Abstract

Nowadays, human-robot collaboration (HRC) is an important topic in the industrial sector. According to the actual regulations, the robot no longer needs to be isolated in a work cell, but a collaborative workspace in which human operators and robots coexist can be acceptable. Human-robot interaction (HRI) is made possible by proper design of the robot and by using advanced sensors with high accuracy, which are adopted to monitor collaborative operations to ensure the human safety. Goal of this work is to implement a fuzzy inference system, based on the ISO/TS 15066, to correctly compute the minimum protective separation distance and adjust the robot speed by considering different possible situations, with the aim to avoid any collisions between operators and robots trying to minimize cycle time as well.

Riassunto

Al giorno d'oggi, la collaborazione uomo-robot (HRC) è un importante argomento nel settore industriale. Secondo le normative attuali, il robot non ha più bisogno di essere isolato in una cella di lavoro, ma può essere accettabile uno spazio di lavoro collaborativo in cui convivono operatori umani e robot. L'interazione uomo-robot (HRI) è resa possibile dalla corretta progettazione del robot e dall'utilizzo di sensori all'avanguardia con elevata precisione, i quali vengono utilizzati per monitorare le operazioni collaborative per garantire la sicurezza dell'uomo. L'obiettivo di questo lavoro di tesi è di implementare un sistema di inferenza fuzzy, basato sulla ISO/TS 15066, per calcolare correttamente la distanza minima di separazione protettiva e regolare la velocità del robot considerando diverse situazioni possibili, al fine di evitare qualsiasi collisione tra gli operatori e i robot cercando di ridurre al minimo il tempo ciclo.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The work tackles the human-robot collaboration problem by following the line of the current regulations and introducing a new approach to be used in manufacturing industry. The novel method assures human operators safety, without modifying the robot predefined path and defining a safety metric to scale robot trajectory only when indispensable, thus trying to maximize the production time.

This work is carried out in the framework of the *LABOR* (*Lean robotized Assem-Bly and cOntrol of composite aeRostructures*) European project [1], which has the objective to propose novel robotized assembly paradigms of aircraft fuselage panels.

Figure 1.1: LABOR logo.

Until recently, the aerospace industry was still conservative and companies tended to use successful assembly methods that had already been proven to work in the past. Nowadays, many assembly sub-operations try to exploit robotics, e.g., drilling, fastening and sealing tasks. These operations are no longer manually performed by human operators but by industrial robots or by large automated machines dedi-

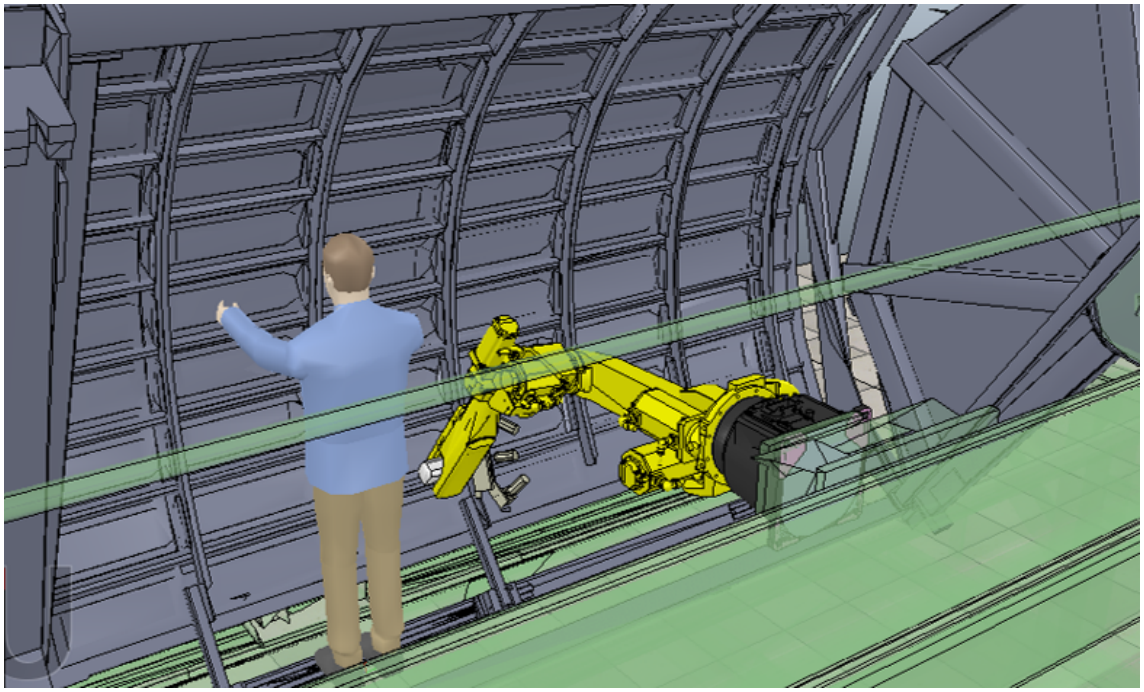


Figure 1.2: Example of a manual assembly operation where the operator shares the work space with a robot.

cated to assembly of specific parts. However, there are some detailed operations which require human capabilities and that must be still executed by operators. This is the case of hybrid metal and composite structures, where, after the drilling operation, some parts have to be manually removed for further manual operations, like deburring, and then re-installed on the skin panel before the sealing and riveting operations, as shown in Figure 1.2.

This requires to setup a robotic cell that has to foresee the presence of a human operator, hence the necessity to monitor the shared workspace.

1.1 Safe Interaction between Human and Robot

Real-time workspace monitoring for human-robot coexistence is not an easy problem to solve. Even more, implementing strategies to maximize the production time and preserve human safety at the same time is a research challenge. The approach proposed here is to adopt a fuzzy inference logic that can update the planned robot velocity in real-time according to robust perception data and a set of rules formulated

based on a risks analysis, explained in detail in Section 1.3. This can lead to a novel, acceptable solution.

Ensuring the safety of a human operator is the main purpose of the current research of industrial collaborative robotics. The workspace sharing involves the continuous human exposure to the presence of machinery. Section 1.2 provides a summary of the minimum and essential elements of reference for the use of collaborative solutions.

1.2 Collaborative Applications Safety

The collaborative applications are made by automated systems in strong interaction with the operator, and therefore fall within the general framework of machine safety regulated by the Machinery Directive 2006/42/EC [2], in force in the version transferred in national legislation D.Lgs. N. 17/2010. The purpose of the Machinery Directive is to protect the operator, the use of machinery conditions, the prevention of accidents at the workplace and the improvement of health conditions (ergonomics, exposure, etc.). The European legislation, through the various directives, aims to establish homogeneous, fair and shared rules for the common market, both outside and inside the European borders. The common rules are designed to safeguard a real competition on the quality of artifacts and systems, starting from minimum levels of guarantee equal for all, clearly extended to non-EU imports.

Essential Health and Safety Requirements (*EHSR*), i.e., a list of objective elements of human safety in the use of machinery, represent a series of characteristics that the machinery must present during its entire life cycle (design, installation, transport, maintenance, etc.) to be considered sure. The specific way to obtain a result of compatibility with the EHSR (CE conformity) is twofold: or it proves punctually with the analytical description of the design and commissioning process of compliance with the EHSR, or the harmonized Technical Standards are adopted. Figure 1.3 shows the relations between Machinery Directive and the harmonized Technical Standards.

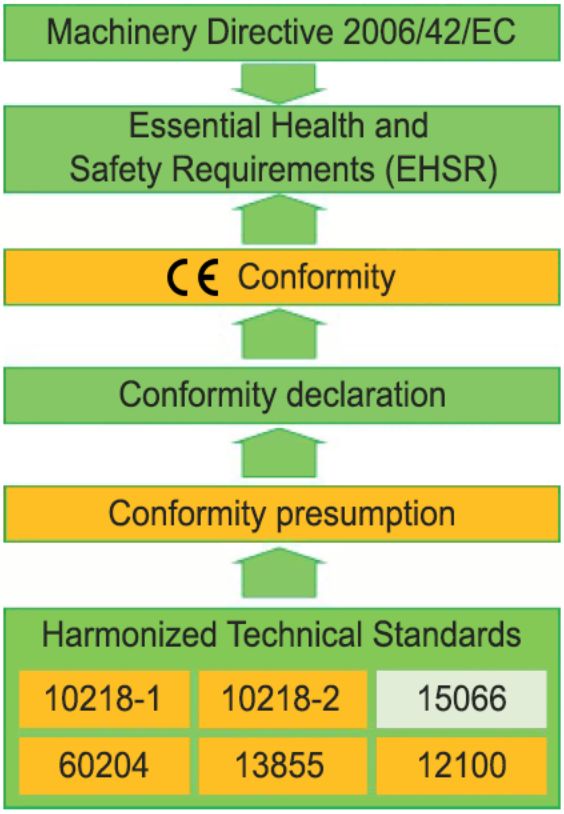


Figure 1.3: Relations in the conformity to the Machinery Directive scheme.

Often, in the case of industrial robotics, and collaborative robotics in particular, the Technical Standard presents a limited number of direct information (e.g., a limit distance, or a maximum speed, etc.). Given the flexibility of the robots and the scale of the possible application results, in fact, the Technical Standards mainly contain general guidelines for compliance with the EHSR. They become, therefore, a design tool, where the experiences and the internationally agreed safety requirements are condensed. The adoption of the Technical Standards of product guarantees to proceed to such Analysis and Evaluation in a complete way.

The harmonized Technical Standards recognized by the European Committee for Standardization (CEN) are divided into three families, based on International Organization for Standardization (ISO):

- Type A standards: basic standards, with general principles and methodologies common to all types of machines. The risk analysis method in ISO 12100 [3] belongs to this type;

- Type B regulations: rules regarding safety aspects related to groups of machines or devices. The general calculation of the safety distances referred to in ISO 10218-2 [4] belongs to this group;
- Type C standards: specific rules for a particular family of machines characterized by homogeneous applications and shapes. They are more limited than type B. ISO 10218 belong to this group. For industrial robotics, including collaborative robotics, the product standards are developed internationally by ISO/TC 299 WG3 [5].

A type C standard, if present and relevant, always takes precedence over the other types: the valid requirements are those formulated according to the product standards for the purpose of the Presumption of Compliance.

The EHSRs are mainly addressed to safety and ergonomics in the use of machinery, therefore they represent the implementation goal of an application. Even if not explained, all aspects related to the perception of robots and to the cognitive/emotional involvement of close interaction should not be neglected during design and use. Although generally valid for automatic machines, EHSRs translate into special implementation requirements for collaborative robots. Table 1.1 shows the most relevant EHSRs in the context of a collaborative experience.

EHSR families	**Description**
1.2.1 Control systems security and reliability of	
1.2.2 Control devices	Group of machinery design requirements (hardware) according to the principles of functional safety. They are to be verified for robots adoption (manipulators, mobile units, etc.), tools and accessory devices, and any other
1.2.3 Start	
1.2.4 Stop	

1.2.5 Command or operating mode selection 1.2.6 Power supply failure	device that is built for the robot system. This group of indications represents the relevant technical pre-requisite for the safety of the machines used in each application.
1.3.3 Risk due to the fall and to the projection of objects 1.3.4 Risks due to surfaces, edges or corners	Group of requirements in common with planning considerations and layout setting. Borders and surfaces, similarly, belong to the sphere of design of the spaces and objects present in the system (e.g., tools and equipment).
1.3.7 Risks due to moving parts	
1.3.8 Choice of protection against the risks arising from moving parts 1.3.9 Risks of uncontrolled movements 1.4.1 General requirements for guards and protection devices	A group of requirements concerning movement and the most immediate and understandable effect of interaction. The drafting of the EHSRs of this group is certainly oriented towards the traditional condition of risk management in automation: separation and adoption of shelters. In this group, the requirements must therefore be interpreted as regards the forms of protection from movements, and intended as "not applicable" with regard to the adoption of physical safeguards.

1.4.2 Special requirements for repair	
1.4.3 Special requirements for protective devices	
1.5.1 Supply of electricity	
1.5.2 Static electricity	
1.5.3 Energy different from that electricity	
1.5.4 Assembly errors	Very heterogeneous group of requirements, which in collaborative applications takes the general notion of various access to machinery. In traditional discipline, operations and risks are usually or often well separable, and operator interventions generally infrequent. In collaborative applications, accesses for changes and maintenance are more common and access to auxiliary systems (air, power, etc.) more exposed. They should therefore not be underestimated and considered in a broader and more complex way.
1.5.15 Risk of slipping, tripping or falling	
1.5.16 Eletric shock	
1.6.1 Machinery maintenance	
1.6.2 Access to operation positions and service points	
1.6.3 Isolation of energy sources	

1.6.4 Operator intervention	

Table 1.1: Most important EHSR families in collaborative robotics.

ISOs 10218-1, 10218-2, and the upcoming ISO Proposed Draft Technical Specification (TS) 15066 are briefly described in Section 1.4, with a particular focus on speed and separation monitoring (SSM) in Section 1.5, since in this work, a strategy to handle the operators safety in industrial SSM scenarios is investigated.

1.3 Risks Analysis

Risk Analysis is strongly proposed by law and practice as the key of the security management of collaborative solutions, throughout the life cycle of applications. The applications are designed to facilitate the operator, changing the way they work: the interaction conditions are varied, each characterized by its own particularity even when using the same type of robot. Even in the variety of applications, it is possible to reduce the conditions of interaction to homogeneous families of cases, but it is nevertheless possible to completely standardize the behaviors and details of a robotic system.

The safety life cycle starts from the birth of the system, from the moment in which the application specifications are conceived and, identified the first hazards, they estimate the risks and prepare the protection measures, considering all the phases of use robots, including programming and installation.

The distinction between "risk" and "danger" is very clear and clear according to the ISO 12100 [3]:

- Damage: physical injury or damage to health;
- Danger: damage potential source;
- Risk: measure of the possibility of transforming the danger into damage.

As regards collaborative applications, he dangers lurk in all situations from which a wound or injury (damage) due to the movements of the robot, to the actions performed by or through the mounted tools, to the interaction with the surrounding environment can result. It is not relevant, however, how much an event involving a robot is dangerous, but how risky it is.

To simplify and make this concept quantifiable, ISO 12100 has defined the following separate concepts of risk:

- Gravity: entity of the damage resulting from the danger; physical injury or damage to health;
- Probability of the damage occurrence: measure of the effective transformation of the danger into reality.

In traditional industrial robotics, this concept of combination of severity and probability has been translated into a high risk estimation by definition. Considering the dangers of a mechanical nature, or any condition of interaction with the machine organs (links or tools) in motion, the most obvious result is an injury because the machine has moving masses and rigid joints that transfer large amounts of energy.

The direct consequence of this standard risk assignment is twofold:

- the only way to reduce the risk is to completely prevent the occurrence, not being able to vary the nature of the intensity of the danger. In terms of regulations, the requirements to be adopted and verified to obtain the required prevention are illustrated by the protection means and by the measures to prevent unexpected movements [4];
- the components of the command functions used to prevent the the occurrence of danger must have a high level of reliability.

The risk estimate in collaborative robotics, on the other hand, recovers the concept of specific analysis of the dangerous event, in order to identify the most appropriate risk reduction measure: in addition to preventing the occurrence, in fact, one can act on the other components, as for example on gravity, reducing it. Increasing the number of variables that can be estimated in the risk assessment, the following scenarios are opened:

- for collaborative robotics applications there are no evaluations prepackaged.

In contrast to Section 5.10 of ISO 10218-2 [4], dedicated to the means of protection (guards, interlocks, muting, etc.), Section 5.11 dedicated to collaborative modalities and, above all, ISO/TS 15066 [6], offers a series of indications of method and verification;

- technical or design solutions that reduce one or more factors risk are multiple and contribute or are alternatives to the same effectiveness of protection. Being able to reduce the gravity of the danger, it is possible to admit a greater probability of occurrence of the danger itself, obtaining the same danger. If the danger of interference with a moving robot is less severe, the possibility of it happing may be tolerated.

Proper analysis of risk limitation requirements is necessary also for the communication of project choices. The assumptions and verifications of the whole analysis must be understood and shared even by those who use them: users and application controllers.

1.4 ISO Analysis

Requirements and guidelines concerning the inherent safe design, measures for protection and information for use of industrial robots are specified in ISO 10218-1 [7]. It provides both the description of basic hazards associated with robots and requirements aimed to eliminate or sufficiently reduce the associated risks. However, in this part of ISO 10128 the robot is not considered as a complete machine. Noise emission is excluded from ISO 10128-1 since it is not a significant hazard for the robot alone. Non-industrial robot are not matter of this part of the ISO but general safety principles from ISO 10128 can be applied to them, too.

ISO 10128-2 [4] provides safety requirements to integrate industrial robot, industrial robots system, as previously specified in ISO 10128-1, and industrial robot cell(s). Design, manufacturing, installation, operations, maintenance, decommissioning as well as the related necessary information, and the component devices of the industrial robot system or cell are included in the integration. This part of ISO 10128 identifies and describes which hazards and hazardous situations are related to

these systems, and gives requirements in order to eliminate or adequately mitigate the connected risks. Hazards associated with processes (e.g., laser radiation, ejected chips, welding smoke) and the respectively requirements are also handled in ISO 10128-2.

ISO/TS 15066 [6] provides requirements concerning safety for collaborative industrial robot systems and the work environment, expanding requirements and guidance on collaborative industrial robot operations specified in ISO 10218-1 and ISO 10218-2. ISO/TS 15066 applies to industrial robot system, as described in both ISO 10218, and address four collaborative scenarios, as shown in Figure 1.4:

1. *Safety-rated monitored stop* (SMS), which requires that the robot stop when a human is in the collaborative workspace;
2. *Hand guiding* (HG), which allows the operator to hand-guide the robot through an hand guiding equipment (e.g., an analog button cell attached to the robot) and an emergency stop conforming to International Electrical Commission (IEC) 60204-1 [8];
3. *Speed and separation monitoring* (SSM), which monitors the robot speed according to the separation distance from the operator;
4. *Power and force limiting* (PFL), which limits the momentum of the robot such that the potential for operator injury upon impact is minimized, according to the established injury standards [9].

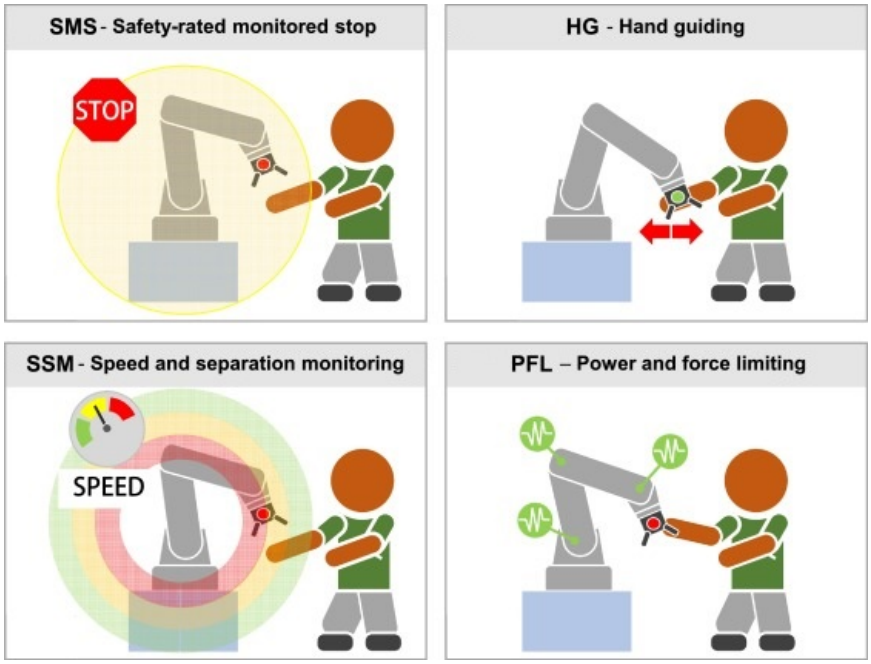


Figure 1.4: ISO/TS 15066 scenarios.

Therefore, the safety collaborative scenarios can be divided into two categories: *post-collision* and *pre-collision* [10]. A *post-collision* system reacts after the physical impact occurs between the robot and the operator. Three main drawbacks can be highlighted: the first one is that a collision could be dangerous if the robot limits are poorly defined; the second one is that any collision will halt task execution, leading to the decrease of the production time; and the last one is that, especially in industrial applications, the robot is equipped with sharp tools, e.g., drilling tools, whose impact can cause serious injury. Human safety can be assured by minimizing the energy transmitted during the contact [11] and by using robots endowed of a compliant structure or sensors for assessing force exchange when the impact occurs, e.g., force or tactile sensors [12].

On the other hand, a *pre-collision* scheme makes use of exteroceptive sensors to detect humans and prevent collisions. Motion capture systems, range sensors or artificial vision systems [13] are crucial in the case of distance monitoring, which is the most suitable approach for pure coexistence in a collaborative workspace. The dangerous zone around the robot is monitored and any operator that accesses it, makes the robot slowing down, until the full stop when the human is too close.

