POLITECNICO
MILANO 1863

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
MASTER OF SCIENCE IN MECHANICAL ENGINEERING

From Collaborative Robot to Collaborative Space: Intelligent Human-Robot Collaboration for Smart Factory

Advisor:
Prof. Irene FASSI

Co-advisor:
Dr. Gloria J. WIENS

Authors:
Matteo LAVIT NICORA
900243
Roberto AMBROSETTI
899723

Academic Year 2018-2019

Matteo Lavit Nicora, Roberto Ambrosetti, *From Collaborative Robot to Collaborative Space: Intelligent Human-Robot Collaboration for Smart Factory*

| Master of Science in Mechanical Engineering, Politecnico di Milano.

| Research in collaboration with University of Florida, Gainesville.

© Copyright December 2019.

Politecnico di Milano:

www.polimi.it

School of Industrial and Information Engineering:

www.ingindinf.polimi.it

University of Florida:

www.ufl.edu

To my family

Acknowledgements

We want to thank our advisor, Prof. Irene Fassi, for the precious guidance provided during these months of work. A big thank also to Dr. Gloria J. Wiens, co-advisor of this research, without whom this incredible experience would not have been possible. We also want to thank all the Action Team for welcoming and supporting us during our research at University of Florida. Thanks to our families for bringing us to where we are today and to all our friends that shared this journey with us.

This research was conducted under the University of Florida J-1 program and in collaboration with STIIMA-CNR of Italy. Funding was provided by Politecnico di Milano scholarship "Thesis Abroad - Borsa di Studio Tesi all'Estero" for both Matteo Lavit Nicora and Roberto Ambrosetti and by the NSF/NRI: INT: COLLAB: Manufacturing USA: Intelligent Human-Robot Collaboration for Smart Factory (Award I.D. #:1830383). The authors wish to acknowledge Jack Wittmayer, undergraduate researcher at University of Florida, for his development of the GUI in support of the research. Any opinions, findings and conclusions or recommendations expressed are those of the researchers and do not necessarily reflect the views of the National Science Foundation.

in *Milano, December 2019* Matteo Lavit Nicora, Roberto Ambrosetti

Contents

List of Figures

List of Tables

Sommario

Sistemi in grado di raccogliere dati, comunicare e rispondere in tempo reale sono componenti critiche per la realizzazione di una collaborazione uomo-robot sicura ed efficace per applicazioni manufatturiere in future "smart factories". È quindi di fondamentale importanza una perfetta integrazione di strumenti cognitivi, di rilevamento e previsione all'interno della struttura di controllo del robot. Inoltre, il robot è inserito in un contesto manufatturiero eterogeneo, caratterizzato dalla presenza sia di operatori che di attrezzatura. L'obiettivo di questa tesi è, quindi, lo sviluppo di una cosiddetta Proactive Adaptive Collaboration Intelligence (PACI) e di una logica di switch, componenti centrali dell'architettura di controllo proposta. Grazie a questo approccio, gli autori intendono conferire al robot la capacità di adattare i propri movimenti dinamicamente sulla base delle traiettorie definite offline e del comportamento rilevato dell'operatore. La sfida sta quindi nello sviluppo di capacità decisionali avanzate finalizzate ad ottenere un sistema robotico innovativo, in grado di comprendere la specifica situazione in cui si trova e reagire di conseguenza, rispettando i requisiti di sicurezza e garantendo alti livelli di produttività.

Gli autori hanno scelto di utilizzare ROS Melodic Morenia (sistema operativo: Ubuntu 18.04 Bionic Beaver) in quanto rappresenta uno standard per la ricerca robotica e assicura grande scalabilità e manutenibilità del sistema. La struttura di controllo, sviluppata all'interno di questa piattaforma, presenta le seguenti caratteristiche: *flessibilità* (adatta ad un ampio spettro di applicazioni), *accessibilità* (interfaccia user friendly), *modularità* (tecniche di offline planning, soluzioni di controllo e comportamento selezionabili e facilmente espandibili), *sicurezza* e *produttività*. I comportamenti reattivi del robot sono stati ottenuti tramite una logica di switch che sfrutta funzioni di costo per attivare in tempo reale il controller più adatto alla situazione. Il "costo di attivazione" è valutato sulla base di dati relativi a sicurezza (distanza dall'operatore o altri ostacoli) e produttività (ritardi di produzione registrati). Sfruttando librerie open-source (MoveIt!), i moduli di controllo sviluppati hanno dimostrato alti livelli di modularità e flessibilità. Un banco di prova con hardware-in-the-loop (e.DO robot) e percezione dell'operatore emulata è stato, infine, sfruttato per validare le performance del sistema soggetto a diversi livelli di interazione uomo-robot.

