Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

MECCANICA E SCIENZE AVANZATE DELL'INGEGNERIA

Ciclo XXXIV

Settore Concorsuale: 09/A2

Settore Scientifico Disciplinare: ING-IND/13

VISION-BASED SOLUTIONS FOR HUMAN-ROBOT COLLABORATION IN INDUSTRIAL WORKCELLS

Presentata da: Ivo Campione

Coordinatore Dottorato

Prof. Marco Carricato

Supervisore

Prof. Marco Troncossi

Esame finale anno 2022

Abstract

Industrial robots are playing a central role within the context of Industry 4.0, enabling the flexible industrial automation typical of the modern Smart Factories. The key feature of industrial robots is that they are functionally flexible and capable of operating at high speed, repeatability and accuracy. One caveat of industrial robots, however, is that, for safety reasons, they have to be relegated inside closed fences and/or virtual safety barriers, to keep them strictly separated from human operators. This can be a limitation in some scenarios in which it can be useful to combine the human cognitive skill and know-how with the accuracy and repeatability of a robot, or simply to allow a safe coexistence in a shared workspace. To fill this gap, in the last decades a new paradigm in robotics has emerged, and it is represented by the collaborative robots, often referred to as *cobots*. Cobots are intrinsically limited in speed and power in order to share workspace and tasks with human operators. Other than that, they provide an additional and very intuitive programming method, the *hand guiding* method, which consists in programming the trajectory by manually guiding the robot through the desired waypoints. Cobots, however, come with some limitations: concerning speed and power, they cannot compete with industrial robots, and are thus useful only in a limited niche, where they can actually bring an improvement in productivity and/or in the quality of the work thanks to their synergy with human operators.

This Thesis falls inside the very recent trend of proposing methods to overcome the limitations of both the pure industrial and the collaborative paradigms, by combining industrial robots with artificial vision. This way, the so-called *human-robot collaboration* is achieved in a non-intrinsic fashion, since it does not leverage on the robot itself, but on additional external sensors. Modern vision sensors provide a detailed real-time perception of the surrounding environment that can serve to enhance the safety and the robot functional flexibility, which are key features in an environment shared simultaneously by human operators and robots. In particular, vision can be exploited for a real-time adjustment of the pre-programmed task-based robot trajectory, by means of the visual tracking of dynamic obstacles (e.g. human operators). This strategy allows the robot to modify its motion only when necessary, thus maintain a high level of productivity but at the same time increasing its versatility. Other than that, vision offers the possibility of more intuitive programming paradigms for the industrial robots as well, such as the *programming by demonstration* paradigm. These possibilities offered by artificial vision enable, as a matter of fact, an efficacious and promising way of achieving human-robot collaboration, which has the advantage of overcoming the limitations of both the previous paradigms yet keeping their strengths.

Acronyms

2D	Two-dimension
3D	Three-dimension
AR	Augmented Reality
CAD	Computer-Aided Design
CCD	Charged Coupled Device
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
DOF	Degree of Freedom
DH	Denavit-Hartemberg
FMS	Flexible Manufacturing System
FOV	Field of View
FPGA	Field Programmable Gate Array
GOT	Graphic Operator Terminal
GPU	Graphics Processing Unit
HDD	Human Demonstration Device
HG	Hand Guiding
HRC	Human-Robot Collaboration
ICP	Iterative Closest Point
IDE	Integrated Development Environment
IR	Infrared
LiDAR	Light Detection and Ranging
NUI	Natural User Interface
OCR	Optical Character Recognition
PbD	Programming by Demonstration
PCHIP	Piecewise Cubic Hermite Interpolating Polynomial
PFL	Power and Force Limiting
PLC	Programmable Logic Controller
PSD	Protective Separation Distance
QP	Quadratic Program
RF	Reference Frame
RGB	Red-Green-Blue
RGB-D	Red-Green-Blue-Depth
SMS	Safety-Rated Monitored Stop
SSM	Speed and Separation Monitoring
TCP	Tool Centre Point
ToF	Time-of-Flight
TUI	Tangible User Interface

UDP	User Datagram Protocol
URDF	Unified Robot Description Format
USB	Universal Serial Bus
VR	Virtual Reality

Contents

1 Introduction 4
1.1 Industrial vs. collaborative robots 5
1.1.1 Industrial robots 7
1.1.2 Collaborative robots 9
1.2 Artificial vision 12
1.2.1 2D cameras 14
1.2.2 3D cameras 15
1.3 Human Robot Collaboration in industrial settings by means of artificial vision 19
1.3.1 Collision avoidance 22
1.3.2 Programming by demonstration 30
1.4 Problem statement: the necessity of a new paradigm 31
2 Experimental setup 34
2.1 Robotic cell 34
2.2 External workstation and programming frameworks 37
3 Collision Avoidance 38
3.1 Online trajectory generation 41
3.1.1 RV4F robot kinematics 41
3.1.2 Pre-programmed reference trajectory 44
3.1.3 Safety constraints 47
3.1.3.1 Single robot-point 47
3.1.3.2 Multiple robot-points 50
3.1.4 Constraints on joint position, velocity, acceleration 51
3.1.4.1 Joint position constraint 51
3.1.4.2 Joint velocity constraint 51
3.1.4.3 Joint acceleration constraint 52
3.1.5 Optimization problem 53
3.1.6 Reference trajectory heading point 55
3.1.6.1 Heading point computation – laws of motion approach 55
3.1.6.2 Heading point computation – geometrical approach 55
3.2 Obstacle tracking 59
3.2.1 Control volume 60
3.2.2 Camera placement 61
3.2.3 Camera extrinsic calibration 62
3.2.3.1 Step 1: estimation of an initial transformation 63

3.2.3.2 Step 2: refinement ... 64
3.2.4 Voxel-based data fusion ... 66
3.2.4.1 Creation of a voxel grid ... 66
3.2.4.2 Filtering, voxelization, data fusion ... 68
3.2.5 Voxel types ... 72
3.2.6 Robot body filter ... 73
3.2.7 Speed estimation by particle filter ... 76
3.2.7.1 Initialization ... 77
3.2.7.2 Evolution ... 77
3.2.7.3 Measurement-based particle selection and voxel speed estimation ... 78
3.2.7.4 Resampling ... 78
3.2.8 Obstacle segmentation ... 79
3.2.9 Input quantities in the online trajectory generation block ... 81
3.3 Other features and remarks ... 82
3.3.1 Fault handling ... 82
3.3.2 Operating modalities ... 83
3.3.3 Enabling the real-time communication with the robot controller ... 84
3.4 Results ... 85
3.5 Discussion ... 97
4 Programming by demonstration ... 100
4.1 Markerless PbD method using a ToF camera ... 100
4.2 PbD using a 2D digital camera and fiducial markers ... 104
4.3 Discussion ... 114
5 Investigation on the placement of onboard cameras ... 116
5.1 Background ... 116
5.2 Materials and Methods ... 118
5.2.1 Problem simplification ... 118
5.2.2 Optimization procedure ... 119
5.3 Results ... 123
5.3.1 First type of positions and orientations discretization ... 123
5.3.2 Second type of positions and orientations discretization ... 125
5.3.3 Effect of the variation of the Field of View ... 127
5.4 Discussion ... 127
5.4.1 Notes on the computational time ... 128
5.4.2 Notes on the 3D case ... 130
5.4.3 Final remarks ... 130

Conclusions 132
Appendix A. Design of the robotic cell 134
A.1 Layout definition 134
A.2 Mechanical design 136
A.2.1 Frame 136
A.2.2 RV4F railway 137
A.2.3 Conveyor belt A 138
A.2.4 Rotary table 139
A.2.5 Conveyor belt B 139
A.2.6 Carters 140
A.3 Wiring and piping design 141
Appendix B. ROS architecture 143
Appendix C. Basic notions on camera calibration 145
Appendix D. GPU implementation of the particle filter 147
References 152

1 Introduction

Since the 60s, when the first industrial robot appeared, robots have evolved, diversified and proceeded to become more and more popular in many fields. Robotics is nowadays a well-known research branch that fuses computer science with engineering and, especially in the industrial field, robots have established as pivotal devices in a plethora of different applications. The main fields in which robots can be found are here listed:

- **Industry**: they are efficaciously used in many tasks as pick and place, palletizing, welding, painting, machining, drilling, machine tending and so on.
- **Medical field:** they aid with surgery, rehabilitation, training.
- **Military field:** mobile robots are used for bomb disposal, surveillance, transport, search, rescue, and other military operations.
- **Domestic use:** domestic robots are utilized to perform various daily tasks at home, for example for cleaning or assistance.
- **Exploration:** space (e.g. the *Mars Curiosity Rover*), deep sea (e.g. the *Ocean One*).
- **Constructions:** they can serve to aid in dangerous and/or impractical tasks which arise when constructing and doing the maintenance of building or other facilities.
- **Entertainment and other social activities:** they have also been used for entertainment purposes in various places such as amusement parks, sports etc.

The focus of this Thesis will be on the robots used in the industrial field, namely both the traditional industrial robots and the more recent collaborative industrial robots. Artificial vision, also central in this Thesis, is nowadays a powerful tool that can be used to acquire real-time detailed information about the environment, thanks to the advancements in technology and to the vast and mature research behind it. The combination of both industrial robots and artificial vision has the potential of being highly beneficial inside the context of *Human-Robot Collaboration* (HRC), which is a recent, multidisciplinary and very hot research topic that aims at breaking down the barriers (both metaphorically and literally) between human and robots, to realize a workplace with enhanced flexibility, productivity and interconnections. This Thesis gives a contribution inside this context by proposing some methods to endow

industrial robots with artificial vision, with the aim of enhancing their functional flexibility, making possible a safe coexistence with human operators and implementing some intuitive robot programming methods. The motivation and the problem statement will be better defined in Section 1.4, after having analysed some concepts concerning industrial and collaborative robots, artificial vision and HRC.

The Thesis is structured as follows:

1. Introduction: some brief notions about industrial robots, collaborative robots and artificial vision are given, by emphasizing the distinctive features of each one. Then, a state of art of industrial HRC is carried out, with particular emphasis on integration of industrial robots with artificial vision. Lastly, the aim and motivation of the Thesis is stated.

2. Experimental setup: the experimental setup is described.

3. Description of the method: the proposed methods to enable HRC are described, and concerns:

- a collision avoidance method;
- two programming by demonstration methods;
- a study on the optimal placement of cameras.

4. Conclusions: conclusions are drawn.

1.1 Industrial vs. collaborative robots

In modern industries both traditional industrial robots and collaborative robots can be found. As reported by the *International Federation of Robotics* [1] and shown in Figure 1.1, the annual operational stock of industrial robots (traditional + collaborative) is significantly rising, almost triplicating in the last decade. This high demand is due to the ongoing trend towards automation and continued technological innovation in industrial robotics.

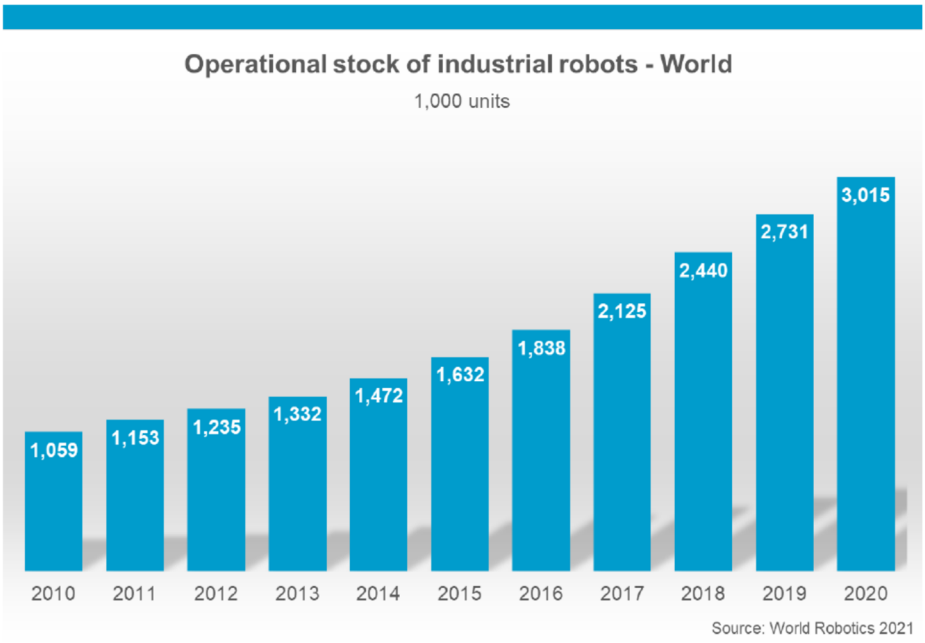


Figure 1.1 Annual operational stock of industrial (traditional + collaborative) robots [1].

Figure 1.2 shows the annual installation of traditional industrial robots and collaborative robots. One first consideration is that traditional industrial robots are the vast majority of the robots installed, although there is an increase in the percentage of collaborative robots. One second consideration is that there is a slightly drop in installation starting from 2019, which reflected the difficult times the two main customer industries, automotive and electrical/electronics, experienced.

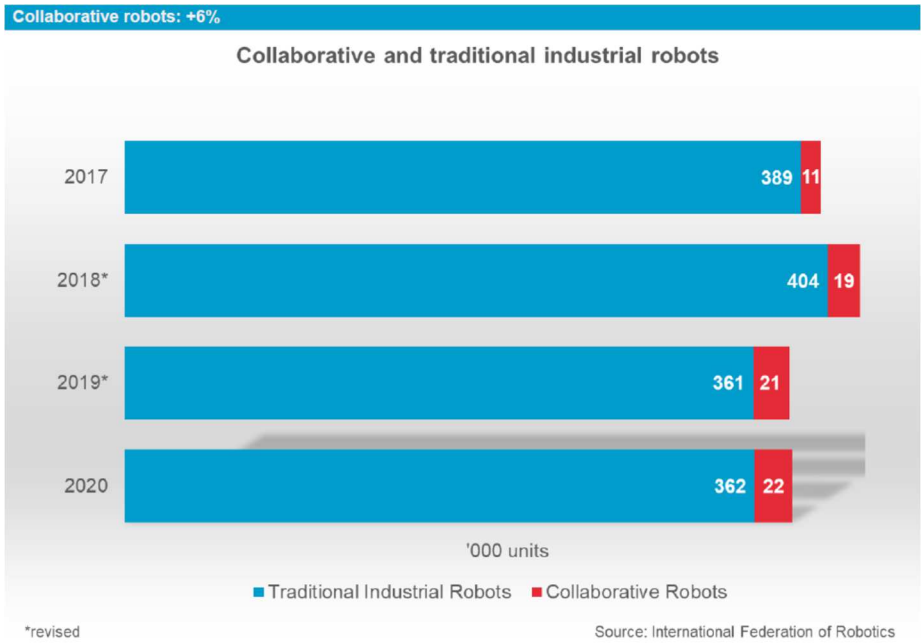


Figure 1.2 Annual installation of collaborative and traditional industrial robots [1].

As it can be seen, collaborative robots are still relegated to a limited niche. To fully understand the reason why, in the next Sections both the traditional industrial robots

and collaborative robots are described, with the aim of highlighting both their strengths and limitations. Henceforth, for sake of brevity, traditional industrial robots will be referred to as industrial robots, whereas collaborative industrial robots will be referred to as collaborative robots or simply *cobots*.

1.1.1 Industrial robots

According to *ISO 8373:2012* [2], An industrial robot is an *"automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications"*.

This definition specifies the key characteristics that classify an industrial device as a robot:

- **Having at least three axes**
- **Automatically controlled**: the robot control system operates in accordance with a task program, which is a set of instructions for motion and auxiliary functions that define the specific intended task.
- **Reprogrammable**: designed so that the programmed motions or auxiliary functions can be changed without alterations of the mechanical systems.
- **Multipurpose**: capable of being adapted to a different application without physical alteration.
- **Manipulator**: machine for the purpose of grasping and/or moving objects like pieces or tools. The end effector is not part of the manipulator
- **Fixed in place or mobile**: the robot can be mounted to a stationary point, but it can also be mounted to a non-stationary point, e.g. railways.

As it emerges, the robot and its controller form a complex and highly versatile system. The mechanical structure is composed of a series of links connected by joints, in a variable number, typically from three to six. The most common architectures that can be found on the market are the following:

- Articulated robot
- SCARA robot
- Delta robot
- Cartesian robot
- Cylindrical robot
- Spherical robot

Among all, the articulated robot is possibly the most common in industrial applications, due to its high dexterity provided by its six actuated revolute joints.

In Table 1.1 the typical features of an industrial serial robot are quantitatively summarized:

Table 1.1 Range of the main indicators of an industrial robot

Axis number	3 – 6
Payload (kg)	3 – 200
Maximum reach (mm)	500 – 3000
Maximum joint speed (deg/s)	600
Repeatability (mm)	0.01 – 0.1
Weight (kg)	30 – 1500

The ranges of values of Table 1.1 are extremely wide, especially in terms of payload, weight and maximum reach. An industrial robot can operate at high power and speed, which makes it a perfect candidate for automated tasks in which both a high productivity and a high functional flexibility are needed. On the other hand, since it operates at high speed and power, it has to comply with strictly safety requirements, and the human-robot collaboration is normally not contemplated. In fact, industrial robots are relegated inside closed fenced and/or virtual safety barrier, which have to be designed in compliance with *ISO 10218-2:2011* [3].

Industrial robots are the ideal candidates to be exploited in a system of programmable or flexible automation, that is to say in a context of production of batches of variable features, since they can easily adapt the sequence of the operations to the gamma of product variations. They can be utilized both in rigid manufacturing systems and in *Flexible Manufacturing Systems* (FMS). Here are reported some of the main applications:

- Pick and place
- Palletization
- Machine tending
- Selection and sorting
- Packaging
- Quality control and inspection

Some other applications concern manufacturing operations, such as:

- Arc welding and spot welding
- Spray painting and coating
- Bonding and sealing
- Laser cutting and water cutting
- Deburring and grinding
- Tightening, wiring and fixing
- Assembly of mechanical groups, electrical groups and electronic boards

1.1.2 Collaborative robots

Collaborative robots, often referred to as *cobots*, usually have a mechanical structure similar to the industrial articulated robot. A variation could be the presence of an additional axis (e.g. the *KUKA LBR iiwa*), generally located on the elbow, or a double arm configuration (e.g. the *ABB YuMi*). These latter configurations are kinematically redundant and allow them to execute the main task while fulfilling a secondary task, for example minimizing the occlusion of a particular workspace area or avoiding collisions by moving the elbow. This comes with the cost of a more complex design and control algorithms.
Some other distinctive features are the presence of torque and/or force sensor on the joints and/or wrist and the design without sharp edges.
Differently from industrial robots, cobots provide for the possibility of interaction with human operators, since they are designed to be intrinsically safe. Incidentally, it is to be pointed out that even if the cobots are intrinsically safe, there could be a risk depending on the application (e.g. if a dangerous tool is mounted on the robot), so an hazard identification and a risk assessment are in any case required. This intrinsic safety is realized in compliance with the technical specification *ISO/TS 15066:2016 Collaborative Robots* [4], which establishes four collaborative modalities, here briefly described (a detailed description is provided by Villani et al. in [5]):

1. **Safety-Rated Monitored Stop** (SMS): it is the simplest type of collaboration; the robot stops with a *Stop of Category 2* [6] (power is left available to the machine actuators after the movement ends) if a human operator enters the collaborative area, and resumes the cycle after the operator leaves the collaborative area. In modern industrial robots, this functionality is usually provided by means of the addition of optional safety modules.
2. **Hand Guiding** (HG): the operator can teach the robot waypoints by hand guiding the end effector through them. This collaborative scenario requires the robot to be equipped with both the *safety-rated monitored stop* and the *safety-rated monitored speed* functionality.
3. **Speed and Separation Monitoring** (SSM): it allows the human presence within the robot workspace, given that a protective separation distance is always guaranteed. This distance can be computed from data acquired through safety-rated monitoring sensors.
4. **Power and Force Limiting** (PFL): this collaborative modality allows the human to physically interact with the robot, at the cost of specific and significant limitations on the robot actuators power and exchangeable forces.

Cobots are designed to enable out-of-the-box the *1*,*2* and *4* collaborative modalities (a risk assessment, however, is in any case needed). Their light-weight mechanical design is a consequence of the necessity of fulfilling the *Point 4 - Power and Force Limiting*, which provides for the possibility of contact with the moving parts of the robot. The contacts can be intentional or non-intentional, and, based on their physical modality, can be classified as *transient* or *quasi-static*. The former refers to a short-dynamic free contact (< 500 ms) where the operator body part is not clamped and can retract from the moving part of the robot system, whereas the latter refers to cases in which the human body part is clamped for an extended time between the robot and another component. The cobots are designed so that the kinetic energy exchanged during the impact is limited and fall under specific thresholds defined in the technical specification (which vary in the case of transient or quasi-static contacts) *ISO/TS 15066:2016* [4]. The fact that the exchanged energy depends on both the mass and the speed entails the limitations on the cobots weight and speed, which are key factors in defining its safe level. When fulfilling the PFL collaborative mode, other than limiting the exchanged energy, one must also comply with limits on other related quantities, specifically the maximum force and pressure that can be exerted on the human body, as outlined in Figure 1.3 (from [4]). These biomechanical limits, based on pain thresholds and reported in [4], depend on the body part and are twice as much higher in the case of transient contacts than in the case of quasi-static contacts. It is worth recalling that since the exchanged energy, exerted force and pressure depend, besides from the robot itself, also on the mounted tool, workpiece, etc. (in general on the application), a risk assessment is always necessary.

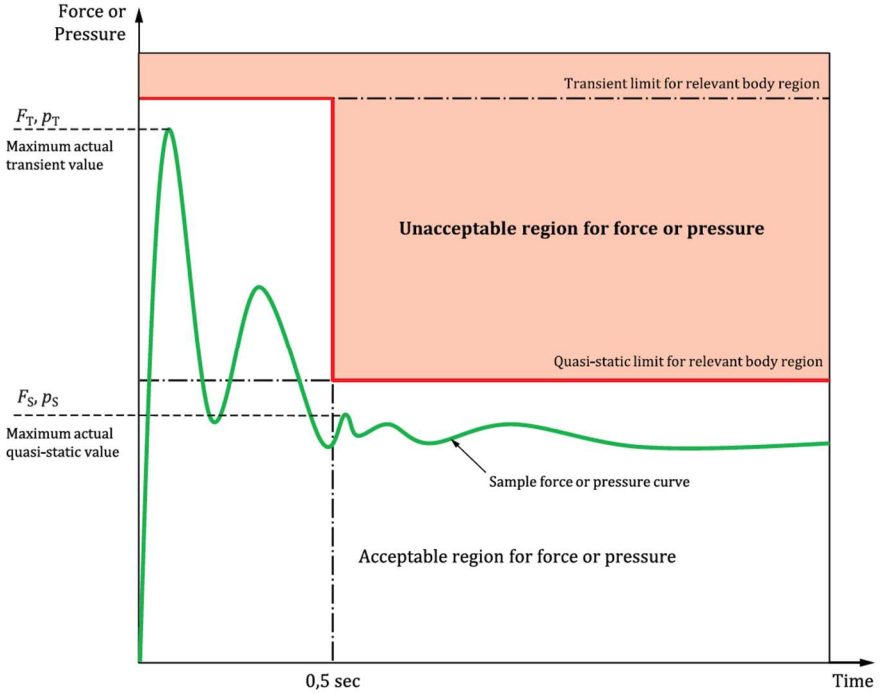


Figure 1.3 Acceptable and unacceptable regions based on the type of contact (transient or quasi-static) and force or pressure threshold.

Collisions can be further mitigated and managed by the use of post-collision methods, which define strategies to react to the collisions based on the torque/force sensors feedback.
The design makes cobots also suitable for the hand guiding collaborative mode. This collaborative modality consists in moving the robot by manually carrying the end effector throughout the workspace, without the robot opposing any reaction force to the movement thanks to the enabled force-control modality. The hand guiding feature is typically used as a convenient tool to facilitate the robot programming, since the identification of the robot waypoints can be done by guiding the robot end effector in an intuitive and straightforward way.
As outlined, the intrinsic safety of a cobot is achieved by limiting its weight and speed. The former is typically a design choice, resulting in a light-weight robot, which can be achieved by a proper choice of light alloy material for the links and lighter motors, thus less powerful, which ultimately impacts on the maximum payload and on the maximum joint speed. The speed limitation can be alternatively achieved by adopting proper and safety-rated control strategies on each axis. In each case, it is to be pointed out that there is a clear trade-off between safety and performances: a robot with high performances, namely high payload, speed, and repeatability, necessarily requires a very rigid structure and powerful actuators, which increment its weight and thus reduce its safety. Vice versa, a safe robot, namely with limited weight and limited maximum speed, will necessarily have lower performances. Table 1.2 summarizes the typical indicators of a collaborative robot.

Table 1.2 Range of indicators of a collaborative robot

Axis number	6 – 7
Payload (kg)	3 – 14
Maximum reach (mm)	500 – 1300
Maximum joint speed (deg/s)	360
Repeatability (mm)	0.03 – 0.1
Weight (kg)	< 35

In the context of Industry 4.0, cobots are used inside the Smart Factories to relieve operators from burdens and stress, to protect them from injuries or to aid them in complex tasks where both the human cognitive skills and the robot accuracy and repeatability are required. The niche in which this is actually beneficial for the productivity is expanding but still limited. The most common cobots applications are:

- Pick and place
- Packaging and palletizing
- Assembly

- Polishing
- Machine tending
- Quality control and inspection

Other advantages of cobots, that are often emphasized in marketing strategies, are the following:

- *ease of programming*: aside from the hand guiding modality, already mentioned, the programming interfaces are particularly user-friendly (touch screens, advanced visualization modalities and features and so on).
- *fast setup*: the installation of both the robot and of its additional components (end effector, additional sensors) is easy and quick and the robot can be made operational in a plug-and-play fashion, resulting in reduced commissioning time.
- *flexible deployment*: due to its lightweight, the robot can be easily moved and re-used with minimum effort for different tasks in different machines. Also, due to its small footprint, no major modifications of the layout are needed.

As a last consideration, it is worth pointing out that there exists the possibility, for some models of industrial robots, of the application of the *AIRSKIN technology* [7], which allows to (partially) convert them into cobots, by means of a covering protective skin able to detect collisions and absorb the impact energy. It is to be noted, however, that this strategy is to be combined with a safe speed limitation (maximum speed up to 1 m/s).

1.2 Artificial vision

Artificial vision can be defined as the collection of systems aiming at creating an approximate model of the real world starting from data acquired from vision sensors. This involves, other than vision sensors, algorithms and methods for acquiring, processing and analysing the data, with the scope of transforming raw data to high level information (similar concept to what human brain does). Modern vision sensors can provide a detailed perception of the environment by means of different core technologies. Digital 2D cameras, which provide bidimensional images or videos, represent the traditional sensors, but they are not the sole. The other prominent category encompasses 3D sensors, which allow to capture a 3D representation of the world. The long-established type of 3D sensors comprehends laser scanners, also called LiDAR, acronym for *Light Detection and Ranging*, that have applications in rapid prototyping, cultural heritage, autonomous navigation, terrestrial and airborne mapping and so on. They are usually bulky devices that can reconstruct a dense and accurate point cloud, but they usually rely on the relative movement between a

unidimensional laser beam and the surface to scan, which makes them not suitable for all the real-time applications in which a one-shot updating of the whole is required at a high frame rate. In the last decade, the use of the so-called depth cameras took off, largely thanks to the pivotal role that the *Microsoft Kinect v1* and *Microsoft Kinect v2* [8] had. Depth cameras can acquire a 3D representation of a whole scene at a relatively high frame rate, making them suitable for real-time applications. Nowadays, there are on the market several models of depth sensors based on different technologies.

Due to its great potential and versatility, artificial vision is currently exploited also in many branches of robotics, absolving different purposes. A list of the main applications of vision in the industrial robotic field is reported below, divided between traditional and most recent ones, the latter mainly validated at laboratory level:

TRADITIONAL APPLICATIONS

- **Object recognition and robot grasping/manipulating** [9]: in this case, artificial vision is used to recognise the pose of unordered objects. This information is then used to define the proper position and orientation of the robot end effector to pick them or perform other operations on them. Common examples can be the picking of unordered objects placed on a conveyor belt or from inside a bin, the latter commonly known as bin picking [10].
- **Aiding in robot assembly or other processes** [11].
- **Visual servoing and tracking** [12]: data obtained from the camera are used as feedback signals in a closed loop control system. In particular, the controller moves the robot with the aim of minimizing the error between the assigned position and the one detected by the camera. The camera information can be used also in an open loop control technique that exploits the information obtained by the camera only to detect the pose to reach, without the correction of a possible mismatching.
- **Quality control and inspection** [13]: typically, a camera is mounted on the robot end effector and used to inspect objects with complex geometry and/or difficult to reach.

CUTTING-EDGE APPLICATIONS

- **Real-time collision avoidance** [14,15]: in this technique, the vision systems are used to detect dynamic obstacles inside the robot workspace in order to adjust the robot motion by means of real-time strategies.
- **Automatic guidance** [16-18]: artificial vision is used for mobile robots to allow a safe navigation in a completely or partially unstructured environment inside a factory.
- **Motion planning** [19,20]: it can incorporate low-to-high-levels planning of sequence of actions used to manage variations in the tasks due to external factors

such as human collaboration. The planner usually makes use of sophisticate control strategies that are not performed within a cycle time.

- **Programming by demonstration** [21-23]: this programming strategy consists in automatically converting a demonstration of a certain task, captured by a vision sensor, into a ready-to-use robot program.
- **Others** [24**,**25]: other novel human-robot interaction modalities are enabled by artificial vision. They can rely on gesture recognition, facial expression recognition, leap motion, augmented reality and so on.

1.2.1 2D cameras

2D cameras are the traditional well-known digital cameras and are here just outlined, an exhaustive description being beyond the scope of this Thesis. The core technology could rely on one of the following sensors:

- **CMOS** (*Complementary Metal-Oxide Semiconductor*): it is a digital sensor that converts the charge from a photosensitive pixel to a voltage at the pixel site. The signal is then multiplexed by row and column to multiple on-chip, digital-to-analog converters. CMOS sensors feature high speed, low sensitivity, and high, fixed-pattern noise.
- **CCD** (*Charged Coupled Device*): it is an analog device that converts light into electrons by means of a silicon chip containing an array of photosensitive sites. Being an analog device, output is immediately converted to a digital signal by an analog-to-digital converter. The voltage is read from each site to reconstruct an image.

CCD sensors are more expensive than CMOS sensors and they consume more power but are traditionally known to achieve higher-quality and lower-noise image. Nowadays, however, CMOS sensor technology has advanced to such an extent that it is fast approaching the quality and capabilities of CCD technology, and with a significantly lower price tag, smaller size, and power consumption, which makes it a good fit for machine vision. For industrial applications, 2D cameras are typically paired with a controller with a proprietary software which contains a collection of functions for object and features recognition, *Optical Character Recognition* (OCR), filters of various type and other functionalities (see for example [26]).
For this type of cameras, illumination plays a key role, highly affecting the image quality. A proper lighting system allows to remove shadows and to uniform the brightness on the scene, avoiding fluctuations and making acquisition repeatable. Several illumination techniques can be utilized, a description of which can be found in [27].

1.2.2 3D cameras

As the name suggests, 3D cameras are sensors capable of acquiring information about the tridimensional nature of a scene. Typically, the sensor acquires a set of distances, that can be converted into a 3D representation of the object, namely a point cloud. They can be seen as an evolution of range/distance/proximity sensors, which output the distance to the object as a single number. Several types of 3D sensors exist. They can be divided into two main categories based on whether they can be used only for offline measurements or can capture a real-time representation of the scene. The former category is not relevant for the scope of this Thesis, so only the latter category will be the focus of this Section. Sensors that can provide a real-time 3D scene information are often referred to as *depth cameras*, since they store depth values (distances) inside a matrix of pixels (a normal digital camera stores RGB values instead). If also an RGB triplet is associated, the device is often referred to as *RGB-D camera*. If a series of camera parameters, namely the intrinsic parameters, are known, the depth matrix can be converted into a 3D point cloud. Usually, this process is already done by the camera manufacturer, which accounts for the possibility of proving an actual point cloud as output, other than the depth matrix. Sometimes however, there can be the need of performing what is called an *intrinsic calibration*, to estimate the intrinsic parameters, because either are unknown or a better estimation than the one provided by the manufacturer is required.
The single depth data matrix and the corresponding point cloud are sometimes referred to as 2.5D [28], since they are obtained by a single-view acquisition. Following this reasoning, the real 3D data is obtained by merging multiple 2.5D acquisitions, obtained from different point of views.
Below a list of the current technologies used in depth cameras is presented and briefly descripted (for an exhaustive description, cf. [29]).

- ***Structured Light**:* it uses one camera and a structured light projector. The depth estimation is obtained by triangulating over the codified rays projected over the scene.
- ***Time-of-Flight (ToF)**:* the core of this system consists in a light transmitter and a light receiver. The light emitted bounces back on the object surface and comes back to the receiver in a specific amount of time, that is measured and used to estimate the distance.
- ***Stereoscopy**:* passive stereoscopy is based on the use of two cameras to triangulate over homologous keypoints on the scene. In modern devices the two cameras are embedded inside small and compact devices, which take care of the stereo-vision algorithms implementation and processing. In some cases, particular expedients are used to increase speed, one of the most recent and

notable being the use of *Field Programmable Gate Array* (FPGA), such as described in [30].

- ***Active Infrared (IR) Stereoscopy:*** it uses two cameras and a light projector to triangulate with the two cameras over the codified rays projected on the scene. This helps the identification of keypoints, since the projector always provide a texture, that might otherwise be absent on the scene.

Recent years have seen an important raising of the available off-the-shelf depth cameras and a significant improvement of their features. Differently from the classical 2D digital cameras, they can directly provide spatial information, which allow a straightforward reconstruction of the 3D scene and are particularly suitable to reconstruct unstructured environments. Whereas in the case of 2D digital cameras the technology has reached a mature stadium, the widespread, improvement and diversification of depth cameras is particularly evident in the current times. This motivates one to keep an eye on their market, with the aim to spot quantum leaps that can have important repercussions, for instance, in safety-related questions. Just to give a practical example, the camera latency, frame rate, data quality and reliability determine the rapidity and certainty by which the camera senses an obstacle, thus directly affecting the safety of the system.

A market analysis was carried out, motivated by these considerations. Table 1.3 reports a collection of the current most popular and cutting-edge depth cameras on the market, with indications on their core technology, depth resolution, depth frame rate, *Field of View* (FOV) and operating range. Here only the depth features are presented, since are the distinctive traits of the camera. It is noteworthy, however, that most of the cameras are paired with an RGB sensor, and the camera has the ability to synchronize and merge 3D data with the colour stream, possibly necessary for certain applications. Furthermore, other parameters have an important impact on the choice of the camera, but are here not reported; for instance, the latency (typically ≈ 1 frame), the accuracy, the spatial noise and the temporal noise. One last note: for each manufacturer, only the most representative cameras are shown.

Table 1.3 Most popular cameras on the market and their main depth features. The first column reports an identification number used for later reference and the second column the model name. The third column reports the core technology, which can be *Structured Light* (SL), *Time-of-Flight* (ToF), *Stereoscopy* (S) or *Active IR Stereoscopy* (AIRS). Since certain cameras allow to stream in different combination of resolutions and frame rates (frame per second, fps), the minimum resolution and maximum resolution streaming modalities with their corresponding achievable frame rates are here reported.

ID	Model name	Tech.	min res – max fps	max res – min fps	FOV [°]	Range [m]
1	Arcure Omega	S	512x256 – 18	1024x512 – 11	100x70	0.5 – 10
2	ASUS Xtion Pro	SL	/	640x480 – 30	58x45	0.8 – 3.5
3	Basler ToF camera	ToF	/	640x480 – 20	57x43	0.5 – 0.8
4	Carnegie Robotics MultiSense S7	S	1024 x 544 – 30	2048x1088 – 7.5	80x45	> 0.2
5	duo3d DUO MC	S	320x120 – 320	752x480 – 45	128x91	0.3 – 2.5
6	Ensenso N35	SL	/	1280x1024 – 10	58x52	0.27 – 3
7	FRAMOS Depth Camera D435e	AIRS	/	1280x720 – 30	86x57	0.2 – 10
8	Ifm O3D302	ToF	/	176x132 – 25	60x45	0.3 – 8
9	Intel Realsense D455	AIRS	848x480 – 90	1280x720 – 30	86x57	0.52 – 10
10	Microsoft Azure Kinect	ToF	/	6540x576 or 1024x1024 – 30	75x65 or 120x129	0.25 – 5.46
11	Microsoft Kinect v2	ToF	/	512x424 – 30	70x60	0.5 – 4.5
12	MYNT EYE S1030	S	/	752x480 – 60	122x76	0.5 – 18
13	Nerian Scarlet	S	832x608 – 125	2432x2048 – 16	80x84	> 0.14
14	Nerian SceneScan Pro + Karmin3	S	640x480 – 100	1600x1200 – 15	variable	variable
15	Occipital Structure Core	AIRS	1280x800 – 60	1280x960 – 54	59x46	0.3 – 5
16	Orbbec Astra Mini	SL	/	640x480 – 30	60x49	0.6 – 5

17	PMD Pico Flexx	ToF	/	224x171 – 45	62x45	0.1 – 4
18	PMD Pico Monstar	ToF	/	352x287 – 60	100x85	0.5 – 6
19	Roboception rc_visard 65	S	214×160 – 25	1280×960 – 1	61x48	0.2 – 1
20	SICK Visionary-T	ToF	/	144x176 – 30	69x56	0.5 – 40
21	Stereolabs ZED	S	640x480 – 120	2208x1242 – 15	96x54	1.5 – 20
22	Zivid Two	SL	/	1944x1200 – 13	50x36	0.3 – 1.5

Figure 1.4 shows the region of the plane *resolution – frame rate* covered by the currently available depth cameras on the market. The maximum resolution (2432x2048) is achieved by the *Nerian Scarlet* (at a frame rate of 16 fps), whereas the maximum fps (320) is achieved by the *duo3d DUO MC* (with a corresponding resolution of 320x120).

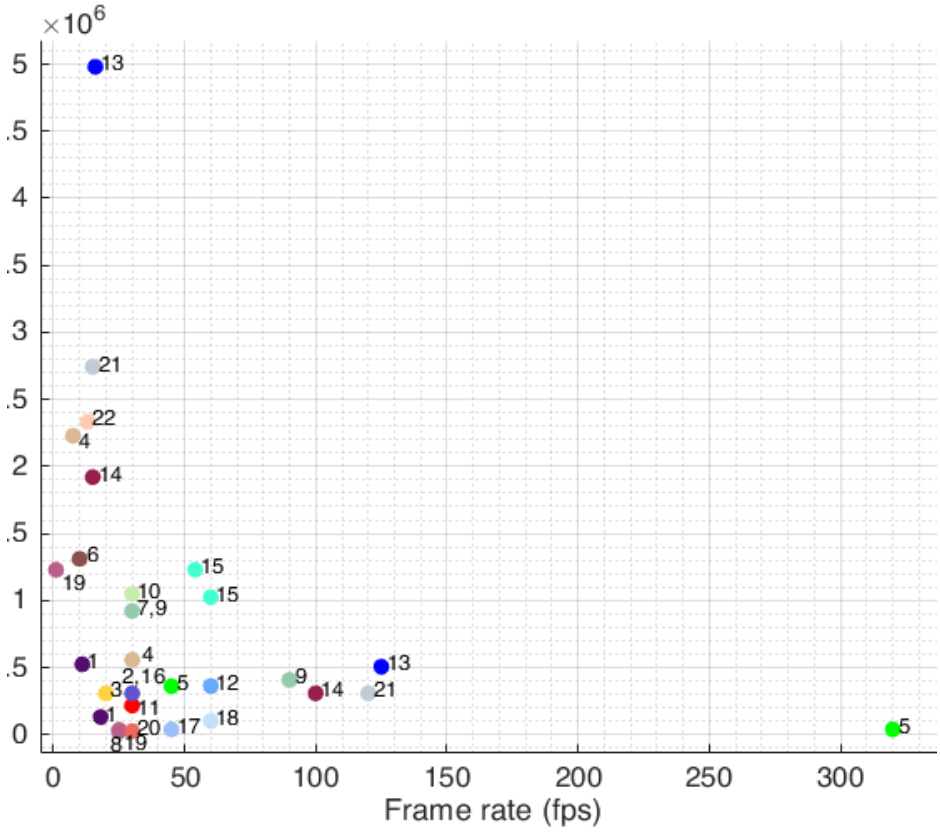


Figure 1.4 Resolution and frame rate of the considered commercial cameras. Annotated numbers refer to the *Camera ID* of Table 1.3.

Figure 1.5a shows the camera FOVs, which are mainly concentrated around 60° x 50° but can be pretty variable depending on the optics used, which can be replaced in

some cameras. Figure 1.5b shows the operational range, which varies between 0.1 m and 40 m. Typically, the minimum distance extends to a maximum of 0.5 m, whereas the maximum varies depending on the camera.

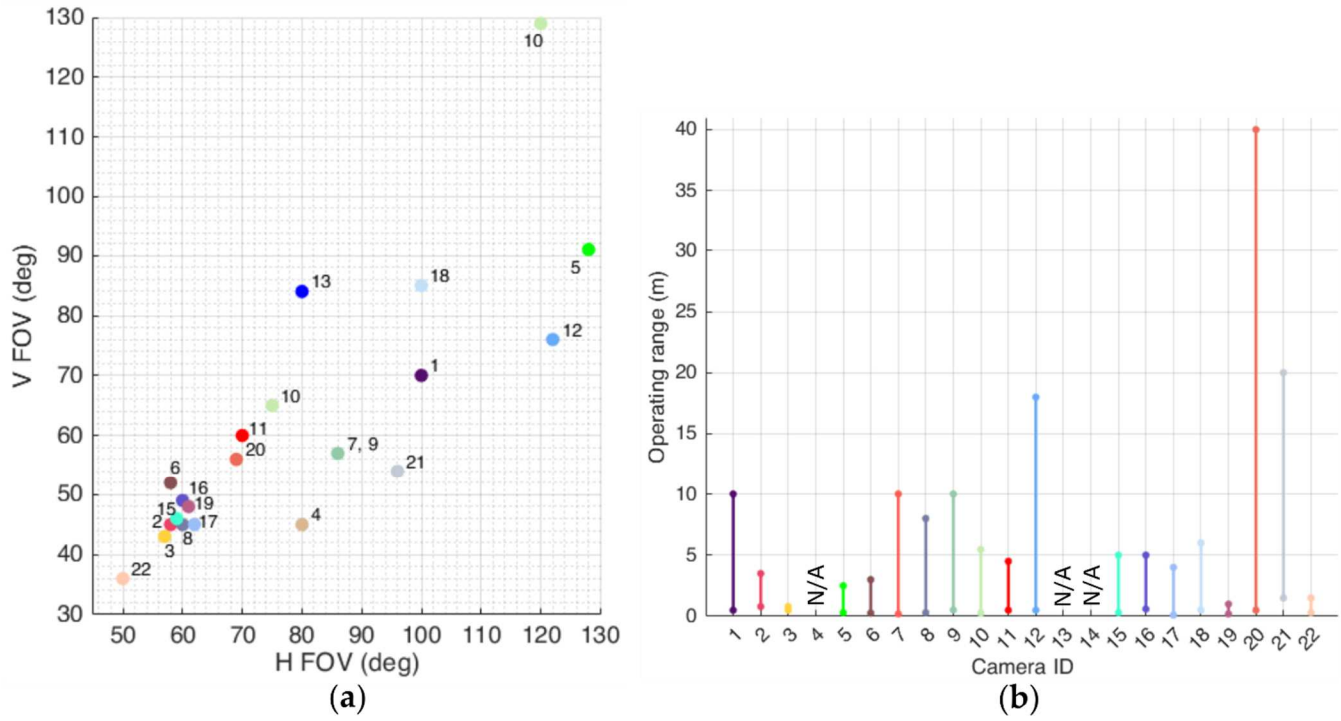


Figure 1.5 (**a**) FOVs (annotated numbers refer to the *Camera ID*) and (**b**) operating ranges distribution of the considered depth cameras. *H* and *V* stand for Horizontal and Vertical, respectively; *Camera ID* is referred to Table 1.3.

1.3 Human Robot Collaboration in industrial settings by means of artificial vision

HRC has recently gained a vast amount of interest in both academic and industrial field. It has the potential of enhancing the functional flexibility and efficiency of several processes typical of the Smart Factories, which realize the current trend of automation and intelligent manufacturing called Industry 4.0. Notably, HRC was considered inside the EU project ROBO-PARTNER [31] which aimed at integrating assembly systems and human capabilities. Specifically, the project focused on intuitive interfaces, safety strategies and equipment, and proper methods for planning and executions, all considered as key enablers for HRC. In literature, different taxonomies of HRC are proposed, with the aim of identifying various levels of interaction between robots and human operators [32-35]. It is noteworthy that some publications, e.g. [36,37], present a semantic distinction between HRC, which refers to the concept of human and robot working together by sharing a common

task, and *Human-Robot Interaction* (HRI) that is limited to the concept of a safe coexistence. The distinction of HRC levels presented by De Luca and Flacco [35] is here adopted: according to the proposed framework, HRC can be divided in three nested levels of interaction between human and robot. The order is progressive, i.e. each level of interaction entails the fulfilment of the lower level(s):

1 – *Safety*: workspace is shared, but task is not shared (no direct or indirect cooperation occurs);

2 – *Coexistence*: workspace is shared, robot and human operator can work on the same object, but without any mutual contact nor any operative coordination;

3 – *Collaboration*: robot and human operator perform a complex task together, through direct interaction and coordination and with possible physical contact.

When dealing with HRC, safety is of paramount importance. In [38], an example of the application of the technical specification *ISO/TS 15066:2016* [4] is presented in the case of a collaborative assembly scenario. Figure 1.6 outlines the laws, the directives, and the standards that are relevant in the field of HRC.

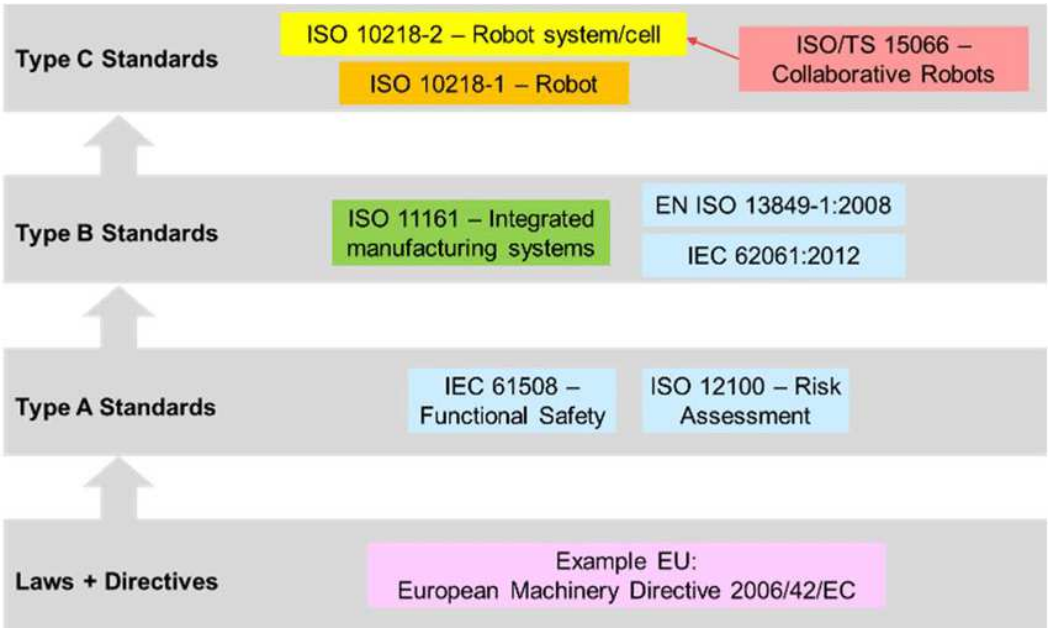


Figure 1.6 Overview of the standards that regulates the field of HRC (figure from [38]).

In [39], Zacharaki et al. survey and summarize features for minimizing the risk of a HRC application. The authors identify and analyse the following categories: perception, cognition, action, hardware features, societal and psychological factors, risk assessment through hazard identification techniques.
In general, HRC can be achieved in several ways and with different level of interaction (according to [35]). The most popular way is undoubtedly through collaborative robots, which are lightweight robots that rely on the PFL collaborative mode of the

technical specification *ISO/TS 15066:2016* [4] (cf. Section 1.1.2), and can achieve the *HRC Level 3 - Collaboration*. Another modality that enables HRC up to *Level 2 - Coexistence* is the *Speed and Separation Monitoring* (SSM) collaborative modality of *ISO/TS 15066:2016* [4]. This modality, deeply analysed in Section 1.3.1, consists in enabling HRC by imposing a safety threshold on the relative distance between the robot and possible obstacles (mainly referring to the human body). This threshold is dynamical since it also depends on the robot-obstacle relative velocity. Other tools suitable for HRC consist in intuitive human-robot interfaces, which can provide highly natural and tangible ways of interaction. Interfaces of these types can be referred to as *Natural User Interface* (NUI) and *Tangible User Interface* (TUI). The idea at the basis of NUI is to offer a reality-based interaction by exploiting users' pre-existing knowledge and using actions that corresponds to daily practice in the real world [40], overcoming the classical interaction devices such as keyboards and mice. The term TUI refers to the interaction systems that rely on embodied interaction, tangible manipulation, physical representation of data and embeddedness in real space [41]. These intuitive interaction systems can be used either to facilitate the robot programming or to interact with the robot online, while it is executing a task. Some examples of intuitive programming modalities are the hand guiding modality typical of collaborative robots, teleoperation through haptic technologies [42] or hand movements [43], or programming by demonstration [21-23]. Methods for interacting with the robot while operating can rely on automatic recognition of gestures, eye gaze, facial recognition, and other physical cues [24,25,44] or even vocal commanding [45,46]. A relevant research branch is focused on motion planning [19,20], that can incorporate low-to-high-levels planning of sequence of actions used to update the robot behaviour when performing a side-by-side collaboration with human operators. In addition, an emerging trend is to combine interfaces with *Augmented Reality* (AR) or *Virtual Reality* (VR), for instance to facilitate robot programming [47] or to increase system productivity while enhancing human safety [48].
One further aspect when considering HRC is that, by cancelling the barrier between human and robots, also the impact on the psychological state of the operator is to be addressed. In fact, it is necessary to ensure that the operator feels comfortable and safe and that mental strains associated with the cooperative tasks are bearable [49].

It is noteworthy the fact, pointed out by Villani et al. in [5], that the vast majority of academic solutions concerning HRC, validated at laboratory, have not found concrete application in industry yet, and an important effort in terms of technology transfer is thus required.

1.3.1 Collision avoidance

Inside the HRC context, collision avoidance strategies can be used to enable the *Level 2 – Coexistence*, or to enhance the *Level 3 – Collaboration*. However, it should be noted that they are not sufficient for the third level, which accounts for physical contact, that hence is to be handled by limiting the power and force and by using post-collision methods. Collision avoidance strategies are, as a matter of fact, pre-collision strategies, and allow collisions to be prevented by means of a real-time online adjustment of the robot motion, possible by exploiting data provided by vision sensor. This has several advantages: the first one is that it allows to enhance the robot functional flexibility, in such a way that it can operate in an environment unstructured to a certain degree. The second one is that it allows, by preventing collisions before they occur, a workspace sharing with human operator, while maintaining high productivity when possible. In fact, the robot can operate at high dynamics when no human operators are detected and activate appropriate safety measures only when the presence of a human operator is detected by vision sensors. These safety measures can range from the simple robot temporary stop or speed limiting to a real-time dynamic adjustment of the robot speed (procedure known as *trajectory scaling*) and/or modifications of the trajectory geometry (this latter by means of the so-called *escape motions*). While the simpler strategies can rely on the use of traditional safety laser scanners and optical barriers, these latter more complex strategies are enabled thanks to the detailed reconstruction and prediction of the human body motion possible by means of modern vision sensors. One interesting aspect of this approach is that it is theoretically applicable to industrial robots, since it does not require intrinsic limitations on the robot design.

In robotics, obstacle avoidance is a well-known topic that emerged way before the raise of the modern industrial HRC. As reported in [50], several methods have been developed, and some of them have been adapted to the HRC context. An example of pioneering work is the one by O. Khatib, who in 1986 proposed a real-time obstacle avoidance approach based on the concept of *artificial potential field* [51]. The idea is that the robot moves in a field of forces, where there are attractive poles in correspondence of the target to be reached and repulsive poles in correspondence of the obstacles to be avoided. In [52], an impedance control method is developed, which establishes virtual spheres between the robot and the surrounding objects. The work [53] used a *danger index*, defined as the product of the distance, velocity and inertia, to generate alternative trajectories when the index exceeds a predefined threshold. The trajectory generation relies on a *virtual force* that aims to push the robot away from the danger area. In [54], the concept of *kinetostatic danger field* is introduced, which is a quantity that captures the complete state of the robot, namely

its configuration and velocity, and further used in [55] to implement a reactive control strategy which exploits information acquired from distributed distance sensors. In [35], the concept of repulsive vectors, applied to a series of robot control points, is used for collision avoidance purpose.

More recent methods, based on the concept of *safety constraints* [15,56], continuous speed adaptation based on *dynamic SSM* [57], and *control barrier functions* [58,59] have been developed focusing on the fulfilment of the SSM collaborative mode presented in the technical specification *ISO/TS 15066:2016* [4]. This approach is particularly safety-oriented and thus proves theoretically suitable for industrial robot applications. These new methods represent a change in the approach, which shifts from considering safety as a requirement that necessarily limits the performance of machines towards a constraint driving the optimization of the robot performance, which is one of the future goals identified in [5]. The main concept around which these latter methods revolve is the formula for the *Protective Separation Distance* (PSD) presented in the SSM collaborative mode section of the *ISO/TS 15066:2016* [4], whose most general formulation is:

$$S(t_0) \geq \int_{\tau=t_0}^{\tau=t_0+T_R+T_S} v_H(\tau)d\tau + \int_{\tau=t_0}^{\tau=t_0+T_R} v_R(\tau)d\tau + \int_{\tau=t_0+T_R}^{\tau=t_0+T_R+T_S} v_S(\tau)d\tau + C + Z_D + Z_R \qquad (1)$$

where:

$S(t_0)$ is the PSD at the current time instant t_0;

T_R is the reaction time of the robot system;

T_S is the stopping time of the robot, from the activation of the stop until the robot is halted (it is not constant but depends on the robot configuration, speed, load);

v_H is the speed of the human operator in the robot direction;

v_R is the speed of the robot in the human operator direction;

v_S is the robot speed in the human operator direction during the stopping operation;

C is the intrusion distance, as defined in *ISO 13855:2010* [60]; it is the distance that a part of the body can intrude into the sensing field before it is detected;

Z_D is the position uncertainty of the operator in the collaborative workspace, as measured by the presence sensing device resulting from the sensing system measurement tolerance;

Z_R is the position uncertainty of the robot system, resulting from the accuracy of the robot position measurement system;

Further discussion and guidelines about the implementation of SSM in collaborative workcells can be found in [61]. Furthermore, it is worth noting that some (uncommon) works exploit the concept of PSD, rather than to develop collision avoidance strategies, on ways to visualize it, specifically by projecting on the robot mounting surface a real-time enclosing shape delimiting its PSD, the interruption of which can be sensed and used to trigger a robot stop [62,63].

To implement the aforementioned collision avoidance strategies, obstacles position and -where needed- obstacles speed have to be estimates in real-time. Several typologies of sensors can be exploited for this purpose, and can be subdivided in four main categories:

- motion captures systems [64];
- pressure-sensitive mats [65]
- proximity/distance/capacitive sensors [55];
- vision sensors (either 2D or 3D) [25,59].

Motion captures systems are typically based on optoelectronic measurement systems and/or inertial sensors, they can be reliable and precise, but they have the significant downside that they need to be wore or attached to the human operator or to the generic obstacle to be monitored, which is thus to be known in advance and to be suitable for the sensor attachment. Pressure-sensitive mats can safely detect the worker feet position, but only operate at the ground level and cannot track the upper-body. Proximity sensors, mounted on robot, can be effective in particular situations, but they provide a non-detailed sensing of the environment, are often limited in range and precision and need a method to process and integrate together the data. One unique example of commercial robotic solution using this concept is the sensor skin based on capacitive sensing developed by Bosch for the *APAS Assistant* robot [66], which has a sensing range of 50 mm. The third category is undoubtedly the one who gained the most popularity in the recent years [67], due to the popularization of depth cameras, analysed in Section 1.2.2. Compared to the other typologies of sensors, they have the advantage of being capable of reconstructing a detailed long-range portion of the environment and the obstacles do not need to be known in advance nor any sensor attachment on them is required. Some methods rely on the use of 2D cameras, but they sometimes need visual markers [68] to track the obstacle, or the obstacle typology needs to be known in advance in order to exploit data driven recognition approaches [69]. 3D vision sensors, on the other hand, can directly output a 3D representation of the environment, which leads to a straightforward obstacle

reconstruction. Traditional safety laser scanners fall inside this category, but they are essentially only used for obstacle detection (typically at the ground level), since they are not suitable (due to their working principle and limitation, e.g. concerning the frame rate) for the more advanced obstacle tracking needed in collision avoidance applications. Depth cameras, on the other hand, can be used to efficaciously reconstruct the obstacle 3D geometry and to estimate its motion. Furthermore, the vast majority of them can provide RGB data as well, in the form of classical RGB images or fused with the 3D data. There are some caveats, however, in the use of vision sensors. The first one is that they are sensible to occlusion, so their positioning is to be carefully chosen, since it has important repercussions on safety. In general, inside the collision avoidance framework, they are placed in a fixed fashion on an external frame, but there are some exceptions [70,71]. This topic will be further discussed in Section 5.1, where considerations on the optimal placement will be made. One other important limitation of depth cameras is the lack of safety certifications, differently from the classical safety laser scanners and optical barriers. One unique exception is the *PILZ SafetyEYE* [72], which was commercialized as the first safe camera. To address this problem, some solutions can be to pair the unsafety sensors with a safety one [25,73] or to create a particular infrastructure that acts as a safe network for unsafe devices [74].
Even with their caveats, depth cameras currently represent the most valid devices to track dynamic obstacles. Furthermore, the depth camera technology is rapidly evolving, by means of enhancements in terms of both resolution and frame rate, and hopefully, the reliability will be subjected to improvements as well.