Several researchers proposed different methods for representing humans and robots geometry and implement *pre-collision* strategies: a volumetric representation of the areas occupied by operators and by the robot has been studied in [14] to slow down or stop the robot when these areas overlap; as well as, [15] proposes a potential field method to be used to generate a collision free path, while an advanced approach based on human detection and intention estimation is described in [16]. A further approach is presented in [17] where a safety index is modeled to modify the robot trajectories and preserve the cooperative task. Many of these approaches rely on evasive actions to increase safety. However, in industrial setting, it is generally recommended to follow the robot predefined path without deviating from it. Relative position and velocity of the human operator and the robot can be used to define a safety metric in an industrial setting for trajectory scaling [18].

Always about *pre-collision*, stopping any powered, mechanical system relies on any of three possible procedures described by IEC60204-1 [8]: removing power to the drives, applying brakes, and actively controlling motors to counter motion. The initiation of a stopping function can be triggered via automatic mechanisms internal to the equipment (e.g., software watchdog systems, electronic monitors, or mechanical limit switches), external safeguards, or manual switches (push buttons, pull cords,or pedal-operated switches). According to IEC60204-1, three categories of stop functions based on the removal of power, application of brakes, and control of the equipment during the braking function are specified. These three categories, showen below, are summarized in Table 1.2:

- Stop category 0 (STOP0): the equipment is stopped immediately by removing power to the actuators and applying the brakes. This results in an "uncontrolled stop", in that the purpose is to stop the equipment as quickly as possible, and does not ensure that the motion of actuators follow a prescribed path;
- Stop category 1 (STOP1): the equipment undergoes a controlled stop (i.e., power to the actuators is maintained) until the equipment comes to a complete stop, at which point the power is removed and the brakes applied;
- Stop category 2 (STOP2): the equipment comes to a controlled stop, but the

brakes are not applied nor is the power removed.

Stop category	**Controlled (Yes/No)**	**Power removed (Yes/No)**	**Brakes applied (Yes/No)**
STOP0	No	Yes	Yes
STOP1	Yes	Yes	Yes
STOP2	Yes	No	No

Table 1.2: Stopping categories as defined in IEC60204-1.

In this work, a stop category 2 (STOP2) has been considered.

The main goal for the investigation of a strategy to handle the operators safety in industrial SSM scenarios is reasonably scale down the size of the protective zone around the robot and improve productivity, taking into account safety regulations. The robot behavior is modified, in terms of trajectory scaling, only if there is a real and imminent risk of collision. The operator approach into the collaborative workspace is deeply analyzed to generalize the computing method of the safety index and face the extreme variability and unpredictability of human behaviours.

The devised solution computes the points at minimum distance between the robot and the closest human and presents several desirable features with respect to other solutions cited above; many of those approaches rely on evasive actions to increase safety. However, in industrial setting, it is generally recommended to follow the robot predefined path without deviating from it, especially in complex work cells, where clashes are likely to occur. The main characteristics of the proposed approach are:

- it considers the whole surface of human operators, without skeleton-based techniques and without approximating the body to a single point of mass;
- it considers the whole robot kinematic chain, the entire volume and possible tools, without factoring only a singular representative coordinate of the robot (e.g., the end effector);
- it explicitly takes into account the regulations;

- it allows for the provision that the human speed, v_H, may be estimated directly and it is not assumed constant;
- it considers the relative directions of velocities, which are not factored into the equation proposed by the actual ISOs;
- it does not modify the robot programmed path and it does not require the task to be aborted.

1.5 SSM state-of-the-art

SSM allows the robot system and the operator to move concurrently in the collaborative workspace. Risk reduction is achieved by maintaining at least the protective separation distance, S, between the human operator and robot at all time. During robot motion, the robot system never gets closer to the operator than S. When the Euclidean separation distance, d, is equal or less than S, the robot system stops, before it can impact the operator, as shown in FIgure 1.5. When the operator moves away from the robot system, the robot system can resume the motion automatically while maintaining at least the protective separation distance.

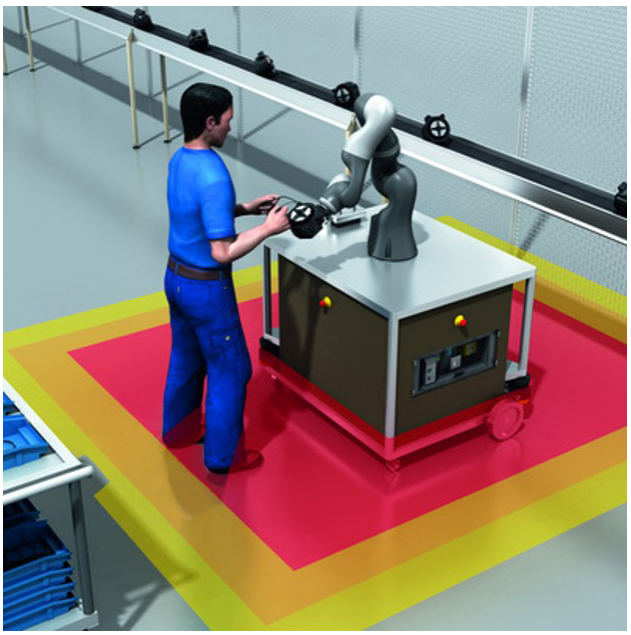


Figure 1.5: Operator in a dangerous (red) zone. The warning zones are in orange and yellow.

ISO 13855 [19] is the first document which investigates about the issue of safeguards positioning for human safety from stationary, active machinery. Specific parameters based on values for approach speeds of parts of the human body are provided in this ISO. The determination of minimum distances to a hazard zone from the detection zone or from actuating devices of safeguards follows a methodology given in ISO 13855. The values of speed for approaches such as walking speed and upper limb movement are time tested and proven in practical experience. In this way is built this International Standard, which gives guidance for typical approaches, except running, jumping or falling. Safeguards taken into consideration in this International Standard are equipments for electro-sensitive or pressure-sensitive protection, two-hand control devices, and interlocking guards without guard locking. The document suggests to compute S as

$$S = vT + C, \tag{1.1}$$

where v is the approach speed of human body parts.

Its value varies according to d

$$d = \begin{cases} 2.0 \text{ m/s} & \text{if} \quad d < 0.50 \text{ m} \\ 1.6 \text{ m/s} & \text{otherwise,} \end{cases} \tag{1.2}$$

where 2.0 m/s is assumed to be as the maximum operator speed. T is the total system stopping performance time, in seconds, and it is a combination of the time required by the machine to respond to the operator's presence (i.e., T_R) and the response time of the machine which brings the robot to a safe, controlled stop (i.e., T_S). C is the intrusion distance safety margin, which represents an additional distance, based on the expected intrusion toward the critical zone prior to the actuation of the protective equipment.

From Equation 1.1, ISO/TS 15066 updates the S meaning by including robot dynamic properties. When the robot system reduces its speed, the protective separation distance decreases correspondingly, i.e.,

$$S(t_0) \geq \int_{\tau=t_0}^{\tau=t_0+T_R+T_S} v_H(\tau)d\tau + \int_{\tau=t_0}^{\tau=t_0+T_R} v_R(\tau)d\tau + \int_{\tau=t_0+T_R}^{\tau=t_0+T_R+T_S} v_S(\tau)d\tau + (C + Z_S + Z_R). \tag{1.3}$$

In Equation 1.3, v_H is the "directed speed" of the closest operator which travels toward the robot, v_R is the speed of the robot in the direction of the operator, v_S is the directed speed of the robot in course of stopping. The remaining terms represents uncertainties: the intrusion distance C is based on the operator reach, Z_R is the robot position uncertainty, and Z_S is the operator position uncertainty (i.e., the sensor uncertainty). Finally, t_0 is considered the current time.

The main issue of ISO 13855 [19] is that the separation distance was initially intended for static machinery, not for dynamic and reconfigurable robotic systems. Therefore, extending what is contained in the standard to the case of industrial robotics is not trivial. Nevertheless, ISO/TS 15066 tries to make a contribution to the HRC problem and describes S using the following linear function

$$S = (v_H T_R + v_H T_S) + (v_R T_R) + (B) + (C + Z_S + Z_R) \tag{1.4}$$

where B is the Euclidean distance travelled by the robot while braking. Note the one-to-one correlation between Equation 1.3 and the linear relationship Equation 1.4. The first term in parentheses describes the contribution attributable to the operator's change in location in the time necessary to bring the robot to a full stop from its current speed. The second term describes the contribution attributable to the robot system reaction time, before it initiates the braking sequence. The third term describes the distance travelled by the robot during its braking. Finally, the fourth term describes the possible distance of intrusion into the robot work volume as a function of the operator reach and the uncertainty of the sensory system and robot kinematics. The values of v_H, T_S, B and C can be found in the safety standards: the values of v_H and C are given in ISO 13855, while guidelines for evaluating T_S and B are given in Annex B of ISO 10218-1 and they result from measurements that

directly depends on the robot system under test. If the stop occurs in category 0, in accordance with IEC 60204-1, the measurements shall be made in the maximum operating conditions envisaged (i.e., maximum speed, maximum load and maximum displacement). For the stop in category 1, the stop time and the distance depend on the speed, the load and the extension, and shall be stated for 33%, 66% and 100% of maximum, unless these values can be derived from the maximum values. In this case, 100% maximum values need to be provided with formula for obtaining intermediate values. Data shall be provided for the three axes of greatest displacement; an example of possible presentation is shown in Figure 1.6.

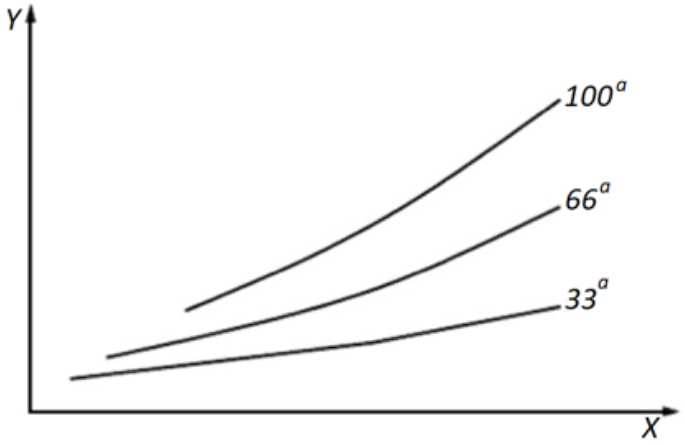


Figure 1.6: Example chart for T_S, illustrating the speed (X axis, [mm/s]) and payload (a, in %) on the stopping time (Y axis, [s]).

About the C value, according to ISO 13855, it is calculated by limiting the direction of approach, relying on the placement of sensors, or requiring a specific control device. These consideration given for C from ISO 13855 are not applicable both for the SSM equation as defined in ISO/TS 15066 (see Equation 1.4) and for the proposed approach of this work. Specifically, because of the dynamic and reconfigurable nature of industrial robots, the direction of approach is ambiguous. Therefore, the C value computation has been carried out in another way, as described in Chapter 7.

This work decomposes and assesses the performance of ISO/TS 15066 SSM minimum protective distance metric and adds a contribution to improve some aspects to allow the applicability in industrial scenarios. In the following Chapters, four main

areas that are directly pertinent to SSM are widely discussed: human detection and tracking, prediction of human and robot motions, safety separation maintenance and robot speed monitoring.

Chapter 2

Hardware and Software

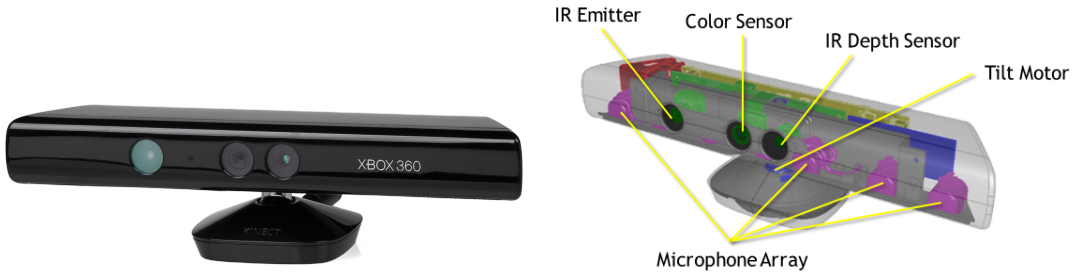
In this chapter, hardwares and softwares used in this work are described. The experimental set-up is composed by two depth cameras, which have been used to monitor the collaborative workspace: a *Microsoft Kinect v1* and an *Intel RealSense D435* (see Figure 3.1(a)). At least two views become necessary to minimize the occlusions of the observed area, as shown in Figure 3.1(b) and Figure 3.1(c). All the code has been developed in *Robot Operating System* (ROS) environment. The operating system used is Ubuntu 16.04.

2.1 Microsoft Kinect v1

The *Microsoft Kinect v1* [20] is a sensor developed by Microsoft for the Xbox 360 console and computer; its function is to achieve high performance 3D image capture, facial recognition and voice recognition, and this is made possible by the software embedded on Kinect, created by Microsoft. The Kinect sensor is USB-powered and is capable of simultaneously tracking up to six people; it is made of:

- a motorized pivot
- a RGB color camera
- a depth sensor
- a microphone
- a set of advanced software to capture motion and gestures

The specifications of the Microsoft Kinect v1 are shown in Table 2.1.



(a) External design (b) Internal structure

Figure 2.1: Microsoft Kinect v1

Feature	**Microsoft Kinect v1**
Color camera	640 × 480 at 30 fps
Depth camera	320 × 240
Max depth distance	∼ 4.5 m
Min depth distance	0.4 m in near mode
Horizontal FoV (Field of View)	57 degrees
Vertical FoV	43 degrees
Skeleton joints defined	20 joints
Full skeletons tracked	2

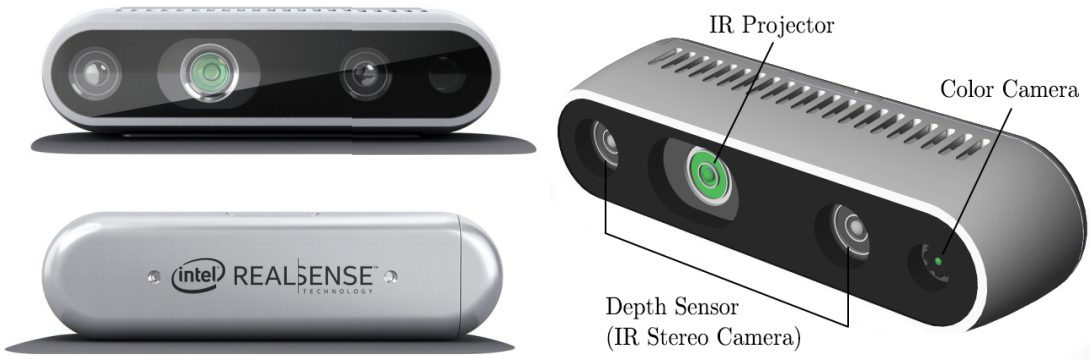
Table 2.1: Microsoft Kinect v1 specifications

2.2 Intel RealSense D435

The *Intel RealSense D435* [21] is an USB-powered depth camera that includes infrared projector and a RGB sensor. It supports the Intel RealSense SDK (Software Development Kit) 2.0, an open source, cross platform development suite including libraries, wrappers, sample code, and tools. The Intel RealSense D435 uses stereo vision to calculate depth; the stereo vision implementation consists of a left imager, right imager, and an optional infrared projector. The infrared projector projects non-visible static IR pattern to improve depth accuracy in scenes with low texture.

The left and right imagers capture the scene and sends raw image data to the vision processor, which calculates depth values for each pixel in the image by correlating points on the left image to the right image, and via shift between a point on the left image and the right image. The depth pixel values are processed to generate a depth frame. Subsequent depth frames create a depth video stream.

The specifications of the Intel RealSense D435 are shown in Table 2.2.



(a) External design (b) Internal structure

Figure 2.2: Intel RealSense D435

Feature	**Intel RealSense D435**
Depth FoV	85.2° × 58° × 94° (+/− 3°)
Depth stream output resolution	Up to 1280 × 720
Depth stream output frame rate	Up to 90 fps
Minimum depth distance	0.2 m
Sensor shutter type	Global shutter
Max range	∼ 10 m
RGB sensor resolution and Frame rate	1920 × 1080 at 30 fps
RGB sensor FoV	69.4° × 42.5° × 77° (+/− 3°)
Camera dimension	90 mm × 25 mm × 25 mm

Table 2.2: Intel RealSense D435 specifications

2.3 ROS

ROS (*Robot Operating System*) [22] is a set of open source software libraries and tools which makes it possible to create applications for robots. ROS allows to control a series of robotic components from a PC. It is issued under the Berkeley Software Distribution (BSD) license, a license that imposes minimal restrictions on the use and redistribution of the software.

Figure 2.3: ROS logo.

ROS is based on the following core concepts:

- *Nodes*: single-purposed executable programs (e.g., sensor driver(s), actuator driver(s), mapper, planner, UI, etc.). Thery are individually compiled, executed and managed, and written using a ROS client library. Nodes can publish or subscribe to a topic and can also provide or use a service.
- *Topics* and *Messages*: topics are names for a stream of messages with a defined type. Nodes communicate with each other by publishing messages to topics. Publishing and subscribing allow a 1-to-N broadcasting. Messages are strictly-typed data structures for inter-node communication.
- *Services*: synchronous inter-node transactions. The service and client model complies with 1-to-1 request-response. Services carry out remote computation and trigger functionality and behavior.
- *ROS master*: provides connection information to nodes so that they can transmit messages to each other. Every node connects to a master at startup to register details of the message streams they publish, and the streams to which that they to subscribe. When a new node appears, the master provides it with the information that it needs to form a direct peer-to-peer connection with other nodes publishing and subscribing to the same message topics.

- *Parameters*: have a hierarchy that matches the namespaces used for topics and nodes. This hierarchy is meant to protect parameter names from colliding. The hierarchical scheme also allows parameters to be accessed individually or as a tree.
- *Stacks* and *Packages*: stacks contain one or more packages, the latter contain nodes and provide a ROS interface.

The ROS version used in this work is the *Kinetic*.

2.3.1 RViz

RViz (*ROS Visualization*) [23] is a 3D visualization tool for ROS. Other than the 3D view, RViz offers a displays list which shows all the loaded displays (e.g., point cloud, robot state).

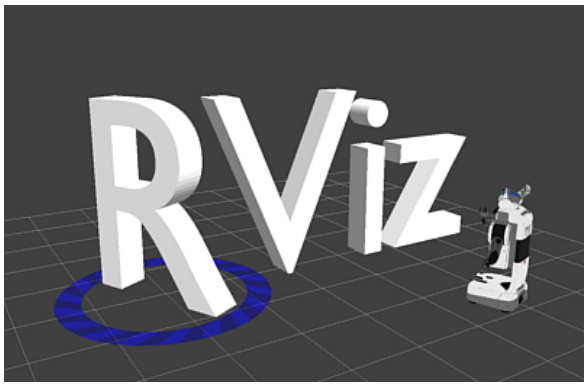


Figure 2.4: RViz logo.

There is a list containing the type of visualization. The type indicates the kinds of data displayed by a given display.

2.4 MATLAB

MATLAB is a scientific computing environment that can be used on multiple levels, from the pocket calculator to the simulation and analysis of complex systems, and is developed by *MathWorks* [24].

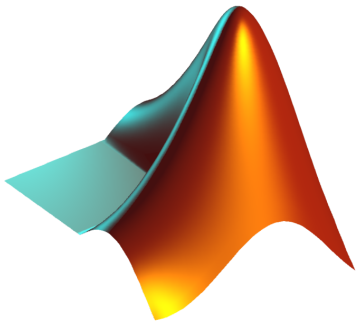


Figure 2.5: MATLAB logo.

MATLAB is supported by operating systems such as Windows, Mac OS, GNU/ Linux and Unix, and is optimized to solve scientific and design problems. Engineers and scientists from all over the world use this software to analyze and design the systems and products that transform our world: MATLAB is in fact active safety systems in automobiles, in interplanetary spacecraft, in health monitoring devices, in smart power grids and in LTE cellular networks. MATLAB is used for machine learning, signal processing, image processing, machine vision, communications, computational finance, control design, robotics and much more. The name *MATLAB* is an abbreviation of *Matrix Laboratory*: in fact, its programming language is based on the matrix (a scalar is a 1x1 matrix), which represents the most natural way in the world to express computational mathematics. The integrated graphics, combined with a large library of toolboxes, simplifies visualization and offers an detailed understanding of the data.

2.4.1 Simulink

Simulink [25] is an environment for modeling, analyzing, and simulating dynamic systems, developed by the American company *MathWorks*. Simulink supports the simulation of linear, and/or non-linear systems, which operate with continuous and/or discrete signals of a continuous and/or discrete time type. Simulink provides a graphical interface that allows to define and build the model through its block diagram. This environment includes a large number of libraries that contain many predefined blocks that can perform different signal operations. Simulink is closely integrated with MATLAB. All blocks and parameters used in this works are

described in Section 5.