Questa tesi si inserisce in un contesto di collaborazione internazionale ed è stata svolta presso University of Florida (Gainesville, FL, USA). I risultati di questo lavoro rappresentano il punto di partenza per un progetto di ampio respiro chiamato "Intelligent Human-Robot Collaboration for Smart Factory" e finanziato dal programma NSF-NRI.

Abstract

To enable safe and effective human-robot collaboration (HRC) in smart manufacturing, seamless integration of sensing, cognition and prediction into the robot controller is critical for real-time awareness, response and communication. Further complicating matters, the robot is co-operating within a heterogeneous manufacturing environment (robots, humans, equipment). Therefore, the specific research objective of this thesis is to provide the robot Proactive Adaptive Collaboration Intelligence (PACI) and switching logic within its control architecture. That is, give the robot the ability to optimally and dynamically adapt its motions given a priori knowledge and predefined execution plans for its assigned tasks, and detected human actions. The challenge lies in augmenting the robot's decision-making process to have a greater situation awareness and to yield robot behaviors/reactions subject to different human-robot actions while simultaneously maintaining safety and production efficiency.

The work was carried out using ROS Melodic Morenia (running on Ubuntu 18.04 Bionic Beaver) since it is today's standard platform for robotic research and ensures great scalability and maintainability of the system. Inside this framework, a control architecture was developed to have features: *flexibility* (suitable for a large range of applications), *accessibility* (user friendly interface), *modularity* (selective and expandable path planning techniques, high-level controllers, behavior definitions), *safety* and *productivity.* Robot reactive behaviors were achieved via cost function-based switching logic activating the best suited high-level controller. The cost is a function of safety (e.g., obstacle/human proximity) and productivity (e.g., induced time delays). Leveraging the availability of numerous path planning and robot controllers in existing open-source robot libraries (MoveIt!), the PACI's underlying segmentation and switching logic framework was demonstrated to yield a high degree of modularity and flexibility. Using a hardware-in-the-loop testbed setup, the performance of the developed control architecture subjected to different levels of human-robot interactions was validated in the University of Florida e.DO robot testbed, simulating perception of the human operator.

This research has been carried out at University of Florida (Gainesville, FL, USA), member of a multi-university/industry international collaboration. It represents the starting point for a long-term project funded by NSF-NRI and called "Intelligent Human-Robot Collaboration for Smart Factory".

Chapter 1

Introduction

Recent advancements in sensing, computational intelligence, and big data analytics have been rapidly transforming and revolutionizing the manufacturing industry towards robot-rich and digitally connected factories. However, effective, efficient and safe coordination between humans and robots on the factory floor has remained a significant challenge. Further complicating matters, the robot is co-operating within a heterogeneous manufacturing environment (robots, humans, equipment). To enable safe and effective human-robot collaboration (HRC), seamless integration of sensing, cognition and prediction into the robot controller is critical for real-time awareness, response and communication.

This thesis focuses on human-robot collaboration for a smart factory with the research objective to provide the robot Proactive Adaptive Collaboration Intelligence (PACI) and switching logic within its control architecture. That is, give the robot the ability to optimally and dynamically adapt its motions given a priori knowledge and predefined execution plans for its assigned tasks, and detected human actions. The challenge lies in augmenting the robot's decision-making process to have a greater situation awareness and to generate robot behaviors/reactions subject to different human-robot actions while simultaneously maintaining safety and production efficiency.

The authors of this thesis chose to exploit ROS (Melodic Morenia) running on Ubuntu 18.04 Bionic Beaver, since it is today's standard platform for robotic research and allows the development of a scalable system, easily adaptable to several robotic cells, while guaranteeing high levels of maintainability of the system itself. To validate and expand upon the multi-objective, decision-making algorithm methodology established in prior work [6], a control architecture was developed to have features: *flexibility* (applicable to a large range of applications), *accessibility* (user friendly interface), *modularity* (selective and expandable path planning techniques, high-level controllers, behavior definitions), *safety* and *productivity*. Robot reactive behaviors were achieved via cost function-based switching logic activating the best suited high-level controller. The cost is a function of safety (e.g., obstacle/human proximity) and productivity (e.g., induced time delays). Leveraging the availability of numerous path planning and robot controllers in existing open-source robot libraries (MoveIt!), the PACI's underlying segmentation and switching logic framework was demonstrated to yield a high degree of modularity and flexibility. The performance of the developed controller was validated by subjecting it to different levels of human-robot interaction. Tests were performed using the University of Florida e.Do testbed for hardware in the loop setup and including a simulated human operator.