In Table 1.4, a list of the most recent and notable collision avoidance systems based on artificial vision are reported, providing the main references used for the development of the real-time collision avoidance method presented in this Thesis. It is worth reporting the recent commercialization of the plug-and-play 3D smart assistant by *SMART ROBOTS* [75]. It is essentially a 3D camera with advanced embedded functionalities that can be paired with any collaborative robot. One function concerns collision avoidance: the system can detect human body parts (body, arms, hands) and human-robot distance in real-time, slowing down the robot before the collision occurs. The provided functionalities, though being safety related, are not certified as such.

Table 1.4 List of some notable publications of the last decade concerning vision-based pre-collision methods that enable HRC. It is mentioned the year of publication, the hardware and software used for the test, where indicated by the authors (a remand to the article is suggested in the case of extensive description given by the authors difficult to summarize), and the topic in which novelties are proposed by the Authors. Abbreviations are used for the sake of conciseness:
For the robot: *DOFs*: Degrees of Freedom; *C*: Collaborative; *I*: industrial
For the sensors: *F*: mounted on a fixed external frame; *R*: mounted on the robot
N/A: not applicable; used in the case where no info was found

Paper Year	Robot	Sensors	Other hardware	Software/Frame works	Main focus
[35] 2012	KUKA LWR-IV **7 DOFs; C**	1 x Kinect V1 **F**	Eight core CPU	N/A	**Collision reaction, avoidance (+ gesture and voice recognition)**
[16] 2012	iRobot ATRV Jr. **Mobile**	2 x Swissranger SR4000S (range cameras) **F** 2 x Tyzx G3 EVS (stereo cameras) **F**	2 x External PC (2.67 GHz Intel Core i7 920 quadcore with 64-bit Ubuntu Linux 10.04 LTS) All connected through a wired gigabit Ethernet network	N/A	**Obstacle detection; Sensor fusion**
[76] 2012	N/A	3 x Kinect V1 **F**	3 x External PC (Intel i7-2600 3.4 GHz processor, 16 GB RAM, Nvidia GeForce GTX 560-1GB)	Point Cloud Library (PCL)	**Obstacle detection**
[77] 2013	Willow garage PR2 (Simulated) **Humanoid** (but standing still)	1 x Kinect V1 **F**	PC	MATLAB Move3D	**Manipulation planning based on human motion prediction**
[78] 2013	KUKA LWR IV **7 DOFs; C**	1 x Kinect V1 **F** 1 x Stingray F201 B (gray 2D camera) **R**	1 x External PC	OpenNI ViSP library Reflexxes	**Visual-based human-robot cooperation**
[74] 2013	COMAU NS16 **6 DOFs; I**	N/A	-See the paper for info-		**Safe network of unsafe devices; Collision avoidance**

[79] 2014	COMAU NS16 **6 DOFs; I**	-No info on the number- MESA ToF cameras F	-See the paper for info-		**Dynamic SSM (speed adjustment)**
[80] 2014	KUKA LWR4+ **7 DOFs; C**	1 x NDI Polaris optical tracker **F**	Real-time CPU	N/A	**Dynamic SSM (speed adjustment)**
[81] 2014	ABB IRB120 **6 DOFs; I**	PhaseSpace motion capture system **On human hand/arm**	External PC (Core i7-3610 QM 2.3GHz, Windows 7)	-See the paper for info-	**Dynamic SSM (speed adjustment)**
[17] 2014	KUKA omniRob **7 DOFs; C (on mobile platform)**	2 x Kinect V1 **F** (on robot platform) 2 x 2D lidar laser scanners **F** (on robot platform)	N/A	N/A	**Obstacle tracking**
[55] 2014	ABB IRB 140 (controller: ABB IRC 5) **6 DOFs; I**	20 x IR LED (Sharp GP2Y0A02YK) **R**	1 x National instrument PCI 6071E board 1 x External PC (Linux OS with Xenomai patch for real-time)	Interface developed for exchanging data with ABB controllers; Strategy developed with Simulink GUI and converted into an executable trough Simulink Real-Time workshop	**Collision avoidance**
[82] 2014	/	2 x AXIS PTZ RGB (fish-eye surveillance cameras) **F**	1 x External PC (cameras connected to it via Ethernet)	ROS (Robot Operating System) OpenCV	**Human Detection and Tracking**
[83] 2014	Lab-Volt 5150 **5 DOFs**	4 x Kinect V1 **F**	1 x client PC + 1 x server PC	Virtual simulation engine based on Tundra software	**Human Detection and Tracking**
[84] 2015	KUKA LWR **7 DOFs; C**	1 x Kinect V1 **F**	1 x External PC	ROS Point Cloud Library (PCL) Reflexxes SoftMotion	**Collision avoidance**
[85] 2015	KUKA LWR-IV **7 DOFs; C**	1 x Kinect V1 **F**	Eight core CPU	N/A	**Collision avoidance**

[62] 2015	Custom-made light-weight robot **5 DOFs**	1 x RGB camera + 1 x light projector **F**	1 x PC	MATLAB	**Dynamic SSM (through projection)**
[73] 2016	ABB IRB 4600 **6 DOFs; I**	1 x Kinect V1 **F** 1 x Sick Laser Scanner **F**	1 x Safety controller 1 x Dynamic safety controller	N/A	**Dynamic safety system**
[18] 2016	KUKA omniRob **7 DOFs; C (on mobile platform)**	2 x Laser scanners -No info on the model- **On mobile platform**	N/A	N/A	**Dynamic SSM (size of safety area based on the platform speed)**
[86] 2016	ABB FRIDA (dual arm) (controller: ABB IRC 5) **14 DOFs; C**	1 x Kinect V1 (scenario 1) **F** 2 x Kinect V1 (scenario 2) **F**	Real Time Linux Xenomai PC	IBM CPLEX Optimization Studio for the LP algorithm	**Dynamic SSM for redundant robots (and non)**
[87] 2017	ABB IRB 140 **6 DOFs; I**	2 x Kinect V1 **F**	External PC (Intel Core i7, CPU of 2.7 GHz, 4 GB RAM, 64-bit Windows 7)	C/C++ libraries, communication and framework implemented in Java	**Collision avoidance**
[88] 2017	Willow garage PR2 **Humanoid**	1 x Kinect V1 **F** (on mobile platform)	/	ROS	**Trajectory planner that consider "interaction potential" (gesture and speech)**
[89] 2017	KUKA LBR iiwa 7 R800 **7 DOFs; C**	Heptagon Taro (TOF) **R**	Camera connected via Wireless to the robot controller unit	N/A	**Obstacle detection**
[90] 2017	KUKA LWR-IV **7 DOFs; C**	2 x Kinect V1 **F**	Fast Research Interface (FRI), Intel core i7-2600 CPU 3.4 GHz, 8Gb of RAM	Code developed in C++	**Collision avoidance (focus on depth maps merging)**
[63] 2017	KUKA LBR iiwa 14 **7 DOFs; C**	Custom setup: LED-DLP projector + RGB camera **F**	N/A	N/A	**Dynamic SSM (through projection, same concept as** [62]**)**

[33] 2017	KUKA KR180 **6 DOFs; I**	2 x Intenta S2000 (stereo cameras) **F** 1 x RGBD camera -No info on the model- **R** 1 x Schunk Multi-axis force/torque sensor	N/A	N/A	**Levels of interaction**
[70] 2018	KUKA LBR iiwa 7 R800 **7 DOFs; C**	SICK Visionary- T camera (TOF) **R**	External PC (camera connected via Ethernet)	MATLAB	**Obstacle detection**
[71] 2018	Universal Robot UR10 **6 DOFs; C**	3 x SICK TiM551 Laser scanners **1 F; 2 R**	N/A	N/A	**Obstacle detection (consequent robot security mode switching)**
[15] 2018	ABB IRB 140 (controller: ABB IRC 5) **6 DOFs; I**	1 x Kinect V1 1 x ASUS Xtion **F**	2 x External PC (Linux OS with Xenomai patch for real-time)	-See the paper for info-	**Dynamic SSM (trajectory + speed adjustment)**
[91] 2019	KUKA LBR iiwa 7 R800 **7 DOFs; C**	Terabee TeraRanger One (single pixel TOF) **R**	External PC (camera connected via USB)	MATLAB	**Obstacle detection**
[57] 2019	ABB IRB140 **6 DOFs; I**	1 x Leuze RSL440 safety laser scanner **F**	External PC	MATLAB	**Dynamic SSM (speed adjustment)**
[25] 2020	ABB IRB 4600 **6 DOFs; I**	2 x Kinect V2 **F** + 2 x Laser scanner KEYENCE SZ-V32n **F**	-See the paper for info-		**Obstacle detection; Gesture recognition**
[58] 2020	Universal Robot UR5 **6 DOFs; C**	1 x Intel Realsense D415 **F**	OROCOS ROS (C++) CVXGEN		**SSM and PFL**
[59] 2020	Universal Robot UR5 **6 DOFs; C**	Tested both with 1 x Intel Realsense D415 and 1 x Asus Xtion **F**	OROCOS ROS (C++) CVXGEN		**Collision avoidance**

1.3.2 Programming by demonstration

There are nowadays different methodologies to program a robot, either traditional or recently popularized. A list is here presented, with a brief overview, whereas a more detailed description can be found in [5].

- ***Lead-through programming***. It is the older and most traditional methodology. It consists in leading the robot through the waypoints by the use of a teaching pendant (also known as teaching box). This operation is often referred to as "robot jogging".
- ***Offline programming***. This methodology is nowadays highly widespread and it consists in programming the robot offline via a dedicated proprietary software and test the program execution on a simulator, which is included in the software.
- ***Walk-through programming***. It has been popularized with the commercialization of cobots, and it relies on the hand guiding collaboration mode of the technical specification *ISO/TS 15066:2016* [4]. It consists in grabbing the tool attached on the end effector and "walking" the robot through the desired waypoints, which are stored and can be used to command the robot the desired path. The possibility of manually moving the end effector is made possible by means of compliant control schemes or force control, which rely on the use of force/torque sensors (typically mounted on robot wrist and/or joints).
- ***Programming by Demonstration (PbD)***. It is not widespread in actual industrial frameworks yet, even if a considerable amount of work has been done at laboratory level. In literature, this technique can refer to both the following scenarios:
  - the robot replicates as is the movement demonstrated (through a single demonstration) by the human operator [21],[92].
  - The robot learns the demonstration movements with the aim of performing them under varying conditions and to generalize them in new scenarios [22,23].
- ***Others***. Further novel miscellaneous programming methods have been proposed, typically validated at laboratory level only, but not appeared in industrial frameworks yet. They can rely on haptic devices, leap motion recognition, augmented or virtual reality, vocal commanding and so on.

Walk-through programming, PbD and the other novel methods perfectly fit inside the HRC context, since they provide intuitive and user-friendly robot interfaces that do not require much cognitive interaction effort, time consuming procedures nor highly skilled and trained human workers.

1.4 Problem statement: the necessity of a new paradigm

As depicted, HRC is a broad and hot topic that has recently gained popularity in both academic and industrial fields. Even if a considerable amount of work has been done, it is still in its infancy and rapidly evolving. Keeping in mind the practical utility in industrial scenarios, especially in terms of productivity enhancement, in the very last years a significant number of research studies shifted the focus from implementing HRC by means of cobots to realizing HRC by exploiting traditional industrial robots. In fact, cobots, even if increasingly widespread, are still relegated to a small niche of applications in which a sharing of tasks (with possible contact) and intentions (*Level 3* of HRC, see Section 1.3) is actually beneficial to the productivity or the quality of work. As outlined in Section 1.1.2, in facts, cobots comes with limitations in speed and power, thus preventing their use in the vast majority of industrial traditional tasks, where high productivity is achieved thanks to robot power and speed. Rather than having to choose between cobots and industrial robots, which are somehow according to current paradigms, the key idea and motivation of this Thesis is to propose some methodologies to enable a new paradigm, which incorporates the perks of both industrial and collaborative robots. The necessity of this new paradigm is also pointed out by the robot company *Comau*, which, at the time of this writing, has commercialized the robot *Comau Racer5 COBOT* [93], which can switch from industrial robot speed to collaborative (limited) speed when a human operator enters its working area.

The solutions proposed in this Thesis rely on the integration of industrial robots with artificial vision systems, in particular exploiting depth cameras. Among other devices, the choice of depth cameras is motivated by several factors, the main ones being the following:

- they are compact, have an affordable price, and the most recent products can achieve high frame rate and resolution;
- they can reconstruct in a straightforward way the objects and the environment in the scene, which neither are to be known in advance nor have to be previously manipulated;
- they are based on a technology which is rapidly improving and thus proves very promising.

When integrated with industrial robots, the mentioned features of depth cameras allow the implementation of the following main HRC enablers:

- Enhancing robot functional flexibility while maintaining high productivity, realizing the HRC collaborative modality SSM, which allows safe coexistence of

human operators and robots inside the same workspace. Other than human operators, artificial vision can be used to sense generic dynamic obstacles, which can ultimately lead to manage unstructured environments.

- Enabling the intuitive programming method *programming by demonstration*: the demonstration of the task can be easily acquired by means of artificial vision.
- Enabling the online interaction with the robot by means of recognition of physical cues such as hand gesture, facial recognition and so on.

This Thesis focuses on proposing some implementations of the first two points, specifically a collision avoidance method based on SSM and two programming by demonstration methods. The latter are here intended in the sense that the robot *replicates*, after a single demonstration, the motion of a *Human Demonstration Device* (HDD), which is manually carried through a series of target waypoints.
A last section of the Thesis is devoted to an investigation on the optimal placement of robot onboard cameras, which was identified as a topic of interest not properly analysed in literature to the best of the Candidate's knowledge. This last topic, even if addressed as a stand-alone problem, well fits into the context of HRC based on vision sensors, since the sensor placement has important implications on safety and on the general effectiveness of vision-based HRC applications. As outlined by its title, the scope of this Thesis is the proposal of vision-based solutions to enable HRC in industrial settings. The key concept is the realization of HRC not by intrinsically limiting the robot, but rather to pair it with vision sensors, which allows the utilization of traditional industrial robots. In this scenario, safety is to be fulfilled, other than by the hardware itself, also by all the involved methods, algorithms and procedures, an important set of those being the ones concerning the integration of the robot and vision sensors. It is clear that a proper camera placement is necessary for an effective and safe monitor of the region in which HRC is intended to be realized, and serves as a basis for the successive stages. Every systematic procedure that helps at better quantifying this grade of safety and effectiveness of the monitoring is relevant to the topic, and, for this reason, the aforementioned study about camera placement was also included in this Thesis.

One last motivation of the present work concerns the fact that, in order to introduce collaborative solutions in small and medium companies, that might have a limited budget to invest in innovation, robot retrofitting could represent a valid option, as pointed out by Villani et al. in [5]. Further to this point, the proposed approach has the advantage of allowing integration of novel HRC solutions in deprecated robots, to upgrade them, where needed, to fulfil the highly flexible and smart production style typical of Smart Factories.

The main contributions of this Thesis to the state of art are here briefly summarized, whereas the technical novelties are outlined and detailed throughout the specific parts of the Thesis.

- The developed collision avoidance method is suitable for the case of generic dynamic obstacles (not only humans) which may be present in either traditional closed workcells or collaborative workcells. Also, it allows an easy integration of different vison sensors and is particularly efficient so that only one pc (but with enough GPU resources) is needed.
- The developed PbD methods have the characteristic traits of relying on cheap vision sensors and hardware, and are easily implementable in a generic framework.
- The study about the optimal placement of onboard cameras is one of the first presented in literature and provides some insights and tool for a more systematic approach for a safe monitoring of the robot workspace.

2 Experimental setup

The work was carried out inside the research and educational laboratory named *TAILOR* (*Technology and Automation for Industry LabORatory*), born nearly at the time of the beginning of the PhD thanks to a collaboration between the University of Bologna and the company *Siropack Italia s.r.l*. (Cesenatico, FC). In particular, the most striking facility of the "Robotics division" of the laboratory is a robotic cell (cf. Section 2.1), whose project was mainly borne by the Candidate (ranging from the design to the operative commissioning phases). A number of commercial products were courteously granted by the Italian branch of *Mitsubishi Electric Europe B.V.*, *Omron* (Italian branch) and *SMC Europe* (Italian branch), which are gratefully acknowledged.

2.1 Robotic cell

Tests are conducted inside the aforementioned robotic cell, which was designed in such a way to resemble an automatic robotized machine, able to realize as a matter of fact the flexible automation typical of the modern Smart Factories, being adaptable to various different tasks. The robotic cell is endowed with a series of movable guards, granting access to various zones of the cell, valuable feature that was exploited to test HRC application directly into industrial settings. This was very beneficial, since the HRC tests are carried out in a realistic industrial scenario rather than on a standalone robot or by using a simplified setup. The robotic cell is composed of the following main components:

- Two industrial robots by *Mitsubishi Electric*:
  - one articulated robot *RV-4FM-1Q1-S15* (4 kg payload, 514.5 mm reach), henceforth referred to as *RV4F*, equipped with a *Mitsubishi Electric 1F-FS001-W200* force sensor mounted on the wrist and a *SCHUNK KGG-70-48 pneumatic gripper*;
  - one SCARA robot *RH-1FHR5515-Q1-S60* (1 kg payload, 550 mm reach) henceforth referred to as *RH1F*, equipped with a custom-made vacuum gripper.
- Additional mechanical drives, driven by brushless motors and set as robots' additional axes, which guarantees ease of programming and flexibility:
  - one railway under the articulated robot (additional axis for the *RV4F*);
  - one conveyor belt traversing the robotic cell (additional axis for the *RV4F*);
  - one rotary table (additional axis for the *RH1F*);
  - one conveyor belt afferent to the rotary table (additional axis for the *RH1F*).

- A series of vision sensors:
  - three *Omron FH-SCX* cameras, paired with the *Omron FH-3050-20* vision controller;
  - one webcam *HD Logitech® C930e*;
  - one *Ifm Electronic O3D302* depth camera;
  - one *Intel Realsense D435* depth camera;
  - one *Microsoft Kinect v2* depth camera.

Figure 2.1 shows some overall pictures of the robotic cell from different points of view, whereas Figure 2.2 shows some close-ups of the inside of the robotic cell. In Figure 2.3 the vision sensors mounted inside the cell are highlighted. For a more detailed overview of the design and components of the robotic cell, see Appendix A.

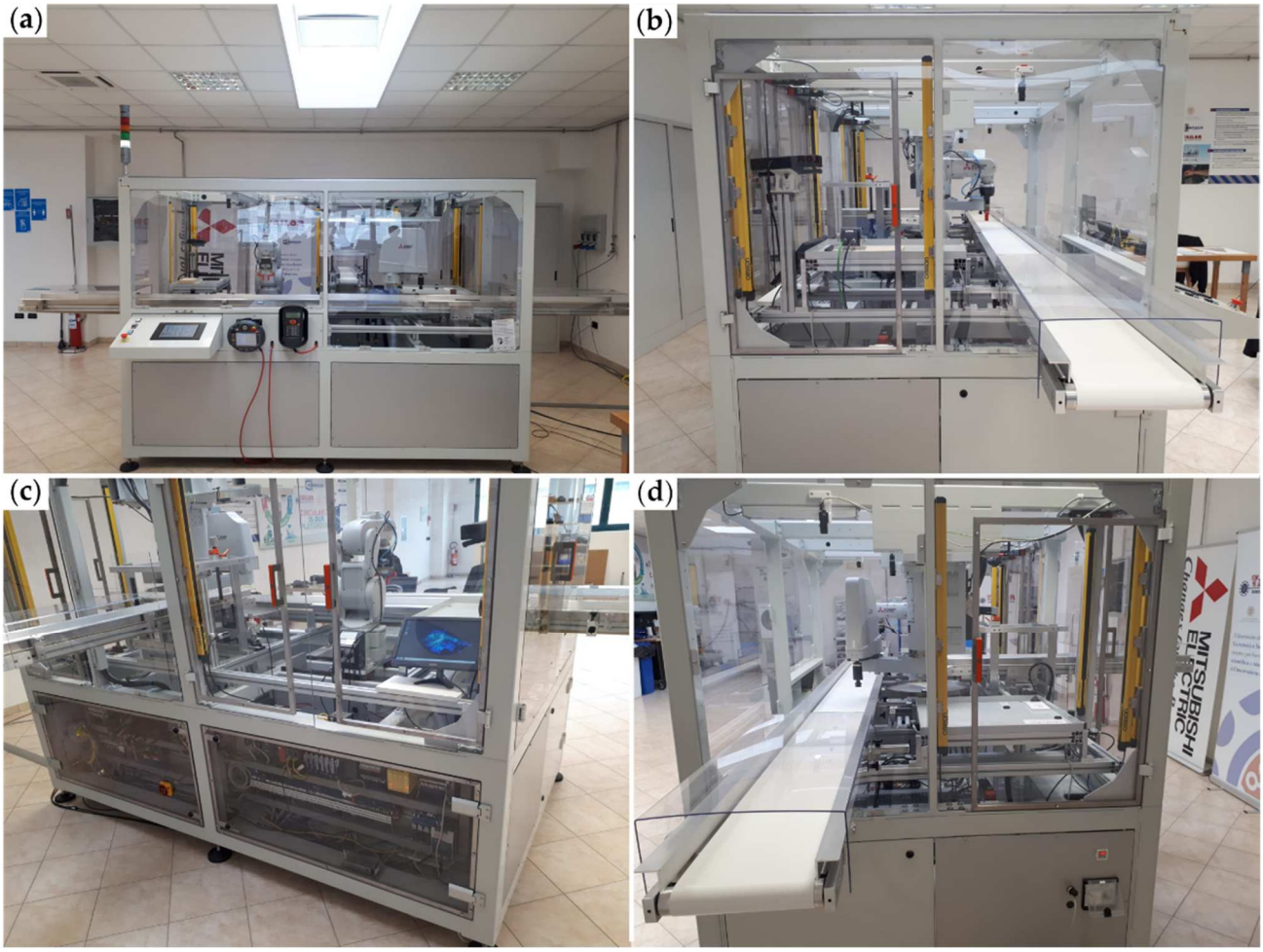


Figure 2.1 Various overall pictures of the robotic cell inside the *TAILOR* laboratory. (**a**) front view; (**b**) left view; (**c**) rear view; (**d**) right view.

Figure 2.2 Close-ups on the inside of the robotic cell, highlighting in particular the *Mitsubishi Electric RV4F* articulated robot (**c**) and the *Mitsubishi Electric RH1F* robot (**d**).

Figure 2.3 Mounting of the vision sensors highlighted, with indications on the type and on the model.

2.2 External workstation and programming frameworks

All the programming is carried out on an external computer, a *Dell Precision 5820 Tower* workstation, featuring an *Intel® Core™ i7-9800X* 3.8 GHz octa-core processor, 16.5 MB of cache, 32 GB (4 x 8 GB) RAM of DDR4 at 2.666 MHz. This external workstation is connected to the robotic cell Ethernet switch, allowing the communication with the various devices inside the cell. The *Robot Operating System* (*ROS*) middleware was chosen as a tool to implement the advanced functionalities required by the research applications, more specifically regarding the developed collision avoidance method. *ROS* was used to efficiently manage the CPU multithreading and acted as a wrapper for the code written in the C++ programming language. The C++ language is a common choice in real-time applications involving vision-based robot control because of both its high performances in terms of speed and the availability to users of versatile and efficient artificial vision and robotics libraries. The communication with the robot controller, managed with the aid of *ROS*, takes place via a UDP socket, enabled thanks to a particular functionality of the robot controller. This allows a data exchange between the external pc and the robot controller with a 7.1 ms cycle time. One advantage of coding in C++ is the possibility to exploit the *Graphics Processing Unit* (GPU) resources, which allow to parallelize certain general-purpose computations and to significantly speed them up, a GPU usage commonly known as *General Purpose GPU* (GPGPU). In practice, this is achieved by exploiting the *Compute Unified Device Architecture* (CUDA), which is a parallel computing platform and application programming interface model created by *NVIDIA* (the only requirement to use it is to have a *NVIDIA* graphics card). This is a promising approach adopted in some recent works in this field, such as [25]. To fully exploit the CUDA capabilities, the workstation was endowed with the recent and powerful graphic card *NVIDIA GeForce RTX 3070,* showed in Figure 2.4, which also reports its main technical specifications. *Linux Ubuntu 18.04* was chosen as operating system. In addition, *MATLAB®* was extensively used as an aid to develop and test several algorithms involved, other than to fully implement some of the algorithms that did not need to run online.

Core clock	1815 MHz (reference card: 1725 MHz)
CUDA cores	5888
Memory clock	14000 MHz
Memory size	8 GB
Memory type	GDDR6
Memory bus	256 bit
Memory bandwidth	448 GB/s
Card bus	PCIe 4.0

Figure 2.4 *NVIDIA GeForce RTX 3070*

3 Collision Avoidance

In this Chapter the developed collision avoidance method is presented. It is inspired by the very recent *safety constraint* approach [15], which is a safety-oriented approach particularly suitable for industrial robots. The method is developed for an articulated robot and tested inside the robotic cell described in Section 2.1 on the *Mitsubishi Electric RV4F* robot. It exploits the depth data from the *Intel Realsense D435* (active IR stereo camera) and the *Microsoft Kinect v2* (ToF camera), shown in Figure 3.1, which also reports their main depth features. The RGB stream, even if available from both the cameras, was not used.

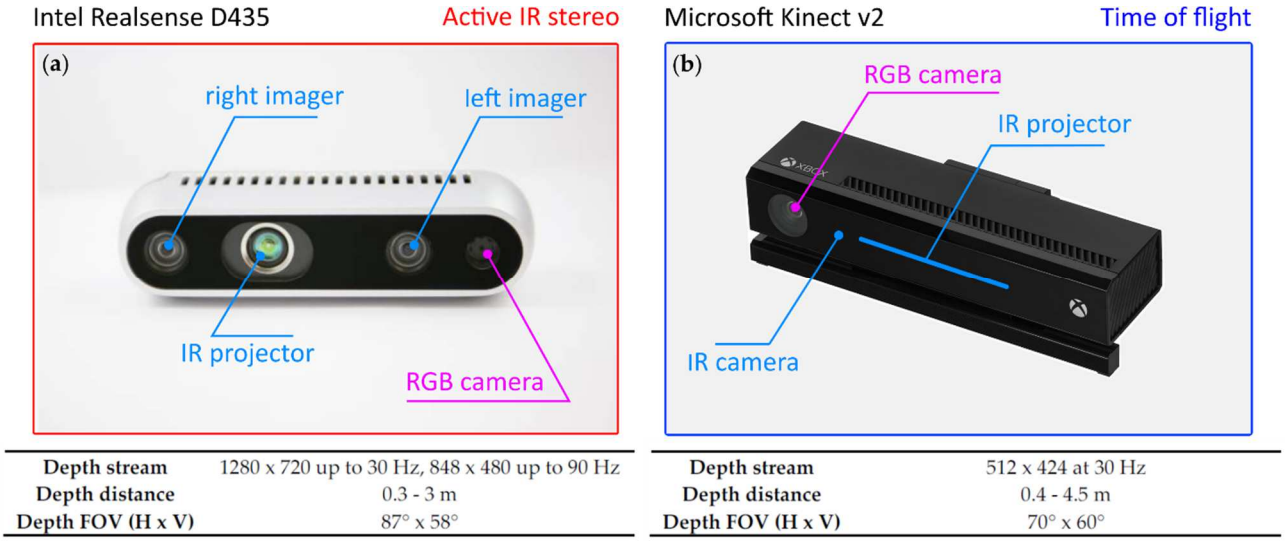


Figure 3.1 Depth sensors used for the collision avoidance method.

The method was developed on the external workstation described in Section 2.2, by using both *MATLAB®* and *ROS*, this latter run on *Linux Ubuntu 18.04*. Inside the *ROS* framework, the code was written in the C++ language, by exploiting the IDE *VS Code*. In Figure 3.2, a high-level pipeline of the method is shown. The method can be conceptually divided into two main blocks, one designed for the *online trajectory generation* and one designated for the *obstacle tracking*. The robot normally follows a pre-programmed task-based reference trajectory; in order to keep the PSD, an adjustment of the robot trajectory, which can be in terms of path (which diverges from the reference one by means of an escape motion) and/or speed, is produced whenever necessary based on the monitored positions and velocities of obstacles possibly present within the robot workspace. The *obstacle tracking* block relies on the use of a voxel grid, in which the occupancy of each voxel is estimated starting from the point clouds acquired by the two depth cameras. By exploiting the voxel occupancy, the obstacles speed is then estimated by means of a particle filter. Based on real-time estimation of positions and speeds, an obstacle segmentation is then

performed, which outputs a set of obstacles and some corresponding relevant quantities, used as an input in the *online trajectory generation* block.

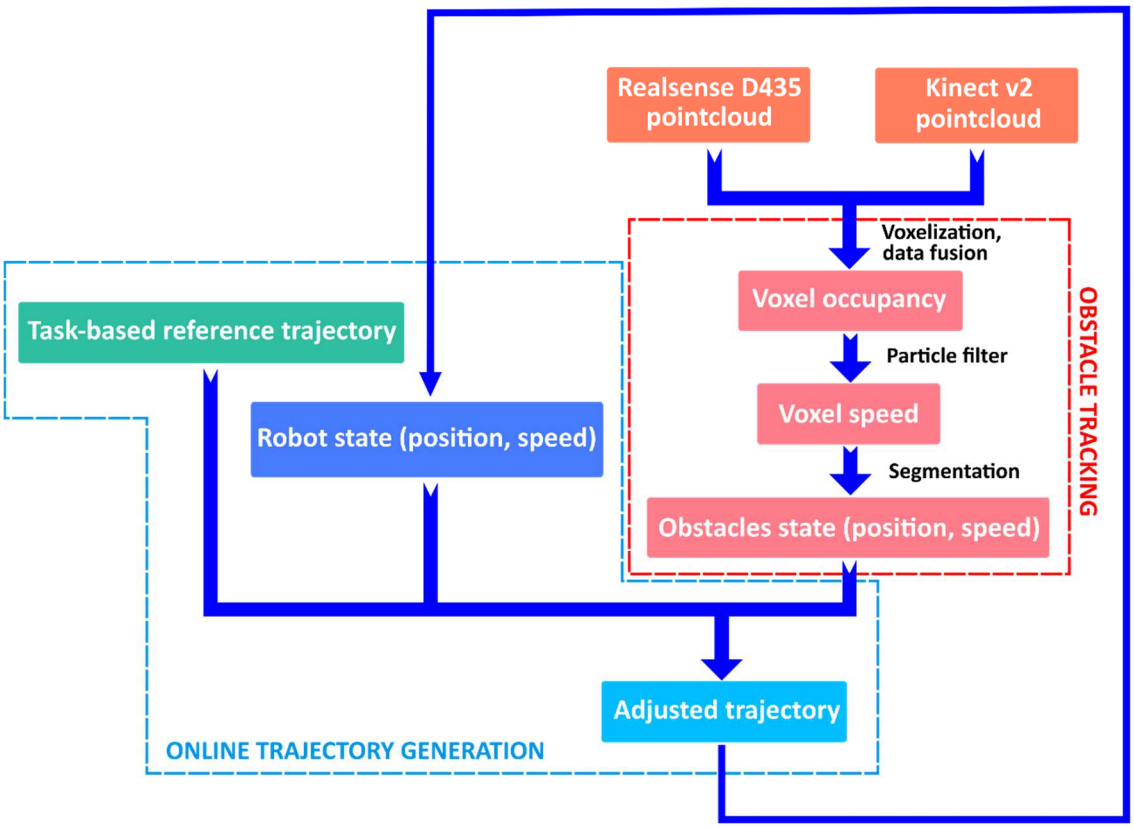


Figure 3.2 High level pipeline of the collision avoidance method.

The proposed implementation introduces some novelties, and has the following distinctive features:

- It is voxel-based.
- It is suitable for generic dynamic obstacles since it does not rely on the typical human-tracking algorithms.
- The voxel-based approach allows an easy fusion of data from different sensors, which can rely on different technologies, thus potentially increasing reliability.
- It can prioritize escape-motions or adherence to the reference trajectory.

Most recent collision avoidance methods, which generates online command at each discrete time step, favour the use of acceleration-based input commands [15,59]. The hypothesis normally made is that the controller closed-control loop is sufficiently reliable and performant, so that there is no significant difference (relatively to the application to develop) between the commanded and the actual trajectory. Typically, modern robot controllers accounts for the possibility of commanding the robot in velocity or acceleration, which results in smoother motion laws. In the developed method, the trajectory generation algorithms were partly affected by the need of

adapting to the external communication modality of the *RV4F* controller, a *Mitsubishi Electric CR751-Q*, featuring a sample time of 7.1 ms. This controller accounts for the possibility of commanding the robot by a direct communication with an external PC, via a UDP socket. The communication modality (named *MXT*, which stands for *Move External*) is highlighted in Figure 3.3 (image from [94]). This modality allows one to command the robot by means of position values, in terms of either joint angles or the Cartesian pose of the *Tool Center Point* (TCP).

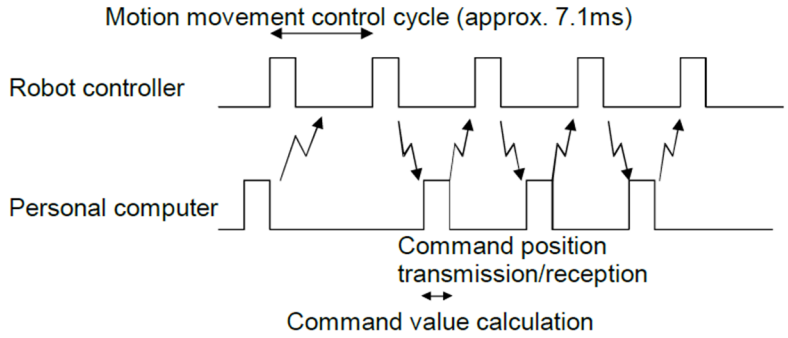


Figure 3.3 Data exchange modality between the robot controller and the personal computer (image from [94]).

The command value generated online at each time step and sent to the robot controller is used as reference in an internal closed-control loop managed by the controller, which exploits the joint encoder feedback values. The available data exchange modality (position) was seen as an opportunity to add originality to the trajectory generation method, since the adherence to this modality fostered variations with respect to the literature methods taken as reference. One of its perks, for instance, was the fact that it demonstrated to be particularly suitable for implementing strategies relying on the motion geometry instead of on the motion laws, which can have some benefits, as will be discussed in Section 3.5.

In the next Section, the online trajectory-generation algorithms are detailed, considering at first the case of a point-like obstacle. Later, in the obstacle tracking Section, the case of real-world obstacles is considered, the algorithms to fuse and transform the data detailed, and the methods for estimating the obstacles positions and speeds presented.

3.1 Online trajectory generation

3.1.1 RV4F robot kinematics

The trajectory generation make use of the robot kinematics, here detailed in the case of the *Mitsubishi Electric RV4F*, which is a wrist partitioned articulated robot. Its mechanical architecture and dimensions are shown in Figures 3.4-3.5.

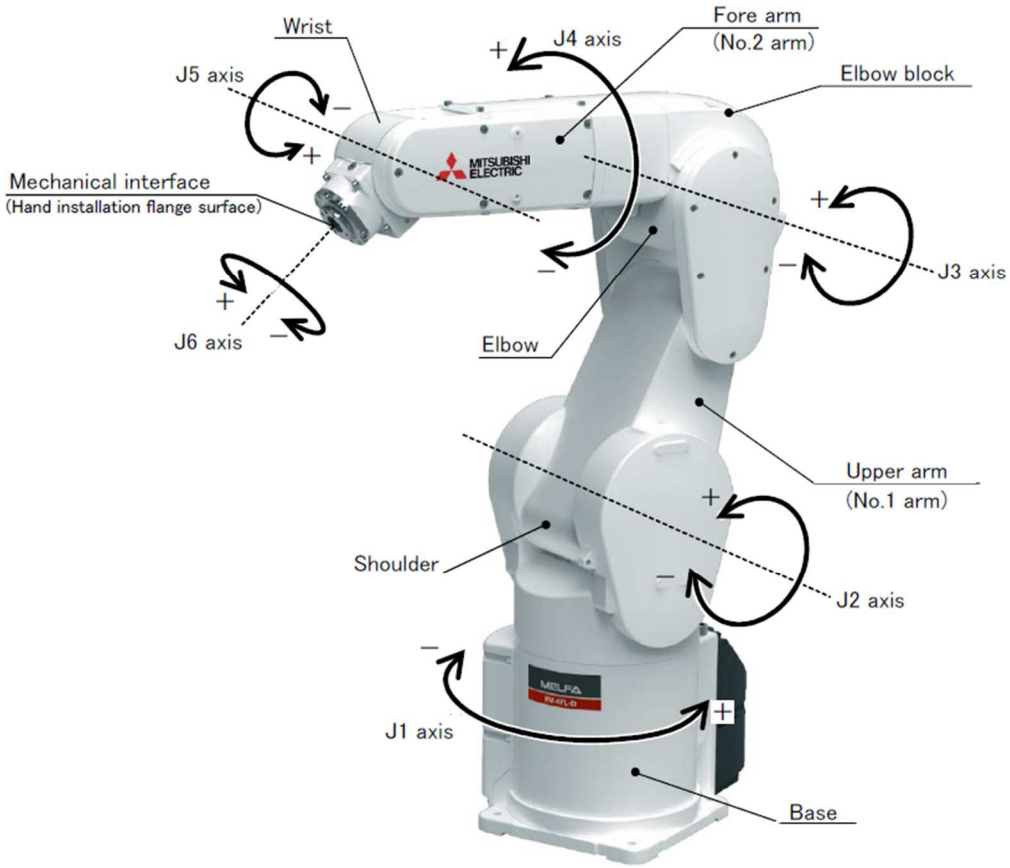


Figure 3.4 *Mitsubishi Electric RV4F* robot architecture, with axes highlighted (image from [95]).

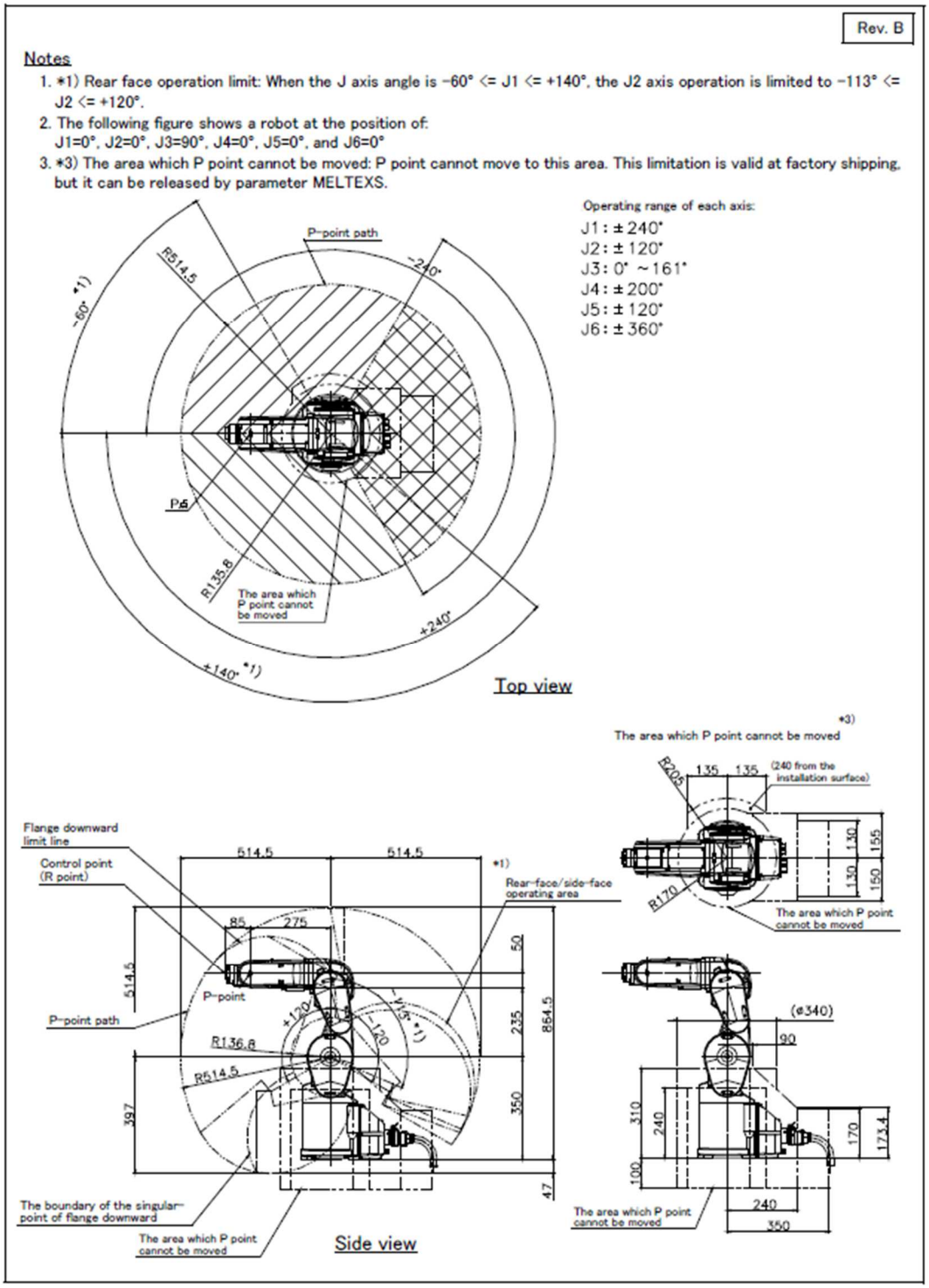


Figure 3.5 *Mitsubishi Electric RV4F* mechanical drawing of the architecture, with highlighted the axis distances and axes limits (image from [95]).

The solution of the robot kinematics can be computed by means of the *Denavit-Hartemberg convention* (DH) [96]. In order to do so, a robot model based on the DH parameters is to be constructed, which was done as shown in Figure 3.6a. Figure 3.6b shows the zero-joint-angles configuration, obtained by adding proper offsets to the joint angles. The corresponding DH parameters are reported in Table 3.1. Other quantities reported in Figure 3.6b are:

O_k: origin of the k^{th} *Reference Frame* (RF) RF_k

e_k: axis of the joint k

E_k: matrix representing the pose of RF_k with respect to the *Robot Base* reference frame (RF_0)

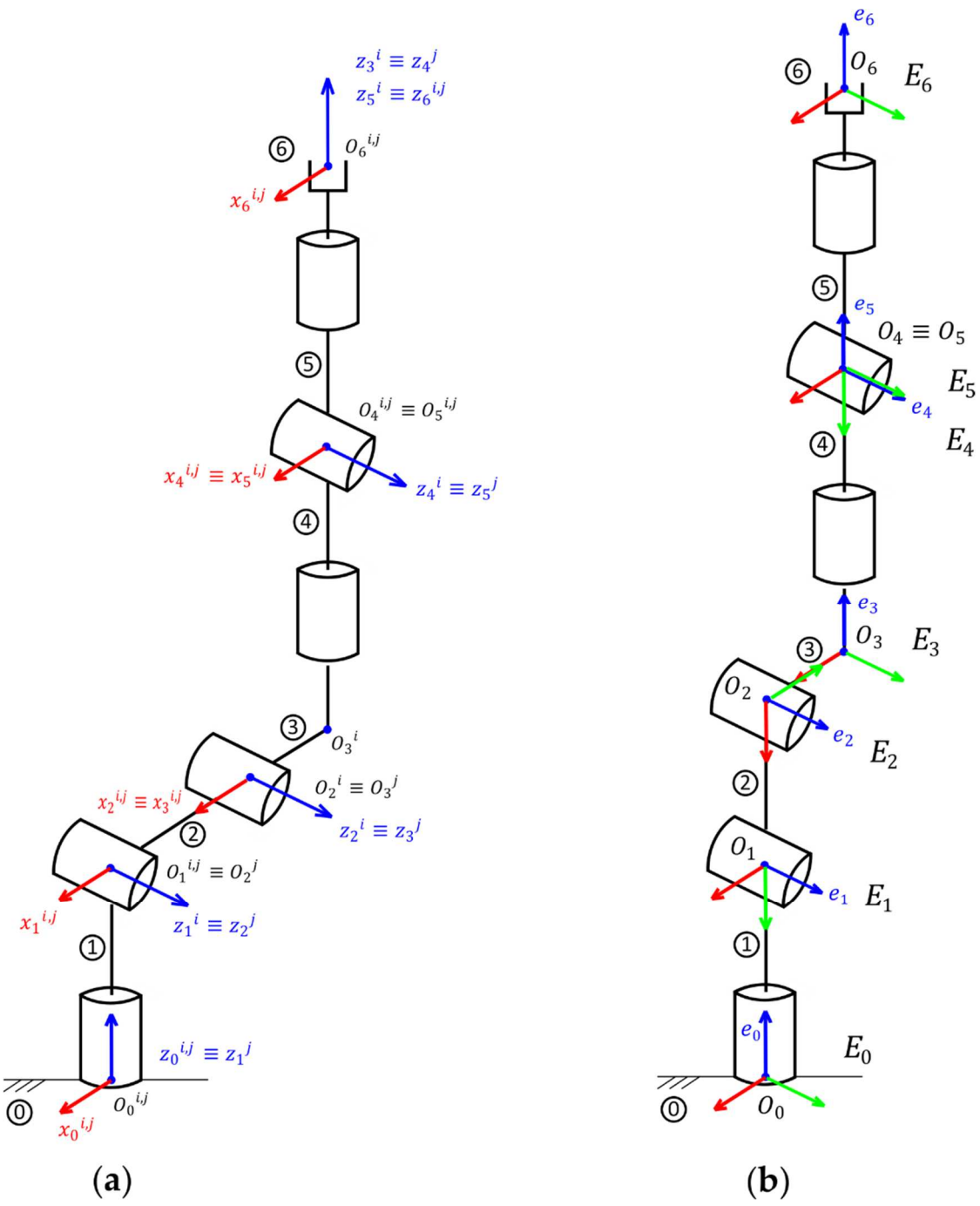


Figure 3.6 *RV4F* DH parameters.

Table 3.1 Table summarizing the *RV4F* DH parameters.

k	a_k	α_k	d_k	θ_k	offset
1	0	-π/2	L_1	θ_1	0
2	$-L_2$	0	0	θ_2	π /2
3	$-L_3$	π /2	0	θ_3	-π/2
4	0	-π/2	L_4	θ_4	0
5	0	π /2	0	θ_5	0
6	0	0	L_6	θ_6	0

where L_1 = 350 mm, L_2 = 235 mm, L_3 = 50 mm, L_4 = 275 mm and L_6 = 85 mm (L_6 = 85+ 191.1 mm if considering the TCP placed on the e_6 axis at the end of the gripper jaws).

Methods for solving the kinematics and other considerations, for instance concerning the singularities, can be found in [96, 97].

3.1.2 Pre-programmed reference trajectory

The reference trajectory is the trajectory of the pre-programmed task, that the robot would normally follow inside the automated working cycle. Typically, this type of trajectories can be programmed inside the robot proprietary offline-programming software, which provides a robot simulator, a proprietary programming language and other tools. There, the most common approach to program robot movements is to define a series of waypoints, usually in the Cartesian space, and the type of movement the robot must perform between them. In the vast majority of cases, the motion laws used to construct the reference trajectory are not defined from scratch, but rely on the use of already available functions, typically:

- *joint interpolated motion*: given a series of waypoints, the laws of motion are generated by interpolating the waypoints in the joint space;
- *linear Cartesian motion*: given two waypoints, the robot follows a straight line in the Cartesian space to move from one to the other.

Inside the proprietary *Mitsubishi Electric* robot programming software *RTToolBox3*, using the proprietary *MELFA* programming language, the joint interpolated motion is identified with the keyword *mov* (move), whereas the linear Cartesian movement with the keyword *mvs* (move straight). For the sake of conciseness, these two keywords will be henceforth utilized to identify these two motion types.
The *mov* motion is typically used in situations in which one has a certain freedom of movement between the waypoints, it is not affected by *configuration singularities* and the *inverse kinematics* needs to be applied only to convert the waypoints from the Cartesian space to the joint space.

The command *mvs*, on the other hand, it is used when the robot needs to necessarily follow a straight line in the Cartesian space, for example in insertion tasks, object picking, etc. The joint commands to send to the robot actuators are obtained by dividing the straight line in the Cartesian space in a high number of points, which approximate the line, for each of which the corresponding joint angles are computed by means of the robot *inverse kinematics*. In this case, configuration singularity problems could arise.

In the case study, there was the necessity of constructing reference trajectories outside the software *RTToolbox3*, so one or more motion types had to be chosen. The main point of the collision avoidance method was to adjust the robot motion based on obstacles information; the *mvs* movement, however, is used when the reference trajectory is strictly required to follow a straight line, often with a required speed, precondition for the fulfilment of that specific part of the task. Thus, since it was more appropriate for the scope of this Thesis, the reference trajectory was constructed by means of the only *mov* motion. Note however that the proposed collision avoidance method can be easily modified to account for reference trajectories based on other motion types, that can be the other typical ones or can be specifically tailored for the task.

The *mov* motion can rely on different types of interpolation, a common choice is the use of cubic splines. Here the *Piecewise Cubic Hermite Interpolating Polynomial* (PCHIP) [98] was used since it produced motion laws similar to the *mov* function of *RTToolbox3*. PCHIP has the advantage of having no overshoot and less oscillation compared to other cubic interpolations methods, however, in the junction points only C^1 *continuity* (continuity up to the first derivative) is granted, which may not be sufficient in some specific cases.

Let us consider the k^{th} joint, the abscissa τ (which can be seen as a dimensionless time), and m waypoints (thus m *-1* spline pieces). The spline interpolating the k^{th} joint target positions has then the form shown in Eq. (3.1):

$$S_k(\tau) = \begin{cases} a_{k,1}\tau^3 + b_{k,1}\tau^2 + c_{k,1}\tau + d_{k,1} & if\ T_1 \le \tau < T_2 \\ a_{k,2}\tau^3 + b_{k,2}\tau^2 + c_{k,2}\tau + d_{k,2} & if\ T_2 \le \tau < T_3 \\ \dots & \\ a_{k,m-1}\tau^3 + b_{k,m-1}\tau^2 + c_{k,m-1}\tau + d_{k,m-1} & if\ T_{m-1} \le \tau < T_m \end{cases} \quad (3.1)$$

The 6-dimensional spline is then a function S: $\mathbb{R} \rightarrow \mathbb{R}^6$, defined according to Eq. (3.2):

$$S(\tau) = \begin{bmatrix} S_1 \\ \dots \\ S_6 \end{bmatrix} \tag{3.2}$$

When considering the totality of the joints, it is useful to organize the spline coefficients into a matrix $C_S \in \mathbb{R}^{6\times[4\times(m-1)]}$, in the following way:

$$C_S = \begin{bmatrix} a_{1,1} & b_{1,1} & c_{1,1} & d_{1,1} & \dots & a_{1,m-1} & b_{1,m-1} & c_{1,m-1} & d_{1,m-1} \\ & & & & \dots & & & & \\ a_{k,1} & b_{k,1} & c_{k,1} & d_{k,1} & \dots & a_{k,m-1} & b_{k,m-1} & c_{k,m-1} & d_{k,m-1} \\ & & & & \dots & & & & \\ a_{6,1} & b_{6,1} & c_{6,1} & d_{6,1} & \dots & a_{6,m-1} & b_{6,m-1} & c_{6,m-1} & d_{6,m-1} \end{bmatrix} \tag{3.3}$$

The matrix C_S and the vector $T = [T_1, \dots, T_m]$, this latter containing the abscissa of each waypoint, completely define the reference trajectory.

These two quantities were computed offline using *MATLAB®*, by specifying a set of waypoints in the joint space and the corresponding T vector, and by carrying out the interpolation by means of the *MATLAB®* built-in function *pchip*. T can be constructed in several ways, for example in order to minimize the working cycle time, or simply by an arbitrary choice of the time interval between the waypoints. In this context, T_i (i = 1, …, m) can be considered as a matter of fact equal to the time instant t_i at which the robot is desired to reach each waypoint.

Let us define a series of variables that will be extensively used from now on:

$i \in \mathbb{N}$ — discrete time step. It will also be used as subscript for other variables to indicate the reference to the time step i.

$\Delta t \in \mathbb{R}$ — robot controller sample time (≈ 7.1 ms).

$u \in \mathbb{R}^6$ — command variable: it is the vector of joint coordinates sent to the robot controller at each time step i. The robot controller moves the robot to the commanded position within the next time step $i+1$. If the robot cannot move to the commanded joint position within the time step $i+1$, the controller outputs the encountered error (e.g. due to the fact that the needed joint speed exceeds the joint speed limit).

In absence of obstacles, u can be computed at each time step i as shown in Eq. (3.4). In this situation, the robot simply follows the pre-programmed task-based reference trajectory.

$$u = S(i\Delta t) \tag{3.4}$$

3.1.3 Safety constraints

In this Section the constraints on the command variable u that grant the compliance with the PSD are obtained.

Figure 3.7 shows the convention used when considering a discrete time domain, by showing the relation between generic positions x and the corresponding velocities v. Inside the time step interval, the approximation of considering constant velocities is made.

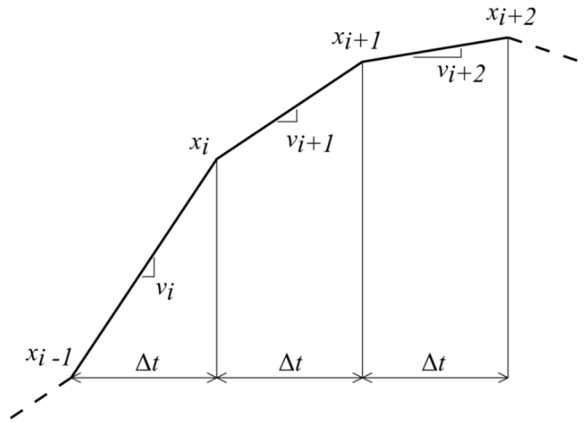


Figure 3.7 Discrete domain conventions and approximations.

In the next Section, the case of a single robot point and a single obstacle point is considered in the computation of the safety constraints. After that, the case of the whole robot body will be addressed. The case of real-word obstacles will be addressed later in the Thesis, in Section 3.2.

3.1.3.1 Single robot-point

Let us consider a point-like obstacle and the robot TCP (but the following discussion can be adapted to a generic robot point). Figure 3.8 depicts the positions and the velocity vectors of the robot point ("rb" subscript) and the obstacle "ob" subscript), in the Cartesian space.

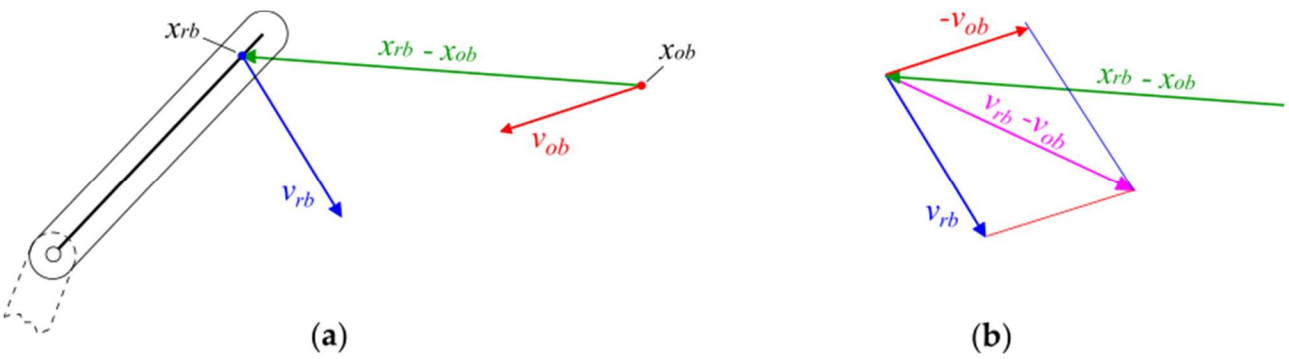


Figure 3.8 Robot ("rb" subscript) and obstacle ("ob" subscript) positions and velocity vectors.

