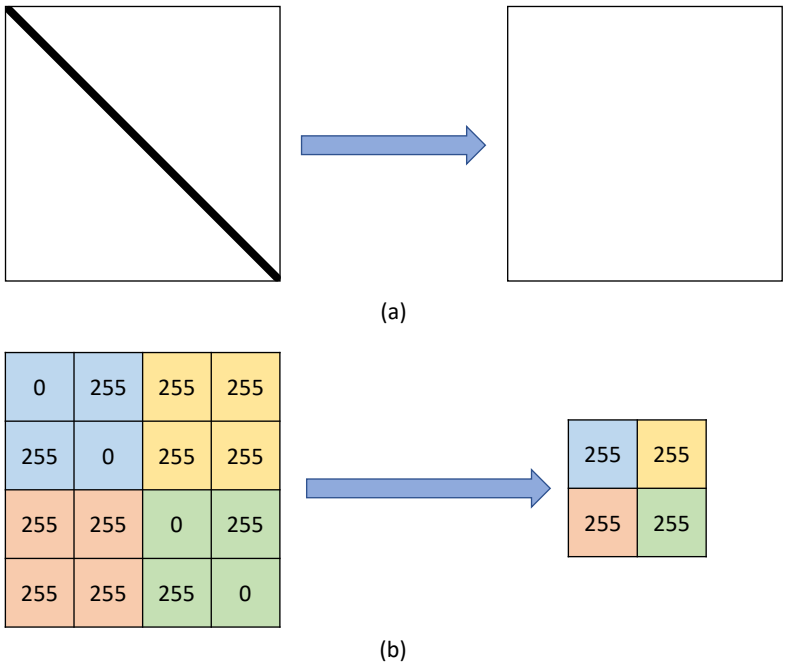


Figure 1.18: On the left, the input feature map, on the right, the result after the max pooling operation (a). The digit example of the max pooling operation. It can be seen that the information about the zeros on the diagonal are loss (b).

of the training dataset. There are several data augmentation operations, such as cropping, rotations, flipping, translations, contrast adjustment, scaling, etc. These operation can be used separately or in combination to make several new versions of a single image. The use of this technique can help to avoid the overfitting problem.

1.5.1 Object detection

In object detection problems it is assumed that the input image contains more than one object. The aim consists of detecting the objects with their correct location inside that image by using CNN models. For Peg-in-Hole applications a supervised approach has been chosen because the surface of industrial object is often textured and this can lead to false positives when using unsupervised computer vision techniques (e.g., the Hough circle transform).

The first CNN model designed for object detection is the Region-based CNN (R-CNN) [76], which has the structure depicted in Fig. 1.19.

A R-CNN is a two-stage detector: in the first stage a set of region proposals is extracted, i.e., a region that may likely contain some object. Then, in the second stage, each

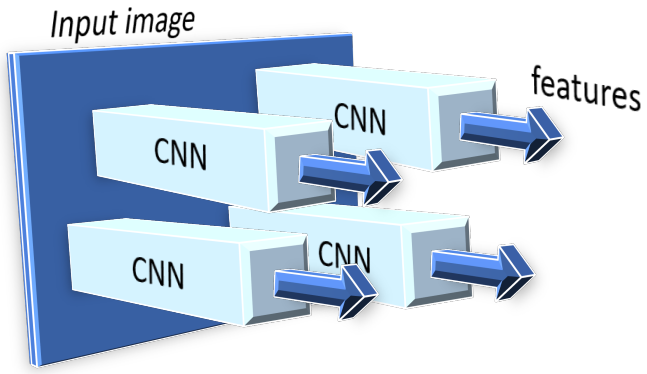


Figure 1.19: R-CNN structure.

proposal is fed to a CNN model to extract features and a classifier is used to detect the object and identify its class. The major drawback of R-CNNs is the slow detection speed, due to the redundant feature computations on the overlapped region proposals.

To overcome such a problem of R-CNN, the Fast R-CNN [77] architecture, shown in Fig. 1.20, uses a convolution layer before extraction of region proposals, in order to achieve better performance. The methods used to find the regions of interest are still slow and time-consuming, although better than those used in basic R-CNN. In fact, when large real-life datasets are considered, this architecture doesn't appear so fast anymore.

In [78] the first near-realtime deep learning detector has been proposed, obtained by the Region Proposal Network (RPN), which enables nearly cost-free region proposals. This architecture is named Faster R-CNN and its structure is shown in Fig. 1.21. The RPN is a fully convolutional network used to produce high-quality region proposals.

In detail, Faster R-CNN is composed by two modules:

1. The Regional Proposal Network (RPN), which is a CNN that outputs the region

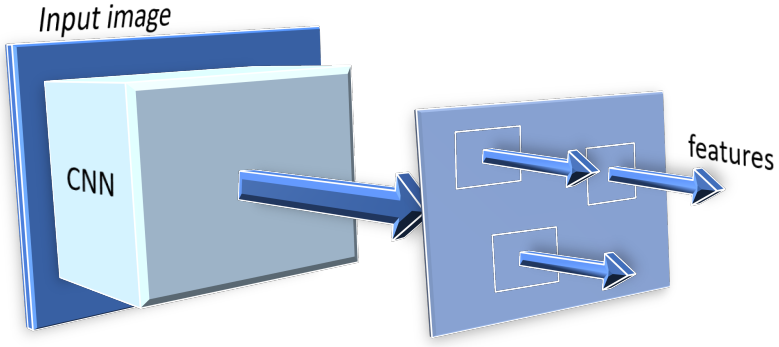


Figure 1.20: Fast R-CNN structure.

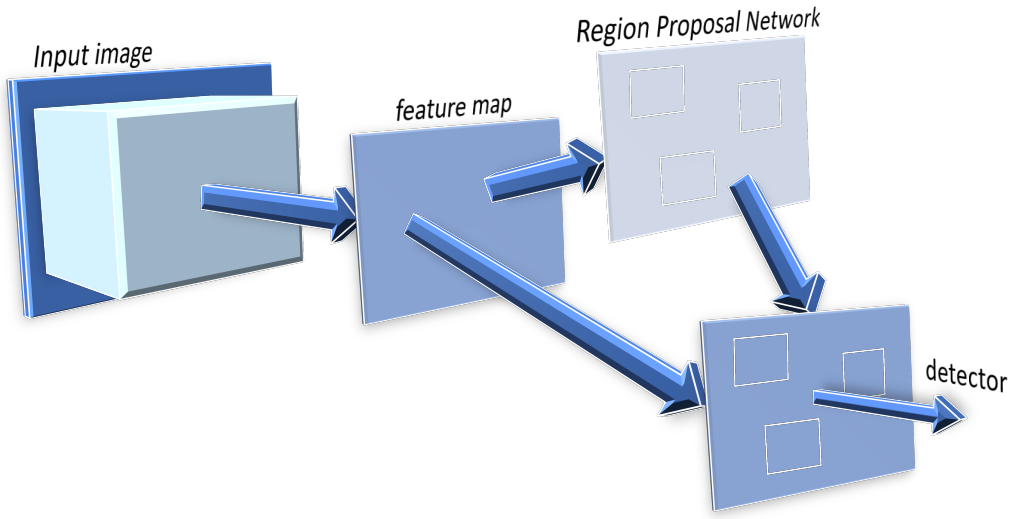


Figure 1.21: Faster R-CNN structure.

proposals with the highest probability of object presence.

2. The Fast R-CNN detector. The regions generated by the RPN are fed to the Fast R-CNN detector in order to refine them and to determine the class membership of the object.

A further improvement has been achieved with the first one-stage detector, named You Only Look Once (YOLO) [79]. Unlike Faster R-CNN, YOLO partitions the image into regions and predicts bounding boxes and probabilities for each region at the same time. The main feature of YOLO is its capability of making predictions considering object detection as a single regression problem. The YOLO architecture consists of three main parts: the *backbone*, that performs the feature extraction, the *neck*, that executes a fusion of the features gathered from the different layers of the backbone model, and the *head*, that predicts the final output, which is composed by a vector containing the bounding box coordinates: width, height, class label and class probability.

The first version of YOLO was characterized by a lower localization accuracy compared with two-stage detectors. However, the new versions of YOLO present an improved detection accuracy still keeping a high detection speed. YOLOv1 [79] divides the input image into $S \times S$ grid cells of equal dimensions. Each grid cell is responsible for object detection if the center of the objects falls inside the cell and can predict a fixed number B of bounding box coordinates. After the prediction, YOLO uses IoU to choose the right bounding box of an object in the grid cell. If IOU exceeds a chosen value, the

bounding boxes with low confidence score are suppressed. To calculate loss, YOLO uses the sum of squared error.

In YOLOv2 [80] batch normalization was added together with convolution layers to improve the accuracy and reduce overfitting problems. Batch normalization is a layer used to normalize the output of the previous layers. Learning becomes efficient by using batch normalization, indeed it can be used to avoid overfitting of the model [81]. Darknet-19 [82] is the CNN used as backbone of YOLOv2.

Since Darknet-19 isn't able to detect small objects, in YOLOv3, the feature extraction backbone was changed to Darknet-53, which is a more extensive network, but is much more accurate and faster [83].

In YOLOv4 [84] again the feature extractors backbone was changed with one based on the Cross-stage Partial (CSP) Connection. The CSP Connection is a technique that divides the feature map of the current layer into two parts, one to pass through convolution layers and the other that would not pass through convolutions. After the convolutions, the parts are aggregated again. This technique significantly improves the speed and accuracy of the algorithm [85].

YOLOv5 [86] is a lightweight version of previous ones and uses PyTorch framework instead of Darknet framework. The head in YOLOv5 is the same as YOLOv4 and YOLOv3, which generates three different output of feature maps to achieve multi scale prediction [87]. YOLOv5 has not been designed by the authors of the other versions, in fact, one of the original authors provided benchmarks comparing YOLOv4 vs. YOLOv5, showing that YOLOv4 is equal or better.

Among the different supervised object detection architectures described above, the YOLO has been chosen, since it is faster than other classifier-based systems but with similar accuracy.

1.5.2 Image segmentation

Image segmentation consists of subdividing an image into multiple segments and objects [88]. There are three types of techniques:

- semantic segmentation, which assigns a corresponding and unique class label to each pixel in an image,
- instance segmentation, that identifies every item or instance of a class visible in an image and gives it a distinct bounding box with a special identification,

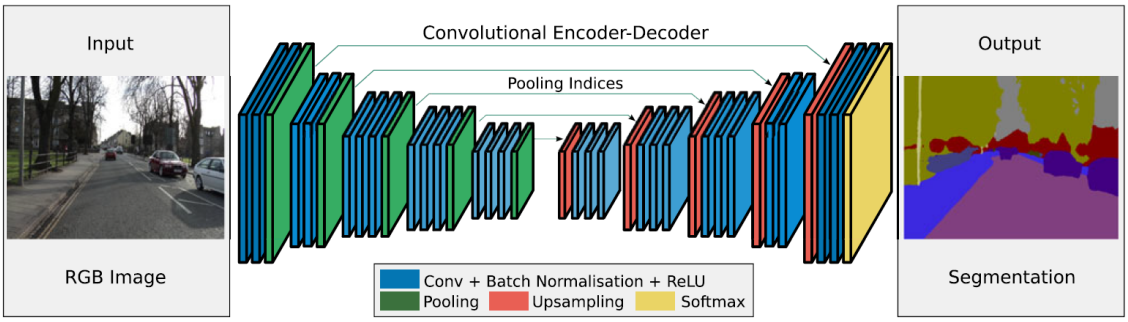


Figure 1.22: SegNet structure. The Figure is taken from [100].

- panoptic segmentation, offers a unified method in which each pixel in an image is given a semantic label (semantic segmentation) and a special instance identification (instance segmentation).

The semantic segmentation is particularly suitable to be addressed with automatic feature extraction-based algorithms such as DNNs [89].

The main advantage of DNN-based approaches is removing the issues related to human engineering and providing an off-the-shelf model usable in real-time applications. Indeed, in the last years, DNN-based approaches for semantic segmentation have been used in several contexts such as medical image analysis [90, 91], virtual and augmented reality [92,93], autonomous driving [94,95], cleaning of point cloud in industrial applications [96,97] and robotic applications [98,99].

One of the first innovative approaches based on DNNs for pixel-wise semantic segmentation is SegNet [100]. SegNet has an encoder network and a corresponding decoder network, followed by a final pixelwise classification layer (Fig. 1.22). The encoder component corresponds to the first 13 layers of another network structure designed for object classification. This structure is the VGG16 (Visual Geometry Group) [101], that is a deep convolutional neural network consisting of 16 convolutional layers.

In SegNet, for each encoder layer there is a corresponding decoder layer. The final decoder output is fed to a multi-class SoftMax classifier to produce class probabilities for each pixel independently. Although SegNet performed about the 90% as pixel-classification accuracy in outdoor scenes, it does not achieve the same accuracy in indoor scenes, due to the increased cluttering.

An important approach for semantic segmentation was provided in [102], which proposed DeepLab, an innovative DNN architecture designed to tackle the classic DNN problems in image segmentation tasks such as reduced feature resolution and objects

at multiple scales. The first challenge was faced by removing downsampling operation from the last few DNN max pooling layers obtaining feature maps computed with a high sampling rate [103]. To address multiple scales objects, the authors proposed a scheme for resampling a feature layer at multiple rates prior to convolution, calling such a method *atrous spatial pyramid pooling* (ASPP).

A novel architecture, called DeepLabv3+, has been proposed in [104] and represents a turning point in the research area. It reached the state-of-the-art DNNs for semantic segmentation and it achieved impressive results on many benchmark datasets and in various research fields [92, 93, 105–108]. For this reason this architecture has been used for image segmentation in one of the two Peg-in-Hole strategies.

1.6 Admittance control for Peg-In-Hole tasks

Since in the Peg-in-Hole application physical interaction between the robot and workpiece arises, in order to keep bounded the interaction wrench, the manipulator has been controlled in a compliant mode by implementing the admittance control schema described below.

Consider a coordinate frame attached to the robot end-effector, $\Sigma_e(O_e, \boldsymbol{x}_e, \boldsymbol{y}_e, \boldsymbol{z}_e)$, such that the $\boldsymbol{z}_e$ axis is coincident with the approach direction of the gripper, as shown in Fig. 1.23, the position of O_e with respect to the robot base frame is expressed by the (3×1) vector $\boldsymbol{p}_e$, while the orientation of Σ_e can be expressed either by the (3×3) orientation matrix, $\boldsymbol{R}_e$, or by a (3×1) vector of Euler angles, $\boldsymbol{\phi}_e$ (e.g., roll-pitch-yaw angles), extracted from $\boldsymbol{R}_e$ [69]. Let $\boldsymbol{x}_e = \left[\boldsymbol{p}_e^{\mathrm{T}}\ \boldsymbol{\phi}_e^{\mathrm{T}}\right]^{\mathrm{T}}$ the (6×1) operational space vector.

Let $\boldsymbol{q}$ ($\dot{\boldsymbol{q}}$) be the $(\nu \times 1)$ vector of joint positions (velocities). The direct kinematics of the manipulator can be expressed either via the functions $\boldsymbol{p}_e = \boldsymbol{p}(\boldsymbol{q})$ and $\boldsymbol{R}_e = \boldsymbol{R}(\boldsymbol{q})$, or by the (6×1) vector function $\boldsymbol{x}_e = \boldsymbol{k}(\boldsymbol{q})$.

The $(6 \times \nu)$ matrix

$$\boldsymbol{J}(\boldsymbol{q}) = \left[\boldsymbol{J}_P^{\mathrm{T}}(\boldsymbol{q})\ \boldsymbol{J}_O^{\mathrm{T}}(\boldsymbol{q})\right]^{\mathrm{T}} \tag{1.11}$$

is the robot geometric Jacobian, which relates the joint velocities to the linear and angular velocity of the end-effector, where $\boldsymbol{J}_P(\boldsymbol{q})$ is the $(3 \times \nu)$ positional part and $\boldsymbol{J}_O(\boldsymbol{q})$ is the $(3 \times \nu)$ orientation part of $\boldsymbol{J}(\boldsymbol{q})$.

It is assumed that the robot manipulator is not equipped with a wrist-mounted force/torque sensor, but only with torque sensors at the joints, as usual in collaborative robots.

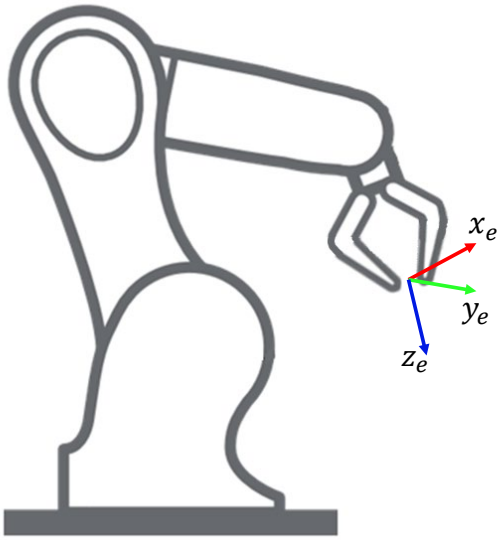


Figure 1.23: Reference frame attached to the robot end-effector.

Using a Lagrangian approach [69], the dynamics of a ν degrees of freedom manipulator is given by

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_e \tag{1.12}$$

where $\boldsymbol{q} \in \mathbb{R}^\nu$ ($\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$) is the vector of joint positions (velocities and accelerations), $\boldsymbol{\tau} \in \mathbb{R}^\nu$ is the vector of joints torques, $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{\nu\times\nu}$ is the symmetric and positive definite inertia matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) \in \mathbb{R}^{\nu\times\nu}$ is the centripetal and Coriolis terms matrix, $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^\nu$ is the vector of gravity terms, $\boldsymbol{F} \in \mathbb{R}^{\nu\times\nu}$ is the matrix of viscous friction terms and the term $\boldsymbol{\tau}_e \in \mathbb{R}^\nu$ represents the torques induced at the joints by the contact wrench at the robot end-effector.

The wrench acting on the end-effector can be estimated via an observer based on the generalized momentum

$$\boldsymbol{\upsilon} = \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}\,. \tag{1.13}$$

By considering the dynamic model (1.12), an estimate of $\boldsymbol{\tau}_e$ can be computed as follows [109]

$$\hat{\boldsymbol{\tau}}_e = \boldsymbol{K}_o\left[(\boldsymbol{\upsilon}(t) - \boldsymbol{\upsilon}(t_0)) - \int_{t_0}^{t}(\boldsymbol{C}^{\mathrm{T}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{F}\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau} + \hat{\boldsymbol{\tau}}_e)d\varsigma\right], \tag{1.14}$$

where t and t_0 are the current and initial time instant, respectively, and $\boldsymbol{K}_o \in \mathbb{R}^\nu$ is a positive definite gain matrix.

By reasonably assuming that $\dot{\boldsymbol{q}}(t_0)$ is null, then $\boldsymbol{\upsilon}(t_0)$ is null as well, and by considering that the derivative of the inertia matrix is $\dot{\boldsymbol{M}}(\boldsymbol{q}) = \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{C}^{\mathrm{T}}(\boldsymbol{q},\dot{\boldsymbol{q}})$, the following

dynamics for the estimation is obtained

$$\dot{\hat{\boldsymbol{\tau}}}_e + \boldsymbol{K}_o \hat{\boldsymbol{\tau}}_e = \boldsymbol{K}_o \boldsymbol{\tau}_e. \tag{1.15}$$

Equation (1.15) is a first-order low-pass dynamic system; therefore, $\hat{\boldsymbol{\tau}}_e \to \boldsymbol{\tau}_e$ as $t \to \infty$ for any positive definite gain matrix $\boldsymbol{K}_o$. The torques $\boldsymbol{\tau}_e$ can be expressed as

$$\boldsymbol{\tau}_e = \boldsymbol{J}^{\mathrm{T}} \boldsymbol{h}_e, \tag{1.16}$$

where $\boldsymbol{h}_e = [\boldsymbol{f}_e^{\mathrm{T}} \; \boldsymbol{\mu}_e^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^6$ is the contact wrench acting on the end-effector, $\boldsymbol{f}_e \in \mathbb{R}^3$ is the force and $\boldsymbol{\mu}_e \in \mathbb{R}^3$ is the moment. Therefore, an estimate of the external wrench is [110]

$$\hat{\boldsymbol{h}}_e = \boldsymbol{J}^{\dagger \mathrm{T}}(\boldsymbol{q}) \hat{\boldsymbol{\tau}}_e. \tag{1.17}$$

where $\boldsymbol{J}^{\dagger}$ is a right pseudo-inverse of $\boldsymbol{J}$.

By following the admittance control scheme proposed in [111], to confer proper compliance to the end-effector, its reference pose $\boldsymbol{x}_{e,r} = [\boldsymbol{p}_{e,r}^{\mathrm{T}} \; \boldsymbol{\phi}_{e,r}^{\mathrm{T}}]^{\mathrm{T}}$ is computed as

$$\boldsymbol{M}_p \ddot{\boldsymbol{x}}_{dr}^e + \boldsymbol{D}_p \dot{\boldsymbol{x}}_{dr}^e + \boldsymbol{K}_p \boldsymbol{x}_{dr}^e = \boldsymbol{T}_A^{\mathrm{T}}(\boldsymbol{\phi}_{dr}^e) \hat{\boldsymbol{h}}^e, \tag{1.18}$$

where $\boldsymbol{M}_p$, $\boldsymbol{D}_p$ and $\boldsymbol{K}_p$ are, respectively, the virtual inertia, damping and stiffness matrices imposed to the end-effector. The vector $\boldsymbol{x}_{dr}^e$ is the relative displacement between the desired and the reference pose, expressed in the frame Σ_e

$$\boldsymbol{x}_{dr}^e = \begin{bmatrix} \boldsymbol{p}_{dr}^e \\ \boldsymbol{\phi}_{e,dr}^e \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_e^{\mathrm{T}}(\boldsymbol{p}_{e,d} - \boldsymbol{p}_{e,r}) \\ \boldsymbol{\phi}_{dr}^e \end{bmatrix}, \tag{1.19}$$

where $\boldsymbol{p}_{e,d}$ and $\boldsymbol{p}_{e,r}$ are the desired and reference position, respectively, while $\boldsymbol{\phi}_{dr}^e$ is the vector of Euler angles expressing the relative orientation between the desired frame (whose orientation is expressed by the rotation matrix $\boldsymbol{R}_{e,d}$) and the reference frame (whose orientation is expressed by the rotation matrix $\boldsymbol{R}_{r,d}$), i.e., is the vector of Euler angles extracted from the matrix $\boldsymbol{R}_{e,d}^{\mathrm{T}} \boldsymbol{R}_{e,r}$. The matrix $\boldsymbol{T}_A(\cdot)$ in (1.18) is given by

$$\boldsymbol{T}_A(\cdot) = \begin{bmatrix} \boldsymbol{I}_{3\times 3} & \boldsymbol{O}_{3\times 3} \\ \boldsymbol{O}_{3\times 3} & \boldsymbol{T}(\cdot) \end{bmatrix},$$

where $\boldsymbol{T}(\cdot)$ is the matrix that maps the time derivative of the Euler angles to the angular velocity [69], $\boldsymbol{I}_{3\times 3}$ and $\boldsymbol{O}_{3\times 3}$ are the (3×3) identity and null matrices, respectively. The output of the admittance filter (1.18) is the reference trajectory, $\boldsymbol{x}_{e,r}$, that can be commanded to the robot.

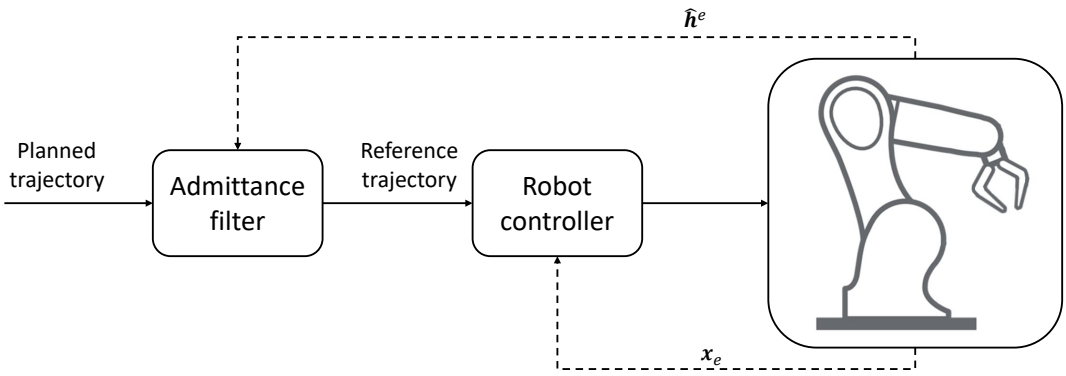


Figure 1.24: Admittance filter scheme.

A scheme of the described admittance filter is shown in Fig. 1.24.

In the Peg-in-Hole application, in order to overcome the drawback of representation singularities, as illustrated in [111], the admittance filter has been implemented at the peg tip level and the variables involved in the filter have been written with respect to Σ_p.

Chapter 2

Application of collaborative robotics for assembly tasks in partially structured environments

The methods presented in the previous Chapter are combined to address the Peg-in-Hole task. In particular, two strategies have been developed. The first [112] is more accurate and exploits the 3D-DIC technique to reconstruct the surface, a CNN to detect the holes on the surface and an admittance control scheme to insert the peg. The second strategy is faster than the first one, and uses the ICP method to reconstruct the surface through a set of point clouds filtered by using a neural network for image segmentation, a CNN to detect the holes and an admittance control strategy to perform the insertion.

In the following Sections, the strategies are detailed and the conducted experiments are presented. For both the strategies, the setup consisted of a robot manipulator equipped with a camera in eye-in-hand configuration. In particular, the Franka Emika Panda [24] robot has been used, characterized by 7 revolute joints, each mounting a torque sensor. The robot is controlled by using the Franka Control Interface (FCI) with the `libfranka` C++ open source library, that enables direct robot control with an external workstation connected via Ethernet. The interface provides joint positions, velocities and torques. The robot has been controlled in velocity-mode by sending the desired joint velocities computed via (2.7) for the approach and (2.10) for the insertion, respectively. The workstation is equipped with the Ubuntu 18.04 LTS operating system running on an Intel Xeon 3.7 GHz CPU with 32 GB RAM. The vision system is composed by an Intel Realsense D435 camera, mounted on the end-effector via a 3D printed support. The vision software is

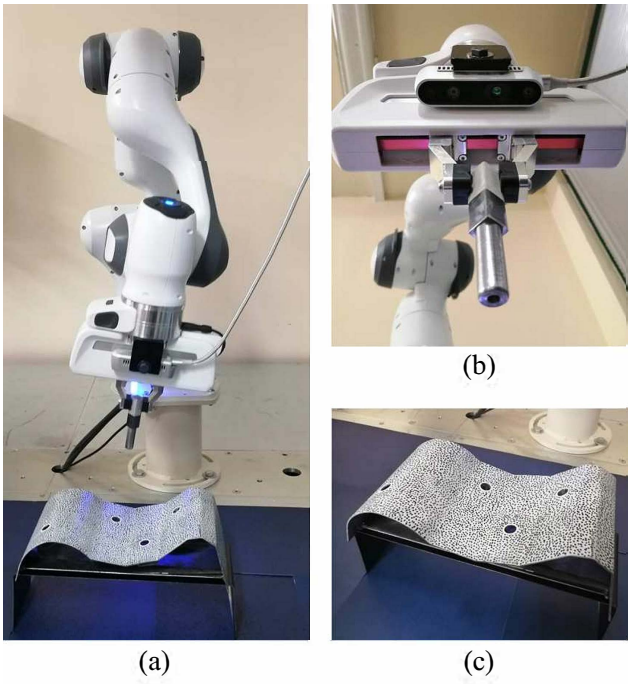


Figure 2.1: First strategy application scenario: a) the Franka Emika Panda robot used in the experiments; b) visual sensor and the peg; c) workpiece.

executed on the above described workstation and the `librealsense2` library has been used in order to acquire the camera data.

2.1 First strategy

For this strategy, the setup is shown in Fig. 2.1. The camera holds two stereo infrared imagers, an RGB module and an infrared projector. The robot carries a steel peg that has a diameter of 12.2 mm, as shown in Fig. 2.1(b), while the workpiece is a surface obtained by bending and forming a steel sheet into a geometry with both flat and curved regions (Fig. 2.1(c)). The surface holds four unchamfered holes, characterized by a diameter of 12.4 mm and their tilt is unknown (Fig. 2.2). The workpiece is positioned in the robot workspace with positional uncertainty much larger than the size of the holes, the peg is symmetric and firmly grasped by the robot gripper.

The coordinated frames of interest are the inertial frame, Σ, coincident with the robot base frame, the frame attached to the robot end-effector, Σ_e, the frame attached to the tip of the peg, $\Sigma_p = \{O_p, \boldsymbol{n}_p, \boldsymbol{s}_p, \boldsymbol{a}_p\}$ (Fig.2.3) and β reference frames $\Sigma_{h_i} = \{O_{h_i}, \boldsymbol{n}_{h_i}, \boldsymbol{s}_{h_i}, \boldsymbol{a}_{h_i}\}$ ($i = 1, \ldots, \beta$), each attached to the center of the β holes (Fig. 2.2).

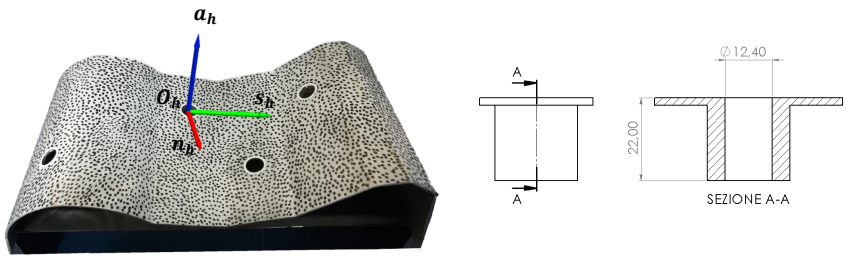


Figure 2.2: Workpiece with a Σ_h frame (left) and hole dimensioning (right).

Due to the assumption of rigid grasp for the peg, the position, $\boldsymbol{p}_p$, of O_p can be described as

$$\boldsymbol{p}_p = \boldsymbol{p}_e + \boldsymbol{R}_e \boldsymbol{p}_p^e, \tag{2.1}$$

where $\boldsymbol{p}_e$ and $\boldsymbol{R}_e$ are, respectively, the position and orientation of the robot end-effector, given by the standard direct kinematics, $\boldsymbol{p}_p^e$ is the constant relative position of O_p in Σ_e. The orientation of Σ_p is represented by the rotation matrix

$$\boldsymbol{R}_p = \boldsymbol{R}_e \boldsymbol{R}_p^e, \tag{2.2}$$

where $\boldsymbol{R}_p^e$ is the constant relative rotation matrix of the peg frame with respect to end-effector frame.