This research has been carried out at University of Florida, member of a multi-university/industry international collaboration. It establishes and evaluates an underlining framework of the robot control architecture for the “Intelligent Human-Robot Collaboration for intelligent HRC in smart manufacturing.

1.1 Intelligent HRC for Smart Factory

Three U.S. universities (Missouri University of Science and Technology, University of Florida, Case Western Reserve University), the National Research Council of Italy (Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing) and industry Comau LLC (COnsorzio MAcchine Utensili) have recently started an international partnership, in order to push the boundaries of safe and effective human-robot collaboration in manufacturing by launching a project called “Intelligent Human-Robot Collaboration for Smart Factory”, funded by the National Science Foundation/National Robotics Initiative (NSF/NRI) program. The aim is to develop an integrated set of algorithms and robotic testbeds to sense, understand, predict and control the cooperation of human workers and robots in collaborative manufacturing cells, for significantly improved productivity of hybrid human-robot production systems towards deployment in future “smart factories”. This project addresses several fundamental challenges in human-robot collaboration in the manufacturing environment, such as the limitation of one-to-one sensing between humans and robots, the lack of adaptive and stochastic modeling methods for reliable recognition and prediction of human actions and motions in different manufacturing scenarios, and multi-scale human-robot coordination.

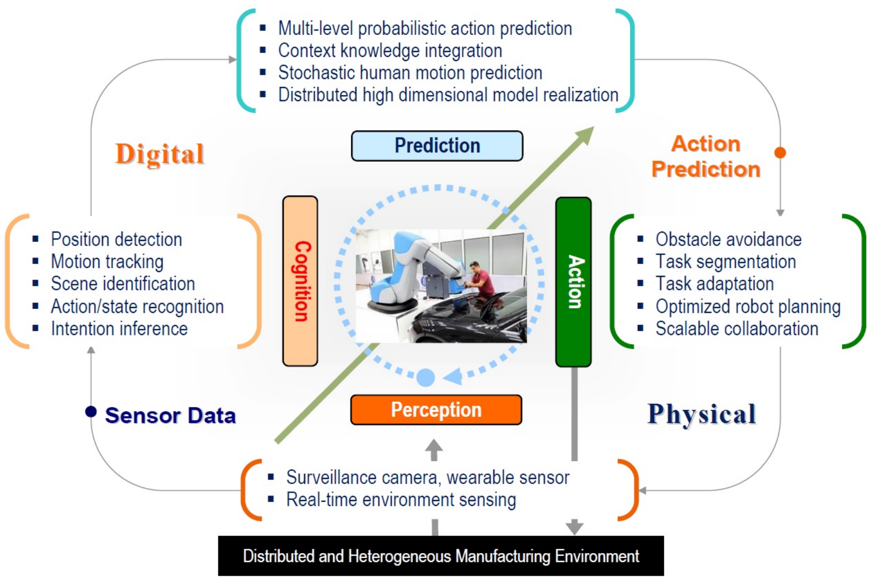

Multi-level probabilistic action prediction
Context knowledge integration
Stochastic human motion prediction
Distributed high dimensional model realization
Digital
Prediction
Action Prediction
Position detection
Motion tracking
Scene identification
Action/state recognition
Intention inference
Cognition
Action
Obstacle avoidance
Task segmentation
Task adaptation
Optimized robot planning
Scalable collaboration
Sensor Data
Perception
Physical
Surveillance camera, wearable sensor
Real-time environment sensing
Distributed and Heterogeneous Manufacturing Environment

Figure 1.1: Research tasks and subdivision scheme

To address these challenges, multi-disciplinary research involving sensing, machine learning, stochastic modeling, robot path planning, and advanced manufacturing is being conducted collaboratively under this partnership. In particular, four research tasks aimed at the realization of the envisioned human-robot collaboration have been identified and are presented in Figure 1.1. The tasks represeted in the scheme have been assigned to different teams with the following roles:

- **Sense** where objects (e.g., robots, humans, parts or tools) are located;
- **Cognize** what each worker is doing;
- **Predict** what the next human action will be;
- **Plan and control** safe and optimal robot trajectories for individualized job-specific-collaboration between human and robot, avoiding worker injury proactively.

The authors of this thesis are part of the Action Team, which deals with the 'plan and control' research task conducted at the University of Florida. Over the course of four years, the team will investigate multilayer and modular control structures that allow stable mode switching for flexibility in defining the 'optimized' real-time robot response, to safely adapt to human worker planned and unplanned interactions, and to maintain production efficiency. The research will also focus on the development of optimal safe real-time obstacle avoidance and robot motion control amongst a shared space with humans and other moving equipment.