Eq. (3.5), which is the starting equation, is a simplified version of Eq. (1), which gives the general expression for the PSD. Here the simplified version adopted in [15] is used.

$$d_{i+1} \geq -\dot{d}_{i+1}\Delta T_S + \Delta \tag{3.5}$$

where:

T_S is the robot worst-case stopping time;

d_{i+1} is the distance between the robot and the obstacle at the time step i + 1;

$\dot{d}_{i+1}$ is the robot-obstacle relative speed at the time step i + 1;

Δ is a constant positive offset on the separation distance, which can be seen as the PSD to guarantee at robot-obstacle relative speed equal to zero.

Considering the convention of Figure 3.7, d_{i+1} can be computed according to Eq. (3.6).

$$d_{i+1} = d_i + \dot{d}_{i+1}\Delta t \tag{3.6}$$

The relative velocity can be expanded according to Eq. (3.7), where the operator · denotes the scalar product:

$$\dot{d}_{i+1} = (v_{rb_{i+1}} - v_{ob_{i+1}}) \cdot \hat{k} \tag{3.7}$$

where $\hat{k}$ is the unit-vector pointing in the robot-obstacle direction and can be estimated according to Eq. (3.8):

$$\hat{k} \approx \frac{x_{rb_i} - x_{ob_i}}{\left\| x_{rb_i} - x_{ob_i} \right\|} \tag{3.8}$$

Substituting Eq. (3.6) in Eq. (3.5) and rearranging the terms leads to Eq. (3.9):

$$\dot{d}_{i+1}(T_S + \Delta t) \geq \Delta - d_i \tag{3.9}$$

Substituting Eq. (3.7) in Eq. (3.9), expanding and rearranging the terms leads to Eq. (3.10):

$$v_{rb_{i+1}} \cdot \hat{k} \geq \frac{\Delta - d_i}{T_S + \Delta t} + v_{ob_{i+1}} \cdot \hat{k} \tag{3.10}$$

The Cartesian velocity can be computed from the joint velocity, as shown in Eq. (3.11).

$$v_{rb_{i+1}} = J_t(q_{i+1})\dot{q}_{i+1} \tag{3.11}$$

where J_t is the tangential part of the Jacobian, which has dimension 3 x 6, and q identifies the joint coordinates.

The approximations of Eqs. (3.12-3.13) are made:

$$J_t(q_{i+1}) \approx J_t(q_i) \tag{3.12}$$

$$v_{ob_{i+1}} \approx v_{ob_i} \tag{3.13}$$

The joint velocity can be computed according to Eq. (3.14).

$$\dot{q}_{i+1} = \frac{q_{i+1} - q_i}{\Delta t} \tag{3.14}$$

By combining Eqs. (3.10-3.14), Eq. (3.15) is obtained:

$$[J_t(q_i)q_{i+1}] \cdot \hat{k} \geq \frac{\Delta t}{T_S + \Delta t}(\Delta - d_i) + v_{ob_i} \cdot \hat{k}\Delta t + J_t(q_i)q_i \cdot \hat{k} \tag{3.15}$$

By exploiting the commutative property of the scalar product and the fact that it can be rewritten in terms of a row and a column vector multiplication leads to Eq. (3.16):

$$\hat{k}^T J_t(q_i)q_{i+1} \geq \frac{\Delta t}{T_S + \Delta t}(\Delta - d_i) + \hat{k}^T v_{ob_i}\Delta t + \hat{k}^T J_t(q_i)q_i \tag{3.16}$$

q_{i+1} represents the command variable u, and Eq. (3.16) is a linear inequality with respect to it, since it has the form shown in Eq. (3.17).

$$au \geq b \tag{3.17}$$

where:

$$u = q_{i+1} \qquad \in \mathbb{R}^{6x1} \tag{3.18}$$

$$a = \hat{k}^T J_t(q_i) \qquad \in \mathbb{R}^{1x6} \tag{3.19}$$

$$b = \frac{\Delta t}{T_S + \Delta t}(\Delta - d_i) + \hat{k}^T v_{ob_{i+1}}\Delta t + \hat{k}^T J_t(q_i)q_i \qquad \in \mathbb{R} \tag{3.20}$$

Eq. (3.16), or equivalently Eqs. (3.17-3.20) represent the safety constraints considering one robot point and one obstacle point.

3.1.3.2 Multiple robot-points

The adherence to the constraint of Eqs. (3.17-3.20) guarantees that the robot point considered is always at a safe separation distance from the point-like obstacle considered. However, the safe separation distance is to be guaranteed for all the robot dangerous moving parts. To address this, one solution can be to consider a set of spheres encapsulating the robot dangerous moving parts, by properly choosing their centres and radiuses. Figure 3.9 shows an example of four spheres, the centres and radiuses of which are chosen to encapsulate the most dangerous moving part of the robot. A higher number of spheres can be considered to include all the moving parts and/or to better approximate a properly enlarged robot shape. The use of spheres instead of other geometrical shapes such as cuboids or capsules to approximate the robot shape is adopted for the sake of simplicity and efficiency, since it has a direct connection with Eqs. (3.17-3.20). Indeed, each safety constraint on the generated command has the following geometrical interpretation: the command is generated so that the sphere of radius Δ (cf. Eq. (3.20)) centred on the robot point considered will not come into contact with the obstacle point considered.

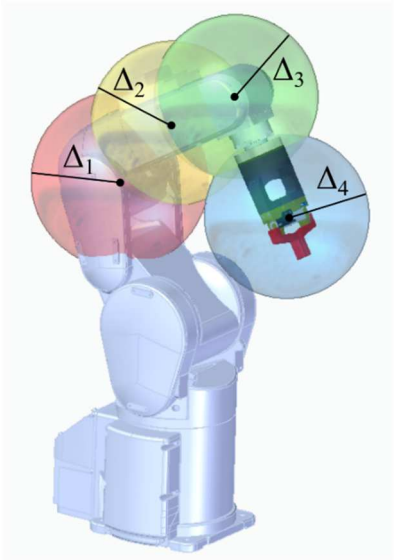


Figure 3.9 Four spheres used to encapsulate the robot.

Once the sphere centres and radiuses are defined, they can be used to construct a set of safety constraints. Let us suppose the quantity J_t and x_{rb} are already available for each robot point O_k of Figure 3.6, one can then construct the safety constraints for the generic robot point P, lying between O_{k-1} and O_k, in the following way:

- Δ is set equal to the radius of the sphere centred in P;
- J_t and x_{rb} can be computed according to Eqs. (3.21-3.23), which exploits the concept of natural coordinates $s \in [0, 1]$.

$$x_P = x_{O_{k-1}} + s(x_{O_k} - x_{O_{k-1}}) \tag{3.21}$$

$$J_{t_P} = J_{t_{O_{k-1}}} + s(J_{t_{O_k}} - J_{t_{O_{k-1}}}) \tag{3.22}$$

where:

$$s = \frac{\|P - O_{k-1}\|}{\|O_k - O_{k-1}\|} \tag{3.23}$$

Let us suppose n_s spheres are used to encapsulate the robot; this results in a set of n_s safety constraints, that can be organized in the form shown in Eq. (3.24).

$$\Gamma u \geq \beta \tag{3.24}$$

where $\Gamma \in R^{n_s \times 6}$ is a matrix constructed by concatenating by rows the vector a of Eq. (3.19) of each safety constraint, u is the command vector and $\beta \in R^{n_s \times 1}$is a vector containing the scalar b of Eq. (3.20) of each safety constraint.

3.1.4 Constraints on joint position, velocity, acceleration

Other than the safety constraints, the generated joint command $u\ (= q_{i+1})$ must satisfy other constraints, here described.

3.1.4.1 Joint position constraint

u must contain angles that lie within the *RV4F* joint limits, reported in Table 3.2.

Table 3.2 Joint position limits.

	q_1	q_2	q_3	q_4	q_5	q_6
Lower bounds	-240°	-120°	0°	-200°	-120°	-360°
Upper bounds	+240°	+120°	+161°	+200°	+120°	+360°

Eq. (3.25) shows the constraint on the joint position, where q_{lb} e q_{ub} are the lower and upper joint position bounds, respectively:

$$q_{lb} \leq q_{i+1} \leq q_{ub} \tag{3.25}$$

3.1.4.2 Joint velocity constraint

The *RV4F* joint velocity limits are reported in Table 3.3. The maximum velocity is the same in both the joint motion direction, so only the module is reported.

Table 3.3 Joint velocity limits.

$\dot{q}_{1\_max}$	$\dot{q}_{2\_max}$	$\dot{q}_{3\_max}$	$\dot{q}_{4\_max}$	$\dot{q}_{5\_max}$	$\dot{q}_{6\_max}$
450°/s	450°/s	300°/s	540°/s	623°/s	720°/s

The *RV4F* joint velocity that results from u must lie within the joint speed limits, which can be expressed with Eq. (3.26), which leads to the constraint on u of Eq. (3.27).

$$\dot{q}_{lb} \leq \frac{q_{i+1} - q_i}{\Delta t} \leq \dot{q}_{ub} \tag{3.26}$$

$$q_i + \dot{q}_{lb}\Delta t \leq q_{i+1} \leq q_i + \dot{q}_{ub}\Delta t \tag{3.27}$$

3.1.4.3 Joint acceleration constraint

If considering constant velocity between two time-steps, one wants to find a command that limits the velocity variation. The bounds on the acceleration can be expressed as shown in Eq. (3.28):

$$\ddot{q}_{lb} \leq \frac{\dot{q}_{i+1} - \dot{q}_i}{\Delta t} \leq \ddot{q}_{ub} \tag{3.28}$$

The velocities can be expressed according to Eq. (3.29):

$$\dot{q}_{i+1} - \dot{q}_i = \frac{q_{i+1} - q_i}{\Delta t} - \frac{q_i - q_{i-1}}{\Delta t} = \frac{q_{i+1} - 2q_i + q_{i-1}}{\Delta t} \tag{3.29}$$

which leads to the constraint on u of Eq. (3.30):

$$\ddot{q}_{lb}\Delta t^2 + 2q_i - q_{i-1} \leq q_{i+1} \leq \ddot{q}_{ub}\Delta t^2 + 2q_i - q_{i-1} \tag{3.30}$$

or equivalently, of Eq. (3.31):

$$\ddot{q}_{lb}\Delta t^2 + \dot{q}_i\Delta t + q_i \leq q_{i+1} \leq \ddot{q}_{ub}\Delta t^2 + \dot{q}_i\Delta t + q_i \tag{3.31}$$

Given that the bounds on the position, velocity and acceleration intersect (which must be previously verified), the three bounds can be combined to form a unique bound, in the way shown in Eqs. (3.32-3.34).

$$q_{lb\_tot} = \max\left\{q_{lb}, (q_i + \dot{q}_{lb}\delta t), (q_i + \dot{q}_i\delta t + \ddot{q}_{lb}\delta t^2)\right\} \tag{3.32}$$

$$q_{ub\_tot} = \min\left\{q_{ub}, (q_i + \dot{q}_{ub}\delta t), (q_i + \dot{q}_i\delta t + \ddot{q}_{ub}\delta t^2)\right\} \tag{3.33}$$

$$q_{lb\_tot} \leq q_{i+1} \leq q_{ub\_tot} \tag{3.34}$$

3.1.5 Optimization problem

The safety constraints can be used to constraint the joint angles commanded to the robot so that the PSD is always kept. The generated command must also satisfy the constraints derived from the limitations on the joint position, velocity and acceleration, which, as seen, can be combined into the single constraint of Eq. (3.34). A constraint of this type is a particular type of linear inequality constraint, known in literature as *simple bounds*. Both the safety constraints, organized in the matrix form of Eq. (3.24) and the simple bounds of Eq. (3.34) act on the pre-programmed reference trajectory, that the robot follows whenever it is admissible. The aim of this part is to set up a proper optimization problem which can generate a trajectory that satisfy all the constraints while minimizing the distance to the pre-programmed reference trajectory. The underlying mathematical problem involved is a *Quadratic Program* (QP), which is an optimization problem with a quadratic objective function and linear constraints [99]. In this specific case it can be stated as a linear-inequality-constrained least-norm problem, which has the form of Eqs. (3.35-3.37). In the case study, the norm represents the distance between the command and a reference point on the pre-programmed task trajectory.

$$\min_x \frac{1}{2}\|Cx - d\|^2 \qquad C \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n, d \in \mathbb{R}^n \tag{3.35}$$

s.t.

$$Ex \geq f \qquad E \in \mathbb{R}^{m\times n}, f \in \mathbb{R}^m \tag{3.36}$$

$$x_{lb_k} \leq x_k \leq x_{ub_k} \qquad k = 1, \dots, n \tag{3.37}$$

Instead of the form presented in Eq. (3.35), the QP objective function is usually presented in the form of Eq. (3.38) [99]:

$$\min_x \frac{1}{2}x^T G x + x^T c \qquad G \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n, c \in \mathbb{R}^n \tag{3.38}$$

where G is a symmetric n x n matrix, named *Hessian matrix* and c and x are vectors in $\mathbb{R}^n$.

It can be shown that the form of Eq. (3.35) can be converted in the form of Eq. (3.38), in the following way:

$$\min_x \frac{1}{2}\|Cx - d\|^2 = \min_x \frac{1}{2}(Cx - d)\cdot(Cx - d) = \min_x \frac{1}{2}Cx \cdot Cx - \frac{1}{2}Cx \cdot d - \frac{1}{2}Cx \cdot d + d \cdot d$$
$$= \min_x \frac{1}{2}(Cx)^T Cx - (Cx)^T d = \min_x \frac{1}{2}x^T C^T Cx - x^T C^T d$$

Given the formulation of Eq. (3.35), the Hessian matrix G and the vector c can be computed according to Eq. (3.39) and Eq. (3.40), respectively:

$$G = C^T C \tag{3.39}$$

$$c = -C^T d \tag{3.40}$$

In the case study, the C matrix is the identity matrix I, whereas the d vector is the reference joint vector. If the Hessian matrix is positive semidefinite, the QP is said to be *convex*, which significantly reduces its difficulty (it is the case, and, more precisely, since the identity matrix is positive definite, the QP is said to be *strictly convex*). Eqs. (3.41-3.43) shows the final form in which the optimization problem is presented, detailed for the case study:

$$\min_u \frac{1}{2} u^T G u + u^T c \qquad G\ (=I) \in \mathbb{R}^{n \times n}, u \in \mathbb{R}^n, c\ (=-q_{ref}) \in \mathbb{R}^n \tag{3.41}$$

s.t.

$$\Gamma u \geq \beta \qquad \Gamma \in \mathbb{R}^{n_s \times n}, \beta \in \mathbb{R}^{n_s} \quad \text{safety constraints} \tag{3.42}$$

$$q_{lb\_tot_k} \leq u_k \leq q_{ub\_tot_k} \qquad k = 1, \dots, n \quad \text{simple bounds} \tag{3.43}$$

Note on the dimension n:

Note that, if the TCP is chosen on the joint axis e_6 (see Figure 3.6), J_t does not depend on q_6. This is common in practical scenarios and will be considered as hypothesis. If TCP-related quantities do not depend on q_6, neither q_6 appears on the safety constraints of the other robot points; in fact, if considering robot points with decreasing distance to the robot base, the safety constraints will depend on a decreasing number of joint angles. This means that the last column of the matrix Γ contains all zeros. That being the case, for the sake of computation speed, n was set equal to 5 in Eqs. (3.41-3.43), adopting a different strategy for the command angle u_6, which was generated according to Eq. (3.44):

$$u_6 = \max\left\{q_{6_{lb\_tot}}, \min\left\{q_{6_{ref}}, q_{6_{ub\_tot}}\right\}\right\} \tag{3.44}$$

QP resolution

The QP of Eqs. (3.41-3.43) is to be solved at each time step. In the final *ROS* implementation, the problem was solved by means of *qpOASES* [100], which is an open-source C++ implementation of the online active set strategy [101], which is one of the methods for solving a QP.

It is worth noting that a custom *MATLAB®* implementation of the active set strategy for convex QP, based on the algorithm presented in [99, p. 472], was exploited to carry out preliminary tests of the method. The implementation, specifically tailored for the case study, outperformed the more general *MATLAB®* built-in function *lsqlin* (designed to solve constrained linear least-squares problems), which had proved to be too slow for the intended tests.

Choice of q_{ref}

At each time step, the reference joint position vector q_{ref} is to be chosen along the reference pre-programmed task trajectory. Two ways of choosing it are proposed, a conventional one based on the motion laws and an original one based on a geometric approach. Both are detailed in the next Section.

3.1.6 Reference trajectory heading point

At each time step i, the term *heading point*, denoted with q_{ref_i}, is used to identify the value, belonging to the reference trajectory, to which the robot tries to head (minimize the joint space distance to it, while adhering to the various constraints, according to Eqs. (3.41-3.43)).

3.1.6.1 Heading point computation – laws of motion approach

This is the conventional approach, based on the motion laws: at each time step, the heading point is computed considering the current time instant as abscissa of the reference pre-programmed trajectory (which is the 6-dimensional spline S, see Section 3.1.2), according to Eq. (3.45):

$$q_{ref_i} = S(i\Delta t) \tag{3.45}$$

If $i\Delta t$ becomes higher than T_m, q_{ref_i} is kept equal to the spline end-point $S(T_m)$, until the robot reaches it. Since tasks are normally implemented as cycles, a similar process is then repeated, resetting the value of i and considering a (symmetrical) spline joining the end of S with the beginning of S. Once the robot reaches the beginning, the cycle is closed and can be repeated for the wanted number of times. This logic is applied as well to the case examined in the next Section.

3.1.6.2 Heading point computation – geometrical approach

In this approach, the heading point on the reference trajectory is chosen based on the minimum distance between the current joint position vector and the reference trajectory, in the joint space. In the next part, a method for finding the spline point of minimum distance with respect to a given external point is detailed. For sake of

simplicity, let us consider a specific time step and, at first, a single spline piece (which has dimensionality of six). The expression of the squared distance D between the current robot joint position q_0 and the spline piece considered can be computed as a function of the abscissa τ, as shown in Eq. (3.46).

$$D(\tau) = \sum_{k=1}^{6} (a_{1_k}\tau^3 + a_{2_k}\tau^2 + a_{3_k}\tau + a_{4_k} - q_{0_k})^2 \tag{3.46}$$

To find the τ that minimizes D, one can compute the first derivative of Eq. (3.46) with respect to τ and search for stationary points. The derivative of D with respect to τ is shown in Eqs. (3.47-3.53):

$$D'(\tau) = (\sum_{k=1}^{6} c_{5_k})\,\tau^5 + (\sum_{k=1}^{6} c_{4_k})\,\tau^4 + (\sum_{k=1}^{6} c_{3_k})\,\tau^3 + (\sum_{k=1}^{6} c_{2_k})\,\tau^2 + (\sum_{k=1}^{6} c_{1_k})\,\tau + (\sum_{k=1}^{6} c_{0_k}) \tag{3.47}$$

where:

$$c_{5_k} = 6{a_{1_k}}^2 \tag{3.48}$$

$$c_{4_k} = 10 a_{1_k} a_{2_k} \tag{3.49}$$

$$c_{3_k} = 8 a_{1_k} a_{3_k} + 4{a_{2_k}}^2 \tag{3.50}$$

$$c_{2_k} = 6 a_{2_k} a_{3_k} + 6 a_{1_k}\left(a_{4_k} - q_{0_k}\right) \tag{3.51}$$

$$c_{1_k} = 2{a_{3_k}}^2 + 4 a_{2_k}(a_{4_k} - q_{0_k}) \tag{3.52}$$

$$c_{0_k} = 2 a_{3_k}(a_{4_k} - q_{0_k}) \tag{3.53}$$

Eq. (3.47) represent a fifth-grade polynomial. Let us consider a spline piece indexed by *j* (*j* = 1, …, *m*-1), and defined in the closed interval $[T_j, T_{j+1}]$. The adherence to all the *Conditions 1-4* is sufficient for τ_0 to be a local minimizer:

- *Cond. 1*: $\Im(\tau_0) = 0$ — τ_0 has imaginary part equal to zero
- *Cond. 2*: $\tau_0 \in]T_j, T_{j+1}[$ — τ_0 belongs to the spline-piece open interval
- *Cond. 3*: $D'(\tau_0) = 0$ — τ_0 is a stationary point
- *Cond. 4*: $D''(\tau_0) > 0$ — τ_0 is a local minimum

Cond. 2 comes from the fact that in the spline-piece junction points the second derivative is not continuous, so if *Conds. 1,3,4* hold true but τ_0 is a juncture point, no conclusion can be drawn. Also, if $D''(\tau_0) = 0$ but *Conds. 1,3* and $\tau_0 \in [T_j, T_{j+1}]$ hold true, no conclusion can be drawn. In these cases, τ_0 can be stored, along with the τ values that satisfy all *Conds. 1-4*, for successive comparisons. For each spline piece, a

set of candidates can be stored this way. The global minimizer can then be identified by evaluating which candidate corresponds to the minimum distance, also considering the extrema of the whole spline, which can be global minimizers even without being stationary points.
Here the final algorithm for finding the minimum distance between a point and a spline is outlined:

//Initialization of τ_dmin, done considering the spline extrema ($\tau = T_1$, $\tau = T_m$).
$\tau\_dmin = T_1$
if $(D(T_1) \leq D(T_m))$ $\tau\_dmin = T_m$

//Computing the candidates for each spline piece and choosing the one with min distance
for each spline piece *j*

$\tau\_cand_j = \{\tau \mid D'(\tau) == 0\}$

$\tau\_cand_j \leftarrow \{\tau\_cand_j \mid \Im\,(\tau\_cand_j) == 0\}$

$\tau\_cand_j \leftarrow \{\tau\_cand_j \mid \tau\_cand_j \in [T_j, T_{j+1}]\}$

$\tau\_cand_j \leftarrow \{\tau\_cand_j \mid (D''(\tau\_cand_j) \geq 0 \;||\; \tau\_cand_j == \text{junction point})\}$

$\tau\_cand_j \leftarrow \min\{\tau\_cand_j\}$

if $(\tau\_cand_j \leq \tau\_dmin)$ $\tau\_dmin = \tau\_cand_j$

end for

In practical applications, the trajectory can be usually defined through a low number of waypoints, which results in a low number of spline pieces. The core computations take place in the process of extracting the roots of Eq. (3.47). Eq. (3.47) is a quintic function, and the *Abel-Ruffini theorem* [102] states that there is no algebraic expression (that is, in terms of radicals), for the solutions of general quintic equations over the rational numbers. Nevertheless, several numerical methods are available for estimating the roots of a generic n^{th} degree polynomial. However, since this computation needed to be carried out at each time step and for each spline piece, an efficient method was required. The C++ implementation [103] of the iterative method [104], specifically designed to find the solutions of quintic equations, was exploited, since it proved to be effective and particularly fast (it outputs all the solutions, even the complex ones, which motivates *Cond. 1*). Note however that, if the spline pieces are especially numerous, it may be more efficient to use other methods to find the minimum distance to the spline.
Once the τ corresponding to the minimum distance τ_{dmin} at the time step *i* is obtained, it is used to compute the heading point q_{ref_i}, according to Eq. (3.54):

$$q_{ref_i} = S(\tau_{dmin} + \Delta\tau) \tag{3.54}$$

where $\Delta\tau$ is a τ increment chosen to satisfy the *Conditions A*, *B*, *C*:

Cond. A: if the distance to the spline is zero (i.e. the current joint position vector belongs to the spline, at a certain τ, which corresponds to the τ_{dmin}), the reference point is to be computed so that the robot follows the motion laws in a nominal way, as in absence of external disturbances, that is to say according to Eq. (3.55):

$$\Delta\tau = \Delta t \quad if\ d_{min} = 0 \tag{3.55}$$

Cond. B: if the minimum distance is superior to a certain threshold, $\Delta\tau$ is to be proportional to the minimum distance. This condition derives from the idea of approaching the reference trajectory with a certain *heading angle*, which appeared a suitable and simple strategy.

Cond. C: the transition between *Cond. A* and *Cond. B* must be smooth.

These three conditions can be satisfied by constructing a function as shown in Eq. (3.56), and by guarantying the C^1 continuity in the junction point d_0.

$$\Delta\tau(d_{min}) = \begin{cases} a{d_{min}}^2 + b & if\ 0 \le d_{min} \le d_0 \\ d_{min}\mathrm{tg}\alpha & if\ d_{min} > d_0 \end{cases} \tag{3.56}$$

where, a, b, α and d_0 are parameters that have to be properly chosen. The angle α can be chosen by the user depending on the wanted heading angle; from the condition of Eq. (3.55) derives the fact that $b = \Delta t$. a and d_0 can be determined by imposing the continuity of both the function and its derivative at the juncture point d_0, which results in the system of Eq. (3.57) of two equations and two unknowns.

$$\begin{cases} a{d_0}^2 + b = d_0\mathrm{tg}\alpha & \text{continuity of } \Delta\tau \text{ in } d_0 \\ 2ad_0 = \mathrm{tg}\alpha & \text{continuity of } \Delta\tau' \text{in } d_0 \end{cases} \tag{3.57}$$

This system can be easily solved for example by isolating d_0 from the second expression and substituting it into the first one to then solve for a.
It results that:

$$a = \frac{1}{4b}\mathrm{tg}^2\alpha \tag{3.58}$$

$$d_0 = \frac{2b}{\mathrm{tg}\alpha} \tag{3.59}$$

Eq. (3.56) finally becomes:

$$\Delta\tau(d_{min}) = \begin{cases} \frac{1}{4\Delta t}\mathrm{tg}^2\alpha {d_{min}}^2 + \Delta t & if\ 0 \le d_{min} \le \frac{2\Delta t}{\mathrm{tg}\alpha} \\ d_{min}\mathrm{tg}\alpha & if\ d_{min} > \frac{2\Delta t}{\mathrm{tg}\alpha} \end{cases} \tag{3.60}$$

Figure 3.10a shows the graph of the function of Eq. (3.60), considering $\alpha = 30°$ and Δt = 0.0071 s. Figure 3.10b depicts a simplified two-dimensional example of the approaching joint trajectory generated (blue trajectory). It is traced starting from an initial perturbed joint position until it reaches a simple linear reference trajectory (cyan dashed line).

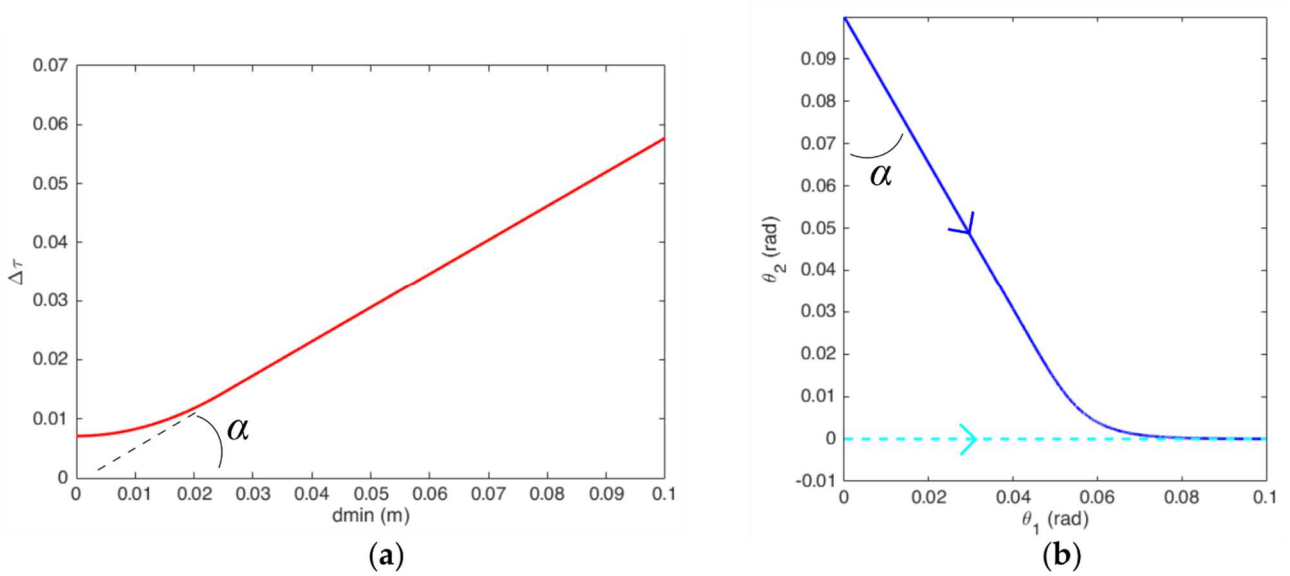


Figure 3.10 (**a**) $\Delta\tau$ as a function of the minimum spline distance; (**b**) two-dimensional example of the approaching joint trajectory generated (blue trajectory), which reaches a simple linear reference trajectory (cyan dashed line).

3.2 Obstacle tracking

In this Section, the method developed to handle real-world obstacles is presented. The scope is to define a procedure aimed at detecting all the generic dynamic obstacles in the scene and estimating their position and velocity, a procedure commonly known as *obstacle tracking*. These estimations are then used as inputs in the trajectory generation block, in a way that will be detailed in the last part of this Section.

As for the online trajectory generation, in parts of this Section there will be references to discrete time steps, once again denoted with the letter i, and with duration Δt. In this case, however, Δt will not refer to the controller sample time but to the inverse of the camera frame rate. The *Kinect v2* run at ≈ 30 fps, whereas the *Realsense D435*,

can achieve higher frame rates. Nonetheless, for the scope of this Thesis, the *Realsense* frame rate was set to 30 fps and its depth resolution at 848 x 480, a choice mainly motivated by experimental evaluations. Each camera acquisition and processing cycle was managed by *ROS* independent nodes (for an overview of the *ROS* architecture see Appendix B); after a *data fusion* stage, computations are performed inside *ROS* nodes running at a specific frequency (≈ 35 fps). After the data fusion, time step will thus refer to the time step of these latter *ROS* nodes.
In addition, for the sake of conciseness, henceforth the *Realsense D435* camera will be referred to *Realsense*, whereas the *Kinect v2* will be referred to as *Kinect*. Furthermore, the subscript *1* and *2* will be used, in some occasions, to refer to the *Realsense* and the *Kinect*, respectively.

3.2.1 Control volume

The first step was the definition of the space region here called *control volume*, namely a volume which identifies the robot operational zone, which is to be monitored by the cameras. The control volume was placed in the rear-part of the robotic cell, which gives the chance of interacting with the robot thanks to the movable guards (cf. Figure 2.1c). The control volume was constructed as a cuboid and placed as shown in Figure 3.11 (red volume); the robot workspace (in blue), is also highlighted, referred to a TCP positioned at the extremity of the gripper jaws; the robot is depicted with the J_1 axis rotated by 180 degrees.

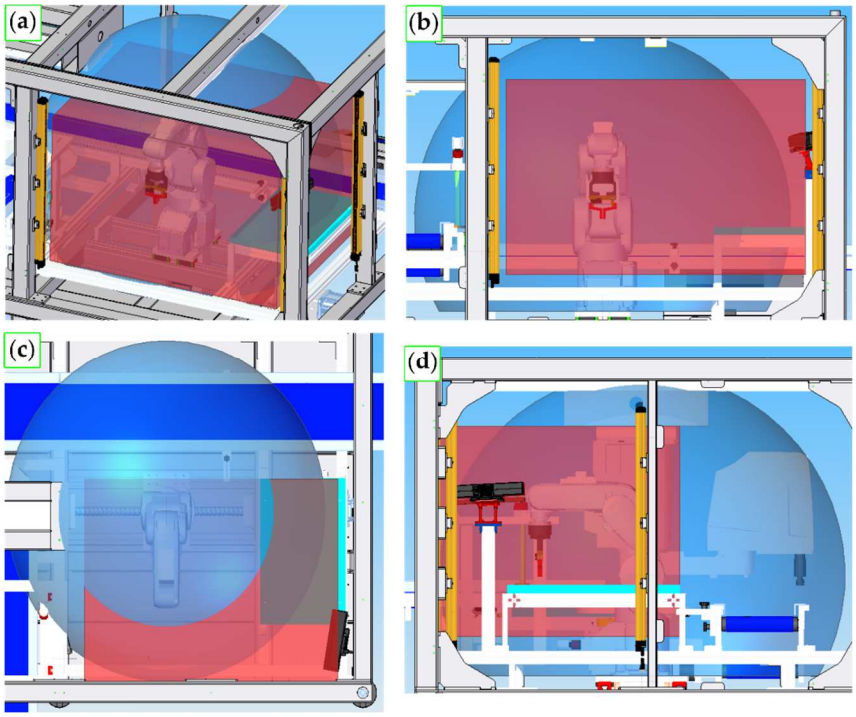


Figure 3.11 Four different view of the control volume (in red). The robot workspace, referred to a TCP positioned at the extremity of the gripper jaws, is shown in blue. The robot is depicted with the axis J_1 rotated by 180 degrees.

The control volume bounds and dimensions are listed in Table 3.4, referring to the *Robot Base RF*.

Table 3.4 Lower bounds (lb), upper bounds (ub) and dimensions (dim), in the x, y and z direction of the control volume, with respect to the *Robot base RF*.

	lb (mm)	**ub** (mm)	**dim** (mm)
x	-712.50	135.50	848.0
y	-768.75	370.75	1139.5
z	144.25	859.75	715.5

3.2.2 Camera placement

Once defined the control volume, the two depth cameras were placed so that the union of their camera viewing frustum completely covers the control volume. The concept of camera frustum is depicted in Figure 3.12a: it is the volume defined by the blue lines, which represents the space region that the camera can frame. Depth cameras typically have a blind spot near the optical centre, so this volume is a truncated pyramid, fully defined by the Horizontal and Vertical FOV and by the depth range, both reported in Figure 3.1 for the camera models used. Figure 3.12b and Figure 3.12c show an isometric and a top view, respectively, of the camera placement with respect to the control volume, in which the camera viewing frustums are highlighted.

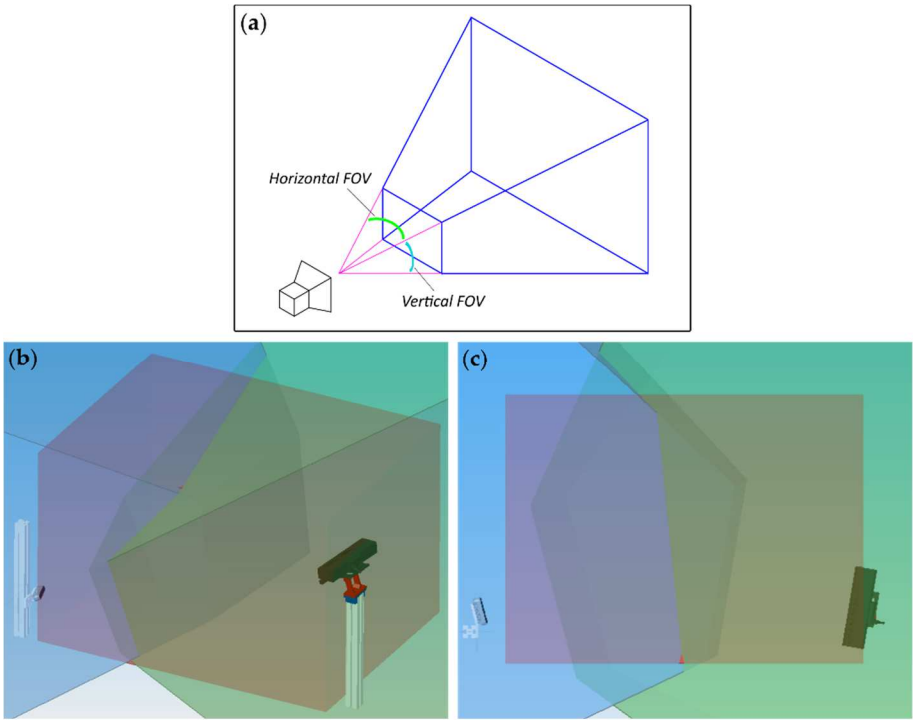


Figure 3.12 (**a**) Illustration of the concept of camera frustum; (**b**), (**c**): two views of the camera placement (*Realsense* at the left, *Kinect* at the right) with respect to the control volume (cuboid in red), with highlighted their viewing frustum (in blue the one of the *Kinect*, in green the one of the *Realsense*).

It is worth pointing out that the camera placement has important implications on the safety, since it affects the visibility and determines whether an obstacle is spotted or not inside the area that one wants to monitor. In an ideal scenario, the cameras have to be placed to maximize the control volume coverage, to frame potential keypoints with sufficient quality and to minimize the possibility of occlusions. Some studies for the optimal placement of fixed cameras are available in literature [105,106]. This topic will be better analysed in Chapter 5 of the Thesis, in a different case study. In general, it is clear that the positioning and dimensions of the control volume inevitably determines both the minimum number of cameras needed for an effective monitoring and their placement. In the case addressed in this Section, given the simple geometry of the control volume, the usage of only two cameras and some constraints on the available mounting places, the placement was made based on the sensibility of the Candidate, with successive refinements relying on CAD visualization tools and practical experimental tests.

3.2.3 Camera extrinsic calibration

Camera calibration is a fundamental procedure omnipresent in computer vision. In Appendix C some basic notions and nomenclature are reported. For a detailed discussion, see for example [29].

The two depth cameras output the point clouds with respect to their reference frame, but they need to be referred to the *Robot base RF*, so that obstacle-related quantities can be used in the trajectory generation block.
The method here used for the extrinsic calibration exploits the fact that both the cameras used contain an RGB sensor; since the RGB sensor and the depth module are separated entities, each one has its own *Camera RF*, henceforth named *Depth camera RF* and *RGB camera RF*. In the case study, extrinsic calibration consists in the process of estimating:

A_{d1}^{r} the homogeneous matrix representing the pose of the *Depth camera RF* of the *Realsense* depth module with respect to the *Robot base RF*;

A_{d2}^{r} the homogeneous matrix representing the pose of the *Depth camera RF* of the *Kinect* depth module with respect to the *Robot base RF*.

This process was carried out in two steps, one used for an initial estimation of A_{d1}^{r} and A_{d2}^{r} and the second one for a refinement of the transformations found.

3.2.3.1 Step 1: estimation of an initial transformation

The first step, used for an initial estimation of A_{d1}^{r} and A_{d2}^{r} , involves the use of *ArUco* markers [107,108], acquired by means of the RGB sensors present on both the *Realsense* and the *Kinect* (see Figure 3.1). *ArUco* markers are 2D fiducial markers whose pose can be estimated starting from an RGB image in which the marker is visible and specific related algorithms, given that the intrinsic parameters of the RGB sensor are known. One *ArUco* marker was fixed on a robot part in such a way that could be framed by the RGB sensors of both the cameras (as shown in Figure 3.13), and so that its pose with respect to the *Robot base RF* could be easily estimated via CAD. Furthermore, for each camera also the rigid transformation between its own *RGB camera RF* and *Depth camera RF* was estimated. For the sake of conciseness, let us consider only the *Realsense* (the same applies to the *Kinect*); the pose of its *Depth camera RF* with respect to the *Robot base RF* can be computed according to Eq. (3.61):

$$A_{d1}^{r} = A_{m}^{r}(A_{m}^{rgb1})^{-1}A_{d1}^{rgb1} \tag{3.61}$$

where:

- A_{rgb1}^{r} is the pose of the *RGB camera RF* with respect to the *Robot base RF*.
- A_{m}^{r} is the pose of the *ArUco* marker with respect to the *Robot base RF* (estimated via CAD).
- A_{m}^{rgb1} is the pose of the *ArUco* marker with respect to the *RGB camera RF* (output of the *ArUco*-related function).
- A_{d1}^{rgb1} is the pose of the *Depth camera RF* with respect to the *RGB camera RF* (value that can be either found in literature/given by the manufacturer/estimated by specific function available in the camera libraries).

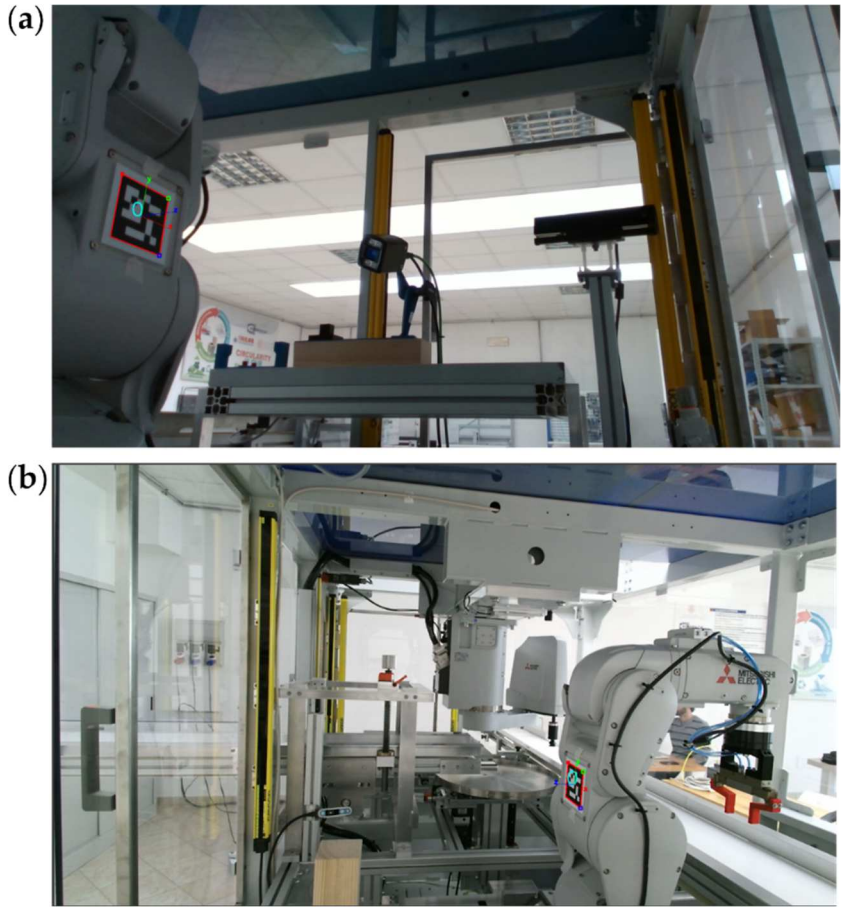


Figure 3.13 *ArUco* marker placement, edges recognition and reconstruction of its 3D pose.

3.2.3.2 Step 2: refinement

For each camera, the transformation obtained in *Step 1* was used as initial transformation for an *Iterative Closest Point* (ICP) algorithm [109]. ICP is a very well-known algorithm which allows one to align generic 3D shapes having the same geometry. Through a series of iterations, it finds the transformation that minimizes the distance between two sets of points. This algorithm is very versatile but works better if a good initial estimation of the transformation is given. Otherwise, in fact, it can easily get stuck in local minima, not properly converging to the right transformation. The ICP algorithm was run between a point cloud of the robot shape acquired by each camera and the CAD robot model, previously converted into a point cloud with an appropriate point density, by means of the open-source software *CloudCompare* [110]. For the acquisition, the robot was moved in a position in which each camera could acquire a significant amount of its shape with good quality. Before using the point clouds inside the ICP algorithm, a box filter was applied to keep only the robot shape, which was then filtered to remove the noise and properly downsampled. These routines were carried out in *MATLAB®*, and the results shown in Figure 3.14.

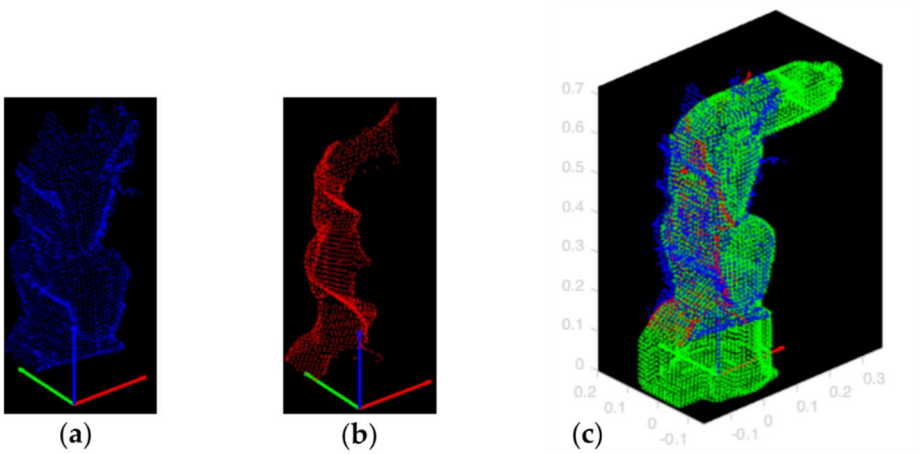


Figure 3.14 (**a**) Final robot shape obtained from the *Kinect* point cloud, (**b**) final robot shape obtained from the *Realsense* point cloud. (**c**) alignment of the point cloud (**a**) and (**b**) on the point cloud obtained from the robot CAD.

The final obtained value of A_{d1}^{r} and A_{d2}^{r}, other than the poses of the *Depth camera RF*s with respect to the *Robot base RF*, can be seen as the transformation matrices that can convert a point cloud, referred to the *Depth camera RF*, to the *Robot base RF*, according to Eq. (3.62).

$$\begin{bmatrix} X_1 & \dots & X_N \\ Y_1 & \dots & Y_N \\ Z_1 & \dots & Z_N \\ 1 & \dots & 1 \end{bmatrix} = A_{d1}^{r} \begin{bmatrix} X_{C_1} & \dots & X_{C_N} \\ Y_{C_1} & \dots & Y_{C_N} \\ Z_{C_1} & \dots & Z_{C_N} \\ 1 & \dots & 1 \end{bmatrix} \tag{3.62}$$

where N denotes the number of point cloud points.

Figure 3.15 summarizes the various steps of the extrinsic calibration.

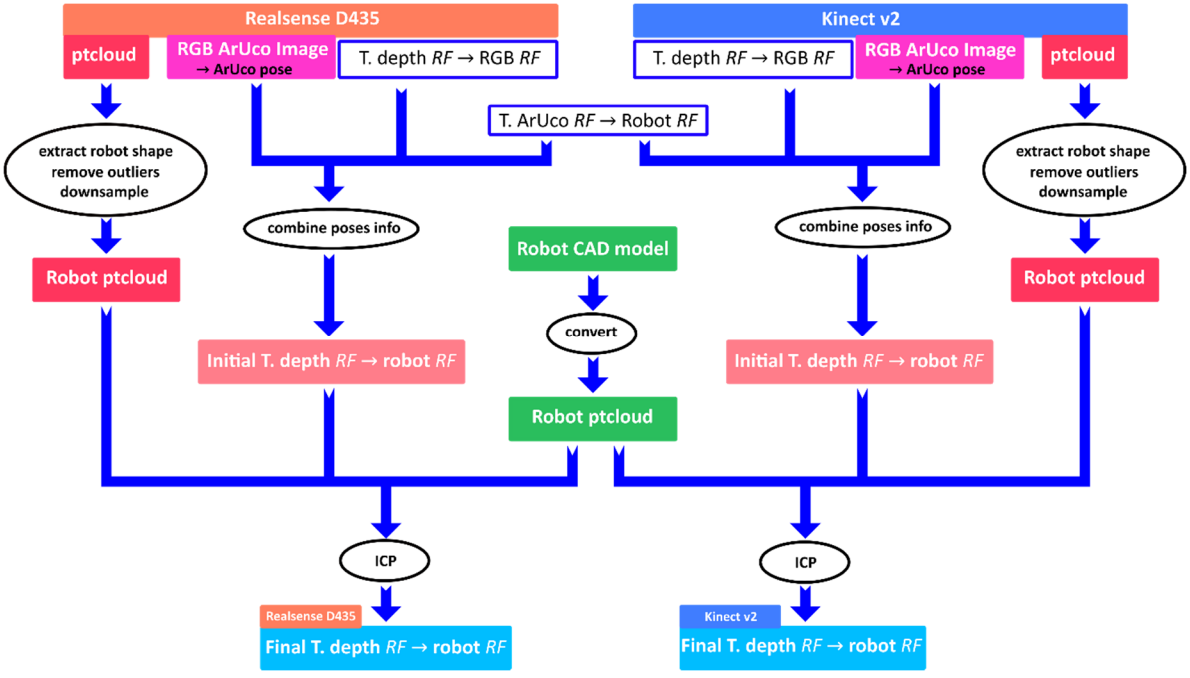


Figure 3.15 Conceptual pipeline of the extrinsic calibration method. "T." stands for Transformation.

3.2.4 Voxel-based data fusion

In this Section, the method to filter and fuse the 3D data acquired by the two depth cameras is detailed. The concept of control volume and the knowledge of the transformations estimated by means of extrinsic calibration in Section 3.2.3 are exploited. These latter transformations are computed offline and need to be re-computed for each sensor only if its pose, relatively to the *Robot base RF*, changes. In this Section the concept of voxel grid is introduced, and will serve as a basis for the development of the algorithms used to online manipulate the 3D data that the depth cameras output and to ultimately estimate the obstacle positions and velocities. The use of a voxel grid has the advantage of creating a structure that can be manipulated with simpler and faster algorithms (that can be GPU-parallelized), but has also other perks, such as the fact that acts itself as a spatial filter, in a way better detailed later. A notable example of the use of voxel-based GPU-accelerated algorithms can be found in [111, 112], in which Hermann et al. exploit them for collision detection and mobile manipulation planning.

3.2.4.1 Creation of a voxel grid

The control volume defined in Section 3.2.1 is divided into a set of cubic voxels of the same side length L_{vox}, set equal to 26.5 mm, resulting in a voxel grid, composed of N_x, N_y and N_z number of voxels on each side, as shown in Figure 3.16. A generic voxel of the grid is identified by a set of three indexes j_x (= 1, …, N_x), j_y (= 1, …, N_y), j_z (= 1, …, N_z) which are incremented with respect to a RF located at a grid vertex, henceforth named *Voxel grid RF*. These concepts are shown in Figure 3.16.

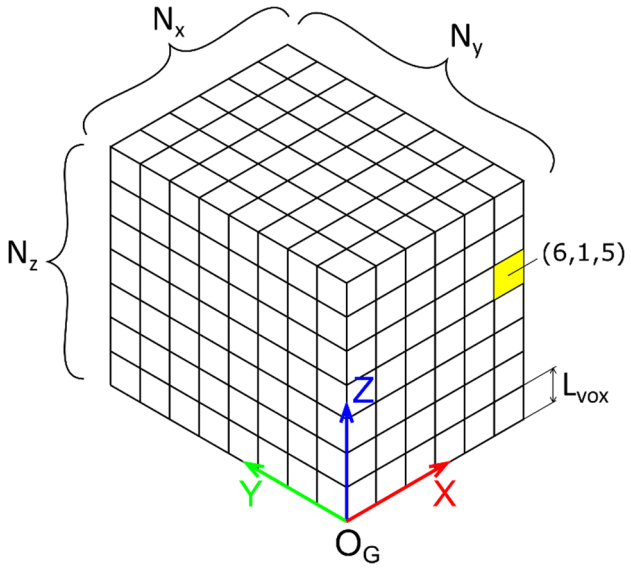


Figure 3.16 Voxel grid and related quantities.

This choice of L_{vox} is the result of a trade-off between different necessities. One is to limit the computational burden, which is affected by the number of voxels involved, which depends on L_{vox}. On the other hand, a too high value of L_{vox} might lead to an over-simplification of the obstacle shape and to consequent losses of precision in the various related estimations. However, if voxels are too small compared to the density of the sensors point cloud, there can be problem in the voxelization process, since might result in a series of sparse isolated voxels. The voxelization process consist in assigning an occupancy to each voxel corresponding to the number of point cloud points that are contained in that voxel. The occupancy can be seen as a measure of the confidence that a particular voxel actually contains a part of an obstacle (is *non-empty*). Among other perks, the use of a voxel grid has the advantage of creating a structure that acts as a spatial filter: voxel occupancies under a certain threshold δ can in fact be set to zero to filter out the spatial noise of the 3D data. The dimension of the voxels also affects the effectiveness of this spatial filter, hence becoming a further factor to account in the choice of L_{vox}. Furthermore, the voxel size also affects other stages of the method in a non-trivial way.

Given its complex role, the final value of L_{vox} was mainly decided based on empirical evaluations. Once a proper value was found, it was slightly tweaked, alongside with a first choice of the control volume dimensions (final ones reported in Table 3.4), to generate an integer number of voxels multiple of 32, which is the *warp size* of the most recent *NVIDIA* GPUs. This grants a minor improvement in speed in the GPU-parallelized algorithms, as reported in [113], which were used for the implementation of a *particle filter* (cf. Section 3.2.7).

The final values of voxel-grid features are summarized in Table 3.5. N_{vox} denotes the total number of voxels.

Table 3.5 Voxel grid features.

L_{vox} (mm)	N_x	N_y	N_z	N_{vox}
26.5	32	43	27	37152

It is worthy to point out that the voxel grid here described is a fixed structure, which makes it particularly suitable for GPU-accelerated algorithms. One different, thus common approach to speed up computations consists in exploiting *octrees* or *k-d trees*, which are dynamically growing data structures, most suitable for the programming paradigm typical of CPU algorithms, even if some GPU implementations have been proposed [112]. In the case study, the number of voxels and the algorithms involved suggested an approach based on a fixed voxel grid and GPU-accelerated algorithms. In the case of significantly higher number of voxels, which could arise for instance in the case of mobile robots or automotive applications, where a remarkably

higher volume is to be monitored, the use of an approach based on octrees or k-d tree might be preferred for the sake of efficiency [114].
In the actual implementation, arrays were used to store voxel-related quantities instead of a 3D voxel grid. Given that N_x, N_y, N_z are known, an index triplet (j_x, j_y, j_z), identifying a voxel of the voxel grid, can be converted into an index j in the voxel-related array, in the following way:

$$j = j_z N_y N_x + j_y N_x + j_x \tag{3.63}$$

In case, given the index j, one can revert back to the 3D index in the following way:

$$\begin{cases} j_x = \left(j\%\left(N_y N_x\right)\right)\%N_x \\ j_y = \left\lfloor \left(j\%\left(N_y N_x\right)\right)/N_x \right\rfloor \\ j_z = \left\lfloor j/\left(N_y N_x\right) \right\rfloor \end{cases} \tag{3.64}$$

where % denotes the modulo operator and $\lfloor x \rfloor$ is the *floor* operator, which takes as input a real number and gives as output the greatest integer less than or equal to x.

Henceforth, the index j will be used to index a voxel, considering that it refers to the voxel identified by the corresponding 3D index.

3.2.4.2 Filtering, voxelization, data fusion

The various filtering and processing need to be performed for each voxel at each time step, thus need to be efficient. One first expedient is to limit the number of 3D points involved more upstream possible in the pipeline, which is shown in its entirety in Figure 3.17. For this purpose, a first filter consists in a range filter, which, for each camera, allows to filter out the 3D points with a distance from the optical centre higher than a certain threshold. This filter was already available for both the *Realsense* and the *Kinect*. The threshold values were evaluated by means of a CAD assembly containing both the control volume and camera viewing frustum (cf. Figure 3.12). For each camera, the threshold distance is set equal to the maximum distance for which some points of its viewing frustum are contained in the control volume. After that, the point clouds are transformed into the *Robot base RF*, according to Eq. (3.62). Then, a box filter was applied to eliminate the 3D points outside the control volume. Next, the voxelization process was performed, separately for each camera. This process consisted in evaluating the number of 3D points inside each voxel. In order to do so, for a generic 3D point (X, Y, Z) referred to the *Robot base RF* one can compute the corresponding voxel index j, according to Eq. (3.65).

$$j(X,Y,Z) = \left\lfloor \frac{Z - O_{G_x}}{D_z} \right\rfloor N_y N_x + \left\lfloor \frac{Y - O_{G_y}}{D_y} \right\rfloor N_x + \left\lfloor \frac{X - O_{G_z}}{D_x} \right\rfloor \quad (3.65)$$

where:

D_x, D_y, D_z are the voxel grid dimensions;

$O_{G_x}, O_{G_y}, O_{G_z}$ are the coordinate of the origin of the *Voxel grid RF* (cf. Figure 3.16).

$\lfloor x \rfloor$ is the *floor* operator, which takes as input a real number and gives as output the greatest integer less than or equal to x.