In Fig. 2.4, the pipeline of this strategy is shown and it is detailed in the following:

- The robot moves its eye-in-hand camera over a semi-sphere spanning its workspace, in such a way to scan the workpiece surface. In each position, a pair of images is acquired with the previously calibrated stereo-cameras system. In the current setup, the two Realsense IR sensors are used.
- The acquired images are the input for the CNN that detects the holes on the workpiece surface.

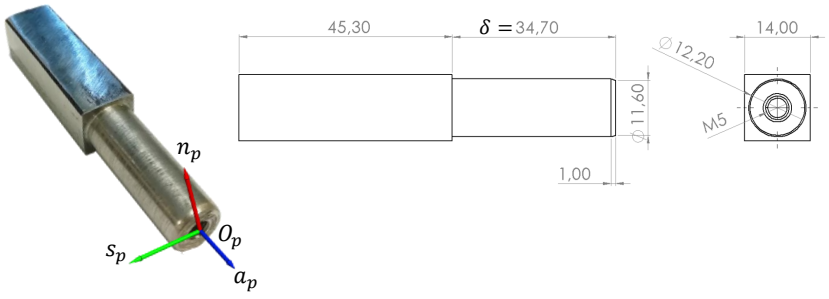


Figure 2.3: Peg with the attached frame (left) and its dimensioning (right).

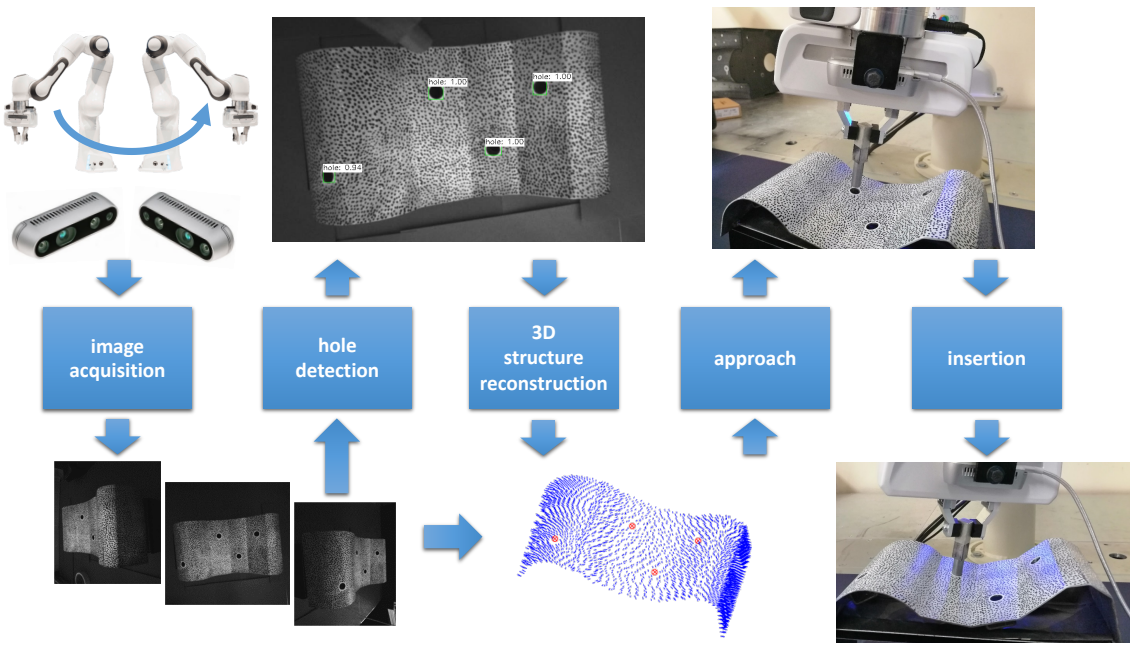


Figure 2.4: The Peg-in-Hole pipeline designed in the first strategy.

- Each pair of stereo-images from the acquired series is processed with the 3D-DIC algorithm to retrieve the 3D position of a highly dense set of points on a portion of the workpiece surface; a subsequent merging operation allows the transformation of each reconstructed point cloud from the local coordinate system to a common global reference system. Stereo-triangulation is used to determine the 3D position of the centers of the holes. Finally, the distribution of the local normal vector is computed over the tessellated surface reconstructed with 3D-DIC.
- The robot, moves the peg close to the hole, aligning the approach unit vector with the hole axis (*approach* phase).
- The peg is inserted in the hole by moving the robot under an admittance control strategy, so as to confer suitable compliance to the peg (*insertion* phase).

2.1.1 3D surface reconstruction

The workpiece surface is provided with a stochastic black speckle pattern on a white background, thus allowing the implementation of an area-based image registration with a subset-based DIC approach (see Section 1.2). Examples of image pairs acquired, at the beginning of the process, by the stereo IR Realsense sensors at the maximum spatial resolution (1280×720 pixels), are shown in Fig. 2.5.

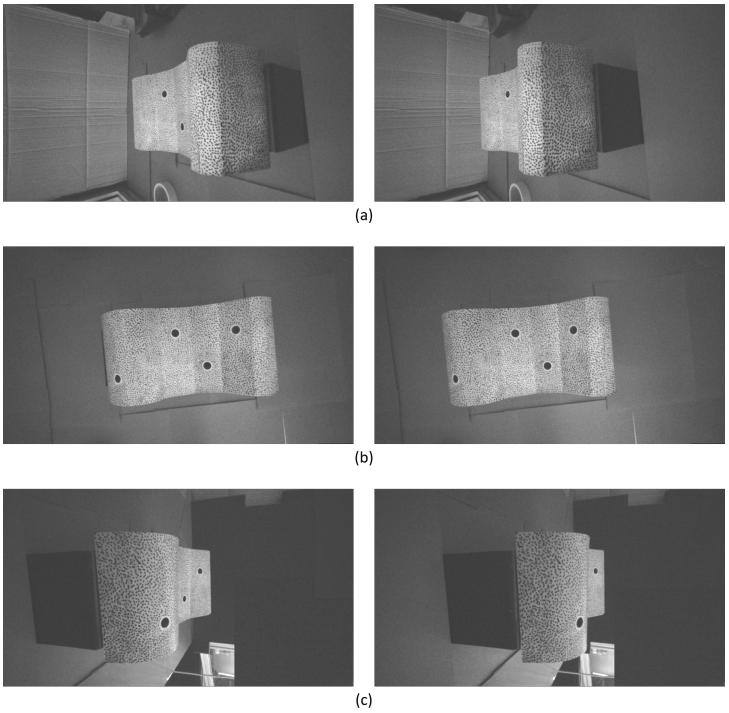


Figure 2.5: Examples of infrared image pairs acquired by the sensor in the first position of the robot trajectory (a), in a middle position (b), and in the last position of the spanning trajectory (c). The images brightness has been increased in this figure to better see the surface.

For each captured image pairs, a regular dense grid of points (5 pixels of step size) is defined over the region of interest (ROI) of the reference image, i.e, the image taken in the master camera position. The master camera position refers to the first position of the spanning trajectory used to scan the surface. The DIC algorithm then seeks for the best correspondence in the target image on the basis of the local distribution of the greyscale intensity over a subset of 21×21 pixels around each data point. The sensor coordinates of the pairs of corresponding image points are hence used to reconstruct the position of the 3D workpiece point via triangulation. To this aim, the stereo-camera system was previously calibrated by using 30 images of a 2D checkerboard flat pattern according to the camera calibration method proposed in [113]. An average reprojection error of 0.036 and 0.042 pixels was found for the right and left camera, respectively. The target was reconstructed in the 30 positions over the measurement volume with an error of 0.24 ± 0.17 mm.

For each position of the scanning sequence, a point cloud of a portion of the workpiece surface is reconstructed in the reference system of the master camera (Fig. 2.6).

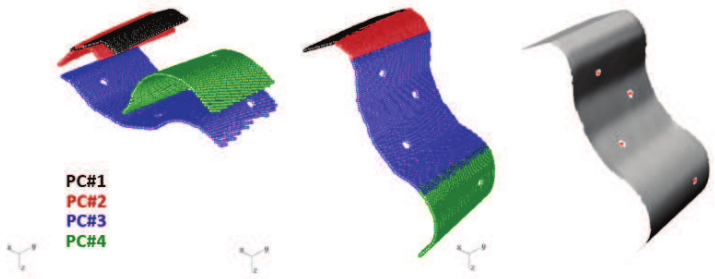


Figure 2.6: 3D reconstruction and merging of the point clouds from four different contiguous positions of the eye-in-hand camera. The z axis is the optical axis of the master camera in the first position of the sequence.

DIC is then used to match overlapping portions of the ROIs in image pairs from contiguous positions. Finally, the rigid transformation that overlaps with the minimum distance the corresponding points data in two contiguous point clouds is found through non-linear optimization. These transformations are used to move and merge the point clouds into a unique reference system, corresponding to the master camera reference frame. The merging error, defined as the average Euclidean distance between corresponding points from four contiguous views, is 0.26 ± 0.18 mm, that is only slightly larger than the reconstruction error. From the highly dense and regular set of points measured via DIC, a triangular mesh was automatically built via the Delaunay tessellation algorithm [114]. A plane was calculated for each triplet of points of the mesh, thus allowing to retrieve the distribution of the local normal vector over the whole reconstructed surface with the same spatial resolution of the DIC points grid reconstruction (about 2 mm spacing). Fig. 2.6 shows four different contiguous point clouds in their own reference frame, the 3D-reconstruction, in which the four point clouds are merged in the master camera reference frame, and the obtained tessellated mesh with superimposed the position of the hole centers computed via triangulation.

2.1.2 Hole detection

To create the detector, 175 images of size 640×480 and captured at different distances from the holes, have been annotated using the `LabelImg` tool. In Fig. 2.7 are shown some images used to build the detector. The training set was composed by 108 images and the remaining 67 were considered as test set. An Intel Xeon 3.7 GHz CPU 32 GB RAM with an Nvidia Quadro P4000 8GB GPU has been used to carry out the training

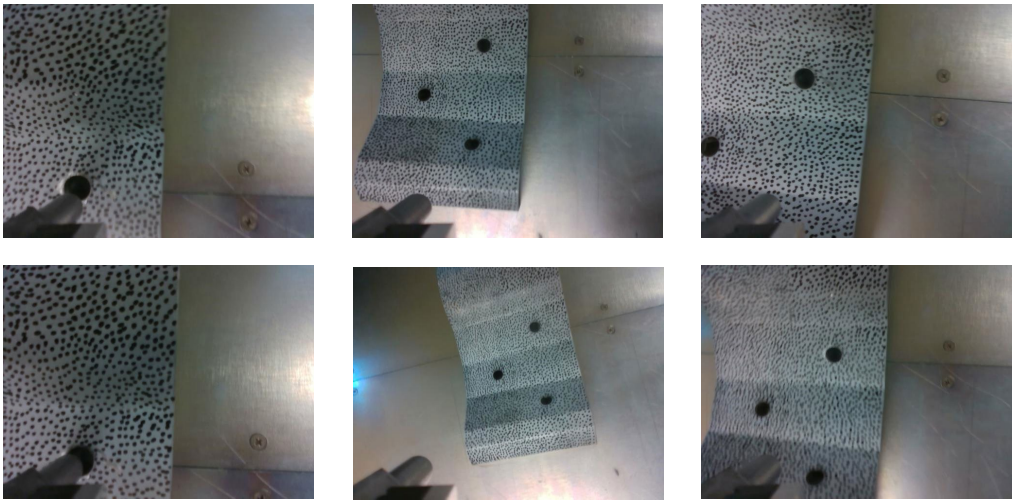


Figure 2.7: Samples of images used to build the hole detector.

phase, which required about 10 hours to complete with batch and subdivision sizes of 64 and 16, respectively, and with the YOLOv3 implementation. The batch size indicates the number of images processed before updating the network weights, while the subdivision size represents how many images can fit in the GPU at once. The processing time for detecting the holes on a single frame is about 27 ms, with an average IoU of 84.30%, with a threshold of 0.5.

Holes are detected into each image pair acquired by the camera, and then a binary mask is created to facilitate the greyscale thresholding and centroids computation. To compute the binary mask, the bounding box coordinates provided by the network are used to extract the pixels corresponding to holes in the images. In Fig. 2.8 an image with the detected holes and the relative binary mask is shown.

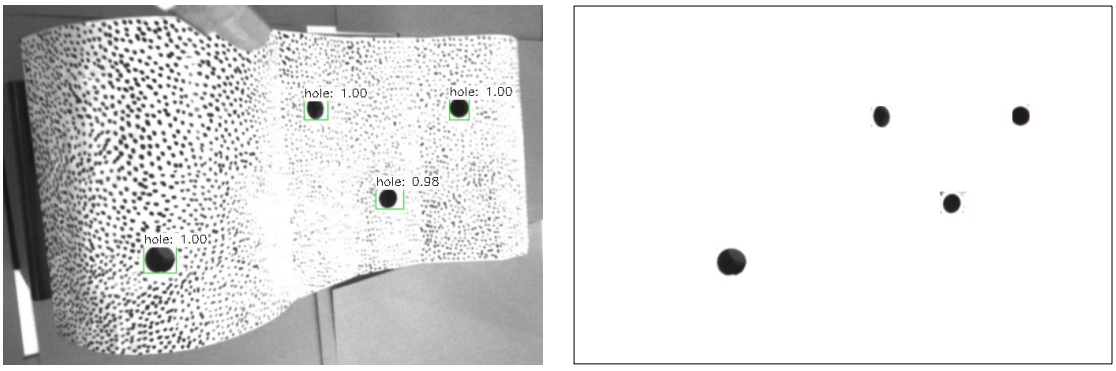


Figure 2.8: Image with the detected holes (a) and relative binary mask (b).

2.1.3 Approach to the hole

The above procedure allows to compute the positions, $\boldsymbol{p}_{h_i}^{c^\star}$ ($i = 1, \ldots, \beta$), of the hole centers, O_{h_i}, and their normal unit vector, $\boldsymbol{a}_{h_i}^{c^\star}$, in the reference frame $\Sigma_{c^\star}$ of the master camera. In order to perform the Peg-in-Hole task, such vectors needs to be transformed in the inertial frame by using the (4×4) homogeneous transformation matrices $\boldsymbol{A}_{p^\star}$, performing the transformation between the peg frame in the first position of the sequence, $\Sigma_{p^\star}$, and the inertial frame, and $\boldsymbol{A}_c^p$, performing the transformation between the camera frame, Σ_c, and the peg frame, Σ_p, obtained via the calibration method. Hence, the generic hole center position in the inertial frame can be computed as

$$\begin{bmatrix} \boldsymbol{p}_h \\ 1 \end{bmatrix} = \boldsymbol{A}_{p^\star} \boldsymbol{A}_c^p \begin{bmatrix} \boldsymbol{p}_h^{c^\star} \\ 1 \end{bmatrix}, \tag{2.3}$$

where the subscript i is omitted for readability. Similarly, the vector normal to the workpiece surface in O_h can be transformed in the inertial frame as

$$\boldsymbol{a}_h = \boldsymbol{R}_{p^\star} \boldsymbol{R}_c^p \, \boldsymbol{a}_h^{c^\star}, \tag{2.4}$$

where $\boldsymbol{R}_c^p$ ($\boldsymbol{R}_{p^\star}$) is the rotation matrix extracted by the homogeneous transformation matrix $\boldsymbol{A}_c^p$ ($\boldsymbol{A}_{p^\star}$).

In order to approach the hole, the robot motion is planned to move the origin of Σ_p in the neighborhood of the workpiece surface and, at the same time, to align the axis $\boldsymbol{a}_p$ to $\boldsymbol{a}_h$. To this aim, a closed-loop inverse kinematics algorithm with task priority [115] has been implemented, where two tasks are assigned: *peg alignment* (primary task with highest priority) and *position tracking* of O_p (secondary task with lower priority). The goal of the *peg alignment* task is to align the unit vector $\boldsymbol{a}_p$ to $\boldsymbol{a}_h$, thus the task function is

$$\sigma_1 = (\boldsymbol{a}_h - \boldsymbol{a}_p)^{\mathrm{T}} (\boldsymbol{a}_h - \boldsymbol{a}_p), \tag{2.5}$$

with corresponding Jacobian matrix

$$\boldsymbol{J}_1 = 2(\boldsymbol{a}_h - \boldsymbol{a}_p)^{\mathrm{T}} \boldsymbol{S}(\boldsymbol{a}_p) \boldsymbol{J}_O(\boldsymbol{q}) \in \mathbb{R}^{1 \times \nu}, \tag{2.6}$$

where $\boldsymbol{S}(\cdot)$ is the skew symmetric operator ([69], pp. 106-107) and $\boldsymbol{J}_O(\boldsymbol{q})$ is the orientation part of the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ defined in (1.11). The *position tracking* task is aimed at tracking a desired trajectory, $\boldsymbol{p}_{p_d}$, for the peg tip O_p, from its current position to a target point P_1, whose coordinate in the reference frame Σ_h are $\{0, 0, \Delta_1\}$, i.e, a point belonging to the axis $\boldsymbol{a}_h$ at a distance Δ_1 from O_h. The task function is the position of O_p, $\boldsymbol{p}_p$, and the

task Jacobian, $\boldsymbol{J}_2$, is the positional part of the Jacobian, $\boldsymbol{J}_P(\boldsymbol{q})$. In order to compute the commanded joint velocities, a Null-Space Behavioral control [115] is devised, in which the velocities of the secondary task are projected onto the null space of the primary task Jacobian, i.e.,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_1^{\dagger}(-k\sigma_1) + (\boldsymbol{I}_\nu - \boldsymbol{J}_1^{\dagger}\boldsymbol{J}_1)\boldsymbol{J}_2^{\dagger}(\dot{\boldsymbol{p}}_{p_d} + \boldsymbol{K}(\boldsymbol{p}_{p_d} - \boldsymbol{p}_p)), \tag{2.7}$$

where $\dot{\boldsymbol{p}}_{p_d}$ is the desired linear velocity of O_p, while k and $\boldsymbol{K} \in \mathbb{R}^{3\times3}$ are, respectively, a positive scalar and a positive definite matrix gain. The matrix $(\boldsymbol{I}_\nu - \boldsymbol{J}_1^{\dagger}\boldsymbol{J}_1)$ is a projector onto the null space of $\boldsymbol{J}_1$, where $\boldsymbol{I}_\nu$ is the $(\nu \times \nu)$ identity matrix.

2.1.4 Peg insertion

Once the approach phase has been completed, the desired pose $\boldsymbol{x}_{p,d} = [\boldsymbol{p}_{p,d}^{\mathrm{T}} \ \boldsymbol{\phi}_{p,d}^{\mathrm{T}}]^{\mathrm{T}}$ for O_p can be computed. In particular, the orientation is kept constant while the desired position trajectory is a fifth degree polynomial from the approach position to a target point P_2, whose coordinate in the Σ_h are $\{0, 0, -\Delta_2\}$, where Δ_2 includes the height of the cylindrical part of the peg (δ in Fig. 2.3), and a further translation along $-\boldsymbol{a}_h$ that favors the insertion.

In order to limit the mechanical stresses and ensure compliance to the peg, an admittance control has been implemented at the peg tip level.

As described in Section 1.6, the reference pose for Σ_p, $\boldsymbol{x}_{p,r} = [\boldsymbol{p}_{p,r}^{\mathrm{T}} \ \boldsymbol{\phi}_{p,r}^{\mathrm{T}}]^{\mathrm{T}}$, and the corresponding velocity, $\dot{\boldsymbol{x}}_{p,r}$, are computed via the admittance filter

$$\boldsymbol{M}_p\boldsymbol{\Delta}\ddot{\boldsymbol{x}}_p + \boldsymbol{D}_p\boldsymbol{\Delta}\dot{\boldsymbol{x}}_p + \boldsymbol{K}_p\boldsymbol{\Delta}\boldsymbol{x}_p = \boldsymbol{T}_A^{\mathrm{T}}(\boldsymbol{\phi}_p)\hat{\boldsymbol{h}}_p, \tag{2.8}$$

where $\boldsymbol{M}_p$, $\boldsymbol{D}_p$ and $\boldsymbol{K}_p$ are, respectively, the virtual inertia, damping and stiffness matrices imposed to the peg, $\boldsymbol{\Delta}\boldsymbol{x}_p = \boldsymbol{x}_{p,d} - \boldsymbol{x}_{p,r}$, $\hat{\boldsymbol{h}}_p$ is the estimate of the external wrench acting on the end-effector, and

$$\boldsymbol{T}_A^{\mathrm{T}}(\boldsymbol{\phi}_p) = \begin{bmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{O}_{3\times3} \\ \boldsymbol{O}_{3\times3} & \boldsymbol{T}(\boldsymbol{\phi}_p) \end{bmatrix}, \tag{2.9}$$

with $\boldsymbol{T}(\boldsymbol{\phi}_p)$ the matrix that maps the time derivative of the Euler angles $\boldsymbol{\phi}_p$, representing the peg orientation, to the angular velocity ([69], pp. 130-131). The external wrench $\hat{\boldsymbol{h}}_p$ is estimated by using the wrench observer (1.14) and the dynamic parameters identified in [116]. Such parameters have been suitably modified in order to take into consideration the contribution to the inertia and gravity terms of the gripper and the peg. Finally, the

Table 2.1: Controller and observer gains.

Gain	Value
k eq. (2.7)	1
$\boldsymbol{K}$ eq. (2.7)	$\mathrm{diag}[150, 150, 150]$
$\boldsymbol{K}_o$ eq. (1.14)	$\mathrm{diag}[10, 10, 10, 10, 15, 15, 15]$
$\boldsymbol{K}_p$ eq. (2.8)	$\mathrm{diag}[45, 45, 150, 0.15,\ 0.15,\ 0.15]$
$\boldsymbol{D}_p$ eq. (2.8)	$\mathrm{diag}[500, 500, 1000, 25, 25, 25]$
$\boldsymbol{M}_p$ eq. (2.8)	$\mathrm{diag}[10, 10, 10, 0.5, 0.5, 0.5]$
$\boldsymbol{\Lambda}$ eq. (2.10)	$\mathrm{diag}[150, 150, 150, 20, 20, 20]$

joint velocities are computed as

$$\dot{\boldsymbol{q}} = \left[\boldsymbol{T}_A(\boldsymbol{\phi}_p)\boldsymbol{J}\right]^{\dagger} (\dot{\boldsymbol{x}}_{p,r} + \boldsymbol{\Lambda}(\boldsymbol{x}_{p,r} - \boldsymbol{x}_p)), \tag{2.10}$$

where $\boldsymbol{\Lambda} \in \mathbb{R}^{6\times 6}$ is a positive definite gain matrix.

2.1.5 Experimental results

In order to perform the 3D surface reconstruction, 4 pairs of images have been taken by the stereo infrared imagers of the Intel Realsense camera and, then, holes detection is performed on each image.

Table 2.1 reports the inverse kinematics gains as well as the observer and admittance filter parameters. It is worth noticing that the virtual stiffness, $\boldsymbol{K}_p$, has been tuned in such a way the peg is more stiff along the z axis and more compliant along the other axes in order to simplify the insertion in the hole. For practical implementation of the control scheme, the following adjustments have been made:

- The estimated contact wrench $\hat{\boldsymbol{h}}_e$ output by the wrench observer (1.17) is filtered with a digital low-pass filter before sending it to the admittance filter.
- To suppress non-existent small force and torque estimations owing to unmodeled dynamics and sensor noise, a *dead zone* on the admittance filter input has been implemented: any value of force component below 3 N and any value of moment below 1 Nm estimated by the observer were neglected. Moreover, to achieve a continuous wrench signal, the same thresholds have been subtracted from higher estimations. This implies that the admittance filter receives as input a wrench slightly lower than the estimated one.

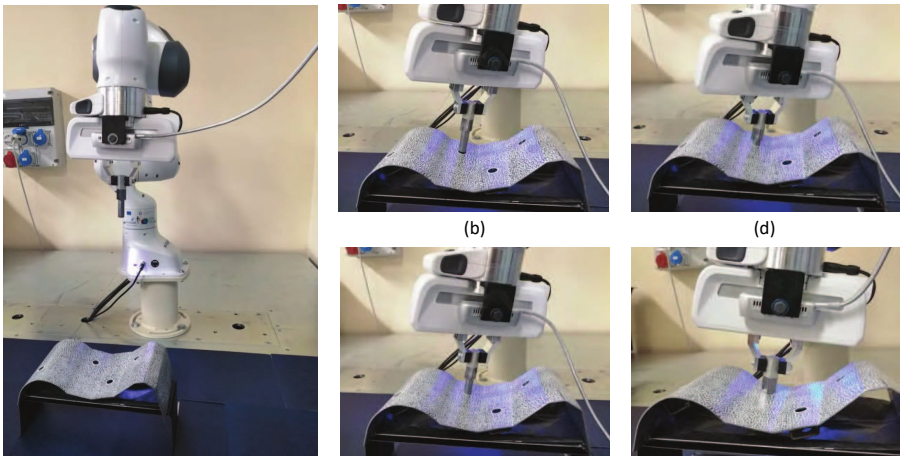
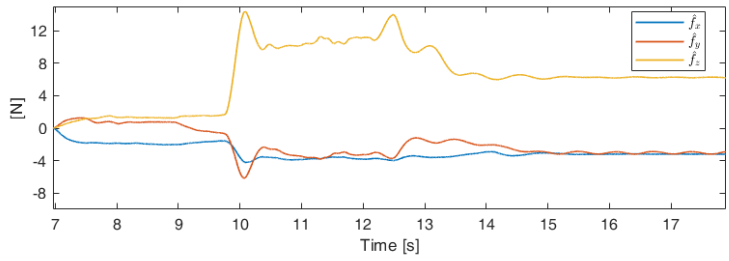
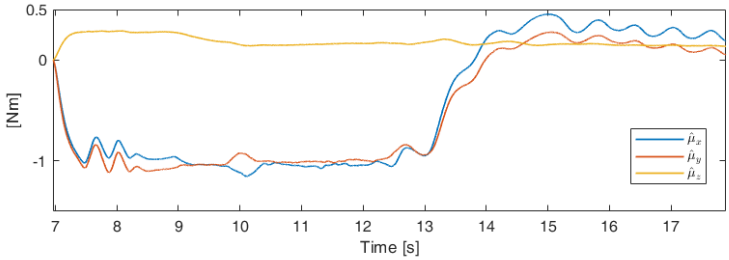


Figure 2.9: Snapshots of the Peg-in-Hole assembly task: (a) robot initial pose; (b) approach phase; (c) beginning of the insertion phase; (d) insertion phase at the maximum contact wrench; (e) end of the insertion.

Fig. 2.9 reports some snapshots taken during the experiment. Fig. 2.9(a) shows the random selected robot initial pose; in Fig. 2.9(b), the robot reaches the target point P_1 at the end of the approach phase; in Figs. 2.9(c) and 2.9(d), the poses corresponding, respectively, to the beginning of the insertion phase (at about 7 seconds) and the maximum

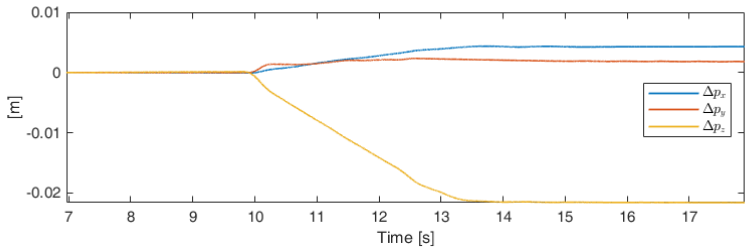


(a) Estimated external forces.

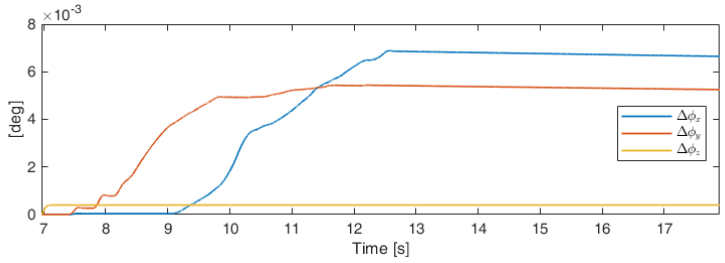


(b) Estimated external moments.

Figure 2.10: Estimated external wrench (forces (a) and moments (b)) exerted by the environment on the robot.



(a) Admittance control position error.



(b) Admittance control orientation error.

Figure 2.11: Error between the desired and reference pose: position (a) and orientation (b).

contact wrench are reported; finally, Fig. 2.9(e) shows the end of the insertion phase (at about 15 seconds).

Fig. 2.10 reports the filtered external forces (Fig. 2.10(a)) and moments (Fig. 2.10(b)) estimated by the observer during the insertion. The maximum force is experienced along the z axis, coherently with the assigned virtual stiffness and the tilt of the considered hole, and thus, non-negligible moments around the x and y axis arise. Such forces and moments cause a deviation of the trajectories output by the admittance filter with respect to the desired ones (Fig. 2.11).

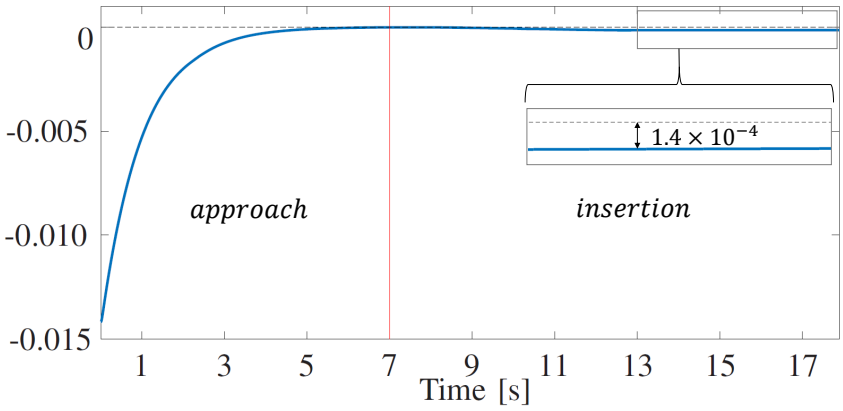


Figure 2.12: Alignment peg task error.

Finally, Fig. 2.12 shows the peg alignment task error: it can be noticed that the error converges to zero during the approach phase, then, during the insertion, when the task is not active, due to uncertainties on workpiece reconstruction and on camera calibration, the peg slightly modifies its tilt and the error grows up. The value of $1.4 \cdot 10^{-4}$ at the end of the insertion represents the squared norm error between the actual hole tilt and the estimated one.

As mentioned in Section 1.2.2, the 3D-DIC algorithm strongly depends on the spatial resolution of the captured images, the quality of the pattern on the surface and the illumination. For the application described in this Section, a quite low resolution sensor and an high quality pattern on the surface are used. Of course, in practice, it is not always possible to ensure the presence of a suitable pattern on the surface. For this reason, another strategy has been designed.

2.2 Second strategy

In the second strategy, the whole pipeline is quite similar to the previous one. More than steps, the changes concern the methods used to perform the steps. In particular, the method used to reconstruct the surface of the object, the method used to find the hole positions and their normal unit vectors and the method to perform the insertion are different.