A multi-layer control structure, shown in Figure 1.2, is selected for interoperability between algorithms and devices and builds upon the backbone of the industrial robotic cell. This layer of control retains the functionality of the original industrial robot and its safety features, robot sensor feedback (encoders, vision, proximity), cell management of robot and other moving equipment.

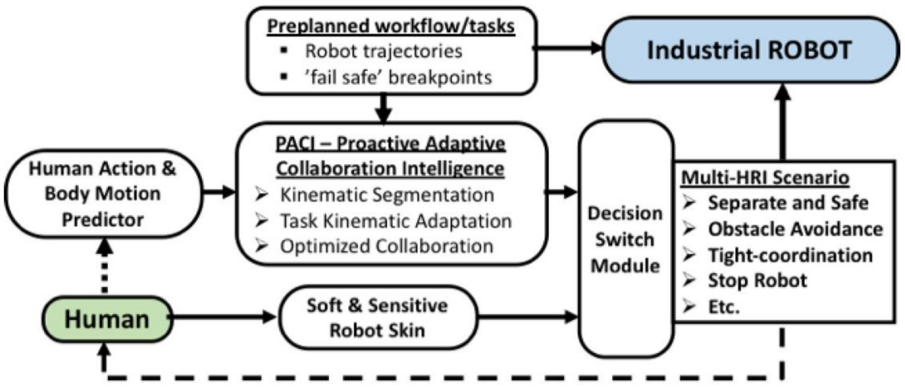


Figure 1.2: Proposed multi-layered control structure

The core of the proposed control structure in Figure 1.2 is represented by the Proactive Adaptive Collaboration Intelligence (PACI) layer, in charge of modifying the robot motion based on the input from the Human Action and Body Motion Predictor and the Preplanned workflow/tasks module. The former provides as input the predicted sequence of human actions and body motion trajectories accompanied by the confidence intervals of the trajectories. The latter provides a preplanned workflow/task schedule including a library of robot trajectories and ideal task breakpoints for 'fail safe' priority. The PACI algorithm performs kinematic segmentation for effectively parsing the data received from the prediction algorithm and for identifying impact on current/preplanned robot trajectories and tasks. Then PACI kinematically adapts tasks in real-time for safe controlled robot motions, optimizing the collaboration and mitigating production disruptions. Another feature of the proposed control structure is a soft and sensitive skin mounted on the robot, used in order to provide the operator with feedback intervention capability to stop the robot motion by simply touching it. The output of the PACI algorithm combined with direct human intervention (e.g., interrupts via Soft and Sensitive Robot Skin interface), represent the input for the Decision Switch Module. This module toggles the control action to match with the optimized Human-Robot Interaction (HRI) scenario and sends modified trajectory/controller updates to the Industrial Robot. As a further layer of safety, the controller will have the capability to signal the humans when necessary to alert them of a need to change their intended behavior. A robot task in progress may need to avoid an unsafe situation by communicating its state to the human, for example by means of visual tools or haptic arm bands.

1.2 Thesis objective

This thesis focuses on the modules of the proposed control structure that have been highlighted in Figure 1.3, designing and implementing them as a series of multi-layered algorithms achieving efficient and safe coordination between humans and robots for future deployment in "smart factories".

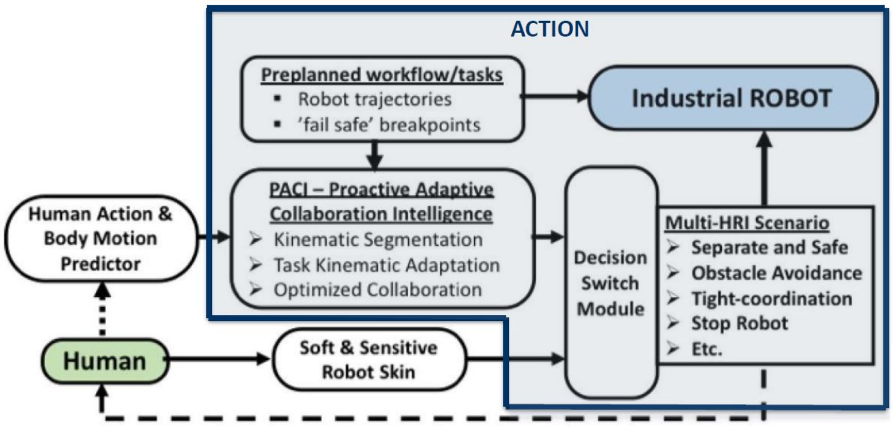


Figure 1.3: Proposed control structure with highlight on the addressed modules

To meet the current challenges of state-of-the-art, “fenceless” and smart HRC solutions, the authors envision a robotic system with the following features:

- **Flexibility**: seamless adaptability of the system to a wide variety of applications and advanced customizability of the product, main needs of today’s industry 4.0;
- **Accessibility**: intuitive, fast and easy programmability of the robotic system, accessible to the operator without the need of any advanced knowledge;
- **Modularity**: easily manageable code structure that enhances the reusability of the code by allowing developers to add new features or discard obsolete ones without rearranging the whole system;
- **Safety**: real-time awareness and response capabilities ensuring safety of humans, robots and equipment co-operating within the manufacturing cell;
- **Productivity**: smart human-robot collaboration that guarantees high levels of productivity.