For the *Realsense*, an occupancy vector n_1 (with length equal to N_{vox} and all elements initialized to zero) can be constructed as follows: for each point of the *Realsense* point cloud, the corresponding index j is computed, and the quantity $n_1[j]$ incremented by one. This way, at the end of the process, each cell of n_1 will contain the number of points of the *Realsense* point cloud contained in the voxel j. The same process is carried out for the *Kinect* point cloud, considering this time an array n_2. Then, a filter is applied to set to zero all the occupancies under a specific threshold (different for the two cameras and set empirically), to filter out part of the spatial noise. The next step consists in merging the two occupancies vectors n_1 and n_2 into a single one. For this purpose, the function of Eq. (3.66) was created, considering the following:

- The two occupancies n_1 and n_2 are to be manipulated so that they are comparable. In fact, each depth camera generates point clouds with different densities (which, in addition, could not be uniform in the space). This depends on the sensor technology and resolution.
- If, inside a voxel, points from both the *Realsense* and the *Kinect* are present, the resulting combination is to be greater than the sum of the occupancies, since the confidence benefits from the data heterogeneity. This is accounted by an additional term which is product of the single scaled occupancies.

$$\sigma = f(n_1, n_2) = \left\lfloor c_1 {d_1}^{\alpha} n_1 + c_2 {d_2}^{\beta} n_2 + c_3 {d_1}^{\alpha} {d_2}^{\beta} n_1 n_2 \right\rfloor \quad (3.66)$$

where:

σ, n_1, n_2, d_1, d_2 are vectors and their product and operators applied on them are to be intended *component-wise* (equivalently, they can be considered referred to the voxel j, which subscript is here omitted for sake of clarity). c_1, c_2, c_3, α and β are scalar parameters. More precisely:

σ is a vector containing a combination of the final voxel occupancies, henceforth named *voxel confidence* and representing the confidence that each voxel is *non-empty*;

d_1 is a vector containing the distances of the centres of the voxels from the *Realsense depth RF optical centre*;

d_2 is a vector containing the distances of the centres of the voxels from the *Kinect depth RF optical centre*;

α, β are exponents that, combined with d_1 and d_2, are used eliminate the dependency of the point cloud density on the distance from the optical centre;

c_1, c_2, c_3 are constant scaling factors.

In the case of a stereo camera, the density of 3D points decreases as the camera distance increases [114], more precisely, the maximum number of points remains constants inside windows obtained by intersecting planes parallel to the image plane with the camera viewing frustum at distance d from the optical centre. It can be easily seen that the area of these windows is proportional to d^2, so a value of $\alpha = 2$ was chosen. In the case of *Kinect*, a less remarkable variation of the point cloud density was observed in the range interval considered, and a value of $\beta = 0.5$ was set, based on empirical observations. However, a more accurate choice is planned to be done in future experiments.
For better performances, the vectors d_1^{α} and d_2^{β} are pre-computed offline, considering the Cartesian coordinates of each voxel centre of the voxel grid.
The values of c_1, c_2, c_3 can be set according to different strategies, here the following was used:

- c_2 was set equal to 1;
- c_1 was computed so that the mean of the vector ${d_1}^{\alpha} n_1$ was equal to the mean of ${d_2}^{\beta} n_2$. These means were computed considering whole sets of frames, not a specific time step; this resulted in a value of $c_1 \approx 1.1$.
- c_3 was set equal to $c_1 c_2$.

The σ obtained according to Eq. (3.66) was further scaled and capped to a maximum of 255. The choice of 255 allowed to store the final occupancy in a vector containing 8-bit values (*UInt8MultiArray ROS* message type), with the aim of both limiting the computational burden and allow an easier interpretation: this final value of σ represents the voxel confidences as scores between zero and 255. The higher the value, the higher is the probability that the voxel is *non-empty*. Emphasis is placed on

the fact that the voxel confidence account for the sensor heterogeneity, which was quantified by a specific term in Eq. (3.66).
The confidence vector σ, paired with a specific confidence threshold δ_3 (δ_3 = 50) can be used to construct a Boolean voxel map BV, in the following way:

if $\sigma[j] > \delta_3$ $\quad BV[j] = 1$ $\quad$ non-empty voxel

else $\quad BV[j] = 0$ $\quad$ empty voxel

The pipeline of the method is summarized in Figure 3.17.

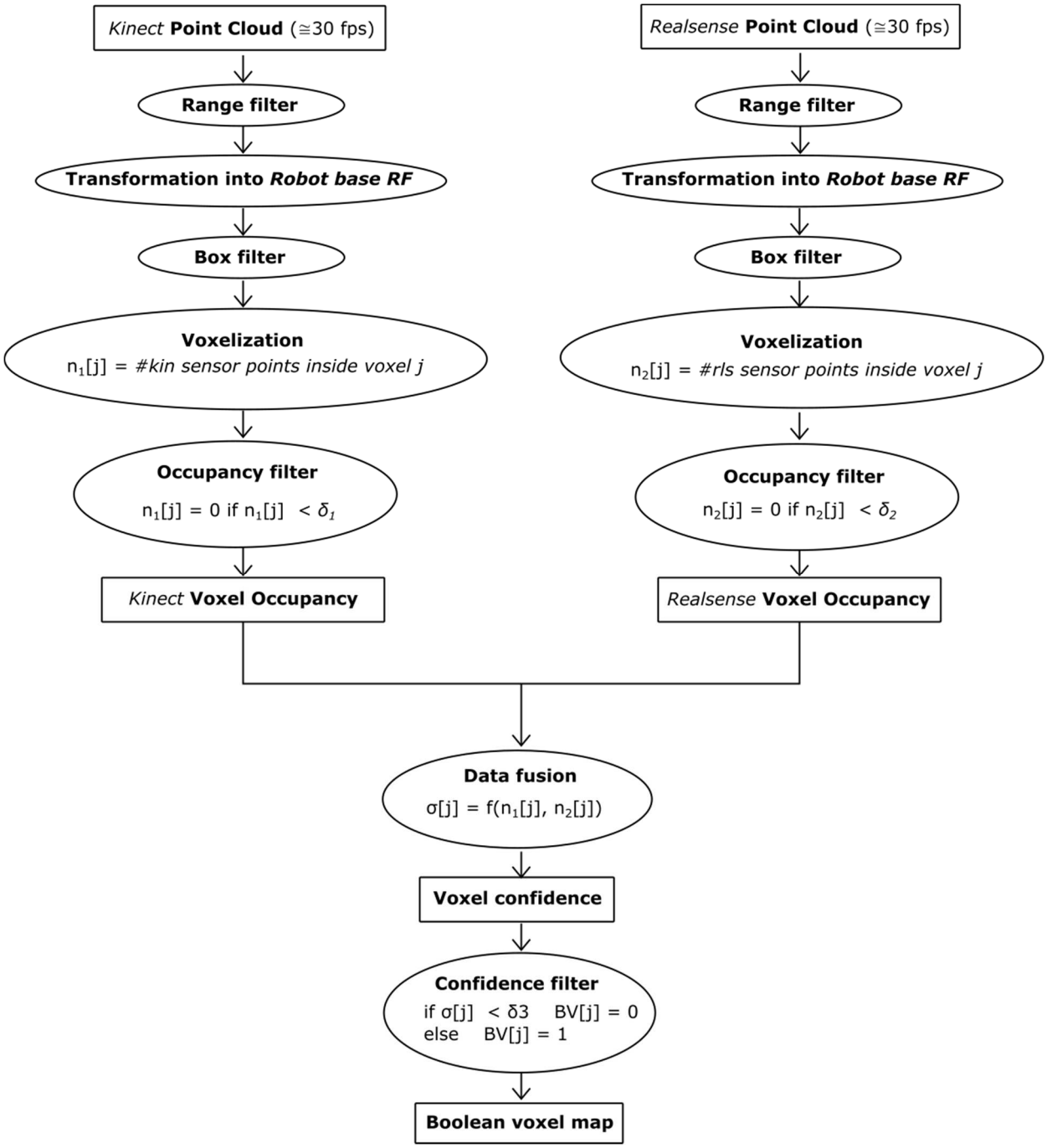


Figure 3.17 Pipeline consisting in all the steps utilized to covert the point clouds acquired by the two depth cameras into a final Boolean voxel map.

This method, even if presented in the case of two depth cameras, can be adapted to a generic number of sensors. Eq. (3.66) can include different additional terms for all the combinations of sensors, properly choosing only the scalar parameters involved.

In the actual implementation, the algorithms presented in this Section were not parallelized, since no performance enhancement was deemed necessary. Note however that, in case of need, most of them can be implemented in a GPU-parallelized way, which is significantly facilitated by the fact that they rely on the use of a voxel grid.

3.2.5 Voxel types

In the previous Section, the classification of voxels in *empty* and *non-empty* was carried out, starting from the depth cameras point clouds. The *non-empty* voxel identified, however, can belong to different entities, which are:

- *static obstacles*: fixed parts of the background, which remains constant at each time step;
- *robot body*: the robot body is part of the scene, and thus acquired by the depth cameras;
- *dynamic obstacles*: obstacles whose positions may change at each time step.

This information was stored in the *vox_type* vector, constructed by assigning a natural number to each cell, identifying the *voxel type*, in the following way (j = 1, …, N_{vox}):

$vox\_type_j = 0$: *type0-voxel*: empty voxel

$vox\_type_j = 1$: *type1-voxel*: part of a static obstacle

$vox\_type_j = 2$: *type2-voxel*: part of the robot body

$vox\_type_j = 3$: *type3-voxel*: part of a dynamic obstacle

The static obstacles can be theoretically considered as a subset of the dynamic obstacles with speed equal to zero, however, since they are known in advance, it is useful to treat them separately. More precisely, the *vox_type* indices corresponding to a static obstacle are computed offline and remains constant. This provides the following benefits:

- The voxels corresponding to static obstacles are *certainly* non-empty, which allows to completely avoid the possibilities of false negatives which can arise in the case of reconstructing them starting from sensor point clouds.
- They are not subjected to occlusion.

- Efficiency is higher, in particular related to the part of the speed estimation: static obstacles are simply not considered in that part.

As part of the static obstacles, a voxel-composed external shell of the control volume was also considered, with the aim of restricting the robot movement inside the control volume, in a way better detailed later.

The computation of the voxel indices corresponding to static obstacles was carried out offline in *MATLAB®*, by exploiting the CAD model of the robotic cell, which is comprehensive of all its components. Part of the code exploits the *polygon2voxel* function [115] from *MATLAB Central File Exchange*, which allows one to convert an .stl file into a voxel map.

The *type2-voxels*, belonging to the robot body, have to be filter out, so that the robot itself is not categorized as obstacle. The way this is achieved is detailed in the next Section.

3.2.6 Robot body filter

The robot body filter is a common filter having the function of filtering out the points belonging to the robot body captured by the sensors, so that are not considered as obstacles in the successive computations. Common implementations of this type of filter make use of the *Unified Robot Description Format* (URDF), which is a format representing the robot model, and, if it is the case, it is sometimes referred to as *URDF filter*. Some examples of available ROS implementation are [116,117]. In the proposed method, a custom implementation of the robot body filter was developed, specifically designed to operate on a voxel grid. This was a performance-motivated choice, for two reasons:

1 - it allows its application only once per cycle, after the data from the different sensors are integrated on the voxel grid. Other available implementations, as a matter of fact, operate on depth images or point clouds, so, in case of multiple sensors, the filter normally needs to run separately on each sensor data, which makes it computationally expensive (and, in general, this type of filter is already computationally expensive, as reported in [116]).

2 – it takes advantage of the regular structure of the voxel grid and of the previous processing in the pipeline, which reduced the data size and complexity.

The developed robot body filter has the function of identifying a series of voxels that belong to the robot body, so that are not considered as part of dynamic obstacles. In order to do so, the robot links and end effector were encapsulated in cuboids, as shown in Figure 3.18.

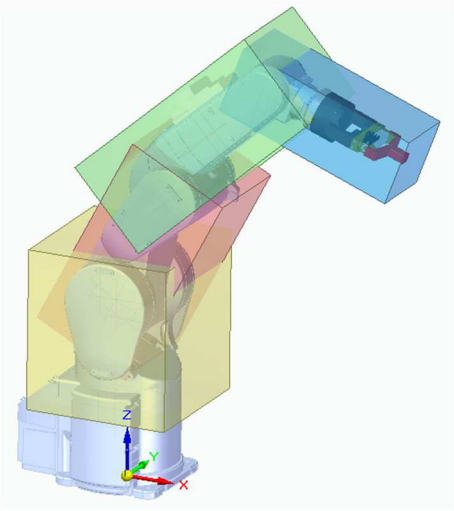


Figure 3.18 Cuboids encapsulating the various robot parts.

The choice of cuboids was motivated by the fact that are particularly suitable for the successive operation described involving the voxel grid. Each cuboid is divided into a set of points, evenly distanced in the three dimensions, thus forming a grid, henceforth named *cuboid grid*.

At each time step, each point of the cuboid grids is transformed as it is attached to the link that the cuboid encapsulates. This is made by exploiting the real-time knowledge of the robot joint position and forward kinematics, available from *the online trajectory generation* block.

For each point of the cuboid grids, the corresponding voxel index is computed by means of Eq. (3.65). If the cuboid grids are sufficiently dense, this results in sets of adjacent voxels, which can be exploited to properly filter out the robot body. To better explain this concept, let us consider the simpler 2D case: the cuboid grid and the voxel grid corresponds to a rectangular grid of evenly distanced points and a matrix of pixel, respectively, shown in Figure 3.19. Figure 3.19a shows the case of a sparse rectangular grid, whereas Figure 3.19b of a dense one. The pixels that contain at least one point of the rectangular grid are highlighted.

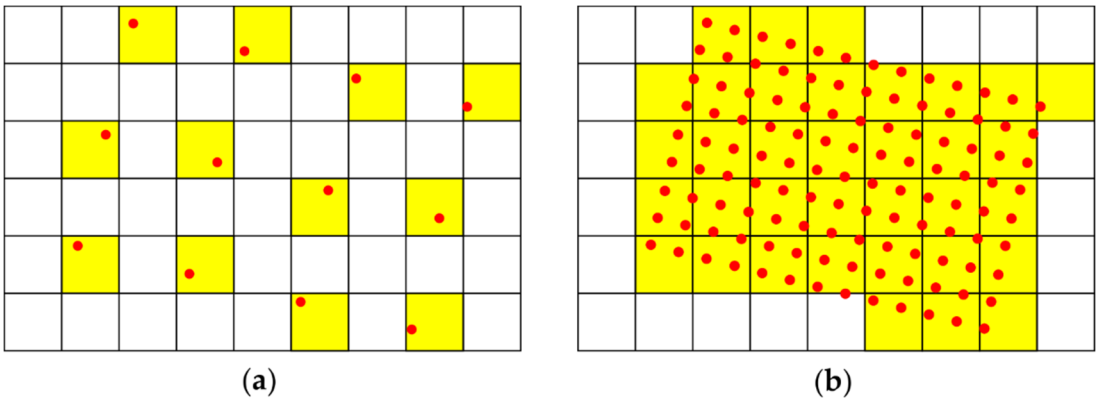


Figure 3.19 Rectangular grid of (**a**) sparse points and (**b**) dense points and a pixel matrix containing them.

On one hand, one wants that the voxels containing the points of each cuboid grid are all adjacent, for all the possible pose of the cuboid grid, which can be achieved by a small step size of the grid Δg. On the other hand, performances benefit from a low number of points, thus a high value of Δg.
To take into accounts both necessities, it is convenient to choose Δg the maximum value that results in all adjacent voxels (i.e. each voxel has to contain at least one point of the grid), for all the possible pose of the cuboid grid.
To evaluate that, the worst-case pose is to be considered. In the 2D version of the problem, the worst case, and the consequent maximum value of Δg chosen, is shown in Figure 3.20.

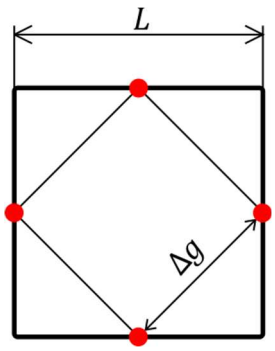


Figure 3.20 Worst-case pose of four point of the rectangular grid of the 2D version of the problem.

The limit case of Figure 3.20 can be used to choose Δg as follows:

$$\Delta g = L/\sqrt{2} \tag{3.67}$$

It is easy to see that this reasoning is appliable to the 3D case as well. In this case, the maximum step of the cuboid grid that results in all adjacent voxels can be computed according to Eq. (3.68).

$$\Delta g = L_{vox}/\sqrt{3} \tag{3.68}$$

Some last remarks are the following:

- the lower L_{vox} is, the better the shape of the cuboid is approximated, so for very high values of L_{vox} this method can produce very poor cuboid shapes, which can lead to false negatives (dynamic obstacles not properly classified since recognized as part of the robot shape). If being the case, alternative methods are to be considered;
- even if it was implemented using the CPU, this method is suitable for a GPU parallelization.

3.2.7 Speed estimation by particle filter

Inside the context of HRC, *Kalman filter* is typically used to estimate the human motion or to enhance its accuracy and reliability [15,59,87,118]. Kalman filter, however, is known for having limitations in dealing with high non-linear models, or when some of the model parameters are not known [43,119]. To tackle the case of generic dynamic obstacles, *particle filter* was chosen instead, since it can commendably deal with non-linearity and does not require the obstacle to be modelized. Particle filter relies on the use of a set of particles, generated with a series of *Monte Carlo* algorithms, that are used to reconstruct the motion of the obstacle. One of its drawbacks is that it normally causes a high computational burden. To overcome this, a GPU-parallelized implementation was developed. The specific details of the GPU implementation are not presented here, but in Appendix D instead. The particle filter is used to associate a speed to each of the *type3-voxels*. The proposed method is inspired by the work of Morales et al. [114] in the automotive field; in particular, as in [114], the implemented filter has the following features:

- it is specifically designed for a voxel grid;
- it accounts for the particle age: to each particle, other than a position and a speed, also an age is associated. The higher the age, the higher the probability that the particle motion resembles the obstacle motion;
- particles are not used as an indicator for the occupancy probability but only to compute voxel speeds.

An overview of the algorithm flow of the implemented particle filter is shown in Figure 3.21. It is composed of the following main stages:

- *Initialization:* (only once, at the first time step): a set of particles is generated by means of a Monte Carlo method;
- *Evolution*: the particle positions are updated according to a specific law of motion;
- *Measurement-based particle selection*: particles whose position is in disagree with the data from the depth cameras are eliminated;
- *Voxel speed computation*: the voxel speed is estimated based on the survived particles inside that voxel;
- *Resampling*: the survived particles are resampled by exploiting Monte Carlo methods.

Each stage will be detailed in the next Sections.

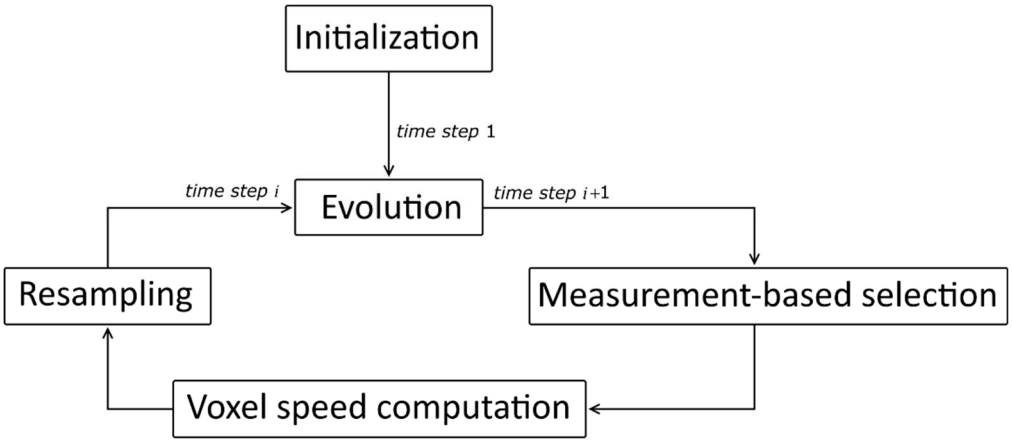


Figure 3.21 Algorithm flow overview.

3.2.7.1 Initialization

At an initial time step, a number of particles Q_{max} (= 96) is generated inside each *type3-voxel*. A particle generated inside the voxel j is denoted with p_{jk}. It is an array, with a position, speed and age as components, as shown in Eq. (3.69).

$$p_{jk} = (x_{jk}, y_{jk}, z_{jk}, u_{jk}, v_{jk}, w_{jk}, \zeta_{jk}) \tag{3.69}$$

where $x, y, z \in \mathbb{R}$ are the position components, $u, v, w \in \mathbb{R}$ are the speed components and $\zeta \in \mathbb{N}$ is its age.

Inside each *type3-voxel*, the particles are generated in the following way:

- Each position is generated inside the voxel with a uniform random distribution;
- Each velocity component is generated with a uniform random distribution in the interval $[-\|\gamma\|_{\max}/\sqrt{3}, \|\gamma\|_{\max}/\sqrt{3}]$, where $\|\gamma\|_{\max}$ is the maximum velocity norm, set to 1 m/s. It is to be noted that there is a trade-off between the choice of the maximum velocity and the precision of the velocity estimation: the higher the maximum velocity, the coarser the velocity estimation, since the same number of particles are generated but considering a wider velocity range.
- The age ζ is set to 0.

3.2.7.2 Evolution

In this stage, the particle state is updated based on a specific law of motion, represented by Eq. (3.70); the particle age is also included, which is incremented by 1. For the sake of clarity, the subscripts j and k are here omitted; the subscript i refers to the time step.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \\ u_{i+1} \\ v_{i+1} \\ w_{i+1} \\ \zeta_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ u_i \\ v_i \\ w_i \\ \zeta_i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \delta \\ \delta \\ \delta \\ 1 \end{bmatrix} \tag{3.70}$$

where δ is a noise generated with uniform random distribution in the interval [-0.05, 0.05] m/s.

3.2.7.3 Measurement-based particle selection and voxel speed estimation

This stage consists in keeping only the particles that, after the evolution stage, end up inside the control volume AND inside a *type3-voxel*.

The survived particles are used to associate a speed $V_j \in \mathbb{R}^3$, to each *type3-voxel*, computed as a weighted mean (where ζ_{jk} is the weight) of the particle speed $\gamma_{jk} = (u_{jk}, v_{jk}, w_{jk})$, of the number Q_j of the particles inside the voxel j, as shown in Eq. (3.71).

$$V_j = \frac{\sum_{k=1}^{Q_j} \zeta_{jk} \gamma_{jk}}{\sum_{k=1}^{Q_j} \zeta_{jk}} \tag{3.71}$$

3.2.7.4 Resampling

In this stage, the following operations are carried out, considering each *type3-voxel*:

- If the voxel contains a number of particles higher than Q_{max}, only the Q_{max} older particles (i.e. with higher values of ζ) are kept.
- If the voxel contains no particles, Q_{max} particles are generated. Each one is generated as follows:
  - its position is generated with a uniform random distribution inside the voxel;
  - each velocity component is generated with a uniform random distribution in the interval $[-\|\gamma\|_{\max}/\sqrt{3}, \|\gamma\|_{\max}/\sqrt{3}]$, where $\|\gamma\|_{\max}$ is the max velocity norm, set to 1 m/s (the same trade-off between maximum velocity and precision holds true here as well as for the case of the initialization stage).
  - The age ζ is set to 0.
- If the voxel contains a number of particles higher than zero but minor than Q_{max}, n_b blocks of particles are generated, each containing a *block_size* number of particles. n_b is set equal to the minimum integer number for which n_b*$block\_size \geq Q_{max}$. Each particle is generated as follows:
  - its position is generated with a uniform random distribution inside the voxel;

- each velocity component is generated with a normal random distribution, with mean equal to V_j and standard deviation *SD*, which was empirically tweaked to a value of 0.15 m/s.
- The age ζ is set to 0.

3.2.8 Obstacle segmentation

The obstacle segmentation refers to the process of identifying all the different dynamic obstacles on the scene. In the case study, this means to form different clusters of *type3-voxels*. The clustering process relies on both the knowledge of the indices of *type3-voxels* (thus their adjacencies) and of their voxel speed V_j, estimated through the particle filter. Adjacent voxels might not belong to the same obstacle (e.g. in the case of two obstacles passing one very close to the another while moving in opposite directions), but an obstacle is always composed of a group of contiguous voxels. To address this, the clustering algorithm exploits the concept of velocity similarity, also used in [114]. Let us consider two tangential velocities $v_1 \in \mathbb{R}^3$ and $v_2 \in \mathbb{R}^3$, their similarity was evaluated by means of a Boolean function s (returning 1 if they are considered similar and 0 otherwise), constructed according to the following pseudocode:

```
if ||v1|| < ε1 AND ||v2|| < ε1      s = 1   // can be noise, considered similar
    else if (v1 · v2)/(||v1||||v2||) < ε2      s = 0   // condition on the cosine
        else if |||v1|| − ||v2||| > ε3      s = 0   // condition on the norm difference
            else   s = 1
        endif
    endif
endif
```

where ε_1, ε_2, ε_3 are threshold parameters empirically set.

If both the velocities have a norm below a threshold ε_1 they are considered as roughly null velocities plus random noise, and thus similar. Otherwise, the similarity on the direction is evaluated (by considering a threshold on the cosine of the angle between them) and on the norm (by considering a threshold on the absolute value of the norm difference).

The clustering algorithm exploits the function s and the voxel adjacencies. It is here outlined in form of a pseudocode. In the pseudocode, the index i is here used to identify the i^{th} obstacle ϑ (not the time step, which is here fixed) and the function

adj(*vox*) finds all the adjacent *type3-voxels* of a *type3-voxel*. One voxel is here considered adjacent to another if they are direct neighbours (for each voxel, the 26 neighbours are considered).

While ∃ a voxel of *Type 3* marked as "non-assigned" do

pick a "non-assigned" *Type 3* voxel *vox_init*

ϑ_i = {*vox_init*}

A = {vox_init}

while *A* != { } do

A_new = { }

for each $vox_j \in A$

find vox_k | {$vox_k \in$ adj(vox_j) AND vox_k not already marked as "assigned"}

for each vox_k

if s($V(vox_k)$, $V(vox_j)$) == 1

$A\_new \leftarrow \{A\_new, vox_k\}$

$\vartheta_i \leftarrow \{\vartheta_i, vox_k\}$

mark vox_k as "assigned"

endif

endfor

endfor

$A \leftarrow A\_new$

endwhile

$i = i + 1$

endwhile

The clustering algorithm outputs a two-level array structure: an array of obstacles in which each obstacle, in turn, is represented by an array of voxel indices. In the presented version, this algorithm is not suitable for a parallelization, since it relies on sequential computing of voxels neighbours. It proved suitable for the case study, however alternative faster versions should be used in case performances require it. It is noteworthy the fact that obstacles with a structure with more than one degree of freedom (such as the human body), might legitimately result in more than one obstacle, since different parts could have significantly different speeds.

First, this segmentation is used as an additional spatial filter: if an obstacle contains a number of voxels lower than a threshold, it is considered noise and eliminated. Then, a velocity is associated to each remaining obstacle, computed as the mean of the velocity of the voxels by which is composed. To each obstacle, also a barycentre is associated, computed as the mean of the Cartesian coordinates of the centres of the voxels by which is composed. Furthermore, each voxel speed V_j is updated and set equal to the velocity of the obstacle to which the voxel belongs, which allows to reduce the noise in the voxel speeds. These latter operations are legit only for motions that, at each time instant, can be approximated to pure translational motion, which is here assumed. Considering a fixed time instant, points belongings to an object subjected to a roto-translational motion, in fact, would have velocities with the same direction but different module, depending on the distance from the rolling point.

3.2.9 Input quantities in the online trajectory generation block

The entity "obstacle" as a whole, is actually not used in the trajectory generation block, but only for visualization purposes; the segmentation process served essentially as an additional spatial filter and as a filter for the voxel velocities. At each time step, for each robot point k, only the voxel with the minimum dynamical distance is considered in the computation of the safety constraints (cf. Section 3.1.3), picked considering both *type1-voxels* (belonging to static obstacles, V_j = 0) and *type3-voxels*, according to Eq. (3.72).

$$j \text{ such that } \left(d_{kj} + \dot{d}_{kj} T_S\right) \text{ is minimum} \tag{3.72}$$

where:

$$d_{kj} = \left\| x_{rb_k} - vox\_centre_j \right\| \tag{3.73}$$

$$\dot{d}_{kj} = (v_{rb_k} - V_j) \cdot \frac{x_{rb_k} - vox\_centre_j}{d_{kj}} \tag{3.74}$$

where *vox_centre* is an array containing at the index *j* the coordinate of the centre of the voxel *j*, the operator · denotes the scalar product and V_j is the voxel speed, updated after the segmentation process (set equal to the speed of the obstacle to which it belongs).

For the sake of efficiency, *vox_centre* was pre-computed offline and used as a lookup table. If needed, the computation of the voxel with the minimum dynamical distance can be easily GPU-parallelized on the index j in order to optimize performances (according to experimental tests it was not required in the case study).

The fact that the computation of the safety constraints relies on the use of single voxels rather than obstacles is an original approach, having its pros and cons, and it is worth some further remarks. This approach allows one to consider an indefinite number of obstacles with generic shapes, without additional computational burden, which only depends on the number of voxels (computations can be easily GPU-parallelized). Other than that, it allows one to define in a straightforward way particular zones, identified by clusters of *type1-voxels*, that can be used to constraint the robot Cartesian movement. This principle was used to create the external shell of *type1-voxels*, representing the control volume boundaries, exploited to confine the robot movement inside of it: the PSD is always to be maintained with respect to the *type1-voxels* composing the external shell, so the robot cannot exit the control volume. A possible improvement can be to specify a different expression for the dynamical distance for the static and dynamic voxels, so that, for instance, all else being equal, the robot is allowed to move closer to *type1-voxels* with respect to *type3-voxels*.
On the other hand, one downside of this approach is that, by considering only a single "most critical" voxel at the time, undesired motion phenomena can appear, deriving from the fact that the motion adjustment is not being constrained considering the totality of the obstacle voxels. By varying the time step, oscillations can possibly appear in the selection of the voxel with minimum dynamical distance, degrading the quality of the generated motion (especially in terms of smoothness). In experimental tests, however, this phenomenon was not prevalent, and it was reduced by lowering the joint acceleration limits. Nevertheless, further research is planned to address this liability.

3.3 Other features and remarks

In this Section some other features involving the whole method are outlined, and some additional remarks are made.

3.3.1 Fault handling

One situation that can occur is that at one time step the optimization problem of Eqs. (3.41-3.43) results infeasible. This can be due to different causes, for example to the fact that the obstacle speed/and or acceleration are so high that one or both of the following can occur:

- the robot, with its limitation in speed and acceleration, cannot produce a modification of its motion able to keep the distance under the threshold imposed by the PSD;
- the depth camera frame rates and robot sample time are not high enough to account for very high obstacle accelerations: the hypothesis of velocity approximately constant between two time steps does not hold anymore.

Other than that, there can be faults due to other issues, for examples false positives in depth camera acquisitions, which might result in an infeasible optimization problem. In addition, in the presented methodology, the *recursive feasibility* [120] is not formally guaranteed, so there is the possibility that the state is driven to a region where the optimization problem has no solution. Even if in the conducted experimental tests this has not appeared to have a significant impact, further tests and developments are planned to address this issue, with the aim to enhance the method reliability.
In the current implementation, if the problem results infeasible, to the robot is commanded a position equal to its current one. In this case, the robot controller stops the robot with the maximum deceleration possible. The robot stay stopped until the optimization problem becomes feasible again.
The handling of low-level errors is still made by the robot controller, which checks that each command actually respects the position and velocity limits (by definition, the command is generated so that it does, so it is an additional check), and stops the robot prompting specific errors in case it does not occur. This can be due for example to the fact that the external computer used acts as a *soft* real time system, which has a higher jitter in loop times compared to a *hard* real-time system, which conversely, guarantees time-determinism. The soft real-time communication is handled by *ROS* and the jitter was observed to be negligible in the case study, but is in general a factor to consider and that could affect the system behaviour. One way to improve the system reliability could be the use of a real-time operating system; this can be done for example by means of the *Linux Xenomai* or the *RT PREEMPT* kernel patch for Linux.

3.3.2 Operating modalities

One additional feature of the method is that the robot can switch between two modalities. If no dynamic obstacles are present inside the control volume, the robot operates at its full dynamics; when a dynamic obstacle is detected inside the control volume, the robot switches to a collaborative modality: the joint speed and acceleration limits are lowered and the motion adjustment enabled. This limiting is not strictly necessary, since the safety constraints by themselves guarantee that robot

motions are generated so to keep the PSD, but it serves both to add a layer of safety and to reduce the human operator mental strain, by raising the perceived safety. If no dynamic obstacles are present anymore inside the workspace, joint speed and acceleration limits are restored to their original values, in order to fully exploit the industrial robot dynamic capability and maximize the productivity.

3.3.3 Enabling the real-time communication with the robot controller

A basic conceptual scheme outlining how the (soft) real-time communication between the various elements is shown in Figure 3.22.

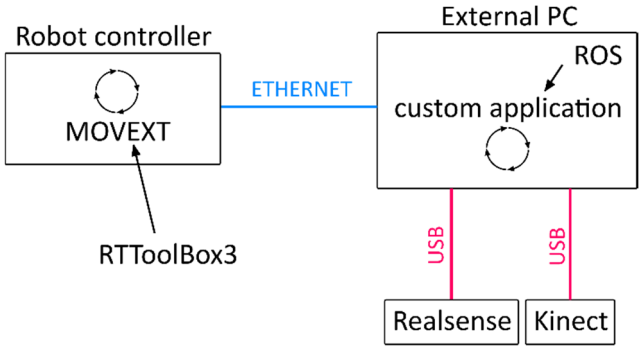


Figure 3.22 Conceptual scheme of the various elements involved in the real-time communication.

The custom application, developed by means of *ROS* (see Appendix B for more details about the implemented *ROS* architecture), runs inside the external workstation, processing data acquired from the external sensors and generating a command, sent to the robot controller through Ethernet. For the communication of the robot command to be effective, a specific program must run inside the robot controller. This program, named *MOVEXT* in Figure 3.22, can be written and loaded inside the controller by using the *RTToolbox3 Mitsubishi Electric* proprietary software. Its structure is very simple, and it is reported in Figure 3.23. It basically moves the robot into a starting position and enables an external communication by activating a specific UDP port of the controller. Also, a time filter is specified as a parameter in the function which takes care of receiving external command data. The filter was set to 30 ms and allowed to produce smother trajectories, an aspect better highlighted in the next Section.

```
1  GetM 1
2  Servo On
3  Wait M_Svo = 1
4  Mov JSTART
5  Open "ENET:192.168.1.50" As #1 'Open the communication with the TAILOR workstation
6  Mxt 1,1,30 'Move with real-time external control, joint coordinates, filter time constant set to 30 msec
7  Close #1 'Closing the communication
8  Hlt
9  End
```

Figure 3.23 *RTToolBox3* program, running on the controller and enabling the real-time external communication.

3.4 Results

To enable a real-time visualization of the various elements involved in the collision avoidance method (and providing an HRC interface), the rear part of the cell was endowed with a display, connected to a *Raspberry Pi4*, in turn remotely connected with the workstation. During the tests, the display was just used as a monitor, showing a custom-made visualization window; a remote control of the workstation was also possible in case. The visualization window was created by means of the *ROS* visualization tool *RViz*, refreshing in real-time all of its elements with a rate set equal to 30 Hz. All the elements of the visualization windows are highlighted in Figure 3.24. The robot model in the visualization window replicates in real-time the movement of the real robot. Figure 3.24c and Figure 3.24d are close-ups of parts of Figure 3.24a and Figure 3.24b, respectively, showing how velocities are visualized: a purple arrow that points in the motion direction and whose length is proportional to its norm. The four robot points considered are identified by different values of k, as shown in Figure 3.24a; this will be later used to reference them in various plots.

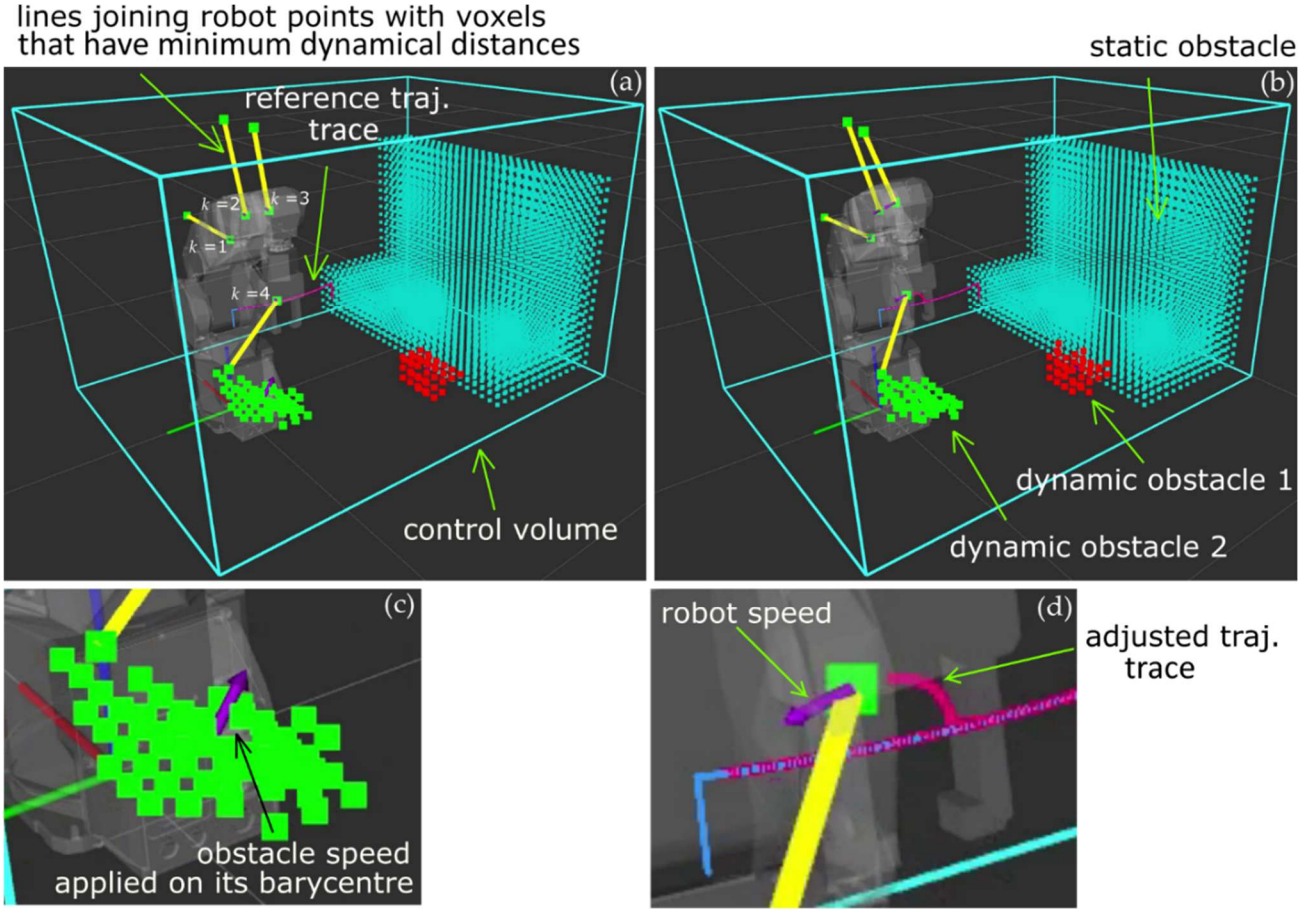


Figure 3.24 Visualization window and all its elements.

Figure 3.25 shows the effect of the robot body filter. The *type2-voxels*, belonging to the robot body (Figure 3.25a) are filtered out by means of the violet voxels (Figure 3.25b) resulting from the use of the cuboid grid as outlined in Section 3.2.6.

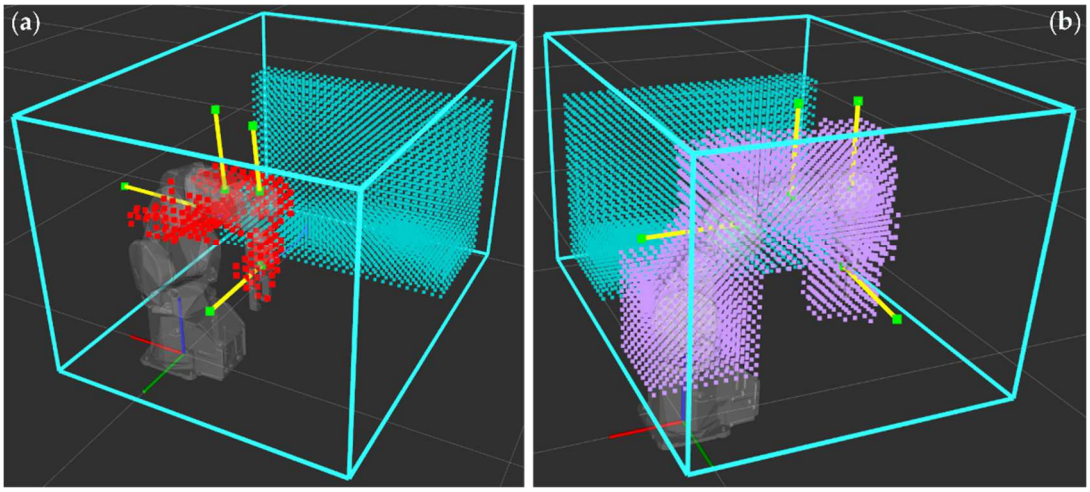


Figure 3.25 Robot body filter.

Tests reported here were carried out considering a task-based trajectory having the shape shown in Figure 3.26, consisting of six frames extracted from a recording of the visualization window, numbered according to their temporal order. The task based-reference trajectory was constructed by means of four waypoints (robot traverse them in the images of Figure 3.26 identified by *1*, *2*, *5*, *6*); the robot moves from one to another by means of joint interpolated motions.

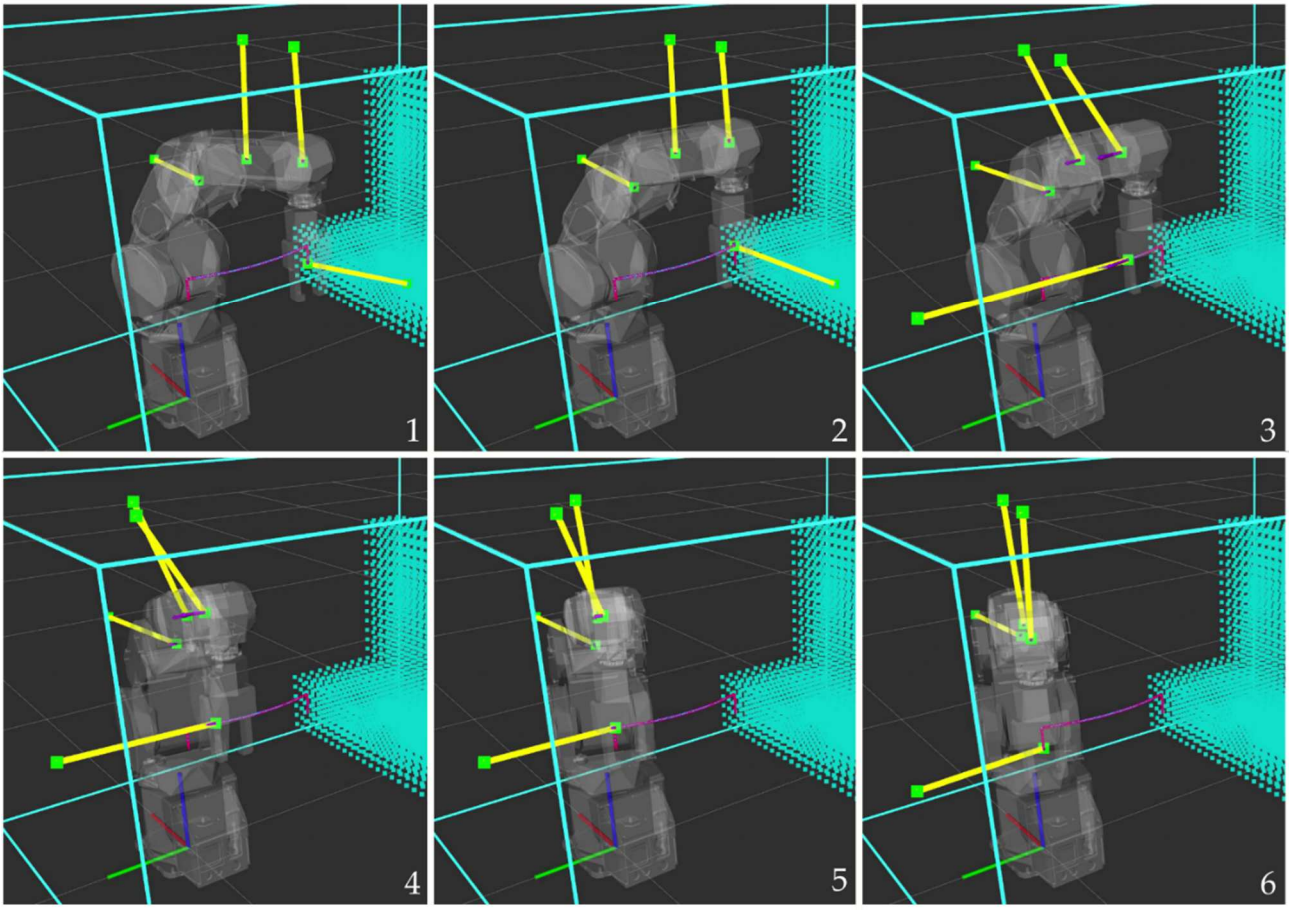


Figure 3.26 Task-based pre-programmed reference trajectory.

In absence of dynamic obstacles, the robot follows the task-based reference trajectory. When a dynamic obstacle enters the workspace, the robot performs a trajectory adjustment, whose trace can be monitored online, along with other quantities, as shown in Figure 3.27, which report four frames (with increasing timestamps from left to right) extracted from a recording of the visualization windows. The dynamical obstacle is the cluster of red voxels; the yellow lines join each robot point with the voxel having minimum dynamical distance from it, among both *type1-voxels* (static obstacles, in cyan) and *type3-voxels* (dynamic obstacles, in red). The control volume is composed of an external shell of *type1-voxels*, not visualized as the other *type1-voxels* and *type3-voxels* since it would clutter the visualization and obscure the elements inside the control volume. It can be seen however, that some of the yellow lines are connected to that external shell, since the voxels with minimum dynamical distance are there.

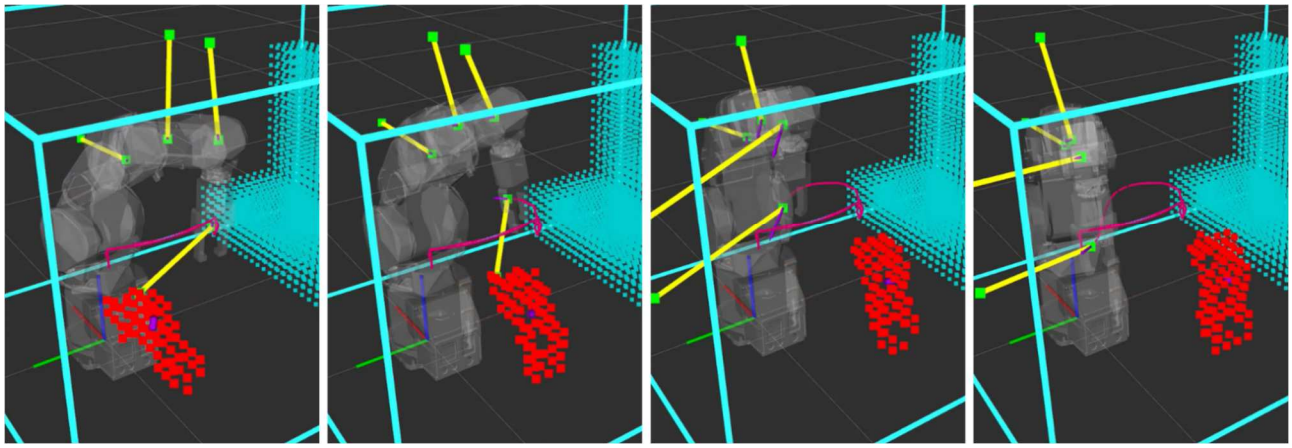


Figure 3.27 Example of trajectory adjustment, allowing the robot to keep the PSD in case of an approaching dynamic obstacle (in red). Time stamps increases from left to right.

The next part (Figures 3.28-3.35) shows in detail one example of trajectory adjustment in the case of heading point computation based on motion laws (cf. Section 3.1.6.1). All the video frames and plots refer to synchronize data, so they can be compared considering the reported timestamps.

For safety reasons and to minimize the risk of damages to the equipment, tests on the real robot were performed using a wooden shaft with rubber at one end. This is shown in Figure 3.28, consisting in some frames extracted from a video where the robot performs an evasive motion to maintain the PSD. Figure 3.29 shows the corresponding frames extracted from a recording of the visualization window.

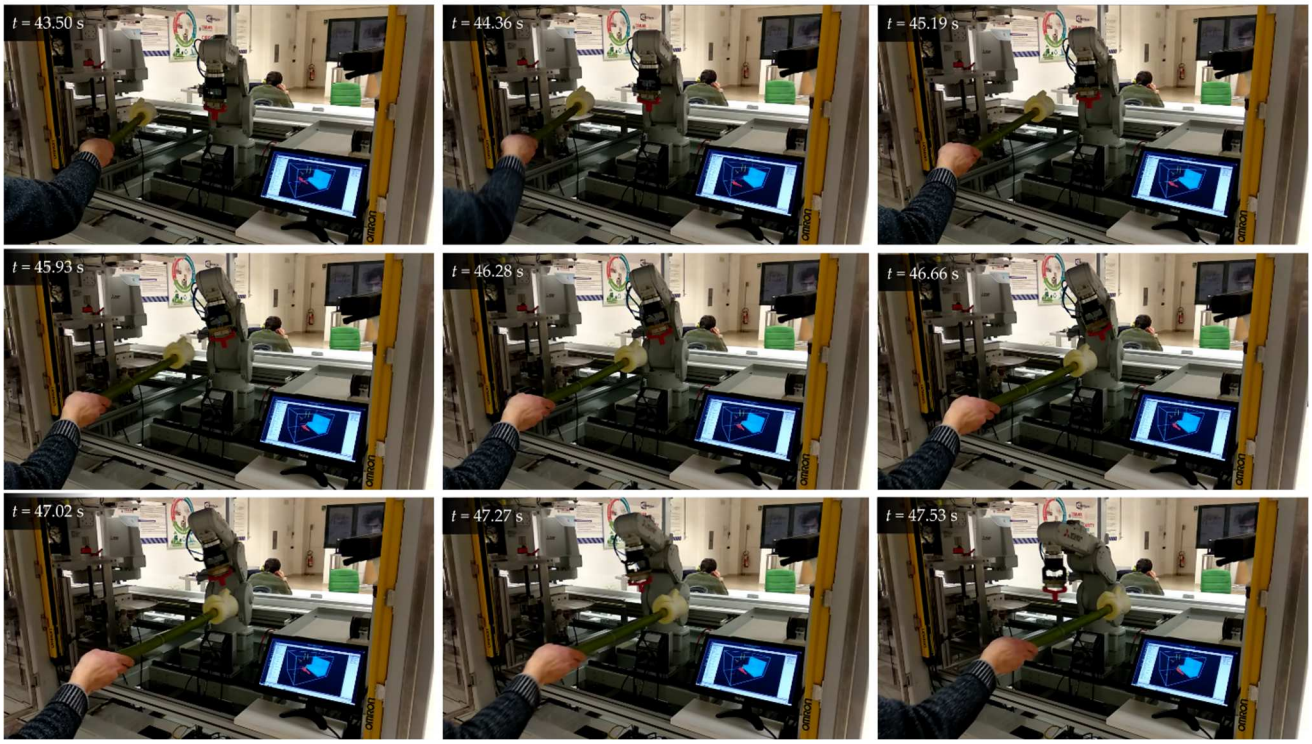


Figure 3.28 Robot performing an evasive motion to maintain the PSD while the moving shaft is approaching.

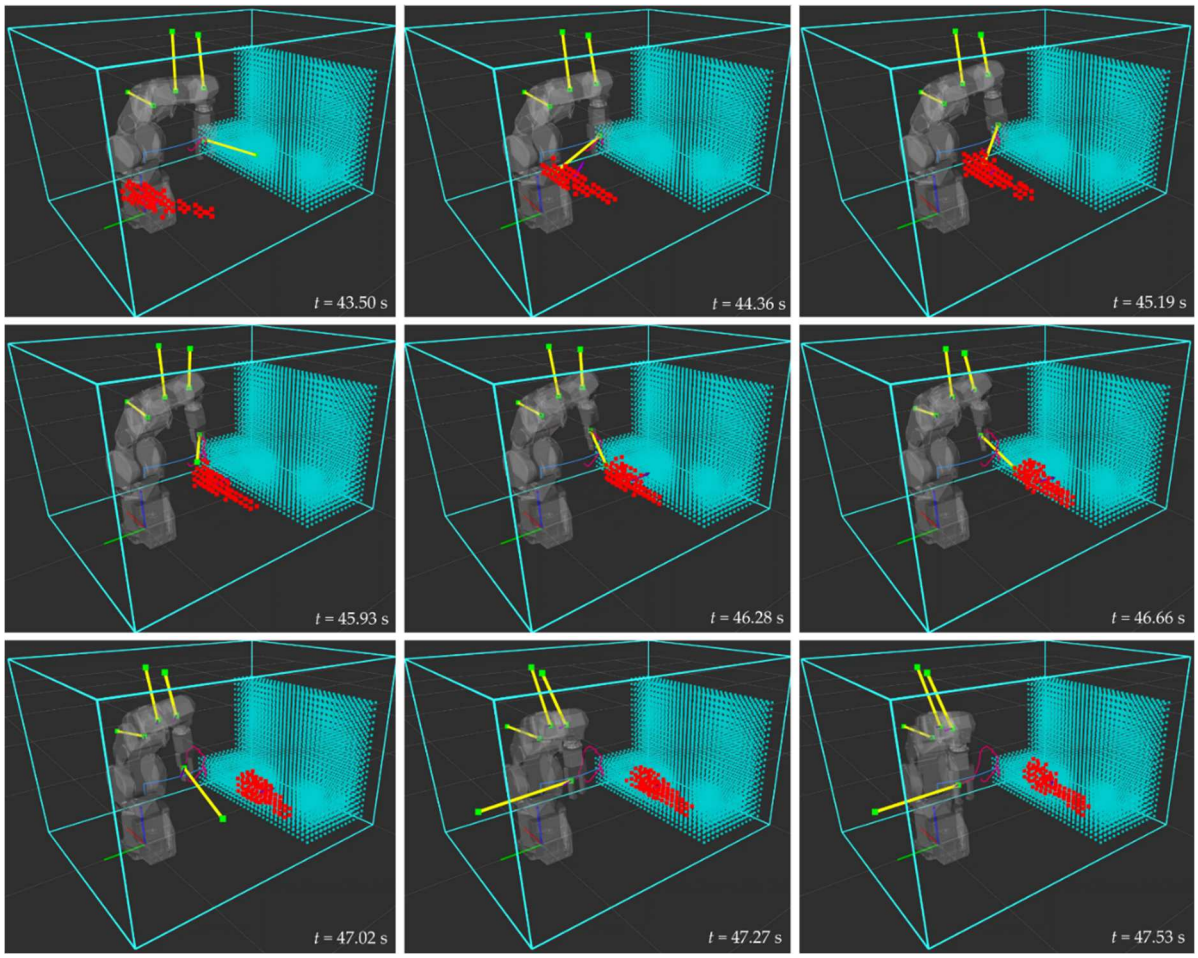


Figure 3.29 Frames extracted from the visualization windows corresponding to the frames of Figure 3.28.

Figure 3.30a shows the Cartesian trajectory generated by the commanded joint position, whereas Figure 3.30b shows the actual robot Cartesian trajectory, reconstructed by means of the joint position feedback values available from joint encoders. Henceforth, the term *command* and *feedback* will refer to this distinction.

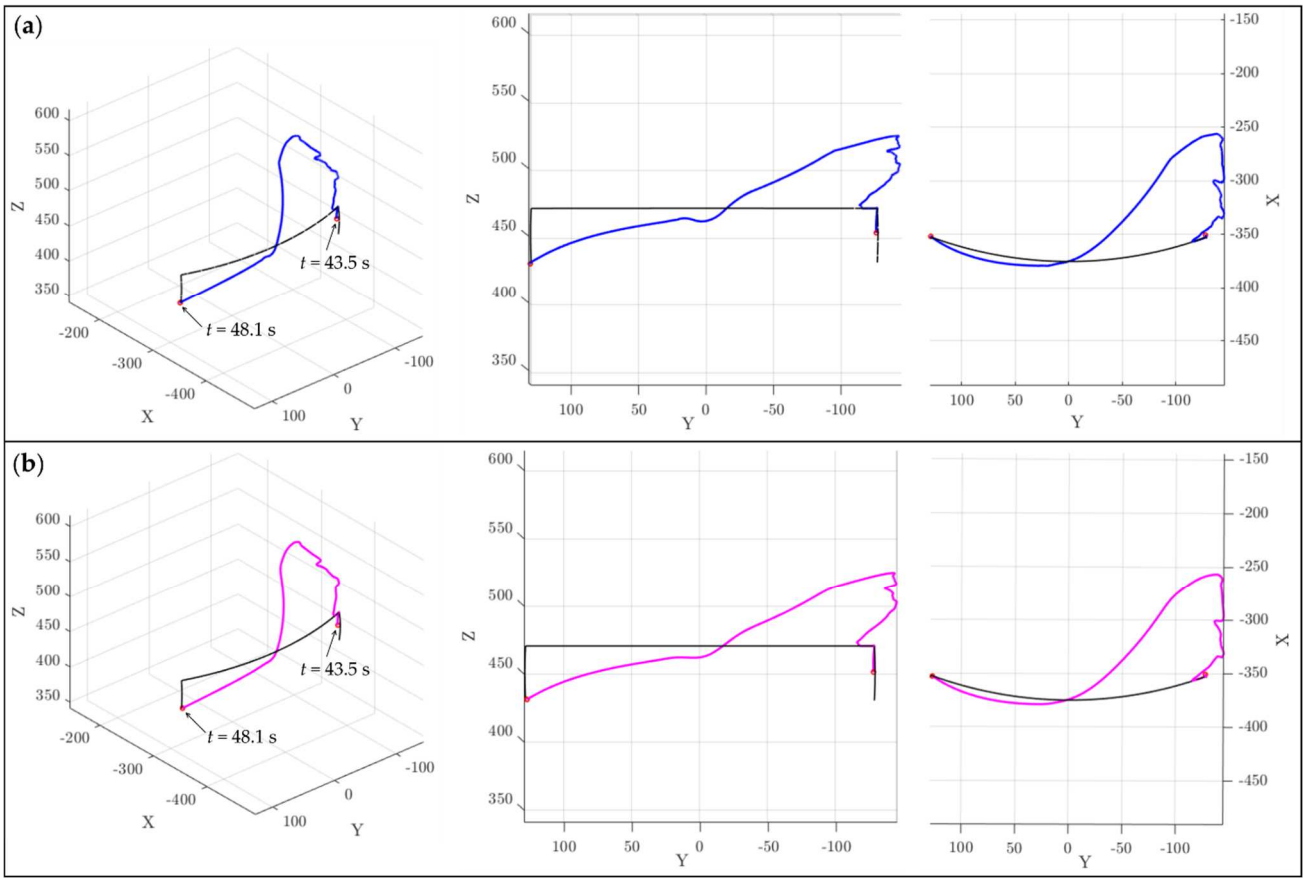


Figure 3.30 (**a**) Cartesian trajectory resulting from the commanded joint position; (**b**) feedback Cartesian trajectory. The axes unit of measurement is mm.