This strategy is shown in Fig. 2.13 and it can be summarized as follows:

- The robot moves the eye-in-hand depth camera along a planned path spanning its workspace, in order to scan the surface of the target object. Along the trajectory, in a predefined set of positions, the depth measures are acquired and a point cloud of the surface is stored.
- The whole surface is reconstructed by aligning the acquired point clouds via an ICP registration algorithm.
- The reconstructed surface is aligned to a known CAD model of the workpiece, in such a way to detect the hole positions and their normal unit vectors on the reconstructed surface.
- Since the accuracy of the hole position estimates, in the presence of small peg-hole clearances, might not guarantee successful peg insertion, a search phase is performed, where the peg tip explores the neighborhood of the hole by sliding on

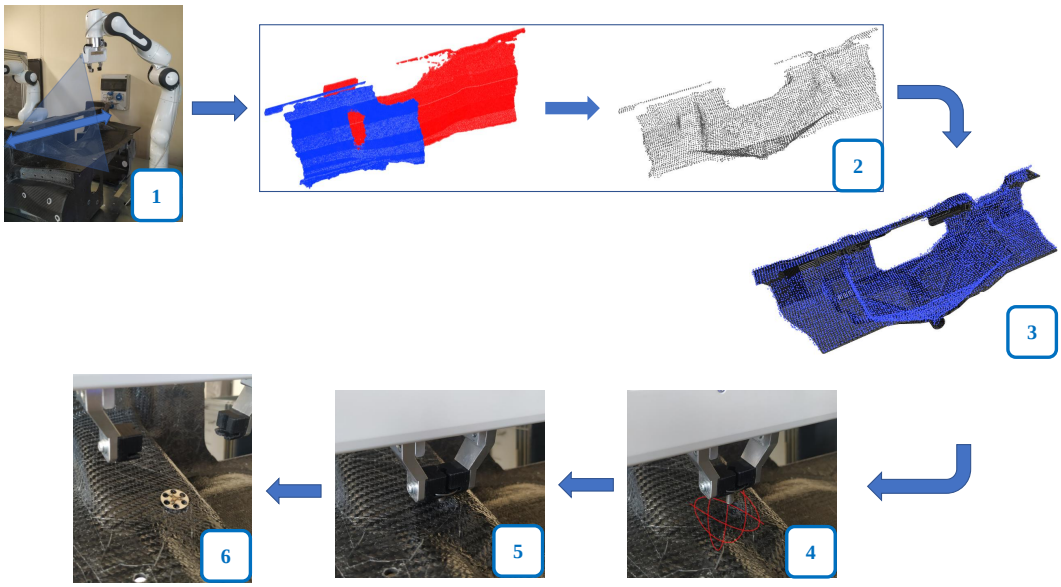


Figure 2.13: Proposed strategy. (1) workpiece surface scanning; (2) surface reconstruction; (3) alignment of the reconstructed surface with a known CAD model, in order to estimate the holes position; (4) search phase; (5) insertion phase; (6) release of the peg.

the surface along a trajectory described by Lissajous functions. During the search phase and the subsequent insertion phase, the robot is commanded to be compliant at the peg tip level by means of an admittance control.

For each execution cycle, including the insertion of N pegs, the specific process sequence is detailed in the sequence diagram shown in Fig. 2.14.

In this case, the camera is used to provide the depth information, while the robot scans the workpiece surface. The robot control unit roughly knows the geometry of the workpiece (from the CAD model) and its pose with respect to the robot base frame. On the basis of the available information on the surface geometry and pose, as well as on the camera features (range, field of view), n measurement points, each corresponding to an end-effector pose, are determined such that the whole surface is spanned. The choice of n is determined in such a way to ensure an overlapping of each two consecutive views: the more wide is the camera field of view, the less are the measurement points needed. Then, the robot visits the n measurement poses and, in each pose, a point cloud is built on the basis of the acquired depth measures.

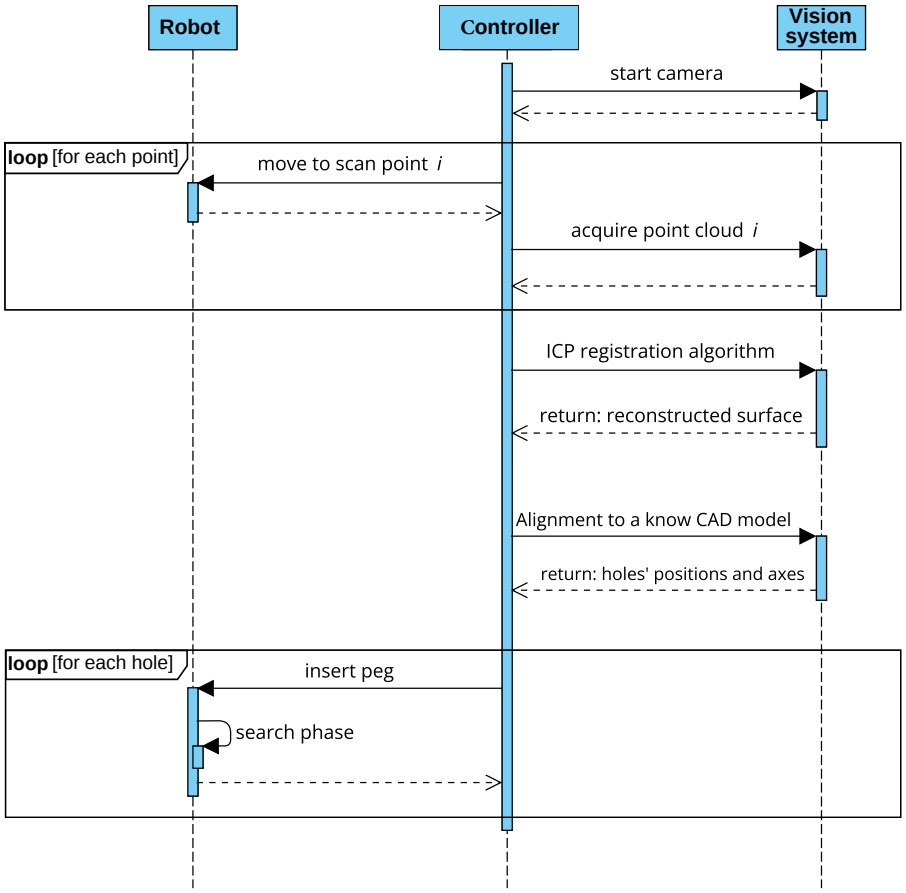


Figure 2.14: Sequence diagram of the execution process.

2.2.1 3D surface reconstruction

In order to reconstruct the workpiece surface, the acquired point clouds are aligned by using the ICP registration algorithm described in Section 1.3, and the result is down-sampled by using a volume elements (voxel) grid filter [70]. The output of this process is a point cloud, $\mathcal{P}_r$, representing a reconstruction of the surface of the whole workpiece in the coordinate frame $\mathcal{F}_c^1$ (see Section 1.3). An example of reconstructed 3D surface is reported in Fig. 2.15.

2.2.2 Holes localization

The next step is the comparison of the obtained point cloud with a known one, $\mathcal{P}$, obtained, via the CAD model. To this aim, for each point of $\mathcal{P}_r$, a vector of local features, called Fast Point Feature Histograms (FPFH), are computed [117]. The process to compute the FPFH can be divided into two steps. In the first step, for each point $\boldsymbol{\pi}_s \in \mathcal{P}_r$,

Figure 2.15: Example of reconstructed 3D surface.

the Simplified Point Feature Histogram (SPFH) is computed. This histogram contains informative pose-invariant local features which represent the surface model properties at the point $\boldsymbol{\pi}_s$. For every pair of points, $\boldsymbol{\pi}_s$ and $\boldsymbol{\pi}_t$ ($s \neq t$), in the k-neighborhood of $\boldsymbol{\pi}_s$ and their estimated normal vectors, $\boldsymbol{n}_s$ and $\boldsymbol{n}_t$, a uvw coordinate frame at one of the point is defined as:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{n}_s, \\ \boldsymbol{v} &= \boldsymbol{u} \times \frac{(\boldsymbol{\pi}_t - \boldsymbol{\pi}_s)}{\|\boldsymbol{\pi}_t - \boldsymbol{\pi}_s\|_2}, \\ \boldsymbol{w} &= \boldsymbol{u} \times \boldsymbol{v}, \end{aligned}$$

A set of triples α, ψ, θ, representing the angular variations of n_s and n_t, are computed as follows:

$$\begin{aligned} \alpha &= \boldsymbol{v} \cdot \boldsymbol{n}_t, \\ \psi &= \boldsymbol{u} \cdot \frac{(\boldsymbol{\pi}_t - \boldsymbol{\pi}_s)}{\|\boldsymbol{\pi}_t - \boldsymbol{\pi}_s\|_2}, \\ \theta &= \arctan(\boldsymbol{w} \cdot \boldsymbol{n}_t, \boldsymbol{u} \cdot \boldsymbol{n}_t), \end{aligned}$$

In [118], besides the three features mentioned above, a fourth one was included in SPFH. This feature represents the Euclidean distance from $\boldsymbol{\pi}_s$ and $\boldsymbol{\pi}_t$, but some experiments showed that its exclusion from the SPFH doesn't decrease robustness [117].

In the second step to compute the FPFH, for each point $\boldsymbol{\pi}_s$, its k neighbors are determined and the neighboring SPFH values are used to compute the final histogram (FPFH)

of $\boldsymbol{\pi}_s$ ad follows:

$$FPFH(\boldsymbol{\pi}_s) = SPFH(\boldsymbol{\pi}_s) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_i}\cdot SPFH(\boldsymbol{\pi}_t)$$

where the weight ω_i represents the distance between the point $\boldsymbol{\pi}_s$ and a neighbor point $\boldsymbol{\pi}_t$. For each point $\boldsymbol{\pi}_s$, the result is a vector that describes the local geometric properties of $\boldsymbol{\pi}_s$.

Then a RANSAC (RANdom SAmple Consensus) algorithm [119] is adopted to find the corresponding points of the two point clouds. At each iteration, μ points are randomly extracted from $\mathcal{P}_r$; then, the corresponding points of $\mathcal{P}$ are determined by looking for the nearest points on the basis of the extracted features. False matchings, i.e., connections between two points that do not actually correspond in the point clouds, can be deleted by resorting to pruning algorithms [120].

In particular, two types of pruning algorithms are used. The first one checks that the point clouds are close in terms of distance, i.e., if the distance between the point cloud, after overlapping them, is greater than a certain threshold, the matching is discarded. The second type of pruning technique used in the algorithm checks that the length of the line connecting two arbitrary points in the first point cloud is similar to the length of the line connecting the corresponding points in the second one. Only matches that pass the pruning step are used to compute a transformation, which is validated on the entire point cloud.

The transformation matrix between $\mathcal{P}_r$ and $\mathcal{P}$ obtained via the RANSAC algorithm could be not very accurate, thus it is adopted as initial guess of the ICP algorithm that is in charge of refining the alignment. An example of surface-CAD alignment is shown in Fig. 2.16.

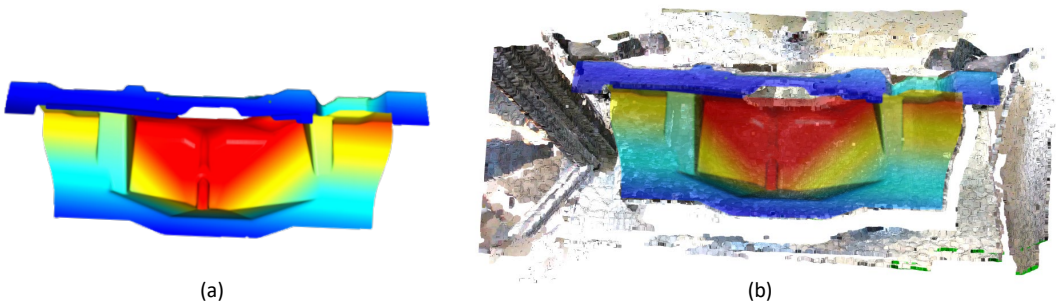


Figure 2.16: Holes localization. (a) The point cloud extracted from the CAD model. (b) Alignment of the reconstructed surface and the CAD model.

Algorithm 1: Holes localization

Input : $\mathcal{P}_r$, $\mathcal{P}$, $\boldsymbol{p}_{h_i}$, $\boldsymbol{n}_{h_i}$

Output: Estimated position of the holes center $\hat{\boldsymbol{p}}_{h_i}$ and their axis $\hat{\boldsymbol{n}}_{h_i}$

for each $\boldsymbol{\pi}^1_{r,i} \in \mathcal{P}_r$ **do**
    Extract FPFH vector
end for
while $i < max\_iterations$ **do**
    Randomly extract μ points from $\mathcal{P}_r$ and find the μ corresponding points in $\mathcal{P}$ via RANSAC algorithm, as the nearest point on the bases of the FPFH features
    Compute the transformation matrix $\boldsymbol{T}^i_R$ between $\mathcal{P}_r$ and $\mathcal{P}$
    if `EvaluationError`$(\boldsymbol{T}^i_R) <$ `EvaluationError`$(\boldsymbol{T}_0)$ **then**
        $\boldsymbol{T}_0 \leftarrow \boldsymbol{T}^i_R$
    end if
    $i++$
end while
$\boldsymbol{T} \leftarrow$ `ICP_algorithm`$(\mathcal{P}, \mathcal{P}_r, \boldsymbol{T}_0)$
for $i \leftarrow 0$ **to** $holes\_number - 1$ **do**
    $\hat{\tilde{\boldsymbol{p}}}_{h_i} \leftarrow \boldsymbol{T}\tilde{\boldsymbol{p}}_{h_i}$
    $\hat{\tilde{\boldsymbol{n}}}_{h_i} \leftarrow \boldsymbol{T}\tilde{\boldsymbol{n}}_{h_i}$
end for
return $\hat{\boldsymbol{p}}_{h_i}$, $\hat{\boldsymbol{n}}_{h_i}$

The position of each hole center $\boldsymbol{p}_{h_i}$ and its corresponding axis, i.e. the unit vector normal to the surface, $\boldsymbol{n}_{h_i}$, are known on the point cloud $\mathcal{P}$. Thus, thanks to the above described alignment procedure, it is possible to estimate them in the reconstructed point cloud $\mathcal{P}_r$ and localize them in the coordinate frame $\mathcal{F}^1_c$. The previous steps are summarized in Algorithm 1. Finally, it is possible to localize the holes in the robot base coordinate frame by using the camera-end-effector transformation obtained via a camera calibration process [121].

2.2.3 Approach to the hole

Once the estimates of the hole positions, $\hat{\boldsymbol{p}}_{h_i}$, and the corresponding axes, $\hat{\boldsymbol{n}}_{h_i}$, are determined, the Peg-in-Hole task is performed in three phases:

1. *Approach*: in this phase the robot moves the peg close to the hole and aligns the peg axis to the estimated hole axis, $\hat{\boldsymbol{n}}_{h_i}$.

2. *Search*: in this phase the neighborhoods of the estimated hole position, $\hat{\boldsymbol{p}}_{h_i}$, are explored in such a way to compensate errors in the position estimate.

3. *Insertion*: in this phase the peg is inserted in the hole.

In all phases the robot has been controlled with the admittance scheme described in Section 1.6.

Also in this case, the robot motion to approach the hole is commanded via a closed-loop inverse kinematics algorithm with two tasks: the first one is aimed at aligning the peg to the hole axis, while the second task is aimed at moving the peg close to the surface.

The alignment task and the corresponding Jacobian matrix are the same expressed in (2.5) and in (2.6), and the joint references are computed using (2.7).

2.2.4 Search phase

The search phase starts when a contact along the direction normal to the reconstructed surface is detected. Such a contact is detected when the projection of the estimated contact force defined in (1.17) on $\hat{\boldsymbol{n}}_{h_i}$ exceeds a certain threshold.

The search is performed by sliding the peg tip on the workpiece surface, by following a path in the xy-plane of $\mathcal{F}_e$ described by Lissajous functions

$$\begin{aligned} x^e &= a_x \sin(\omega_x(t - t_c)), \\ y^e &= a_y \sin(\omega_y(t - t_c)), \end{aligned} \tag{2.11}$$

where a_x, a_y, ω_x and ω_y are the sine wave amplitudes and frequencies, respectively, and t_c is the time instant when the peg tip comes in contact with the surface. The superscript e denotes that the trajectory is expressed in the frame $\mathcal{F}_e$.

A compliant behaviour at the peg tip level is imposed to the robot, through the admittance control described in Section 1.6, in order to reduce mechanical stresses both on the workpiece and on the robot's joints and to allow the sliding of the peg. Table 2.2 reports the gain matrices used in this case. In order to compute the joint references for the

Table 2.2: Controller and observer gains.

Gain	Value
k eq. (2.7)	5
$\boldsymbol{K}$ eq. (2.7)	$\mathrm{diag}[150, 150, 150]$
$\boldsymbol{K}_p$ eq. (2.8)	$\mathrm{diag}[45, 45, 150, 1.5,\ 1.5,\ 1.5]$
$\boldsymbol{D}_p$ eq. (2.8)	$\mathrm{diag}[30, 30, 30, 1, 1, 1]$
$\boldsymbol{M}_p$ eq. (2.8)	$\mathrm{diag}[15, 15, 15, 0.5, 0.5, 0.5]$
$\boldsymbol{\Lambda}$ eq. (2.10)	$\mathrm{diag}[150, 150, 150, 30, 30, 30]$

robot from the operational-space references output by the admittance filter, the inverse kinematics algorithm (2.10) is used.

2.2.5 Peg insertion

During the search phase, the planned trajectory has been chosen some millimeters under the hole surface, in order to allow a coupling force along the z axis of Σ_e. When the peg tip detects the hole, the peg goes downward and the search phase is stopped. From this peg's position a new trajectory along the estimated hole's axis is planned, in order to perform the insertion. When this trajectory is concluded the robot manipulator releases the peg. During the insertion the admittance control scheme in Section 1.6 has been used with the gain matrices reported in Table 2.2.

It is worth noticing that the compliance conferred by the admittance scheme is in charge of compensating small orientation errors between the peg and the normal of the hole. Such errors can cause resistant torques that, through (2.8), provide a new orientation for the peg tip.

The joint references for the robot are computed from the operational-space references output by the admittance filter via the inverse kinematics algorithm (2.10).

2.2.6 Experimental results

The surface reconstruction has been performed, according to the procedure described in Section 1.3, by using the `multiway registration` algorithm provided by the `Open3D` library [120]. The reconstructed surface is overlapped and compared with the point cloud extracted by the CAD model, according to the procedure described in Section 2.2.2, by

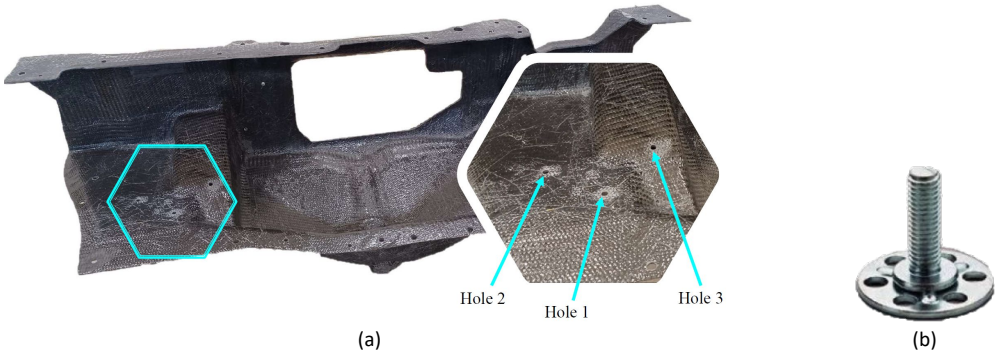


Figure 2.17: Workpiece and holes (a). Big Heads (b).

using the `global registration` algorithm provided by the `Open3D` library [120]. The camera system has been calibrated by using 16 images of a 2D checkerboard flat pattern according to the eye-in-hand camera calibration method proposed in [121], implemented in the ViSP library [122]. A carbon fiber 3D workpiece has been used, representing a portion of a supercar's safety cell, while steel bolts, named *Big Heads*, have to be inserted in 3 holes (Fig. 2.17). A very small clearance, below 1 mm, is present between the hole and the Big Heads; moreover, the task is made more challenging by the fact that the peg is threaded while the holes are very rough.

In order to have statistically significant results, 35 tests have been carried out in different light conditions and by randomly positioning the object in the robot workspace, so as to mimic the actual work cycle, where the workpiece is manually positioned and errors up to about 10-15 cm and 15-20 deg can be experienced with respect to the planned nominal pose.

In the first phase, the robot scans the 3D surface moving along a fixed path, planned by means of a linear interpolation between the first and last camera poses. Such poses, in terms of position and orientation of the camera, have been computed in such a way to have the nominal position of the workpiece in the camera field of view, with a certain tolerance to take into account the errors due to the manual workpiece positioning.

During the scanning phase, for each test, $n = 8$ different point clouds have been acquired via the depth sensor to reconstruct the workpiece surface. The number of point clouds is chosen in order to have a wide overlap between two consecutive ones. An example of reconstructed surface is shown in Fig. 2.15.

In order to evaluate the quality of the reconstruction, the re-projection error between the 3D reconstructed surface and the CAD model of the workpiece has been computed.

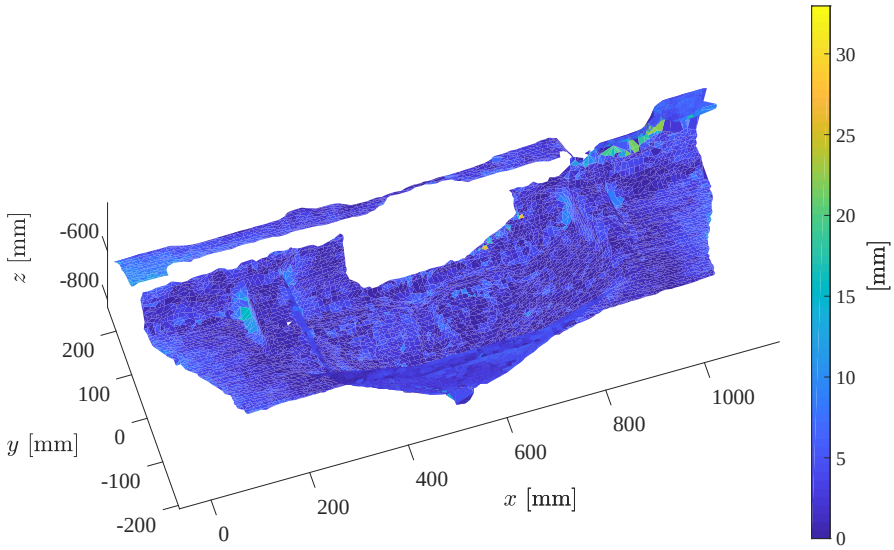


Figure 2.18: Root mean square error of the reconstructed point cloud with respect to the CAD model.

More in detail, the acquired points have been projected onto the CAD surface using Rhinoceros 3D software [123] and then the distance between the acquired and projected points has been computed. In Fig. 2.18, the root mean square error for a single test is reported, the average error is around 5 mm and it is quite homogeneous along the whole surface. The mean square error computed by considering all the 35 tests is about 6 mm. It is worth noticing that different reconstruction errors have been experienced due to both the different positioning and the different light conditions of the experiments.

Regarding the performance of the task execution, the following indices have been considered: the duration of the search phase and the error between the estimated and actual hole position. In order to compute the duration of the search phase, the initial time is the instant, t_c, when the first contact between the peg tip and the surface arises, while the search ends when the peg tip is inserted of about 1 mm in the hole. The duration of the search phase evaluated for each hole in the 35 tests is reported in Fig. 2.19, from which it can be seen that the average search time is around 5 seconds.

In test 27, for hole 3, the search phase has not been executed, since the bolt has been directly inserted in the hole.

Some snapshots of the search phase are reported in Fig. 2.20: Fig. 2.20(a) shows the bolt approaching the surface with a certain error, Fig. 2.20(b) shows the bolt sliding on the surface, Fig. 2.20(c) shows the phase in which the hole is detected and the insertion

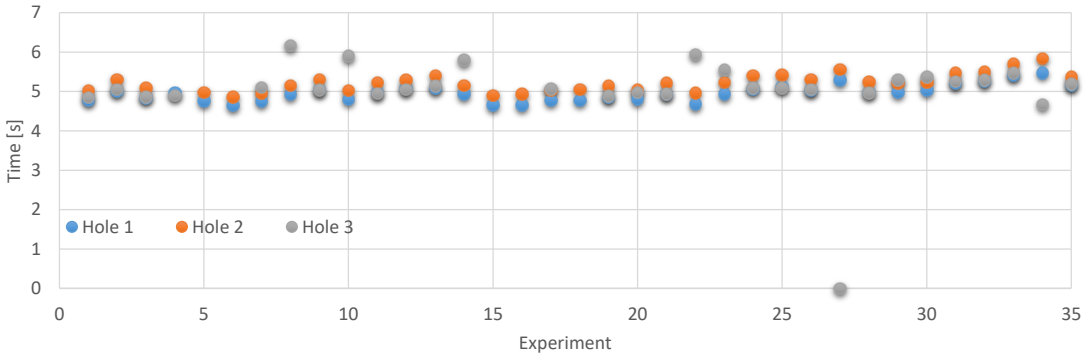


Figure 2.19: Search phase duration.

is performed. The insertion is considered completed when the robot pushes the peg into the hole for additional 7 mm.

In Fig. 2.21 the holes position errors computed in the 35 tests are reported. More in detail, the bottom side of the grey box is the minimum error obtained for the three holes in each test, while the upper side is the average one. The upper side of the orange box represents the maximum error obtained for the three holes in each test. For each hole, the position error is evaluated by considering the distance, in the plane of the hole neighborhood, between the reconstructed and the actual hole position. The average error is about 4 mm, the minimum error has been registered for test 27, equal to zero. The worst performance is experienced at test 29, and depends on an imperfect reconstruction that leads to an error of 16.5 mm for the third hole.

Table 2.3 reports the numerical data registered in the experiments, the first three

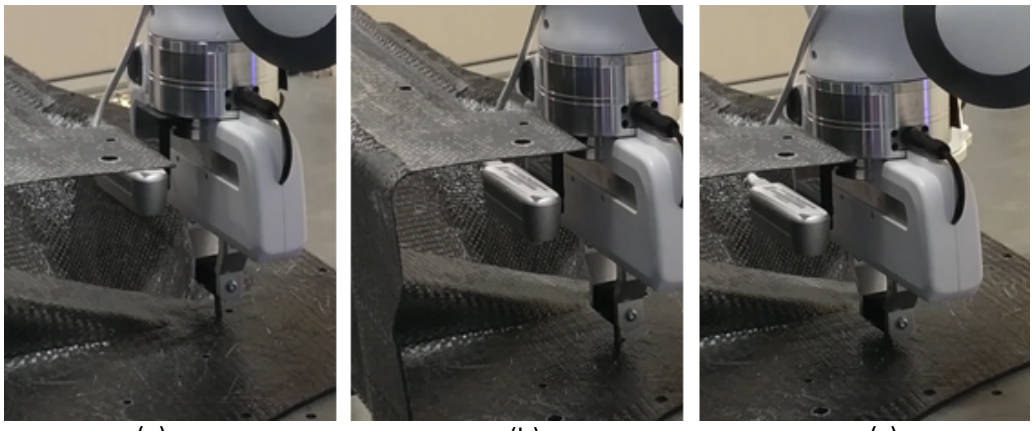


Figure 2.20: Snapshots of the search phase.

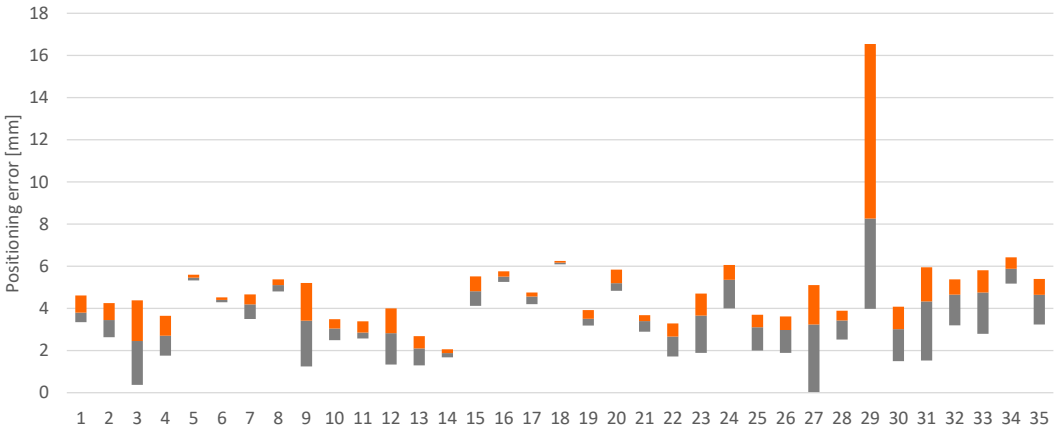


Figure 2.21: Error between the reconstructed and the actual holes' position.

columns report the search phase duration and the last three contain the position error measured for each hole. On the total of 105 insertions only 6 failed, obtaining an overall insertion success rate of 94.2%.

A set of additional experiments has been carried out to assess the performance of the approach in the presence of non-negligible surface irregularities, by adding artificial protrusions in the neighborhood of the second hole, as shown in Fig. 2.22. Also in this case, the task has been correctly executed, since the admittance control confers a suitable compliant behaviour to the robot, which is able to adapt to the surface profile if the protrusion is not characterized by too sharp edges.

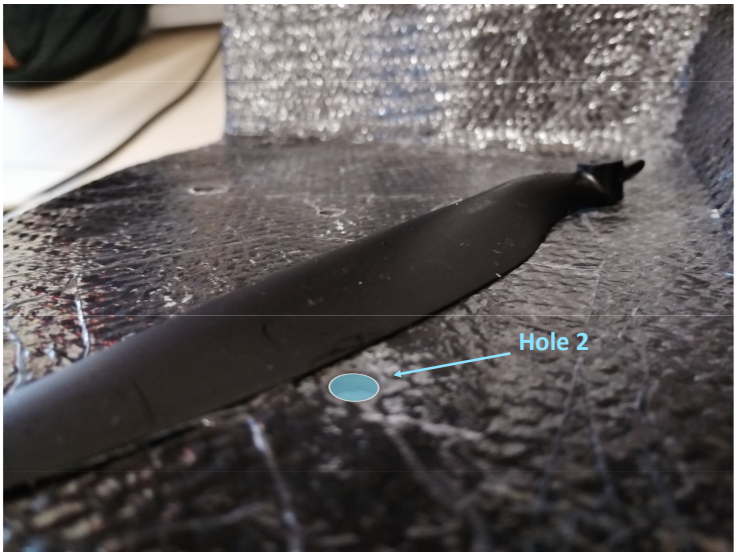


Figure 2.22: Artificially generated protrusion by using a drone propeller.