2.5 MoveIt!

MoveIt! is a software designed to have collision detection ability as one of its features and it can be defines as a collection of software packages and tools integrated with ROS to provide multiple capabilities such as motion planning, 3D perception, collision detection, kinematics solving, manipulation and control.

Figure 2.6: MoveIt! logo.

MoveIt! has a plugin based architecture, which allows the user to add their own capabilities without altering the architecture of `move_group`, MoveIt!'s central node. Figure 2.7 shows the MoveIt! architecture.

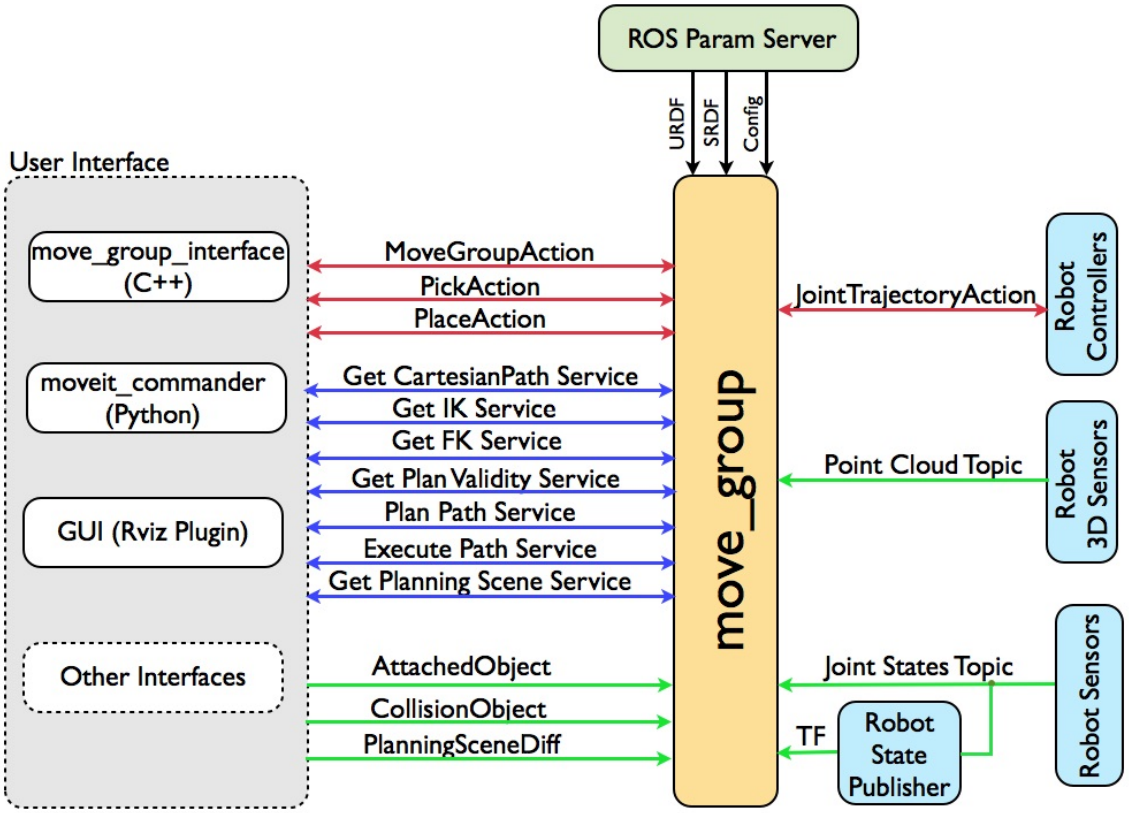


Figure 2.7: MoveIt! System Architecture.

The `move_group` node integrates the capabilities of the MoveIt! and make them available for the users through ROS actions and services. Note that this node does not execute any motion planner algorithms, instead it enables the plugin based architecture for all functionalities to integrate with it. So, its rule is simply to link every single system features through plugins. In this work, the trajectory has been programmed with MoveIt!, while the planner used for the trajectory computation is *STOMP* (*Stochastic Trajectory Optimization for Motion Planning*) [26]. It is an optimization-based motion planner that consists in generating a number of noisy trajectories that can allow to explore space around an initial given trajectory. Each of them has its own cost and this parameter will be examined to select, amongst all, who is the candidate to be the best solution, that means the trajectory with the lowest cost value. At each iteration, the algorithm optimizes a cost function based on a combination of several factors. No gradient information is required for this very good algorithm.

2.6 Point Cloud Library

A *point cloud* is a set of points characterized by their position in a coordinate system and by any intensity values (color, depth, etc.) associated with them. It is the most used 3D representation because of its simple extraction from cameras. A point cloud is essentially an unordered collection of vertices; the fact that there is no order among the vertices means that it is very diffcult to perform search algorithms around the points neighbors, unless there is first built a hierarchical representation of the points (such as the *KD-tree*). More, the fact that there is no topology of the points, that is the set of connections between them, means that it is complicated to discriminate the external surfaces of an object from the internal ones.

Determining the surface direction is the typical surface rendering problem, based on the notion of *mesh*. Meshes can be defined as point clouds which have a topology and tipically they are analized to estimate shadows or normal vector. On the other hand, the introduction of a particular order between the points of a point cloud originates the so-called voxelized cloud, where the notion of voxel flows in the pixel's 3D interpretation. On the other hand, the introduction of a particular order between the points of a point cloud originates the so-called *voxelized cloud*, where the notion of *voxel flows* in the pixel's 3D interpretation. In fact, a voxelized cloud is a 3D grid of light intensity values and this approach represents the most appropriate representation of data to obtain efficient 3D analysis algorithms, since the coordinates of a particular point are implicitly defined by the index representing the point position within the 3D grid. The presence of this index means that this is an orderly representation of points without topology (but easily accessible from the grid), which is easy to perform search algorithms and is great for viewing 3D scenes (the detail level depends on grid resolution). The only disadvantage lies in the fact that, depending on the device used, it takes a further step to get the voxelized cloud from other representations. Some particular types of sensors, such as metal detectors, directly obtain the voxelized cloud.

If it is possible to project a voxelized cloud on a 2D grid, it is generated the *range image* which is also an orderly representation of a 3D scene and this representation is

particularly useful for analyzing, segmenting and describing them. Figure 2.8 shows these transformations.

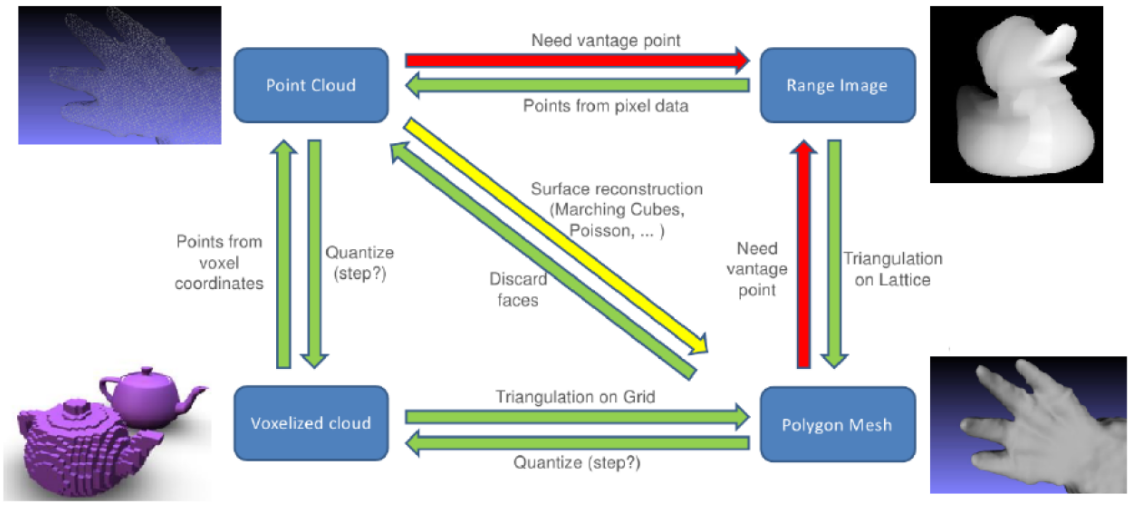


Figure 2.8: Transformations between 3D representations.

Point Cloud Library (PCL) [27] is an open source library, licensed under BSD terms which contains powerful processing tools and algorithms to implement 3D data perception applications. PCL provides algorithms for filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation.

Figure 2.9: Point Cloud Library logo.

The PCL structure splits the provided algorithms into smaller code libraries that can be compiled separately. This modularity is important for distributing PCL on platforms with reduced computational or size constraints. PCL presents several modules [28] and many of them have been adopted for the proposed application:

- Common: contains the common data structures and methods used by the majority of PCL libraries. The core data structures include the Point Cloud class and a multitude of point types that are used to represent points, surface normals, and RGB color values and feature descriptors;
- Filters: contains outlier and noise removal mechanisms for 3D point cloud data filtering applications. It also contains generic filters used to extract subsets of point cloud, or to exclude parts of it. It provides a VoxelGrid class to down-sample a point cloud by intersecting it with a lattice of points;
- I/O: contains classes and functions for reading and writing point cloud data (PCD and PLY) files, as well as capturing point clouds from a variety of (OpenNI compatible) sensing devices;
- Kd-tree: provides the kd-tree data-structure, using FLANN implementation that allows for fast nearest neighbour searches. A Kd-tree (k-dimensional tree, in most cases in this work it is a 3d-tree) is a space partitioning data structure that stores a set of k-dimensional points in a tree structure that enables efficient range searches and nearest neighbour searches. Nearest neighbour searches are a core operation when working with point cloud data and can be used to find correspondences between groups of points or feature descriptors or to define the local neighbourhood around a point or points;
- Search: provides methods for searching for nearest neighbours using different data structures, including kd-trees, octrees, brute force and specialized search for organized datasets;
- Segmentation: contains algorithms for segmenting a point cloud into distinct clusters. These algorithms are best suited to processing a point cloud that is composed of a number of spatially isolated regions. In such cases, clustering is often used to break the cloud down into its constituent parts, which can then be processed independently. It also contains algorithms to find differences between two point cloud, that can be used for example for quality inspection purposes.
- Visualization: this library was built for the purpose of being able to quickly prototype and visualizes the results of algorithms operating on 3D point cloud

data.

In this work, PCL represents the core of the developed pipeline, from the *Background subtraction* till the separation distance computation, detailed in Section 3.3.

Chapter 3

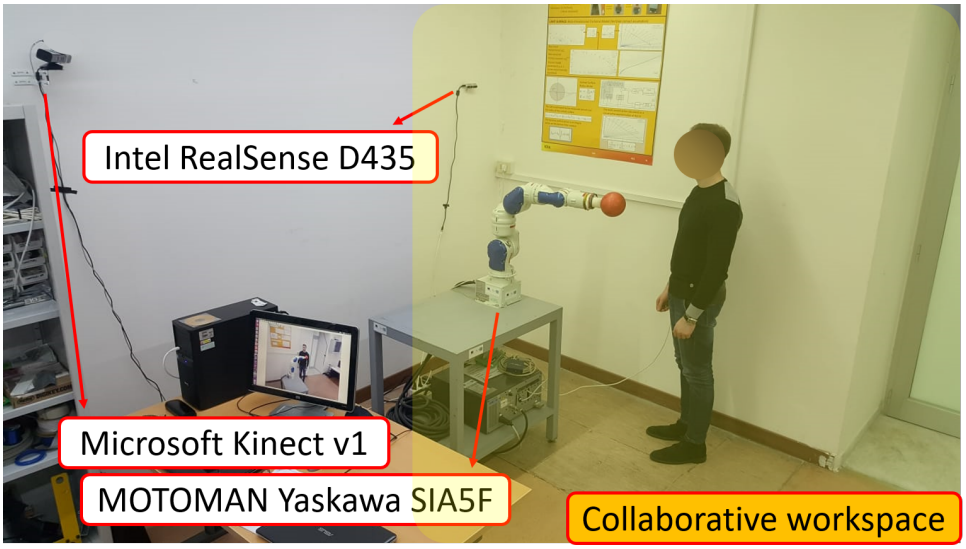
Human-Robot Interaction

The robot control system must be able to adapt the robot trajectory to the current observed scene and to perform its task efficiently and safely. This means that control system must be able to detect the presence of human operators inside the collaborative workspace, to track the human which is closest to the machine and, finally, to modulate the robot speed according to the minimum protective distance S.

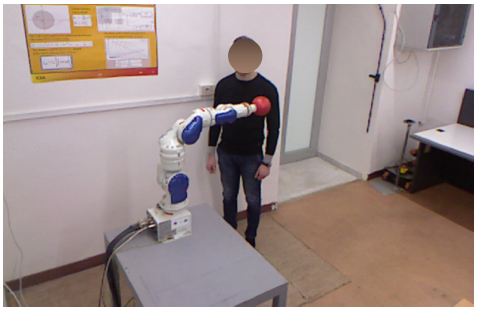
The HRC has been addressed dividing it into two distinct problems: human detection and tracking (HDT) and intention estimation (IE).

3.1 Perception System

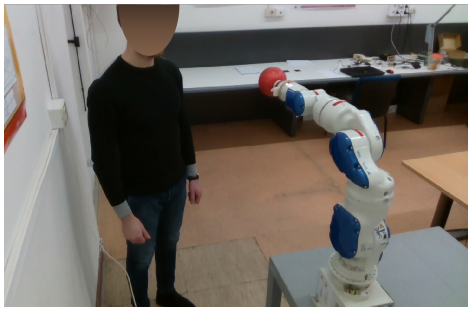
As mentioned in Chapter 2, two depth cameras have been used in this work to monitor the collaborative workspace. An intrinsic calibration, whose procedure is described in the Section 3.2.1, is necessary to update the rough intrinsic default parameters, as well as, a sphere-tracking procedure has been developed for extrinsic calibration (see Section 3.2.2). The obtained matrices, $T^{robot}_{camera1}$ and $T^{robot}_{camera2}$, express the poses of the camera frames with respect to the robot base frame.



(a) Perception system.



(b) Kinect RGB view.



(c) RealSense RGB view.

Figure 3.1: Experimental set-up.

3.2 Camera Calibration

Camera calibration is an essential task for extracting metric information from 2D images. A camera is calibrated using a model defined by two families of parameters: intrinsic parameters, which describe the camera regardless of its position in space, and extrinsic parameters, which describe the position of the camera in space independently of its internal features. This model must be appropriately corrected to reflect the optical distortions introduced by the camera lenses.

3.2.1 Intrinsic camera parameters

Microsoft Kinect v1

The instrinsic calibration of Microsoft Kinect v1 was carried out using the tool in *camera_ calibration* ROS package [29], which allows to automatically estimate the parameters of a camera by using patterns such as chess boards or grids, as shown in Figure 3.2. This tool allows to extract the intrinsic parameters of the Kinect in K matrix form and save them in a `.yaml` file, which is recalled when the camera drivers are launched. The K matrix structure is the following

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

To make the tool works correctly, the number of rows and columns of the pattern in question, i.e., a chess, the shape of the pattern and the dimensions has been set.

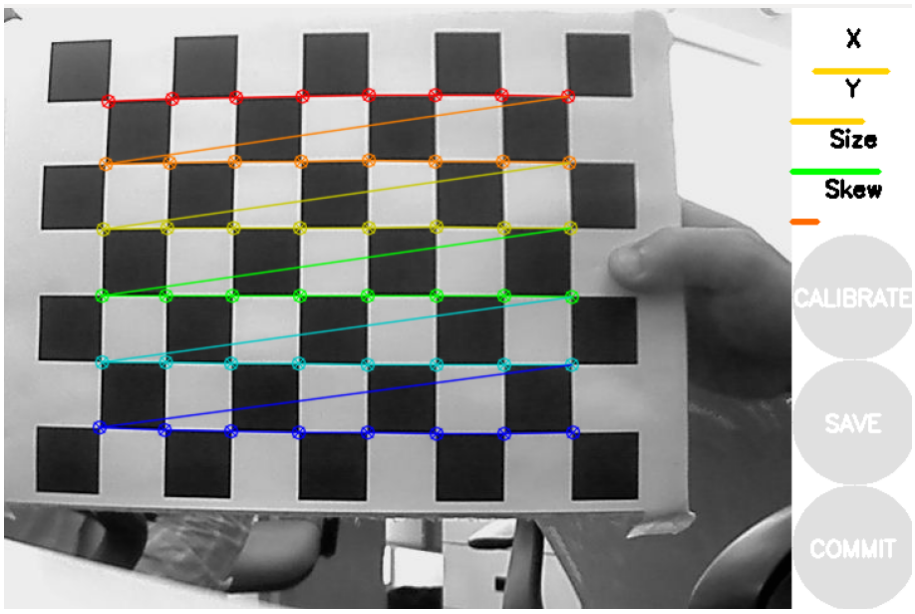


Figure 3.2: Example of intrinsic parameters estimation with a chesss.

Following are the results obtained for the Microsoft Kinect v1

$$K_{depth} = \begin{bmatrix} 587.0563067729362 & 0 & 320.6153422514201 \\ 0 & 590.2429336923416 & 250.4053454922714 \\ 0 & 0 & 1 \end{bmatrix}; \quad (3.2)$$

$$K_{rgb} = \begin{bmatrix} 505.3915336030262 & 0 & 324.8264408337666 \\ 0 & 506.3435945502578 & 262.4795332991785 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.3)$$

Intel RealSense D435

Regarding the Intel RealSense D435, the `camera_calibration` tool could not be used because this camera does not support `.yaml` but `.json` files. However, Intel provides a real-time executable viewer, as shown in Figure 3.3, to be able to calibrate all calibration parameters (about a hundred) based on a target.

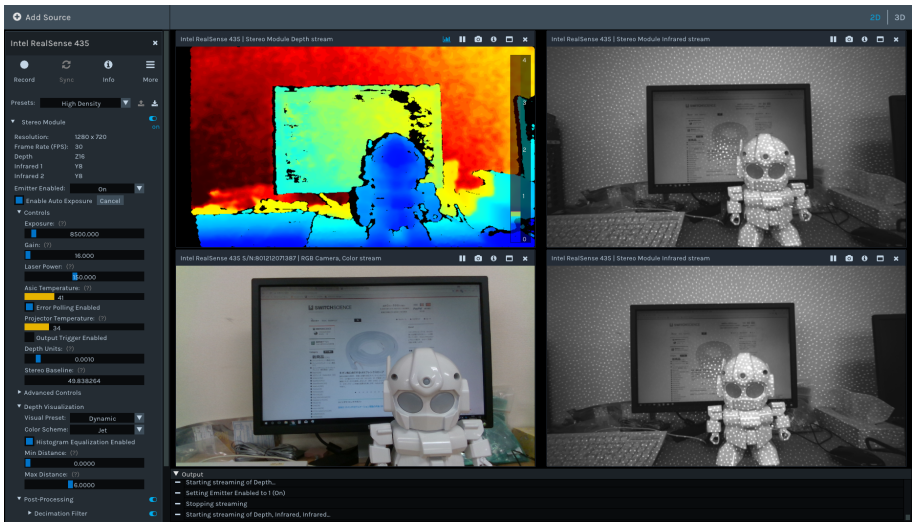
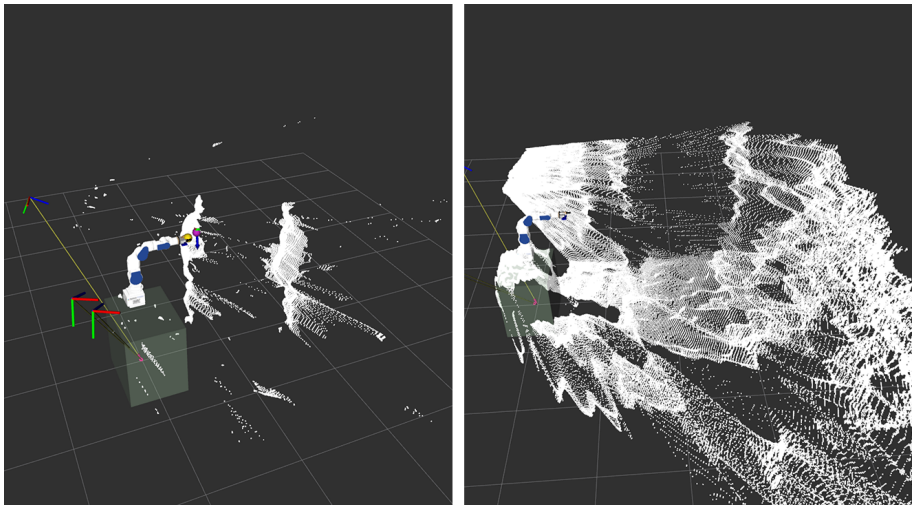


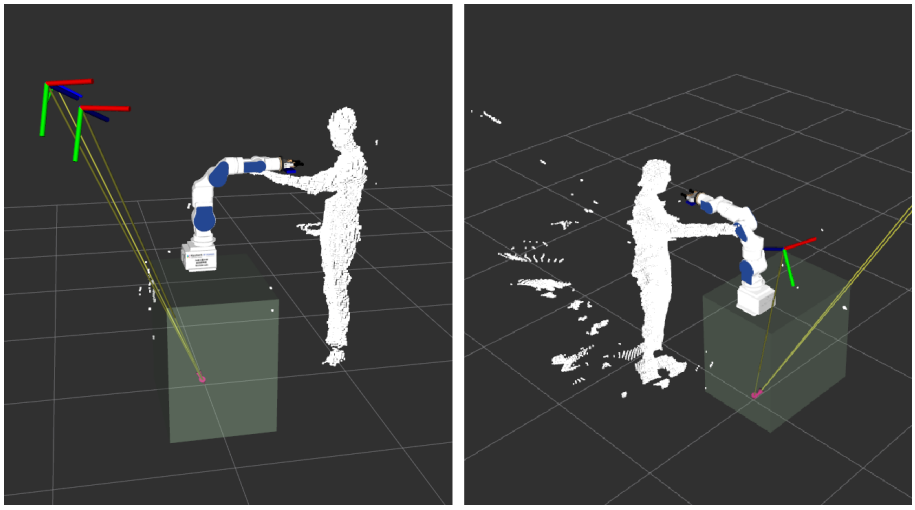
Figure 3.3: Example of real-time executable viewer.