Figure 3.31 and Figure 3.32 shows the joint positions (in degree) and velocities (in degree/s), respectively. The reference, command and feedback joint values are reported.

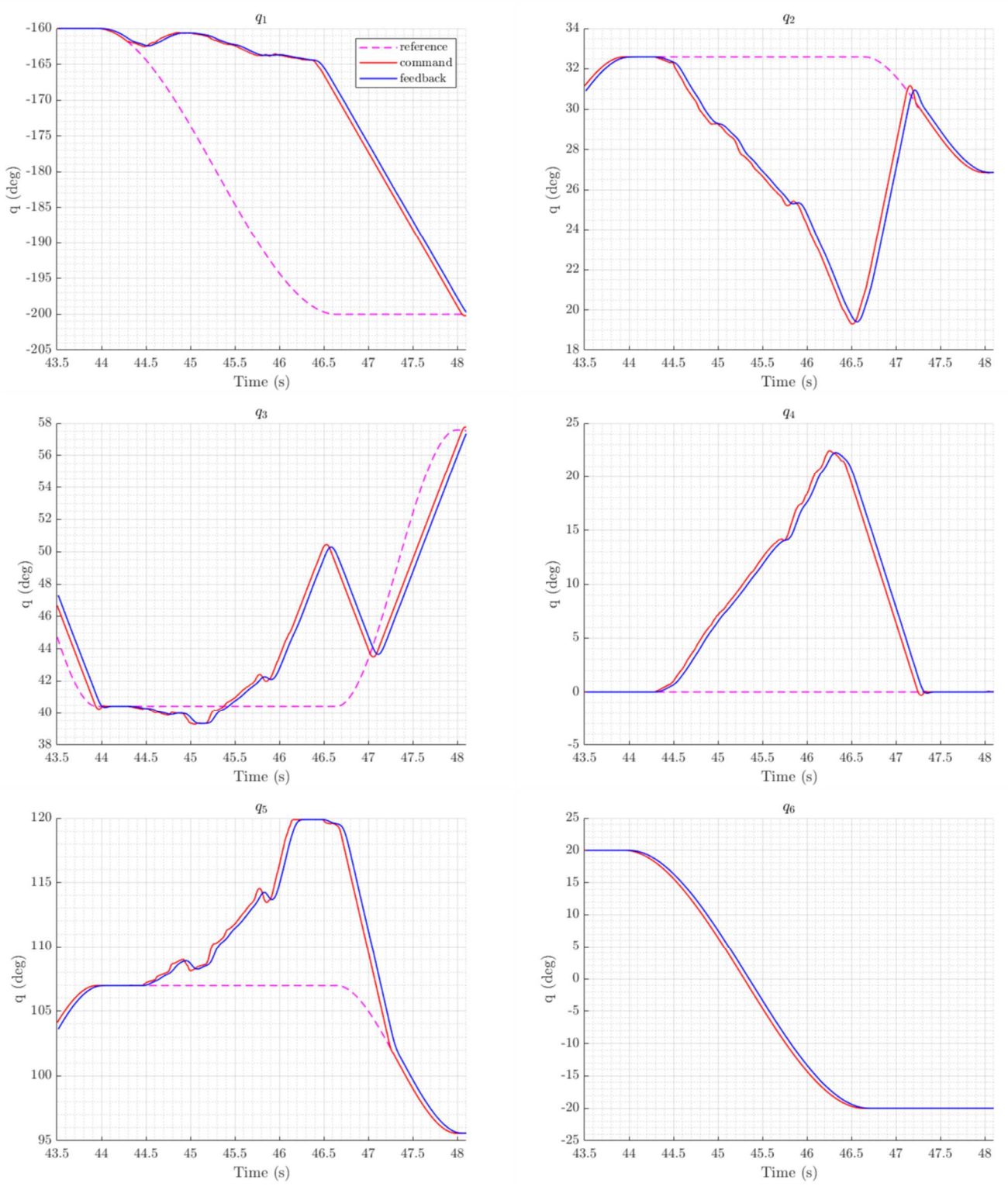


Figure 3.31 Joint reference (magenta dotted line), command (red solid line) and feedback (blue solid line) positions. Values are reported in degree.

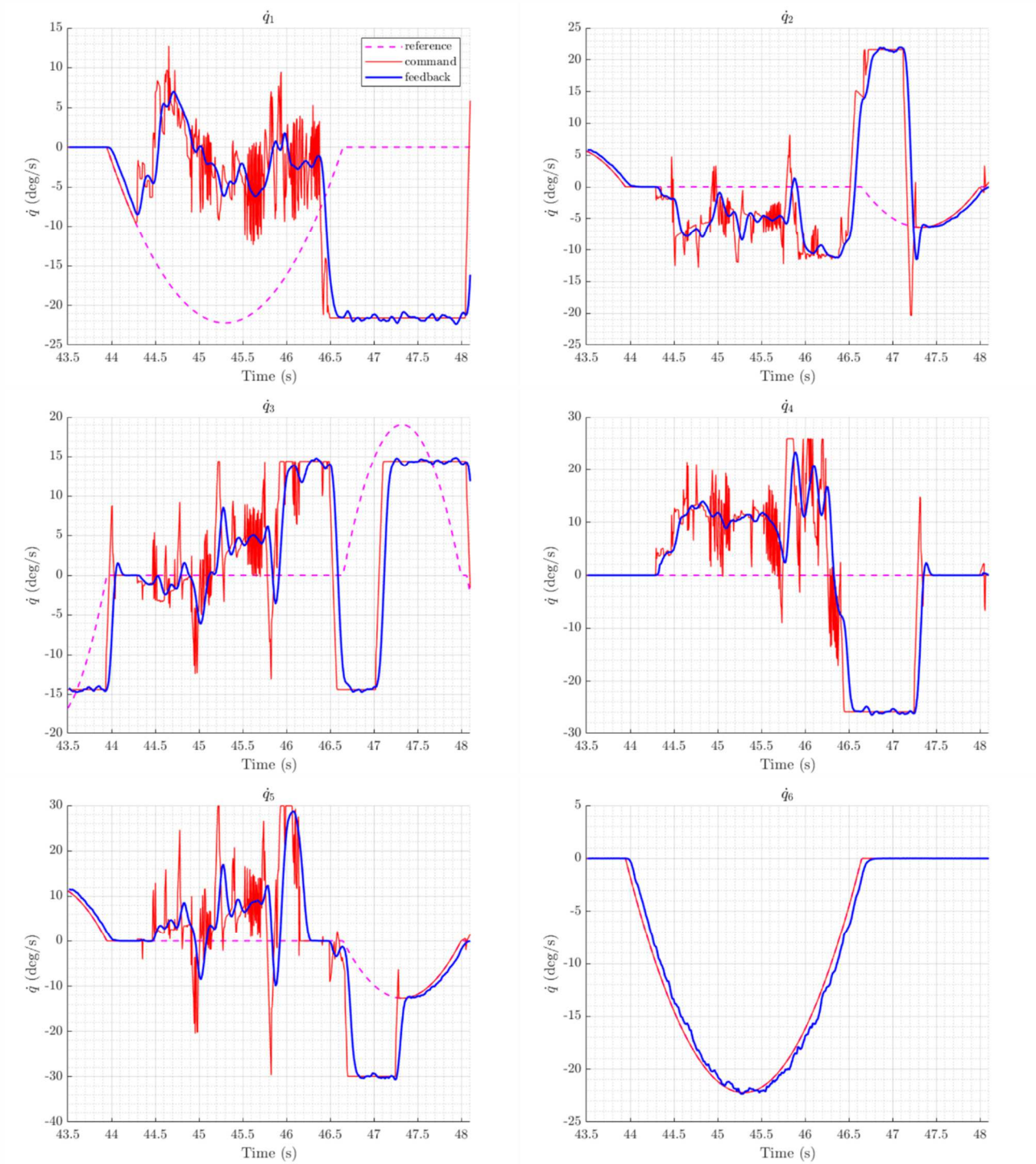


Figure 3.32 Joint reference (magenta dotted line), command (red solid line) and feedback (blue solid line) velocities.

Figure 3.33 and Figure 3.34 shows a comparison, for each of the four robot points (identified by different k, according to Figure 3.24a) between the distance and the PSD, computed considering at each time step the voxel with the minimum dynamical distance (cf. Section 3.2.9). Figure 3.33 reports the values computed considering the

commanded robot joint positions, whereas Figure 3.34 reports the values computed considering the robot joint feedback positions.

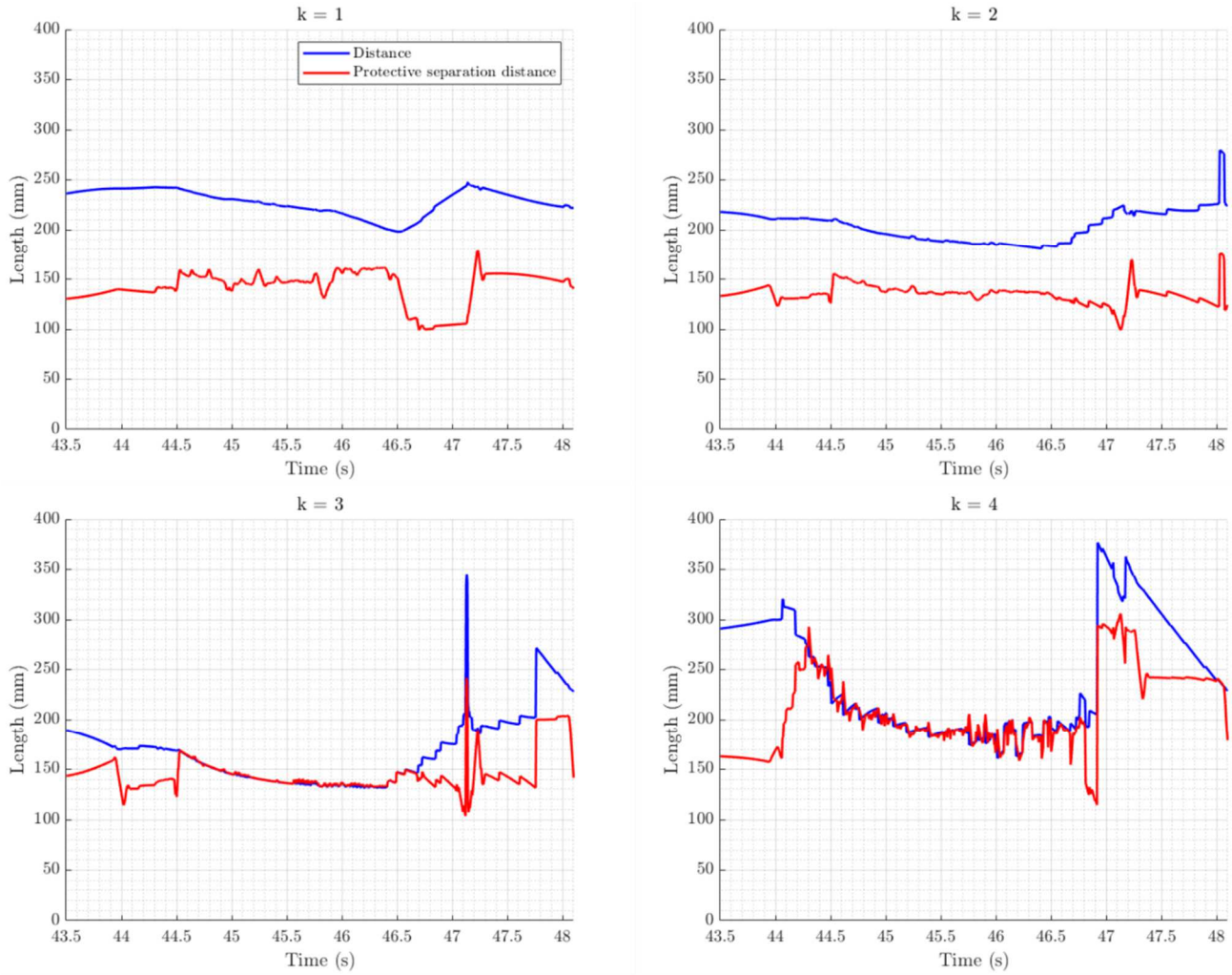


Figure 3.33 Distance (blue) and PSD (red), for each of the four robot points considered, identified by different values of k. Values are obtained considering the commanded robot joint positions.

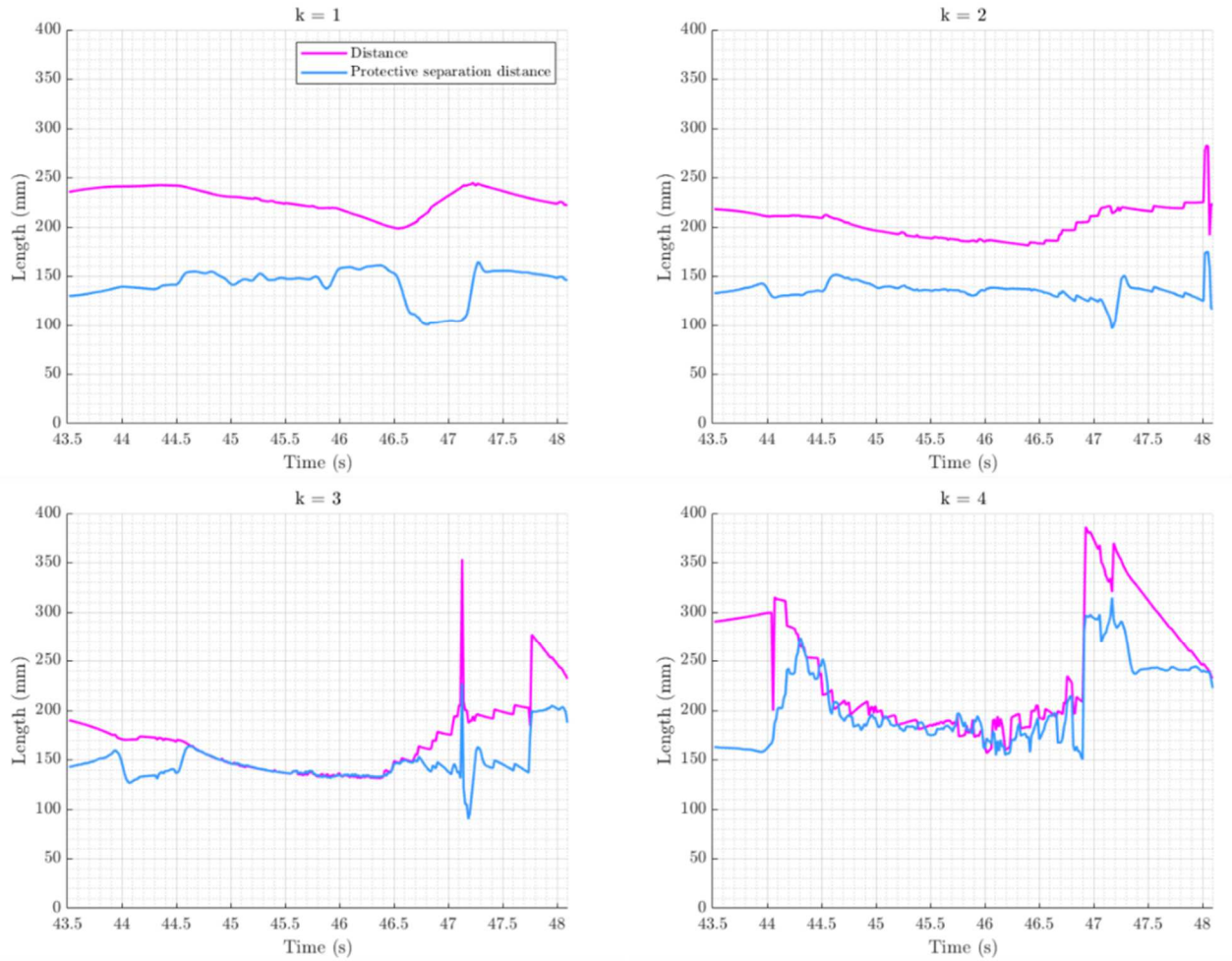


Figure 3.34 Distance (magenta) and PSD (light blue), for the four robot points, identified by different values of k. Values are obtained considering the feedback robot joint positions.

Figure 3.35 shows the x-y-z velocity components of the considered voxels (in the case of k = 4), estimated through the particle filter.

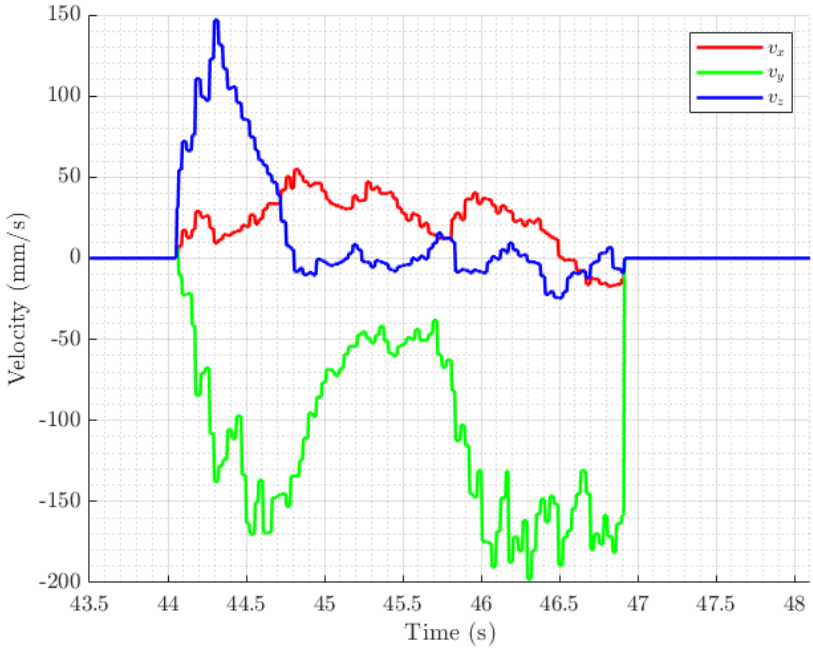


Figure 3.35 x-y-z velocity components of the considered voxels (in the case of k = 4), estimated by means of the particle filter.

The next part (Figures 3.36-3.38) shows some relevant plots in the case of heading point computation based on the geometrical approach (cf. Section 3.1.6.2). In this case, in Figure 3.37, the dotted lines do not represent the reference motion laws but are the set of heading points q_{ref} computed online according to the procedure explained in Section 3.1.6.2.

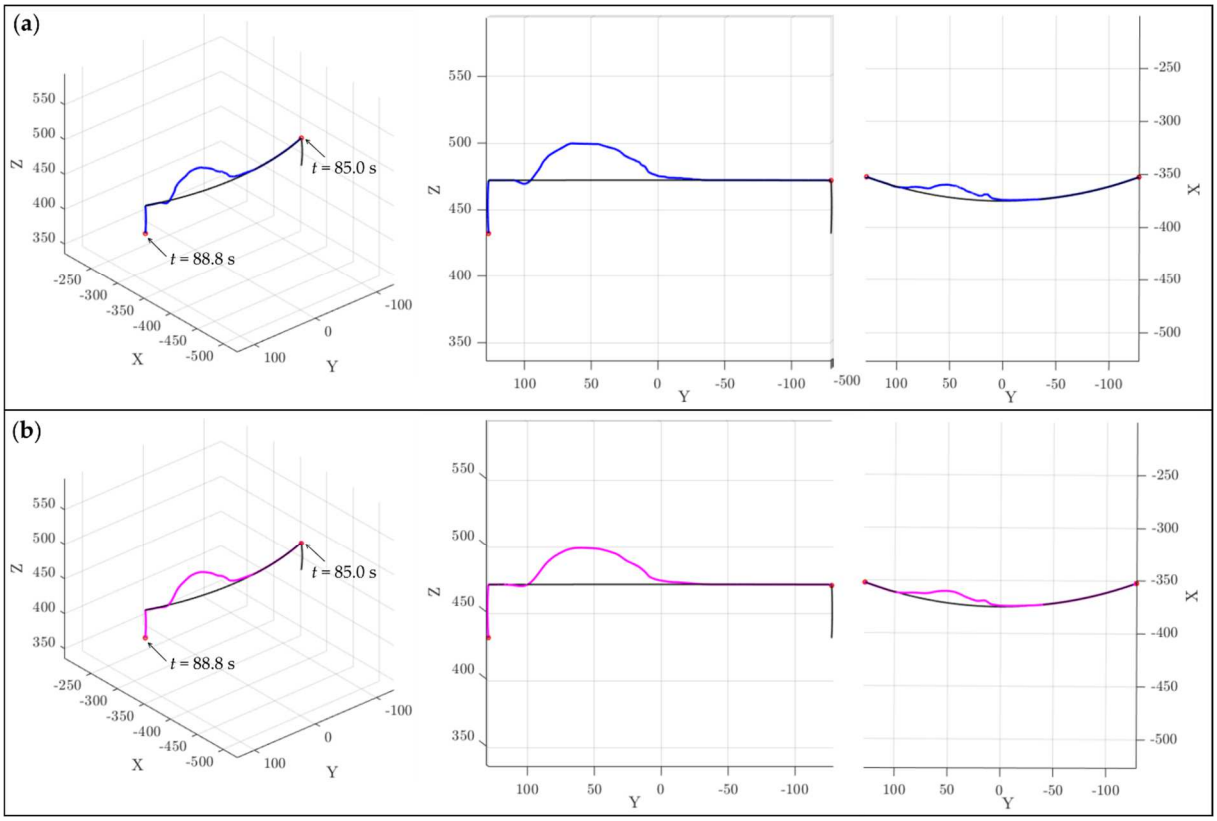


Figure 3.36 (**a**) Cartesian trajectory resulting from the commanded joint position; (**b**) feedback Cartesian trajectory. The axes unit of measurement is mm.

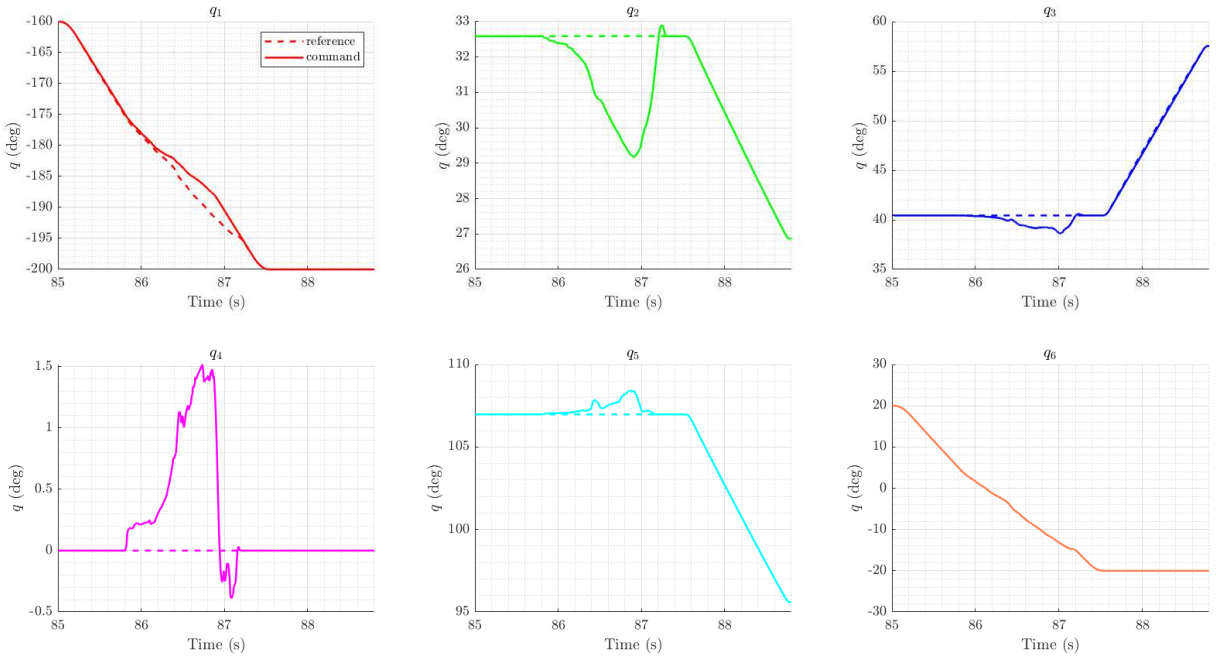


Figure 3.37 Reference (dotted lines) and commanded (solid lines) joint positions.

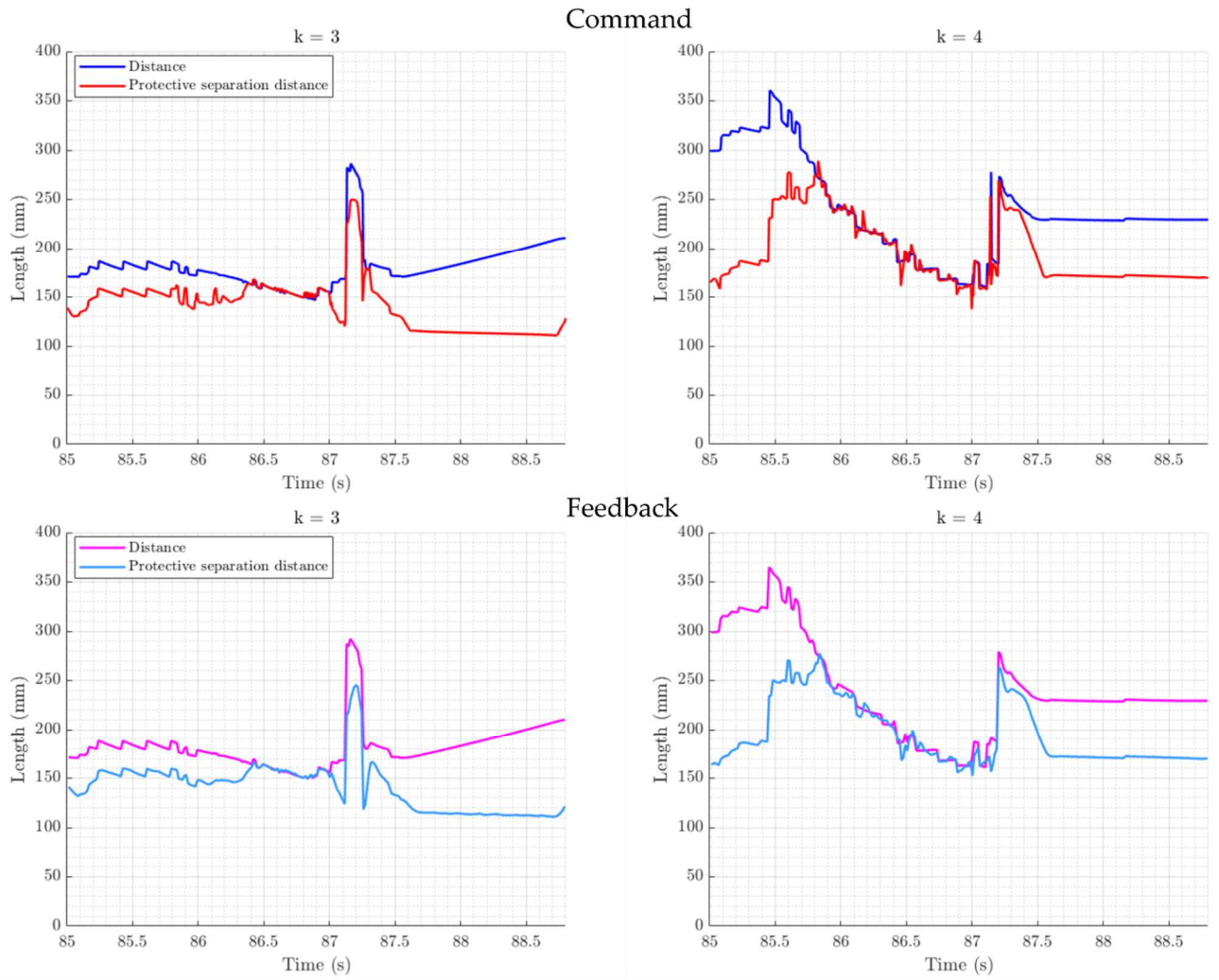


Figure 3.38 Distance and PSD, for two of the four robot points (k=3, k=4). Data are obtained considering the joint commanded position (two plots on the upper part) and the joint feedback positions (two plots on the lower part).

Figures 3.39 and Figures 3.40 highlight the obstacle speed estimation by means of particle filter, in a case in which the speed changes direction. Both figures report frames numbered according to their temporal order, and have roughly corresponding timestamps.

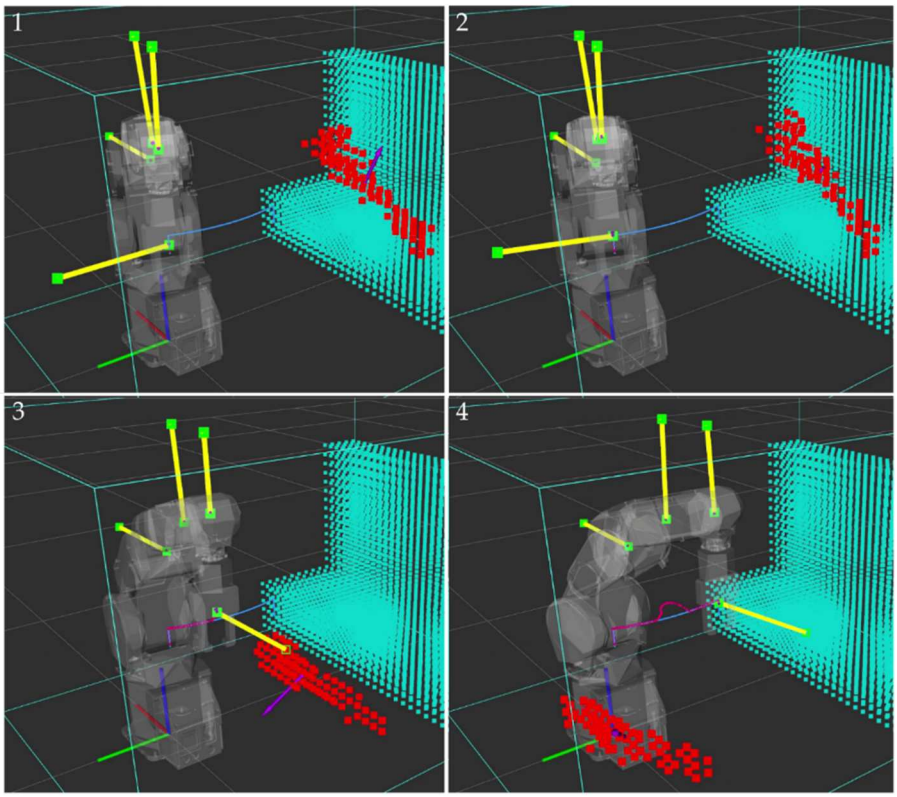


Figure 3.39 Online visualization of the obstacle speed: purple arrow, applied in the obstacle barycentre, having the obstacle speed direction and length proportional to the obstacle speed norm.

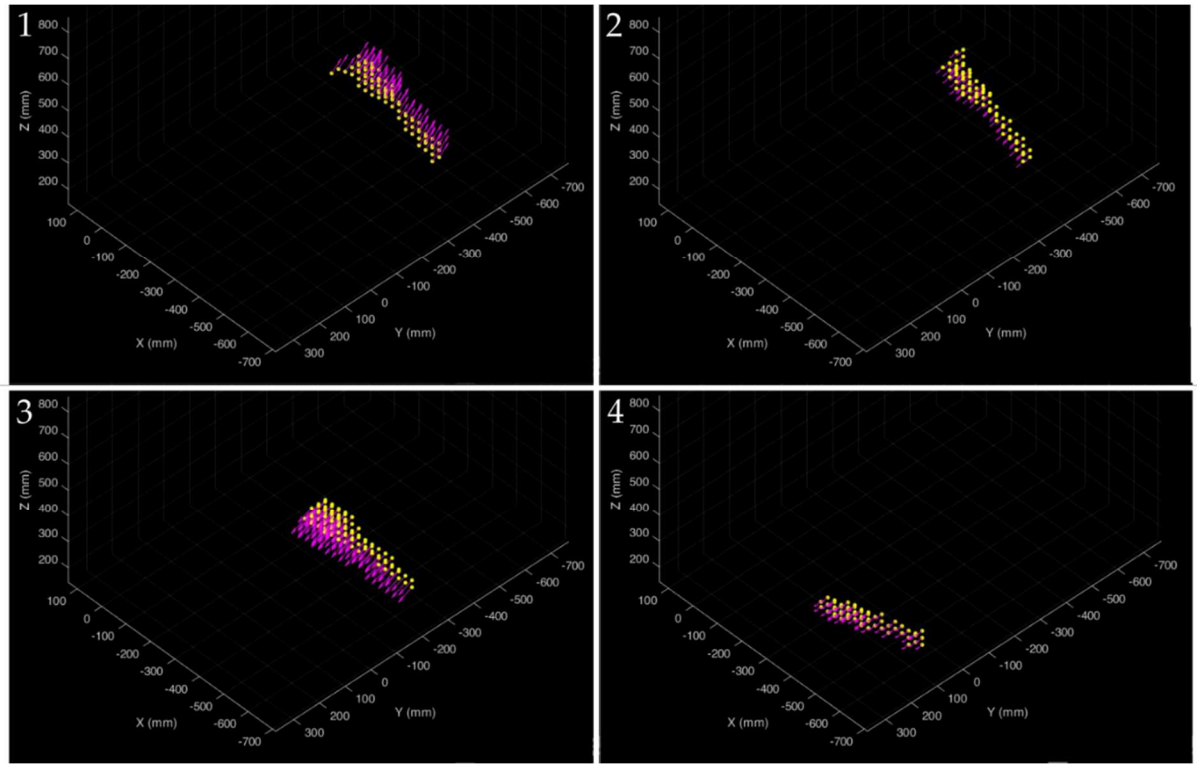


Figure 3.40 Alternative (offline) visualization, where an arrow (in purple) is associated to each voxel (in yellow). The arrows point into the motion direction and their length is proportional to obstacle speed norm.

Figure 3.41 shows the switching between the industrial modality and the collaborative modality, this latter with lower maximum joint speed and acceleration limits, when a dynamic obstacle enters the control volume.

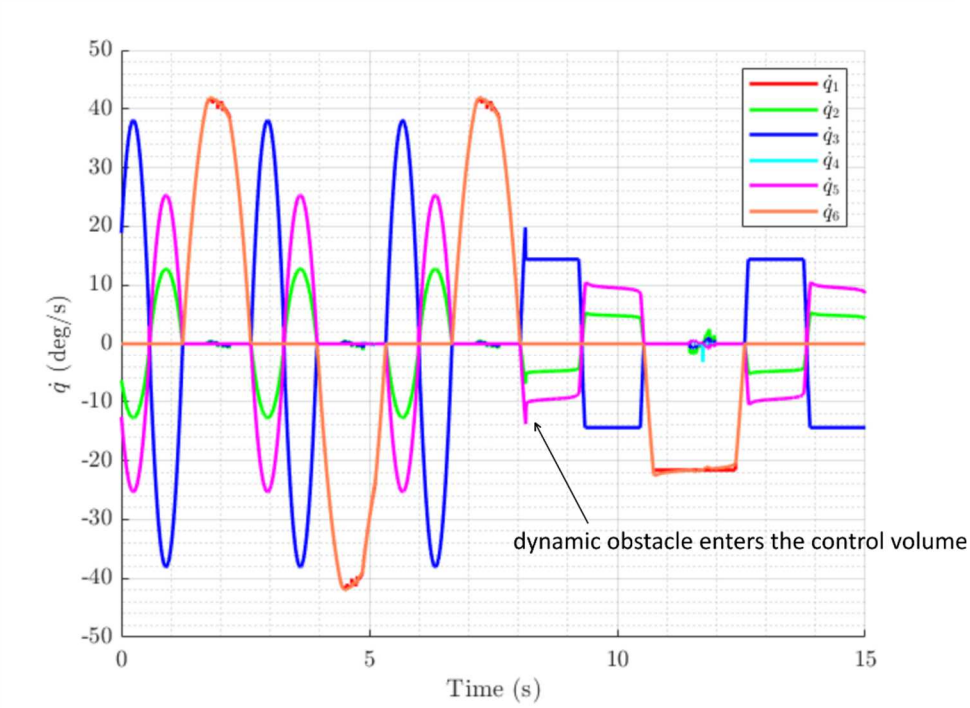


Figure 3.41 Lowering of the maximum joint speed when a dynamic obstacle enters the control volume.

3.5 Discussion

The adherence to the safety constraints has the aim of generating a robot command in such a way that the robot distance from obstacles does not drop below the corresponding PSD. If considering the *command* case, the experimental results of Figures 3.33 and Figure 3.38 show that this approximately happens, if neglecting the occurrence of short and low-amplitude spikes that cause very short temporary violations of the PSD threshold. This can be due to various factors, such as:

- errors in the prediction of the obstacle position, due to approximations concerning their velocity, considered constant inside the time step duration;
- approximations caused by the discretization of the environment by means of voxels;
- approximations assumed in the computations of the safety constraints;
- nature of the strategy for the voxel selection detailed in Section 3.2.9, which considers only the “most critical” voxel at each time step.
- soft real-time system;

- small imperfections and variability in the obstacle reconstructions due to inevitable sensor limitations.

In the *feedback* case, the distance and dynamical distances are less spiky, and in the case study the controller proved sufficiently performant to guarantee that an acceptable difference from the command is kept throughout the motions. In the various conducted tests, the values assumed by the distances with respect to the PSD were considered tolerable due to the negligibility of the drops under the PSD threshold. Nonetheless, further research is planned to better understand and limit the observed phenomenon in the command case.

It is very evident (e.g. in Figure 3.32) that the actual (feedback) trajectory is smoother than the commanded one, which is a positive aspect; this is due to different factors:

- the robot controller performs a smoothing action before actually commanding the robot, by exploiting a temporal filter whose sample time is specified by the user (a filter of 30 ms was applied in the case study);
- the closed loop control system implemented in the robot controller and the robot dynamics affects the smoothness.

The smoothing action of the robot controller is a valid feature, which allows it to handle position commands in an optimal way. Future implementations of the method aim at directly generating smoother commands without relying on this feature of the robot controller, to make the method more fitting for a general use.

The trajectory resulting from the heading point computation based on the *laws of motion approach* (cf. Section 3.1.6.1) could present some anomalies, that can be seen for instance in the last part of the Cartesian trajectory of Figure 3.30: it diverges from the reference one, but not because of the need to fulfil the safety constraints: the obstacle is left behind and assumes nearly zero speed, thus not affecting the robot trajectory anymore. This behaviour can be explained by thinking the heading point as constantly moving forward following the reference laws of motion, but not being closely followed by the robot, which in the meantime must perform a different motion to adhere to the various constraints. As an extremum exemplificative case, let us consider the following: the robot is deviated for a long time from the reference trajectory, and when the external disturbance ends the heading point has already reached the spline end, assuming a constant value; at this point the robot does not follow anymore the reference trajectory to reach it, even if it is on it, but finds another path, generated piece by piece by the optimization problem, which considers a constant q_{ref_i} equal to the last trajectory point $S(T_m)$. This behaviour does not occur in the heading point computation based on the *geometrical approach*, since the heading point progresses on the reference trajectory based on the current robot joint

positions. At each time step, the robot tries to approach the geometrical shape of the trajectory (in the joint space), rather than to follow the laws of motions. This latter approach guaranteed a better adherence of the motion to the geometrical shape of the reference trajectory. In doing so, a trend commonly observed was the performing of speed adjustments rather than escape motions. Another way to put it is that, in this latter approach, the algorithm is aware about the deviation from the original trajectory, differently from the case of the *laws of motion approach*, which thus produces huge variations with respect to the reference trajectory. A way to remove/mitigate this problem in the *laws of motion approach* could be a replanning, which, however, is a routine typically computationally expensive not suitable for an execution within a cycle time, and that would thus run occasionally according to a specific rule. The *geometrical approach*, on the other hand, has the advantage of being intrinsically capable of overcoming this issue. In general, however, the possible advantages of a replanning will be investigated in future developments.

One further and related consideration concerns the fact that, since the optimization problem is solved in the joint space, the corresponding generated shape of the escape motions in the Cartesian space may sometimes appear not very intuitive to a human operator and not predictable by them, which can be the cause of mental strains. This is accentuated in the case of the *laws of motion approach*, which, as has been pointed out, produces motion that can significantly diverge from the geometry of the reference trajectory, even after the influence of the obstacle ends.

4 Programming by demonstration

This Chapter is devoted to the developments of methods, enabled by artificial vision, aimed at facilitating the robot programming, which is normally a mansion relegated to highly trained operators, since it requires specific programming knowledge and may be difficult and time-consuming. As already mentioned, one important enabler of HRC consists in all the tools, methods and interfaces that can facilitate the interaction with robots, making easier, more natural and intuitive the various common tasks and routines performed by humans in which robots are involved. In the case of industrial robots, the hand guiding modality typical of collaborative robots is not available out-of-the-box, so it is useful to investigate and propose other suitable methods easily appliable to traditional industrial robot as well. In this Chapter, two methods aimed at enabling the intuitive PbD paradigm by means of artificial vision are presented. Here, PbD is referred to the case in which the robot replicates as is the movement demonstrated by the human operator and not to the case of the generalization of the movement from a set of demonstrated motions (c.f. Section 1.3.2). Both the proposed methods do not need an online interaction with the robot, which makes them suitable for industrial robots without additional layers of safety. The two methods exploit different types of vision sensors and algorithms to reconstruct the trajectory. The first one exploits a ToF camera, whereas the second one a normal 2D digital camera.
In both cases, the general workflow is as follows:

1. The human operator performs the task-based movement by using a *Human Demonstration Device* (HDD), while data acquired by a vision sensor are recorded.
2. The recorded data are processed, the movement reconstructed and converted into a set of poses.
3. Starting from the poses, a ready-to-use program, written in the *Mitsubishi Electric MELFA* proprietary language is automatically generated. The program makes the robot replicate the demonstrated motion.

The algorithms of *Points 2* and *Point 3*, operating offline on the data recorded on *Point 1*, were developed in *MATLAB®*.

4.1 Markerless PbD method using a ToF camera

This first method relies on the use of a ToF camera. It was tested by using the *Microsoft Kinect v2*, mounted as shown in Figure 4.1, also highlighting the three RF

involved (in red, the *Robot base RF*, in yellow the *Kinect depth RF*, in green the *Robot tool RF*).

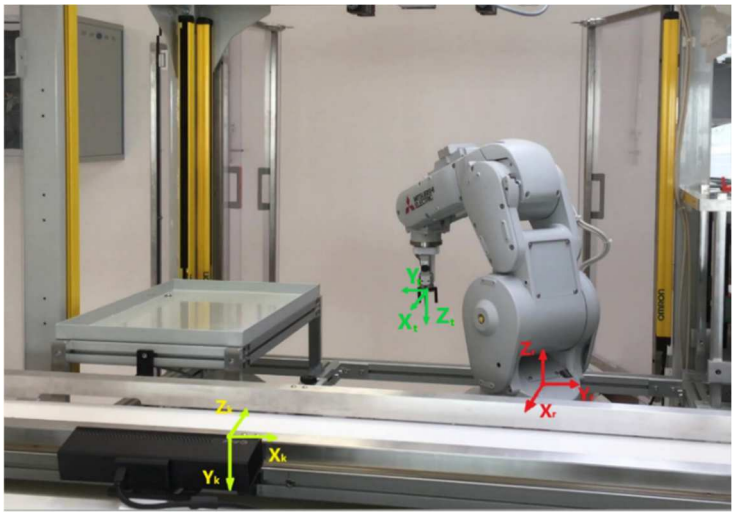


Figure 4.1 Kinect mounting and RFs involved. In red: *Robot base RF*; in green: *Robot tool RF*; in yellow: *Kinect Depth RF*.

This method does not rely on the use of markers; instead, the motion is reconstructed by means of step-by-step (referring to *time steps*) point cloud alignments of the acquired HDD shape, performed by means of the ICP algorithm. The HDD, shown in Figure 4.2, is composed of an external shell, whose dimension and geometry were designed in order to enhance the ICP algorithm effectiveness. Figure 4.2a depicts a CAD model of the external geometry, consisting in a regular polygonal base extruded in depth while wrapping on itself (swept blend). Figure 4.3b shows the actual realization of the device, by means of additive manufacturing (differently from Figure 4.2a, it is hollow inside). The device is grabbed by an internal handle and contains a manual mechanism to open and close a mock-up gripper, used to simulate manipulation tasks. The external geometry can be modified by detaching parts of the external shells.

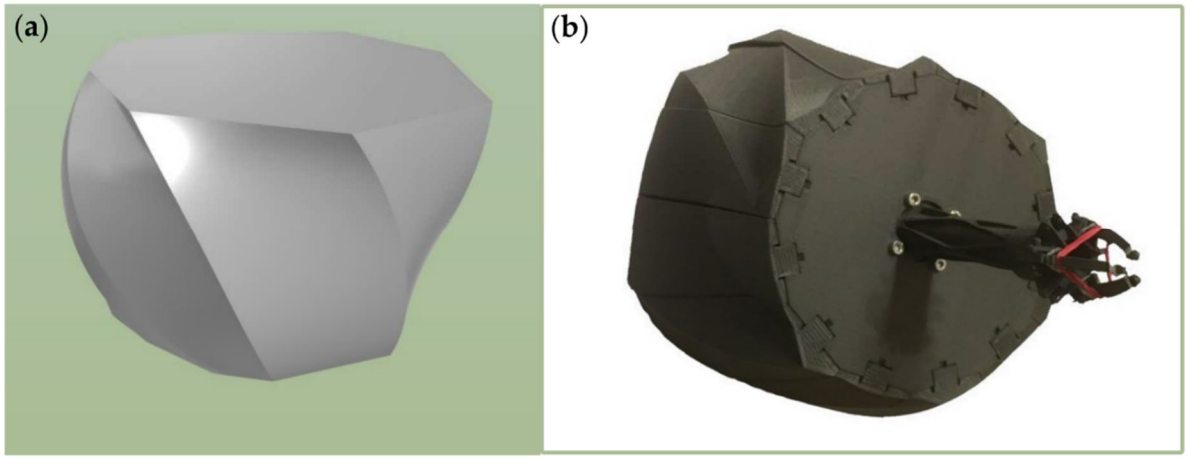


Figure 4.2 (**a**) CAD model of the external shell; (**b**) actual realization of the device, by means of 3D printing.

Figure 4.3 shows some aligned consecutive point clouds (red and blue) by the ICP algorithm. The resulting alignment is good since point clouds does not vary significantly from one time step to the consecutive one. The ICP algorithm is executed on the acquired 3D points belonging to the HDD, extracted from the background. This extraction is automatically performed at each time step by exploiting a bounding box of fixed dimensions, which encapsulates the HDD and is used to filter out other external points. It is defined at an initial instant and transformed step-by-step according to the ICP transformations, as if attached to the HDD.

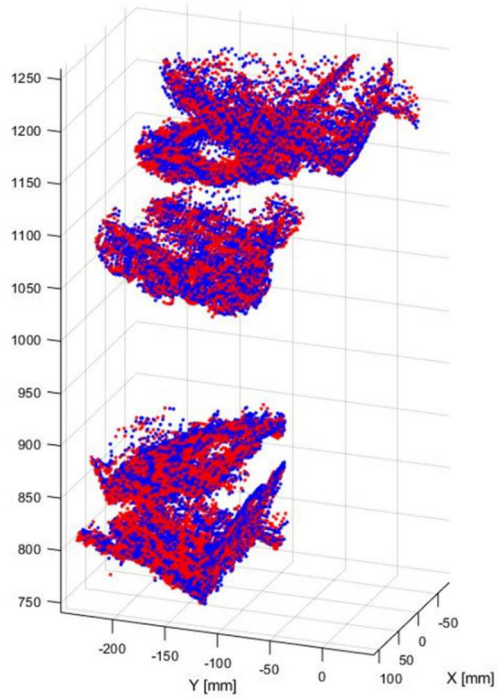


Figure 4.3 Step-by-step ICP alignment process.

Let us consider an acquisition in a time interval composed of *n* time steps, each having a duration equal to the inverse of the Kinect frame rate, being ≈ 30 Hz. The pose A_i^k of *HDD TCP RF* (corresponding to the *Robot tool RF* of Figure 4.1 but placed on the HDD instead), at the time step *i*, can be computed according to Eq. (4.1).

$$A_i^k = T_i T_{i-1} \dots T_3 T_2 A_1^k \tag{4.1}$$

where:

the superscript k stands for "with respect to the *Kinect depth RF*";

A_1^k is the homogeneous matrix representing the initial *HDD TCP RF* pose;

T_i is the homogeneous transformation matrix that allows one to align the HDD point cloud *i*-1 (moving) with the point cloud *i* (fixed), and it is computed by means of the built-in ICP *MATLAB®* function *pcregistericp*.

The pose A_i^k is then referred to the *Robot base RF* (superscript r) by means of Eq. (4.2):

$$A_i^r = T_{rk} A_i^k \tag{4.2}$$

where T_{rk} is the homogeneous transformation matrix that links the *Kinect depth RF* with the *Robot base RF*.

A_1^k and T_{rk} are to be estimated. T_{rk} is fixed and depends on the relative pose between the Kinect mounting position and the robot, and it was estimated via CAD; A_1^k, being the initial pose of *HDD TCP RF*, with respect to the *Kinect depth RF*, was estimated by a manual selection of a set of points on the initial HDD point cloud, univocally identifying the pose of *HDD TCP RF*. These two estimation modalities are coarse and lead to a loss of accuracy, so a potential improvement could consist in exploiting more accurate estimation methods.

The trajectory is defined by the set of poses A_i^r, with i ranging from an initial time step to a final one. In occurrence, this set can be downsampled. Figure 4.4 shows an example of the final set of poses, defining the motion.

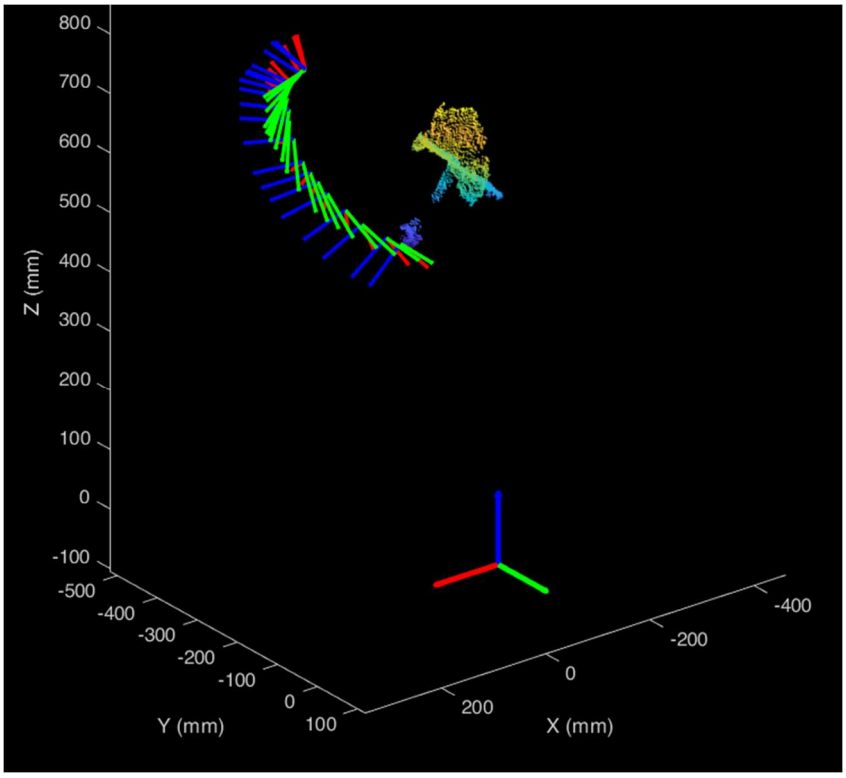


Figure 4.4 Reconstructed motion. The *Robot base RF* and the initial point cloud of the HDD are shown as well.

The final set of poses is then utilized to create a ready-to-use program, written in the *Mitsubishi Electric MELFA* proprietary language. Each pose is converted into a specific string containing the pose information in the *MELFA* language, and a Cartesian linear

movement (*mvs* function in *MELFA*) is specified between each pose (same strategy used in [21]). The junction points are filleted by means of the *MELFA* function *cnt*. Speed is also set. The program generation was automatized, so that the process directly outputs a program file, the only additional operation being to load it within the *Mitsubishi Electric* robot programming software *RTToolBox3*. There, it is possible to test it on a robot simulator and carry out possible final adjustments, before loading it into the robot controller and use it on the real robot. Figure 4.5 shows a comparison between a set of frames extracted from a video of the movement demonstration and the replicated movement performed by the robot.

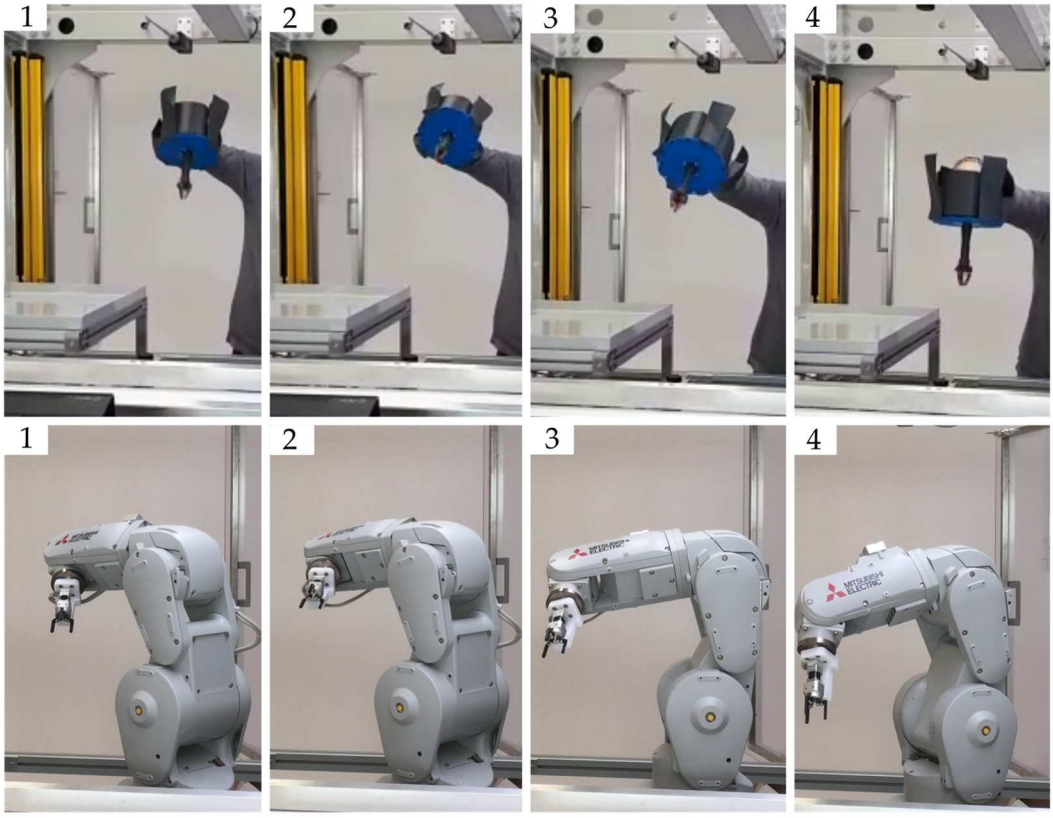


Figure 4.5 Movement demonstration by means of the HDD (top four images, in temporal order from left to right) and replicated movement performed by the robot (bottom four images, in temporal order from left to right).