The whole cycle from scanning to release of the first peg has an average duration, computed over the 35 tests, of about 2 minutes and 17 seconds. More in detail, about 2 minutes are needed for the scanning phase, executed once at the process start, including the storage of the point clouds and the reconstruction computation. This duration is mainly due to the need to starting the communications with the camera and save the point clouds, that include a total of 921600 points. The average time needed for point cloud registrations, surface reconstruction and comparison with the point cloud extracted from the CAD model is about 14 seconds.

In the remaining 17 seconds the peg is moved in contact with the surface and the search phase (which has an average duration of 5 seconds) is performed.

Table 2.3: Test results. The empty cell represents the failed insertions.

Test	Search duration [s]			Position error [mm]		
	Hole 1	Hole 2	Hole3	Hole 1	Hole 2	Hole3
1	4.772	5.028	4.85	3.3526	3.4395	4.6157
2	5.016	5.307	5.056	4.2496	3.4747	2.6349
3	4.841	5.086	4.861	4.3832	0.37241	2.5932
4	4.97	–	4.905	3.6456	–	1.7596
5	4.755	4.973	–	5.3298	5.5974	–
6	4.642	4.863	–	4.2966	4.5197	–
7	4.746	4.946	5.103	3.5001	4.411	4.6674
8	4.927	5.15	6.171	5.377	4.8082	5.1504
9	5.044	5.309	5.038	3.8065	5.2065	1.2463
10	4.797	5.023	5.903	3.1594	2.4921	3.4863
11	4.973	5.227	4.955	2.5786	3.3899	2.599
12	5.082	5.29	5.06	3.1414	4.0013	1.3339
13	5.082	5.404	5.141	2.3173	2.6899	1.3026
14	4.93	5.154	5.796	1.8998	2.058	1.6834
15	4.655	4.906	–	4.1183	5.5196	–
16	4.653	4.937	–	5.2602	5.7614	–
17	4.77	5.03	5.066	4.7524	4.7199	4.2072
18	4.776	5.051	–	6.0915	6.2506	–
19	4.843	5.139	4.885	3.4184	3.1839	3.9258
20	4.805	5.04	4.982	4.8406	4.9056	5.8433
21	4.936	5.208	4.942	3.6834	3.624	2.8966
22	4.682	4.962	5.94	1.7234	2.9898	3.284
23	4.939	5.227	5.545	4.398	4.7071	1.8894
24	5.067	5.401	5.096	6.0223	6.0605	4.0043
25	5.125	5.415	5.098	3.7016	2.0056	3.617
26	5.037	5.308	5.048	3.4188	3.623	1.892
27	5.294	5.558	0	4.608	5.1097	0
28	4.979	5.246	4.967	3.8716	3.8885	2.5253
29	4.981	5.22	5.285	3.9818	4.2679	16.544
30	5.031	5.233	5.372	3.4597	4.0809	1.499
31	5.201	5.469	5.248	5.5154	5.953	1.5262
32	5.268	5.503	5.3	5.377	5.3839	3.1954
33	5.403	5.705	5.469	5.6531	5.8107	2.7984
34	5.472	5.833	4.671	6.0433	6.4256	5.17683
35	5.163	5.372	5.19	5.3041	5.399	3.2412

2.3 Second strategy: improvement

As described in the previous Section, the total error is determined by calibration errors, surface reconstruction errors, errors in the overlapping process with the CAD and robot positioning errors. These errors are not very large if they are considered separately, but their combination in a complex process can reach values that do not guarantee the peg insertion, especially in the presence of small clearance.

In this Section, an improvement for the method in Section 2.2 is presented. The whole pipeline is depicted in Fig. 2.23. It is quite similar to the previous one except for two steps:

- the point clouds acquired in the surface scanning phase are segmented by using a neural network that detects the workpiece and deletes its background (Fig. 2.23(2)). After that, they are merged to reconstruct the surface with the same process described in Section 2.2;
- the estimates of the hole position provided by the overlapping process are used as initial guess for positioning the robot and, then, a CNN is adopted to detect the presence of the hole and identify its actual position in the robot base frame to perform the insertion (Fig. 2.23(4)).

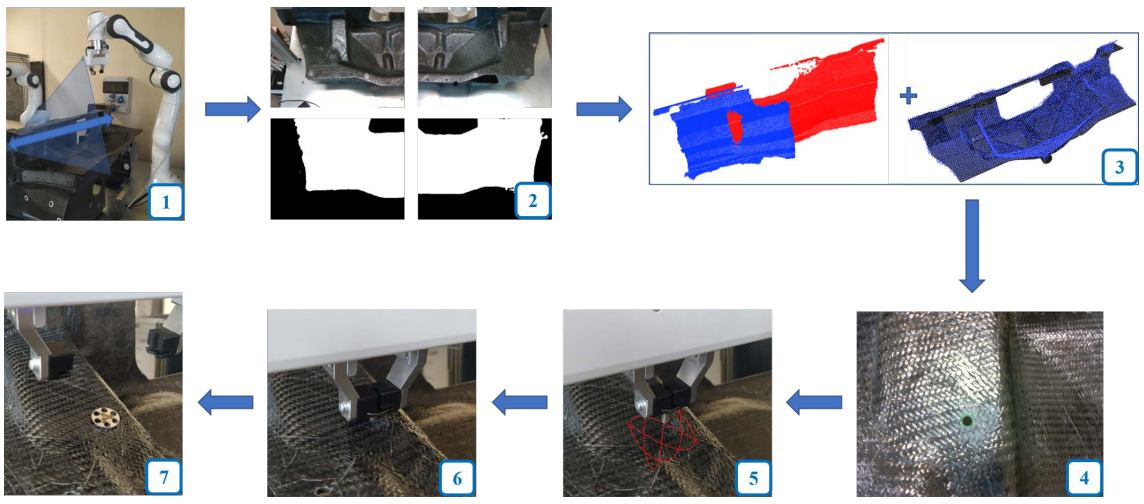


Figure 2.23: Strategy summary: (1) workpiece surface scanning; (2) the segmentation neural network detects the workpiece and deletes the background; (3) surface reconstruction and alignment of the reconstructed surface with the point cloud extracted from the CAD to have the initial guess estimation of the holes' position; (4) hole detection via the CNN; (5) search and insertion phase.

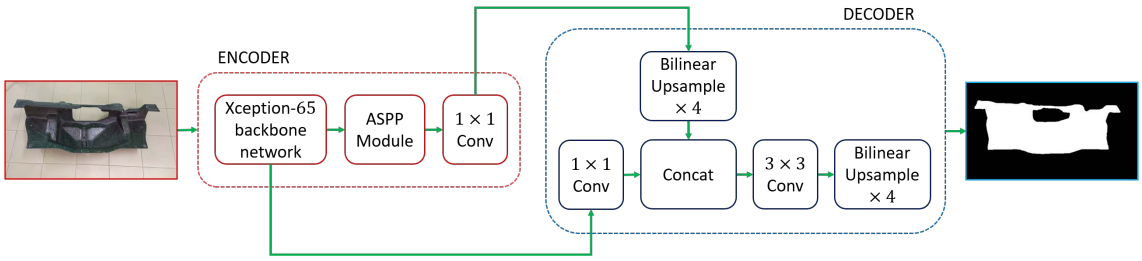


Figure 2.24: Network model based on the DeepLabv3+ encoder-decoder architecture. The input is an RGB image showing the carbon fiber workpiece, while the output is the segmentation binary mask.

In the rest of the Section, these two steps are discussed in detail.

2.3.1 Workpiece segmentation

A DNN, based on the DeepLabv3+ architecture, has been designed for the carbon fiber workpiece segmentation. As can be seen in Fig. 2.24, DeepLabv3+ consists of two main components. The first one is the encoder block, which extracts semantic information and low-level features from the RGB input image, gradually reducing the feature maps size. The second component is the decoder block, which is used to retrieve spatial and detailed object boundary information.

The encoder includes a backbone network, followed by an ASPP module [102] and a 1×1 convolutional layer. The ASPP module captures multi-scale context information and consists of three atrous convolutions [124], a 1×1 convolution and an image pooling layer in parallel with each other. Atrous (or dilated) convolutions extend standard convolutions introducing a atrous (or dilation) rate parameter to enlarge the field of view of the convolutional filters without increasing the computational cost and the network parameters [125]. The atrous rate of the atrous convolutions in the ASPP module have been set to 6, 12 and 18, respectively. The the Xception network with 65 layers adapted to the segmentation task in [126] has been chosen as the backbone network, which allows extracting low-level features that are passed to the decoder.

The decoder block is built using convolutional and bilinear upsampling operations. In particular, the features extracted by the backbone network are given as input to a 1×1 convolution and then concatenated with the upsampled encoder output. Finally, a 3×3 convolution and a further bilinear upsampling are applied and a binary segmentation mask is obtained.

2.3.2 Segmentation network: dataset description

The main drawback of using deep learning is the need for a huge training dataset with ground-truth data (or labels). Since obtaining accurate labels from real images is time-consuming and sometimes impractical, synthetic and semi-synthetic data have been widely collected in recent years [127, 128].

Synthetic data are completely artificial samples produced using simulation software and computer graphics techniques, obtaining perfect labels quickly and with little effort. However, a pre-processing step is usually required to bring realism to synthetic data.

Semi-synthetic data combines real and computer-generated information, better approximating the real data. Therefore, a large and varied semi-synthetic dataset has been collected to train the DNN. The resulting dataset consists of composite images built integrating real foreground images, showing the carbon fiber workpiece in multiple positions, and different background images.

In particular, the first step in the dataset creation concerned the acquisition of the foreground information. Several videos using a green-screen setup has been captured. This setup consisted of an opaque green drape and two lights, one illuminating the object and the other the green background to remove as many shadows as possible. The videos were recorded at 60 frames per seconds (FPS) using a common RGB camera with 1920×1080 resolution. The chosen frame rate, instead of the most common 30 value, allows to reduce the blur effect on the object caused by camera movement. Then the green background has been removed and the foreground images have been extracted. In addition, for each frame, an alpha channel (or alpha matte) mask is generated. Such a mask presents values in the range $[0, 255]$, where 0 is pure background, 255 indicates pure foreground pixels, and the values in between represent the transition region. To obtain accurate ground truth binary masks with 0 values in the case of background and 1 for foreground pixels, each alpha matte has been binarized by using the Otsu's global image thresholding method [129].

The second step in the dataset creation was the collection of background scenes and image compositing. In detail, several indoor videos have been recording at 30 FPS using two RGB cameras with 1920×1080 and 3140×2160 resolutions, respectively. To generate the composite images, the following equation has been used:

$$C_i = \alpha_i F_i + (1 - \alpha_i) B_i, \tag{2.12}$$

where C_i, F_i and B_i represent the pixels of the obtained composite image, the foreground

image and the new background, respectively and $\alpha_i \in [0, 1]$ is the alpha channel, that represents the degree of transparency (or opacity) of a color. An automatic procedure that takes four foreground images per second and two background frames per second from each video has been designed. The definition of a sampling step equal to 15 avoids considering too similar frames. Then, each selected foreground was composited with all background scenes.

To further increase the variety of data and allow a better network generalization level, before compositing, each F was randomly transformed by:

1. rotating it with a random angle between $[-30, 30]$ degrees and using bicubic interpolation and cropping to fit its original dimensions;
2. horizontal flipping;
3. vertical flipping.

The same transformations were simultaneously applied to the respective labels and alpha channels. Moreover, a bilateral filter was randomly applied to C. This is a nonlinear filter often used for noise reduction and composite image smoothing, while preserving edges of foreground objects [128, 130]. Each transformation was applied with a probability of 0.5. Finally, all images and labels were resized to 360×640 pixels to speed up the network training and data augmentation was used by randomly left/right mirroring training data on the fly during training. Some examples of training images and the corresponding labels are shown in Fig. 2.25.

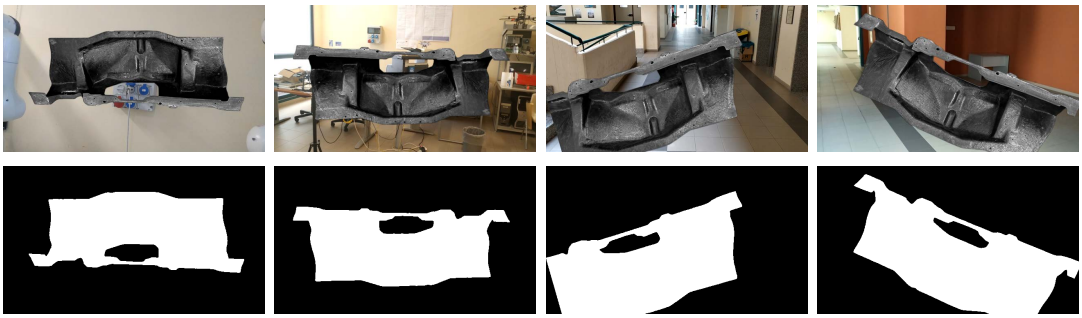


Figure 2.25: Some input images (first row) and labels (second row) from the semi-synthetic dataset.

2.3.3 Segmentation network: training details

The DNN was trained using the above described segmentation dataset. Similar to [131] and [93], the SGDM (Stochastic Gradient Descent with Momentum) optimization algorithm with polynomial learning rate policy has been used. This algorithm has been proven to be more effective and with faster convergence than other learning rate update policies [102, 132]. In particular, the value of the learning rate is modified according to the following formula:

$$\alpha_t = \alpha_0 \times \left(1 - \frac{t}{T}\right)^p, \tag{2.13}$$

where α_t is the learning rate at the current iteration step t, α_0 is the base learning rate set to 0.0001 for the training phase, T is the total number of iterations set to $5 \cdot 10^4$, and p is the power value set to 0.9. Moreover, the momentum γ of the SGDM algorithm has been set to 0.9 and the batch size to 4.

Since training a DNN from scratch requires a copious amount of data and resources in terms of memory, computation and time, starting from a pre-trained models on a large dataset is usually recommended. Therefore, weights pre-trained on the ImageNet [133] and MS-COCO [134] datasets have been used. The pre-trained weights are publicly available on the DeepLab project page [135]. ImageNet is a huge and generic dataset employed for classifying and detecting $1,000$ different object categories, while MS-COCO is smaller and used for classification, detection and segmentation of 80 classes. For this reason, a segmentation network pre-trained on both datasets may benefit more from the learned features than using only a general ImageNet pre-training [126]. The DNN training was performed on a desktop computer equipped with an Intel Core i7-3rd generation CPU, 16 GB RAM and an Nvidia Titan Xp GPU with 12 GB memory.

2.3.4 Surface reconstruction, hole's detection and peg insertion

To reconstruct the workpiece surface, the same procedure reported in the Section 2.2.1 has been used. However, in this case, the input of the ICP registration algorithm are the n segmented point clouds obtained after the workpiece scanning and the segmentation phases. The output of this process is a point cloud, $\mathcal{P}_r$, representing the whole workpiece surface in the coordinate frame $\mathcal{F}_c^1$ (see Section 1.3).

After that, a first estimate of the hole positions and tilts is computed by comparing, $\mathcal{P}_r$, with a point cloud, $\mathcal{P}$, extracted from the CAD model. The steps used to compute the hole positions, p_{h_i}, and their normal unit vector to the surface, n_{h_i}, in the robot base

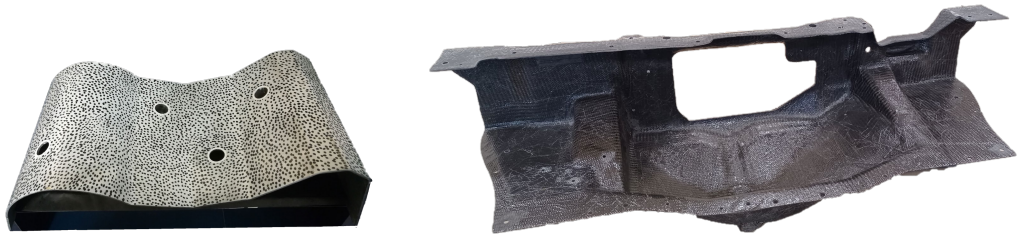


Figure 2.26: Workpiece used to validate the first strategy (a) and the one used for the second strategy (b).

frame, are the same described in Section 2.2.2.

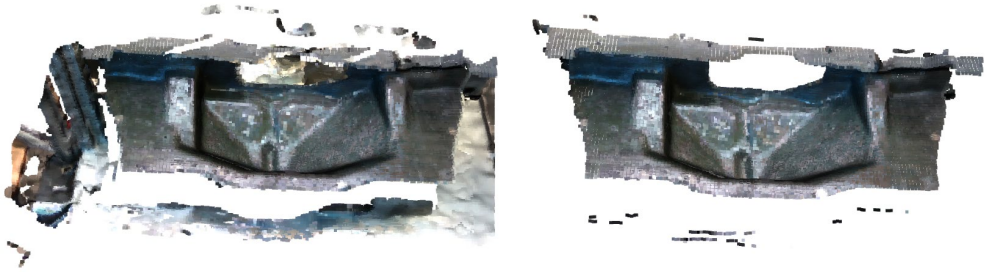
Such initial estimate is likely to be affected by errors, due to reconstruction and calibration process. Therefore, before the insertion, a CNN is exploited to detect with better accuracy the holes on the workpiece surface. The detection is performed by using the model built for the first strategy and described in Section 2.1.2. It is worth noticing that the workpiece used to train the network was different from the one used to validate this strategy. The two workpieces are shown in Fig. 2.26: the first one has a white surface with stochastic black speckle pattern, in which the holes appear as black ellipses, while the one considered for this strategy is a dark carbon fiber workpiece, and the surface is characterized by high reflectivity. Despite this, the detection performance on the carbon fiber object are satisfactory. The hole detector runs with a processing time of about 1.05 seconds per image on CPU.

A pixel corresponding to the hole in the image is extracted using the detection information provided by the network. In particular, the center of the bounding box coordinates is computed in the image reference frame and, then, is transformed in the robot base frame by using the homogeneous matrices provided by the calibration process [121].

Once the hole position and its normal unit vector have been estimated in the robot base frame, the peg insertion is performed with the same process described in Section 2.2.5. The search phase is still present to overcome detection errors due to the fact that the CNN doesn't have an accuracy of 100%.

2.3.5 Experimental results

The experiments have been executed with the same hardware and software of the previous strategy. To have statistically significant results, 26 insertion tests have been carried out by randomly positioning the workpiece in the robot workspace. The effect of the seg-



(a) (b)

Figure 2.27: Workpiece surface reconstruction without (a) and with (b) the application of the segmentation network.

mentation network and the CNN have been evaluated by considering an ablation test. In particular, for each insertion, two experiments have been carried on. In the first one, the surface reconstruction is performed without using the DeepLabv3+ network, i.e. the ICP algorithm is fed by the point cloud directly acquired by the camera and the hole localization does not exploit the CNN. In the second set of experiments, the new improved strategy described in this Section is used.

In each test, the robot initially scans the workpiece and acquires $N = 8$ different point clouds via the depth sensor. As can be noted in Fig. 2.27, the use of DeepLabv3+ allows to remove the background from the acquired point clouds and to have a better reconstruction of the surface. Moreover, the use of the network reduces the time required by the multiway registration algorithm of about 32%, as shown in Fig. 2.28.

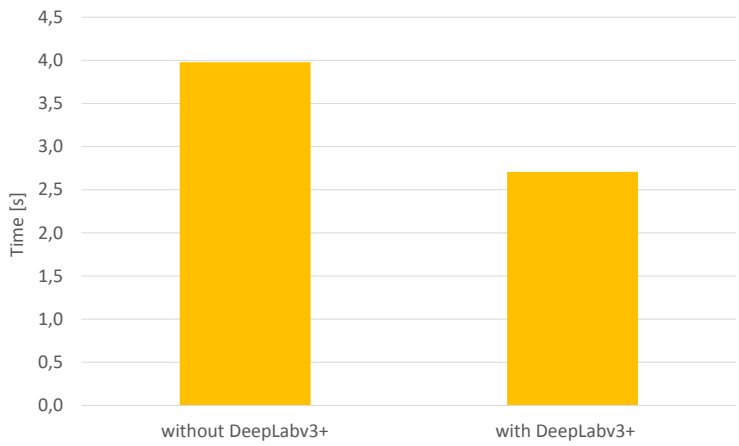
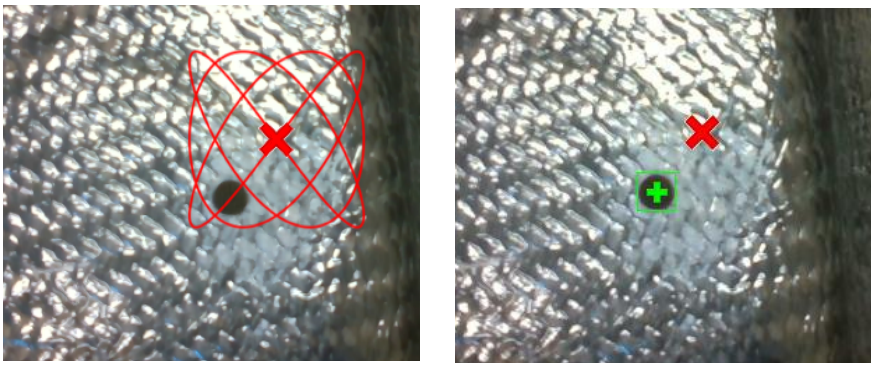


Figure 2.28: Registration time by using the multyway algorithm without (left) and with (right) the application of the segmentation network.



(a) (b)

Figure 2.29: The red cross indicates the hole's position estimation without the use of the CNN. Left: the red line represents the search trajectory to explore the hole neighborhood. Right: the green square is the bounding box of the CNN and the green cross is its center, that corresponds to the final hole's position estimation.

Regarding the performance of the task execution, two indices have been considered: 1) the error between the estimated and actual hole's position and 2) the duration of the search phase before the insertion. Fig. 2.29 shows the estimation obtained by simply comparing the reconstructed surface with the point cloud extracted by the CAD model and that obtained by using the CNN approach.

The adoption of the CNN allows to strongly reduce the error as demonstrated by the results shown in Fig. 2.30, where the mean errors for the 26 tests are compared with and without the CNN. The error is computed as the distance between the estimated position and the actual one computed on the hole's plane. The adoption of the CNN for estimating the hole position allows to halve the mean error on the three holes (from 5.1 mm to 2.3

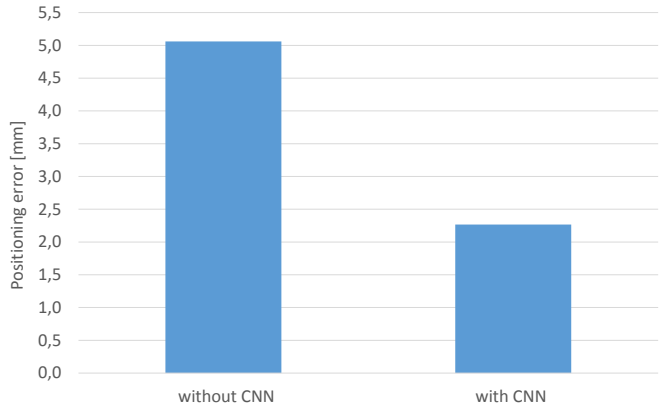


Figure 2.30: Mean hole's estimation error without (left) and with (right) the use of the CNN.

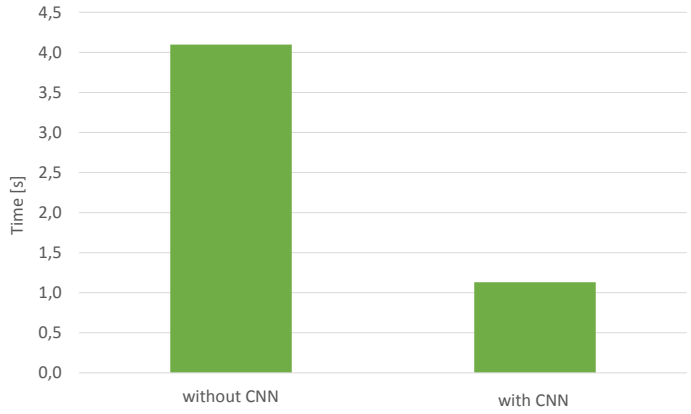


Figure 2.31: Search phase duration without (left) and with (right) the use of the CNN.

mm).

Once the hole position has been estimated, a search phase, in which the peg explores the neighborhood of the estimated position by following a path on the surface planned via Lissajous functions, is necessary, since the clearance between the hole and peg is very small (below 1 mm). The duration of the search phase is strictly related to the estimation error, thus it is not surprising that the adoption of the CNN allows to reduce the search time of about 72.4%. In Fig. 2.31 the mean duration of the search time is shown in both cases.

Finally, Fig. 2.32 reports the success rate computed on the 26 tests. More in detail, it can be viewed that, the segmentation network as well as the adoption of the CNN, allows to obtain a success rate of 98.7%, i.e. only in one attempt on 78 (26 tests for three holes)

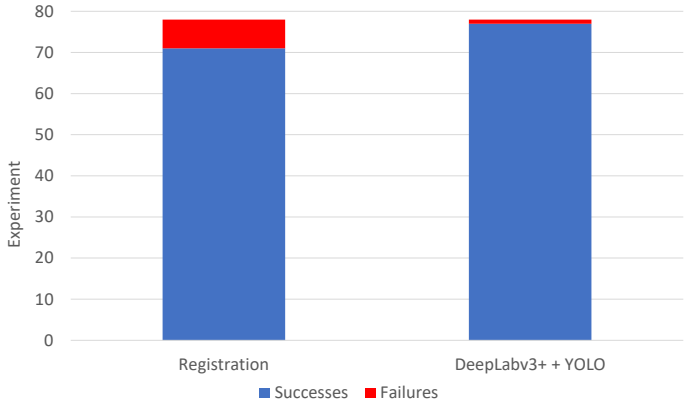


Figure 2.32: Success rate both in the absence (left) and in the presence (right) of the neural networks.

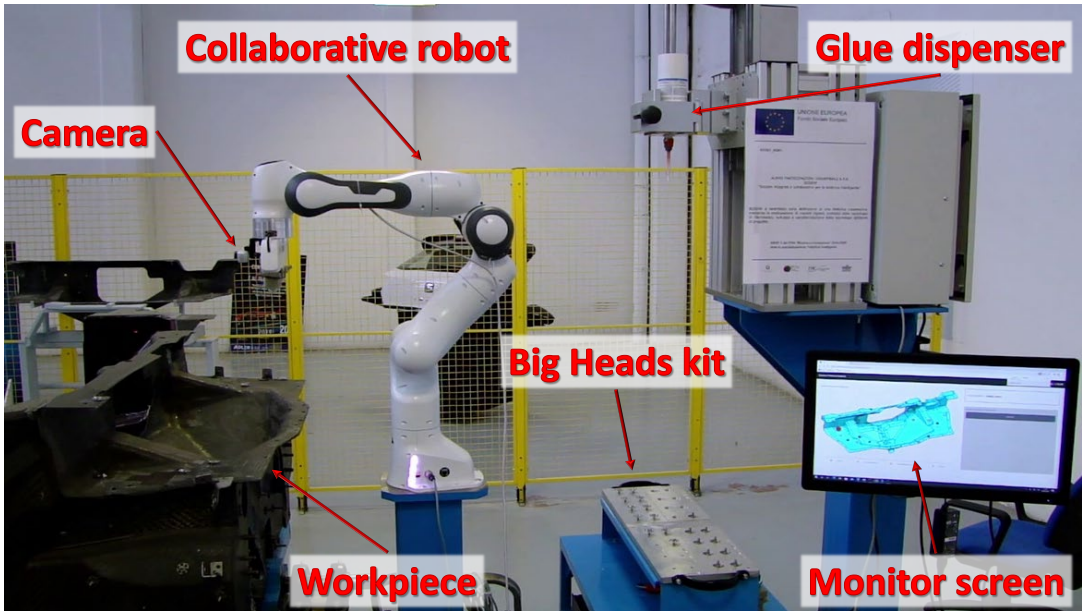


Figure 2.33: Industrial setup at the Tecno Tessile Adler S.r.l. company.

the insertion failed. In the absence of the neural networks, i.e. by using only registration methods, the success rate decreases to 91%.

2.4 Collaboration with human

The workpiece described in the precious sections, represents a portion of a supercar's safety cell and the Big Heads insertion in this workpiece is a task actually implemented in the automotive industry. It is currently performed manually through the use of insertion masks, which the operator places on the object's surface in order to highlight where the Big Heads has to be inserted, after applying some glue on them.

The strategy described in Section 2.2 has been integrated with other modules for taking the Big Heads from the kit, for depositing the glue on them and for monitoring the operation. This integration led to the development of a real setup (Fig. 2.33) at the Tecno Tessile Adler S.r.l. company [136], based in Airola (BN), Italy, where a robot and an human operator collaborate to perform the task.

In detail, the robot has to glue the Big Heads, with two different diameters, in 11 holes spread over the whole surface of the workpiece. Some Big Heads have to be inserted in the holes and others have to be laid on their flat side. The execution can be divided into smaller tasks that are repeated for each inserted/laid Big Head. To facilitate the definition

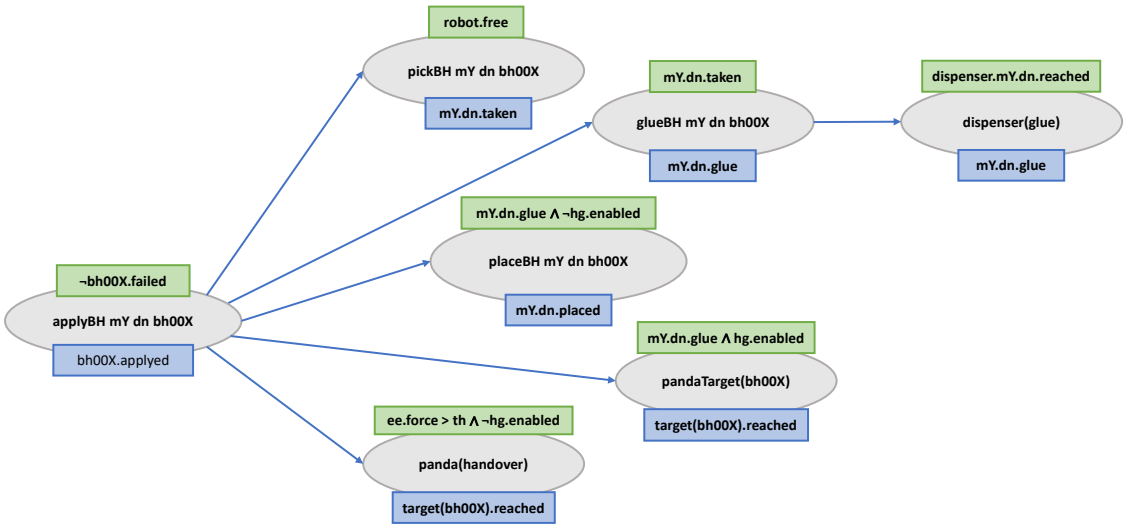


Figure 2.34: Insertion task representation.

of these tasks, they are represented through an annotated rooted directed graph [137], where the nodes represent tasks to be accomplished, while the edges represent parental relations among tasks and subtasks. This graph represents the roadmaps of possible paths that the robot end-effector can follow for the execution of the whole task and can also be used to represent logic constraints for the subtasks. The graph is exploited by a supervisory system to monitor the whole process, i.e., to keep track of the "processed" Big Heads, of those that need to be inserted or laid and to check that all constraints are satisfied before proceeding.