In order to develop a robotic system coherent with the framework of the proposed control structure and that guarantees the desired features listed above, the authors propose a solution composed of two multi-layered modules, resulting in a slightly reorganized control structure, illustrated in Figure 1.4.

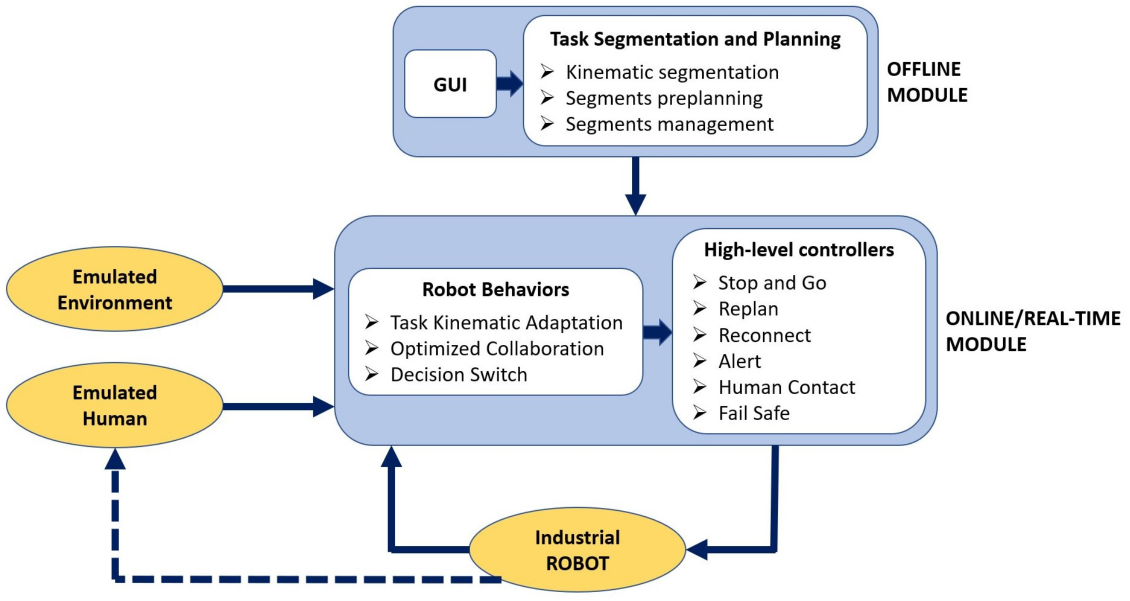


Figure 1.4: Reorganized control structure

As depicted, the Offline Module is equipped with a Graphical User Interface (GUI) that takes as input the requests of the user and feeds the processed information to a second multi-layered module that performs the offline kinematic segmentation of the task, preplans all the segments and manages their distribution. The second main component of the control structure is an Online/Real-time Module that takes as input the information provided by the first module and collects data about the human presence and the external environment. This data is exploited for the task kinematic adaptation of the robot motion to achieve optimized collaboration with the operator, in accordance with the envisioned PACI algorithm. The outcome triggers a Decision Switch layer that real-time activates the high-level controller that best suits the human-robot interaction scenario.

The introduction of a GUI in the robotic system sensibly increases accessibility by lowering the skill threshold required to program the robot coherently with the Task Segmentation and Planning approach. The segmentation process itself provides high levels of flexibility since each segment is managed independently in terms of offline trajectory planning and real-time task adaption. The combination of the two modules determines the motion of the robot that accomplishes high productivity levels while ensuring the safety of the HRC, also exploiting a feedback signal that the robot can use to communicate with the human. Finally, the algorithms that compose the control structure are written in a modular fashion providing a list of possible choices regarding offline planning techniques, decision switch logics and human-robot collaboration scenarios. It should be noted that the inputs of the Online/Real-time Module in Figure 1.4 are labeled as “emulated”. Due to the early stages of the project at the time of the authors’ research, the sensor fusion input envisioned for the complete robotic system was not yet available. For this reason, they have been emulated in order to allow system testing, overcoming the absence of real sensors. In order to demonstrate that the characteristics of the developed control structure match the initial goals of the authors, achieving efficient and safe HRC, a hardware-in-the-loop validation on lab-scale testbeds has been performed.