4.2 PbD using a 2D digital camera and fiducial markers

This PbD method relies on a 2D digital camera, used to detect fiducial markers attached to a specifically designed HDD. More precisely, the webcam *HD Logitech® C930e* was used, mounted as shown in Figure 2.3, whereas as fiducial markers the *AprilTags* [121] were used. Their detection and pose estimation were carried out exploiting the built-in *MATLAB®* function *readAprilTag*. For the pose estimation, the

camera intrinsic parameters were needed, so an intrinsic calibration was performed by using a checkboard pattern and the *MATLAB® Camera Calibration Toolbox*.

The HDD, realized by means of additive manufacturing, is shown in Figure 4.6. It is composed of a cube with five markers attached (on each face except for the bottom one) and a mock-up gripper, comprehensive of a manually actuated mechanism used to open and close it, to simulate manipulation tasks. Figure 4.7 shows an example of the pose detection of the markers: each marker has an RF with origin on the square centre, the *z axis* perpendicular to the marker surface and the *x* and *y axes* parallel to the marker edges; the function *readAprilTag* outputs the pose of each detected *Marker RF* with respect to the *Camera RF*.

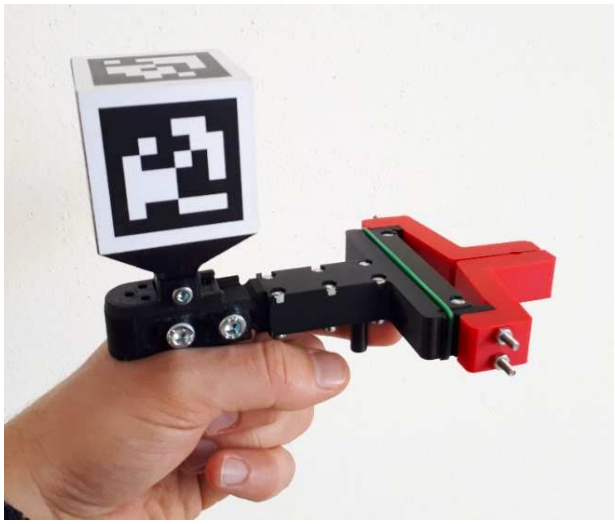


Figure 4.6 HDD device, composed of a cube with markers attached and a mock-up gripper.

Figure 4.7 Example of the detection of three markers and reconstruction of their pose.

Figure 4.8 shows the various RFs involved attached to the HDD.

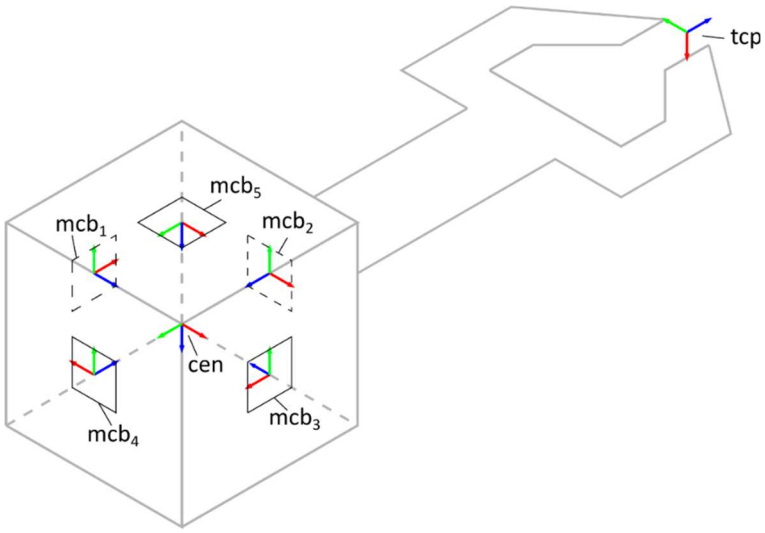


Figure 4.8 Various RFs attached to the HDD.

Let us consider an acquisition in a time interval composed of *n* time steps, each having a duration equal to the inverse of the camera frame rate (being ≈ 60 Hz in the case study, maximum value for the *HD Logitech® C930e*). Let us consider a fixed time step *i*, whose subscript will be omitted for the sake of clarity. Let us denote with A_a^b the homogeneous matrix representing the pose of *a RF* with respect to *b RF*.
The function *readAprilTag* outputs the pose of each tag attached to the cube that is framed and detected by the camera with respect to the *Camera RF* (henceforth abbreviated as *cam RF*). More than one tag can be detected at once, obtaining multiple poses. These poses are combined and used to estimate the pose A_{cen}^{cam}. The poses are combined according to the following steps:

1 – For each detected marker *j*, an estimation of the pose ${}^{j}A_{cen}^{cam}$ is computed as:

$$ {}^{j}A_{cen}^{cam} = {}^{j}A_{mcb}^{cam}\ {}^{j}A_{cen}^{mcb} \tag{4.3} $$

where:

- ${}^{j}A_{mcb}^{cam}$ is the pose of the mcb_j *RF* with respect to *cam RF*, available as output of the *readAprilTag* function.
- ${}^{j}A_{cen}^{mcb}$ is the nominal pose of *cen RF* with respect to mcb_j *RF*, constructed considering the orientations of the RF of Figure 4.8 and the knowledge of the cube side length (= 50 mm);

2 – the translational part $t \in \mathbb{R}^3$ of A_{cen}^{cam} is obtained by averaging the set of translational parts ${}^{j}t \in \mathbb{R}^3$ of ${}^{j}A_{cen}^{cam}$

3 – the rotational part $R \in \mathbb{R}^{3x3}$ of A_{cen}^{cam} is obtained by combining the set of rotational parts ${}^{j}R \in \mathbb{R}^{3x3}$ as follows:

1 – ${}^{j}R$ is converted to a quaternion ${}^{j}Q$

2 – The set ${}^{j}Q$ is averaged by means of the built-in *MATLAB®* function *meanRot,* obtaining a quaternion Q

3 – The quaternion Q is converted back to a rotation matrix R

The matrix A_{cen}^{cam} obtained can be used to estimate the pose of *tcp RF* with respect to *Robot base RF* (abbreviated as *rob RF*), according to Eq. (4.4):

$$A_{tcp}^{rob} = A_{cam}^{rob} A_{cen}^{cam} A_{tcp}^{cen} \tag{4.4}$$

where:

A_{tcp}^{cen} is the pose of *tcp RF* with respect to *cen RF*, and is to be estimated.

A_{cam}^{rob} is the pose of *cam RF* with respect to *rob RF*, and is to be estimated;

A_{tcp}^{cen} was constructed considering the nominal orientations of *cen RF* and *tcp RF* of Figure 4.8 and by a direct measurement of the distances for an estimation of its translational part;

A_{cam}^{rob} was estimated by exploiting another *AprilTag* marker, of larger dimensions (side equal to 160 mm) used this time for a calibration purpose, fixed onto a surface parallel to the robot mounting surface (which is a requisite to use this calibration method, whereas the orientation of the marker in the x-y plane can be arbitrary) and in such a way that was both visible from the camera and reachable by the robot end effector. The robot was equipped with a conic tool and jogged so that the end of the conic tool was coincident with each marker corner, as shown in Figure 4.9.

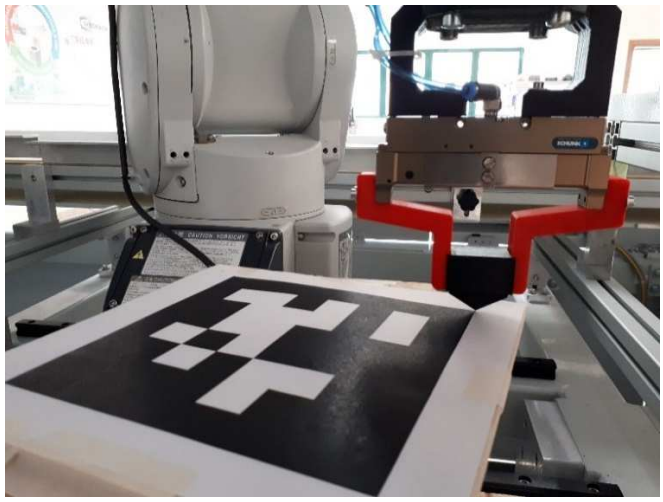


Figure 4.9 Calibration marker, fixed parallel to the robot mounting surface and alignment of the robot end effector with a marker corner, by means of a conic tool grasped by the gripper.

That way, it was possible to read the positions of the four marker corners with respect to the *Robot base RF*. They have a constant Z value (since the marker mounting surface is parallel to the robot base) and different X-Y values. Once the set of marker corner coordinates was read and stored, it was used to fit a square of known side L (L = 160 mm), used to estimate the pose of the *Calibration marker RF* (having origin located at the square centre and x and y axes parallel to the square edges). In order to construct the square from four approximate corner coordinates, two different approaches are proposed and outlined in Figure 4.10a and Figure4.10b respectively, which show them in the exaggerated case, for the sake of visualization, of very unprecise alignments of the conic tool with the marker corners. The actual recorded points are shown in Figure 4.11, along with the constructed square (*Method 2* was used). To construct a square starting from four approximate corner coordinates, the two following methods are proposed:

Method 1: the centre of the square is located at the intersection of the segments joining the opposite points; the orientation of the x and y axes are coincident with the bisectors of the angles created by the segment intersection (note that for construction, the bisectors are always perpendicular between each other).

Method 2: note that each couple of contiguous points is theoretically sufficient to uniquely identify the pose of the *Calibration marker RF* (*mcl RF*): the centre belongs to the median of the segment joining the 2 points, at a distance $L/2$ from the segment. There are four couples of contiguous points. The *mcl RF* origin is obtained as a mean of the coordinates of the centres obtained considering each couple of points; the orientation is obtained as a mean of the orientations, computed, as done previously, by means of quaternions. It is noteworthy that this method is similar to the one used to compute *cen RF* starting from *mcb$_j$ RFs,* but in a space of one lower dimension.

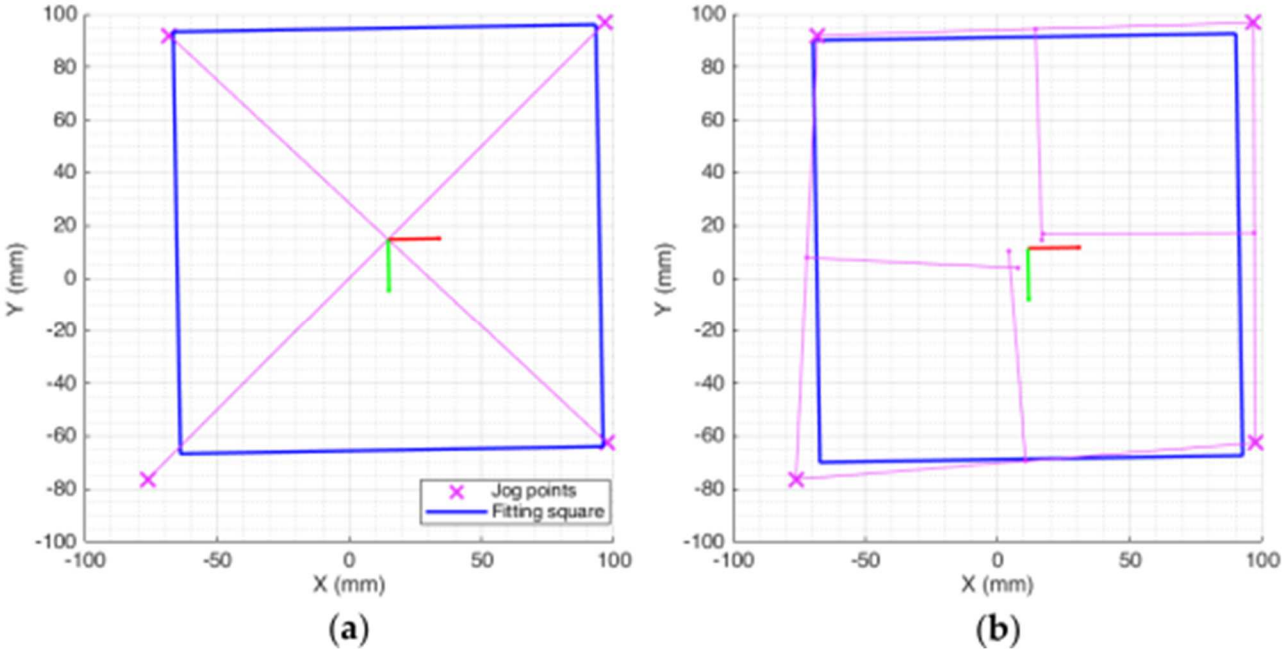


Figure 4.10 Example of application of (**a**) *Method 1* and (**b**) *Method 2* to fit a square of known dimensions (in blue) into four approximate jog points (error exaggerated for the sake of visualization). To better convey how each method works, a set of construction lines (in magenta) are also depicted.

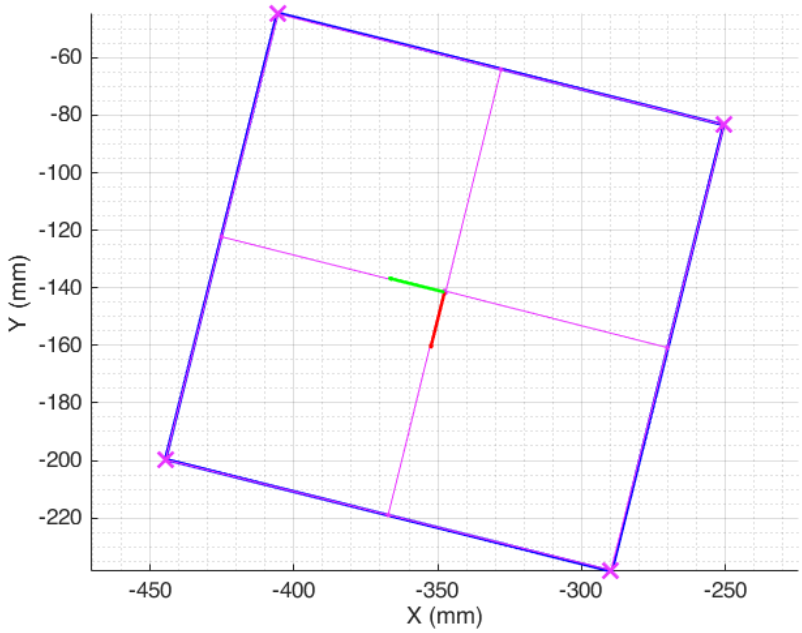


Figure 4.11 Square constructed (*Method 2*) in the case of the actually recorded points by means of robot jogging.

A_{cam}^{rob} can then be computed as the product of two matrices, as shown in Eq. (4.5):

$$A_{cam}^{rob} = A_{mcl}^{rob} A_{cam}^{mcl} \tag{4.5}$$

where:

A_{mcl}^{rob} is the pose of the *Calibration marker (mcl) RF* with respect to the *Robot base (rob) RF*, and is estimated via the previously described procedure, which exploits the alignment between the conic tool and the marker edges.

A_{cam}^{mcl} is the pose of *cam RF* with respect to *mcl RF*. It is computed as the inverse of A_{mcl}^{cam}, this latter being the output of the function *readAprilTag*.

Once each matrix of Eq. (4.4) is estimated, at each time frame the pose of *tcp RF* with respect to *rob RF* can be computed, and thus the motion reconstructed (an example of that is shown in Figure 4.12). Then, the set of poses representing the motion is automatically converted into a ready-to-use program, following the same procedure used for the PbD method of Section 4.1.

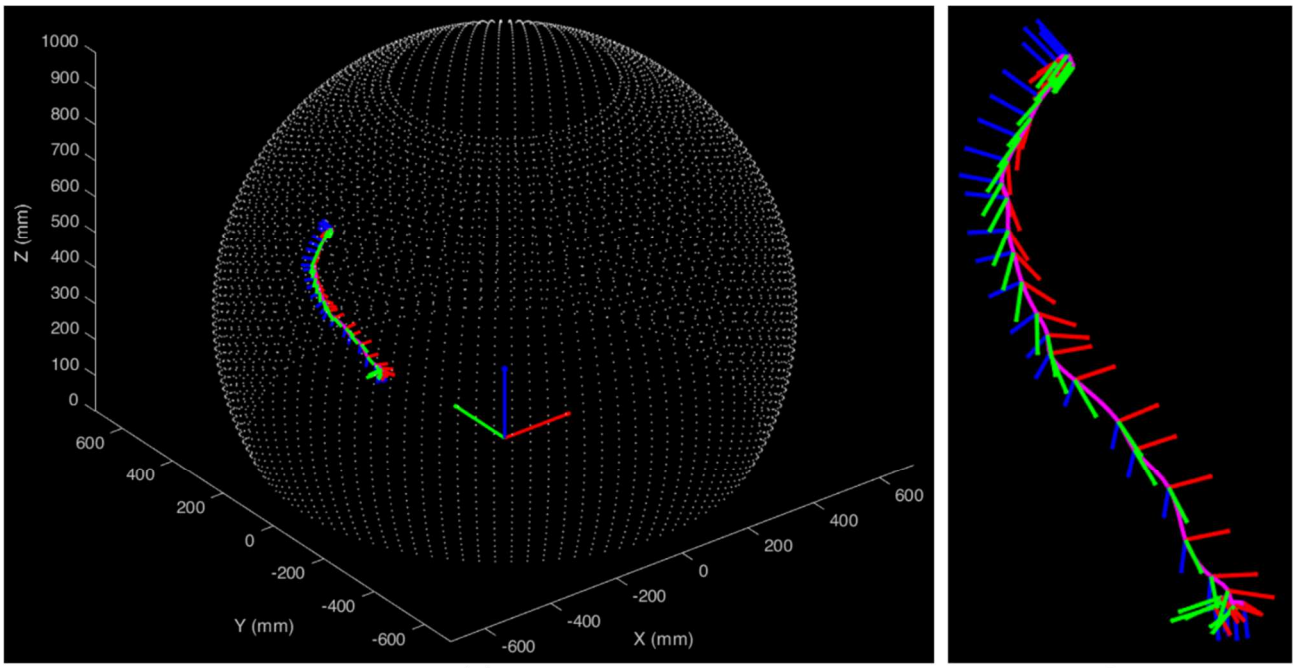


Figure 4.12 (**a**) Reconstructed motion; the *Robot base RF* and a sphere representing the robot workspace are shown as well. (**b**) close-up on the set of poses representing the motion.

Figure 4.13 shows a comparison between a series of frames extracted from videos of (**a**) demonstrated movement and (**b**) movement replicated by the robot, corresponding to a part of the motion of Figure 4.12.

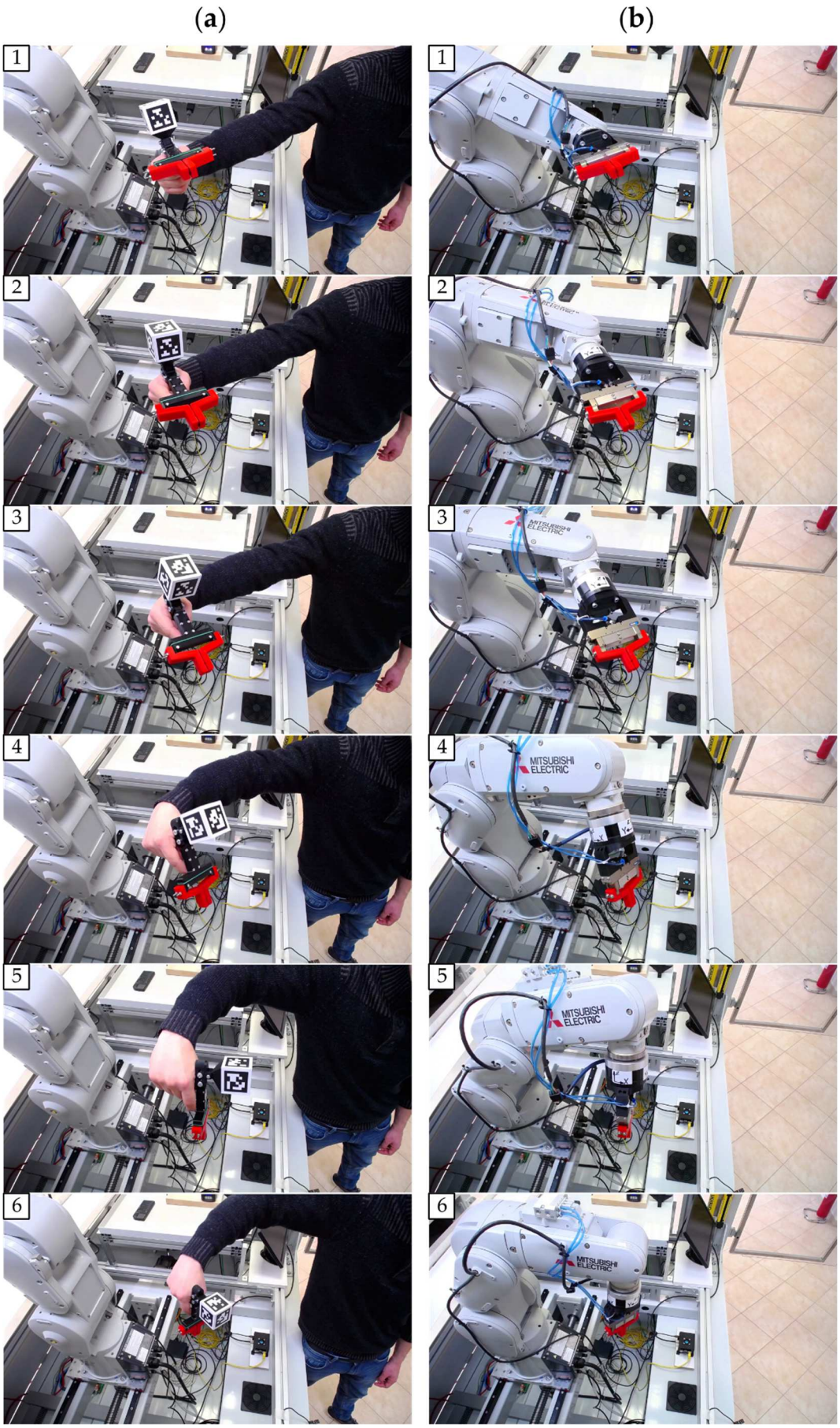


Figure 4.13 Comparison between a series of frames extracted from videos of (**a**) demonstrated movement and (**b**) movement replicated by the robot.

Figure 4.14 shows another comparison between a series of frames extracted from videos of (**a**) demonstrated movement and (**b**) movement replicated by the robot, in case of a simple pick-and-place task. The opening and closing of the gripper were accounted by providing as input to the algorithm, other than the video, the video timestamps in which the gripper was opened and closed (they were graphically evaluated). A more sophisticate and user-friendly way to account for gripper opening/closing is planned to be implemented in future developments.

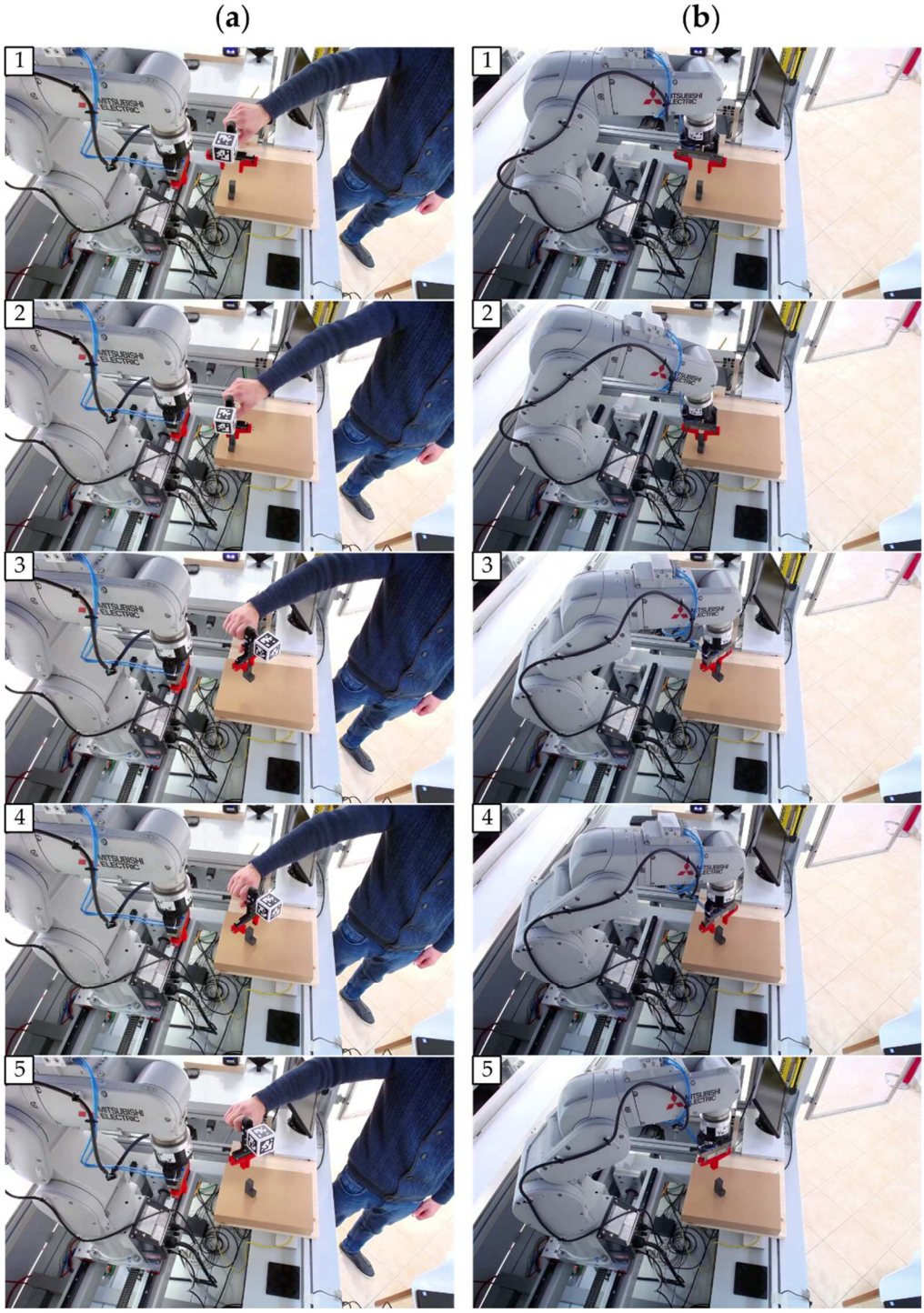


Figure 4.14 Comparison between a series of frames extracted from videos of (**a**) demonstrated movement and (**b**) movement replicated by the robot, in the case of a simple pick-and-place task.

Figure 4.15 shows a distribution of the position error of *cen RF*, obtained considering 993 video frames (only the ones having at least two detected markers, more precisely two or three), extracted from various recorded videos of demonstrated motions. For each single frame, the error is computed according to Eq. (4.6):

$$Error = \frac{\sum_{j=1}^{n_m} \left\| O_j - O \right\|}{n_m} \tag{4.6}$$

where n_m is the number of detected markers, O_j is the origin of *mcb*$_j$ *RF*, O is the origin of *cen RF*.

The distribution has a *mode* of 1.08 mm, a *median* of 1.46 mm and a *mean* of 1.98 mm. Errors lower than 4-5 mm are mainly attributable to the achievable precision of the function *readAprilTag*, dependent on the combination of the tag dimension, working distance, image resolution and image quality. Errors higher than around 4-5 mm are observed to be mainly due to motion blur (higher in case of higher errors), which significantly degrades the image quality and thus the precision and reliability of the marker detection and pose reconstruction.

It is noteworthy to point out that an evaluation of this type is only useful for an estimate of the precision of the method, but not of the accuracy, since a ground truth would be needed. The accuracy was only qualitatively evaluated by visually observing the difference between the poses of the HDD and of the actual robot gripper. This was easier in the case of an interaction with elements of the scene, used as reference, such in the case of the task of Figure 4.14.

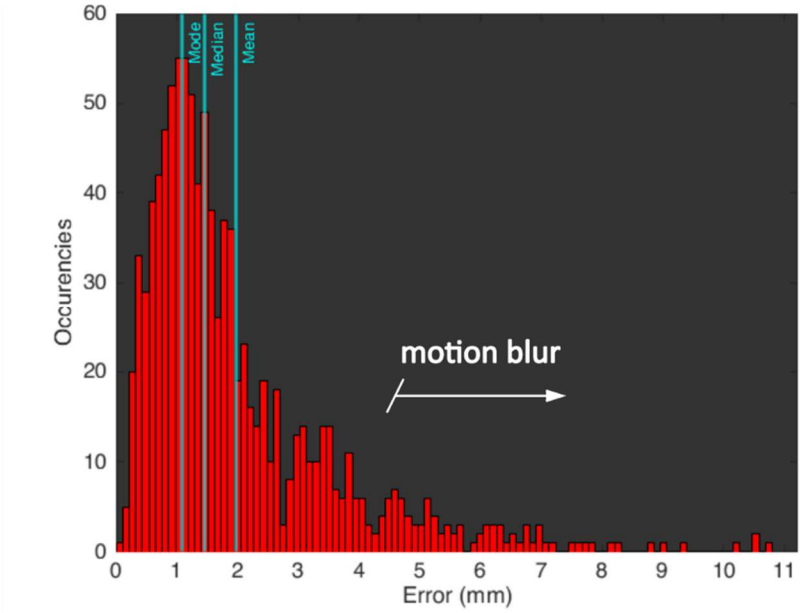


Figure 4.15 Error distribution considering 993 frames extracted from various videos of demonstrated motions. Only the case of multiple detected markers is considered. Cyan vertical lines (from left to right) refer to the following distribution parameters: *mode* = 1.08 mm; *median* = 1.46 mm; *mean* = 1.98 mm. Error higher than around 4-5 millimetres are observed to be due to increasing motion blur.

4.3 Discussion

The two methods, in their current implementations, have different pro and cons, here briefly presented and descripted.

Markerless PbD using a ToF camera:

Advantages:

- Very low dependency on the scene illumination (one of the perks of ToF cameras).
- Movement can be demonstrated at (relatively) high speed.

Drawbacks:

- Low accuracy in the initial pose estimation.
- HDD relatively bulky.
- The movement reconstruction depends on the step-by-step alignment, so disturbance in the acquisition (e.g. temporary occlusion) can lead to problems in the point cloud alignments.

PbD using a 2D digital camera and fiducial markers:

Advantages:

- Compact HDD.
- The movement is reconstructed considering independently each time frame, so disturbances in the acquisition do not affect the whole movement reconstruction. Also, the initial pose is estimated as the others.
- Cheap and commonly available vision sensor (mid-end 2D camera).

Drawbacks:

- Precision dependant on the image quality (more precisely, on the quality of the marker shape in the image), affected by several factors, such as:
  - Illumination.
  - Speed of the demonstrated motion: if the motion is too fast, motion blur might appear, which can drastically lower the precision. Motion blur depends mainly on the camera frame rate, so this issue can be mitigated by choosing a camera model able to operate at a high frame rate.
  - Markers may go out of focus. In the experimental tests, focus was manually set to a fixed value, so that the acquisition was more reliable. However, if the motion spans in a large space portion, the markers could significantly go out of focus, lowering the precision of the detection.

In general, the second method seems more promising since its drawbacks can be drastically mitigated by a careful design of the lighting system and a proper choice of the 2D digital camera. High-end video cameras feature both high frame rate and resolution. The focus issue can be addressed by implementing strategies to dynamically adapt the focus during the acquisition, so that the detected markers are always kept on focus, also if their distance to the optical centre significantly varies. More tests are planned to be conducted to refine the method and find an optimal setup to improve its precision and robustness. An additional way to improve the precision can be to substitute the cube of markers with a different polyhedron (or to consider a completely different marker disposition and number), for instance an icosahedron [21] or a dodecahedron [122]. Polyhedrons of these latter types guarantee a higher number of detected markers (higher precision), but, on the other hand, being equal the HDD dimensions, markers have to be smaller (lower precision).

5 Investigation on the placement of onboard cameras

In this Chapter, an investigation on the placement of onboard cameras to maximize the observability of the workspace of an articulated robot is presented. As was emphasised throughout the Thesis (and applied in the collision avoidance method of Chapter 3), inside the context of HRC, vision sensors can have the function of enabling a safe coexistence between robots and human operators, by monitoring the shared area with the aim of detecting dynamic obstacles. The placement of vision sensors is a crucial aspect, since it determines the efficacy in detecting the obstacles present in the area that one wants to monitor, and thus has important implications on safety. In the ideal case, one wants to be able to detect with sufficient certainty every obstacle that could possibly appear inside the area to monitor, which can coincide with robot workspace, include part of it or its totality. That is a non-trivial problem, and one first step to tackle it concerns the choice of the sensor mounting configuration, which can be divided in two types: vision sensors can be mounted on a fixed frame or on the robot links (onboard). This latter configuration has the advantage of being effective in unstructured environments and can efficaciously monitor the workspace region that the robot itself would occlude if monitored by external fixed cameras. Whilst a number of procedures have been proposed to optimally locate fixed cameras, to the best of the Candidate's knowledge no optimization technique for placing cameras on robot links is present in literature. This was the main motivation of the study presented in this Chapter, consisting in a numerical procedure for optimizing the placement of cameras on the moving links of an articulated robot, with the aim of maximizing the observability of its workspace. Given the nature of the topic, the indices and variables used in this Chapter, defined case by case, are to be intended stand-alone and not linked to the ones of the other Chapters.

5.1 Background

Inside the context of HRC, cameras can serve to predict collisions before they occur and enable the *safe coexistence* collaboration level. To fulfil this scope, cameras can be mounted on the robot or on a fixed frame, but rather than pointing largely towards the robot end effector (*eye-to-hand*) or being fixed on the robot end effector (*eye-in-hand*), the idea is to monitor the whole robot operating area and the space region

nearby, to account for any approaching dynamic obstacles. In this sense, one procedure is to mount fixed cameras in the upper parts of the frame, pointing downwards towards the robot [76,82,123]. Whilst on one hand this leads to the monitoring of a wide and strategic view, on the other hand, as outlined in [55,91], placing fixed camera around the robot workspace could lead to some limitations: one is the fact that these fixed camera systems are not capable of detecting obstacles that, for certain robot configurations, happen to be occluded by the manipulator itself: in facts, it could happen that certain space regions become confined between the robot links and the ground, thus being hidden or difficult to monitor by fixed external cameras. The occlusion problem can be particularly marked when dealing with big industrial robots, such as the *Kuka KR180* (180 Kg of payload), which was used in [33] for testing a new human-robot safety strategy. One further and relevant limitation is that fixed sensors require to some extent a structuring of the environment. In the case of robots mounted on mobile platforms ([17,18,124]) the assumption of a structured environment is not valid. Furthermore, one key feature of collaborative robots is that they can be easily set to be operative, so that their mounting spot can be changed without difficulty if needed. Another aspect that goes against the assumption of a structured environment is the fact that modern workcells can have a modular or modifiable structure. To overcome these limitations, a different strategy is to mount vision sensors on robot (onboard). Some recent papers in which vision or distance sensors are mounted onboard are for example [55,71,91,125]. This strategy appears promising also because of the recent development of small ToF cameras, such as the *PMD Pico Flexx*, the *Stereolabs Zed Mini* or the even smaller ones embedded in the new generation smartphones.

In general, when there is the need of monitoring a space region by mounting a set of cameras, the problem that arises is to find the optimal number of cameras and their optimal placement. In the case of fixed cameras in a robotic framework, some studies have been carried out in literature (e.g. [105,106]). In the case of onboard sensors, in [125] an optimization of the distance sensors arrangement is carried out (tested and further developed in [55]): distributed distance sensors are used (i.e. a high number of spot sensors mounted in clusters), whose optimal position and number was investigated in a set of feasible nodes. Although the optimal placement of onboard distance sensors has already been investigated, to the best of the Candidate's knowledge no studies are present in literature concerning the optimal placement of onboard vision sensors.

This study aims at finding the optimal placement of cameras on the links of an articulated robot, with the scope of maximizing the observability of the workspace region that is likely to be hidden to external cameras, which can result in critical issues

in the case of the presence of dynamic obstacles. In the case that both fixed and mobile cameras are present, the application of this method can be used to decouple the problem of the optimal placement of fixed sensors from the one concerning onboard sensors. The fixed sensors can be placed with the aim of maximizing the overall monitoring of the workspace and the adjacent space region, whereas the onboard sensors with the aim of monitoring the area hidden to the external cameras by the robot. The general idea is to provide some insights and instruments to tackle a broader problem, which is the one concerning the comprehensive monitoring of the full space around the robot, which is a very hot topic due to the important implications on tasks where safety of human operators is to be guaranteed. The problem addressed in this Chapter concerns the monitor of a variable region of space which is enclosed by the robot, and thus variable as a function of configuration. The aim is to find the placement of cameras on the robot links that, for a given set of robot configurations, maximizes the monitoring of the *space region covered by the robot*, namely the space region confined between the robot links and the robot mounting surface.
The underlying mathematical problem is an original variation of the *camera positioning problem*, which in turn is a variation of the *art gallery problem*. An exhaustive explanation of the problem (which has computational complexity *NP-hard*) and its most common variations can be found in [126].

The main contribution of this study is the definition of an original method to tackle the problem, by adapting existing discrete approaches used in the case of the optimization of the placement of fixed cameras ([126-128]). The problem is addressed in a simplified bidimensional version, in the case of an articulated robot, and for a number of cameras ranging from one to five.

5.2 Materials and Methods

5.2.1 Problem simplification

The problem is tackled in the case of an articulated robot (with 6 DOFs) and under the following hypotheses, depicted in Figure 5.1:

1. cameras are mounted exclusively on the robot links;

2. the problem is treated as bidimensional;

3. cameras can only be mounted on the front side of each robot link.

To tackle the 2D optimization problem, the robot links are modelled as segments. To explain the *hypothesis 3* let us consider each segment representing each link as an

oriented segment, that starts from the point P_i and ends at the point P_{i+1} (see Figure 5.1). The front side of each robot link is then defined as the segment side that faces the right half-space (with respect to its orientation) formed by the straight line containing the segment.

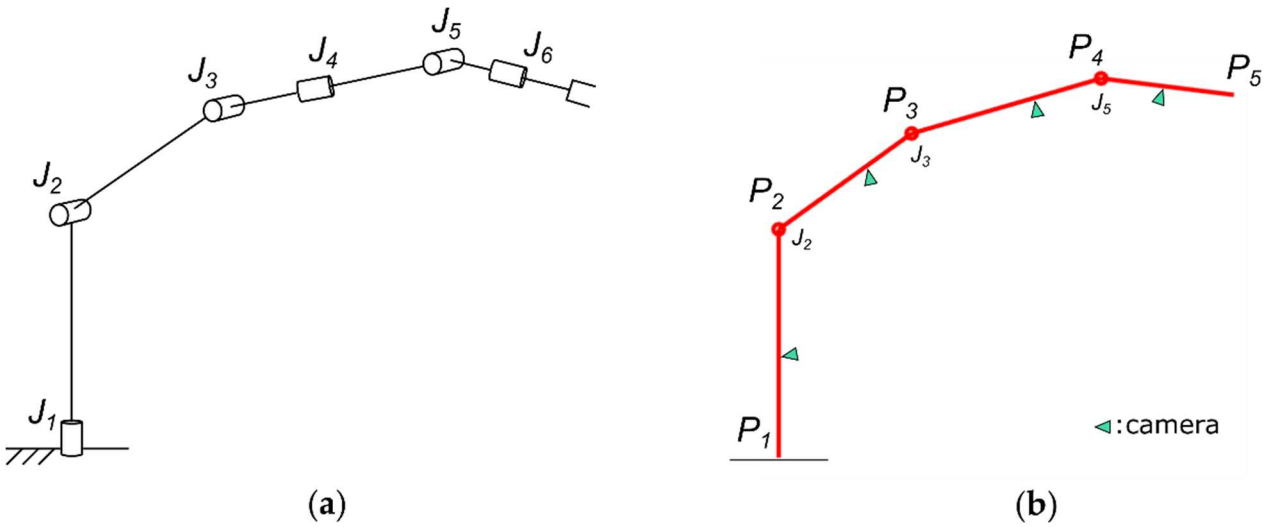


Figure 5.1 (**a**) Three-dimensional robot model, with highlighted the six revolute joint J_1, ..., J_6. (**b**) Bidimensional robot model, with a camera placement that satisfies the hypotheses (cameras represented as green triangles).

5.2.2 Optimization procedure

First, a definition of the *space region covered by the robot* is presented (see Figure 5.2): let us consider the polygon obtained by casting the "vertical shadow" of the robot links. Only the polygon parts that have at least one link with the front side that faces the generated shadow are kept, since no camera placement allows to monitor the other polygon parts (because of *hypothesis 3* of Section 5.2.1). The final geometries of these polygons, which are hereinafter referred to as *robot polygons*, are shown in Figure 5.2 (dark orange regions) for different robot configurations. The joint limits and link lengths were modelled similarly to the ones of the articulated robot *Mitsubishi Electric RV4F*. The last link was extended by 100 mm to consider the presence of an end effector. The more general problem of the case in which *hypothesis 3* is not made can be tackled by first solving the problem under *hypothesis 3* and then solving a specular and independent problem with the hypothesis that the cameras are to be mounted only on the rear side of the links (left half-space of the oriented segments) instead of on the front side. If the joint limits are symmetric, then this latter problem has the same solution of the first one (same optimal placement but on the rear side).

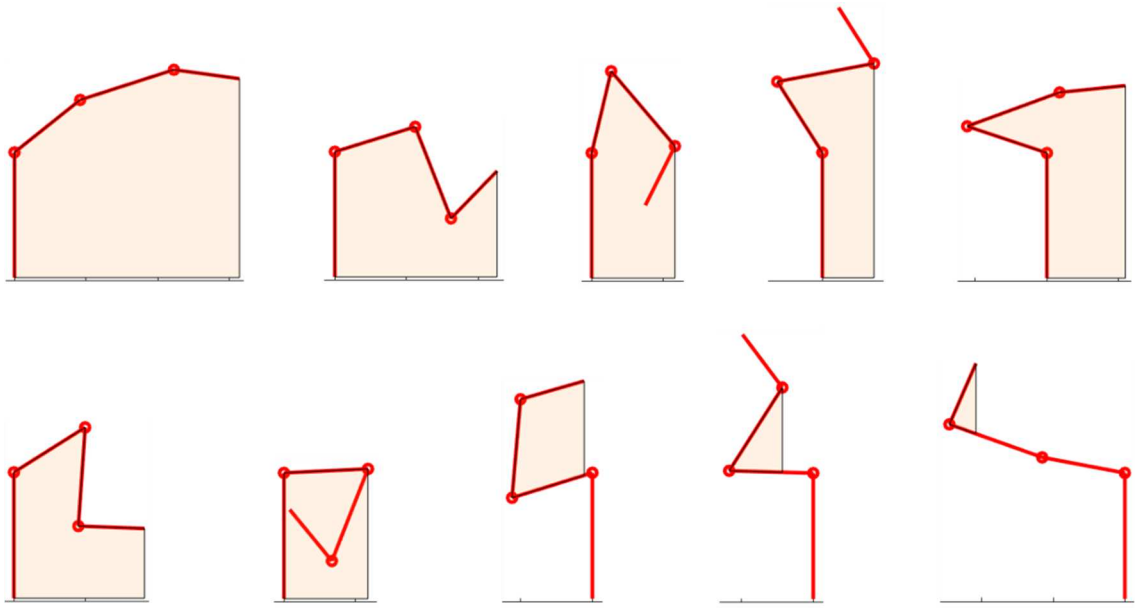


Figure 5.2 *Robot polygons* for different robot configurations.

The *robot polygons* are the polygons whose monitor is to be maximized. The idea is to maximize a quantity, henceforth referred to as *degree of monitoring*, that represents the effectiveness of the camera placement considering the whole set of robot configurations. In each robot configuration, cameras monitor a fraction of the *robot polygon*. This fraction depends on their placement. With reference to Figure 5.3, let us call R_i the *robot polygon* of the i^{th} configuration (i = 1, ..., N) and C_j the viewing frustum of the j^{th} camera (j = 1, …, n, where n is considered fixed in this analysis).

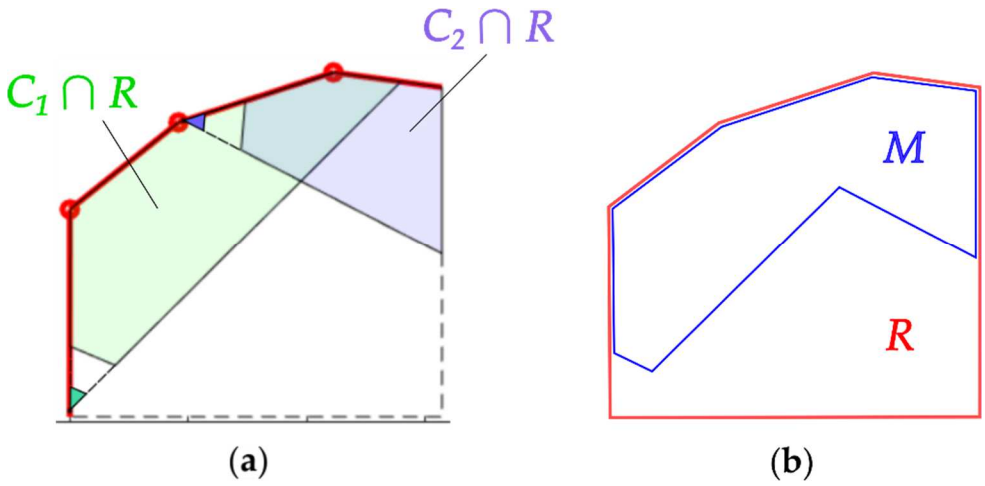


Figure 5.3 Various polygons involved. (**a**) In light green, intersection of the viewing frustum of the *camera 1* with the *robot polygon*, in violet, intersection of the viewing frustum of the *camera 2* with the *robot polygon*. (**b**) In red: *robot polygon*; in blue: total fraction of the *robot polygon* seen by the cameras.

In the robot configuration i, the fraction M_i of the *robot polygon* R_i seen by the cameras can be computed according to Eq. (5.1).

$$M_i = \bigcup_j (R_i \overset{ray}{\bigcap} C_j) = f(s, \gamma) \tag{5.1}$$

C_j is a function of both the vector s = (s_1, ..., s_n) containing the position of each camera, and the vector γ = (γ_1, ..., γ_n) containing the orientation of each camera. The intersection between R_i and C_j is not a simple intersection, but it is a ray intersection, since the *robot polygon* is not necessarily convex (for an example, see Figure 5.4).

The *degree of monitoring* is quantitatively represented by three indices, presented in Eqs. (5.2-5.4) and described hereafter, each one more suitable for a specific optimization scope.

$$r_1 = \frac{\sum_{i=1}^{N} \text{area}(M_i)}{\sum_{i=1}^{N} \text{area}(R_i)} \tag{5.2}$$

$$r_2 = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{area}(M_i)}{\text{area}(R_i)} \tag{5.3}$$

$$r_3 = \min_i \left\{ \frac{\text{area}(M_i)}{\text{area}(R_i)} \right\} \tag{5.4}$$

- the index r_1 weights the robot configuration based on the *robot polygon* area, so robot polygons with a small area (e.g. the one in bottom-right in Figure 5.2) affects less the camera placement;
- the index r_2 gives the same weight to each robot configuration regardless of the area of the *robot polygon*, and it is to be interpreted as a simple mean among all the robot configurations of the ratio M_i/R_i, which quantifies the *degree of monitoring* for a single configuration;
- the index r_3 considers only the minimum value of the ratio M_i/R_i and it is suitable to be considered in situations in which a minimum camera coverage is to be guaranteed in all the configuration, for example for safety purposes.

To speed up the computations, after the ray intersections of Eq. (5.1), for which a custom analytical algorithm was exploited, the various polygons involved were converted into masks composed of pixel elements. These masks are binary matrices, and the mask areas can be computed simply by counting the number of pixels equal to one in the corresponding binary matrices. The parameters search space was discretized as well.

The camera viewing frustum was modelled as the one of the depth camera *PMD Pico Flexx* (considering its vertical FOV, equal to 45°), which is a ToF camera. ToF cameras have a blind spot (in which the measure is not reliable) near the optical centre, so the FOV results to be a trapezoid. 1000 robot configurations were considered, obtained by randomly sampling a set of reachable configurations via a Monte Carlo method, as done in [125, 129].

Figure 5.4 shows an example of the polygon resulting from a ray intersection between a camera viewing frustum and the *robot polygon*, converted afterwards into a binary matrix, showed for different resolution in *Pixel Per Centimetre* (PPC).

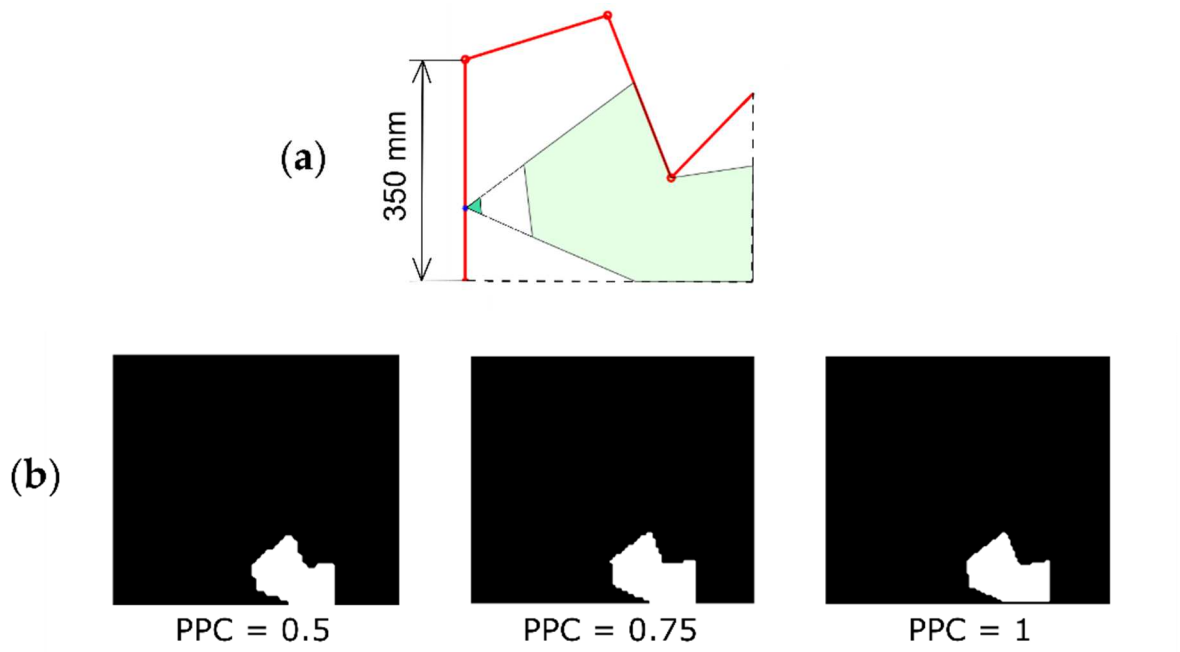


Figure 5.4 (**a**) Ray intersection check between the camera frustum and the *robot polygon*. Light green region: polygon obtained after the ray intersection between the camera viewing frustum and the *robot polygon*; small dark green triangle: camera; (**b**) conversion of the polygon obtained into a binary mask of different resolutions (expressed in pixels per centimetre).

Once defined the three indices of Eqs. (5.2-5.4) The optimization problem is then formulated according to Eq. (5.5):

$$\max_{s,\gamma} f \tag{5.5}$$

where, based on which quantity one wants to maximize, f can be intended as r_1, r_2 or r_3 (referring to Eqs. (5.2-5.4)).

The study is performed considering up to five cameras and carried out separately for each number of cameras. The optimization problem has computational complexity *NP-hard*, and the computational time rises exponentially by rising the number of cameras. Up to four cameras it was possible to solve the problem via an exhaustive

search. In the case of five cameras, a simple trust-region based metaheuristic algorithm was exploited. The algorithm was run starting from each combination of camera positions so that it explores only the γ space. A fixed number of random neighbours are generated in the *n*-dimensional γ space within a certain radius and the greatest one is picked and becomes the current γ if it is greater than the current γ. The cycle repeats with a gradually decreasing radius, until the process becomes a simple hill-climbing algorithm.

5.3 Results

In this section, the results concerning the optimization procedures in the case of number of cameras ranging from one to five are reported. Two studies were conducted, based on the choice of the candidate positions and orientations. These two studies, presented in Section 5.3.1 and Section 5.3.2, were carried out by considering a fixed FOV equal to 45° for each camera. One further analysis, presented in Section 5.3.3, concerned the evaluation of the effect of the variation of the FOV of the cameras, and was carried out in the case of one and two cameras.

The algorithms were implemented in *MATLAB®* 2020b, by exploiting the parallel computing toolbox to speed up the computations, whereas the hardware used consisted of a Dell Precision 3520 Laptop Intel Core i7- 7700HQ-CPU 2.80 GHz, 4 cores.

5.3.1 First type of positions and orientations discretization

In this first analysis, ten positions and nine orientations (for each position) were considered as candidates for the camera placements, and chosen as shown in Figure 5.5.

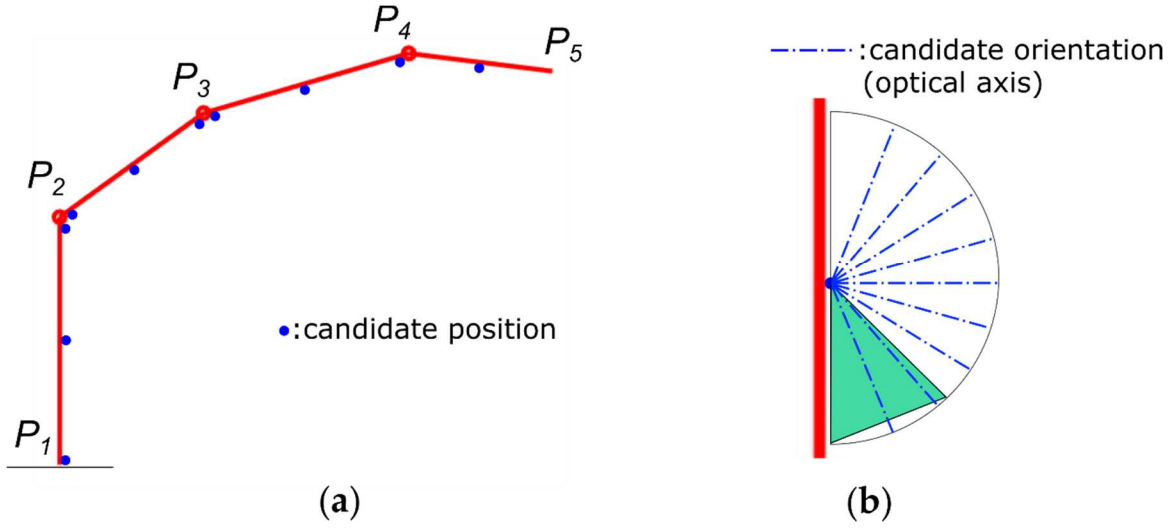


Figure 5.5 Camera candidate positions and orientations.

In this analysis, only the placement of one camera in each position is considered. The candidate camera positions were chosen in the median point of each robot segment and in proximity of each segment endpoints, except for the segment P_4P_5, where only the median point was considered as a candidate point. The candidate camera orientations were chosen by dividing in eight parts the angle interval that has as extrema the case in which the camera has the FOV tangent to the link where it is mounted. Figure 5.6 shows the optimal placement of the cameras obtained by maximizing in turn each one of the three different indices. The figure contains only one robot configuration, but the optimal placement is to be intended for the whole set of the robot configurations.