A single insertion task is identified in the graph via the parametric instance "*applyBH dim or id*", where the parameter "applyBH" is the high level task name, "dim" and "or" represent, respectively, the diameter of the Big Head to be inserted and the gripping orientation, i.e., if the Big Head needs to be grasped from the pin or from the flat part and "id" represents the identifier of the position (or hole) in which to insert/lay the Big Head. The graph representing a single insertion task for a generic Big Head is shown in Fig. 2.34.

However, it is possible that the insertion or the laying are not successful. In this case, the robot detects the error and communicates it to the supervisory system with an error message. The error indication also appears on the screen near the operator, who will have to solve the problem. After putting the robot in a safe condition by using the appropriate button, the operator can intervene in two ways:

1. manual insertion/laying in the relative hole: the human approaches the robot, takes the Big Head from the end-effector and places it manually (handover). After that, she/he communicates to the supervisory system that the insertion/laying has been carried out;

2. automatic insertion/laying in another hole: the human approaches the robot and, in hand-guidance mode, moves it in the direction of another hole. The system detects which hole the robot is moving towards and visualizes it on the screen. The operator, looking at the screen, notices that the system has recognized her/his intention and confirms the prediction. Then, the robot inserts/lays the Big Head in automatic mode.

After receiving confirmation from the operator, the system resumes the task execution for the next Big Head. The human intention is detected with the method described in [138], where attention regulation mechanisms and intention recognition processes are fully integrated and the sequence of past interactions are exploited to assess the human intent during the task.

An insertion task can be split in three subtasks:

1. the task identified through the instance "*pickBH mY dn bh00X*", that requires the robot to pick up a Big Head with a diameter of Y mm from the pin. This operation can be performed if the robot has no objects in the gripper (logic constraint) and is considered completed when a Big Head with the specified characteristics has been correctly taken by the robot;

2. the task identified through the instance "*glueBH mY dn bh00X*", that provides for the addition of glue on the Big Head and can be performed if a Big Head compatible with the required characteristics has been grasped by the robot and is considered completed when the glue is successfully applied;

3. the task identified through the instance "*placeBH m5 dn bh00X*", that involves inserting/laying the Big Head in position bh00X and can be performed if a Big Head compatible with the required characteristics has been grasped by the robot and if the hand-guidance mode is disable. This operation ends when the Big Head is correctly applied in place.

As mentioned, in case of error, the operator can decide to act in two different ways, and the relative subtasks are the following:

(a) (b)

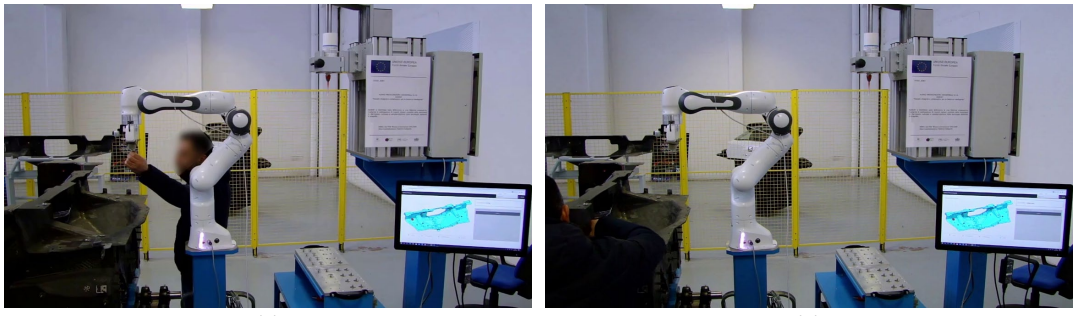
Figure 2.35: Big Head picking form the kit (a) and application of glue by using the dispenser (b).

1. the operator *"panda(handover)"* is relative to the manual insertion/laying. It detects the human intention to manually insert/lay the Big Head and commands the robot to release it. This process starts only if the forces exerted on the end-effector exceed a certain threshold and if the hand-guidance mode is not active;

2. the operator *"pandaTarget(bh00X)"* monitors the human intention to insert the Big Head in the bh00X hole by moving the robot end-effector to indicate the new location to perform the automatic insertion/laying. This process starts only if a compatible Big Head is held by the robot and if the hand-guidance mode is active.

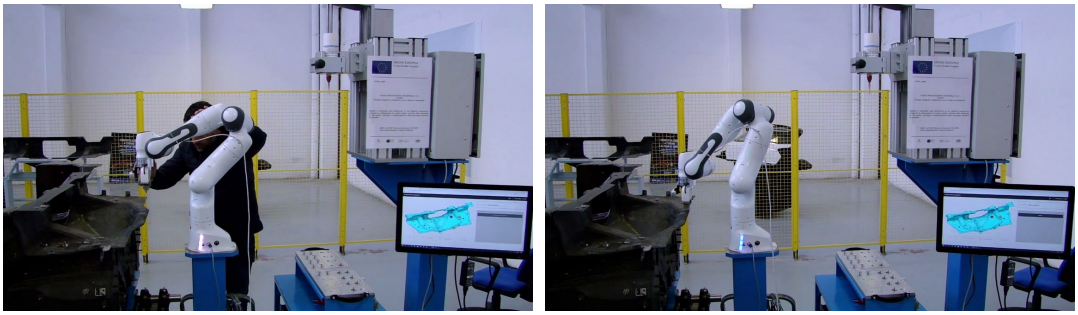
Some snapshots of the whole task execution are shown in Figs. 2.35-2.37. The robot knows the Big Head positions in the kit and, based on the command received by the supervisory system, it takes the ones from the pin or the ones from the flat part (Fig. 2.35(a)), then, the robot reaches the dispenser to apply glue on the Big Head (Fig. 2.35(b)).

The handover is shown in Fig. 2.36(a): the operator expresses the intention of performing the manual insertion/laying by taking the Big Head from the end-effector. Then, she/he manually places it, while the system is waiting for the command to continue (Fig. 2.36(b)).



(a) (b)

Figure 2.36: The operator takes the Big Head from the end-effector (a) and manually places it (b).



(a) (b)

Figure 2.37: The operator guides the robot towards the new target (a), the robot automatically inserts the Big Head (b).

Fig. 2.37(a) shows the operator that manually moves the robot to lead it towards a new target between those compatibles that are shown on the monitor screen. Once the supervisory system recognizes the human intentions (the new target candidate is highlighted on the screen), the operator commands the robot to continue the execution and it will perform the insertion/laying (Fig. 2.37(b)).

2.5 Conclusion

In this Chapter, two approaches to autonomous execution of robotic assembly tasks in partially structured environments is developed and experimentally demonstrated through a classical Peg-in-Hole task. The first approach is a preliminary study of an accurate blind method. In particular, the 3D-DIC is used the reconstruct the object surface roughly positioned in the robot workspace and an admittance control scheme is used to perform the peg insertion after the estimation of their position.

Despite the high accuracy, the method is difficult to automate and the performance of the 3D-DIC strongly depends on the quality of the pattern on the surface and the illumination. Moreover, it is not always possible to ensure the presence of a suitable pattern on the surface of industrial objects.

In the second approach a completely autonomous pipeline has been designed for the insertion of mechanical parts on a carbon fiber workpiece in a cooperative industrial scenario. Also in this case, the workpiece is positioned near the robot manually by a human operator, thus creating uncertainty on the relative position between the robot and the workpiece.

At the beginning of the process, a 3D reconstruction of the workpiece is computed

by using a registration algorithm and it is matched with a corresponding point cloud extracted from the CAD model to have an estimate of the holes' locations. Then, the robot approaches the hole and explores the hole neighborhood by sliding on the surface to complete the insertion.

Experimental results confirm the effectiveness of the approach even with wide object initial positioning errors. Moreover, the search phase strengthen the procedure by adding robustness to reconstruction errors: even if large estimation errors are experienced, the insertion can be successfully completed.

An improvement to this approach is made by adding deep neural network in the pipeline. In particular, a network for the image segmentation has been used to better discriminate the workpiece from the background and improve the overlapping with the CAD model, and a network for object classification is exploited to refine the hole's localization and reduce failures.

As future work, a more extensive experimental campaign could be conducted in real industrial scenarios and with different workpieces to test the capabilities of the image segmentation in the presence of different backgrounds and of the CNN with different light conditions.

Chapter 3

Object grasping and cooperative robotics

Grasping partially known objects in unstructured environments is one of the most challenging issues in robotics. It is a complex task and requires to address multiple subproblems in order to be accomplished, including object localization and grasp pose detection. In particular, deciding where the end-effector of the robot should come in contact with the object and the type and amount of forces that should be applied are challenging tasks.

In the literature, the approaches to the object grasping problem can be roughly classified into analytic and empirical [139]. Analytic methods are based on a certain degree of knowledge of the object (e.g., geometry, mass, material, etc.) and require at least a simplified contact model [140]. Empirical (or data-driven) approaches rely on grasp pose candidates for the object, chosen on the basis of given metrics [141].

Data-driven methods, and in particular deep-learning approaches, benefit from the availability of powerful GPUs. If the objects to grasp are known, deep learning methods are very effective as long as a database containing geometric object models and a number of good grasp poses are available. For example, Convolutional Neural Network (CNN) are used in [142], where an improvement of the structure of the Faster R-CNN neural network is designed to achieve a better performance and a significant reduction in running time of detection; experiments on a mobile manipulator show that the detection results allow to carry out a successful grasping. In CNN-based grasping approaches, once the grasp pose is determined, a robot executes a motion towards that pose without any other feedback. For this reason, a precise calibration between the camera and the

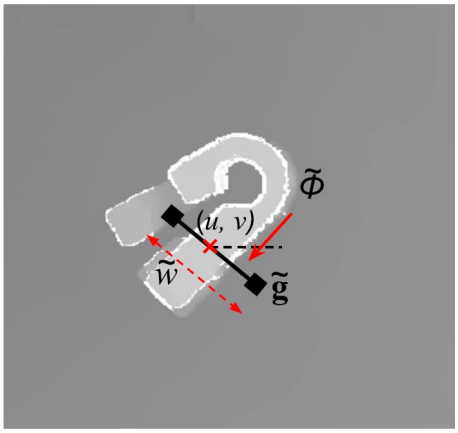


Figure 3.1: The grasp pose $\tilde{g}$ in the depth image is defined by its center pixel $\boldsymbol{s} = (u, v)$, its rotation $\tilde{\phi}$ around the image axis and perceived width $\tilde{w}$. The Figure is taken from [143].

robot, a precise robot control and a completely structured environment are required.

In [143, 144] a Generative Grasping Convolutional Neural Network (GG-CNN) has been proposed. It directly generates a grasp pose and quality measure for every pixel in an input depth image and is fast enough to performing grasping in dynamic environments. Given a depth image $\boldsymbol{I} \in \mathbb{R}^{h \times w}$, where h and w are the height and width of the image, respectively, a grasp is described by $\tilde{\boldsymbol{g}} = (\boldsymbol{s}, \tilde{\phi}, \tilde{w}, q)$, where $\boldsymbol{s} = (u, v)$ is the center in pixel of the box representing the grasp pose, $\tilde{\phi}$ is the grasp rotation in the camera reference frame, $\tilde{w}$ is the grasp width in image coordinates, i.e., the gripper width required for a successful object grasp, and q is a scalar quality measure, representing the chances of grasp success. An example of grasp description is shown in Fig. 3.1.

The set of grasps in the image space can be referred as the *grasp map* of $\boldsymbol{I}$, $\boldsymbol{G} = (\boldsymbol{\Phi}, \boldsymbol{W}, \boldsymbol{Q}) \in \mathbb{R}^{3 \times h \times w}$, where $\boldsymbol{\Phi}$, $\boldsymbol{W}$ and $\boldsymbol{Q}$ contain the values of $\tilde{\phi}$, $\tilde{w}$ and q respectively at each pixel $\boldsymbol{s}$. From $\boldsymbol{G}$, it is possible to compute the best visible grasp in the image reference frame and then, through the calibration matrices, this pose is expressed in the inertial reference frame to command the robot and grasp the object.

More challenging is the case of unknown objects, where it is assumed neither object knowledge nor grasp pose candidates are available. In this case, some approaches require to approximate the object with shape primitives, e.g., by determining the quadratic function that best approximates the shape of the object using multi-view measurements [145]. Other approaches require to identify some features in sensory data for generating grasp pose candidates [146]. A possible workaround is to assume that new objects are similar to known ones, in terms of shape, color, texture or grasp poses (familiar objects [141]). In

this case, the goal is to classify the objects on the basis of a similarity metric so to transfer the grasp experience. In [147] the grasp pose candidates are defined by identifying parts to which a grasp pose has already been successfully tested. In [148] the objects are divided in category and for each category the same grasp pose candidates are assumed.

In the rest of the Chapter, methodologies to address the grasping problem are presented and analyzed. In particular, the *Background subtraction* technique applied to industrial problems and a method to detect the grasp pose for industrial objects are detailed. Finally, an application that include object grasping and the cooperation between two robots is described.

3.1 Background subtraction

Background subtraction (BS) is a common method for detecting moving objects from images taken by static cameras, able to achieve real-time performance. This method is usually used in intelligent video surveillance [149], in monitoring road traffic applications [150] and, more recently, in monitoring maritime traffic [151]. The basic idea is to identify moving regions by comparing the current image with a model of the scene background.

In this Thesis, BS has been used to solve the problem of object grasping in an industrial scenario, where usually the objects to be grasped can vary, while the boxes that contain the objects remain the same. The advantage in modeling the background instead of trying to detect the objects of interests (i.e., the foreground) is significant, because a single, fixed detection model is used without the need of creating a new one when a new object needs to be grasped.

According to this technique, given an image $\boldsymbol{I}$ from the camera, the foreground mask $\boldsymbol{F}$, containing the objects of interest, can be detected by applying the following formula:

$$\boldsymbol{F} = \boldsymbol{I} - \boldsymbol{B}, \tag{3.1}$$

where $\boldsymbol{B}$ is the background model, i.e., a model representing the scene without the objects. All quantities mentioned above have the same dimensions, depending on the resolution of the camera. BS includes three main phases:

1. initialization, in which the first model of the background is built;

2. detection, where the foreground is determined;

3. update, in which the model is updated with respect to the changes of the scene.

The major disadvantage of the basic BS is that it cannot handle sudden changes in background, e.g., due to varying light conditions, shadows, and camera movements. For this reason, a variant has been analyzed and exploited in the developed application. The Independent Multi-modal Background Subtraction (IMBS) is the variant proposed in [151] and has been designed to deal with highly dynamic backgrounds, such as water. Moreover, it is robust with respect to illumination changes and camera jitters.

3.1.1 Background initialization

The background model $\boldsymbol{B} \in \mathbb{R}^{w \times h}$, where w and h are the width and the height of the input images, respectively, is computed through a per-pixel statistical analysis of N images (or samples). These images can be sequentially taken from the camera or can be extracted from a video, by using a sampling period, P.

Each element (i, j) of the matrix $\boldsymbol{B}$ consists of a set of couples $C = (c, f(c))$, where c is a value in a given color space (e.g., RGB or HSV) and $f(c)$ is a function that counts how many pixels in the scene have their value equal to c. It is worth noticing that using a single value for all the color channel has the advantage of detect the statistical dependence between the color channels. This would not happen if each channel was considered independently.

Once the last sample has been processed, if there is a couple $C^* = (c^*, f^*(c))$ where $f^*(c) \geq D$, then the color value c becomes a significant background value. The scalar D represents a threshold value which allows to distinguish objects that are "passing" through the scene. In fact, unlike other methods, IMBS allows for obtaining an accurate BG model, even if moving objects are present in the scene. This is possible because moving objects will appear in different positions in the images and for short time, therefore the background values related to those objects will be discarded by using the threshold D.

Statistic models are widely used to model the background, both with a single Gaussian distribution and with a mixture of Gaussians [152]. However, building an accurate model in the presence of varying light conditions, shadow, and camera movements is a challenging task. The choice about how to generate the background model depends also on the specific application environment. Models built with a single Gaussian are not suitable for outdoor environment with highly dynamic background.

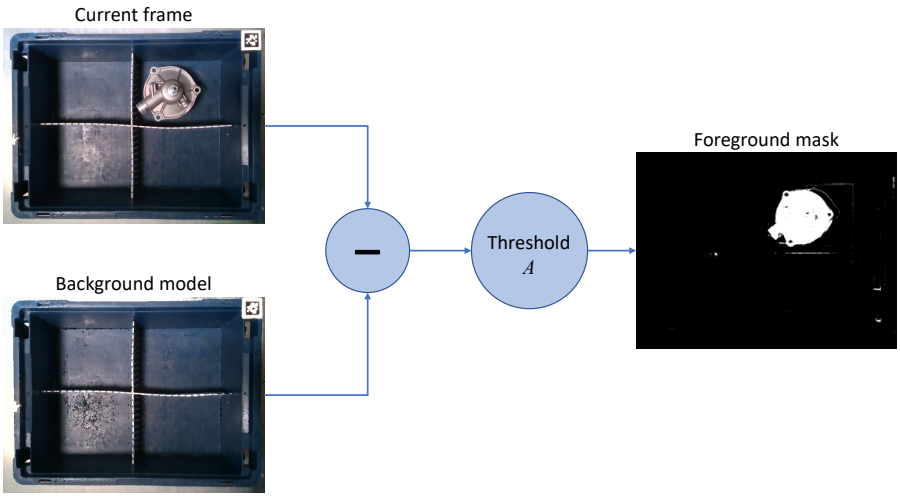


Figure 3.2: Foreground detection.

3.1.2 Foreground computation

The foreground mask is obtained by comparing the current image $\boldsymbol{I}(t)$ with the background model $\boldsymbol{B}(t)$, with a pixel-wise subtraction. As shown in Fig. 3.2, if the result of the subtraction is lower than a threshold A, the pixel is considered as part of the background and is labeled with black color in the foreground mask. Otherwise, the pixel is labeled with white color in the mask and considered as foreground.

3.1.3 Model update

After the creation of the first background model, which requires a time $T = NP$, where N is the number of images and P is the sampling period, IMBS continuously builds a new model, independent from the previous one, to adapt it to changes in the scene. The update is performed by using a FIFO (First In, First Out) strategy, i.e., the oldest sample, or pair, is discarded and a new sample, or pair, is added to the model.

In [153] two strategies to update the background are proposed. The first is the *selective* (or *conditional*) *update*, in which a new sample is added to the model only if it is classified as background. This method improves target detection, because the target pixels are not added to the model, but it requires to find a way to decide if a pixel is part of the background or not. A simple method could be the use of the detection result as an update decision, but any incorrect detection decision will result in persistent incorrect detection later and the background model will never adapt to it. The second strategy to update the background, proposed in [153], is the *blind update*, where new samples are always added

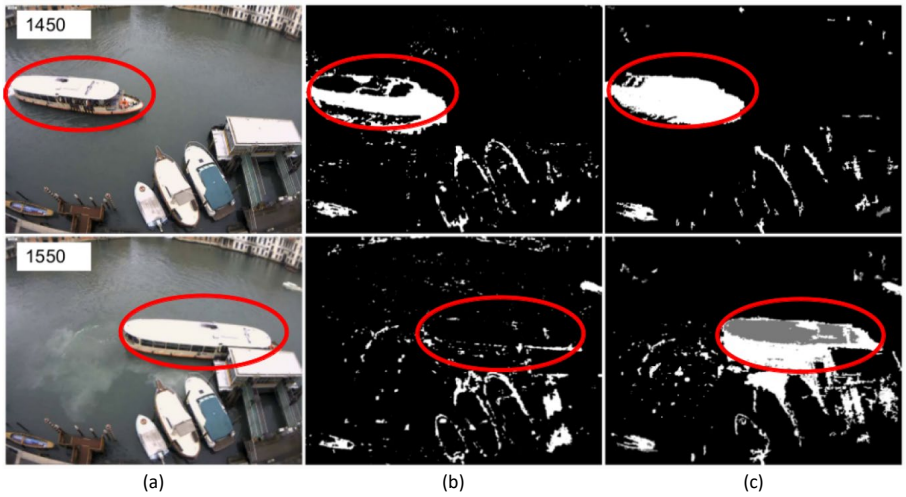


Figure 3.3: A boat enters the monitored scene and remains in the same position over several frames (a). A blind update gives a wrong result since the obtained model includes incorrectly the boat as part of the scene background (b). IMBS model update (c): the boat is identified as a potential foreground region (grey pixels). The Figure is taken from [151].

to the model. This method does not suffer of the persistent incorrect detection problem, but the detection of targets as false negatives might happen, since they erroneously become part of the model.

In IMBS methods a different solutions is proposed, aiming at solving the problems of both selective and blind update. Given a scene sample and a value of the pixel (i, j), for each couple $C \in B(i, j)$, if the pixel has been labeled as foreground in the previous foreground mask and the difference between its value and the value c is less than a parameter A, then C is labeled as a "foreground couple". During foreground computation, if $I(i, j)$ is associated with a foreground couple, then the pixel is classified as a *potential foreground* point.

This solution allows to detect not moving objects that don't belong to the background. An example of this situation is shown in Fig. 3.3, where a boat that remains in the same

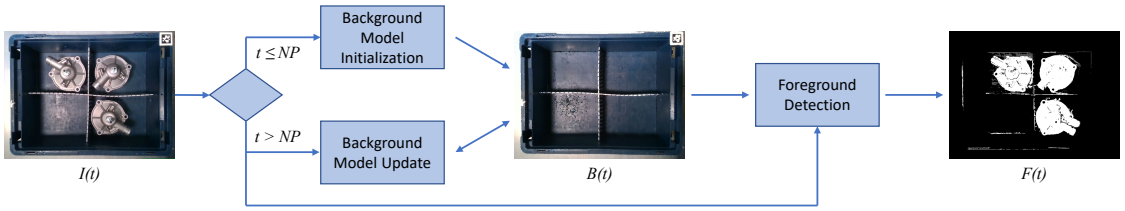
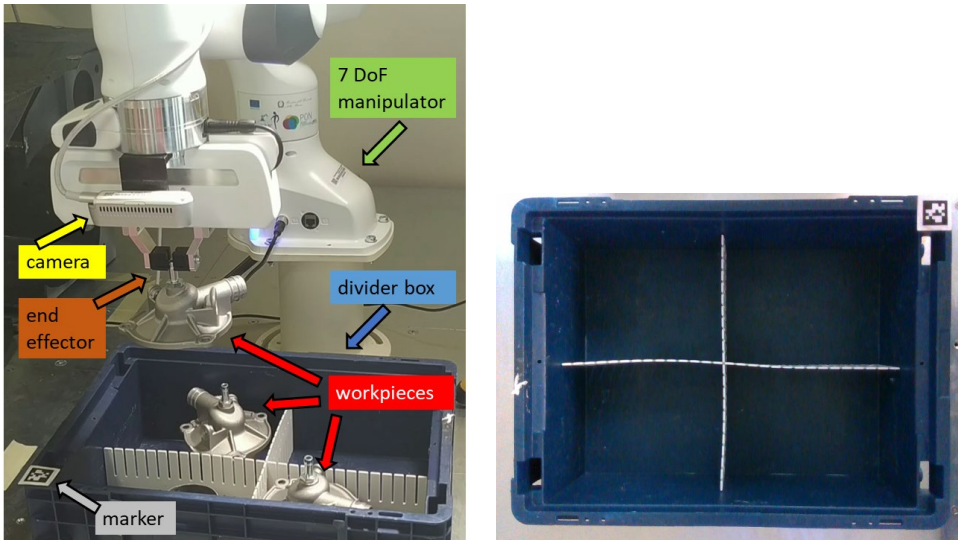


Figure 3.4: Background subtraction phases.



(a) (b)

Figure 3.5: Operational setting (a) and the box partitioned in subspaces by using the dividers (b).

position in several frames has been detected as a potential foreground. If a pixel is classified as potential foreground consecutively for a time period longer than a predefined value, it becomes part of the background model.

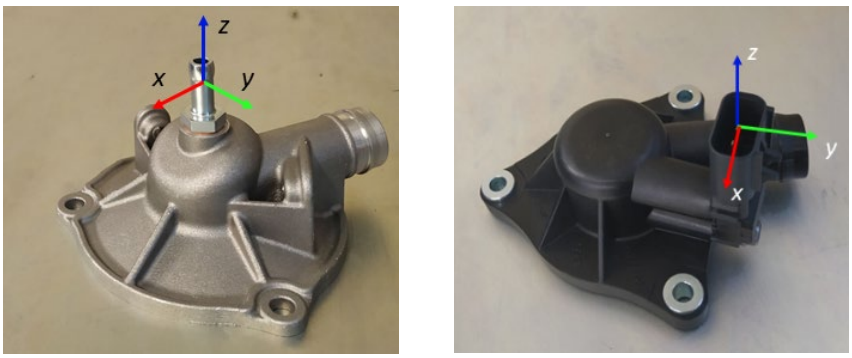
The three phases described above are shown in Fig. 3.4, where $I(t)$ is the input image at time t, $B(t)$ is the background model, and $F(t)$ is the foreground binary mask, which contains the object pixels.

3.2 Object grasping application

In this Section, an application of the BS technique in industrial environment is described and experimental results are presented. The objects of interest are placed in a divider box, whose position is uncertain and they need to be grasped by a robot equipped with a RGB-D camera (Fig. 3.5(a)). The box is partitioned with dividers in such a way to obtain different subspaces, each subspace can host a single object in order to avoid overlapping (Fig. 3.5(b)).

The above described application scenario can be defined as a quasi-static industrial environment, since the box containing the objects is placed within the robot workspace. The following assumptions on the object are done:

Assumption 1. *The object geometry is not known to the robot.*



(a) (b)

Figure 3.6: Objects considered and corresponding reference frames: metal oil separator crankcase (a) and plastic oil separator crankcase (b).

Assumption 2. *For each object, the grasp pose candidate is positioned on the top and a coordinate frame $\mathcal{F}_o = \{O, x_o y_o z_o\}$ is attached to the grasp point, so that the axis z_o coincides with the approach direction of the end-effector (Fig. 3.6).*

Due to the position uncertainties of the box, the robot motion cannot be planned off-line and the vision system must be adopted in order to 1) detect online the presence of an object in the subspace, 2) determine the grasp point, and 3) compute the best orientation to perform grasping.

Since the background subtraction requires the presence of a fixed background, the relative position between the camera and the box with the objects must be constant. Only slight positioning errors can be handled by the modelling procedure. In order to ensure that the camera-box relative position is constant after the box positioning, an AprilTag marker [154] has been attached to the box's edge (Fig. 3.7(a)).

Assumption 3. *The objects are always positioned in the box in such a way that the axis z_o of the grasping frame is aligned to the z axis of the AprilTag marker coordinate frame.*

Therefore, the robot aligns the camera reference frame to the marker reference frame by using the visual servoing algorithm implemented in the ViSP library [122] (Fig. 3.7(b)). Then, the background subtraction technique, described in Section 3.1, has been used to distinguish the objects from the box background (Fig. 3.7(c)). Once the presence of the object is detected, the point cloud, provided by the camera, is segmented by means of the foreground mask (Fig. 3.7(d)). This segmented point cloud is suitably filtered (Fig. 3.7(e)) as follows:

- a downsampling of the point cloud is performed by using a voxel grid filter [155];

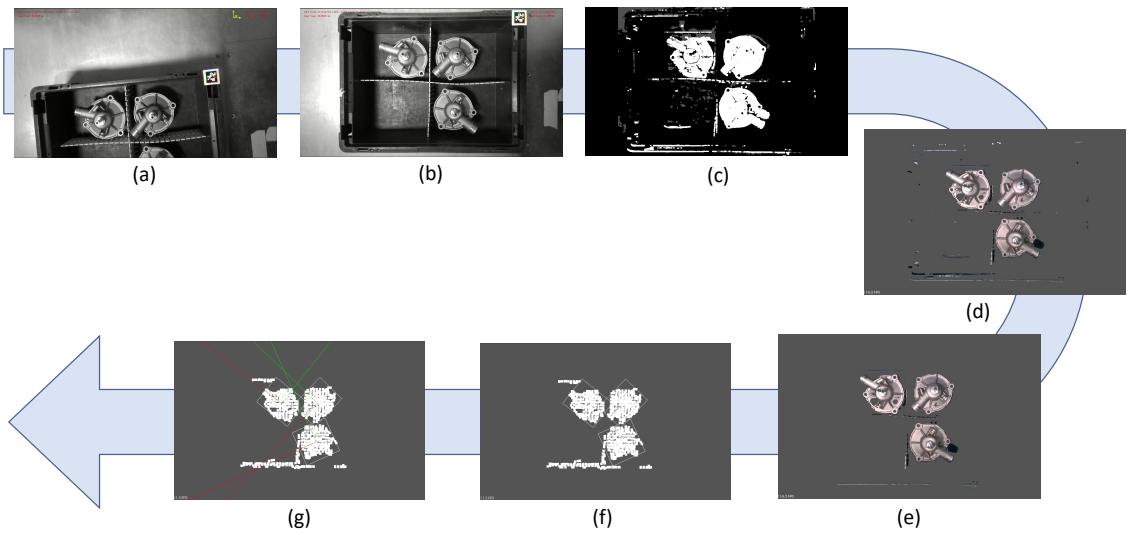


Figure 3.7: Functional scheme: a) initial condition; b) alignment with the marker; c) execution of the background subtraction algorithm; d) point cloud segmentation; e) point cloud filtering; f) point cloud clustering; g) estimation of the grasping pose.

- the points whose coordinates exceed the box dimensions are cropped.

The point cloud is clusterized by means of an algorithm based on Euclidean distance, proposed in [156] and detailed in Algorithm 2. Thus, a number of clusters are isolated, each corresponding to an object. Due to Assumptions 2 and 3, for each cluster, the vision system has to detect the point characterized by the minimum distance from the origin of the camera frame, that is the upper point of the object. Once the upper point is computed, the points whose distance is bigger than a certain threshold are removed from the point cloud. The remaining points are projected on a plane, in such a way to have a 2D cluster (Fig. 3.7(f)).

For each cluster, the oriented bounding box and the coordinate of cluster center $C = [x_C^c, y_C^c, z_C^c]^{\mathrm{T}}$, expressed in the camera frame, are computed by using an algorithm of the Point Cloud Library [157] (Fig. 3.7(g)). This algorithm computes the covariance matrix C_M, for each point $\boldsymbol{\pi}_i$ in the current cluster and, then, the eigenvalues and the eigenvectors of C_M are extracted and used to find the cluster center C and the orientation of the point cloud.

The system estimates the object grasp point in the camera frame as follows:

$$\widehat{O} = [x_C^c, y_C^c, \hat{z}_O^c]^{\mathrm{T}},$$

where $\hat{z}_O^c$ is the depth measure, obtained by the camera, at the point $[x_C^c, y_C^c]^{\mathrm{T}}$. In or-

Algorithm 2: Euclidean clustering.