1.3 Thesis structure

The thesis starts from a literary review (Chapter 2) aimed to provide the reader with a clear overview of the state of the art of collaborative robotics for industrial manufacturing and the current challenges in this field. Then, after a presentation of the main software and hardware tools exploited in this research (Chapter 3), the thesis follows the structure of the control scheme. First, the Task Segmentation and Planning approach is introduced with a detailed explanation of the developed GUI and of the logic used to create, plan and manage the segments that compose a task (Chapter 4). The following chapter deals with the online capabilities of the robotic system addressing the content of the Online Module with an in-depth description of the so-called Robot Behaviors and High-level Controllers (Chapter 5). A description of the experimental scenario used to test the control system and a presentation of the obtained results follow (Chapter 6), in order to validate the outcome of the research. Finally, the thesis ends with a conclusive discussion on the research and with a proposal for future developments (Chapter 7).

Chapter 2

State of the art

2.1 Introduction to industrial robotics

Many definitions of “industrial robot” are available, but the commonly accepted one is reported in the international standard ISO 8373:2012 [7] in which the following can be found:

"An industrial robot is an automatically controlled, reprogrammable, multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications.”

Industrial robots are a crucial part of the progress of manufacturing industry as can be seen from the last estimates by the International Federation of Robotics. With reference to Figure 2.1, the new World Robotics report [1] shows record breaking annual global robot sales of 16.5 billion USD. Additionally, it reported a 6% percent shipment increase in 2018 compared to the previous year, with a total of 422,000 units shipped globally. The IFR’s long term outlook shows that the ongoing automation trend and continued technical improvements will keep growing, with an estimate of about 584,000 units in 2022.

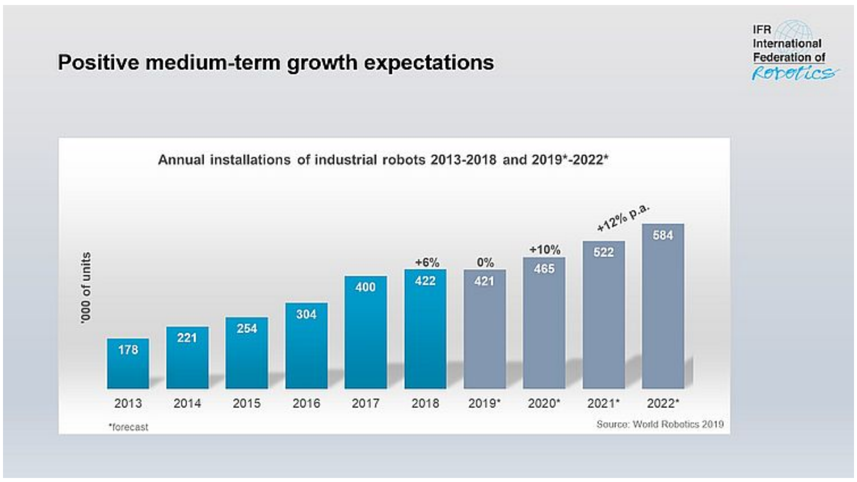


Figure 2.1: Data on the annual installations of industrial robots, [1]

Looking at the plot in Figure 2.2, Asia is still the world's largest industrial robot market. However, the combination of the decline in installation experienced by China and Republic of Korea together with the considerable growth of Japan result in an overall 1% growth of the region's market for this past year. In Europe, on the other hand, the market has increased by 14% and reached a new peak for the sixth year in a row, similarly to the Americas that registered a growth rate of 20%, again marking a new record.