In this case, the target was to have the edges of the frame objects as clean as possible (see Figure 3.4(b)) to avoid measurement errors of the default configuration of the Intel itself (see Figure 3.4(a)). For this purpose, some filters have been applied; the main ones are described below:

- *Sub-sampling*: since the applications may have their speed and performance negatively impacted by having to process many input data having many depth points (i.e., 1280×720 resolution), the sub-sampling of the input has been made right after it has been received (i.e., 848×480 resolution);
- *Edge-preserving filtering*: this type of filter smooth the depth noise while attempting to preserve edges;
- *Spatial filtering*: it is an image processing technique for changing the intensities of a pixel according to the intensities of the neighboring pixels;
- *Temporal filtering*: it is recommend to use some amount of time averaging to improve the depth, making sure not to let "holes" (depth= 0) influence the computations. Here, there is an alpha parameter which represent the extent of the temporal history that have to be averaged, and a threshold parameter, delta. They allow to reduce temporal smoothing near edges to not include holes in the averaging.



(a) Intrinsic parameters by default.



(b) Intrinsic parameters calibrated.

Figure 3.4: Intel RealSense D435 intrinsic parameters.

Once the desired depth has been obtained, it is possible to download the `.json` calibration file and refer to it in the launch file of the camera drivers.

3.2.2 Extrinsic camera parameters

At the state-of-the-art (SoA), there are different approaches for identifying the extrinsic calibration matrix of a depth camera, T_c^w. The goal is to obtain an accurate identification of the camera pose, which guarantees the minimum relative positioning error when the two camera views are merged. The adopted solution implements

a sphere tracking approach to obtain the transformation matrix between the robot base frame and the depth camera frame, T_c^b. Many in detail, a red sphere of 0.12 m diameter has been mounted at the robot end effector, as shown in Figure 3.5.

While the center coordinates of the sphere expressed in robot base frame, P_i^b, have been easily obtained through the robot forward kinematic, the corresponding points expressed in depth camera frame, P_i^d, have been estimated through a point cloud-based tracking approach. This procedure is described in the Section 3.2.2. At the end, the resulting corresponding couples of points have been analyzed and an optimization of a cost function provides the desired transformation matrix.

P_i^d computation

The procedure described below was adopted for both depth cameras. The implementation, realized in MATLAB [24], consists of:

1. positioning of the robot in a configuration that allows to correctly distinguish the sphere from the background;
2. acquisition of point cloud;
3. identification of a ROI (Region Of Interest) to restrict the search area to an area in which the sphere is present;
4. execution of the *M-estimator SAmple Consensus* (*MSAC*) algorithm for the estimation of the parameters of the mathematical model of the sphere;
5. evaluation of the model obtained according to a radius constraint;
6. repetition of steps 2. to 5. for the acquisition of K models that meet the constraint;
7. computation of the mean of K valid results.

This procedure has been iterated to acquire N points that describe the coordinates of the center of the sphere positioned in several points of the robot workspace. The robot, due to the hardware limitations about the noise of the depth cameras and the considerable distance of the sphere with respect to the sensor frame, has been moved manually because only a limited portion of the space allowed the depth camera a satisfactory recognition of the sphere. The pseudo-code of the procedure is reported in Algorithm 1. Then, a more detailed explanation of the seven points

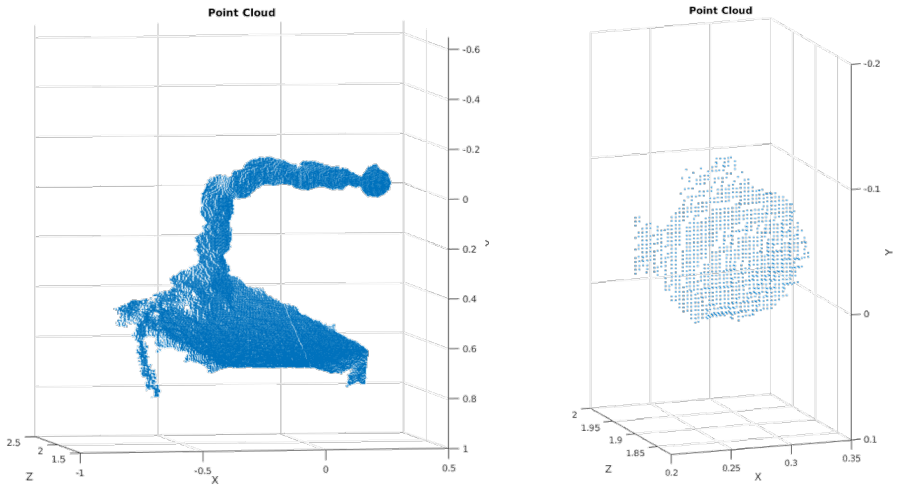
described above.

Algorithm 1 P_i^d computation

input:
$r_n \leftarrow$ Nominal sphere radius
$r_e \leftarrow$ Estimated sphere radius
$\Delta r \leftarrow$ Acceptable radius displacement
$ROI \leftarrow$ Region Of Interest
$K \leftarrow$ Minimum number of constrained samples to be acquired
$C \leftarrow$ Number of acquired, constrained samples
$PC_o \leftarrow$ Original point cloud
$PC_s \leftarrow$ Segmented point cloud
$acc_m \leftarrow$ Accumulator matrix of estimated sphere models
$P_i^d \leftarrow i$-th target point expressed in depth frame
procedure SPHERE CENTER ESTIMATION
    $C = 1$
    while $C < K$ **do**
        $PC_o = pcrec$
        $PC_s = pcseg(ROI, PC_o)$
        $[p, r_e] = pcfitspehere(PC_s)$;
        if $|r_e - r_d| <= \Delta r$ **then**
            $acc_m[C, 1:3] = p$
            $acc_m[C, 4] = r_e$
            $C = C + 1$
        end if
    end while
    $P_i^d = mean(acc_m[:, 1:3])$
end procedure

First of all, after having connected to the ROS network, it is necessary to acquire the point cloud of the used camera (see Figure 3.5(a)). The MATLAB function that allows to return a geometric model that describes the sphere is *pcfitsphere*. This function, which returns the center of the sphere and the radius, uses the *MSAC* algorithm [30], a variant of the *Random Sample Consensus* (*RANSAC*) algorithm [31], to find the sphere (see Figure 3.5(b)). In order to use the *pcfitsphere* function, it is necessary to provide it the ROI as an input parameter; for this reason, the

function *findPointsInROI* has been used. Then, a constraint has been imposed by exploiting the knowledge of the radius of the sphere: of all the outputs returned by *pcfitsphere* function, only those with a radius in the range of 0.058 m to 0.062 m have been considered.



(a) Point cloud acquisition. (b) Search for the sphere .

Figure 3.5: Experimental set-up.

Once the coordinates of the center of the sphere subject to the constraint have been obtained, a mean of them has been made to obtain a single frame that has a radius that is more faithful to the true radius of the sphere. For each frame calculated, the corresponding configuration of the robot in joints space has been recorded. Collected a sufficient number of points (i.e., $N = 8$ in the present case) and knowing the robot kinematics, the transformation matrix of the camera frame expressed in basis frame of the robot T_c^b has been calculated by means of an algorithm for optimizing a cost function that weighs the collected points proportionally to the distance of the points from the camera. Then, calculated the transformation matrix of the camera frame from the world frame T_c^w, and obtained the quaternion from the rotation matrix R of T_c^w, position and orientation of the camera frame expressed in world frame were obtained. The transformation matrix of the Microsoft Kinect v1 frame from the world frame T_k^w and the transformation matrix of the Intel RealSense D435 frame from the world frame T_i^w are shown below:

$$T_k^w = \begin{bmatrix} 0.8495 & -0.1645 & 0.5013 & -0.7516 \\ -0.5267 & -0.3191 & 0.7879 & -1.8008 \\ 0.0304 & -0.9333 & -0.3577 & 2.1462 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{3.4}$$

$$T_i^w = \begin{bmatrix} -0.2539 & -0.4330 & 0.8649 & -0.8735 \\ -0.9672 & 0.1089 & -0.2295 & 0.7020 \\ 0.0052 & -0.8948 & -0.4465 & 1.9651 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{3.5}$$

3.3 Human Detection and Tracking

Realizing a safe HRC application requires a very fast HDT algorithm, which detects human operators in real time. In this work, a novel point cloud-based methodology is presented to compute the minimum distance between the whole body of the detected operators and a robot. Since this operation is computationally heavy, a *Background Segmentation* (BS) algorithm is developed to subtract the static environment from the whole observed scene and to process exclusively the information related to the dynamic objects. The developed pipeline is shown in Figure 3.6, while the pseudo-code of the whole procedure is reported in Algorithm 2.

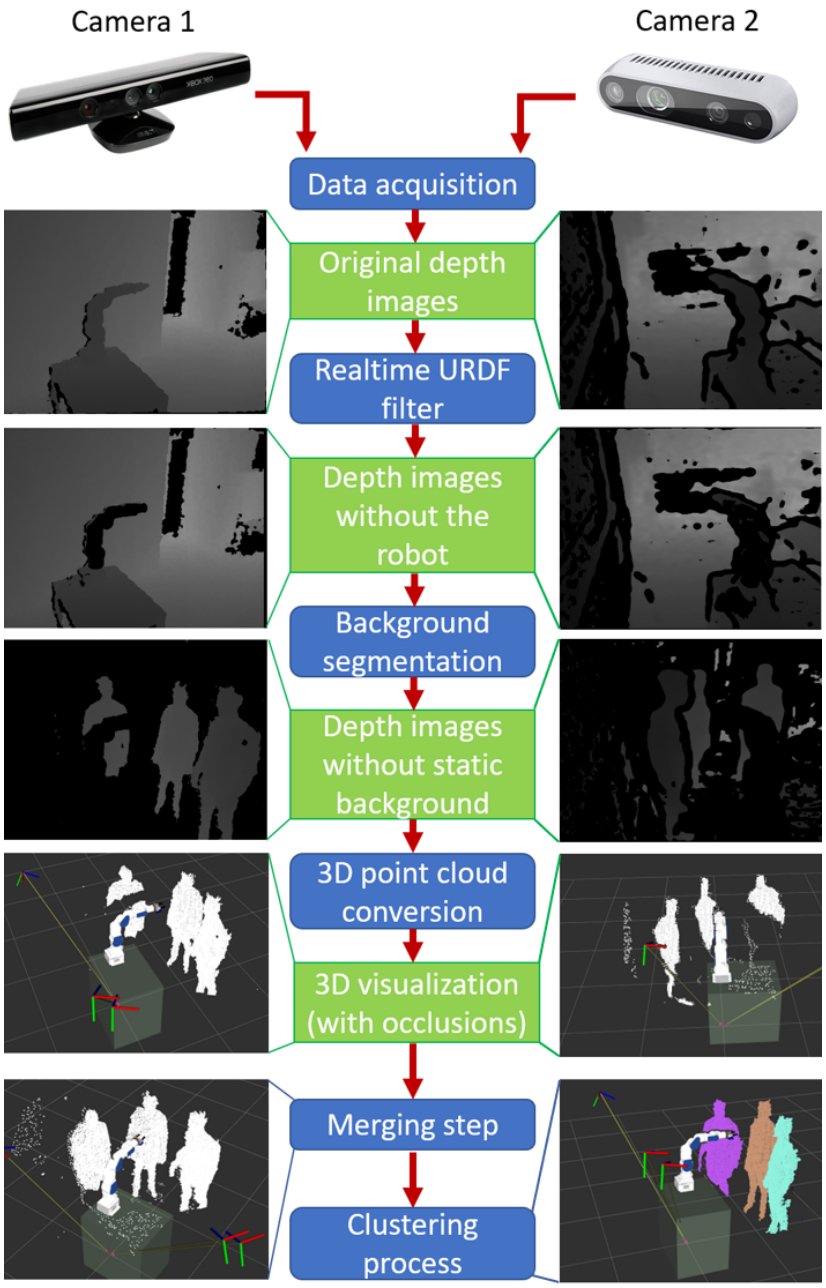


Figure 3.6: Implemented HDT pipeline.

The perception system described in Section 3.1 observes the surroundings of the manipulator and the robot kinematic chain is fully visible. While the workspace is monitored, the robot executes its task, thus it becomes a dynamic entity. Therefore, the *Realtime URDF Filter* [32] is used at the beginning of the pipeline to remove the robot from the scene.

Algorithm 2 d_s computation and closest cluster identification

input:
$S_{source1} \leftarrow$ Original depth image frame from camera 1
$S_{source2} \leftarrow$ Original depth image frame from camera 2
$S^0_{source1} \leftarrow$ Frame to be subtracted from the current depth image 1
$S^0_{source2} \leftarrow$ Frame to be subtracted from the current depth image 2
$PCD_1 \leftarrow$ Point cloud data corresponding to depth image 1
$PCD_2 \leftarrow$ Point cloud data corresponding to depth image 2
$PCD_{1,2} \leftarrow$ Merged point cloud data
$R_m \leftarrow$ URDF robot model
$clusters \leftarrow$ PCD clusters extracted from the current scene
$acc \leftarrow$ Accumulator matrix of clusters separation distances
$d_s \leftarrow$ Robot-closest operator separation distance
$cluster \leftarrow$ Closest operator cluster
procedure SEGMENTATION PIPELINE
    $bool\ first\_frame = true$
    $S_{source1} = depthAcquisition(camera1)$
    $S_{source2} = depthAcquisition(camera2)$
    $S_{source1} = S_{source1} - R_m$
    $S_{source2} = S_{source2} - R_m$
    if $first\_frame == true$ **then**
        $S^0_{source1} = S_{source1}$
        $S^0_{source2} = S_{source2}$
        $first\_frame = false$
    end if
    $S_{source1} = S_{source1} - S^0_{source1}$
    $S_{source2} = S_{source2} - S^0_{source2}$
    $PCD_1 = pcdConversion(S_{source1})$
    $PCD_2 = pcdConversion(S_{source2})$
    $PCD_{1,2} = PCD_1 + PCD_2$
    $clusters = EuclideanClusterExtraction(PCD_{1,2})$
    for $i = 1$ to $clusters.size$ **do**
        $acc_i = computeMinDist(R_m, clusters_i)$
    end for
    $(d_s, cluster) = min(acc)$
end procedure

The implementation of the BS step consists of an efficient algorithm that performs the subtraction of a stored background, at pixel level: 50 frames of a static scene in the absence of human workers are initially captured and the minimum value of each pixel is stored in a memory area. Therefore, the stored frame is subtracted from the current frame at every acquisition.

The algorithm makes use of the Point Cloud Library (PCL) [27]: the depth information is converted into Point Cloud Data (PCD) and a uniform sampling filter can be applied to make the algorithm more reactive, by decreasing the PCDs density.

Subsequently, a reference camera has been selected to express the entire output of the perception system relative to a single camera frame. In the case of study, the the Kinect sensor has been selected. The point clouds have been combined through the *merging step* (MS), which allows to obtain a single point cloud starting from the inital ones. The accuracy reached during the extrinsic calibration procedure, described in Section 3.2.2, allowed to obtain a satisfying correspondence. To obtain a single point cloud, the following two operations are required:

1. computation of the transformation matrix between the frame in which the point cloud is expressed and the reference frame passed as an input parameter;
2. transformation of both point clouds in the reference frame and sum of them in a single final point cloud.

Finally, the *clustering process* (CP) provides as many clusters as single dynamic areas are detected in the foreground. The *Euclidean cluster extraction* method is performed to highlight all the human clusters of the collaborative workspace. The bottom right image of Figure 3.6 shows three detected human operators, whose shapes are distinguishable by different colors. To compensate the sensors measurement noise that could sometimes provide false clusters, the areas in the foreground should be large enough to represent a human body. As a consequence, a minimum PCD cardinality threshold has been experimentally determined to discriminate the validity of the cluster.

3.4 Human-Robot Separation Distance

The goal of the proposed HRC strategy is to identify the nearest pair of points, one belong to the robot (P_R) and the other one belonging to the operator (P_H), that minimize the distance, i.e.,

$$P_H \in \mathcal{H}, P_R \in \mathcal{R} \mid d(P_H, P_R) \leq d(P'_H, P'_R) \quad \forall P'_H \in \mathcal{H}, P'_R \in \mathcal{R} \tag{3.6}$$

where $d(\cdot, \cdot)$ is the Euclidean distance between two points, $\mathcal{H}$ and $\mathcal{R}$ represent the set of all points that belong to the operator and to the robot, respectively.

Therefore, alongside the HDT strategy, a robot modeling method has been also implemented: unlike the SoA assumptions that consider only a singular representative coordinate of the robot (e.g., the end effector), introducing an inefficient estimation of the distance between the operators and the robot kinematic chain, or, on the other hand, report the pose of the robot only in terms of either joint configurations or in terms of the Cartesian pose of the robot kinematic frames, without taking into account the link volumes but considering only specific points, the proposed solution models the entire robot kinematic chain and its volume. A computationally efficient way to represent the whole robot is to use primitive shapes, e.g., ellipses and spheres [33]. A similar convention was proposed in [34]. This work is inspired by the same idea, but pays attention to some aspects: since the robot links can have a variable length, its kinematic chain has been padded through dummy frames to protect the robot homogeneously, and a 0.10 m diameter security sphere has been created around each frame, taking into account the last frame that can incorporate a tool and/or an attached object (e.g., during a pick and place task).

Under such assumptions, the pair of human-robot points that are closest to each other can be immediately identified. This step strongly justifies the choice of a point cloud-based pipeline. In fact, the point cloud provides much more detailed information, accuracy and precision if compared to the major HDT techniques present in the SoA literature cited in Chapter 1. Unlike common skeleton-based techniques,

the proposed approach allows tracking humans also when they are carrying objects. Moreover, it is not necessary that human operators are in front of the camera view: the point cloud will recognize them anyway. Furthermore, detecting the pair of human-robot points at a minimum distance (Equation 3.6) is particularly immediate. The algorithm calculates the distance between all points of a clusters point cloud and the origin of every robot frame. Eventually, the robot point P_R will be the one on the surface of the virtual sphere, around the identified frame, which lies on the line connecting the origin of this frame and the closest point in the cluster. From these results, the closest human cluster is indirectly selected if more than one human have been detected.

Figure 3.7 shows the results. Note that the proposed approach is able to identify more detailed body parts, e.g., a elbow, the head, an hand, the chin or the chest, and also that P_R can be detected along the whole robot kinematic chain. Figure 3.8 demonstrates the effectiveness of the proposed approach in multi-humans scenarios. The results of the experimental tests described in Chapter 7 will be used to evaluate the performances ofthe algorithm.

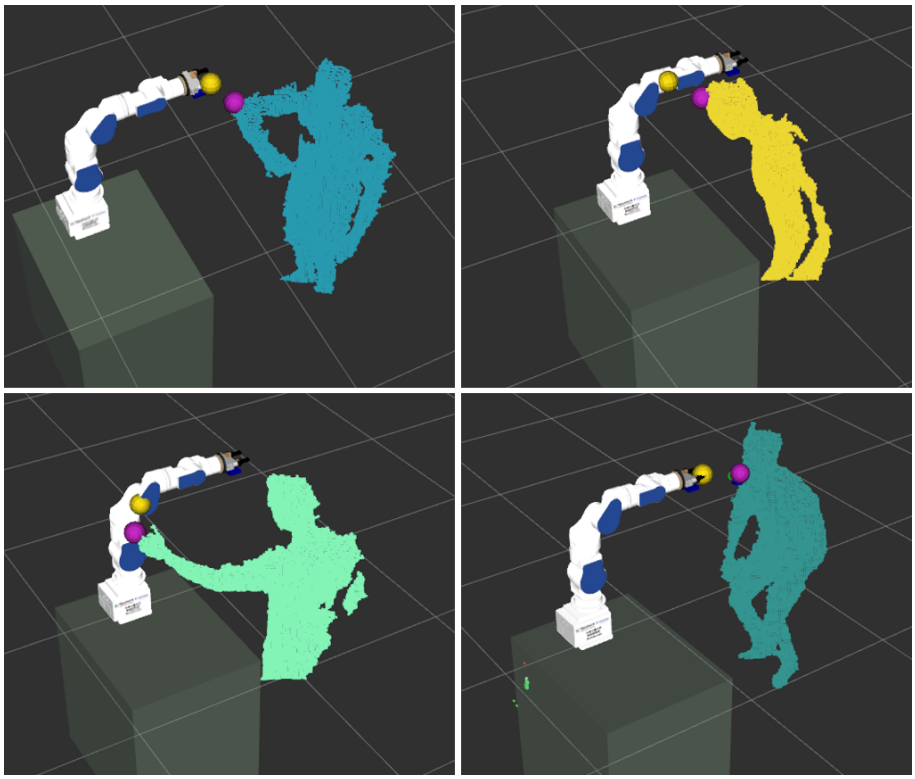


Figure 3.7: Identification of the minimum distance points.