Input : Input point cloud $\mathcal{P}$, distance threshold d_{th}

Output: List of clusters $\mathcal{C}$

Create a hierarchical data structure from $\mathcal{P}$.
Set up an empty list of clusters $\mathcal{C}$.
Set a queue $\mathcal{Q}$ which contains points that need to be processed.
for each $\boldsymbol{p}_i \in \mathcal{P}$ **do**
    add $\boldsymbol{p}_i$ to $\mathcal{Q}$
    for each $\boldsymbol{p}_j \in \mathcal{Q}$ **do**
        $\mathcal{P}_j^k \leftarrow$ set of point neighbors of $\boldsymbol{p}_j$ which are in a sphere with radius $r < d_{th}$.
        for each $\boldsymbol{p}_j^k \in \mathcal{P}_j^k$ **do**
            if $\boldsymbol{p}_j^k \notin \mathcal{Q}$ **then**
                add $\boldsymbol{p}_j^k$ to $\mathcal{Q}$.
            end if
        end for
    end for
end for
return $\mathcal{C}$

der to perform the grasp, the end-effector has to be commanded to align its reference frame (Fig. 3.8) to the bounding box coordinate frame. Since the coordinates of the estimated grasp point are determined by the vision system in the camera frame, they must be transformed in the robot base frame before commanding the motion. To this aim, the camera-end-effector transformation must be determined via a calibration [121, 158].

3.2.1 Experimental results

The experimental setup consists of a collaborative robot Franka Emika Panda equipped with the Intel RealSense D435 depth camera (Fig 3.5). In order to evaluate the effectiveness of the proposed approach, two different objects used in automotive factories, have been considered (Fig. 3.6).

The box model has been built on the basis of 43 images acquired with different light conditions and positioning the box in different locations, in order to simulate positioning

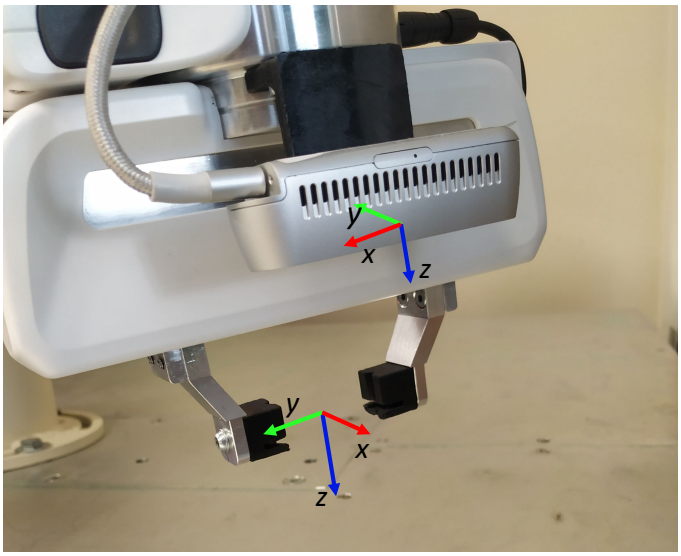


Figure 3.8: End-effector and camera frames.

uncertainties. Fig. 3.9 shows three samples and the generated background model.

The segmented point clouds before and after the filtering operation, for both the objects, are shown in Fig. 3.10. It can be noted that, before filtering, the background is mostly removed, but, due to different light conditions and/or slight alignment errors some residual points are still present in the foreground.

Then, the cluster algorithm is applied to the segmented point cloud. In this phase, it is crucial the choice of the cluster tolerance, i.e., the maximum distance between the points belonging to the same cluster. If a small tolerance is chosen, a single object could be seen as multiple clusters; on the other hand, if a very high value is chosen, multiple objects could be seen as a single cluster.

In the next step, the upper point of the object, assumed coinciding with the grasp point candidate, is determined by means of the depth measures, and a reduced point cloud is obtained considering only the points in its neighborhood. Such a reduced point cloud is projected on a plane in such a way to compute an oriented bounding box (Fig. 3.11).

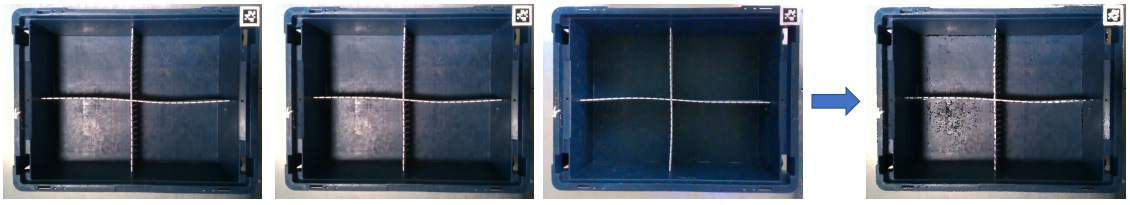


Figure 3.9: Three examples of images (left) taken to build the model (right).

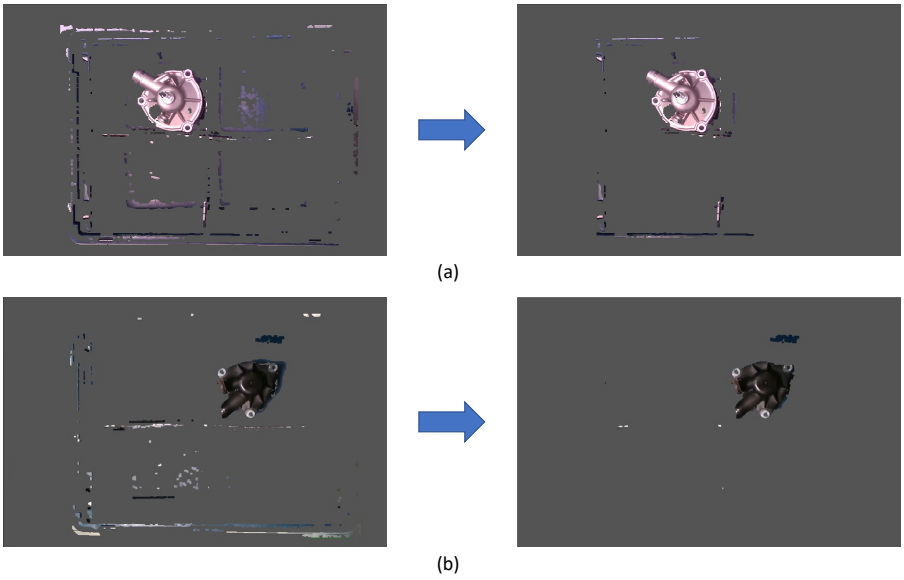


Figure 3.10: Left: Segmented point clouds before the filtering. Right: Point clouds after the filtering for metal oil separator crankcase (a) and plastic oil separator crankcase (b).

The coordinate frame attached to the bounding box, together with the depth measures, allow to compute the estimated grasp point and the desired orientation of the end-effector. More in details, due to the Assumption 3, only a rotation around the z axis is assigned, in such a way to align the y axis of the end-effector frame to the x axis of the object frame. Moreover, due to the particular shape of the chosen objects, for the metal crankcase the orientation is negligible, since it is grasped in a cylindrical part.

In order to have statistically significant results, 54 tests have been executed: 28 for the metal crankcase and 26 for the plastic one. Quantitative experimental results, in terms of the error between the actual grasp point position and the estimated one, expressed in the end-effector frame, are reported in Table 3.1, for the metal crankcase, and in Table 3.2, for the plastic one. In the latter case, also the orientation error between the end-effector y axis and the actual x axis of the object frame, is reported.

For the metal crankcase, it is worth noticing that not all the test have been concluded with a successful grasping, since, due to the object shape, even an error of few centimeters can lead to a failure. However, the estimation errors are rarely higher than 1 centimeter and only in a couple of tests it reaches 2 centimeters.

For the plastic crankcase, only one test fails the grasping (in boldface in Table 3.2), since the grasping area is bigger than the previous one and the bounding box is more accurate. It can be seen that the failed grasping is due to a large orientation error.

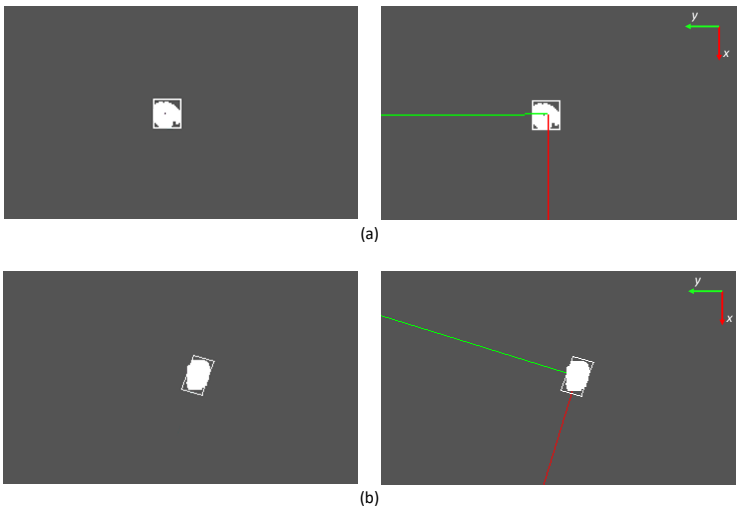


Figure 3.11: 2D reduced point cloud (left) and oriented bounding box (right) for metal crankcase (a) and plastic crankcase (b).

Finally, since the gripper is characterized by parallel fingers, in both cases higher errors along the closing direction are tolerated, while errors along the orthogonal direction

Table 3.1: Errors on grasp point for metal oil crankcase.

Test	$\boldsymbol{e}$ [mm]	Grasped	Test	$\boldsymbol{e}$ [mm]	Grasped
1	$(4.6, 11.0)$	Yes	15	$(1.2, 2.3)$	Yes
2	$(8.5, 10.7)$	No	16	$(5.3, 5.5)$	Yes
3	$(3.4, 8.7)$	No	17	$(2.6, 7.6)$	Yes
4	$(10.7, 11.4)$	Yes	18	$(0.2, 12.5)$	Yes
5	$(18.2, 8.3)$	No	19	$(2.7, 8.9)$	Yes
6	$(5.7, 5.2)$	Yes	20	$(2.0, 3.8)$	Yes
7	$(2.5, 12.7)$	Yes	21	$(7.0, 19.1)$	No
8	$(9.5, 13.4)$	Yes	22	$(0.5, 11.9)$	Yes
9	$(4.0, 2.8)$	Yes	23	$(0.3, 1.2)$	Yes
10	$(7.7, 5.3)$	Yes	24	$(22.8, 2.8)$	No
11	$(12.5, 5.8)$	No	25	$(6.8, 3.7)$	Yes
12	$(7.9, 5.3)$	Yes	26	$(5.8, 2.2)$	Yes
13	$(28.5, 2.6)$	No	27	$(5.5, 0.7)$	Yes
14	$(1.3, 10.5)$	Yes	28	$(10.8, 0.4)$	Yes

Table 3.2: Errors on grasp pose for plastic oil crankcase.

Test	$\boldsymbol{e}$ [mm]/[deg]	Test	$\boldsymbol{e}$ [mm]/[deg]
1	$(1.8, 2.2)/(4.3)$	**14**	**(1.8, 6.0)/(-20.2)**
2	$(0.4, 3.9)/(6.7)$	15	$(2.9, 5.9)/(-7.1)$
3	$(2.3, 0.6)/(8.7)$	16	$(0.9, 3.8)/(-4.9)$
4	$(2.4, 5.1)/(-0.8)$	17	$(0.2, 4.2)/(6.6)$
5	$(1.9, 3.6)/(-2.6)$	18	$(0.6, 5.9)/(-12.4)$
6	$(2.0, 4.4)/(-11.7)$	19	$(0.5, 2.0)/(-2.1)$
7	$(2.2, 4.4)/(0.7)$	20	$(0.5, 6.9)/(-9.2)$
8	$(0.1, 7.2)/(7.0)$	21	$(2.6, 5.6)/(4.0)$
9	$(2.3, 2.5)/(-9.5)$	22	$(1.3, 3.0)/(-0.3)$
10	$(0.1, 6.8)/(3.0)$	23	$(1.1, 3.3)/(0.8)$
11	$(0.1, 4.8)/(-5.6)$	24	$(0.7, 2.8)/(0.0)$
12	$(0.0, 7.6)/(2.6)$	25	$(1.2, 6.6)/(8.1)$
13	$(0.3, 4.3)/(-3.5)$	26	$(0.3, 5.3)/(-9.2)$

may cause a grasping failure. As for the detection time, for the metal crankcase the vision algorithm takes an average of 2.4 seconds for each single detection, while for the plastic one the average time is equal to 3.3 seconds. Such a difference is due to the different size of the 2D reduced point cloud, which makes faster the algorithm that computes the oriented bounding box.

3.3 Model generation

The methodology described in Chapter 2 is an alternative to background subtraction, when the object CAD model, in which the gripping point has been labeled, is available and the point cloud provided by the sensor is quite detailed. When the CAD is not available, the Digital Image Correlation, described in Section 1.2, could be used to build a model of the object. As mentioned, this technique is difficult to automate and the performance depends on the quality of the pattern on the surface and the illumination. For this reason, the use of the second reconstruction technique presented in the Section 2.1 has been tested on a set of mechanical workpieces used in automotive factories to verify its applicability even in the absence of the CAD model.

Figure 3.12: Robot and camera setup for data acquisition.

In this Section, a methodology to generate a model to be used as a substitute of the CAD is presented, and some experiments are performed to highlight its advantages and disadvantages. The basic idea to handle the grasping problem can be summarize as follows:

- the object data are acquired from different points of view, in order to obtain different point clouds of various portions of the object;
- the point clouds are merged to obtain the model of the object surface, through the ICP algorithm (see Section 1.3);
- a frame that represents the optimal grasping pose for the object is fixed on a point of the model built in the previous step;
- during algorithm execution, the model is overlapped on the current point cloud, in order to be able to transport the grasp pose on the current object;
- the current grasp pose is transformed into the robot coordinates frame;
- the robot is commanded to perform the grasp.

The setup to acquire camera data to build the model is shown in Fig. 3.12. The object is located above the table surface to allow a faster background elimination from the point cloud. In general, a segmentation neural network could also be used for this task (as done in Section 2.3.1), but given the high amount of pre-processing work and the fact that in an industrial environment objects could vary very rapidly, the use of supports to

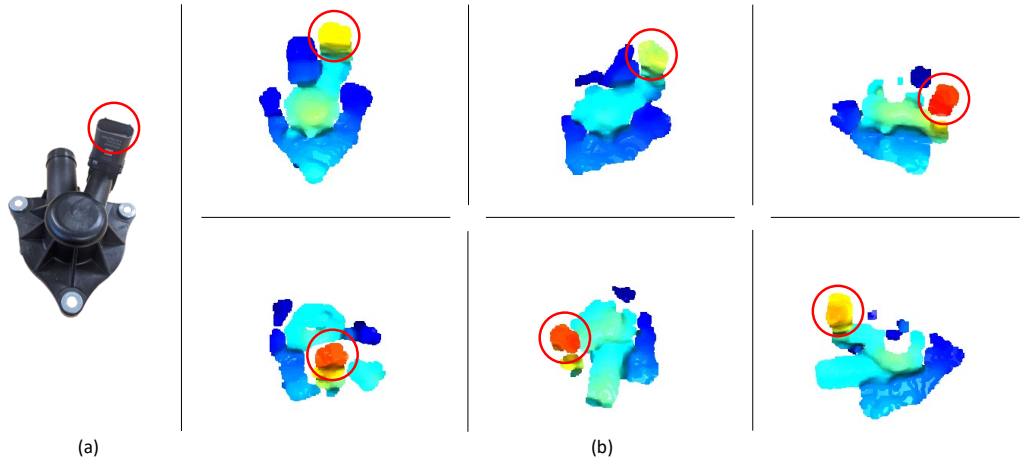


Figure 3.13: Some examples of point cloud (b) for the plastic oil separator crankcase (a). The red circle indicates the same part in the various views.

keep the object above the table has been chosen for this case, even if the accuracy of this technique is lower.

To generate the model, the position of the camera is fixed and the object is rotated to allow the data acquisition in different configurations. Some examples of acquired point cloud are shown in Fig. 3.13. Then, these point clouds are merged by using the ICP algorithm and a point cloud of the whole object is obtained (Fig. 3.14(a)). This point cloud represents the object model in which the grasp pose will be manually fixed, by using any modeling software. An example of a grasping pose is shown in Fig. 3.14(b).

When an object needs to be grasped, its point cloud is acquired and it is overlapped with the one that represents the model. In this way, the labeled grasp pose can be trans-

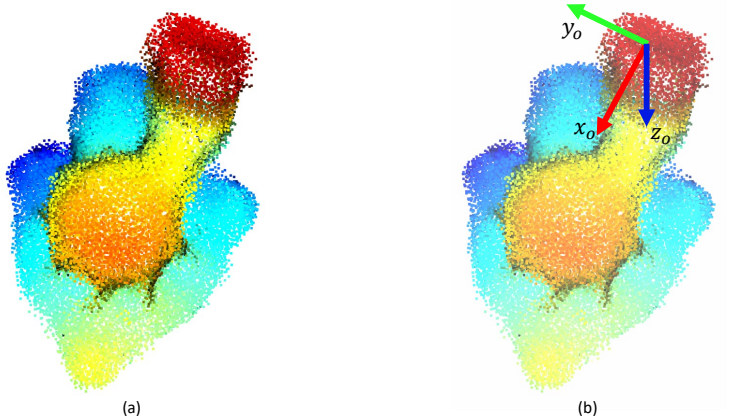


Figure 3.14: Example of the generated model for one object (a) and the relative grasp pose (b).

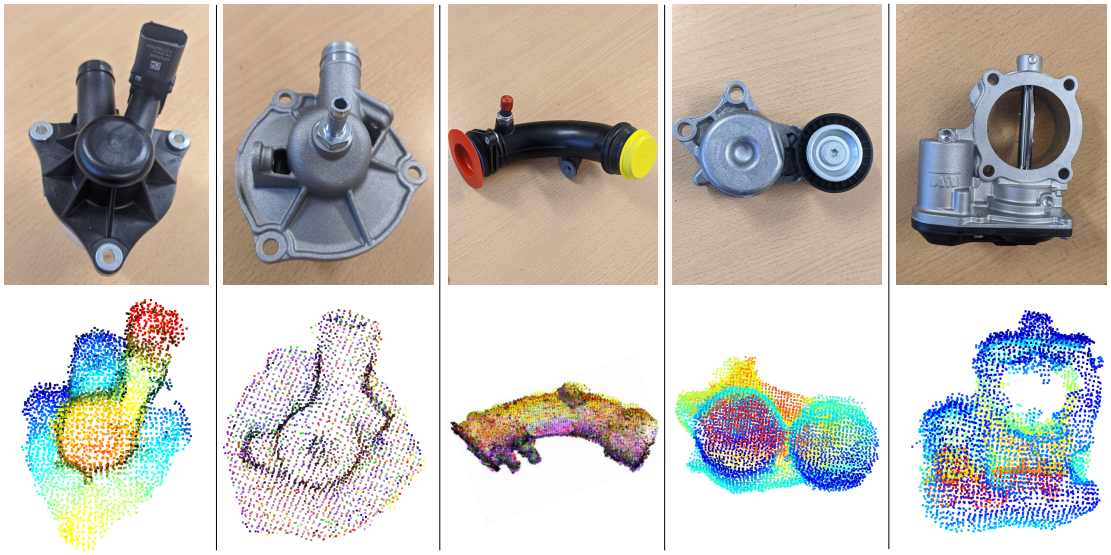


Figure 3.15: Mechanical workpieces and relative generated models: plastic oil separator crankcase (a), metal oil separator crankcase (b), air pipe (c), belt tensioner assembly (d), throttle body (e).

ported on the current object, that is referred with respect to the robot base frame. After a further transformation, by using the camera-end-effector calibration matrix, the robot can be commanded to perform the object grasp. The mechanical workpieces used in the experiments and their generated model are shown in Fig. 3.15.

During the experiments, it was noted that, when the current object has a very different orientation from the modeled one, the overlapping between the model and the current point cloud was not successful. This happens because the point clouds provided by the sensor are not very detailed and accurate; thus, some parts could be confused with others in the matching search process. Therefore, both the model and its grasping pose have been rotated, in order to have various models with different orientations. The current point cloud is compared with all the models and the model with the best match is selected to compute the grasping point.

The best match is measured through a parameter which measures the overlapping area between the two point clouds, named *fitness*. In particular, one of the two point clouds is chosen as a target and the fitness is computed as the ratio between the number of correspondence points and the number of the points in the target point cloud. A correspondence point represents a point for which has been found the corresponding point in the target point cloud. An example of correct overlapping and one of the incorrect overlapping for three of the considered workpieces are shown in Fig. 3.16.

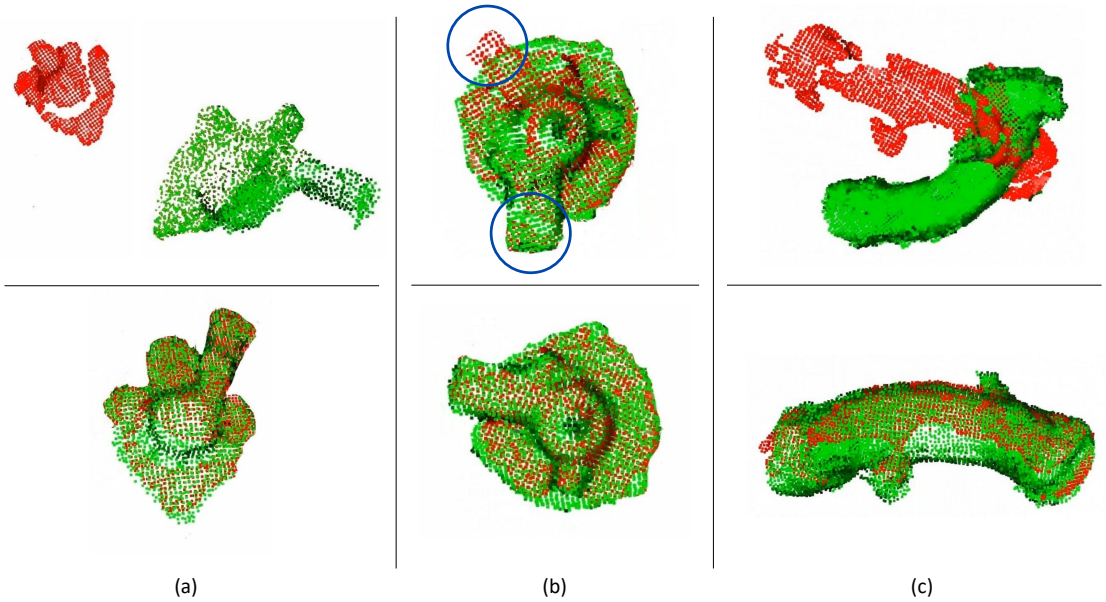


Figure 3.16: Examples of overlap failure (top row) and successful (bottom row) for three objects: plastic oil separator crankcase (a), metal oil separator crankcase (b), air pipe (c). Two blue circles highlight the non-overlapping for the metal oil separator crankcase by indicating the same object part not aligned.

Although the model is well-built, for three object types (air pipe, belt tensioner assembly and throttle body), the experiments show that the search for the best match was not successful. This is due to the symmetry that characterizes the object and the fact that the model is not very accurate. In these cases, the algorithm wasn't able to find the match because many portions of the object are quite similar.

For the air pipe, the correct overlap was found only when the object position was very similar to that of the original model. In this case, there were very few successful attempts in the testing phase, therefore no results can be derived from them.

For the remain types of object (plastic and metal oil separator crankcases), the algorithm was able to find the match and the robot was able to grasp the object. The error between the desired position (orientation) for grasping and the actual position (orientation) of the end-effector was about 6.1 mm (8.09 degrees) for the plastic oil separator crankcase and 6.6 mm (3.2 degrees) for the metal one.

3.4 Cooperative application: shafts handover

The object handover is a typical industrial task involving cooperative robots, e.g. in logistic applications, where robots are widely adopted in picking operations. Nowadays, collaborative robots are often used in handover with humans [159], and, in quasi-autonomous production plants, even with robots, e.g., a logistic robot and an assembly one. The robot that is in charge of grasping the object is called *giver* (or giver robot). To complete the task, after taking the object, the giver needs to pass it to the other robot, that is called *receiver* (or receiver robot). Object handover can be partitioned in two phases: the pre-handover and the physical exchange phase. The pre-handover phase includes the object detection, in which the giver must recognize the presence of the object to hand over in its workspace, handle the object grasping, the transportation and the synchronization, i.e., finding an agreement about the exchange location and timing [160]. In recent years, interest in object detection has burst due to the rapid development of deep learning techniques [161]. In particular, CNNs for object classification are mostly used, because they represent the best trade-off among accuracy and the detection speed.

Regarding the synchronization phase, it can require the giver robot to explicitly or implicitly communicate to the receiver. Explicit communication implies to share both sensory data and control signals among the robots, while implicit communication occurs when information is acquired only via sensors, e.g., via a force/torque sensor measuring the interaction wrenches, tactile sensors and/or visual sensors [162]. Explicit communication has been largely adopted for cooperative robots, since it allows to easily handle synchronization issues. However, in the presence of many and heterogeneous agents, the communication load can increase. Even in an industrial scenario involving only few manipulators, the communication channel, due to other devices connected, can experience packet loss and delays, which are detrimental to performance and can even cause production scraps. The use of an implicit communication, even if the control scheme becomes usually more complex and the performance worse, improves the flexibility and the scalability of the system.

The physical exchange phase starts at the instant of the first contact between the receiver robot and the object grasped by the giver robot and ends when the giver fully releases the object to the receiver [159]. The physical exchange requires cooperation among the giver and receiver robots, thus vision and force feedback can be adopted by the giver in order to understand if the receiver has grasped the object. Only when the grasping is

safe the giver can start to release the object and allow the transition to the receiver.

Regarding the object handover, during the physical exchange phase, the object load is shared by the giver and the receiver and they must guarantee the object safety. Different studies have been conducted for the force exchanged by humans during handover operations. For example, in [163], it is found that the grip force of both giver and receiver is modulated during the object exchange, i.e., while the giver decreases its force, the receiver increases it until the load is transferred. Then, after the unloading, the giver still applies a grasping force even though its sensed load is almost zero [164]. These results can be also applied to robot-to-robot handovers, where techniques for grip force modulation must be proposed. In [165], the sole communication mean between the two agents is provided by custom force/tactile sensors measuring the interaction force and moment. In this approach, the giver adopts a slipping detection algorithm that allows to foresee the possibility that the receiver cannot keep the object orientation and thus dangerous releases are avoided.

3.4.1 Proposed strategy

The considered task is an autonomous robot-to-robot object handover, in the absence of any explicit communication. The setup consists of two collaborative robot manipulators Franka Emika Panda, both equipped with a camera in eye-in-hand configuration. The camera on the giver robot is an Intel Realsense D435, while that mounted on the receiver robot is an Intel Realsense L515. The manipulated objects are a couple of counter-rotating shafts of different length and shape (Fig. 3.17).

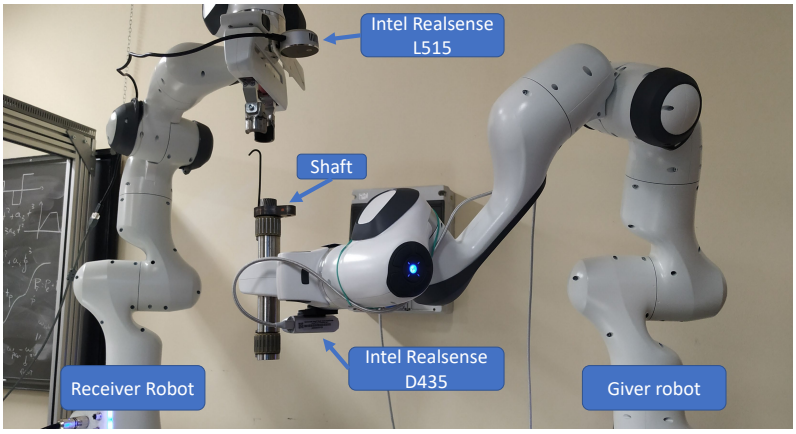


Figure 3.17: Experimental setup: two Panda Emika Franka robots equipped with depth cameras are used for handing over a shaft.

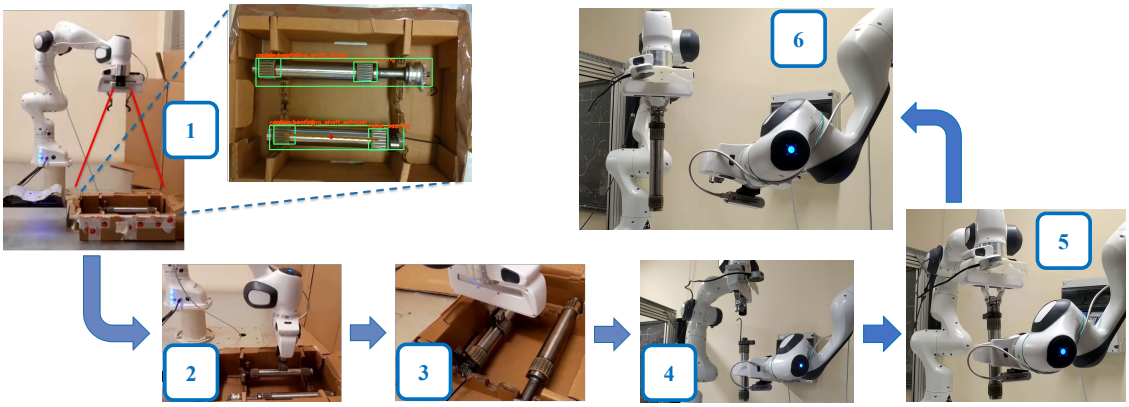


Figure 3.18: Functional scheme. 1) Object detection phase. 2) Estimation of the grasping point. 3) Object grasping and motion to the exchange point. 4) Detection of the giver at the exchange position. 5) Object grasped by the receiver. 6) Object released by the giver.

It is assumed that the objects are placed in a box roughly positioned in the field of view of the giver's camera. Due to the position uncertainties, the robot motion cannot be off-line planned and a vision system is adopted in order to detect the presence, identify the class and compute the pose of the objects. Among the object detection technique described in the previous Sections, the deep neural network approach has been chosen. Despite the tiring work to build and label the dataset, it is faster then background subtraction and the model-based technique.

The following strategy (shown in Fig. 3.18) is proposed:

1. When the box containing the objects is completely in the camera field of view of the giver robot, a CNN detects the presence, the orientation and the type of the shafts. The CNN recognizes also the roller bearings, that are used to determine the shaft axes and estimate the grasping pose.

2. The position of the grasping point is estimated by computing the center of the roller bearing bounding boxes and the shaft axis.

3. The giver grasps the shaft and moves it to the exchange point, assumed within the field of view of the receiver's camera.

4. The receiver robot recognizes, by using an eye-in-hand camera and a marker attached to the giver gripper, the giver pose and, thus, the object exchange point.