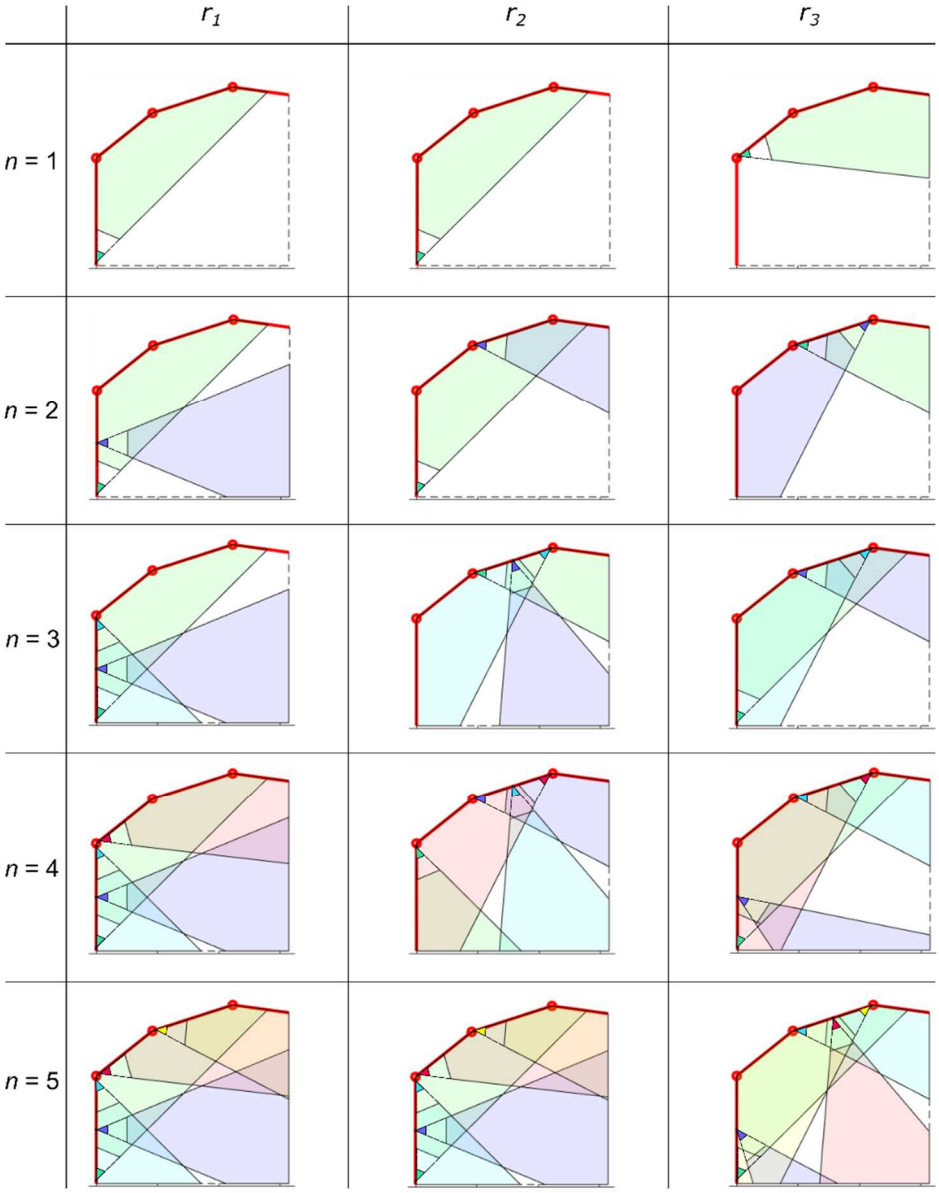
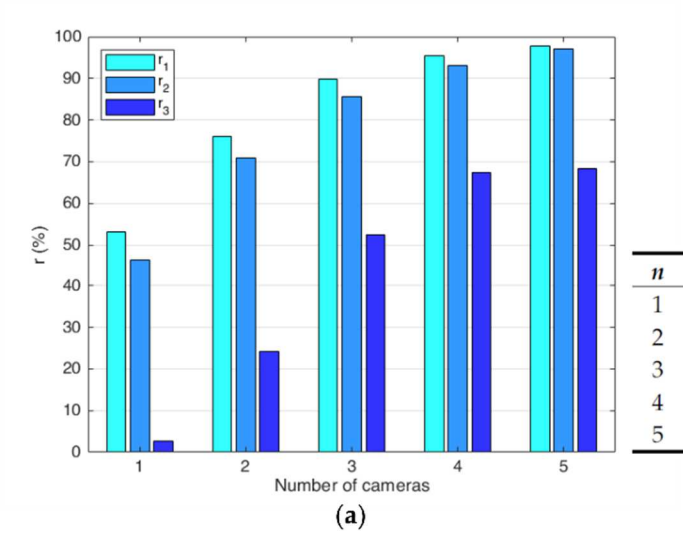


Figure 5.6 Optimal camera placements (one to five cameras) according to the maximization of the three different indices r_1, r_2 and r_3.

Figure 5.7 shows the optimal values of the indices, expressed in percentage, corresponding to the placements illustrated in Figure 5.6.



(a)

n	r_1 (%)	r_2 (%)	r_3 (%)
1	53.2	46.3	2.6
2	76.1	70.8	24.1
3	89.8	85.6	52.5
4	95.4	93.0	67.5
5	97.7	97.1	68.3

(b)

Figure 5.7 Optimal values of the three indices considered, expressed in percentage, for a different number of cameras, in a bar plot (**a**) and in tabular form (**b**).

5.3.2 Second type of positions and orientations discretization

In this second study, the number of candidate positions was increased whereas the number of candidate orientations was reduced. In particular, the same positions of the first study were considered, but this time with the possibility of placing multiple cameras in each one, whereas the candidate orientations were limited to five for each position. Figure 5.8 shows the placements found with this study. Only the placements for which an improvement in the values of the indices was achieved with respect to the first analysis are shown. The placements not shown are the same of the corresponding ones in the first analysis.

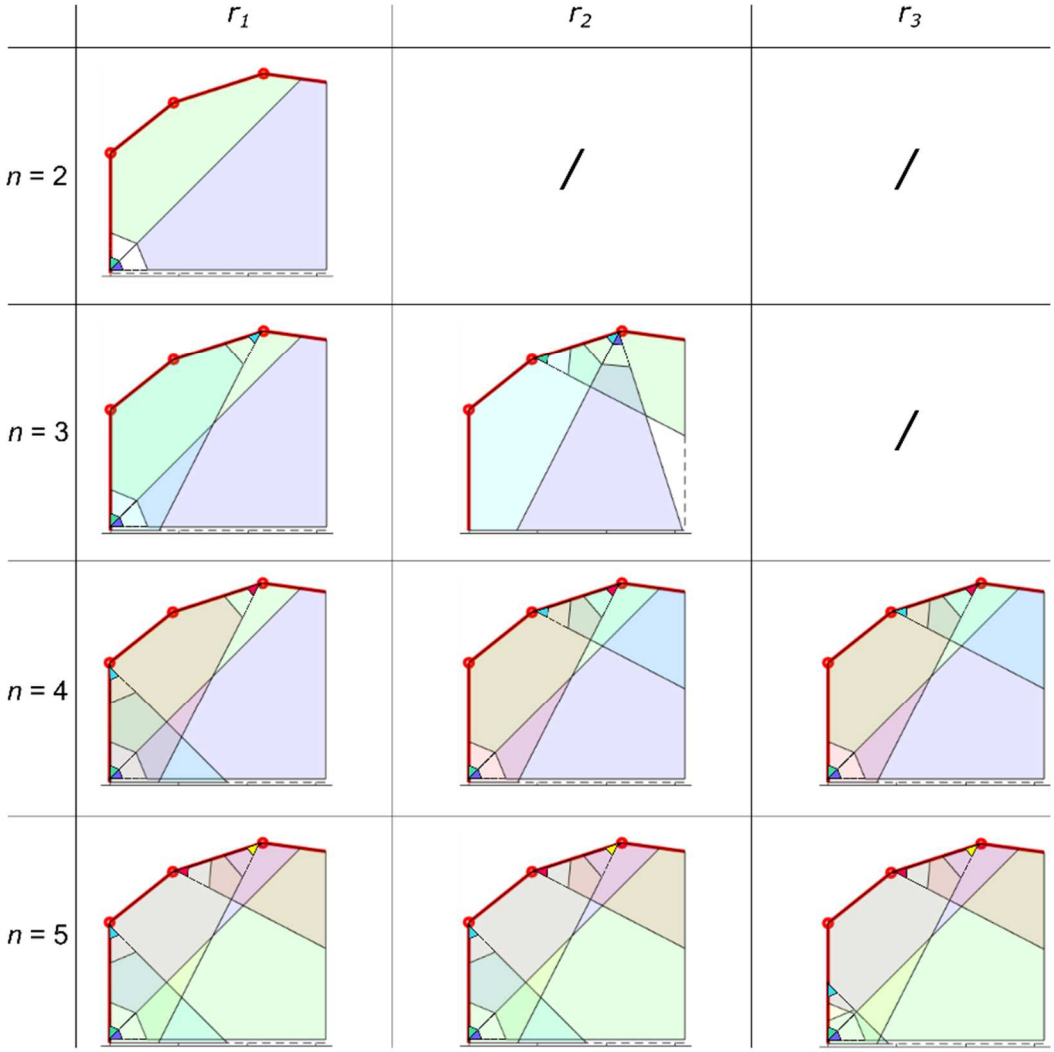
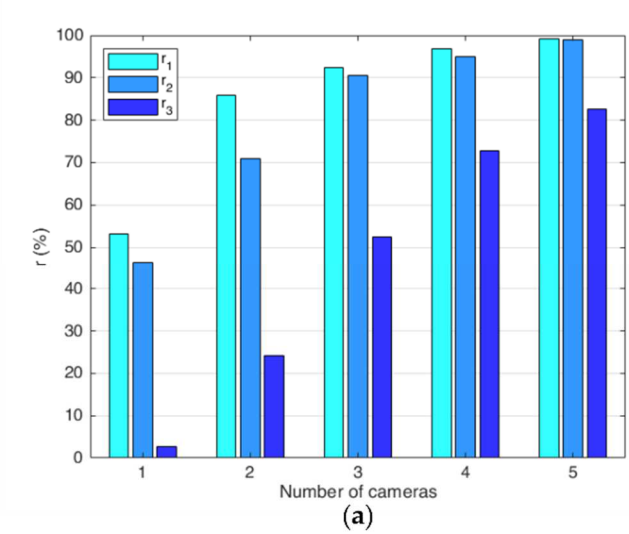


Figure 5.8 Optimal camera placements (one to five cameras) according to the maximization of the three different indices r_1, r_2 and r_3.

Figure 5.9 shows the optimal values of the indices, expressed in percentage, obtained by considering, where occurred, the improvements obtained by the second study.



(a)

n	r_1 (%)	r_2 (%)	r_3 (%)
1	53.2	46.3	2.6
2	85.8	70.8	24.1
3	92.4	90.6	52.5
4	96.8	95.0	72.7
5	99.2	98.9	82.7

(b)

Figure 5.9 Optimal values of the three indices considered, expressed in percentage, for a different number of cameras, in a bar plot (**a**) and in tabular form (**b**), obtained considering the two studies.

5.3.3 Effect of the variation of the Field of View

This analysis was conducted with the aim of assessing the influence of the FOV on the optimal value of the three indices. This study was carried out in the case of one (Figure 5.10a) and two cameras (Figure 5.10b), and the FOV of both cameras was varied between 15° and 60° (with a step of 5°). The results shown are obtained by performing both the two studies presented in Section 5.3.1 and Section 5.3.2 and by choosing the highest values obtained from them.

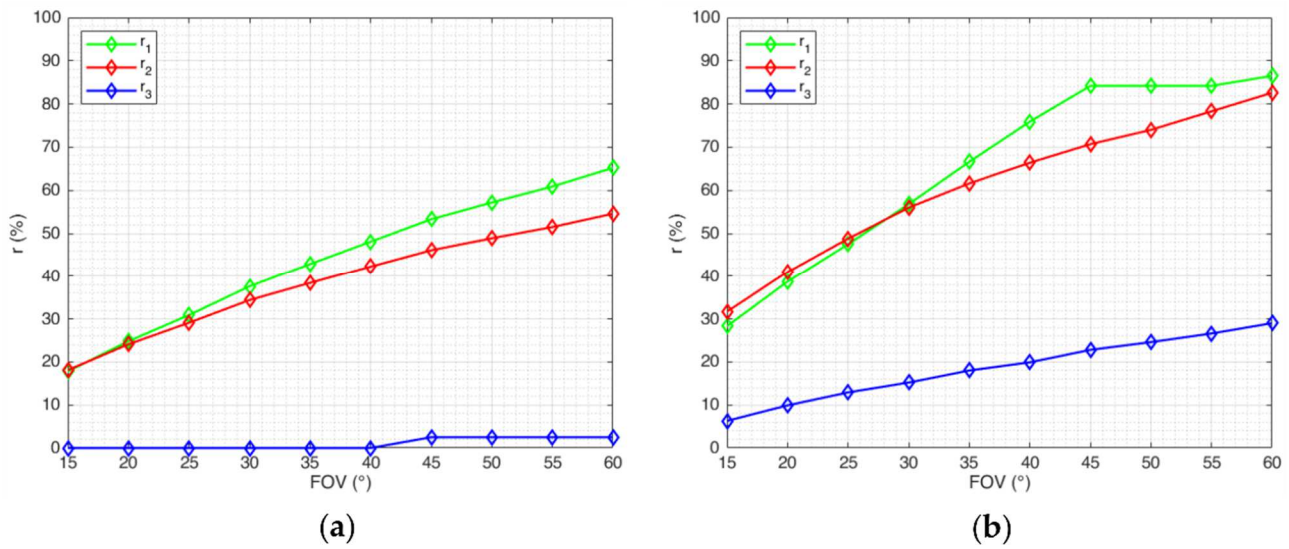


Figure 5.10 Variation of the optimal indices by varying the FOV of the cameras, in the case of (**a**) one and (**b**) two cameras.

5.4 Discussion

The results obtained give various insights about the optimal placement of cameras on the links of an articulated robot. One first consideration, which emerges by analysing Figure 5.6 and Figure 5.8, is that, at optimum, cameras tend to be placed oriented with the FOV tangent to the robot link on which they are mounted, or possibly tangent to the FOV of another camera, if mounted on the same position. Furthermore, the optimal positions are likely to be on the first and third robot segment, near the joints. Another consideration (cf. Figure 5.9) is that the optimal value of the indices r_1 and r_2 increase in a less-than-linear way by increasing the number of cameras. In scenarios in which there are no constraints on a minimum percentage of the *robot polygon* to be always monitored, so that the indices r_1 or r_2 are suitable, results show that a good degree of monitoring can be achieved (up to roughly 99% with five cameras). An optimization based on the index r_3, on the other hand, can be particularly useful in

situation in which the sensing of the obstacles in the critical area under the robot is of utmost importance in every robot configuration, for example for safety reasons. In the case of five cameras, the optimal values of the index r_3 is around 83%, which is a high value but still seems not suitable to ensure a proper monitoring for safety purposes. However, it can be significantly improved in specific cases in which the set of configurations taken into account can be restricted. For what concerns the effect of the camera FOV, the graphs of Figure 5.10 show that it has a big impact on the indices r_1 and r_2, but it has a less significant impact on the index r_3, especially in the case of one camera.

The advantage of using vision sensors instead of distributed distance sensors to ensure a safety-aimed monitoring is that it is possible non only to sense obstacles, but to extract more detailed information about the obstacles, that can be better combined and elaborated, with, for instance, the aim of recognize the obstacle shape and type and behave consequently. This type of approach seems particularly promising considering the increase in the availability of small off-the-shelf ToF cameras: the more the sensor is compact, the more of them is possible to mount on the robot links; using a sufficiently high number of cameras, it is predictable that for a certain number of cameras the r_3 index will step to 100% (which automatically implies an equality to 100% even for r_1 and r_2). This would grant that each *robot polygon* can be fully monitored in each configuration, which is particularly suitable for safety purposes. With proper computational capabilities, and possibly with an improvement of the meta-heuristic algorithm, this method can be applied to a greater number of cameras.

5.4.1 Notes on the computational time

Considered the *NP-Hard* nature of the problem, the numerical implementation of the exhaustive search had to be carefully designed to avoid redundant computations, and its basic concept is here briefly described. The problem was divided into three main parts. For each part, the number of computational steps T is here indicated, in terms of the "big O" notation.

N: number of robot configurations; n_s: number of camera positions; n_γ: number of camera orientations (for each position).

1. Computation of the *robot polygon* (R_i in Eqs. (5.1-5.4), Section 5.2.2) for every robot configuration. $T(N) = O(N)$.
2. Computation of each possible polygons obtained by ray-intersecting the frustum of a single camera with the *robot polygon*, considering all the possible

placements (position and orientation) of a single camera and all the robot configurations. $T(N, n_s, n_\gamma) = O(N*n_s*n_\gamma)$. These polygons are then converted to binary masks.

3. This is the *NP-hard* part, which is the core of the problem. For an exhaustive search, all the possible unions of n masks extracted from the set generated in the *Point 2* are to be evaluated. $T(N, n_s, n_\gamma, n) = O(N*C_{ns,n}* n_\gamma{}^n)$ where $C_{ns,n}$ represents the number of combinations of n_s positions by group of n elements. This is because different cameras are considered to have different positions, whereas different cameras can have the same orientations (still, the case of k cameras placed on the same position can be considered by inserting that position k times as input in the algorithm).

To limit the computational time, the product $N*n_s*n_\gamma$ has to be sufficiently low. For the *Point 3*, this condition is not enough for a large number of cameras, since the steps increase exponentially. In this case, the exhaustive search can be substituted by a proper optimization method (usually metaheuristic methods are used in this type of scenarios), so that the steps do not increase exponentially; the downside, however, is that only a sub-optimal solution is guaranteed. One further consideration on the computational time is that it is highly affected (Figure 5.11a) by the resolution of the grid used to compute the binary masks. This resolution is to be set as low as possible without compromising the results; the value of 0.5 pixels per centimetre was chosen by an experimental evaluation on how its variation impacts on the results (Figure 5.11b shows it in the case if two cameras).

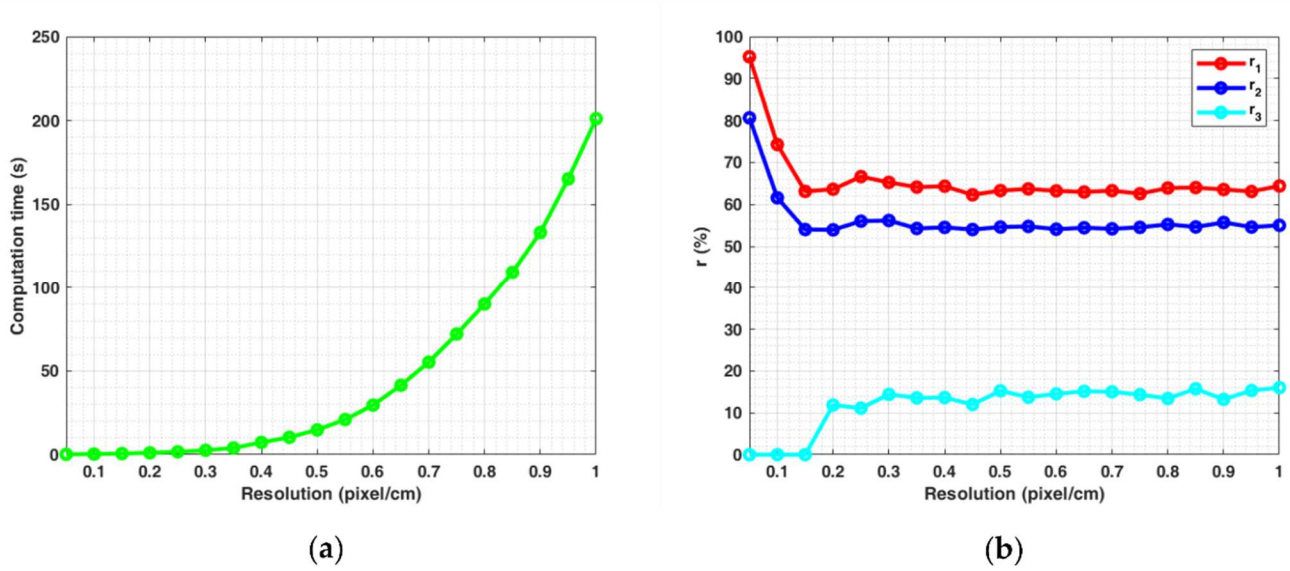


Figure 5.11 (**a**) Influence of the resolution of the binary mask grid on the computation time, in the case of two cameras; (**b**) optimal values of the indices for different values of the resolution, in the case of two cameras.

5.4.2 Notes on the 3D case

The problem tackled was a bidimensional problem, even if, in reality, the robot configuration can lie outside the plane of Figure 5.1b (if the joint angle J_4 is different from zero) in a way that depends on the combined value of the joint angle J_4 and J_5. Furthermore, cameras mounted after the joint J_4 rotate outside the plane jointly with J_4 and, if mounted after J_6, their 3D orientation depends on both J_4, J_5, and J_6. In the bidimensional case, results show that no optimal placement is on the robot last link, so the implications of mounting cameras after the joint J_6 will not be discussed. For small rotations of the joint J_4, a common situation in a series of practical applications, the hypothesis of bidimensional problem holds true. In case the hypothesis of small rotations of J_4 does not hold true, the solution of the 2D problem can be used as a starting point to find the solution for the more general 3D problem. One way to proceed could be to consider the 2D problem as a sub-problem of the 3D one, and solve this latter by mounting additional cameras given the optimal placement found solving the proposed 2D problem. Future research is aimed at better investigating and developing this point.

5.4.3 Final remarks

The high-level aim of this study was to provide some additional tools and insights useful to tackle the complex problem of granting a complete and effective monitoring of the workspace of an articulated robot. This examined question is closely connected to HRC and its safety-related issues. More specifically, the aim of this study was to propose a methodology to maximize the observability of the space region of the workspace that tends to be confined between the robot and its mounting surface. This region is difficult to be monitored by fixed external cameras, since it can be occluded by the robot itself, so onboard cameras were considered. Also, external fixed cameras have the drawback of requiring some structuring of the environment, differently from onboard cameras. A bidimensional version of the problem was tackled by using a discrete approach, which is typical when dealing with *NP-hard* problem of this type. The optimization was carried out up to five cameras on three different proposed indices r_1, r_2 and r_3, which quantify the effectiveness of the camera placement. The indices r_1 and r_2 quantify the overall observability, whereas the index r_3 consider the worst-case observability and thus it is more safe-oriented. Several insights emerge from the analysis: one is that, at optimum, cameras tend to be placed near the robot joints and oriented with the FOV tangent to the link where they are mounted. Furthermore, up to five cameras, the indices are shown to increase in a less-than linear way by increasing the number of cameras. Even if the proposed

optimization relies on these indices, it is to be pointed out that the proposed approach can still be effective on indices constructed in different ways and tailored to specific situations.

Conclusions

Various vision-based solutions aimed at enabling HRC inside industrial workcells have been proposed in this Thesis. These solutions have the scope to overcome the limitations of both the pure industrial and the collaborative paradigms, by endowing industrial robots with artificial vision. This way, the high productivity typical of industrial robots can be combined with the high versatility typical of collaborative robots.

The first solution, central in the Thesis, consisted in the development of a collision avoidance method, which permits a safe coexistence between human operators and industrial robots. The method was developed for an articulated robot, but can be adapted, with minor modifications, also to other serial robot models, such as SCARA robots. The method produces real-time adjustments of a pre-programmed task-based robot trajectory, by exploiting data acquired from depth cameras. This allows to modify the trajectory only when needed, enhancing the robot functional flexibility while maintaining, when possible, the high dynamics typical of industrial robots, which ultimately grants elevated productivity. The proposed methodology has some distinctive features, concerning both the online trajectory generation and the obstacle tracking. The trajectory generation relies on the use of safety constraints, a recent approach which guarantees that the robot always keeps a distance from obstacles higher that a protective separation distance, while minimizing the distance to a reference setpoint value. Two methodologies to generate the trajectory have been proposed, one more conventional in which the setpoint is computed based on the reference motion laws and an alternative one in which the setpoint is computed based on geometrical considerations. The latter was observed to produce motions with better adherence to the shape of the reference trajectory, deviating from it only for a period strictly necessary to maintain the protective separation distance. For what concern the obstacle tracking, a methodology to track generic dynamic obstacles has been proposed based on the use of a voxel grid in combination with a GPU-accelerated particle filter, the latter being used for a fast and obstacle-independent speed estimation. Some significant perks of the use of the voxel grid consist in making it possible a straightforward data fusion, enhanced with algorithms that accounts for boosts in measurement confidence in the case of using heterogeneous sensors. The fact that the obstacle tracking was designed for generic obstacles is significant, since it can extend the use of this methodology to the case of a widely unstructured environment, where it might happen that not only human operators, but also robots and other generic moving objects (e.g. moving parts of a

machinery) enter the robot workspace. Further tests are planned to improve several aspects of the proposed methodology, in particular the smoothness of the generated command and the general reliability and robustness.

In the second part of the Thesis, two vision-based programming by demonstration methods have been presented. The scope was to provide intuitive interfaces (suitable tool for HRC) to facilitate the programming of industrial robots, for which the hand guiding modality typical of collaborative robots is not available. The methods consist in making the robot mimic a motion performed by a human operator through a human demonstration device, whose movement is captured by vision sensors. The first method relies on the use of a ToF camera and a markerless human demonstration device, whereas the second one on the use of a 2D digital camera combined with 2D fiducial markers attached to a specifically designed human demonstration device. The second method resulted more promising and further developments are planned, in order to improve its accuracy and precision.

In vision-based HRC applications, camera placement has important implications on safety, which motivates the study, presented on the third part of the Thesis, on the optimal placement of onboard cameras, to maximize the observability of the workspace of an articulated robot. The aim was to maximize the observability of the space region of the workspace that tends to be confined between the robot and its mounting surface, thus being difficult to be monitored by fixed external cameras, which also have the drawback of requiring some structuring of the environment. The analysis, carried out considering a bidimensional simplified case, reports the optimal camera placement in case of different numbers of cameras, and gives a series of insights possibly useful also in real and more complex scenarios. Improvement of the method and future developments may consist in considering a more realistic model of the robot links and addressing the more general three-dimensional optimal placement problem of onboard vision sensors, with the aim of improving the robot functional flexibility and safety.

Appendix A. Design of the robotic cell

In this Section the design of the robotic cell, to which the Candidate devoted the most part of the first year of the PhD, is described. The cell was designed from scratch, with the aim of realizing a functionally flexible testbed suitable for both academic research and industrial applications.

A.1 Layout definition

The knowledge of the two robot models, which were provided to the *TAILOR* laboratory by *Mitsubishi Electric*, represented the starting point in the design of the robotic cell. Exploiting this initial information, the first step was the definition of the layout, specifically the choice of the distance between the two robots, the choice of the various devices to include and where to place them, and the overall dimensions of the frame. Figure A.1 shows a top-view (**a**) and front view (**b**) of the final layout, in which the robot workspaces, the rotary table, the conveyor belts and the fixed worktables are depicted. These devices were chosen with the aim of realizing a layout possibly resembling an automatic robotized machine. At the same time, the focus was not on a specific task, but on granting the possibility of testing different types of applications. Since the *RV4F* can translate due to the railway, its workspace (showed in red) is not fixed, but can be moved (dashed lines) towards the worktable at its left or towards the *RH1F* robot (workspace fixed, shown in blue), up to a significant workspace intersection, that allows tasks in cooperation. The rotary table, shown in green, was placed in a position reachable from both the *RH1F* and the *RV4F* (given that is properly positioned on the railway). In occurrence, the position of the rotary table can be modified to a certain extent (dashed green lines). In addition, also the height of the rotary table can be adjusted by manually turning a knob (the mechanism is descripted more in detail in Section A.2.4).

The conveyor belt crossing the whole the robotic cell, henceforth named *Conveyor belt A*, was placed in such a way that it intersects the workspaces of both robots, in order to be accessible by both, for example for pick-and-place operations “on the fly” on the moving conveyor belt. The other conveyor belt, henceforth named *Conveyor belt B*, has the function of transporting objects on the rotary table.

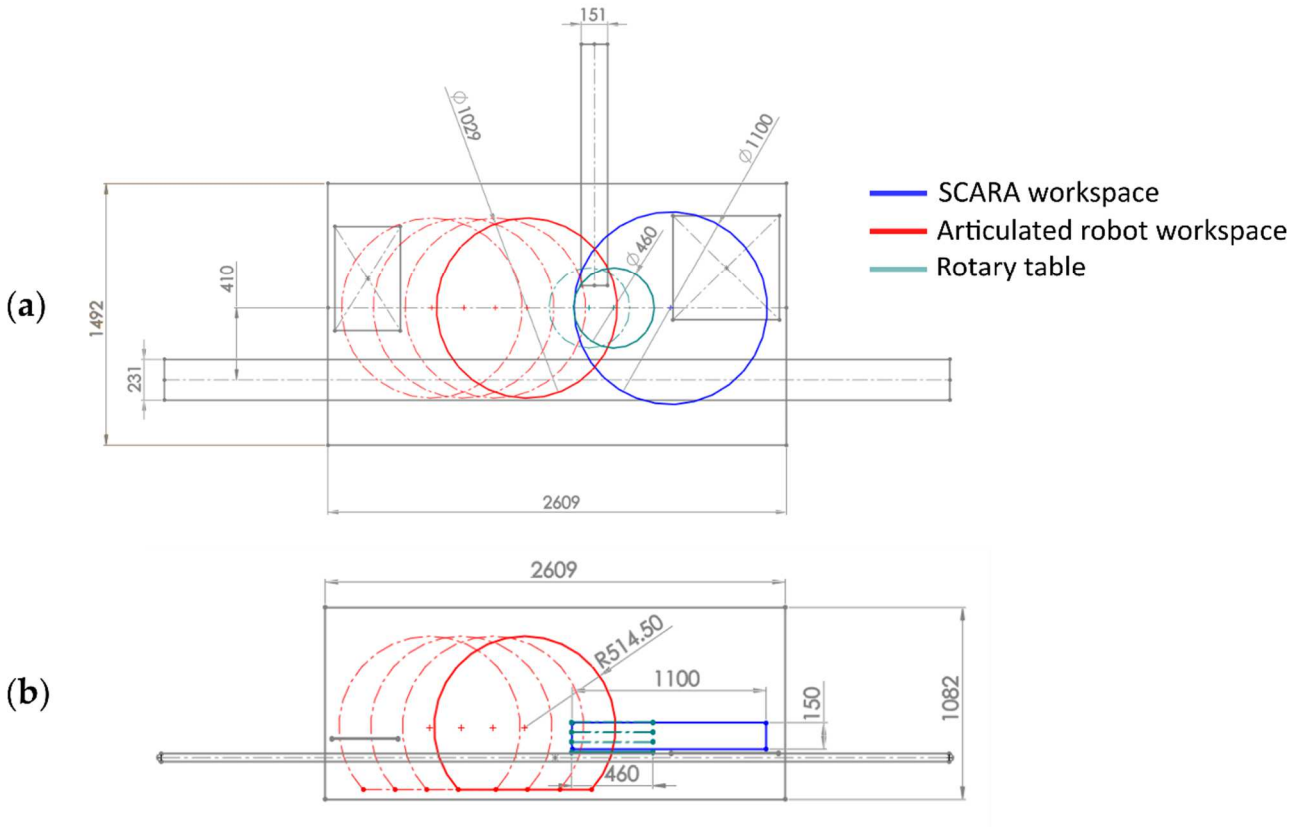


Figure A.1 Layout of the robotic cell. (**a**) top view; (**b**) front view. In red: *RV4F* workspace (dashed line: possible translation, thanks to the railway); in blue: *RH1F* workspace; in green: rotary table (dashed line: possible translation); other elements in grey: robotic cell internal facades, conveyor belts, worktables.

Figure A.2 shows a top view of the CAD model, to further illustrate the disposition of the various elements inside the robotic cell. Other than the devices of Figure A.1, also the areas (at the height of the *Conveyor Belt A*) framed by the three *Omron FH-SCX* cameras (cf. Figure 2.3) are shown. Referring to Figure A.2, *CAMERA 0* and *CAMERA 1* were mounted over the *Conveyor belt A* to frame the objects transported on it, either entering or exiting the cell (the conveyor motion is possible in both directions). The camera optics were chosen to frame only a window of the width of the conveyor belt, so that the resolution is fully exploited. *CAMERA 1* has wider optics and it is placed so that it can frame both the rotary table and the *Conveyor belt A*.

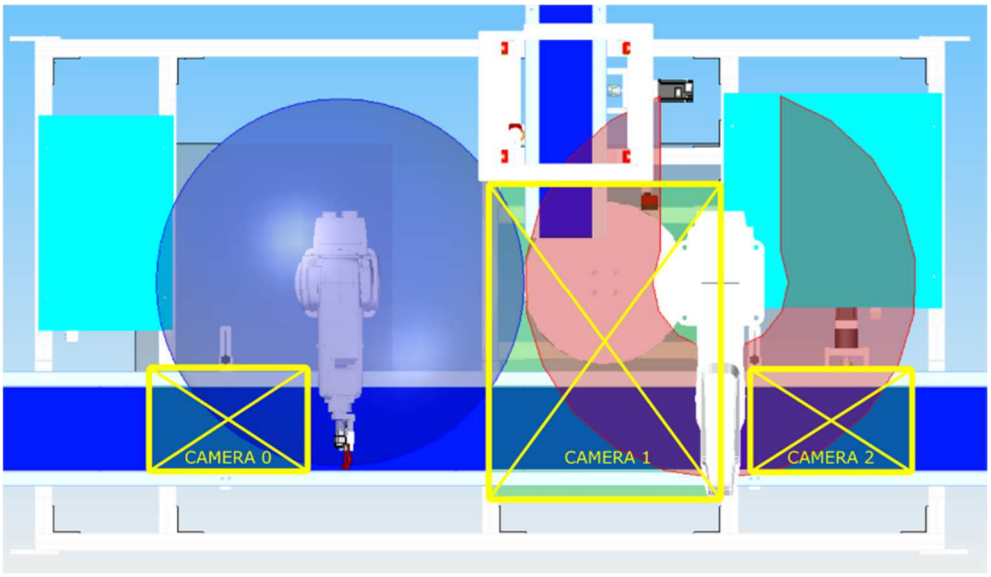


Figure A.2 Top view of the robotic cell CAD, where the main elements of the layout can be seen: in clear blue and clear red the workspace of the *RV4F* and of the *RH1F*, respectively; in blue the conveyor belts, in light blue the fixed worktables, at the centre the rotary table, in yellow the areas framed by the *Omron FH-SCX* camera at the height of the *Conveyor belt A*.

A.2 Mechanical design

For the mechanical design, the CAD Software *SolidEdge* was utilized. First, starting from the layout previously defined, a set of assembly groups were identified, and for each the various components were 3D designed. Afterwards, the as-built drawings were created for each single and assembly parts. For the manufacturing, except for the frame, most of the non-commercial components of the various assembly groups were realized in aluminium by CNC machining. In the next part, each main assembly group is briefly described.

A.2.1 Frame

Two main requirements were accounted in the design of the frame. The first one was to bestow it with sufficient rigidity, needed to limit the vibrations due to the robots' high dynamic, especially of the *RH1F* robot (SCARA robots are renowned for their high speed). The second one was to make it transportable through the stairs of the *TAILOR* laboratory, located at the first floor of the building. This second necessity required the division of the frame in different parts, to be assembled once transported in place. It is noteworthy the fact that dividing the frame in several parts affects its rigidity in a negative way, so a trade-off between the two requirements was necessarily made. The structure was realized in a series of welded parts as shown in Figure A.3, transported in place and then assembled together by means of bolts. The structure is mainly composed of *Fe360* hollow tubulars of 80 x 80 mm section and 4 mm thickness.

Also, a thinner welded part was realized in the rear as a place for the electrical panel. To confer further rigidity to the structure, gusset plates were mounted at the frame corners, as shown in Figure A.3b. Figure A.3b also shown other elements, such as supporting plates in the lower part of the frame, a thick plate onto which the *RH1F* robot was mounted (ceiling mounting) and a planar structure composed of aluminium profiles, which served as a mounting place for the two conveyor belts, the rotary table, and the fixed worktables.
The aluminium profiles were used since they have the advantage of granting an easy fixing of the components over them and the possibility of modifying their position in a straightforward way, which accounts for future modifications, thus granting more functional flexibility.

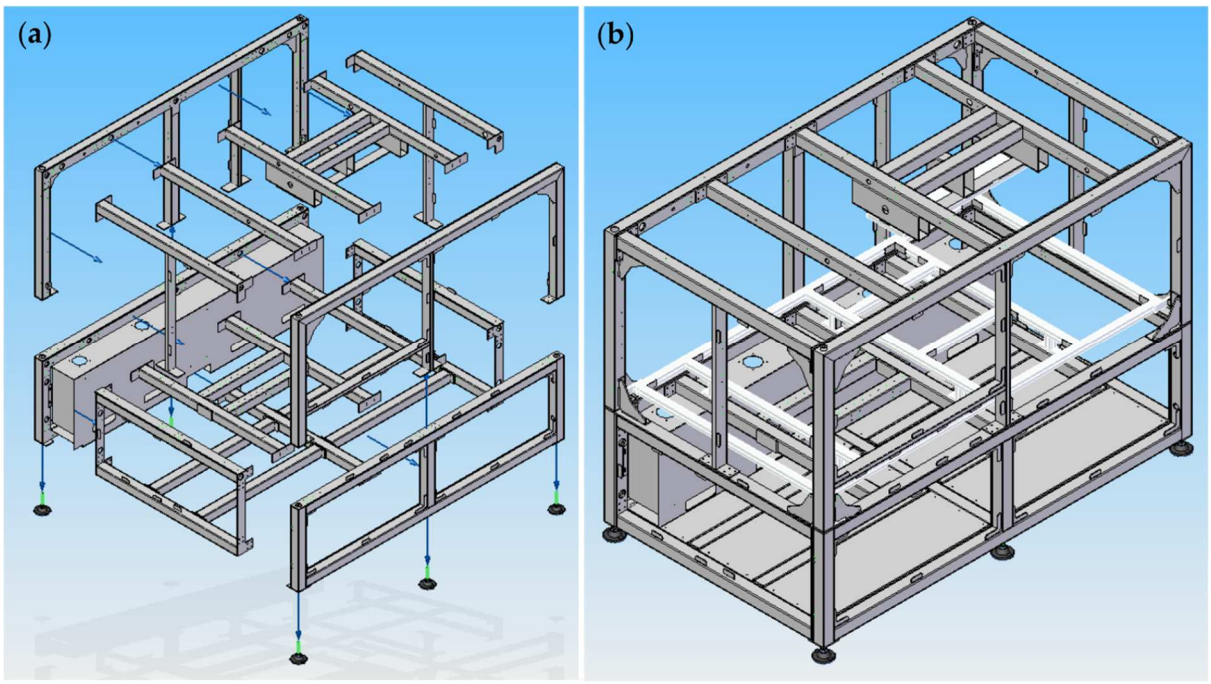


Figure A.3 (**a**) Welded parts to be mounted to constitute the frame. (**b**) additional elements, such as supporting plates in the lower part, the gusset plates, the plane of aluminium profiles and the fixture plate for the *RH1F* robot.

A.2.2 RV4F railway

The railway for the articulated robot *RV4F* is moved by means of a recirculating ball screw (by the company *HIWIN*), connected to a *Mitsubishi Electric* 1.5 kW brushless motor coupled with a reductor with a reduction ratio of 16. Figure A.4 shows the CAD assembly (**a**) and (**b**) and two pictures (**c**) and (**d**) of the railway. The railway has a total mechanical stroke of 540 mm.

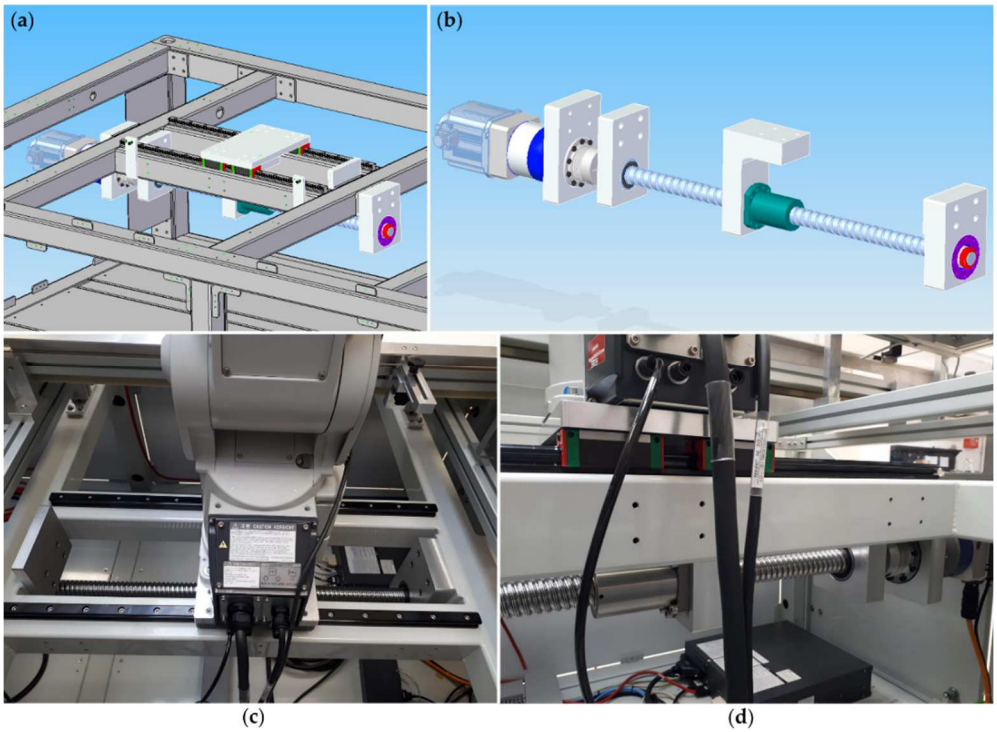


Figure A.4 (**a**) CAD railway composed of all its elements; CAD (**b**) detail of the recirculating ball screw. (**c**), (**d**): picture of the railway.

A.2.3 Conveyor belt A

The *Conveyor belt A* (CAD assembly shown in Figure A.5) can transport objects from one end to the cell to the other (cf. Figure 2.1), goes through the robots' workspace, so that they can pick objects from it, and is monitored by three *Omron FH-SCX* cameras. It is driven by a *Mitsubishi Electric* 400 W brushless motor coupled with a reductor with a reduction ratio of 7. The belt fabric (blue in Figure A.5 for visualization purpose) is made of polyurethane, which confers it high friction, useful to prevent unwanted slips of the objects due to rapid variations of the conveyor belt speed. At the belt sides, it has two barriers that prevent objects from falling, one of which is adjustable and allows to reduce the disposable width. At its beginning and end, it is endowed with polycarbonate tunnels to prevent the intrusion of limbs into the interior dangerous parts of the robotic cell.

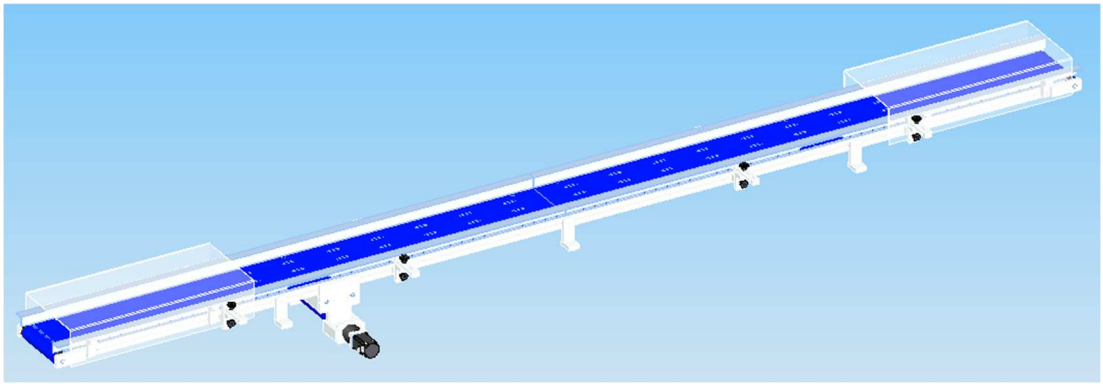


Figure A.5 CAD assembly of the *Conveyor belt A*.

A.2.4 Rotary table

The rotary table is shown in Figure A.6 (two CAD assembly views and a picture). The assembly is composed of a mechanism to adjust the rotary table height. This can be done by manually turning a knob, the rotation of which is transmitted to a trapezoidal screw (shown in green) which makes a nut translate. The nut is connected in a fixed way to the rotary table plate, which moves accordingly. The rotation of the table is driven by a *Mitsubishi Electric* 400 W brushless motor coupled with a reductor with reduction ratio of 50, which transmits the motion to the rotary table by means of a belt-pulleys coupling, with a reduction gear of roughly 2.

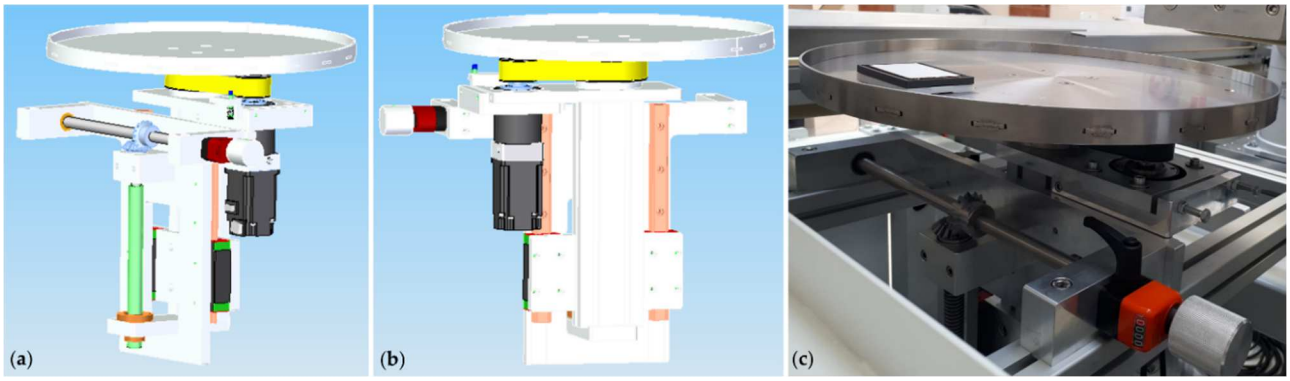


Figure A.6 (**a**), (**b**) Two different views of the CAD assembly of the rotary table. Both its driving mechanism and height regulation mechanism can be seen. (**c**): picture of the rotary table.

A.2.5 Conveyor belt B

The *Conveyor belt B*, shown in Figure A.7, has the task of transporting objects to the rotary table. It is equipped with a mechanism to manually adjust its height that works with the same principle of the one of the rotary table. Like the *Conveyor belt A*, it is driven by a *Mitsubishi Electric* 400 W brushless motor coupled with a reductor with a reduction ratio of 7. Also, its belt fabric is polyurethane, and it is endowed with one adjustable barrier and a polycarbonate tunnel, this latter for safety purposes.

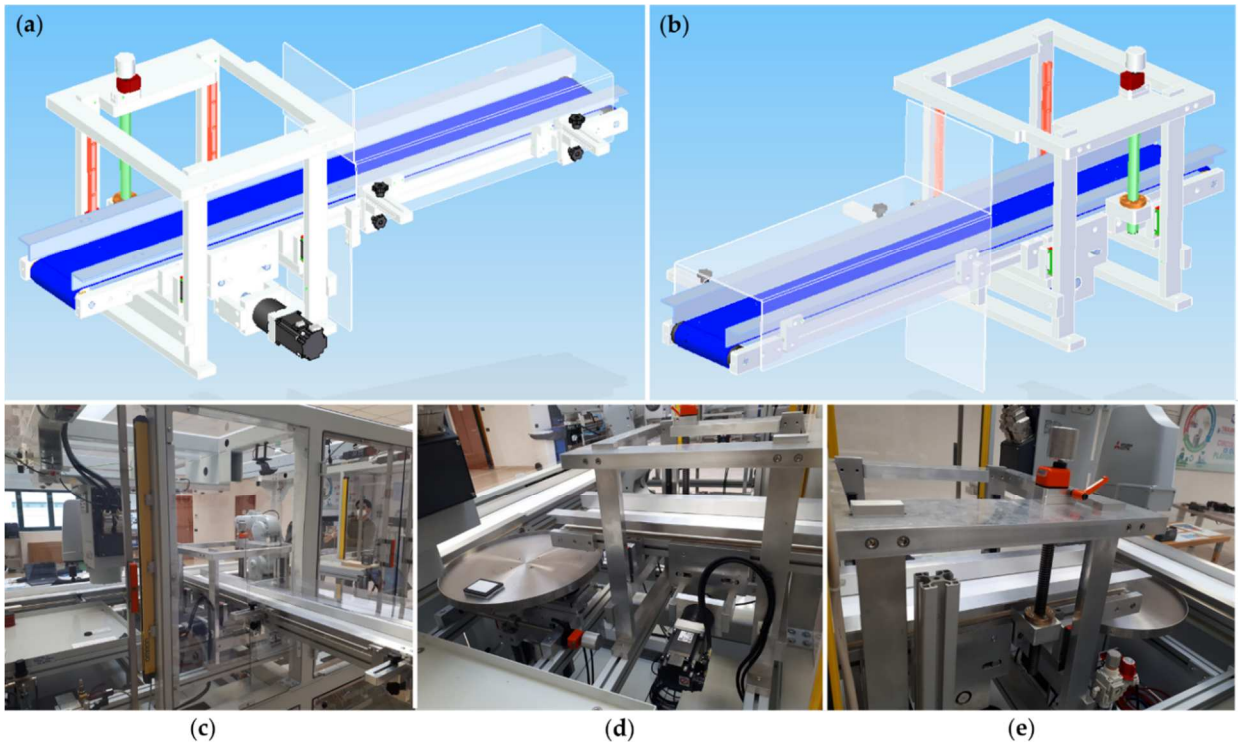


Figure A.7 (**a**), (**b**) Two different views of the CAD assembly of the *Conveyor belt B*. Both its driving mechanism and height regulation mechanism can be seen. (**c**), (**d**), (**e**) Three pictures of the *Conveyor Belt B*.

A.2.6 Carters

Figure A.8 shows the robotic cell's both fixed and mobile carters. The rear-upper part of the cell is endowed with three movable guards, whereas other movable guards are present on each of the upper part of the short sides of the cell. This way, the access to the various devices is granted, allowing regulations, modifications and maintenance, other than to deposit workpiece on the worktables. Each movable guard is equipped with security switches and safety light curtains. The rear lower part is reserved to the electrical panel. The carters are manufactured in polycarbonate.

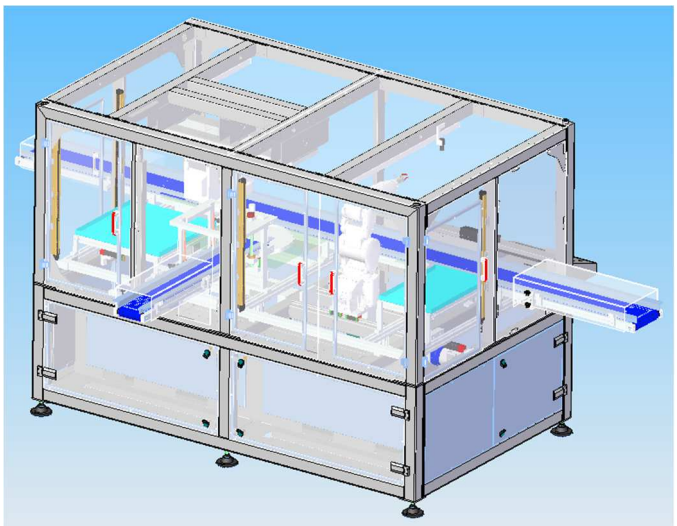


Figure A.8 Rear view of the robotic cell, complete of all its components.

A.3 Wiring and piping design

A high-level conceptual scheme of the wiring and piping is shown in Figure A.9. The various mechanical drives other than the robots (railway, conveyor belts, rotary table) are connected and configured as robot additional axes (each robot can manage up to two additional axes). More precisely, the railway and the *Conveyor belt A*, are configured as *RV4F* additional axes, whereas the rotary table and the *Conveyor belt B* are configured as *RH1F* additional axes. This configuration (possible if using brushless motors) has the advantage of granting both higher versatility and ease of programming of the working cycle. In fact, the motions of the additional axes can be directly programmed inside the *Mitsubishi Electric* proprietary robot programming software *RTToolbox3*. The programming is normally carried out on an external computer connected to the robotic cell ethernet switch, which allows to transfer the robot programs to and from the robots' controller. By means of the *Graphic Operator Terminal* (GOT), also connected to the ethernet switch, it is possible to start/stop the working cycle, regulate the cycle speed and perform other operations, while directly monitoring the cycle (it is attached to the very front left of the robotic cell, at the left of the two robot teaching pendants, cf. Figure 2.1a).
For additional monitoring purposes, in the rear of the cell a *Raspberry Pi 4*, connected to the ethernet switch, is used to output visualization data on a screen. Furthermore, robots are connected to their teaching pendants, which enable robot jogging. The pneumatic circuits of the scheme of Figure A.9 derives from the necessity of managing the robots' pneumatic grippers, whose closing/opening is commanded by means of electrovalves (also known as solenoid valves). The device that controls the various signals is a modular PLC, which manage the communications between the miscellaneous devices inside the cell, both in terms of inputs/outputs and in terms of more complex data types. The various vision sensors are connected in different ways, some of them to the ethernet switch of the robotic cell, whereas some of them (the ones that come with USB connectors) are directly connected to the external computer.

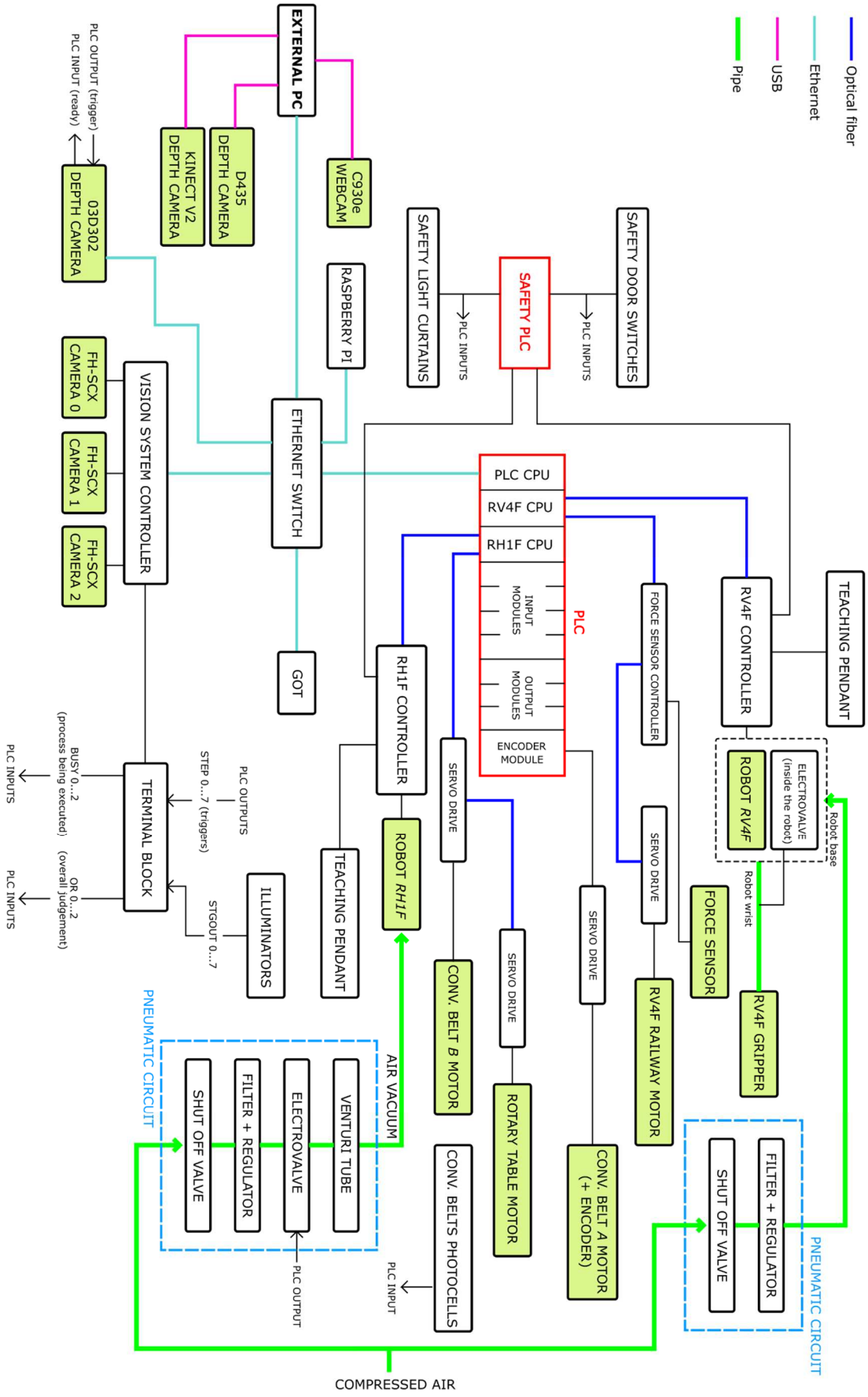


Figure A.9 Conceptual scheme of the wiring and piping of the robotic cell.

Appendix B. ROS architecture

One of the primary purposes of the use of *ROS* was the need of an easy and efficient way to handle multithreading, given the various processes involved running at different frame rates. *ROS* makes use of *nodes* and *topics*, represented in Figure B.1. Nodes are processes that run independently at their own frequency and can communicate with other nodes by means of topics. A node can *publish* messages to a topic; these messages can be retrieved by other nodes that *subscribe* to that topic.

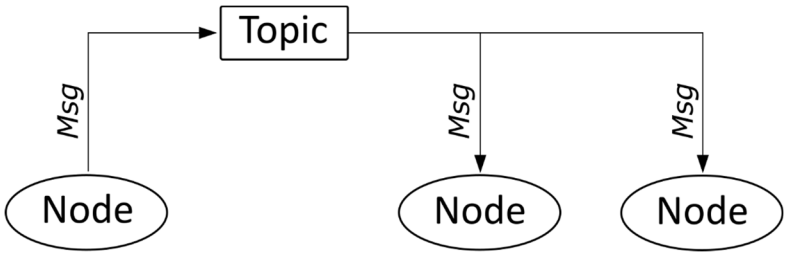


Figure B.1 ROS nodes exchanging messages (*Msg*s) by means of a topic.