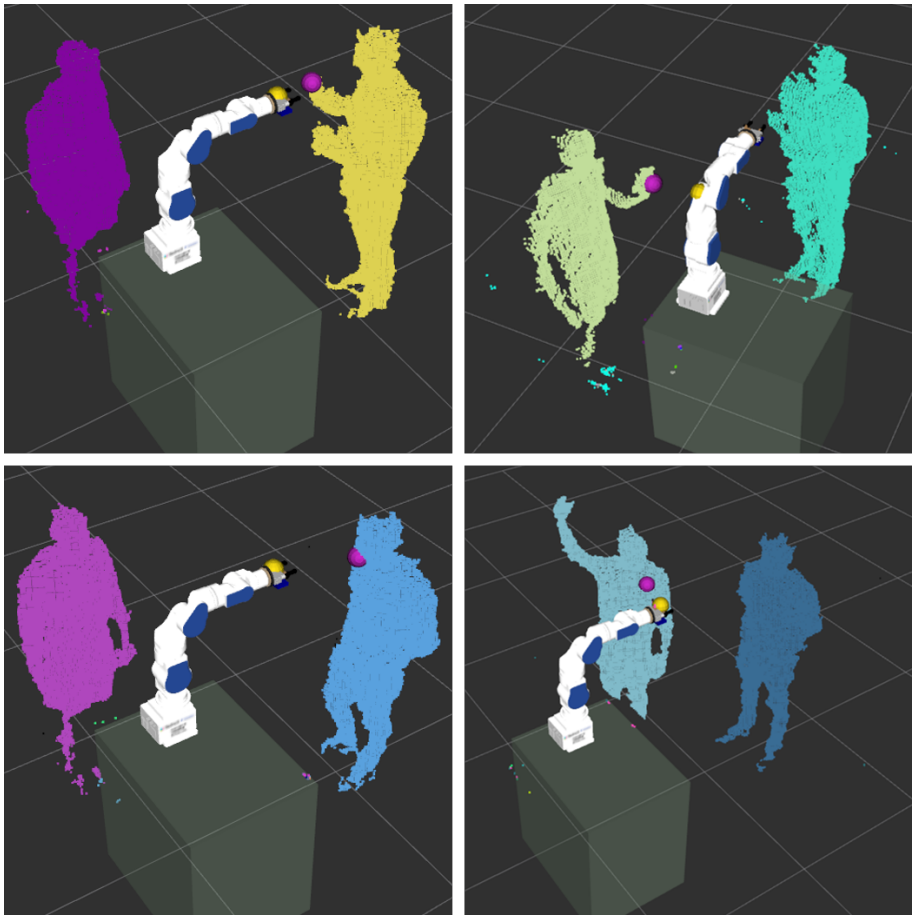


Figure 3.8: Multi-humans tracking.

3.5 Estimation of operator and Robot velocities

In addition to the HDT strategy, another fundamental function of the HRC problem is represented by IE, i.e., the prediction of human movement. From such information, the robot control system will select the most appropriate value of its joint speeds to avoid a potentially dangerous situation, as explained in explained in Chapter 6 and according to ISOs presented in Section 1.5.

IE consists in estimating the next position and velocity of the trajectory performed by the operator on the basis of a series of positions previously acquired.

The sensor fusion strategy that has been integrated into this work is based on a Linear Kalman Filter (LKF) [35], which tries to solve the problem of estimating the state of a discrete-time process governed by the equations

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} \boldsymbol{I}_3 & \Delta t \boldsymbol{I}_3 \\ \boldsymbol{O}_3 & \boldsymbol{I}_3 \end{bmatrix} \boldsymbol{x}_k + \boldsymbol{w}_k \tag{3.7}$$

$$\boldsymbol{y}_k = \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{O}_3 \end{bmatrix} \boldsymbol{x}_k + \boldsymbol{n}_k, \tag{3.8}$$

where Δt is the sampling time, $\boldsymbol{I}_3$ and $\boldsymbol{O}_3$ are the identity and zero matrices of size 3×3, respectively; $\boldsymbol{w}$ and $\boldsymbol{n}$ are the process and measurement noises with covariance matrices $\boldsymbol{W}$ and $\boldsymbol{N}$, respectively. Finally, $\boldsymbol{x}$ is the state vector of the system, i.e., the position and the velocity of the operator $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_H^T & \dot{\boldsymbol{p}}_H^T \end{bmatrix}^T$, and the measured output $\boldsymbol{y}$ is a vector containing the coordinates of the point P_H described in Section 3.4. The covariance matrix $\boldsymbol{N}$ is experimentally estimated, while the covariance matrix $\boldsymbol{Q}$ has been chosen as

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{I}_3 \Delta t^2 & \boldsymbol{O}_3 \\ \boldsymbol{O}_3 & \boldsymbol{Q}_2 \end{bmatrix} \tag{3.9}$$

where $\boldsymbol{Q}_2$ quantifies the uncertainty on the velocity dynamics (assumed constant) of the state equations.

Based on the vector nature of the velocity, it is possible to make some considerations about the direction (trend) of the operator, that is to say, to predict in which direction he/she is traveling to. Chapter 4 describes how to take advantage from these considerations for industrial collaborative applications with the aim to maximize productivity.

The LKF equations implemented in this work are the standard ones and are shown in Table 3.1.

Predict	
Predicted (a priori) state estimate	$\hat{\mathbf{x}}_{k\|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1\|k-1} + \mathbf{B}_k\mathbf{u}_k$
Predicted (a priori) error covariance	$\mathbf{P}_{k\|k-1} = \mathbf{F}_k\mathbf{P}_{k-1\|k-1}\mathbf{F}_k^\mathsf{T} + \mathbf{Q}_k$
Update	
Innovation or measurement pre-fit residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\|k-1}$
Innovation (or pre-fit residual) covariance	$\mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k\mathbf{P}_{k\|k-1}\mathbf{H}_k^\mathsf{T}$
Optimal Kalman gain	$\mathbf{K}_k = \mathbf{P}_{k\|k-1}\mathbf{H}_k^\mathsf{T}\mathbf{S}_k^{-1}$
Updated (a posteriori) state estimate	$\hat{\mathbf{x}}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1}+\mathbf{K}_k\tilde{\mathbf{y}}_k\hat{\mathbf{x}}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1}+\mathbf{K}_k\tilde{\mathbf{y}}_k$
Updated (a posteriori) estimate covariance	$\mathbf{P}_{k\|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\mathbf{P}_{k\|k-1}\,(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^\mathsf{T}$ $+\mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^\mathsf{T}$
Measurement post-fit residual	$\tilde{\mathbf{y}}_{k\|k} = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\|k}$

Table 3.1: Linear Kalman Filter equations.

where

- $\hat{\mathbf{x}}_k$ is the a *posteriori* state estimate at time k given observations up to and including at time k;
- $\hat{\mathbf{P}}_k$ is the a *posteriori* error covariance matrix;
- $\mathbf{F}_k$ is the state-transition model which is applied to the previous state $\mathbf{x}_{k-1}$;
- $\mathbf{H}_k$ is the observation model which maps the true state space into the observed space;
- $\mathbf{Q}_k$ is the covariance of the process noise;
- $\mathbf{R}_k$ is the covariance of the observation noise;
- $\mathbf{B}_k$ is the control-input model which is applied to the control vector $\mathbf{u}_k$;

The tuned parameters are fully described in Chapter 7.

Figure 3.9 shows sample movements of the operator and the three components of his/her estimated speed.

The linear velocity $\dot{\boldsymbol{p}}_R$ of the point on the robot closest to the operator can be

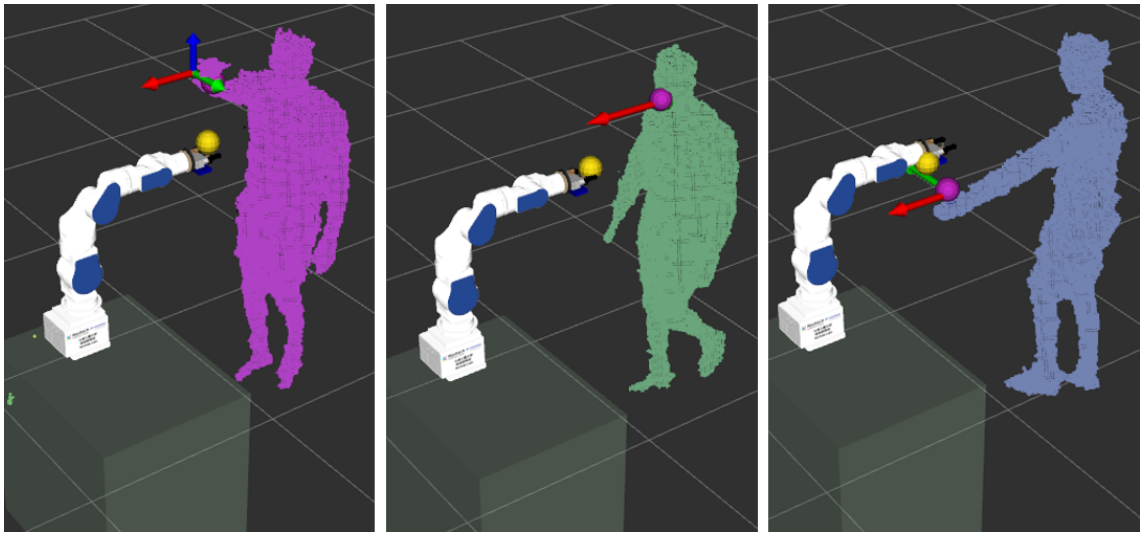


Figure 3.9: Estimation of operator velocity.

computed according to the differential kinematics equation

$$\dot{\boldsymbol{p}}_R = \boldsymbol{J}_p(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{3.10}$$

where $\boldsymbol{q}$ [rad] and $\dot{\boldsymbol{q}}$ [rad/s] are the robot joint position and velocity vectors, respectively; $\boldsymbol{J}_p$, on the other hand, is the position part of the Jacobian matrix calculated till the closest point.

The ISO/TS 15066 states that the "directed speeds" of the robot and the human should be used to compute S. This means that, in Equation 1.4, v_h is the operator speed in the direction of the moving part of the robot and v_R is the robot speed in the direction of the selected operator. Note also that these speeds are vector magnitudes, hence they are always grater or equal to 0. Therefore, the velocity terms of Equation 1.4 can be computed as

$$v_H = \left| \hat{\dot{\boldsymbol{p}}}_H^T \left(\frac{\boldsymbol{p}_R - \hat{\boldsymbol{p}}_H}{\|\boldsymbol{p}_R - \hat{\boldsymbol{p}}_H\|} \right) \right| \tag{3.11}$$

$$v_R = \left| \dot{\boldsymbol{p}}_R^T \left(\frac{\hat{\boldsymbol{p}}_H - \boldsymbol{p}_R}{\|\hat{\boldsymbol{p}}_H - \boldsymbol{p}_R\|} \right) \right|, \tag{3.12}$$

where $\hat{\boldsymbol{p}}_H$ and $\hat{\dot{\boldsymbol{p}}}_H$ are the operator position and velocity estimated by the LKF, respectively, and $\boldsymbol{p}_R$ is a vector containing the coordinates of the point P_R defined in Section 3.4.

Chapter 4

Fuzzy Logic

The protective separation distance S (Equation 1.4), computed by using the velocities of Equations 3.11 and 3.12, does not take into account the relative travel direction of the robot and the operator. This means that, if the robot and the operator are going away from each other, the value of S unnecessary increases (proportionally to the computed speed). To improve the production time considering also this situation, the protective separation distance has been redefined as follows

$$S = \alpha[(v_H T_R + v_H T_S) + (v_R T_R)] + (B) + (C + Z_S + Z_R) \tag{4.1}$$

where α is a coefficient in the interval $[0, 1]$ that is 1 when the operator and the robot are actually approaching to each other and is smaller than 1 otherwise. To chose the value of α, a fuzzy inference approach has been implemented.

4.1 What is Fuzzy logic

The *fuzzy logic*, also called *faded logic*, is a methodology in which each proposition represents a degree of truth into the interval $[0, 1]$ [36]. It starts out as an alternative to *traditional*, or *binary* or *crisp logic*, to reason with truth values that are not only 1 (true) and 0 (false). Examples like that in Figure 4.1 help to understand the concept of fuzzy logic.

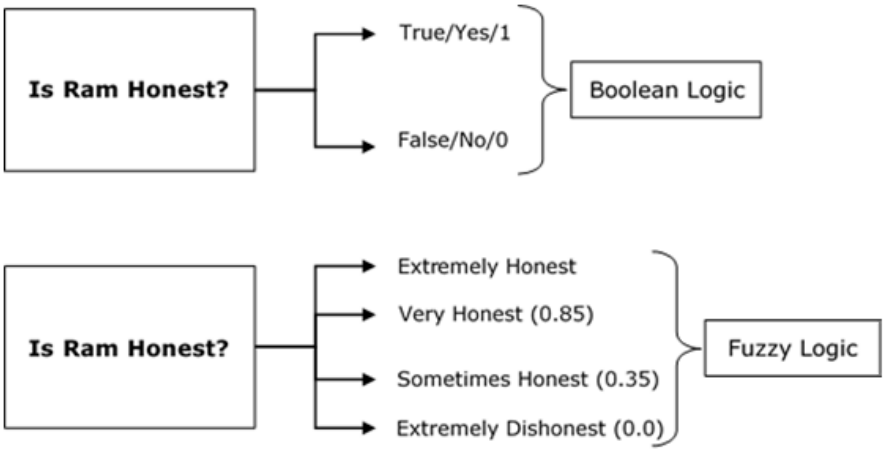


Figure 4.1: Difference beetween fuzzy and traditional logic.

In the traditional logic there are net thresholds beyond which one belongs to a category rather than another; however, in the fuzzy logic, the schemes intersect each other, so a given subject may belong in variable percentages to either one or the other group. For example, trms such as "tall" and "young" are fuzzy because they cannot be crisply defined. To classify a person as tall or young, it is impossible to decide if the person is in a set or not. By giving a degree of pertinence to the subset, no information is lost when the classification is made. This reasoning therefore allows a more gradual passage from one category to another according to the presence or absence of some characteristics. This has led to attach a certain importance to the fuzzy logic, up to its applications in software (knowledge-based systems) with the aim of replacing as much as possible the human decision-makers, and in the hardware (systems control) to replace the human operator. Table 4.1 shows some pros and cons of the fuzzy logic.

Pros	**Cons**
Conceptually easy to understand	Manual tuning of membership functions and other parameters which might be tedious and time-consuming
Flexibility	Not well adaptable to big and complicated problems
Tolerance inaccurate data	Crisp models can be more efficient and even convenient
Modelling of non-linear functions complexity of arbitrary complexity	No systematic approach to fuzzy system designing

Table 4.1: Pros and cons of fuzzy logic.

4.2 Fuzzy Sets

Fuzzy Sets are sets in which many degrees of membership are allowed, and indicated with a number between 0 and 1. The point of departure for fuzzy sets is simply the generalization of the valuation set from the pair of numbers $\{0, 1\}$ to all the numbers in $[0, 1]$. This is called a *membership function* and is denoted as $\mu_A(x)$.

Membership functions are mathematical tools for indicating flexible membership to a set, modeling and quantifying the meaning of symbols. They can represent a subjective notion of a vague class, such as chairs in a room, size of people, and performance among others. Commonly there are two ways to denote a fuzzy set. If X is the universe of discourse, and x is a particular element of X, then a fuzzy set A defined on X may be written as a collection of ordered pairs

$$A = \{(x, \mu_A(x))\} \qquad x \in X, \tag{4.2}$$

where each pair $(x, \mu_A(x))$ is a *singleton*. In a crisp set singletons are only x, but in fuzzy sets it is x and $\mu_A(x)$. For example, the set A may be the collection of the following integers, as in 4.3

$$A = \{(1, 1.0), (3, 0.7), (5, 0.3)\}. \tag{4.3}$$

Thus, the second element of A expresses that 3 belongs to A to a degree of 0.7. The *support set* of a fuzzy set A is the set of elements that have a membership function different from zero. Alternative notations for the fuzzy sets are summations or integrals to indicate the union of the fuzzy set, depending if the universe of discourse is discrete or continuous.

4.2.1 Types of Sets

Sets can be classified into many types [37]; some of them are listed below:

- *Finite set*: it contains a definite number of elements;
- *Infinite set*: it contains infinite number of elements;
- *Subset*: a set X is a subset of set Y $(X \subseteq Y)$ if every element of X is an element of set Y;
- *Proper subset*: a set X is a proper subset of set Y $(X \subset Y)$ if every element of X is an element of set Y and $|X| < |Y|$.
- *Universal set*: it is a collection of all elements in a particular context or application. All the sets in that context or application are essentially subsets of this universal set;
- *Empty set* or *Null set*: it contains no elements. As the number of elements in an empty set is finite, empty set is a finite set. The cardinality of empty set or null set is zero;
- *Singleton set* or *Unit set*: it contains only one element;
- *Equal set*: if two sets contain the same elements, they are said to be equal;

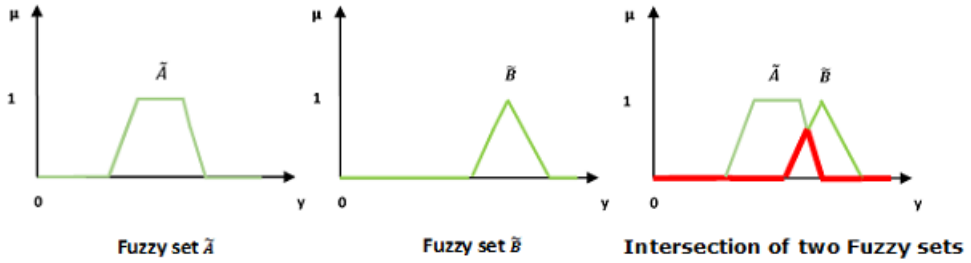
- *Equivalent set*: if the cardinalities of two sets are same, they are called equivalent sets;

4.2.2 Operations on Fuzzy Sets

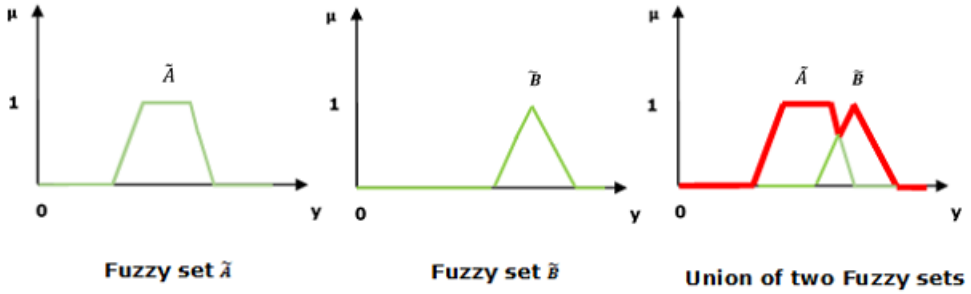
Operations such as *intersection* and *union* are defined through the *min* ($\wedge$) and *max* ($\vee$) operators respectively, which are analogous to *product* and *sum* in algebra. Formally the *min* and *max* of an element are denoted by Equations 4.4 and 4.5. Figure 5.7 helps to understand how *intersection* (Figure 4.2(a)) and *union* (Figure 4.2(b)) work.

$$\mu_{\widetilde{A}} \wedge \mu_{\widetilde{B}} = min(\mu_{\widetilde{A}}, \mu_{\widetilde{B}}) \equiv \begin{cases} \mu_{\widetilde{A}} & \text{if and only if} \quad \mu_{\widetilde{A}} \leq \mu_{\widetilde{B}} \\ \mu_{\widetilde{B}} & \text{if and only if} \quad \mu_{\widetilde{A}} > \mu_{\widetilde{B}} \end{cases} \tag{4.4}$$

$$\mu_{\widetilde{A}} \vee \mu_{\widetilde{B}} = max(\mu_{\widetilde{A}}, \mu_{\widetilde{B}}) \equiv \begin{cases} \mu_{\widetilde{A}} & \text{if and only if} \quad \mu_{\widetilde{A}} \geq \mu_{\widetilde{B}} \\ \mu_{\widetilde{B}} & \text{if and only if} \quad \mu_{\widetilde{A}} < \mu_{\widetilde{B}} \end{cases} \tag{4.5}$$



(a) Intersection of two Fuzzy sets.