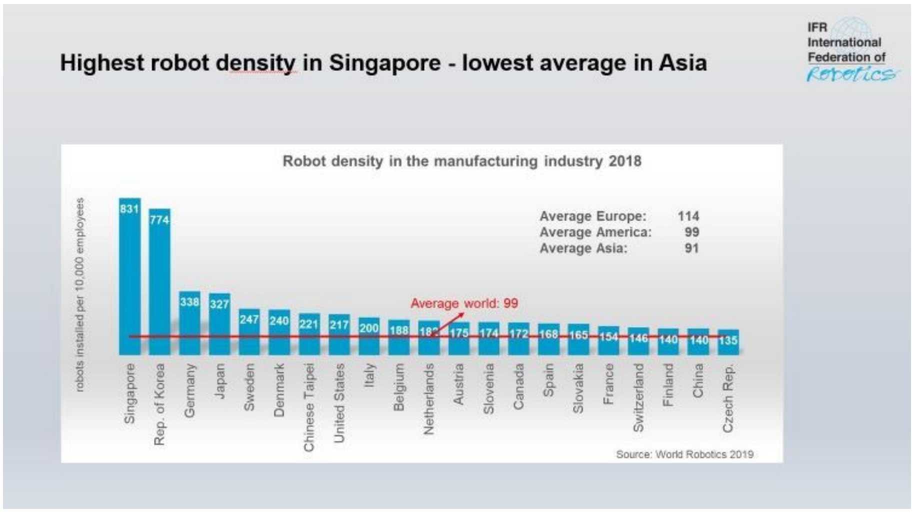


Figure 2.2: Robot density by country, manufacturing industry, 2018, [1]

As reported by IFR, an increasing need of robots has been found in all industrial sectors, but the strongest demand still pertains to the automotive industry which accounts for 30% of the total supply. For the first time this year, World Robotics has analyzed the market for collaborative robots (cobots) with interesting results, presented in Figure 2.3. The number of units installed is still very low with a share of 3.24% of the total market but the number of annual installations of cobots shows a promising 23% increase from 2017 to 2018.

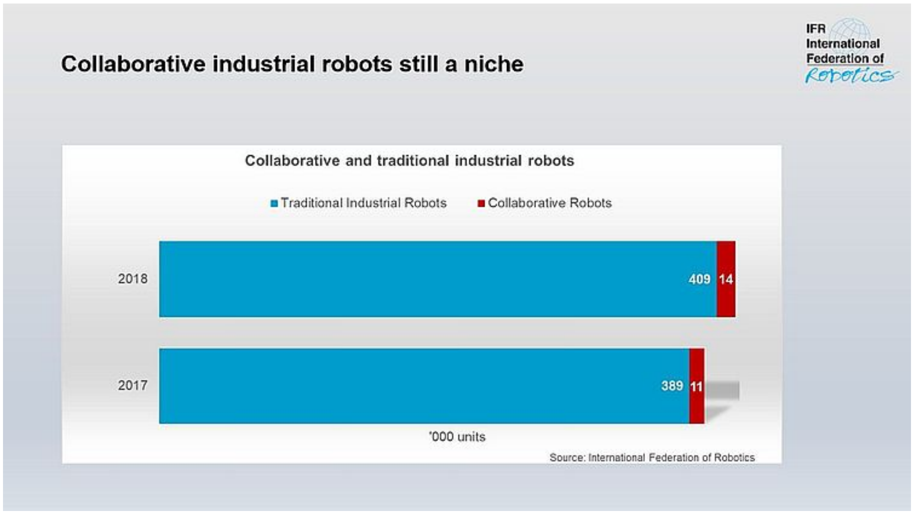


Figure 2.3: Collaborative and traditional industrial robots in 2017 and 2018, [1]

The reason for the rapid growth of industrial robots registered in the past few years is easily explained. Robots can be programmed to perform dangerous and repetitive tasks with consistent precision and accuracy and their automated functionality allows them to operate no-stop, in the presence of hazardous materials and environments, therefore leaving personnel free to perform other tasks. Robotic technology also leads to sensible increase in productivity and profitability and, at the same time, eliminates labor-intensive activities that might cause physical strain or potential injury to workers [8]. Robotic systems therefore seem to be perfect machines for the fully automated processes, but many limitations start to rise when trying to adapt this technology to a world in which the demand of customized products is arising exponentially. The internet connectivity at the core of Industry 4.0 has enabled "mass-customization" of products and market research shows that the trend is already evolving towards a so called "mass-personalization" leading to the need of an extremely high level of flexibility of the production system [9]. All these challenges could find a promising solution in the fact that where robots begin to struggle, humans excel and vice versa. Robots are perfectly suited to execute repetitive tasks with force and precision, humans are naturally able to perform complex handling tasks while quickly adapting to changes in the environment and to new process sequences. In particular, a series of aspects in which humans have an advantage over robots can be identified:

- They are naturally equipped with a wide array of sensors (touch, vision, hearing, pattern recognition. . .);
- They have the ability to learn and make decisions even when the required data is not available;
- They are flexible and easily trainable;
- They are characterized by a high degree of mobility;
- They represent a low capital investment.