5. The receiver aligns its gripper to the shaft axis and moves toward the object until a contact is detected, then it closes the gripper.

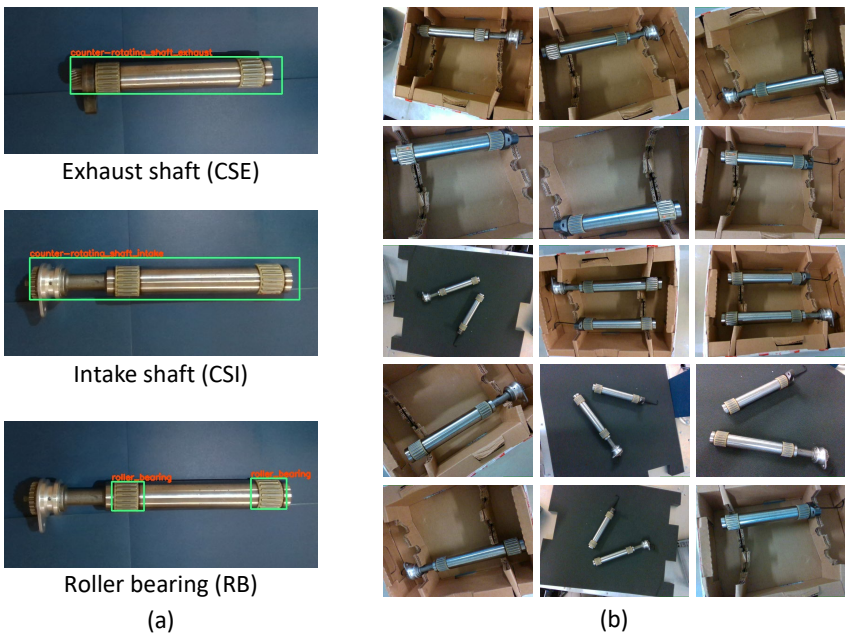


Figure 3.19: Classes for the objects of interest (a). Samples with different orientation and background (b).

6. The receiver moves the grasped shaft while the giver compliantly follows its motion. When the force exerted on the giver exceeds a threshold, the gripper is open and the shaft is released.

3.4.2 Shaft detection and estimation of the grasping point

To carry out the shaft detector, a CNN-based approach has been used. Two CNN architectures, Faster R-CNN and YOLOv4 (see Section 1.5.1), have been compared with respect to their performance to select the best one. For the Faster R-CNN, the Inception V2 [166, 167] was chosen, due to its high degree of accuracy. It is designed to reduce the complexity of CNN, by building an architecture that is wider than deep. In [166], the structure changes improve the efficiency of the network and generates a reduced model size, which, in turn, can help to reduce the overfitting problem.

Three object classes have been defined (Fig. 3.19(a)):

- counter-rotating shaft exhaust (CSE);
- counter-rotating shaft intake (CSI);
- roller bearing (RB).

Table 3.3: Average Precision for IoU value of 0.5.

Network Architecture	CSE	CSI	RB	mAP
YOLOv4	**0.952**	**0.966**	**1.000**	**0.973**
Faster R-CNN	0.928	0.855	**1.000**	0.914

Detecting CSE and CSI is useful to identify the shaft's type, while RB is used to determine the shaft axes and estimate the grasping point. A dataset made of 342 images, with 640×480 size, has been built by considering two shafts, captured at different distances and with different orientation and background (Fig. 3.19(b)).

The dataset has been manually annotated (using the `LabelImg` tool) and then split in two non-overlapping sets, namely the training (245 images) and test (97 images) sets. A data augmentation step has been carried out to create a larger training set. In particular, for each image, horizontal flipping, cropping and zooming have been performed. The augmented training dataset, made of 980 images, has been used to train both the Faster R-CNN and YOLOv4 networks.

An Intel Xeon 3.7 GHz CPU 32 GB RAM with a NVIDIA Quadro P4000 8GB GPU has been used to carry out the training phase, which required about 8 hours for the Faster R-CNN and 14 hours for the YOLOv4. To evaluate the detection performance, the mean average precision (mAP) metric has been considered [168]. The Average Precision (AP) for each class represents the integral of the precision-recall curve, measured for a certain value of the Intersection over Union (IoU, see Section 1.4). The mAP is the AP averaged over all classes. Table 3.3 shows detection results on the test set for a value of IoU of 0.5.

The detector is implemented in C++ and it runs on 640×480 images coming from the sensor mounted on the end-effector of the giver robot via the `librealsense2` library. The detection process takes on GPU an average time per image of 58 ms for the Faster R-CNN and 44 ms for the YOLOv4. On the basis of these results and performance, the YOLOv4 architecture has been chosen to implement the shaft detector.

Once the shaft has been detected, it is necessary to compute the position of the grasping point and the shaft orientation (Fig. 3.20). The latter is crucial to compute the gripper orientation, since the shaft must be grasped by the receiver always from the cogwheel. In order to identify the orientation, the distance, δ (Fig. 3.20), between the top-left corner of the CSI bounding box and the edge of the left RB bounding box has been computed.

It is assumed that, for both shafts, a suitable grasping point should be located along

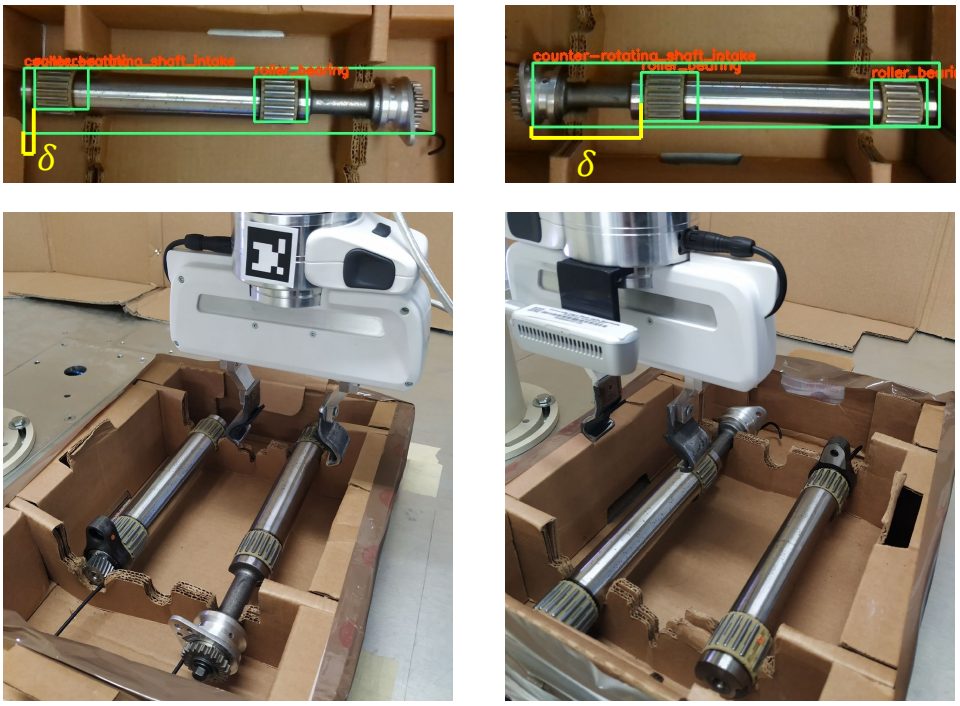


Figure 3.20: Two possible configurations of the shafts. The distance δ is computed in order to detect the shaft orientation.

the longitudinal axis. More in detail, the best grasping point for the CSE is equidistant from the center of the two roller bearings, while for the CSI, it is closer to the cogwheel. A two step procedure is used to estimate the position of the grasping points: 1) the shaft axis is estimated by connecting the centers of the two RB bounding boxes, 2) the grasping point is computed along this axis (Fig. 3.21). It is worth noticing that, due to possible variations in the detected bounding boxes size, the grasping point estimation could be not perfect. However, such uncertainties will be managed in the physical exchange phase by ensuring a suitable compliance to the robots.

The grasping point is detected in the image frame and, in order to perform the grasp, must be transformed in a 3D reference position, expressed in the robot base frame. To this aim, the RGB camera has been calibrated with a Direct-Linear-Transformation (DLT) method [169] using a 3D target.

3.4.3 Shaft handover

Once the grasping point position and the shaft orientation have been detected, the giver robot can be commanded to grasp the shaft in such a way to align the x_e axis of its end-

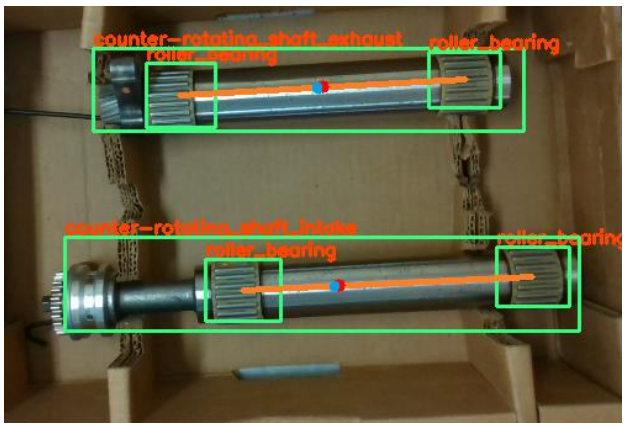


Figure 3.21: Shafts axes and grasping points: actual ones in blue and estimated ones in red.

effector reference frame (Fig. 3.22) with the shaft axis, and move it to the exchange point. The commanded motion is computed by using a closed-loop inverse kinematics algorithm [69]. The exchange point is off-line planned on the basis of the work-cell configuration in such a way the marker attached on the giver's gripper is in the field of view of the receiver's camera. It is worth noticing that the planned point is related to the end-effector reference frame, while the shaft tip position is unknown, due to the uncertainties of the grasping point estimation and the different length of the two shafts. Moreover, due to the absence of communications, the exchange point is unknown to the receiver, thus its motion cannot be planned off-line and performed via a pure positional control.

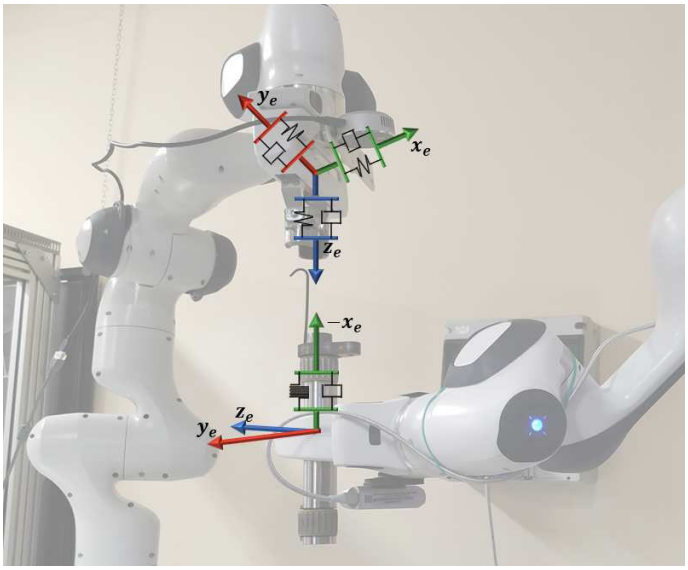


Figure 3.22: Reference frames for the robots' end-effectors.

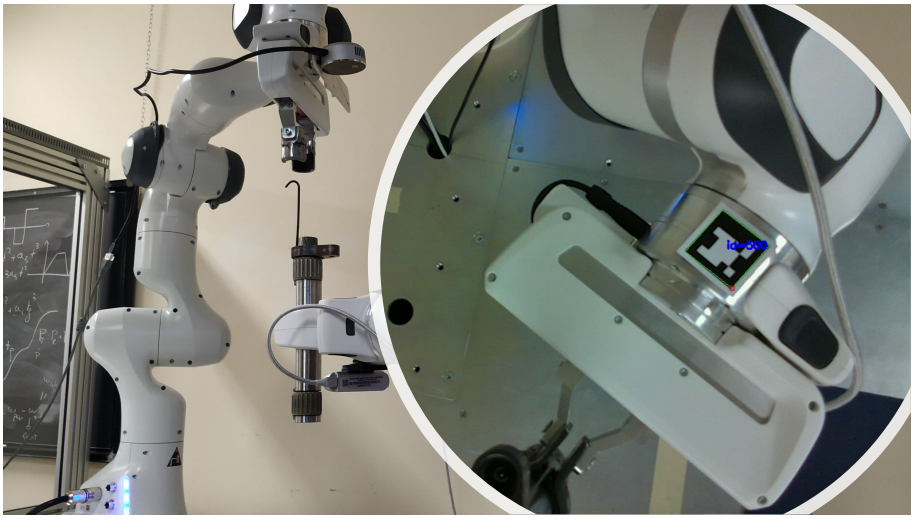


Figure 3.23: Marker detection performed by the receiver robot to detect the presence of the giver one.

The receiver needs an exteroceptive sensor to detect the presence of the shaft and to align its gripper to the shaft axis. To this aim, it is equipped with a Intel Realsense L515 camera, while an Aruco marker [170] is positioned on the giver robot (Fig. 3.23). In order to avoid collisions, the receiver can start its motion only when the giver reaches the exchange point. Therefore, the camera detects the position of the center of the marker, by using the `OpenCV` library, in two consecutive frames: if the difference between the detected positions is below a certain threshold for at least 0.5 s, the robot assumes that the giver has reached the exchange position and starts its motion. Firstly, the receiver aligns the x_e and y_e axis of its end-effector reference frame (Fig. 3.22) to the marker frame, then, since the shaft position with respect to the marker is fixed and assumed known, moves the end-effector in order to align the z_e axis with the shaft axis. Finally, it moves along the shaft axis and stops when a contact is detected. Since wrist mounted force-torque sensors are not present, the contact detection is performed via the momentum-based observer (1.14) described in Section 1.6.

3.4.4 Contact estimation and physical exchange phase

In the contact detection, the external wrench $\hat{\boldsymbol{h}}_p$ is estimated by using the wrench observer (1.14) (see Section 1.6) and a set of dynamic parameters identified in [116]. The parameters have been suitably modified in order to take into account the contribution of the gripper to the inertia and gravity terms and, only for the giver robot, of the shafts.

Moreover, in order to suppress non-existent small force and torque estimations owing to unmodeled dynamics and sensor noise, a *dead zone* has been implemented, i.e., any estimated value of force component below 3 N and any estimated value of moment below 1 Nm are neglected. Finally, to achieve a continuous wrench signal, the same thresholds have been subtracted from all estimated values.

During the physical exchange phase, different behaviors for the giver and the receiver are required. When the receiver hits the object, a force along the shaft axis is perceived by both robots. In this phase, in order to successfully perform the handover, the giver has to keep constant its position and orientation, crucial to make possible the grasping of the receiver, while the receiver must be compliant enough to avoid large contact forces and mechanical stresses on the shaft. Then, after the receiver grasps the object, it moves upward and the giver has to compliantly follow it.

To enforce the desired behaviors to the robots, the admittance control schema described in Section 1.6 has been used, with a small change in eq. (1.18), that becomes:

$$\boldsymbol{M}_p \ddot{\boldsymbol{x}}^e_{dr} + \boldsymbol{D}_p \dot{\boldsymbol{x}}^e_{dr} + \boldsymbol{K}_p \boldsymbol{x}^e_{dr} = \bar{\boldsymbol{S}} \boldsymbol{T}_A^{\mathrm{T}}(\boldsymbol{\phi}^e_{dr}) \hat{\boldsymbol{h}}^e, \tag{3.2}$$

where $\bar{\boldsymbol{S}}$ is a 6×6 diagonal selection matrix of ones and zeros, whose (i,i) element is 0 (1) if the robot must be rigid (compliant) with respect to the i-th component of the wrench $\hat{\boldsymbol{h}}^e$. For the receiver robot, the matrix $\bar{\boldsymbol{S}}$ has been set as $\bar{\boldsymbol{S}}_r = \boldsymbol{I}_6$, while, for the giver, in order to enforce the above described behavior, it has been set as a matrix with all zeros except the element (1,1) given by

$$\bar{\boldsymbol{S}}_g(1,1) = \frac{1 - \mathrm{sgn}(\hat{f}^e_{g,x})}{2}, \tag{3.3}$$

where $\mathrm{sgn}(\cdot)$ is the sign function and $\hat{f}^e_{g,x}$ is the estimated force acting on the giver end-effector along the axis x_e, expressed in the robot end-effector frame. In other words, such a condition means that the giver robot is commanded to be compliant with respect to force along the axis $-x_e$ and rigid with respect to other forces and moments (Fig. 3.22).

Table 3.4: Controller gains.

Gain	Value
$\boldsymbol{M}_a$ eq. (3.2)	$\mathrm{diag}[15, 15, 15, 0.5, 0.5, 0.5]$
$\boldsymbol{D}_a$ eq. (3.2)	$\mathrm{diag}[30, 30, 30, 1, 1, 1]$
$\boldsymbol{K}_a$ eq. (3.2)	$\mathrm{diag}[45, 45, 45, 1.5, 1.5, 1.5]$

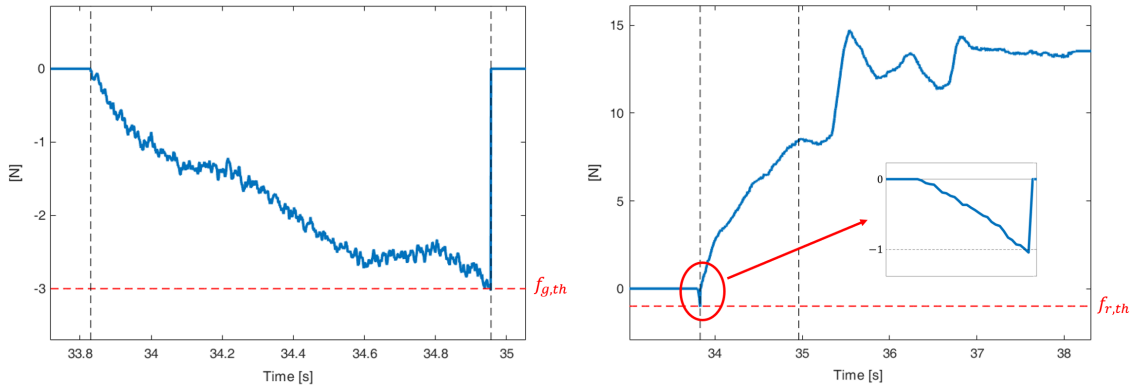


Figure 3.24: Estimated contact forces along the CSI axis: force $\hat{f}^e_{g,x}$ acting on the giver robot (a), force $\hat{f}^e_{r,z}$ acting on the receiver robot (b). Vertical dashed lines delimit the physical exchange phase. Horizontal dashed red lines represent the thresholds.

The admittance filter parameters used in this application are reported in Table 3.4. The estimated forces acting on the two robots along the CSI axis are reported in Fig. 3.24. In Fig. 3.24(a), i.e., the one relative to the force acting on the giver robot, only the time range in which the physical interaction takes place is represented. Outside this range, the estimated external force is equal to zero, since the robot is still and the weight of the shaft has been compensated within the observer.

In detail, the receiver detects the first contact when the force $\hat{f}_{r,z}$, i.e. the estimated force acting on the receiver end-effector along the axis z_e, overcomes a threshold, $f_{r,th}$, that has been set as -1 N (Fig. 3.24(b)). Once the contact is detected, the receiver closes the gripper and starts to move compliantly followed by the giver robot. During this phase, a force along the shaft axis, $\hat{f}_{g,x}$, is experienced on the giver's end-effector (Fig. 3.24(a)), it decreases until the threshold $f_{g,th} = -3$ N is reached. At the same time, the force $\hat{f}_{r,z}$ increases until reaching the gravitational force due to the shaft's mass (Fig. 3.24(b)). Once the threshold is reached, the giver opens the gripper and the object is fully released to the receiver. The results for the CSE handover are reported in Fig. 3.25. They are analogous to those showed above for the CSI shaft and have different time range on the x-axis for the same reason of the other figure.

3.4.5 Cooperation in industrial environment

The application and the experiments described in Section 3.4 represent a preliminary study that allowed the development of a setup at the Campus Manufacturing of the Cen-

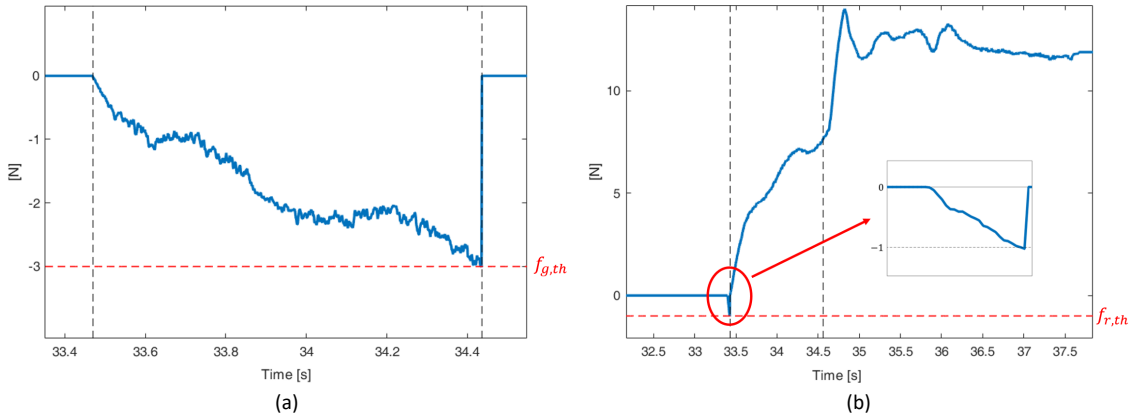


Figure 3.25: Estimated contact forces along the CSE axis: force $\hat{f}^e_{g,x}$ acting on giver robot (a), force $\hat{f}^e_{r,z}$ acting on receiver robot (b). Vertical dashed lines delimit the physical exchange phase. Horizontal dashed red lines represent the thresholds.

tro Ricerche FIAT S.C.p.A. (CRF), part of the Stellantis group, based in Melfi (PZ), Italy. The complete task involves the assembly of the two counter-rotating shafts (both the exhaust and intake) and the assembly of accessory components into an engine block. Until now, the operation is manually carried out: the human operator takes and inspects the shafts from an automated shelf. Then, she/he inserts them, by using a special tool, inside the engine block, which is located on a handling line and, subsequently, assembles the accessory components.

CRF is investigating technologies to automate this process by including two collaborative robots in the assembly process. In the pilot, the present production line has been split in two lines, made automatic through two Automated Guided Vehicles (AGVs). In particular, a logistics line is used for transporting the components to be assembled on the engine (counter-rotating shafts and accessory elements) and a main line for transporting the engine block.

Two collaborative robots create a decoupling between the logistics line, i.e. the AGV that transport the components kit, and the assembly line. In addition to monitoring the whole execution through an interactive screen, the operator is in charge of inspecting the counter-rotating shafts and picking and assembling accessory components into the engine. In this way, a collaborative space-sharing area between the operator and the robots and a cooperation area between the robots are created. The final setup with the two collaborative robots and the two AGVs is shown in Fig. 3.26.

For security reasons, in this setup, the exchange takes place with an explicit communi-

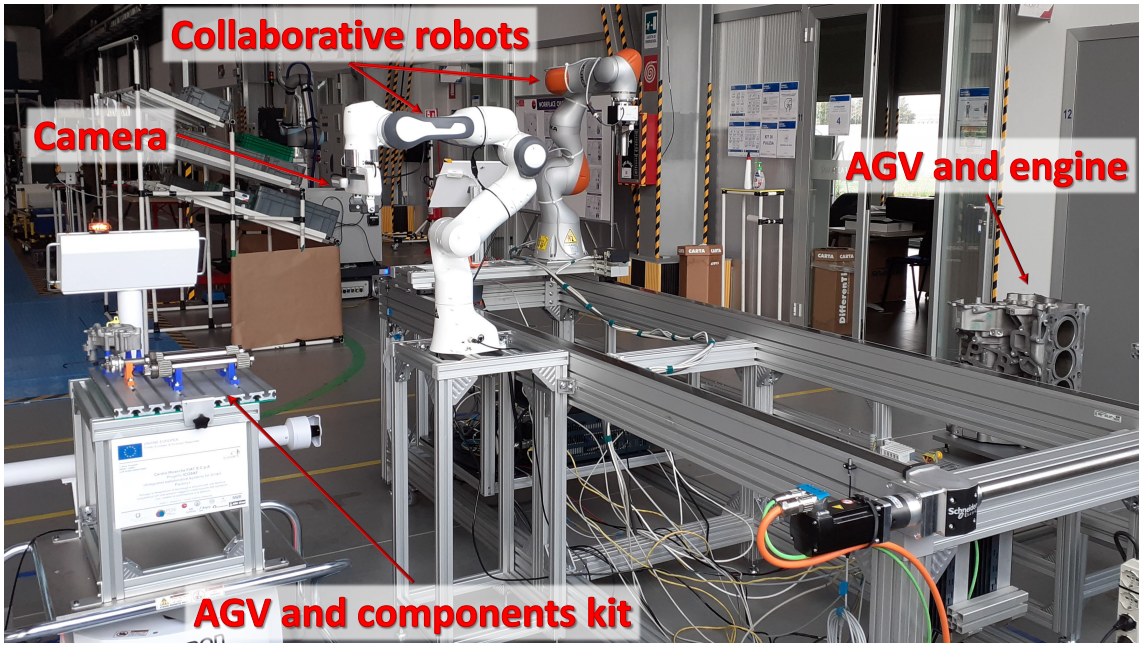
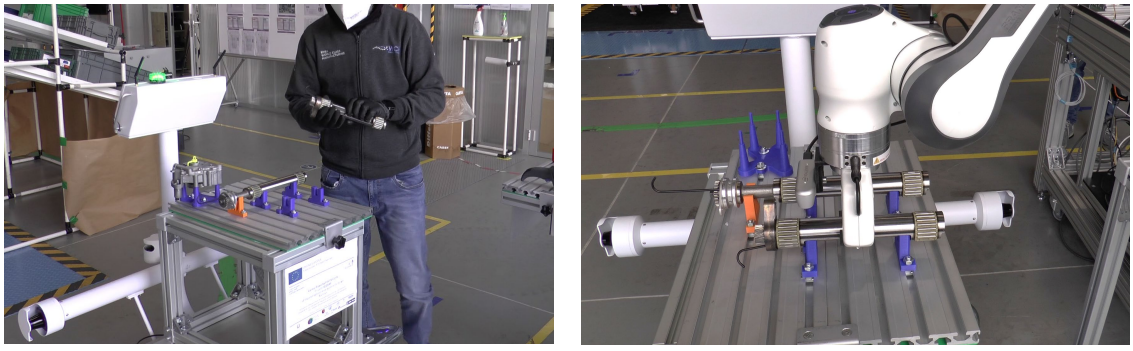


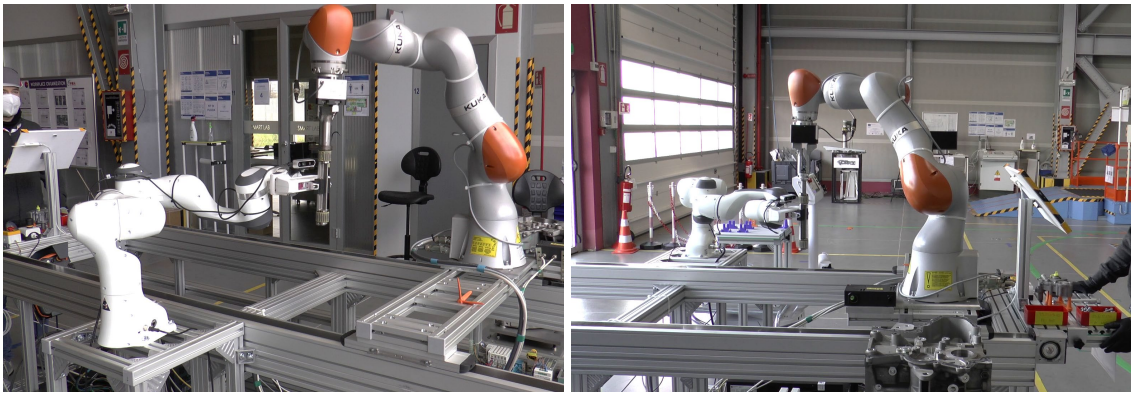
Figure 3.26: Industrial setup for shafts assembly into an engine block.

cation between the agents. In particular, a client-server architecture model (with TCP/IP and UDP sockets) has been implemented on a specific workstation for each agent and on a workstation (leader), which is in charge of receiving messages from all agents and sending the indications for the next operations they must execute. Some operations are blocking for the whole system, e.g., if the AGV with the components kit does not arrive in its final position or if the first shatf is not taken, the leader commands the agents not directly connected with these operations to keep waiting. Other operations are performed in parallel, e.g., the insertion of the first shaft into the engine and the grasp of the second



(a) (b)

Figure 3.27: Counter-rotating shafts inspection (a) and shaft grasping performed by one of the collaborative robots (b).



(a) (b)

Figure 3.28: Exchange of the exhaust (a) and intake (b) shaft between the two robots.

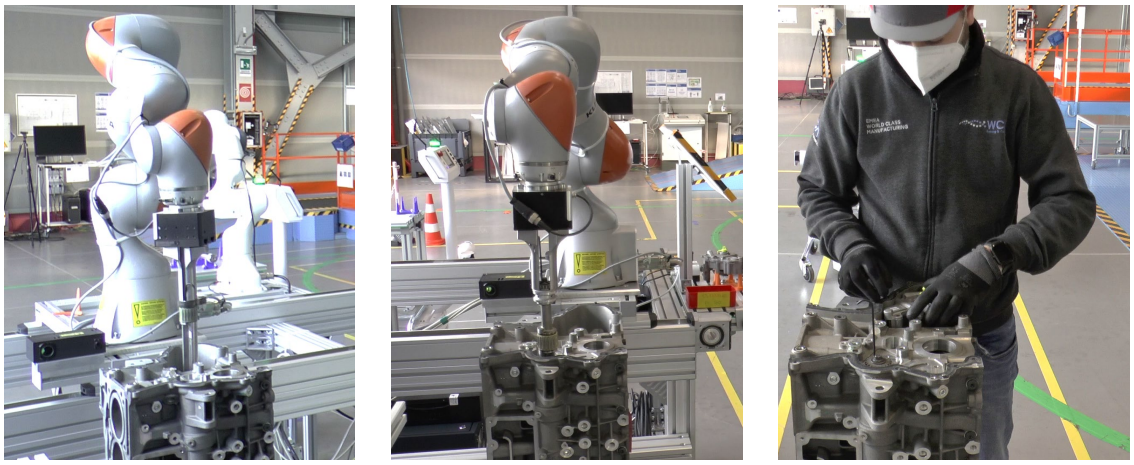
shaft from the components kit.

Some snapshots of the whole task execution are shown in Figs. 3.27-3.29. When the AGV that transport the components kit arrives in the area, the first step is executed by the operator that inspects the shafts and scans the code on the objects (Fig. 3.27(a)). Then, the AGV enters in the field of view of the robot, that detects the shafts and grasps one of them (Fig. 3.27(b)).