(b) Union of two Fuzzy sets.

Figure 4.2: Intersection and Union on Fuzzy Sets.

Empty fuzzy set	It is empty if its membership function is zero everywhere in the universe of discourse.	$A \equiv \emptyset$ if $\mu_A(x) = 0, \forall x \in X$
Normal fuzzy set	It is normal if there is at least one element in the universe of discourse where its membership function equals one.	$\mu_A(x_a) = 1$
Union of two fuzzy sets	The union of two fuzzy sets A and B over the same universe of discourse X is a fuzzy set $A \cup B$ in X with a membership function which is the maximum of the grades of membership of every x and A and B. This operation is related to the OR operation in fuzzy logic.	$\mu_{A \cup B}(x) \equiv \mu_A(x) \vee \mu_B(x)$
Intersection of fuzzy sets	It is the minimum of the grades of every x in X to the sets A and B. The intersection of two fuzzy sets is related to the AND.	$\mu_{A \cap B}(x) \equiv \mu_A(x) \wedge \mu_B(x)$
Complement of a fuzzy set	The complement of a fuzzy set A is denoted as $\bar{A}$.	$\mu_{\bar{A}}(x_a) \equiv 1 - \mu_A(x_a)$

Product of two fuzzy sets	$A \cdot B$ denotes the product of two fuzzy sets with a membership function that equals the algebraic product of the membership function A and B.	$\mu_{A \cdot B}(x_a) \equiv \mu_A(x) \cdot \mu_B(x)$
Power of a fuzzy set	The β power of A (A^β) has the equivalence to linguistically modify the set with *VERY*.	$\mu_{A^\beta}(x) \equiv [\mu_A(x)]^\beta$
Concentration	Squaring the set is called *concentration* or *CON*.	$\mu_{CON(\beta)}(x) \equiv (\mu_A(x))^2$
Dilation	Taking the square root is called *dilation* or *DIL*.	$\mu_{DIL(A)}(x) \equiv \sqrt{\mu_A(x)}$

Table 4.2: The most important fuzzy operations.

4.3 Membership Function

Membership function represents the degree of truth in fuzzy logic. Membership functions characterize fuzziness (i.e., all the information in fuzzy set), whether the elements in fuzzy sets are discrete or continuous, and they are represented by graphical forms, as shown in Figure 4.3. Rules for defining fuzziness are fuzzy too.

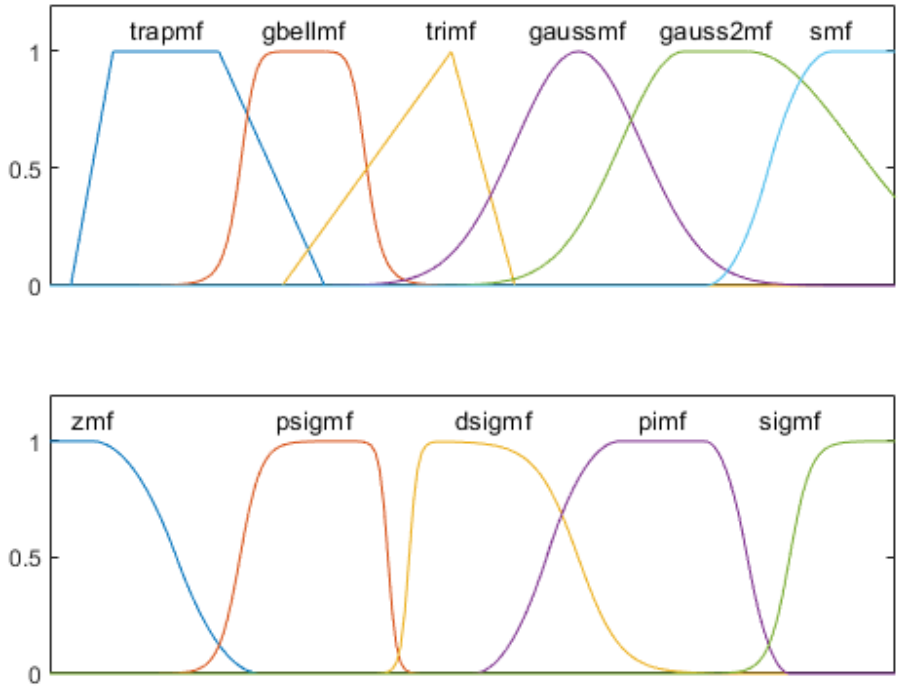


Figure 4.3: Examples of membership functions.

All membership function types are described in Table 4.3.

Membership Function Type	**Description**
gbellmf	Generalized bell-shaped membership function
gaussmf	Gaussian membership function
gauss2mf	Gaussian combination membership function
trimf	Triangular membership function
trapmf	Trapezoidal membership function
sigmf	Sigmoidal membership function
dsigmf	Difference between two sigmoidal membership functions
psigmf	Product of two sigmoidal membership functions
zmf	Z-shaped membership function
pimf	Pi-shaped membership function
smf	S-shaped membership function
constant	Constant membership function for Sugeno output membership functions
linear	Linear membership function for Sugeno output membership functions

String or character vector	Name of a custom membership function in the current working folder or on the MATLAB path. Custom output membership functions are not supported for Sugeno systems
Function handle	Handle to a custom membership function in the current working folder or on the MATLAB path. Custom output membership functions are not supported for Sugeno systems.

Table 4.3: Fuzzy membership functions.

Membership functions have different features, graphically shown in the Figure 4.4:

- *Core*: for any fuzzy set $\widetilde{A}$, the core of a membership function is that region of universe that is characterize by full membership in the set. Hence, core consists of all those elements y of the universe of information such that,

$$\mu_{\widetilde{A}}(y) = 1;$$

- *Support*: for any fuzzy set $\widetilde{A}$, the support of a membership function is the region of universe that is characterize by a nonzero membership in the set. Hence core consists of all those elements y of the universe of information such that,

$$\mu_{\widetilde{A}}(y) > 0;$$

- *Boundary*: for any fuzzy set $\widetilde{A}$, the boundary of a membership function is the region of universe that is characterized by a nonzero but incomplete membership in the set. Hence, core consists of all those elements y of the universe of information such that,

$$1 > \mu_{\widetilde{A}}(y) > 0;$$

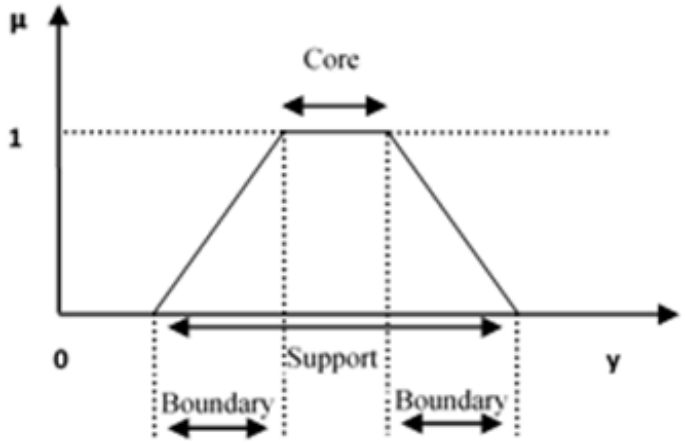


Figure 4.4: Features of membership function.

4.4 Fuzzy Inference Process

The fuzzy inference process is the set of deduction rules that must be applied to a given system to obtain results through the use of fuzzy logic. It can be divided into the following phases:

1. *Fuzzification*: it may be defined as the process of transforming a crisp set to a fuzzy set or a fuzzy set to fuzzier set. Basically, the current values are applied to the membership functions to determine the degree of truth of each rule of the premise;
2. *Inference*: calculated the truth value for the premises, the result is applied to the final part of each rule. The results obtained in a fuzzy subset must be assigned to each output variable for each rule. To do this, only *min* or *product* rules are usually used. In the first rule the output membership function is "truncated" to the degree of truth calculated from the rule of the premise, while in the second rule the exit membership function is multiplied by the degree of truth calculated by the rule of the premise.
3. *Composition*: all fuzzy subsets assigned to each output variable are combined together to form a single fuzzy subset for each output variable. Usually *max* or *sum* are used. In the *max* composition, the fuzzy subset obtained is constructed by taking the maximum value among all those obtained from the

fuzzy subsets assigned to a variable by the inference rule (logical function OR), while in the *sum* composition the combined output of the fuzzy subset is constructed by taking the sum of the maximum values obtained from all the fuzzy subsets assigned to an output variable from the inference rule.

4. *Defuzzification*: it may be defined as the process of reducing a fuzzy set into a crisp set or to convert a fuzzy member into a crisp member. There are may defuzzification methods [38] (see Figure 4.5) like:
   - *Centroid*: also known as *center of gravity* or *center of area*, it returns the center of area under the curve. This is the most commonly used technique; the only disadvantage of this method is that it is computationally difficult for complex membership functions;
   - *Bisector*: it is the vertical line that will divide the region into two subregions of equal area. It is sometimes, but not always, coincident with the centroid line;
   - *Middle*, *Smallest* and *Largest of Maximum*: MOM, SOM, and LOM stand for Middle, Smallest, and Largest of Maximum, respectively. These three methods key off the maximum value assumed by the aggregate membership function. If the aggregate membership function has a unique maximum, then MOM, SOM, and LOM all take on the same value.

Generally the centroid method is the most used.

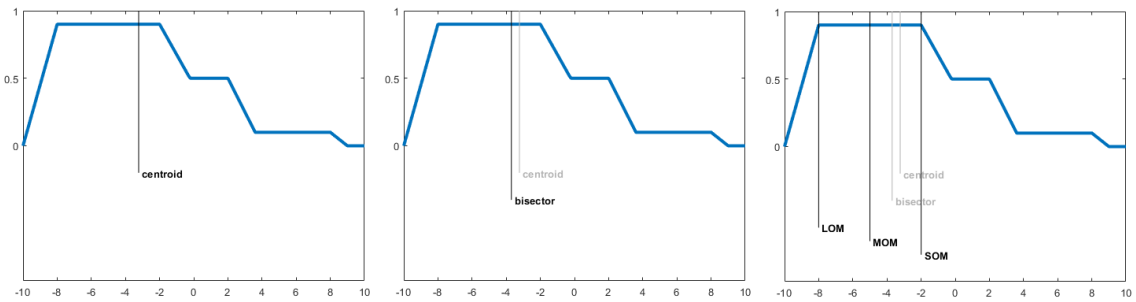


Figure 4.5: Defuzzyfication methods.

4.5 Fuzzy Rules

Fuzzy rules are used to infer an output based on input variables. A rule is in the form:

- Premise: x is A;
- Implication: IF x is A THEN y is B;
- Consequent: y is B.

In crisp logic, the premise "x is A" can only be true or false. However, in a fuzzy rule, the premise "x is A" and the consequent "y is B" can be true to a degree, instead of entirely true or entirely false. This is achieved by representing the linguistic variables A and B using fuzzy sets.

Rules can connect multiple variables through fuzzy set operations using AND, OR and NOT operators.

Chapter 5

Fuzzy Inference System

The variable α must be classified taking into account some qualitative attributes and it may has varying levels of validity between a maximum (1) and a minimum (0). For this reason, it is necessary to generate linguistic rules of fuzzy inference to realize a mapping of the inputs on the desired output.

Fuzzy Inference System (FIS) is the key unit of a fuzzy logic system having decision making as its primary work. It uses the "IF...THEN" rules along with connectors AND, OR and NOT for drawing essential decision rules. Following are some characteristics of FIS:

- the output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp;
- it is necessary to have fuzzy output when it is used as a controller;
- a defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables.

The five functional blocks shown in Figure 5.1 give an overview of the construction of FIS.

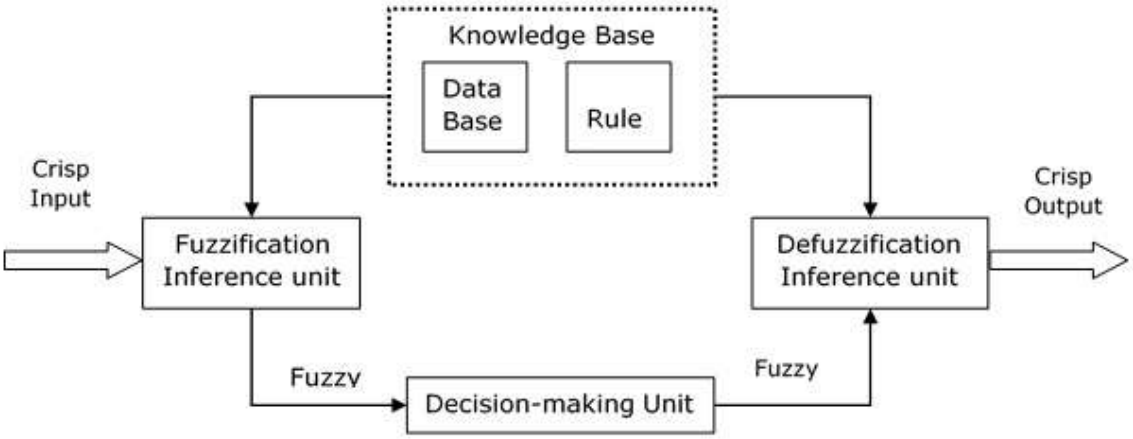


Figure 5.1: Functional blocks of FIS.

- *Database*: it contains fuzzy IF-THEN rules;
- *Rule Base*: it defines the membership functions of fuzzy sets used in fuzzy rules;
- *Decision-making Unit*: it performs operation on rules;
- *Fuzzification Interface Unit*: it converts the crisp quantities into fuzzy quantities.;
- *Defuzzification Interface Unit*: it converts the fuzzy quantities into crisp quantities.

5.1 Methods of FIS

There are two methods of FIS, having different consequent of fuzzy rules: *Mamdani Fuzzy Inference System* and *Takagi-Sugeno Fuzzy Model*.

5.1.1 Mamdani FIS

In this work, *Mamdami* FIS [39] has been chosen to build a simple fuzzy inference system, which consists of a minimum number of variables, and efficiently solves the HRC problem. In particular, the fuzzy inference process has been developed as a two-input, one-output, three-rule problem, as shown in Figure 5.3. The whole fuzzy inference system has been developed with the *Fuzzy Logic Designer* of MATLAB (see Figure 5.2), which generates a `.fis` file.

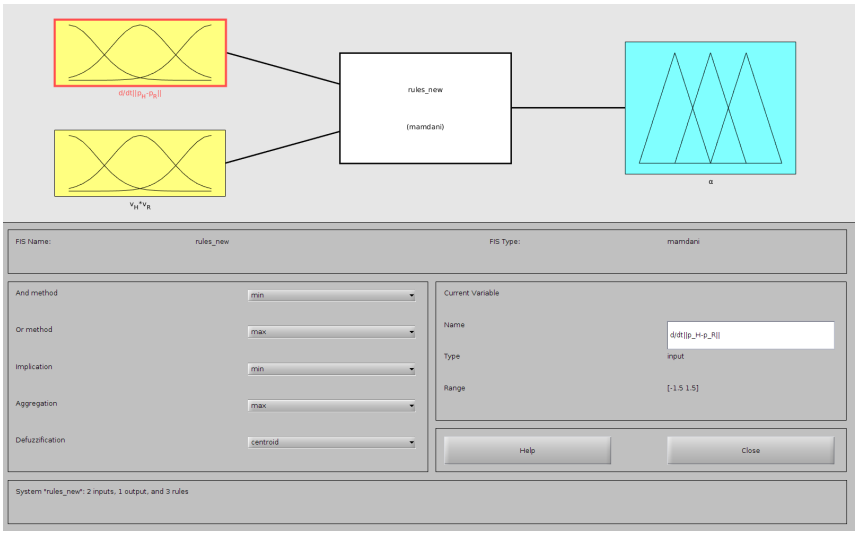


Figure 5.2: Fuzzy Logic Designer.

Below, Mamdami FIS is directly explained with the implementation of the work.

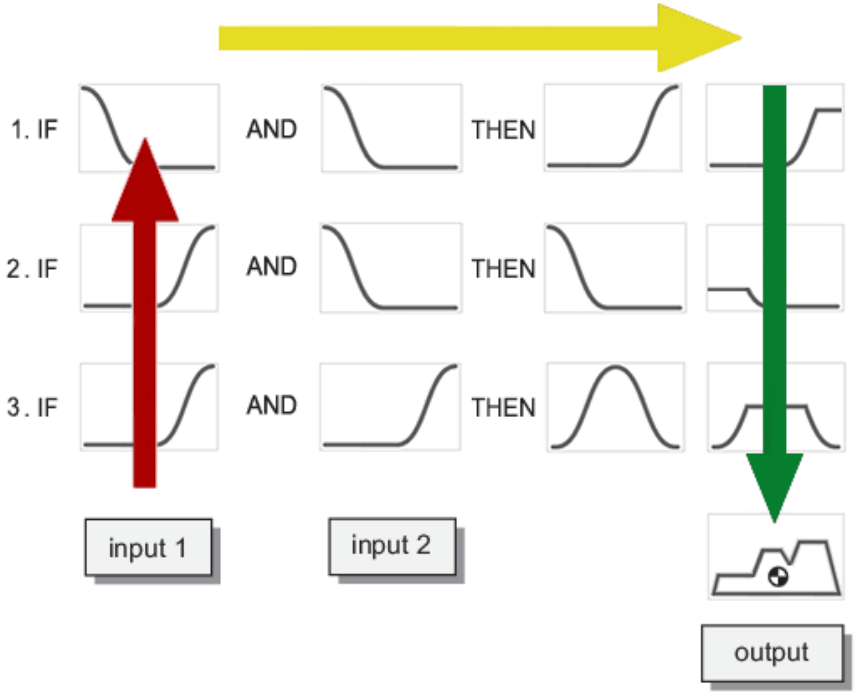


Figure 5.3: Fuzzy inference system.

The first step is to select the inputs. Two data inputs have been selected:

1. the time derivative of the distance between human and robot, i.e., $\dot{d} = \frac{\mathrm{d}\|\hat{\boldsymbol{p}}_H - \boldsymbol{p}_R\|}{\mathrm{d}t}$;
2. the scalar product between the robot and the human velocity vectors, i.e., $\dot{\boldsymbol{p}}_R^T \hat{\dot{\boldsymbol{p}}}_H$.

The first input is useful to distinguish cases when the operator and the robot are getting closer and cases when they are moving away from each other. The scalar product specifies the relative direction of travel of the operator and the robot.

The next step is the *fuzzification step* (red arrows of Figure 5.3). The ranges of variability of each input have been defined, and the appropriate membership function of each interval has been selected. This step requires attention to correctly determine the degree to which the input belongs to each of the appropriate fuzzy set, by assigning a fuzzy degree of membership in the interval from 0 to 1. Two membership functions have been selected to represent positive (P) and negative (N) values, a Z-shape and a S-shape, respectively. These functions, with different parameters, have been chosen to describe both $\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H$ and $\dot{d}$. More detail about the input ranges and membership functions are described in Chapter 7.

After the inputs are fuzzified, the *implication step* (yellow arrows of Figure 5.3) determines the degree to which each part of the antecedent is satisfied for each rule. The antecedent of the developed fuzzy inference rules has three parts, combined through an AND method *min* to obtain an implicated number that represents the result of the rule antecedent. Each rule is designed to represent one possible and distinguishable scenario.

Since the final decision is based on the result of all the tested rules, the outputs of the rules must be combined in some way. The *aggregation step* (green arrow of Figure 5.3) is the process by which the fuzzy sets representing the outputs of each rule are combined into a single fuzzy set, before the last *defuzzification step*. For each interval of the consequent, the maximum value of the fuzzy set is chosen and the defuzzification method is the calculation of the centroid, which returns the center of the area under the curve, as shown at the end of Figure 5.3.

The output value, α, has been generated by analyzing different possible risk situations, with the aim both to avoid any collisions between human and robot, and to be aligned with current ISO/TS 15066. The three rules are shown in Table 5.1.

antecedent		consequent
$\dot{d}$	$\dot{\boldsymbol{p}}_R^T \hat{\dot{\boldsymbol{p}}}_H$	α
N	$\sim$	H
P	N	S
P	P	M

Table 5.1: Fuzzy rules: [S] Small, [M] Medium, [H] High, [N] Negative, [P] Positive, [$\sim$] any.

Note that the scalar product between the operator velocity and the robot velocity (second input) is a complementary information to the derivative with respect to the time of the distance between human and robot (third input). Since $\dot{\boldsymbol{p}}_R^T \hat{\dot{\boldsymbol{p}}}_H = \|\dot{\boldsymbol{p}}_R\| \|\hat{\dot{\boldsymbol{p}}}_H\| \cos\theta$, when $\theta = 180°$, a critical situation is possible. The result of the scalar product is a negative value, $\dot{\boldsymbol{p}}_R^T \hat{\dot{\boldsymbol{p}}}_H < 0$, but it is not possible to distinguish the cases shown in Figure 5.4, in which the directions are opposite but it is not known if the human and the robot are getting closer or are moving away from each other. This is the reason why it is necessary to combine the scalar product information with the time derivative of the distance between the human operator and the robot.