The implemented *ROS* architecture is shown in Figure B.2. The various nodes of the *Realsense* and the *Kinect*, encapsulated in coloured blocks, come from the already available packets [130] and [131], respectively. In the actual implemented architecture, the online trajectory generation is handled by the node *generate_traj*, whereas the obstacle tracking part by the other various nodes.

The *ROS* architecture also comprehends a hardware interface, having the function of communicating with the robot controller (in a nutshell, packing command data in a specific format and sending them to a UDP socket). The hardware interface was implemented by modifying the already available *ROS* code [132], written for a similar *Mitsubishi Electric* robot controller.

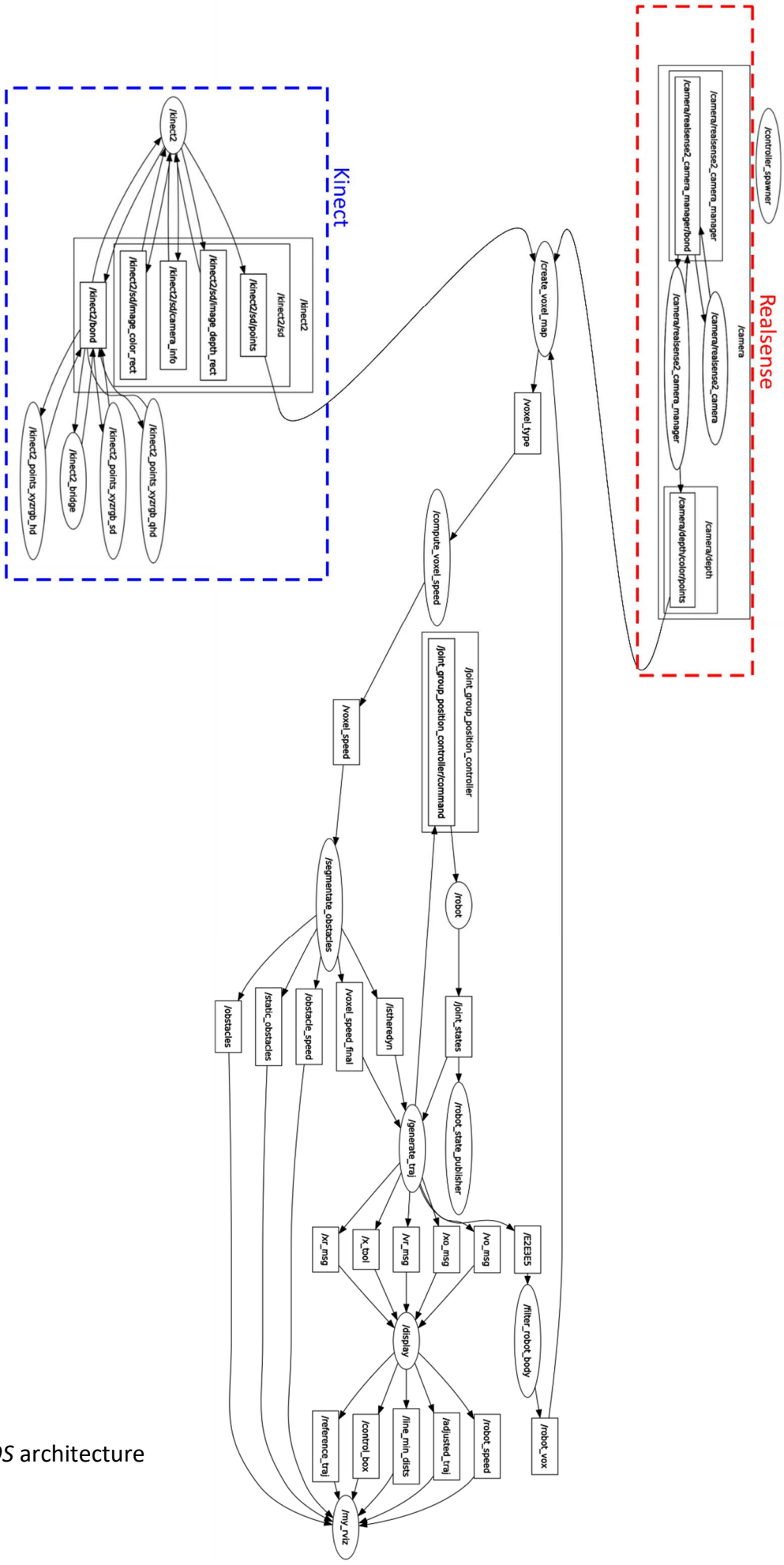


Figure B.2 *ROS* architecture

Appendix C. Basic notions on camera calibration

The term *calibration* can refer to either *intrinsic calibration* or *extrinsic calibration*. Figure C.1 highlights the difference and shows the main elements involved, here reported:

Optical Axis Z_c

Optical centre O

Image plane: plane perpendicular to the optical axis, located at a distance f (focal length) from it.

Principal point C: intersection between the optical axis and the image plane.

Three reference frames are involved:

Camera RF (X_c, Y_c, Z_c): 3D reference frame with origin on the optical centre and optical axis as *z*-axis.

Image RF (x, y): 2D reference frame located on the image plane, with origin in C and *x* and *y* axis orientated as the one of *Camera RF*.

Global RF (X, Y, Z): generic 3D reference frame with a pose different from *Camera RF*.

Intrinsic calibration refers to the process of estimating the perspective transformation (which incorporates f and C) that allows one to transform a point P from the camera coordinates to the image coordinates. This relies on the use of the so-called pinhole camera model, that can be added complexity if accounting for example for lens distortions. For RGB cameras, intrinsic calibration usually relies on the use of chessboards patterns, whose images, acquired in different poses, are then processed by specific algorithms that estimates the camera intrinsic parameters.

Extrinsic calibration refers to the process of estimating the rigid transformation between the reference frame of the sensor considered and an external global reference frame. Extrinsic calibration can be done in different ways, depending on the sensor type, features of the scene, needed accuracy, position of the global reference frame and so on. A common case is the estimation of the transformation between the *Camera RF*s of two RGB cameras. In this case the extrinsic calibration can once again

rely on the use of a chessboard pattern, acquired by both cameras in different poses, and related processing algorithms.

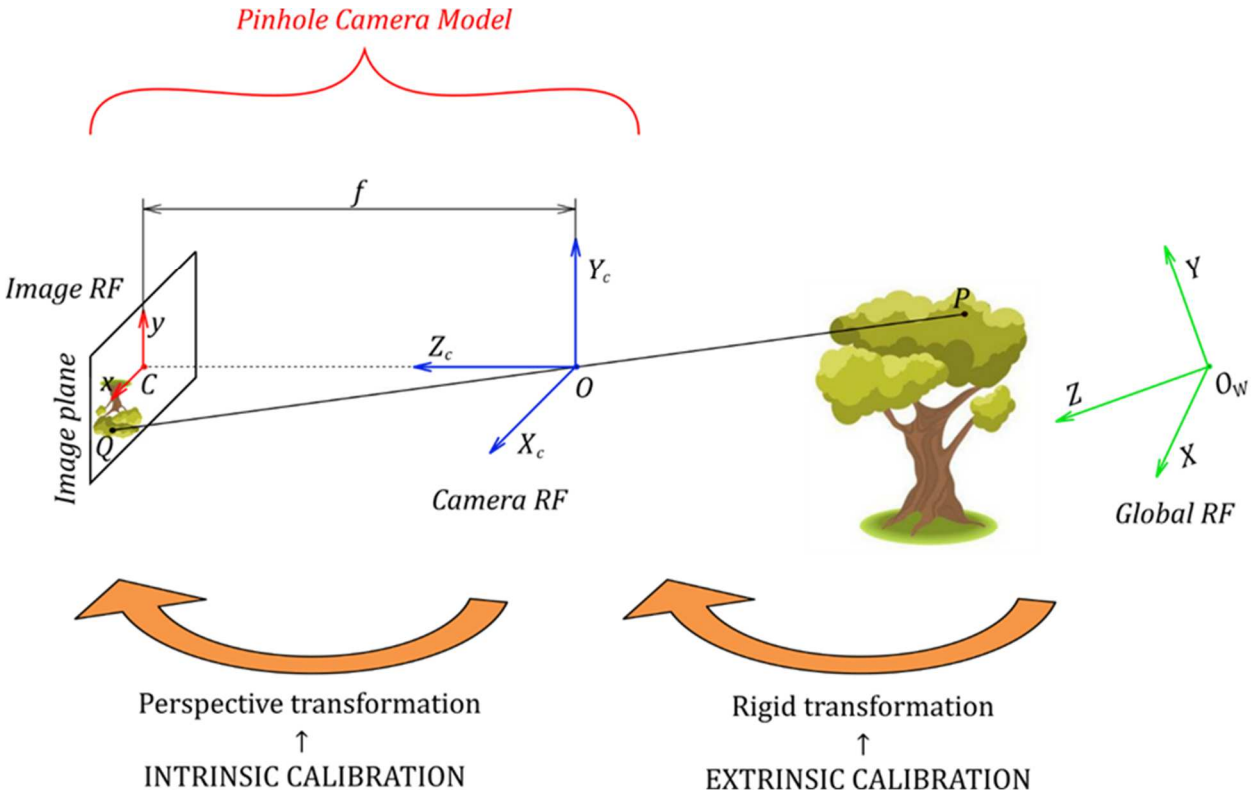


Figure C.1 Main elements involved in intrinsic and extrinsic calibration.

Appendix D. GPU implementation of the particle filter

Here some insights about the GPU-parallelized implementation of the particle filter are given. It is to be considered, however, that for the sake of conciseness only the main steps are presented and in a simplified way.
The GPU implementation exploited the functionalities of the *NVIDIA CUDA* (*Compute Unified Device Architecture*) parallel computing platform. In particular, other than some of the *CUDA Toolkit basic functionalities*, explained in the book [133], the following CUDA libraries were used:

- *cuRAND*: random number generation library, which contains efficient generation of high-quality *pseudorandom* and *quasirandom* numbers;
- *Thrust*: provides a rich collection of data parallel primitives, such as containers and parallelized versions of common algorithms.

1 – Creation and initialization of GPU particle arrays

The particle-related quantities are created on the GPU as seven large memory arrays, each of fixed length equal to the maximum number of total particles $n_p = Q_{max} * N_{vox}$ (= 96*37156 = 3566892 particles) and initialized with zeros. Let us denote with the index k the k^{th} cell of a GPU array (k = 1, …, n_p). A representation on how the particles are stored on the GPU is shown in Figure D.1.

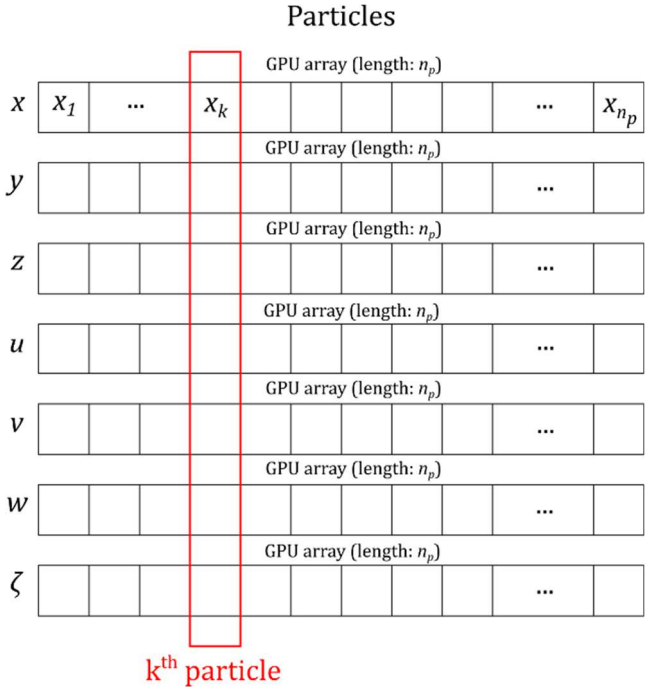


Figure D.1 Representation of the GPU memory arrays used to store the particle-related quantities.

In the *initialization stage* of the particle filter, a number of particles equal to Q_{max} times the number of *type3-voxels* is generated and stored in a first fraction of the GPU arrays, leaving the other cells to the initialization value (zero). This is done by exploiting the CUDA library *cuRAND*.

2 – Particle evolution

Each particle is evolved according to Eq. (3.70). The evolution is parallelized over the total number of particles.

3 – Particle selection

1 – For each particle, the index *j* of the voxel where it belongs is computed (if after the evolution stage it ended up inside the control volume) and stored in the GPU array *par_vox*. This is parallelized over the total number of particles.

2 – If the particle ended up outside the control volume or not inside a *type3-voxel*, the corresponding cell of x is set to zero. It is worthy to note that zero can be used as a value to mark a particle as "eliminated" since the probability of a particle having x exactly equal to zero (accounting also the machine precision) is nearly zero, and in the case it happens simply the particle is eliminated, not causing unexpected behaviours. To evaluate if the particle belongs to a *type3-voxel*, a GPU array version of the vector *vox_type* is used.

Figure D.2 shows the GPU arrays and the manipulations involved in this stage.

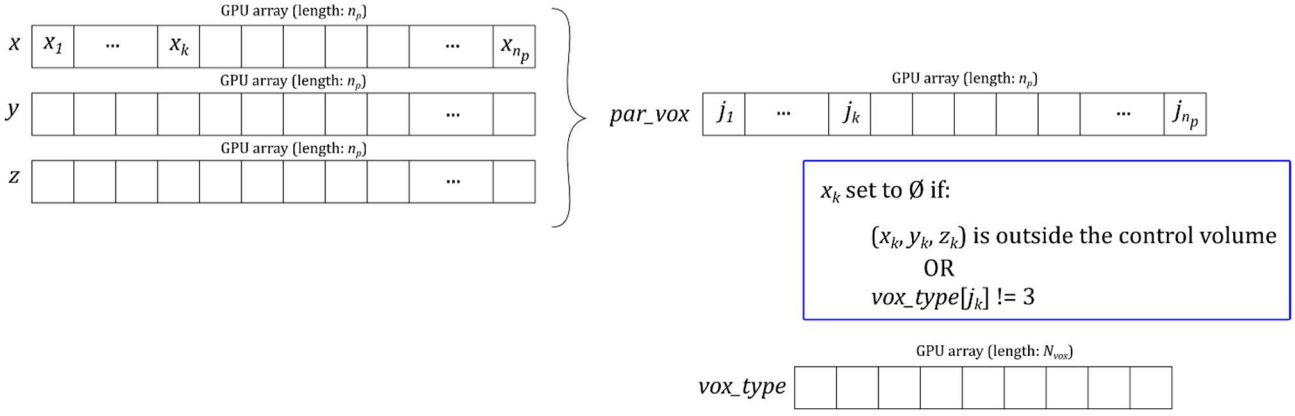


Figure D.2 *Particle selection* methodology.

4 – Particle reorganization

In this stage, summarized in Figure D.3, all the particle-related GPU arrays are reorganized, using the functions of the CUDA library *Thrust*. More specifically, the x array is partitioned so that all the non-zero elements are collected into a first part and all the zero element into a second part. y, z, u, v, w, ζ, *par_vox* are then reordered

according to the same index permutation used for the x partition. Then, the first n_{ps} (number of survived particles) particles are reordered so that the corresponding voxel index is in ascending order, and, inside the groups of particles with the same voxel index, the particles are sorted in ascending order of ζ. This was done in an efficient way by exploiting two consecutive *stable sort* algorithms operating on the first part of the arrays, containing only the survived particles.

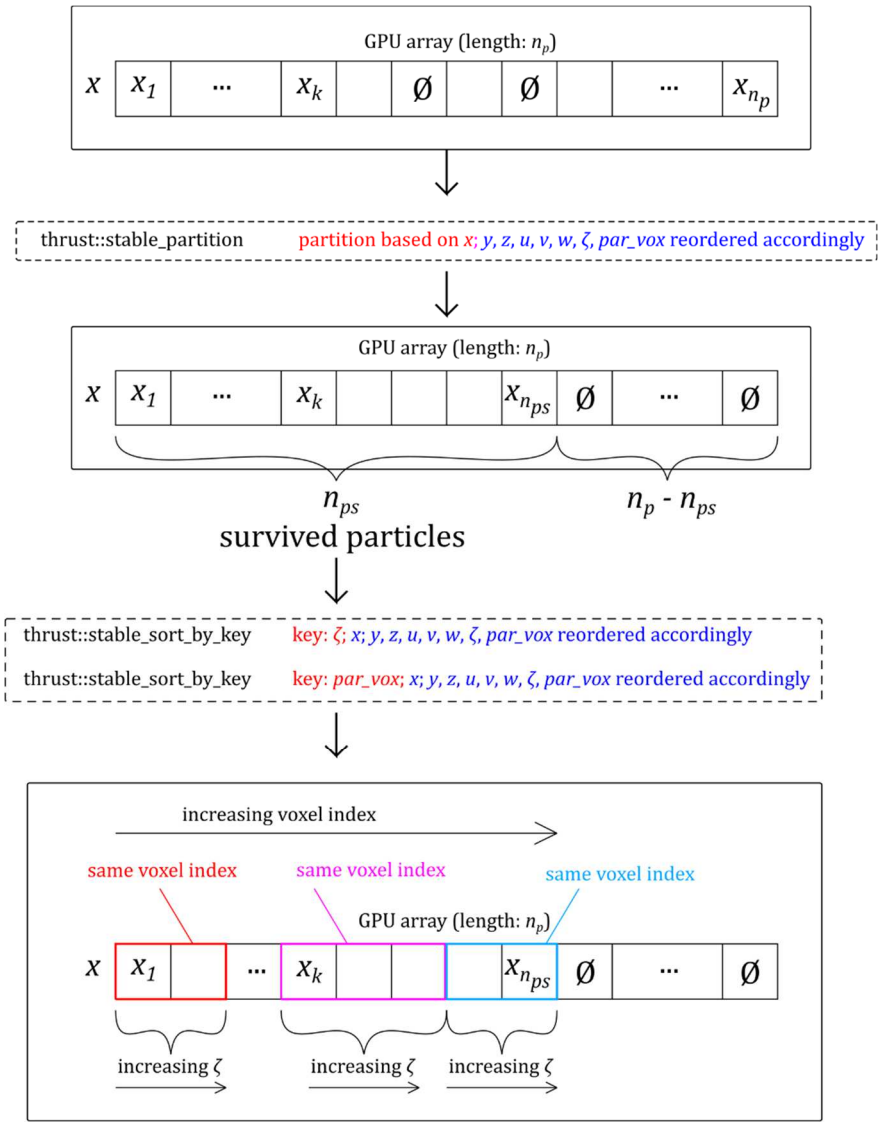


Figure D.3 *Particle reorganization* steps.

5 – Counting of the number of particles associated to each voxel and computation of the voxel speed

The voxel speeds are stored in a GPU array of length N_{vox}. The current number of particles contained in each voxel is stored in another GPU array, of length N_{vox}, named Q, used in the next stage. Some passage needed for the construction of Q and for computing the velocity to associate to each voxel (cf. Eq. (3.71)), exploited the CUDA function *atomicAdd*.

6 – Resampling

By exploiting the GPU arrays Q, *vox_type* and the scalar Q_{max} (number of max particles per voxel), it is possible to fill the GPU arrays Q_{add}, and Q_{rmv}, containing the number of particles to be added or removed in each voxel. These latter, in turn, can be used to fill the GPU arrays ind_{add} and ind_{rmv}, which contains the indices of x (and others particle-related arrays) where to start to add or remove (set to zero) block of particles (whose dimensions are contained in Q_{add} and Q_{rmv}). ind_{add} and ind_{rmv} were computed by exploiting the *exclusive_scan* function of the Thrust library: the k^{th} cell of ind_{add} (or ind_{rmv}) is obtained by summing the previous k-1 cells of Q_{add} (or Q_{rmv}). The particle addition consists in overwriting part of the memory region containing zeros on the second part of the GPU particle-related arrays. Their organized structure, combined with use of Q_{add}, ind_{add} and Q_{rmv}, ind_{rmv} allows to fully parallelize this stage, which was the main scope of the *reorganization stage*. The generation of the new particles relies on the CUDA library *cuRAND*. The main passages are summarized in Figure D.4 and Figure D.5.

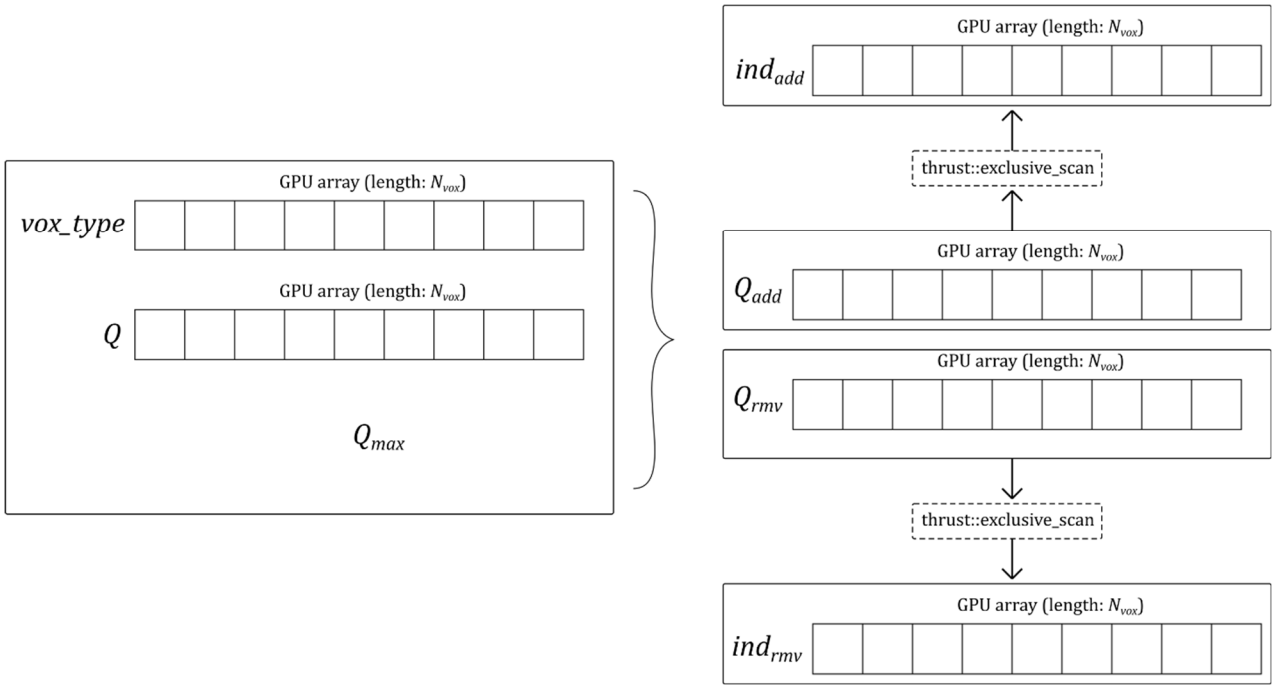


Figure D.4 Various GPU arrays involved in the *resampling stage*.

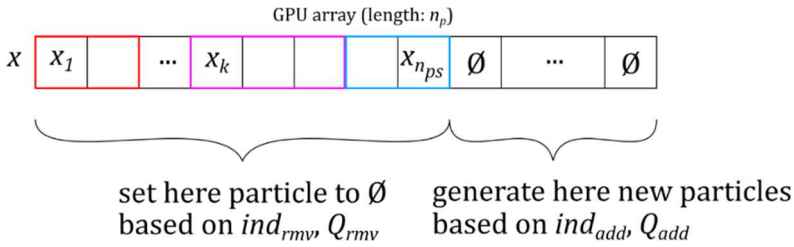


Figure D.5 Particle remotion and generation, carried out during the *resampling stage*.

One last consideration concerns the fact that, when adding particles, it is theoretically possible that the particle-related arrays overflow, because the particles set to zero in this stage are kept in the first part of the array, and no new particles can be added there. In practice, however, this is extremely unlikely, since the dynamic obstacles occupy only a relatively small fraction of the entire voxel grid and n_p demonstrated to be a sufficient length in all the experimental tests conducted. In any case, the actual implementation accounts for a possible overflow and takes specific measures in occurrence. The consideration that dynamic obstacles typically occupy a small fraction of the voxel grid can be viewed as a cue to implement strategies to reduce the GPU memory usage, considering for example shorter arrays to store the particle-related quantities or variable-size arrays.

References

[1] International Federation of Robotics (https://ifr.org/).

[2] ISO 8373:2012 Robots and robotic devices – Vocabulary. ISO; 2012.

[3] ISO 10218-2:2011 Robots and robotic devices — Safety requirements for industrial robots — Part 2: Robot systems and integration. ISO; 2011.

[4] ISO/TS 15066:2016 Collaborative robots. ISO; 2016.

[5] Villani, V.; Pini, F.; Leali, F.; Secchi, C. Survey on Human–Robot Collaboration in Industrial Settings: Safety, Intuitive Interfaces and Applications. *Mechatronics* **2018**, *55*, 248–266. https://doi.org/10.1016/j.mechatronics.2018.02.009.

[6] IEC 60204-1:2016 Safety of machinery – electrical equipment of machines – Part 1: general requirements. IEC; 2016.

[7] AIRSKIN, https://www.airskin.io/ (accessed 13/05/2022)

[8] Wasenmüller, O.; Stricker, D. Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision. In *Computer Vision – ACCV 2016 Workshops*; Chen, C.-S., Lu, J., Ma, K.-K., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, 2017; Vol. 10117, pp 34–45. https://doi.org/10.1007/978-3-319-54427-4_3.

[9] Bellandi, P.; Docchio, F.; Sansoni, G. Roboscan: A Combined 2D and 3D Vision System for Improved Speed and Flexibility in Pick-and-Place Operation. *Int J Adv Manuf Technol* **2013**, *69* (5–8), 1873–1886. https://doi.org/10.1007/s00170-013-5138-z.

[10] Liu, M.-Y.; Tuzel, O.; Veeraraghavan, A.; Taguchi, Y.; Marks, T. K.; Chellappa, R. Fast Object Localization and Pose Estimation in Heavy Clutter for Robotic Bin Picking. *The International Journal of Robotics Research* 2012, *31* (8), 951–973. https://doi.org/10.1177/0278364911436018.

[11] Nerakae, P.; Uangpairoj, P.; Chamniprasart, K. Using Machine Vision for Flexible Automatic Assembly System. *Procedia Computer Science* **2016**, *96*, 428–435. https://doi.org/10.1016/j.procs.2016.08.090.

[12] Chaumette, F.; Hutchinson, S. Visual Servo Control. I. Basic Approaches. *IEEE Robot. Automat. Mag.* **2006**, *13* (4), 82–90. https://doi.org/10.1109/MRA.2006.250573.

[13] Edinbarough, I.; Balderas, R.; Bose, S. A Vision and Robot Based On-Line Inspection Monitoring System for Electronic Manufacturing. *Computers in Industry* **2005**, *56* (8–9), 986–996. https://doi.org/10.1016/j.compind.2005.05.022.

[14] Flacco, F.; Kroger, T.; De Luca, A.; Khatib, O. A Depth Space Approach to Human-Robot Collision Avoidance. In *2012 IEEE International Conference on Robotics and Automation*; IEEE: Saint Paul, MN, 2012; pp 338–345. https://doi.org/10.1109/ICRA.2012.6225245.

[15] Ragaglia, M.; Zanchettin, A. M.; Rocco, P. Trajectory Generation Algorithm for Safe Human-Robot Collaboration Based on Multiple Depth Sensor Measurements. *Mechatronics* **2018**, *55*, 267–281. https://doi.org/10.1016/j.mechatronics.2017.12.009.

[16] Rybski, P.; Anderson-Sprecher, P.; Huber, D.; Niessl, C.; Simmons, R. Sensor Fusion for Human Safety in Industrial Workcells. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: Vilamoura-Algarve, Portugal, 2012; pp 3612–3619. https://doi.org/10.1109/IROS.2012.6386034.

[17] Frese, C.; Fetzner, A.; Frey, C. Multi-Sensor Obstacle Tracking for Safe Human-Robot Interaction. *ISR/Robotik 2014; 41st International Symposium on Robotics*, Munich, Germany, 2014, pp. 1-8.

[18] Magnanimo, V.; Walther, S.; Tecchia, L.; Natale, C.; Guhl, T. Safeguarding a Mobile Manipulator Using Dynamic Safety Fields. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; IEEE: Daejeon, South Korea, 2016; pp 2972–2977. https://doi.org/10.1109/IROS.2016.7759460.

[19] Pellegrinelli, S.; Orlandini, A.; Pedrocchi, N.; Umbrico, A.; Tolio, T. Motion Planning and Scheduling for Human and Industrial-Robot Collaboration. *CIRP Annals* **2017**, *66* (1), 1–4. https://doi.org/10.1016/j.cirp.2017.04.095.

[20] Cheng, Y.; Sun, L.; Liu, C.; Tomizuka, M. Towards Efficient Human-Robot Collaboration With Robust Plan Recognition and Trajectory Prediction. *IEEE Robot. Autom. Lett.* **2020**, *5* (2), 2602–2609. https://doi.org/10.1109/LRA.2020.2972874.

[21] Ferreira, M.; Costa, P.; Rocha, L.; Moreira, A. P. Stereo-Based Real-Time 6-DoF Work Tool Tracking for Robot Programing by Demonstration. *Int J Adv Manuf Technol* **2016**, *85* (1–4), 57–69. https://doi.org/10.1007/s00170-014-6026-x.

[22] Hamabe, T.; Goto, H.; Miura, J. A Programming by Demonstration System for Human-Robot Collaborative Assembly Tasks. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*; IEEE: Zhuhai, 2015; pp 1195–1201. https://doi.org/10.1109/ROBIO.2015.7418934.

[23] Koert, D.; Maeda, G.; Lioutikov, R.; Neumann, G.; Peters, J. Demonstration Based Trajectory Optimization for Generalizable Robot Motions. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*; IEEE: Cancun, Mexico, 2016; pp 515–522. https://doi.org/10.1109/HUMANOIDS.2016.7803324.

[24] El Makrini, I.; Elprama, S. A.; Van den Bergh, J.; Vanderborght, B.; Knevels, A.-J.; Jewell, C. I. C.; Stals, F.; De Coppel, G.; Ravyse, I.; Potargent, J.; Berte, J.; Diericx, B.; Waegeman, T.; Jacobs, A. Working with Walt: How a Cobot Was Developed and Inserted on an Auto Assembly Line. *IEEE Robot. Automat. Mag.* **2018**, *25* (2), 51–58. https://doi.org/10.1109/MRA.2018.2815947.

[25] Magrini, E.; Ferraguti, F.; Ronga, A. J.; Pini, F.; De Luca, A.; Leali, F. Human-Robot Coexistence and Interaction in Open Industrial Cells. *Robotics and Computer-Integrated Manufacturing* **2020**, *61*, 101846. https://doi.org/10.1016/j.rcim.2019.101846.

[26] OMRON, Vision System, FH Series, https://www.ia.omron.com/products/family/3210/ (accessed 13/05/2022)

[27] Vision Doctor, solutions for industrial illumination: illumination techniques, https://www.vision-doctor.com/en/illumination-techniques.html (accessed 13/05/2022).

[28] Tateno, K.; Tombari, F.; Navab, N. When 2.5D Is Not Enough: Simultaneous Reconstruction, Segmentation and Recognition on Dense SLAM. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 2295–2302, doi:10.1109/ICRA.2016.7487378.

[29] Giancola, S.; Valenti, M.; Sala, R. *A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies*; SpringerBriefs in Computer Science; Springer International Publishing: Cham, 2018. https://doi.org/10.1007/978-3-319-91761-0.

[30] Schauwecker, K. Real-Time Stereo Vision on FPGAs with SceneScan. *arXiv:1809.07977 [cs]* **2018**.

[31] Michalos, G.; Makris, S.; Spiliotopoulos, J.; Misios, I.; Tsarouchi, P.; Chryssolouris, G. ROBO-PARTNER: Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future. *Procedia CIRP* **2014**, *23*, 71–76. https://doi.org/10.1016/j.procir.2014.10.079.

[32] Yanco, H. A.; Drury, J. Classifying Human-Robot Interaction: An Updated Taxonomy. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*; IEEE: The Hague, Netherlands, 2004; Vol. 3, pp 2841–2846. https://doi.org/10.1109/ICSMC.2004.1400763.

[33] Bdiwi, M.; Pfeifer, M.; Sterzing, A. A New Strategy for Ensuring Human Safety during Various Levels of Interaction with Industrial Robots. *CIRP Annals* **2017**, *66* (1), 453–456. https://doi.org/10.1016/j.cirp.2017.04.009.

[34] Matthias, B; Kock, S; Jerregard, H; Kallman, M; Lundberg, I; Mellander, R. Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept. *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, 2011, pp. 1-6, doi: 10.1109/ISAM.2011.5942307.

[35] De Luca, A.; Flacco, F. 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: Rome, Italy, 2012; pp 288–295. https://doi.org/10.1109/BioRob.2012.6290917.

[36] Bauer, A.; Wollherr, D.; Buss, M. Human–Robot Collaboration: a survey. *Int. J. Human. Robot.* **2008**, *05* (01), 47–66. https://doi.org/10.1142/S0219843608001303.

[37] Geravand, M.; Flacco, F.; De Luca, A. Human-Robot Physical Interaction and Collaboration Using an Industrial Robot with a Closed Control Architecture. In *2013 IEEE International Conference on Robotics and Automation*; IEEE: Karlsruhe, Germany, 2013; pp 4000–4007. https://doi.org/10.1109/ICRA.2013.6631141.

[38] Matthias, B. Example Application of ISO/TS 15066 to a Collaborative Assembly Scenario. 6. In *Proceedings of ISR 2016: 47th International Symposium on Robotics, 2016*; Munich, Germany, 2016; pp 1–5.

[39] Zacharaki, A.; Kostavelis, I.; Gasteratos, A.; Dokas, I. Safety Bounds in Human Robot Interaction: A Survey. *Safety Science* **2020**, *127*, 104667. https://doi.org/10.1016/j.ssci.2020.104667.

[40] Jacob, R. J. K.; Girouard, A.; Hirshfield, L. M.; Horn, M. S.; Shaer, O.; Solovey, E. T.; Zigelbaum, J. Reality-Based Interaction: A Framework for Post-WIMP Interfaces. In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*; ACM Press: Florence, Italy, 2008; p 201. https://doi.org/10.1145/1357054.1357089.

[41] Hornecker, E.; Buur, J. Getting a Grip on Tangible Interaction: A Framework on Physical Space and Social Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*; ACM: Montréal Québec Canada, 2006; pp 437–446. https://doi.org/10.1145/1124772.1124838.

[42] Jaju, A.; Das, A. P.; Pal, P. K. Evaluation of Motion Mappings from a Haptic Device to an Industrial Robot for Effective Master–Slave Manipulation. *International Journal of Robotics and Automation* **2013**, *28* (1). https://doi.org/10.2316/Journal.206.2013.1.206-3657.

[43] Du, G.; Zhang, P.; Liu, X. Markerless Human–Manipulator Interface Using Leap Motion With Interval Kalman Filter and Improved Particle Filter. *IEEE Trans. Ind. Inf.* **2016**, *12* (2), 694–704. https://doi.org/10.1109/TII.2016.2526674.

[44] Kulić, D.; Croft, E. Pre-Collision Safety Strategies for Human-Robot Interaction. *Auton Robot* **2007**, *22* (2), 149–164. https://doi.org/10.1007/s10514-006-9009-4.

[45] Norberto Pires, J. Robot-by-voice: Experiments on Commanding an Industrial Robot Using the Human Voice. *Industrial Robot* **2005**, *32* (6), 505–511. https://doi.org/10.1108/01439910510629244.

[46] Silaghi, H.; Rohde, U.; Spoiala, V.; Silaghi, A.; Gergely, E.; Nagy, Z. Voice Command of an Industrial Robot in a Noisy Environment. In *2014 International Symposium on Fundamentals of Electrical Engineering (ISFEE)*; IEEE: Bucharest, Romania, 2014; pp 1–5. https://doi.org/10.1109/ISFEE.2014.7050596.

[47] Ong, S. K.; Chong, J. W. S.; Nee, A. Y. C. Methodologies for Immersive Robot Programming in an Augmented Reality Environment. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia - GRAPHITE '06*; ACM Press: Kuala Lumpur, Malaysia, 2006; p 237. https://doi.org/10.1145/1174429.1174470.

[48] Michalos, G.; Karagiannis, P.; Makris, S.; Tokçalar, Ö.; Chryssolouris, G. Augmented Reality (AR) Applications for Supporting Human-Robot Interactive Cooperation. *Procedia CIRP* **2016**, *41*, 370–375. https://doi.org/10.1016/j.procir.2015.12.005.

[49] Arai, T.; Kato, R.; Fujita, M. Assessment of Operator Stress Induced by Robot Collaboration in Assembly. *CIRP Annals* **2010**, *59* (1), 5–8. https://doi.org/10.1016/j.cirp.2010.03.043.

[50] Robla-Gomez, S.; Becerra, V. M.; Llata, J. R.; Gonzalez-Sarabia, E.; Torre-Ferrero, C.; Perez-Oria, J. Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments. *IEEE Access* **2017**, *5*, 26754–26773. https://doi.org/10.1109/ACCESS.2017.2773127.

[51] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotic research* **1986**, 5 (1), 396-404.

[52] Tsuji, T.; Kaneko, M. Noncontact Impedance Control for Redundant Manipulators. *IEEE Trans. Syst., Man, Cybern. A* **1999**, *29* (2), 184–193. https://doi.org/10.1109/3468.747853.

[53] Kulić, D.; Croft, E. Real-time safety for human-robot interaction. *Robot. Auto. Syst.* **2006**, 54 (1), pp. 1-12.

[54] Lacevic, B.; Rocco, P. Kinetostatic Danger Field - a Novel Safety Assessment for Human-Robot Interaction. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: Taipei, 2010; pp 2169–2174. https://doi.org/10.1109/IROS.2010.5649124.

[55] Buizza Avanzini, G.; Ceriani, N. M.; Zanchettin, A. M.; Rocco, P.; Bascetta, L. Safety Control of Industrial Robots Based on a Distributed Distance Sensor. *IEEE Trans. Contr. Syst. Technol.* **2014**, *22* (6), 2127–2140. https://doi.org/10.1109/TCST.2014.2300696.

[56] Maiocchi, M.; Zanchettin, A. M.; Rocco, P. Physical and Perceived Safety in Human-Robot collaboration. In *Proceedings of IRIM 2020: 2nd Italian Conference on Robotics and Intelligent Machines*; Pisa, Italia, 2020; pp 147-148.

[57] Byner, C.; Matthias, B.; Ding, H. Dynamic Speed and Separation Monitoring for Collaborative Robot Applications – Concepts and Performance. *Robotics and Computer-Integrated Manufacturing* **2019**, *58*, 239–252. https://doi.org/10.1016/j.rcim.2018.11.002.

[58] Ferraguti, F.; Bertuletti, M.; Landi, C. T.; Bonfe, M.; Fantuzzi, C.; Secchi, C. A Control Barrier Function Approach for Maximizing Performance While Fulfilling to ISO/TS 15066 Regulations. *IEEE Robot. Autom. Lett.* **2020**, *5* (4), 5921–5928. https://doi.org/10.1109/LRA.2020.3010494.

[59] Ferraguti, F.; Talignani Landi, C.; Costi, S.; Bonfè, M.; Farsoni, S.; Secchi, C.; Fantuzzi, C. Safety Barrier Functions and Multi-Camera Tracking for Human–Robot Shared Environment. *Robotics and Autonomous Systems* **2020**, *124*, 103388. https://doi.org/10.1016/j.robot.2019.103388.

[60] ISO 13855:2010 Safety of machinery — Positioning of safeguards with respect to the approach speeds of parts of the human body. ISO; 2010.

[61] Marvel, J. A.; Norcross, R. Implementing Speed and Separation Monitoring in Collaborative Robot Workcells. *Robotics and Computer-Integrated Manufacturing* **2017**, *44*, 144–155. https://doi.org/10.1016/j.rcim.2016.08.001.

[62] Leso, M.; Zilkova, J.; Vacek, M. Robotic Manipulator with Optical Safety System. In *2015 International Conference on Electrical Drives and Power Electronics (EDPE)*; IEEE: Tatranska Lomnica, Slovakia, 2015; pp 389–393. https://doi.org/10.1109/EDPE.2015.7325326.

[63] Vogel, C.; Walter, C.; Elkmann, N. Safeguarding and Supporting Future Human-Robot Cooperative Manufacturing Processes by a Projection- and Camera-Based Technology. *Procedia Manufacturing* **2017**, *11*, 39–46. https://doi.org/10.1016/j.promfg.2017.07.127.

[64] Corrales, J. A.; Candelas, F. A.; Torres, F. Safe Human–Robot Interaction Based on Dynamic Sphere-Swept Line Bounding Volumes. *Robotics and Computer-Integrated Manufacturing* **2011**, *27* (1), 177–185. https://doi.org/10.1016/j.rcim.2010.07.005.

[65] Safety mat *PSENmat* by Pilz:https://www.automationinside.com/article/the-pressure-sensitive-safety-mat-psenmat-from-pilz-combines-safe-area-monitoring-with-plant-and-machine-operation-virtual-control (accessed 13/05/2022).

[66] Bosh, APAS Assistant, https://apps.boschrexroth.com/microsites/apas/ (accessed 13/05/2022).

[67] Halme, R.-J.; Lanz, M.; Kämäräinen, J.; Pieters, R.; Latokartano, J.; Hietanen, A. Review of Vision-Based Safety Systems for Human-Robot Collaboration. *Procedia CIRP* **2018**, *72*, 111–116. https://doi.org/10.1016/j.procir.2018.03.043.

[68] Feng Duan; Tan, J.; Arai, T. Using Motion Capture Data to Regenerate Operator's Motions in a Simulator in Real Time. In *2008 IEEE International Conference on Robotics and Biomimetics*; IEEE: Bangkok, 2009; pp 102–107. https://doi.org/10.1109/ROBIO.2009.4912987.

[69] Wang, P.; Liu, H.; Wang, L.; Gao, R. X. Deep Learning-Based Human Motion Recognition for Predictive Context-Aware Human-Robot Collaboration. *CIRP Annals* **2018**, *67* (1), 17–20. https://doi.org/10.1016/j.cirp.2018.04.066.

[70] Himmelsbach, U. B.; Wendt, T. M.; Lai, M. Towards Safe Speed and Separation Monitoring in Human-Robot Collaboration with 3D-Time-of-Flight Cameras. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*; IEEE: Laguna Hills, CA, 2018; pp 197–200. https://doi.org/10.1109/IRC.2018.00042.

[71] Long, P.; Chevallereau, C.; Chablat, D.; Girin, A. An Industrial Security System for Human-Robot Coexistence. *Industrial Robot* **2018**, *45* (2), 220–226. https://doi.org/10.1108/IR-09-2017-0165.

[72] https://www.pilz.com/en-INT/products/sensor-technology/safe-camera-systems (accessed 13/05/2022).

[73] Salmi, T.; Marstio, I.; Malm, T; Montonen, J. Advanced Safety Solutions for Human-Robot-Cooperation. Proceedings of the forty seventh international symposium robotics (ISR). VDE; **2016**, pp. 1-6

[74] Pedrocchi, N.; Vicentini, F.; Matteo, M.; Tosatti, L. M. Safe Human-Robot Cooperation in an Industrial Environment. *International Journal of Advanced Robotic Systems* **2013**, *10* (1), 27. https://doi.org/10.5772/53939.

[75] Smart Robots, http://smartrobots.it/product/ (accessed 13/05/2022).

[76] Lenz, C.; Grimm, M.; Roder, T.; Knoll, A. Fusing Multiple Kinects to Survey Shared Human-Robot-Workspaces. Technical report. Technische Universität München. **2012**.

[77] Mainprice, J.; Berenson, D. Human-Robot Collaborative Manipulation Planning Using Early Prediction of Human Motion. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: Tokyo, 2013; pp 299–306. https://doi.org/10.1109/IROS.2013.6696368.

[78] Cherubini, A.; Passama, R.; Meline, A.; Crosnier, A.; Fraisse, P. Multimodal Control for Human-Robot Cooperation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: Tokyo, 2013; pp 2202–2207. https://doi.org/10.1109/IROS.2013.6696664.

[79] Vicentini, F.; Pedrocchi, N.; Giussani, M.; Tosatti, L. M. Dynamic Safety in Collaborative Robot Workspaces through a Net- Work of Devices Fulfilling Functional Safety Requirements. **2014**, 8.

[80] Vicentini, F.; Giussani, M.; Tosatti, L. M. Trajectory-Dependent Safe Distances in Human-Robot Interaction. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*; IEEE: Barcelona, Spain, 2014; pp 1–4. https://doi.org/10.1109/ETFA.2014.7005316.

[81] Lasota, P. A.; Rossano, G. F.; Shah, J. A. Toward Safe Close-Proximity Human-Robot Interaction with Standard Industrial Robots. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*; IEEE: Taipei, 2014; pp 339–344. https://doi.org/10.1109/CoASE.2014.6899348.

[82] Ragaglia, M.; Bascetta, L.; Rocco, P. Multiple Camera Human Detection and Tracking inside a Robotic Cell - An Approach Based on Image War, Computer Vision, K-d Trees and Particle Filtering: In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics*; SCITEPRESS - Science and and Technology Publications: Vienna, Austria, 2014; pp 374–381. https://doi.org/10.5220/0005045703740381.

[83] Morato, C.; Kaipa, K. N.; Zhao, B.; Gupta, S. K. Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-Time Human Tracking. *Journal of Computing and Information Science in Engineering* **2014**, *14* (1), 011006. https://doi.org/10.1115/1.4025810.

[84] Dumonteil, G.; Manfredi, G.; Devy, M.; Confetti, A.; Sidobre, D. Reactive Planning on a Collaborative Robot for Industrial Applications: In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*; SCITEPRESS - Science and and Technology Publications: Colmar, Alsace, France, 2015; pp 450–457. https://doi.org/10.5220/0005575804500457.

[85] Flacco, F.; Kroeger, T.; De Luca, A.; Khatib, O. A Depth Space Approach for Evaluating Distance to Objects: With Application to Human-Robot Collision Avoidance. *J Intell Robot Syst* **2015**, *80* (S1), 7–22. https://doi.org/10.1007/s10846-014-0146-2.

[86] Zanchettin, A. M.; Ceriani, N. M.; Rocco, P.; Ding, H.; Matthias, B. Safety in Human-Robot Collaborative Manufacturing Environments: Metrics and Control. *IEEE Trans. Automat. Sci. Eng.* **2016**, *13* (2), 882–893. https://doi.org/10.1109/TASE.2015.2412256.

[87] Mohammed, A.; Schmidt, B.; Wang, L. Active Collision Avoidance for Human–Robot Collaboration Driven by Vision Sensors. *International Journal of Computer Integrated Manufacturing* **2017**, *30* (9), 970–980. https://doi.org/10.1080/0951192X.2016.1268269.

[88] Mead, R.; Matarić, M. J. Autonomous Human–Robot Proxemics: Socially Aware Navigation Based on Interaction Potential. *Auton Robot* **2017**, *41* (5), 1189–1201. https://doi.org/10.1007/s10514-016-9572-2.

[89] Wendt, T.M.; Himmelsbach, U. B.; Lai, M.; Waβmer, M; Time of flight cameras enabling collaborative robots for improving safety in medical applications. In *International Journal of Interdisciplinary Telecommunications and Networking* **2017**, *9* (4), pp 10-17. https://doi.org/10.4018/IJITN.2017100102.

[90] Fabrizio, F.; De Luca, A. Real-Time Computation of Distance to Dynamic Obstacles With Multiple Depth Sensors. *IEEE Robot. Autom. Lett.* **2017**, *2* (1), 56–63. https://doi.org/10.1109/LRA.2016.2535859.

[91] Himmelsbach, U. B.; Wendt, T. M.; Hangst, N.; Gawron, P. Single Pixel Time-of-Flight Sensors for Object Detection and Self-Detection in Three-Sectional Single-Arm Robot Manipulators. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*; IEEE: Naples, Italy, 2019; pp 250–253. https://doi.org/10.1109/IRC.2019.00046.

[92] Folscher, D. J.; Kruger, K. Saving Time on Robot Programming: Programming by Demonstration Using Stereoscopic Motion Capturing. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*; IEEE: Stellenbosch, South Africa, 2016; pp 1–6. https://doi.org/10.1109/RoboMech.2016.7813133.

[93] Comau Racer-5-0.80 COBOT, https://www.comau.com/it/competencies/robotics-automation/collaborative-robotics/racer-5-0-80-cobot/ (accessed 13/05/2022).

[94] Mitsubishi Electric manual "Mitsubishi Industrial Robot – CR750/CR751 series controller – Ethernet Function Instruction Manual".

[95] Mitsubishi Electric manual "Mitsubishi Industrial Robot – CR750-Q/CR751-Q Controller - RV-4/7/13/20FM-Q-SE Series - Standard Specifications Manual".

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

[97] Hayes, M. J. D.; Husty, M. L.; Zsombor-Murray, P. J. Singular Configurations of Wrist-Partitioned 6R Serial Robots: a Geometric Perspective for Users. *Transactions of the Canadian Society for Mechanical Engineering* **2002**, *26* (1), 41–55. https://doi.org/10.1139/tcsme-2002-0003.

[98] Fritsch, F. N.; Carlson, R. E. Monotone Piecewise Cubic Interpolation. In *SIAM Journal on Numerical Analysis* **1980**, Vol. 17, pp 238–246.

[99] Nocedal, J.; Wright, S. J. *Numerical Optimization*, 2nd ed.; Springer series in operations research; Springer: New York, 2006.

[100] Ferreau, H. J.; Kirches, C.; Potschka A.; Bock, H.G.; Diehl, M. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation* **2014**, 6 (4), pp 327—363.

[101] Ferreau, H. J.; Bock, H. G.; Diehl, M. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control* **2008**, 18 (8), pp 816—830**.**

[102] Ayoub, Raymond G. Paolo Ruffini's Contributions to the Quintic. *Archive for History of Exact Sciences* **1980**, 22 (3), pp 253–277. https://doi.org/10.1007/BF00357046.

[103] https://github.com/jwezorek/quintic (accessed 13/05/2022).

[104] Doyle, P.; McMullen, C. Solving the Quintic by Iteration. *Acta Math.* **1989**, *163* (0), 151–180. https://doi.org/10.1007/BF02392735.

[105] Flacco, F.; De Luca, A. Multiple Depth/Presence Sensors: Integration and Optimal Placement for Human/Robot Coexistence. In *2010 IEEE International Conference on Robotics and Automation*; IEEE: Anchorage, AK, 2010; pp 3916–3923. https://doi.org/10.1109/ROBOT.2010.5509125.

[106] Hanoun, S.; Bhatti, A.; Creighton, D.; Nahavandi, S.; Crothers, P.; Esparza, C. G. Target Coverage in Camera Networks for Manufacturing Workplaces. *J Intell Manuf* **2016**, *27* (6), 1221–1235. https://doi.org/10.1007/s10845-014-0946-z.

[107] Garrido-Jurado, S.; Muñoz-Salinas, R.; Madrid-Cuevas, F. J.; Medina-Carnicer, R. Generation of Fiducial Marker Dictionaries Using Mixed Integer Linear Programming. *Pattern Recognition* **2016**, *51*, 481–491. https://doi.org/10.1016/j.patcog.2015.09.023.

[108] Romero-Ramirez, F. J.; Muñoz-Salinas, R.; Medina-Carnicer, R. Speeded up Detection of Squared Fiducial Markers. *Image and Vision Computing* **2018**, *76*, 38–47. https://doi.org/10.1016/j.imavis.2018.05.004.

[109] Besl, P. J.; B.; McKay, N. D. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1992**, 14 (2), pp 239 – 256.

[110] CloudCompare, 3D point cloud and mesh processing software, https://www.danielgm.net/cc/ (accessed 13/05/2022).

[111] Hermann, A.; Bauer, J.; Klemm, S.; Dillmann, R. Mobile Manipulation Planning Optimized for GPGPU Voxel- Collision Detection in High Resolution Live 3D-Maps. In *ISR/Robotik 2014; 41st International Symposium on Robotics ISR/Robotik,* 2014, Munich, Germany, pp 1–8.

[112] Hermann, A.; Drews, F.; Bauer, J.; Klemm, S.; Roennau, A.; Dillmann, R. Unified GPU Voxel Collision Detection for Mobile Manipulation Planning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*; IEEE: Chicago, IL, USA, 2014; pp 4154–4160. https://doi.org/10.1109/IROS.2014.6943148.

[113] NVIDIA, CUDA C++ Best Practices Guide, 2021, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html (accessed 09/01/2022).

[114] Morales, N.; Toledo, J.; Acosta, L.; Medina, J. S. A combined voxel and particle filter-based approach for fast obstacle detection and tracking in automotive applications. *IEEE Transactions on Intelligent Transportation Systems* **2017**, 18 (7), pp 1824 – 1833.

[115] Dirk-Jan Kroon (2022), Polygon2Voxel. (https://www.mathworks.com/matlabcentral/fileexchange/24086-polygon2voxel), MATLAB Central File Exchange. Retrieved January 10, 2022.

[116] https://github.com/peci1/robot_body_filter (accessed 13/05/2022)

[117] https://github.com/blodow/realtime_urdf_filter (accessed 13/05/2022)

[118] Pereira, A.; Althoff, M. Overapproximative Human Arm Occupancy Prediction for Collision Avoidance. *IEEE Trans. Automat. Sci. Eng.* **2018**, *15* (2), 818–831. https://doi.org/10.1109/TASE.2017.2707129.

[119] Naushad Ali, M. M.; Abdullah-Al-Wadud, M.; Lee, S. L. Moving Object Detection and Tracking Using Particle Filter. *AMM* **2013**, *321–324*, 1200–1204. https://doi.org/10.4028/www.scientific.net/AMM.321-324.1200.

[120] Löfberg, J. Oops! I Cannot Do It Again: Testing for Recursive Feasibility in MPC. *Automatica* **2012**, *48* (3), 550–555. https://doi.org/10.1016/j.automatica.2011.12.003.

[121] Olson, E. AprilTag: A Robust and Flexible Visual Fiducial System. In *2011 IEEE International Conference on Robotics and Automation*; IEEE: Shanghai, China, 2011; pp 3400–3407. https://doi.org/10.1109/ICRA.2011.5979561.

[122] Wu, P.-C.; Wang, R.; Kin, K.; Twigg, C.; Han, S.; Yang, M.-H.; Chien, S.-Y. DodecaPen: Accurate 6DoF Tracking of a Passive Stylus. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*; ACM: Québec City QC Canada, 2017; pp 365–374. https://doi.org/10.1145/3126594.3126664.

[123] Coupeté, E.; Moutarde, F.; Manitsaris, S. Multi-Users Online Recognition of Technical Gestures for Natural Human–Robot Collaboration in Manufacturing. *Auton Robot* **2019**, *43* (6), 1309–1325. https://doi.org/10.1007/s10514-018-9704-y.

[124] Jaiganesh, V.; Kumar, J. D.; Girijadevi, J. Automated Guided Vehicle with Robotic Logistics System. *Procedia Engineering* **2014**, *97*, 2011–2021. https://doi.org/10.1016/j.proeng.2014.12.444.

[125] Ceriani, N. M.; Buizza Avanzini, G.; Zanchettin, A. M.; Bascetta L.; Rocco P. Optimal placement of spots in distributed proximity sensors for safe human-robot interaction. In *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 5858-5863.

[126] Kritter, J.; Brévilliers, M.; Lepagnot, J.; Idoumghar, L. On the Optimal Placement of Cameras for Surveillance and the Underlying Set Cover Problem. *Applied Soft Computing* **2019**, *74*, 133–153. https://doi.org/10.1016/j.asoc.2018.10.025.

[127] Gonzalez-Barbosa, J.-J.; Garcia-Ramirez, T.; Salas, J.; Hurtado-Ramos, J.-B.; Rico-Jimenez, J. -d.-J. Optimal Camera Placement for Total Coverage. In *2009 IEEE International Conference on Robotics and Automation*; IEEE: Kobe, 2009; pp 844–848. https://doi.org/10.1109/ROBOT.2009.5152761

[128] Komabashiri, Y.; Mashita, T.; Photchara, R.; Uranishi, Y.; Koike, M.; Maruyama, K. Optimal Arrangement of Surveillance Cameras Using Space Division and a Genetic Algorithm. In *Proceedings of the 25th International Conference on Intelligent User Interfaces Companion*; ACM: Cagliari Italy, 2020; pp 99–100. https://doi.org/10.1145/3379336.3381488.

[129] Campi, M. C.; Garatti, S.; Prandini, M. The Scenario Approach for Systems and Control Design. *Annual Reviews in Control* **2009**, *33* (2), 149–157. https://doi.org/10.1016/j.arcontrol.2009.07.001.

[130] https://github.com/IntelRealSense/realsense-ros (accessed 13/05/2022).

[131] https://github.com/paul-shuvo/iai_kinect2_opencv4 (accessed 13/05/2022).

[132] https://github.com/vustormlab/mitsubishi_arm (accessed 13/05/2022).

[133] Sanders, Jason, and Edward Kandrot. 2011. *CUDA by example: an introduction to general-purpose GPU programming*. Upper Saddle River, NJ: Addison-Wesley.