The physical interactions between the two robots to accomplish the shaft exchange are shown in Fig. 3.28. As it can be seen, the position of the robot that receives the shaft is different for the two exchanges, since it is mounted on a sled. For this reason, the robot needs to compute the relative position between the engine and itself and this operation is made by using a profilometer set on the sled. This device measures the engine profile with a laser and it is able to position the robot in a certain pose with respect to the engine. In this way, the robots know exactly where are the holes and can insert the shafts (Fig. 3.29(a)-(b)) and, finally, the operator can assembly the accessory components (Fig. 3.29(c)).

3.5 Conclusion

In this Chapter the object grasping problem has been introduced and some methodologies to address it have been presented and analyzed. The first consists of a real-time and robust approach for detecting and grasping different objects. This method exploits the background subtraction technique in order to remove the background and localize the objects. Then, a clustering algorithm applied to the depth data provided by the camera, allows to find the grasp pose. Background subtraction is usually used to detect moving



(a) (b) (c)

Figure 3.29: Insertion of the exhaust (a) and intake (b) shaft and accessory components assembly performed by the operator (c).

objects in intelligent video surveillance applications, but in this Thesis it is exploited in an industrial scenario, where modeling the background instead of the objects can lead to a shorter off-line planning phase. Experiments on two objects, with similar features, demonstrate the feasibility and the effectiveness of the described approach.

Another investigated method consists of a comparison between a point cloud of the object and a model, built from a set of point clouds previously acquired. The experiments, conducted on a set of mechanical workpieces used in automotive factories, show that the method is applicable in case of objects with particular shapes, but not in the case of objects with symmetric shape. Camera features influence the overall performance: a more accurate sensor could allow to build a more detailed model to improve the performance and robustness of the approach.

Moreover, the object grasping problem has been analyzed in the context of a complex application. In particular, an autonomous execution of robot-to-robot object handover task in a partially structured environment and in the absence of explicit communication between the robots has been developed and validated. The proposed approach requires only visual and joint torque sensors and can be easily extended to industrial scenarios for flexible production. Visual measures are adopted for detecting the presence of the object both in the giver and in the receiver robot workspace, while an observer which exploits the joint torque measurements is adopted for modulating the grip force of the two robots. For object detection a CNN-based approach has been chosen, which allows to apply the strategy to different objects by extending the classes detected by the CNN.

In the last Section, an industrial integration of the shaft exchange application has been described. Future work can be devoted to extend the approach to mobile manipulators.

Conclusions and future work

In the era of *Industry 4.0*, the adoption of human-robot and robot-robot collaboration represents the new frontier for industries. The use of collaborative robots (*cobots*), that are smaller than traditional industrial ones and easily reprogrammable, represents an effective solution for companies that often tackle changes in the production mix. Cobots allow the direct interaction with the human operators and are designed to work in unstructured environments by leveraging on learning capabilities. Making robots autonomous in their activities is still an open challenge and, in this Thesis, strategies for collaborative and cooperative robotic applications have been proposed, which use advanced control and vision techniques to confer robots a higher degree of autonomy.

First, advanced vision and control techniques have been studied and exploited in the execution of a Peg-in-Hole task in partially structured environments. In particular, vision techniques have been used to get information about the environment, e.g., where the workpiece is located, while control techniques have been used to handle the interaction with the workpiece.

Then, two solutions for the Peg-In-Hole assembly task, by means of a collaborative robotic arm, in the presence of large uncertainties on the relative pose of the workpiece with respect to the robot's base, have been developed. The first strategy represents a preliminary study of an accurate blind method, in which the workpiece surface is reconstructed with a sub-millimeter accuracy. An admittance control scheme is used to control the robot and confer compliance to the peg tip. This technique is very accurate, but difficult to automate, and the performance strongly depends on the quality of the pattern on the surface and the illumination. The second one is a completely autonomous strategy; experimental results confirm the effectiveness of the approach even with wide object initial positioning errors. This strategy is less accurate than the first one, but the addition of a search phase makes the procedure more robust. The use of deep neural networks improves the holes' localization and reduces the failures.

Then, the problem of object grasping has been tackled by adopting visual approaches. One of these approaches is based on the background subtraction technique, usually used in intelligent video surveillance and traffic monitoring applications. Experiments on two objects, with similar features, have demonstrated the feasibility and the effectiveness of the described approach. Another investigated method consists of a comparison between a point cloud of the object and a model built from a set of point clouds previously acquired. The experiments show that the method is applicable in case of objects with particular shapes, but not in the case of objects with symmetric shape. Finally, a complete solution for a robot-to-robot object handover application has been described. A robot needs to pass two counter-rotating shafts to another robot without any explicit communication, i.e., the robots only use their on-board sensors. Vision techniques are used to compute the shafts pose and start the exchange phase, while control techniques are used to handle the physical interaction with the objects.

In conclusion, the main contributions of this Thesis are:

- the analysis of visual processing techniques to evaluate their performance and their applicability to industrial tasks;
- the application of visual techniques, for the first time, to solve a class of manufacturing tasks by adopting low-cost sensors;
- the integration of control techniques with the visual ones to achieve an autonomous execution of industrial tasks as well as collaboration and cooperation between the robots and/or the humans;
- demonstration of the developed approaches on real industrial setups.

For the Peg-in-Hole task, future work will be aimed at continuing the development of the blind method, looking for solutions to overcome the limits imposed by the type of surface and lighting. For the other insertion method, more experiments could be conducted with different workpieces and with different background to test the robustness of the image segmentation network. As for object grasping, future work will be devoted to extend the developed method to objects characterized by different grasp pose candidates and randomly placed, while the adoption of mobile manipulators can be explored for object handover tasks.

Moreover, mobile robots could be involved in different cooperative and collaborative tasks and the developed techniques will be adapted and used.

Bibliography

[1] I. Karabegović, "Comparative analysis of automation of production process with industrial robots in asia/australia and europe," *International Journal of Human Capital in Urban Management*, vol. 2, no. 1, pp. 29–38, 2017.

[2] A. Weiss, A.-K. Wortmeier, and B. Kubicek, "Cobots in industry 4.0: A roadmap for future practice studies on human–robot collaboration," *IEEE Transactions on Human-Machine Systems*, vol. 51, no. 4, pp. 335–345, 2021.

[3] A. De Luca and F. Flacco, "Integrated control for phri: Collision avoidance, detection, reaction and collaboration," in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, 2012, pp. 288–295.

[4] S. Haddadin, A. Albu-Schaffer, A. De Luca, and G. Hirzinger, "Collision detection and reaction: A contribution to safe physical human-robot interaction," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 3356–3363.

[5] B. Vanderborght, A. Albu-Schäffer, A. Bicchi, E. Burdet, D. G. Caldwell, R. Carloni, M. Catalano, O. Eiberger, W. Friedl, G. Ganesh *et al.*, "Variable impedance actuators: A review," *Robotics and autonomous systems*, vol. 61, no. 12, pp. 1601–1614, 2013.

[6] U. E. Ogenyi, J. Liu, C. Yang, Z. Ju, and H. Liu, "Physical human–robot collaboration: Robotic systems, learning methods, collaborative strategies, sensors, and actuators," *IEEE transactions on cybernetics*, vol. 51, no. 4, pp. 1888–1901, 2019.

[7] R. Meena, K. Jokinen, and G. Wilcock, "Integration of gestures and speech in human-robot interaction," in *2012 IEEE 3rd International Conference on Cognitive Infocommunications (CogInfoCom)*. IEEE, 2012, pp. 673–678.

[8] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, 2018.

[9] F. Sherwani, M. M. Asad, and B. S. K. K. Ibrahim, "Collaborative robots and industrial revolution 4.0 (ir 4.0)," in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*. IEEE, 2020, pp. 1–5.

[10] R. J. Jacob, A. Girouard, L. M. Hirshfield, M. S. Horn, O. Shaer, E. T. Solovey, and J. Zigelbaum, "Reality-based interaction: a framework for post-wimp interfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2008, pp. 201–210.

[11] V. Villani, F. Pini, F. Leali, C. Secchi, and C. Fantuzzi, "Survey on human-robot interaction for robot programming in industrial applications," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 66–71, 2018.

[12] T. Dietz, U. Schneider, M. Barho, S. Oberer-Treitz, M. Drust, R. Hollmann, and M. Hägele, "Programming system for efficient use of industrial robots for deburring in sme environments," in *ROBOTIK 2012; 7th German Conference on Robotics*. VDE, 2012, pp. 1–6.

[13] G. Ferretti, G. Magnani, and P. Rocco, "Assigning virtual tool dynamics to an industrial robot through an admittance controller," in *2009 International Conference on Advanced Robotics*. IEEE, 2009, pp. 1–6.

[14] T. Abbas and B. A. MacDonald, "Generalizing topological task graphs from multiple symbolic demonstrations in programming by demonstration (pbd) processes," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3816–3821.

[15] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 2. IEEE, 2002, pp. 1398–1403.

[16] P. Neto, J. N. Pires, and A. P. Moreira, "High-level programming and control for industrial robotics: using a hand-held accelerometer-based input device for gesture and posture recognition," *Industrial Robot: An International Journal*, 2010.

[17] V. Villani, L. Sabattini, G. Riggio, C. Secchi, M. Minelli, and C. Fantuzzi, "A natural infrastructure-less human–robot interaction system," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1640–1647, 2017.

[18] R. Poppe, "A survey on vision-based human action recognition," *Image and vision computing*, vol. 28, no. 6, pp. 976–990, 2010.

[19] S. C. Akkaladevi and C. Heindl, "Action recognition for human robot interaction in industrial applications," in *2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*. IEEE, 2015, pp. 94–99.

[20] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 338–345.

[21] A. Mohammed, B. Schmidt, and L. Wang, "Active collision avoidance for human–robot collaboration driven by vision sensors," *International Journal of Computer Integrated Manufacturing*, vol. 30, no. 9, pp. 970–980, 2017.

[22] H. Zhang, H. Chen, N. Xi, G. Zhang, and J. He, "On-line path generation for robotic deburring of cast aluminum wheels," in *2006 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2006, pp. 2400–2405.

[23] R. Nogales, F. Mayorga, and J. Vargas, "A proposal for hand gesture control applied to the kuka youbot using motion tracker sensors and machine learning algorithms," in *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2022, pp. 1–4.

[24] https://www.franka.de/.

[25] N. Calzone, M. Sileo, R. Mozzillo, F. Pierri, and F. Caccavale, "Mixed reality platform supporting human-robot interaction," in *Advances on Mechanics, Design Engineering and Manufacturing IV*, S. Gerbino, A. Lanzotti, M. Martorelli, R. Mirálbes Buil, C. Rizzi, and L. Roucoules, Eds. Cham: Springer International Publishing, 2023, pp. 1172–1182.

[26] F. Regal, C. Petlowany, C. Pehlivanturk, C. Van Sice, C. Suarez, B. Anderson, and M. Pryor, "Augre: Augmented robot environment to facilitate human-robot teaming and communication," in *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2022, pp. 800–805.

[27] D. Puljiz, E. Stöhr, K. S. Riesterer, B. Hein, and T. Kröger, "Sensorless hand guidance using microsoft hololens," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2019, pp. 632–633.

[28] A. De Franco, E. Lamon, P. Balatti, E. De Momi, and A. Ajoudani, "An intuitive augmented reality interface for task scheduling, monitoring, and work performance improvement in human-robot collaboration," in *2019 IEEE International Work Conference on Bioinspired Intelligence (IWOBI)*. IEEE, 2019, pp. 75–80.

[29] R. Bridger, *Introduction to human factors and ergonomics*. CRC press, 2017.

[30] S. Panagou, M. Sileo, K. Papoutsakis, F. Fruggiero, A. Qammaz, and A. Argyros, "Complexity based investigation in collaborative assembly scenarios via non intrusive techniques," in *2022 International Conference on Industry 4.0 and Smart Manufacturing*, 2022.

[31] P. Sánchez-Sánchez and M. A. Arteaga-Pérez, "Cooperative robots," in *Artificial Intelligence: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2017, pp. 2920–2973.

[32] F. Pierri, M. Nigro, G. Muscio, and F. Caccavale, "Cooperative manipulation of an unknown object via omnidirectional unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3, pp. 1635–1649, 2020.

[33] J. Leitner, "Multi-robot cooperation in space: A survey," *2009 Advanced Technologies for Enhanced Quality of Life*, pp. 144–151, 2009.

[34] J. E. Inglett and E. J. Rodríguez-Seda, "Object transportation by cooperative robots," in *SoutheastCon 2017*. IEEE, 2017, pp. 1–6.

[35] A. C. Lehman, K. A. Berg, J. Dumpert, N. A. Wood, A. Q. Visty, M. E. Rentschler, S. R. Platt, S. M. Farritor, and D. Oleynikov, "Surgery with cooperative robots," *Computer Aided Surgery*, vol. 13, no. 2, pp. 95–105, 2008.

[36] K. Takase, "The design of an articulated manipulator with torque control ability," in *Proc. 4th Int. Symp. on Industrial Robots, Tokyo*, 1974.

[37] S. Fujii and S. Kurono, "Coordinated computer control of a pair of manipulators," in *Proceeding of 4th IFToMM World Congress*, 1975, pp. 411–417.

[38] L. Villani and J. De Schutter, *Force Control*. Cham: Springer International Publishing, 2016, pp. 195–220. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_9

[39] C. Della Santina, M. G. Catalano, and A. Bicchi, *Soft Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–15. [Online]. Available: https://doi.org/10.1007/978-3-642-41610-1_146-2

[40] T. L. D. Fazio, D. S. Seltzer, and D. E. Whitney, "The instrumented remote centre compliance," *Industrial Robot-an International Journal*, vol. 11, pp. 238–242, 1984.

[41] H. Neville, "Impedance control: An approach to manipulation: Part i˜ iii," *Trans. of ASME Journal of Dynamic System, Measurement, and Control*, vol. 107, pp. 1–24, 1985.

[42] H. Kazerooni, T. Sheridan, and P. Houpt, "Robust compliant motion for manipulators, part i: The fundamental concepts of compliant motion," *IEEE Journal on Robotics and Automation*, vol. 2, no. 2, pp. 83–92, 1986.

[43] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems*, vol. 156, p. 104224, 2022.

[44] A.-N. Sharkawy, P. N. Koustournpardis, and N. Aspragathos, "Variable admittance control for human-robot collaboration based on online neural network training," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1334–1339.

[45] F. Ferraguti, C. Talignani Landi, L. Sabattini, M. Bonfè, C. Fantuzzi, and C. Secchi, "A variable admittance control strategy for stable physical human–robot interaction," *The International Journal of Robotics Research*, vol. 38, no. 6, pp. 747–765, 2019.

[46] A. Sidiropoulos, Y. Karayiannidis, and Z. Doulgeri, "Human-robot collaborative object transfer using human motion prediction based on cartesian pose dynamic movement primitives," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 3758–3764.

[47] H. Park, J.-H. Bae, J.-H. Park, M.-H. Baeg, and J. Park, "Intuitive peg-in-hole assembly strategy with a compliant manipulator," in *IEEE ISR 2013*. IEEE, 2013, pp. 1–5.

[48] Y.-L. Kim, B.-S. Kim, and J.-B. Song, "Hole detection algorithm for square peg-in-hole using force-based shape recognition," in *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2012, pp. 1074–1079.

[49] H.-C. Song, Y.-L. Kim, and J.-B. Song, "Guidance algorithm for complex-shape peg-in-hole strategy based on geometrical information and force control," *Advanced Robotics*, vol. 30, no. 8, pp. 552–563, 2016.

[50] F. Caccavale, C. Natale, B. Siciliano, and L. Villani, "Control of two industrial robots for parts mating," in *Proceedings of the 1998 IEEE International Conference on Control Applications (Cat. No. 98CH36104)*, vol. 1. IEEE, 1998, pp. 562–566.

[51] H. Park, P. K. Kim, J.-H. Bae, J.-H. Park, M.-H. Baeg, and J. Park, "Dual arm peg-in-hole assembly with a programmed compliant system," in *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE, 2014, pp. 431–433.

[52] S. Huang, K. Murakami, Y. Yamakawa, T. Senoo, and M. Ishikawa, "Fast peg-and-hole alignment using visual compliance," in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 286–292.

[53] R.-J. Chang, C. Lin, and P. Lin, "Visual-based automation of peg-in-hole microassembly process," *Journal of manufacturing science and engineering*, vol. 133, no. 4, 2011.

[54] P. Nagarajan, S. S. Perumaal, and B. Yogameena, "Vision based pose estimation of multiple peg-in-hole for robotic assembly," in *International conference on computer vision, graphics, and image processing*. Springer, 2016, pp. 50–62.

[55] Z. Yang, W. Liu, H. Li, and Z. Li, "A coaxial vision assembly algorithm for un-centripetal holes on large-scale stereo workpiece using multiple-dof robot," in *2018 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2018, pp. 1–6.

[56] H. Bruyninckx, S. Dutre, and J. De Schutter, "Peg-on-hole: a model based solution to peg and hole alignment," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1995, pp. 1919–1924.

[57] W. S. Newman, Y. Zhao, and Y.-H. Pao, "Interpretation of force and moment signals for compliant peg-in-hole assembly," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 1. IEEE, 2001, pp. 571–576.

[58] K. Sharma, V. Shirwalkar, and P. K. Pal, "Intelligent and environment-independent peg-in-hole search strategies," in *2013 International Conference on Control, Automation, Robotics and Embedded Systems (CARE)*. IEEE, 2013, pp. 1–6.

[59] H. Park, J. Park, D.-H. Lee, J.-H. Park, M.-H. Baeg, and J.-H. Bae, "Compliance-based robotic peg-in-hole assembly strategy without force feedback," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 8, pp. 6299–6309, 2017.

[60] M. Jokesch, J. Suchỳ, A. Winkler, A. Fross, and U. Thomas, "Generic algorithm for peg-in-hole assembly tasks for pin alignments with impedance controlled robots," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 105–117.

[61] J. C. Triyonoputro, W. Wan, and K. Harada, "Quickly inserting pegs into uncertain holes using multi-view images and deep network trained on synthetic data," *arXiv preprint arXiv:1902.09157*, 2019.

[62] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8943–8950.

[63] G. De Magistris, A. Munawar, T.-H. Pham, T. Inoue, P. Vinayavekhin, and R. Tachibana, "Experimental force-torque dataset for robot learning of multi-shape insertion," *arXiv preprint arXiv:1807.06749*, 2018.

[64] A. Y. Yasutomi, H. Mori, and T. Ogata, "A peg-in-hole task strategy for holes in concrete," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2205–2211.

[65] M. A. Sutton, J. J. Orteu, and H. Schreier, *Image correlation for shape, motion and deformation measurements: basic concepts, theory and applications*. Springer Science & Business Media, 2009.

[66] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[67] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE, 2001, pp. 145–152.

[68] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5556–5565.

[69] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics – Modelling, Planning and Control*. London, UK: Springer, 2009.

[70] A. Kaufman, D. Cohen, and R. Yagel, "Volume graphics," *Computer*, vol. 26, no. 7, pp. 51–64, 1993.

[71] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[72] R. Susmaga, "Confusion matrix visualization," in *Intelligent information processing and web mining*. Springer, 2004, pp. 107–116.

[73] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[74] A. Zafar, M. Aamir, N. Mohd Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta, and S. Almotairi, "A comparison of pooling methods for convolutional neural networks," *Applied Sciences*, vol. 12, no. 17, p. 8643, 2022.

[75] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*. Springer, 2014, pp. 364–375.

[76] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015.

[77] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[78] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[79] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[80] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[81] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, "Fundamental concepts of convolutional neural network," in *Recent trends and advances in artificial intelligence and Internet of Things*. Springer, 2020, pp. 519–567.

[82] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[83] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[84] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[85] A. I. B. Parico and T. Ahamed, "Real time pear fruit detection and counting using yolov4 models and deep sort," *Sensors*, vol. 21, no. 14, p. 4803, 2021.

[86] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, and K. Michael, "ultralytics/yolov5: v6. 1-tensorrt, tensorflow edge tpu and openvino export and inference," *Zenodo, Feb*, vol. 22, 2022.

[87] U. Nepal and H. Eslamiat, "Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs," *Sensors*, vol. 22, no. 2, p. 464, 2022.

[88] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[89] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[90] F. Jiang, A. Grigorev, S. Rho, Z. Tian, Y. Fu, W. Jifara, K. Adil, and S. Liu, "Medical image semantic segmentation based on deep learning," *Neural Computing and Applications*, vol. 29, no. 5, pp. 1257–1265, 2018.

[91] N. Alalwan, A. Abozeid, A. A. ElHabshy, and A. Alzahrani, "Efficient 3d deep learning model for medical image semantic segmentation," *Alexandria Engineering Journal*, vol. 60, no. 1, pp. 1231–1239, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110016820305639

[92] M. Gruosso, N. Capece, and U. Erra, "Exploring Upper Limb Segmentation with Deep Learning for Augmented Virtuality," in *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, P. Frosini, D. Giorgi, S. Melzi, and E. RodolÃ , Eds. The Eurographics Association, 2021.

[93] ——, "Egocentric upper limb segmentation in unconstrained real-life scenarios," Oct 2021. [Online]. Available: https://www.techrxiv.org/articles/preprint/Egocentric_Upper_Limb_Segmentation_in_Unconstrained_Real-Life_Scenarios/16839997/1

[94] D. Feng, C. Haase-SchÃ¼tz, L. Rosenbaum, H. Hertlein, C. GlÃ¤ser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2021.

[95] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, M. Jagersand, and H. Zhang, "A comparative study of real-time semantic segmentation for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[96] C. Yin, B. Wang, V. J. Gan, M. Wang, and J. C. Cheng, "Automated semantic segmentation of industrial point clouds using respointnet++," *Automation in Construction*, vol. 130, p. 103874, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580521003253

[97] Y. Xie, J. Tian, and X. X. Zhu, "Linking points with labels in 3d: A review of point cloud semantic segmentation," *IEEE Geoscience and Remote Sensing Magazine*, vol. 8, no. 4, pp. 38–59, 2020.

[98] I. Alonso, L. Riazuelo, and A. C. Murillo, "Mininet: An efficient semantic segmentation convnet for real-time robotic applications," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1340–1347, 2020.

[99] A. Milioto and C. Stachniss, "Bonnet: An open-source training and deployment framework for semantic segmentation in robotics using cnns," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7094–7100.

[100] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[101] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[102] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[103] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets*. Springer, 1990, pp. 286–297.

[104] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/beda24c1e1b46055dff2c39c98fd6fc1-Paper.pdf

[105] H. Harkat, J. Nascimento, and A. Bernardino, "Fire segmentation using a deeplabv3+ architecture," in *Image and Signal Processing for Remote Sensing XXVI*, vol. 11533. International Society for Optics and Photonics, 2020, p. 115330M.

[106] J. Wang and X. Liu, "Medical image recognition and segmentation of pathological slices of gastric cancer based on deeplab v3+ neural network," *Computer Methods and Programs in Biomedicine*, p. 106210, 2021.

[107] W. Wu, J. Gan, J. Zhou, and J. Wang, "A lightweight and effective semantic segmentation network for ethnic clothing images based on deeplab," in *2021 9th International Conference on Communications and Broadband Networking*, 2021, pp. 34–40.

[108] Y. Kong, Y. Liu, B. Yan, H. Leung, and X. Peng, "A novel deeplabv3+ network for sar imagery semantic segmentation based on the potential energy loss function of gibbs distribution," *Remote Sensing*, vol. 13, no. 3, p. 454, 2021.

[109] A. De Luca and R. Mattone, "Sensorless robot collision detection and hybrid force/motion control," in *2005 IEEE Int.Conf. on Robotics and Automation*, 2005, pp. 999–1004.

[110] F. Ficuciello, A. Romano, L. Villani, and B. Siciliano, "Cartesian impedance control of redundant manipulators for human-robot co-manipulation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2120–2125.

[111] F. Caccavale, C. Natale, B. Siciliano, and L. Villani, "Six-dof impedance control based on angle/axis representations," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 2, pp. 289–300, 1999.

[112] M. Nigro, M. Sileo, F. Pierri, K. Genovese, D. D. Bloisi, and F. Caccavale, "Peg-in-hole using 3d workpiece reconstruction and cnn-based hole detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4235–4240.

[113] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[114] B. Boots, K. Sugihara, S. N. Chiu, and A. Okabe, "Spatial tessellations: concepts and applications of voronoi diagrams," 2009.

[115] F. Arrichiello, S. Chiaverini, G. Indiveri, and P. Pedone, "The null-space based behavioral control for mobile robots with velocity actuator saturations," *International Journal of Robotics Research*, vol. 29, no. 10, pp. 1317–1337, 2010.

[116] C. Gaz, M. Cognetti, A. Oliva, P. R. Giordano, and A. De Luca, "Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4147–4154, 2019.

[117] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 3212–3217.

[118] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Persistent point feature histograms for 3d point clouds," in *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, 2008, pp. 119–128.

[119] H. Li, J. Qin, X. Xiang, L. Pan, W. Ma, and N. N. Xiong, "An efficient image matching algorithm based on adaptive threshold and ransac," *IEEE Access*, vol. 6, pp. 66 963–66 971, 2018.

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

[121] R. Y. Tsai, R. K. Lenz *et al.*, "A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration," *IEEE Transactions on robotics and automation*, vol. 5, no. 3, pp. 345–358, 1989.

[122] É. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics & Automation Magazine*, vol. 12, no. 4, pp. 40–52, 2005.

[123] https://www.rhino3d.com/it/.

[124] G. Papandreou, I. Kokkinos, and P.-A. Savalle, "Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 390–399.

[125] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[126] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[127] S. I. Nikolenko *et al.*, "Synthetic data for deep learning," *arXiv preprint arXiv:1909.11512*, vol. 3, 2019.

[128] M. Gruosso, N. Capece, and U. Erra, "Human segmentation in surveillance video with deep learning," *Multimedia Tools and Applications*, vol. 80, no. 1, pp. 1175–1199, 2021.

[129] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[130] N. Capece, F. Banterle, P. Cignoni, F. Ganovelli, R. Scopigno, and U. Erra, "Deepflash: Turning a flash selfie into a studio portrait," *Signal Processing: Image Communication*, vol. 77, pp. 28–39, 2019.

[131] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[132] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," *arXiv preprint arXiv:1506.04579*, 2015.

[133] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

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

[135] https://github.com/tensorflow/models/tree/master/research/deeplab.

[136] http://www.tta-adler.it/it/.

[137] R. Caccavale and A. Finzi, "A robotic cognitive control framework for collaborative task execution and learning," *Topics in Cognitive Science*, vol. 14, no. 2, pp. 327–343, 2022.

[138] J. Cacace, R. Caccavale, A. Finzi, and R. Grieco, "Combining human guidance and structured task execution during physical human–robot collaboration," *Journal of Intelligent Manufacturing*, pp. 1–15, 2022.

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

[140] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 348–353.

[141] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis–a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2013.

[142] H. Zhang, J. Tan, C. Zhao, Z. Liang, L. Liu, H. Zhong, and S. Fan, "A fast detection and grasping method for mobile manipulator based on improved faster r-cnn," *Industrial Robot: the international journal of robotics research and application*, 2020.

[143] D. Morrison, P. Corke, and J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," in *Robotics: Science and Systems (RSS)*, 2018.

[144] ——, "Learning robust, real-time, reactive robotic grasping," *The International journal of robotics research*, vol. 39, no. 2-3, pp. 183–201, 2020.

[145] C. Dune, E. Marchand, C. Collowet, and C. Leroux, "Active rough shape estimation of unknown objects," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 3622–3627.

[146] D. Kraft, N. Pugeault, E. Başeski, M. POPOVIĆ, D. Kragić, S. Kalkan, F. Wörgötter, and N. Krüger, "Birth of the object: Detection of objectness and extraction of object shape through object–action complexes," *International Journal of Humanoid Robotics*, vol. 5, no. 02, pp. 247–265, 2008.

[147] R. Detry, C. H. Ek, M. Madry, J. Piater, and D. Kragic, "Generalizing grasps across partly similar objects," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3791–3797.

[148] H. Dang and P. K. Allen, "Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1311–1317.

[149] M. Cristani, M. Farenzena, D. D. Bloisi, and V. Murino, "Background subtraction for automated multisensor surveillance: A comprehensive review," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, pp. 1–24, 2010.

[150] N. A. Mandellos, I. Keramitsoglou, and C. T. Kiranoudis, "A background subtraction algorithm for detecting and tracking vehicles," *Expert Systems with Applications*, vol. 38, no. 3, pp. 1619–1631, 2011.

[151] D. D. Bloisi, A. Pennisi, and L. Iocchi, "Background modeling in the maritime domain," *Machine vision and applications*, vol. 25, no. 5, pp. 1257–1269, 2014.

[152] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger, "Review and evaluation of commonly-implemented background subtraction algorithms," in *2008 19th International Conference on Pattern Recognition*. IEEE, 2008, pp. 1–4.

[153] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *European conference on computer vision*. Springer, 2000, pp. 751–767.

[154] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3400–3407.

[155] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3d point cloud," *Signal Processing: Image Communication*, vol. 57, pp. 103–112, 2017.

[156] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[157] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 1–4.

[158] J. Kannala and S. S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 8, pp. 1335–1340, 2006.

[159] V. Ortenzi, A. Cosgun, T. Pardi, W. P. Chan, E. Croft, and D. Kulić, "Object handovers: a review for robotics," *IEEE Trans. on Robotics*, 2021.

[160] R. Koeppe, D. Engelhardt, A. Hagenauer, P. Heiligensetzer, B. Kneifel, A. Knipfer, and K. Stoddard, "Robot-robot and human-robot cooperation in commercial robotics applications," in *Robotics research. the eleventh international symposium*. Springer, 2005, pp. 202–216.

[161] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[162] A. Tsiamis, C. K. Verginis, C. P. Bechlioulis, and K. J. Kyriakopoulos, "Cooperative manipulation exploiting only implicit communication," in *2015 IEEE/RSJ Int.Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 864–869.

[163] A. H. Mason and C. L. MacKenzie, "Grip forces when passing an object to a partner," *Experimental brain research*, vol. 163, no. 2, pp. 173–187, 2005.

[164] W. P. Chan, C. A. Parker, H. M. Van der Loos, and E. A. Croft, "Grip forces and load forces in handovers: implications for designing human-robot handover controllers," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, 2012, pp. 9–16.

[165] M. Costanzo, G. De Maria, and C. Natale, "Handover control for human-robot and robot-robot collaboration," *Frontiers in Robotics and AI*, vol. 8, p. 132, 2021.

[166] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[167] https://github.com/Khaivdo/How-to-train-an-Object-Detector-using-Tensorflow-API-on-Ubuntu-16.04-GPU.

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

[169] Y. Abdel-Aziz, H. Karara, and M. Hauck, "Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 2, pp. 103–107, 2015.

[170] A. Babinec, L. Jurišica, P. Hubinskỳ, and F. Duchoň, "Visual localization of mobile robot using artificial markers," *Procedia Engineering*, vol. 96, pp. 1–9, 2014.