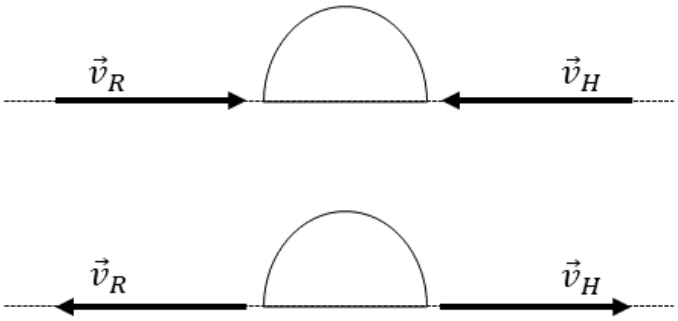


Figure 5.4: Problem of the scalar product.

5.1.2 Takagi-Sugeno FIS

In *Takagi-Sugeno* FIS [40], a typical rule has the form:

$$\text{IF } x \text{ is } A \text{ and } y \text{ is } B \text{ THEN } z = f(x, y)$$

Here, A and B are fuzzy sets in antecedents and $Z = f(x, y)$ is a crisp function in the consequent.

About the fuzzy inference process, it works in the following way:

- Step 1: fuzzifying the inputs. Here, the inputs of the system are made fuzzy;
- Step 2: applying the fuzzy operator. In this step, the fuzzy operators must be applied to get the output.

5.1.3 Comparison between the two Methods

The main differences between Mamdani and Sugeno are shown below:

- Output Membership function: the main difference between them is on the basis of output membership function. The Sugeno output membership functions are either linear or constant;
- Aggregation and Defuzzification procedure: the difference between them also lies in the consequence of fuzzy rules and due to the same their aggregation and defuzzification procedure also differs;
- Mathematical rules: more mathematical rules exist for the Sugeno rule than the Mamdani rule;
- Adjustable parameters: the Sugeno controller has more adjustable parameters than the Mamdani controller.

Therefore, the choise of using Mamdami FIS to compute α is due in wanting a scalar as output and not a function.

5.2 ROS-Simulink interface

Once the `.fis` is generated by the Fuzzy Logic Designer, it has been imported in Simulink to allow the ROS-Simulink interface. The Simulink scheme is shown in Figure 5.5. The following blocks have been used:

- ROS Subscribe, Publish and Blank Message;
- Enabled Subsystem;
- Bus Selector;
- Data Type Conversion

- MATLAB Function;
- Mux;
- Fuzzy Logic Controller with Ruleviewer;
- Bus Assignment.

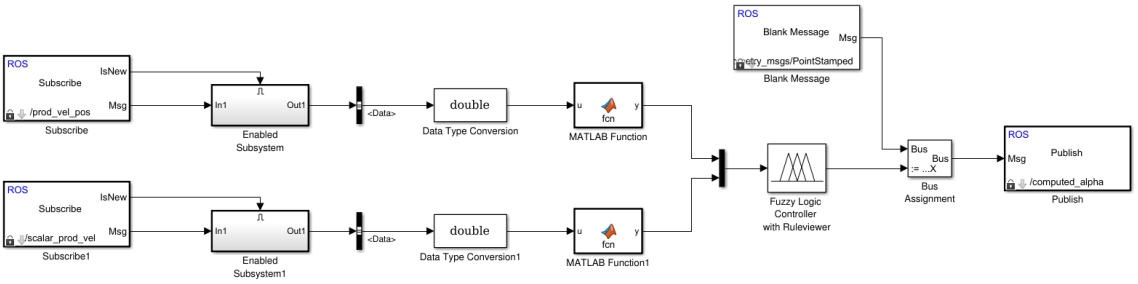


Figure 5.5: Implementation of Simulink scheme.

Subscribe, Blank Message and Publish are part of *Robotics System Toolbox* [41]. Robotics System Toolbox provides algorithms and hardware connectivity for developing autonomous robotics applications for aerial and ground vehicles, manipulators, and humanoid robots. Toolbox algorithms include path planning and path following for differential drive robots, scan matching, obstacle avoidance, and state estimation. For manipulator robots, the system toolbox includes algorithms for inverse kinematics, kinematic constraints, and dynamics using a rigid body tree representation. The system toolbox provides an interface between MATLAB and Simulink and ROS that enables you to test and verify applications on ROS-enabled robots and robot simulators. Robotics System Toolbox is useful for the study and simulation of:

- Classical arm-type robotics: kinematics, dynamics, and trajectory generation. The Toolbox uses a very general method of representing the kinematics and dynamics of serial-link manipulators using Denavit-Hartenberg parameters or modified Denavit-Hartenberg parameters. Operations include forward kinematics, analytic and numerical inverse kinematics, graphical rendering, manipulator Jacobian, inverse dynamics, forward dynamics and simple path planning. It can operate with symbolic values as well as numeric, and provides a Simulink blockset;

- Ground robots: standard path planning algorithms, lattice planning, kino-dynamic planning (RRT), localization (EKF, particle filter), map building and simultaneous localization and mapping, and a Simulink model of a non-holonomic vehicle;
- Flying quadrotor robots.

Two *ROS Subscribe* blocks have been used, one for each input variable of fuzzy inference system ($\dot{d}$ and $\dot{\boldsymbol{p}}_R^T\hat{\boldsymbol{p}}_H$), to acquire the respective values. A *ROS Subscribe* block allows to enter the topic both manually, by writing its name and the message type, or by selecting it from ROS network. The latter option is available only if MATLAB is connected to the ROS master and the ROS network is running; in this implementation the topics have been selected from ROS network. A *ROS Subscribe* block has two outputs:

- *Msg*: this output is a ROS message (bus signal);
- *IsNew*: this output is a boolean indicating whether a message was received during the previous time step: when *IsNew* is true, *Msg* holds the newly-received message, when *IsNew* is false, *Msg* holds the last received message.

Both the ROS Subscribe blocks have a sample time set to -1 because in this way they generate an output whenever a message arrives as input.

The *Enabled Subsystem* block allows to minimize the computation time of the entire implementation. As can be seen in Figure 5.5, two *Enabled Subsystem* blocks have been used.

Since the *Msg* output of the *ROS Subscribe* block is a bus signal, two *Bus Selector* blocks have been used to extract the useful signal. Actually, in this implementation this signal is unique for a block, but in any case the *Bus Selector* block must be used to allow the subsequent operations.

For fuzzy logic, numerical variables are needed as input, therefore two *Data Type Conversion* have been used, one for each bus signal, to convert them in double data type.

Next, two *MATLAB Function* blocks have been used to manage both NaN values, e.g., when there are no operators in the scene, and input values which are smaller than the minimum acceptable value ($min_{\dot{d}}$ for $\dot{d}$ and $min_{\dot{\boldsymbol{p}}_R^T\hat{\boldsymbol{p}}_H}$ for $\dot{\boldsymbol{p}}_R^T\hat{\boldsymbol{p}}_H$) or

higher than the maximum one ($max_{\dot{d}}$ for $\dot{d}$ and $max_{\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H}$ for $\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H$). In detail:

$$\dot{d} = \begin{cases} max_{\dot{d}} & \text{if} \quad \dot{d} = NaN \vee \dot{d} > max_{\dot{d}} \\ min_{\dot{d}} & \text{if} \quad \dot{d} < min_{\dot{d}} \\ \dot{d} & \text{otherwise} \end{cases} \tag{5.1}$$

$$\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H = \begin{cases} min_{\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H} & \text{if} \quad \dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H = NaN \vee \dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H < min_{\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H} \\ max_{\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H} & \text{if} \quad \dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H > max_{\dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H} \\ \dot{\boldsymbol{p}}_R^T\hat{\dot{\boldsymbol{p}}}_H & \text{otherwise.} \end{cases} \tag{5.2}$$

Note that, when no operator is in the scene, the previous relations make a low α value (see Table 5.1).

The outputs of this two blocks have been put into a *Mux* (Multiplexer), whose output is the input of *Fuzzy Logic Controller with Ruleviewer*. The *Fuzzy Logic Controller with Ruleviewer* block implements a fuzzy inference system in Simulink and displays the fuzzy inference process in the Rule Viewer during the simulation. An example is shown in Figure 5.6.

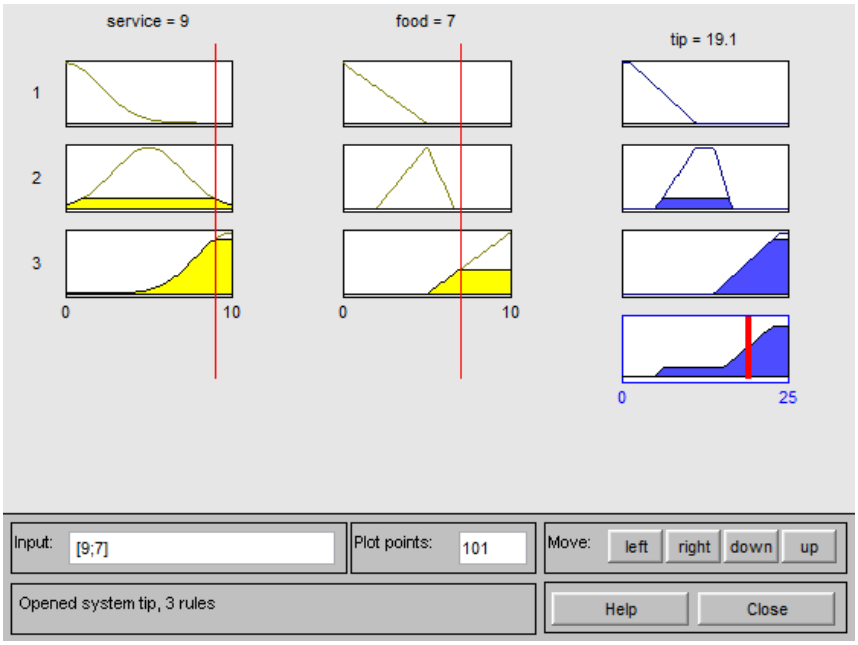


Figure 5.6: Fuzzy Logic Controller with Ruleviewer: Interactive mode.

The *Fuzzy Logic Controller with Ruleviewer* block:

- only supports double-precision data;
- uses 101 points for discretizing output variable ranges;
- only supports interpreted execution simulation mode;
- does not have additional output ports for accessing intermediate fuzzy inference results.

To make the *Fuzzy Logic Controller with Ruleviewer* work, the `.fis` file name must be specified. As refresh rate, i.e., the rate, in second, which allows the viewer, during the simulation, to display updates at the specified speed to show the inference process for the last values of the input signal, the default one has been chosen (i.e., 2 s).

The computation of S (Equation 4.1) has been done in a ROS node, so the α value must be transferred in ROS. To publish the α value on a ROS topic, the *ROS Publish* block has been used; it takes in as its input a Simulink nonvirtual bus that corresponds to the specified ROS message type and publishes it to the ROS network. It uses the node of the Simulink model to create a ROS publisher for a specific topic. This node is created when the model runs and is deleted when the model terminates. If the model does not have a node, the block creates one. For simulation, this input is a MATLAB ROS message; in code generation, it is a C++ ROS message. The procedure for entering the topic name is the same as the one described before for *ROS Subscribe*.

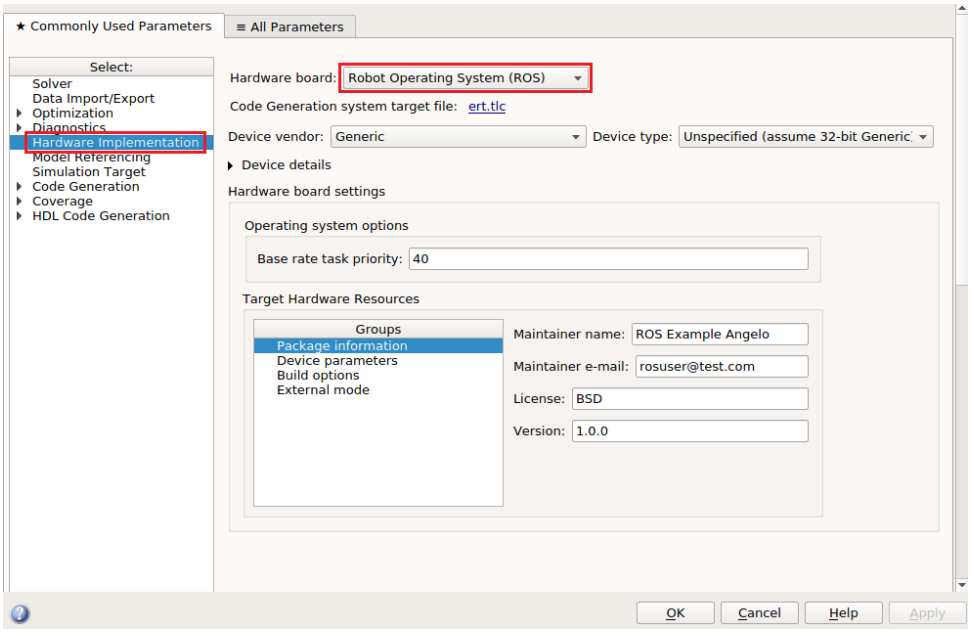
Since the *ROS Publish* block accepts a ROS message, i.e., a bus signal as input, it has been necessary to use a *Blank Message* and *Bus Assignment* block. The *Blank Message* block served to create a blank message with the specified message type (`geometry_msgs/PointStamped` in this implementation), so the output of this block is a bus signal; the Sample time has been set to inf, which is the default value indicating that the block output never changes. This output has been put into a *Bus Assignment* with the output of *Fuzzy Logic Controller with Ruleviewer* block, to assign a new value to the signal (i.e., α) on the bus, (i.e, `geometry_msgs/PointStamped.Point.X`). Finally, the *Simulation stop time* has been set equal to *inf* to allow the scheme to work for the whole duration of the

experiment.

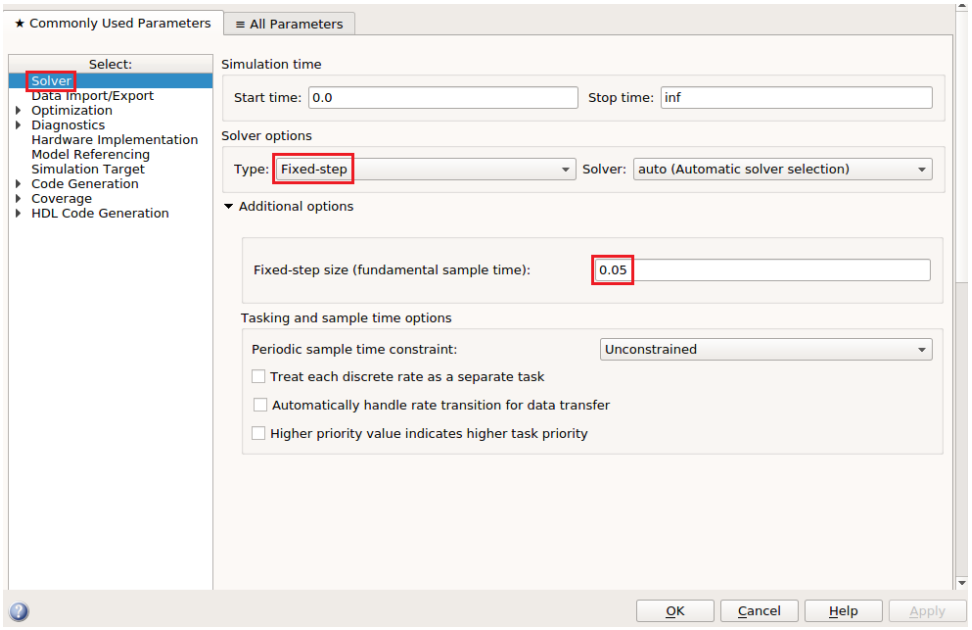
A ROS node from Simulink has been created through the *Build Model* option [42]. This option allows to configure a model to generate C++ code for a standalone ROS node. The guide of this configuration is described below:

- Click on *Model Configuration Parameters* icon;
- In the *Hardware Implementation* pane, set *Hardware board* to *Robot Operating System* (ROS) (see red rectangles in Figure 5.7(a)). The *Hardware board* settings section contains settings specific to the generated ROS package, such as information to be included in the package `.xml` file;
- In the *Solver* pane, in *Solver options* the *Type* must be set to *Fixed-step*, and *Fixed-step size* must be set to 0.05 (see red rectangles in Figure 5.7(b)). In generated code, the *Fixed-step* size defines the actual time step, in seconds, that is used for the model update loop. However, it can be made smaller;

This procedure has been made to make the whole system operating independent from MATLAB and Simulink. In fact, with the ROS node generation, only the ROS environment is required to run the system, and the FIS parameters can be modified from the ROS node directly.



(a) Hardware implementation pane.



(b) Solver pane.

Figure 5.7: Build Model configuration.

Chapter 6

Trajectory Scaling

SSM scenarios usually sacrifice the production time because a lot of time is spent in low speed mode when a human operator is inside the collaborative workspace. On the contrary, the proposed strategy ensures human-robot coexistence according to the standard regulations, and also guarantees the task efficiency by using a time-scaling approach to change robot operating speed without introducing acceleration discontinuities.

A typical industrial pre-programmed task, $\mathcal{T}$, is composed by N positions, $\tilde{q}_i$, associated to velocities $\dot{\tilde{q}}_i$, accelerations $\ddot{\tilde{q}}_i$ and time instants $\tilde{t}_i$ with $i = 1, \ldots, N$. Typically, the pre-programmed joint positions have to be interpolated according to the sampling time T_c required by the robot control interface. In this work a quintic interpolation is used, i.e., the planned interpolated trajectory is

$$\tilde{q}_h = p_5(t_h;\, \mathcal{T}) \tag{6.1}$$

$$\dot{\tilde{q}}_h = p_4(t_h;\, \mathcal{T}) \tag{6.2}$$

$$t_{h+1} = t_h + T_c, \tag{6.3}$$

where t_h is the h-th discrete time instant, p_4 is the derivative of the polynomial p_5, $\tilde{q}_h$ and $\dot{\tilde{q}}_h$ are the *planned* joint position and velocity at time t_h, respectively.

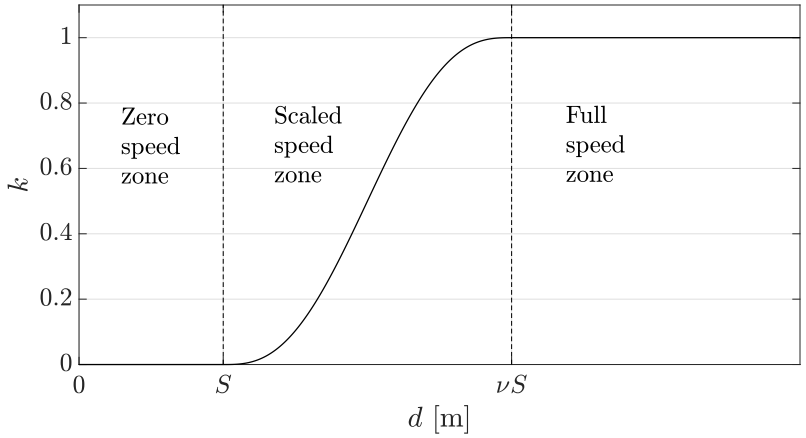


Figure 6.1: Relation between d and k.

The proposed algorithm modulates the robot speed by scaling the time with a *safety scale factor* k, which can assume values in the interval $[0, 1]$. The scale factor is related to d (Section 3.4) as shown in Figure 6.1. When d is below the minimum protective distance S, k is 0 and the robot stops (according to the regulation). When the distance d is far from S, i.e., $d > \nu S$ ($\nu > 1$), the robot can move at full speed to improve the production time. Between S and νS the function in Figure 6.1 smoothly varies to avoid acceleration discontinuities. Obviously, ν is another design parameter that change the size of the scaled speed mode zone.

Practically, the trajectory is scaled computing Equation 6.1 using a scaled time τ_h, i.e.,

$$q_h = p_5(\tau_h; \mathcal{T}) \tag{6.4}$$

$$\tau_{h+1} = \tau_h + kT_s \tag{6.5}$$

where q_h is the actual joint commands at time t_h. Obviously the joint command q_h, as well as the scaled time τ_h, are generated with sampling time T_s.

This approach effectively scales the joints velocities. In fact, using Equation 6.5,

$$\dot{\tau} \approx \frac{\tau_{h+1} - \tau_h}{T_s} = k \tag{6.6}$$

so, by deriving Equation 6.4,

$$\dot{q}_h = p_4(\tau_h; \mathcal{T})k. \tag{6.7}$$

This means that the velocity is scaled by the *safety factor* k.

This approach guarantees that the task $\mathcal{T}$ remains the same in position, but, simultaneously, the resulting velocity is scaled according to k.

When the operator is going to be into a dangerous situation, the robot operates at diminished capacity with limits on velocity that respect human robot collaboration norms, until the safety restoration. Notice that the side effect of the velocity reduction is the reduction of the minimum protective distance S, since this value is proportional to the robot velocity. Experimental results are shown in Chapter 7.

Chapter 7

Experimental results and Validation

This section shows an example of experimental results obtained by simulating an SSM human-robot collaboration task inside the collaborative workspace of Figure 3.5. A manufacturing industrial sealing operation has been virtually realized: the robot executes a pre-planned path at a given nominal speed, while, suddenly, a human operator enters the collaborative workspace to perform some manual operation close to the robot, at different distances.

The main goal of this experiment is to prove the efficiency of the fuzzy inference approach into industrial applications to better handle the production time and, at the same time, to guarantee the safety of the operators when they are inside the collaborative workspace.

Table 7.1 sums up the used hardware and experimental case study.

Robot	Yaskawa SIA5F
Collaborative workspace	4x2 m
Depth camera (1)	Microsoft Kinect v1
Depth camera (2)	Intel RealSense D435
Robot simulated task	Sealing operation
Operator simulated task	Manual piece change

Table 7.1: Case study and available hardware.

The covariance matrix $\boldsymbol{Q_2}$ of Equation 3.9 has been chosen as

$$\boldsymbol{Q}_2 = \text{diag}(0.02,\ 0.05,\ 0.05)\ \text{m}^2/\text{s}^2, \tag{7.1}$$

while the noise covariance has been estimated as

$$\boldsymbol{N} = \text{diag}(0.0009,\ 0.0008,\ 0.001)\ \text{m}^2. \tag{7.2}$$

The description of how the parameters to compute the protective separation distance S of Equation 1.4 and Equation 4.1 have been obtained is present below.

To compute the time required by the robot system to respond to the operator's presence, T_R, the rate of the topic on which the value of the variable d is published has been observed because d is a finite value when someone enters the scene. The rate of this topic is about 10 Hz.

About T_S, the response time of the machine which brings the robot to a safe, controlled stop, of the joint L has been computed. The robot, completely extended in a vertical position, has been stopped after having performed a trajectory maximum speed (i.e., *safety scale factor* k equal to 1) forming a 90° angle, as shown in Figure 7.1.

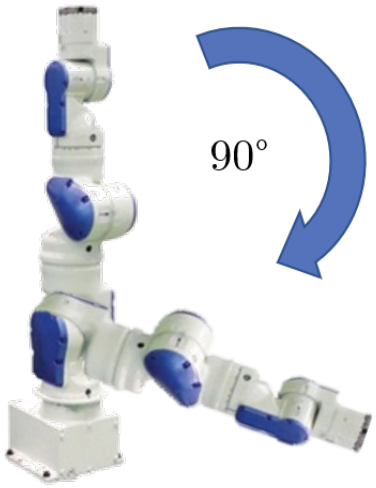


Figure 7.1: Robot configurations.

The choice to position the robot in a vertical configuration is due to the worst case evaluation, and moving only the joint L, kinematic singularity has been avoided. The trajectory has been recorded in order to be able to evaluate T_S by means of a `.bag` file.

To compute the Euclidean distance travelled by the robot while braking, B, the same test has been carried out for the computation of T_S in order to assess here too the worst case.

The absolute accuracy of the robot, Z_R, i.e., the deviation or error between position achieved with the assigned posture and position calculated by forward kinematics. The absolute accuracy of the robot has typical values between 0.2 and 1 mm and they are not present in any datasheet.

The sensor uncertainty Z_S has been calculated as the sum of Microsoft Kinect v1 and Intel RealSense D435 uncertainties. For both depth cameras, a 2D point of pixel coordinates has been taken from the depth image viewer, obtained the z Cartesian coordinate and 2D point of Cartesian coordinates has been calculated from 2D point of pixel coordinates, from z and from the intrinsic parameters of the used depth camera. Finally, the norm of the point in Cartesian coordinates has been computed. The sum of the respective norms has established the value of Z_S.

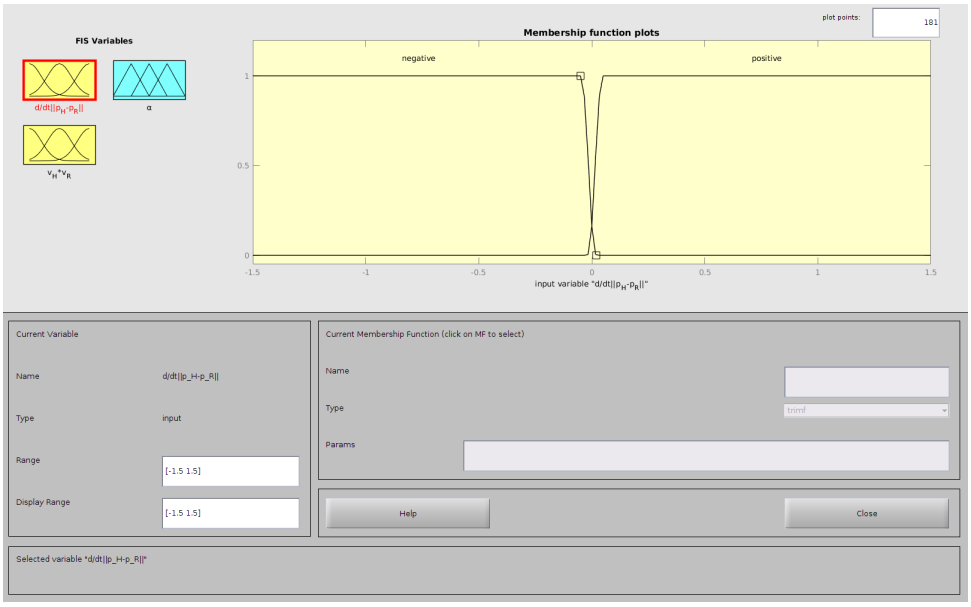
The intrusion distance C, based on the operator reach, has been chosen to better appreciate the zero speed zone.

The values of all these parameters are reported in Table 7.2.

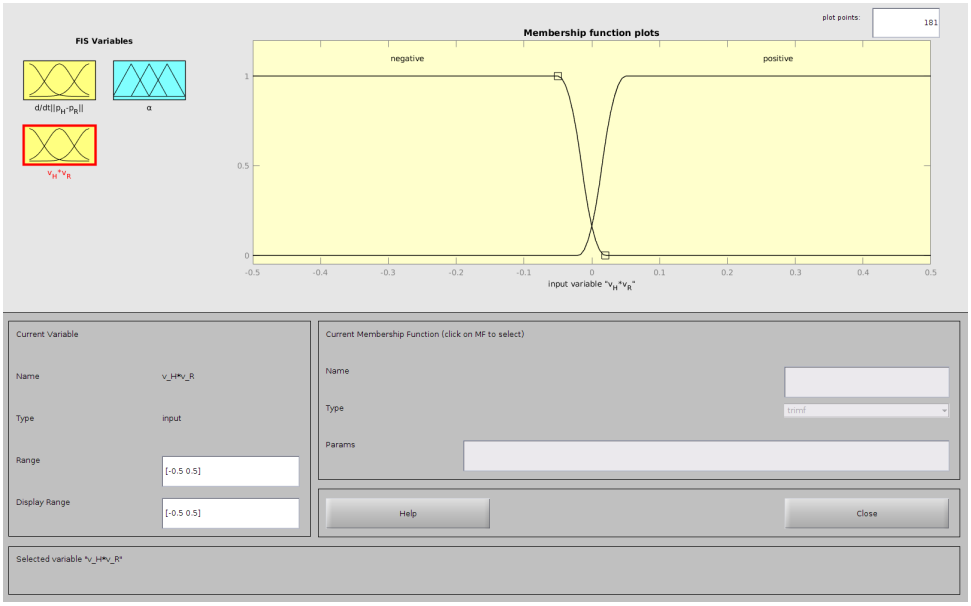
T_R	0.10 s
T_S	0.08 s
B	0.563 mm
Z_R	0.001 m
Z_S	0.1067 m
C	0.20 m

Table 7.2: Constant parameters of S.

About the membership functions and the ranges of FIS input variables, in Figures 7.2(a) and 7.2(b) are shown the choises made for $\dot{d}$ and $\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H$, respectively. In Figure 7.3, instead, are shown the choises made for α.



(a) $\dot{d}$ range and membership functions.



(b) $\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H$ range and membership functions.

Figure 7.2: FIS input variables.

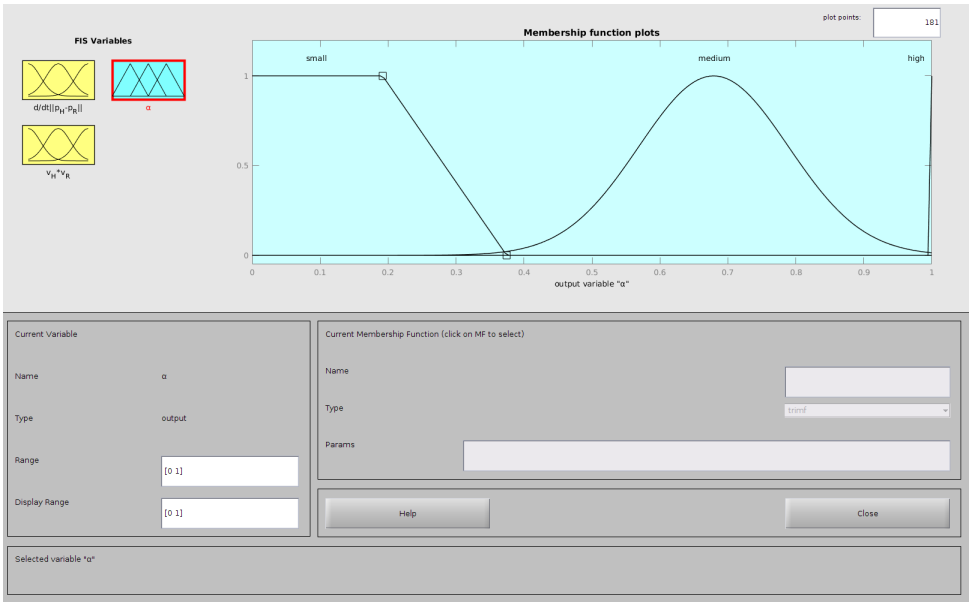


Figure 7.3: FIS output variable.

Figures 7.2 and 7.3 show that:

- $\dot{d} \in [-1.5, 1.5]$;
- $\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H \in [-0.5, 0.5]$.

Therefore, with reference to Equations 5.1 and 5.2:

- $min_{\dot{d}} = -1.5$ m/s;
- $max_{\dot{d}} = 1.5$ m/s;
- $min_{\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H} = -0.5$ m^2/s^2;
- $max_{\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H} = -0.5$ m^2/s^2.

Figure 7.4 shows the results of the experiment.

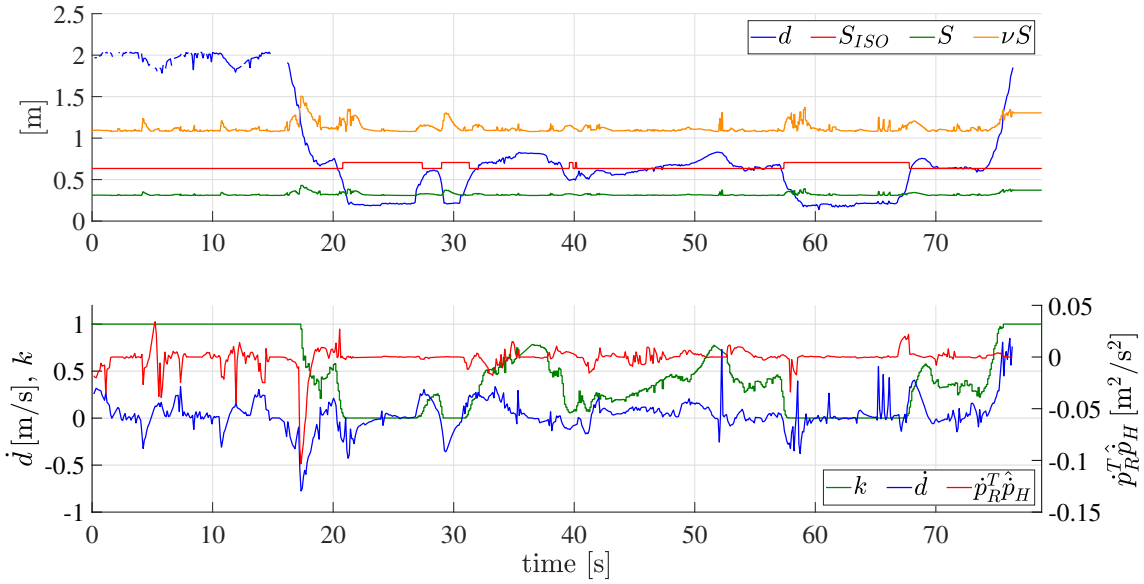


Figure 7.4: Experiment: An operator enters the shared workspace while the robot is moving. The top plot shows the estimated distance robot-operator (d), the protective distances proposed by the regulation without sensing (S_{ISO}) and the protective distance proposed by the work (S). The bottom plot shows the trajectory scaling factor k, the time derivative of the distance $\dot{d}$ and the velocity scalar product.

The graph at the top of the Figure shows the distance between the human operator and the robot and it can be compared with the minimum protective distance computed as in Equation 1.4 (S_{ISO} in the legend) and the two thresholds proposed in this work: S in the legend is the protective distance computed as in Equation 4.1 and νS is the threshold used in the trajectory scaling algorithm (see Chapter 6). In this experiment, the ν parameter has been set to 350% of S. The bottom plot of Figure 7.4 shows the two inputs of the fuzzy inference system ($\dot{d}$ and $\dot{\boldsymbol{p}}_R^T \hat{\boldsymbol{p}}_H$) and the trajectory scale factor k. In this experiment S_{ISO} is not used and it is showed in the plot for comparison purposes.

The robot executes a planned task, suddenly (at about 16 s) an operator enters into the workspace simulating a manual piece change task. This is visible in the top plot of Figure 7.4, where the human-robot distance decreases. Note that for almost the whole task duration the separation distance robot-operator is below the S_{ISO} signal, this would have caused frequent starts and stops of the robot. Instead, through the proposed trajectory scaling algorithm, the robot reduces its velocity

according to the observed separation distance. This is visible in the k signal of the bottom plot that varies according to d. Notice that k goes to 0 only when the distance d goes below the protective distance S.

Moreover, another property of the proposed solution is that S increases only when the distance decreases (i.e., when $\dot{d} < 0$) and not when the distance increases. This is due to the computation of the directed speed and the fuzzy rules.

The shown experiment demonstrates how the proposed approach guarantees a safe human-robot coexistence in the collaborative workspace. This is achieved in accordance with the ISO/TS regulations and, simultaneously, minimizing dead times in the production process.

Chapter 8

Conclusion and Future developments

The human-robot interaction and their intentions to compete or cooperate in collaborative workspaces are challenging research fields. The purpose of this work is to improve the current regulations both to maximize the production time and guarantee the safety of human operators inside the shared workspace. In this paper, the expected human movements relative to the robot are classified to identify all possible industrial SSM scenarios from which fuzzy control rules for the robot reactions are derived. The time schedule of the information flow and their processing are presented to discuss the novel approach and its efficiency. Collisions between robot and human operators are avoided by identifying human-robot intersections through a detection algorithm which processes data obtained from merging of two depth camera images. In the context of the recognition of human approach to a moving robot, travel directions are modeled by fuzzy inference logic. Results obtained from experimental data show the applicability of the presented methods to many common manufacturing industry applications.

For future developments, sensors with better quality than the ones considered in this work or in higher number and of different technlogies could be used in order to improve precision and accuracy of human detection. For example, a thermal camera could be used to allow the humans recognition from other objects in the scene. In this way, the condition $\nu S < d < S$ in which the robot velocity is scaled (Chapter 6) would not be valid for the objects but only for the human operators.

References

[1] LABOR, "Lean robotized AssemBly and cOntrol of composite aeRostructures." [Online]. Available: https://www.labor-project.eu/

[2] "Machinery Directive 2006/42/EC," European Parliament, 2006. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32006L0042

[3] "Safety of machinery - General principles for design - Risk assessment and risk reduction." International Organization for Standardization, Technical report, 2010.

[4] "Robots and robotic devices - Safety requirements for industrial robots. Part 2: Robot system and integration." International Organization for Standardization, Technical report, 2011.

[5] "Industrial safety." International Organization for Standardization, Technical report, 2015.

[6] "Robots and robotic devices - collaborative robots." International Organization for Standardization, Technical report, 2016.

[7] "Robots and robotic devices - Safety requirements for industrial robots. Part 1: Robots." International Organization for Standardization, Technical report, 2011.

[8] "Safety of machinery - Electrical equipment of machines - Part 1: General requirements." International Electrotechnical Commission, Tech. Rep., 2009.

[9] A. Bicchi, M. A. Peshkin, and J. E. Colgate, "Safety for physical human–robot interaction," in *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008, pp. 1335–1348.

[10] J. Heinzmann and A. Zelinsky, "Quantitative safety guarantees for physical human-robot interaction," *The International Journal of Robotics Research*, vol. 22, no. 7-8, pp. 479–504, jul 2003.

[11] S. Haddadin, A. Albu-Schaffer, M. Frommberger, J. Rossmann, and G. Hirzinger, "The dlr crash report: towards a standard crash-testing protocol for robot safety - part II: Discussions," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 280–287.

[12] A. Cirillo, F. Ficuciello, C. Natale, S. Pirozzi, and L. Villani, "A conformable force/tactile skin for physical human–robot interaction," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 41–48, jan 2016.

[13] F. Flacco, T. Kroger, A. D. Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, may 2012.

[14] P. Rybski, P. Anderson-Sprecher, D. Huber, C. Niessl, and R. Simmons, "Sensor fusion for human safety in industrial workcells," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, oct 2012.

[15] P. Zhang, P. Jin, G. Du, and X. Liu, "Ensuring safety in human-robot coexisting environment based on two-level protection," *Industrial Robot: An International Journal*, vol. 43, no. 3, pp. 264–273, may 2016.

[16] L. Bascetta, G. Ferretti, P. Rocco, H. Ardo, H. Bruyninckx, E. Demeester, and E. D. Lello, "Towards safe human-robot interaction in robotic cells: An approach based on visual tracking and intention estimation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, sep 2011.

[17] M. Lippi and A. Marino, "Safety in human-multi robot collaborative scenarios: a trajectory scaling approach," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 190–196, 2018.

[18] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, apr 2016.

[19] "Safety of machinery - Positioning of safeguards with respect to the approach speeds of parts of the human body." International Organization for Standardization, Technical report, 2010.

[20] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, feb 2012.

[21] Intel, "Intel RealSense D435." [Online]. Available: https://click.intel.com/intelr-realsensetm-depth-camera-d435.html

[22] ROS, "ROS website." [Online]. Available: http://www.ros.org/

[23] ROS, "RViz." [Online]. Available: http://wiki.ros.org/rviz

[24] MathWorks, "MATLAB." [Online]. Available: https://www.mathworks.com/products/matlab.html

[25] MathWorks, "Simulink." [Online]. Available: https://www.mathworks.com/products/simulink.html

[26] ROS, "STOMP." [Online]. Available: http://wiki.ros.org/stomp_motion_planner

[27] R. B. Rusu and S. Cousins, "Point cloud library (pcl)," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.

[28] PCL, "PCL: Tutorials and walkthrough." [Online]. Available: http://www.pointclouds.org/documentation/tutorials/walkthrough.php

[29] ROS, "Camera calibration." [Online]. Available: http://wiki.ros.org/camera_calibration

[30] S. Choi, T. Kim, and W. Yu, "Performance evaluation of ransac family," in *Procedings of the British Machine Vision Conference 2009*. British Machine Vision Association, 2009.

[31] C. Papazov and D. Burschka, "An efficient ransac for 3d object recognition in noisy and occluded scenes," in *Computer Vision – ACCV 2010*. Springer Berlin Heidelberg, 2011, pp. 135–148.

[32] N. Blodow, "Realtime urdf filter," 2012.

[33] S. I. Choi and B. K. Kim, "Obstacle avoidance control for redundant manipulators using collidability measure," in *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*. IEEE, 1999.

[34] P. Bosscher and D. Hedman, "Real-time collision avoidance algorithm for robotic manipulators," in *2009 IEEE International Conference on Technologies for Practical Robot Applications*. IEEE, nov 2009.

[35] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[36] T. J. Ross, *Fuzzy Logic with Engineering Applications*. John Wiley & Sons, Ltd, jan 2010.

[37] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, jun 1965.

[38] MathWorks, "Defuzzification Methods." [Online]. Available: https://it.mathworks.com/help/fuzzy/examples/defuzzification-methods.html

[39] I. Iancu, "A mamdani type fuzzy logic controller," in *Fuzzy logic-controls, concepts, theories and applications*. IntechOpen, 2012.

[40] J. J. Buckley, "Sugeno type controllers are universal controllers," *Fuzzy sets and systems*, vol. 53, no. 3, pp. 299–303, 1993.

[41] MathWorks, "Robotics System Toolbox." [Online]. Available: https://www.mathworks.com/products/robotics.html

[42] MathWorks, "Generate a Standalone ROS Node from Simulink." [Online]. Available: https://www.mathworks.com/help/robotics/examples/generate-a-standalone-ros-node-from-simulink.html