All the jobs requiring one or more of these skills therefore find a more immediate and straight-forward implementation in the work of human operators instead of robots. A large number of cases can be found in which, due to the current limitations of this technology, the use of robots is either impossible or impractical. For the sake of brevity, here only one significant example is reported. Final assembly of cars represent a great challenge for automated robotic systems: the flexibility of carpets and wire harnesses makes them unpredictable components that would require extremely advanced sensory capabilities not yet possessed by robots. Another problem arises in this field when dealing with the method used to supply the components: a robot either requires a precise orientation and positioning of each part or a complex vision system in order to be able to handle them while, on the other hand, the same parts could easily be supplied to an operator in bins. Furthermore, the mobility of robotics systems represents another limitation. In general, robotic cells are designed to have a certain degree of redundancy so that, in case of a fault, the workload of a manipulator can be covered by a different one. When this happens, though, a lot of time is wasted in order to adapt the cell to the new configuration while an operator naturally ensures this degree of mobility [10]. It is clear that both robots and human operators have unique advantages and disadvantages over each other. Therefore, the development of robot systems in which a human can take over portions of the task that are too hard for a robot is of great interest. Those systems would enable high-performances and precision while still maintaining a high-degree of flexibility and adaptability. Figure 2.4 shows the trend of the number of product variants as a function of their production volume. The plot also identifies the manufacturing solution best suited for each situation and, as represented, collaborative applications are directly related to mass customization. "Collaborative robotics" has been one of the central topics of research in the robotic world for the past few years and continues to be very important as several challenges still have to be addressed.

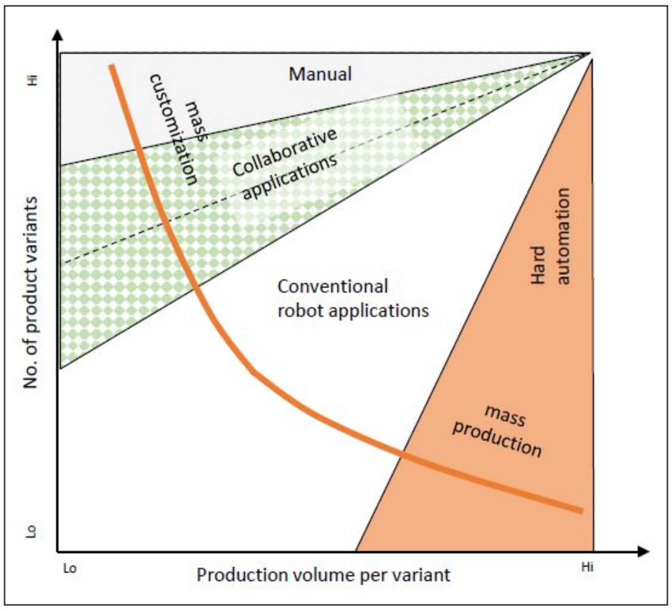


Figure 2.4: Product variants and volumes for the main production solutions

In order to provide the reader with formality around the concept of collaborative robotics, the main definitions reported in ISO/TS 15066:2016 [11] can be found below:

- **Collaborative robot**: A robot capable of being applied for use in a collaborative operation;
- **Collaborative operation**: State in which a purposely designed robot system and an operator work within a collaborative workspace;
- **Collaborative workspace**: Space within the operating space where the robot system (including the workpiece) and a human can perform tasks concurrently during production operation.

This Technical Specification [11] provides general guidance for collaborative robot operations, where a robot system and human operators share the same workspace. In such operations, the integrity of the safety-related control system is of major importance and a comprehensive risk assessment is required to assess both the robot system and the environment around it. This international standard cannot be referenced alone since it basically supplements the requirements and guidance given in a series of additional documents, listed below:

- *ISO 10218-1:2011, Robots and robotic devices – Safety requirements for industrial robots – Part1: Robots*, [12];
- *ISO 10218-2:2011, Robots and robotic devices – Safety requirements for industrial robots – Part2: Robot systems and integration*, [13];
- *ISO 12100:2010, Safety of machinery – General principles for design – Risk assessment and risk reduction*, [14];
- *ISO 13850:2015, Safety of machinery – Emergency stop function – Principles for design*, [15];
- *ISO 13855:2010, Safety of machinery – Positioning of safeguards with respect to the approach speeds of parts of the human body*, [16];

In the following sections, the authors present a brief description of the role played by collaborative robots in today's industry together with a complete state of the art of this technology, in order to provide a wide understanding of the main goals and challenges that have characterized this field of research up to now.

2.2 The role of collaborative robotics

To achieve safety, robotic applications traditionally exclude human access to the operations area while the robot is active. Therefore, a variety of operations requiring human intervention often cannot be automated using robot systems. The objective of cobots is to combine the repetitive performance of robots with the individual skills and abilities of human operators: while the former one exhibits precision, power and endurance, the latter one has an excellent capability of solving imprecise exercises and adapting to the specific situation. It is clear that a number of promising benefits could be the direct consequence of such an implementation, as shown below in Figure 2.5.

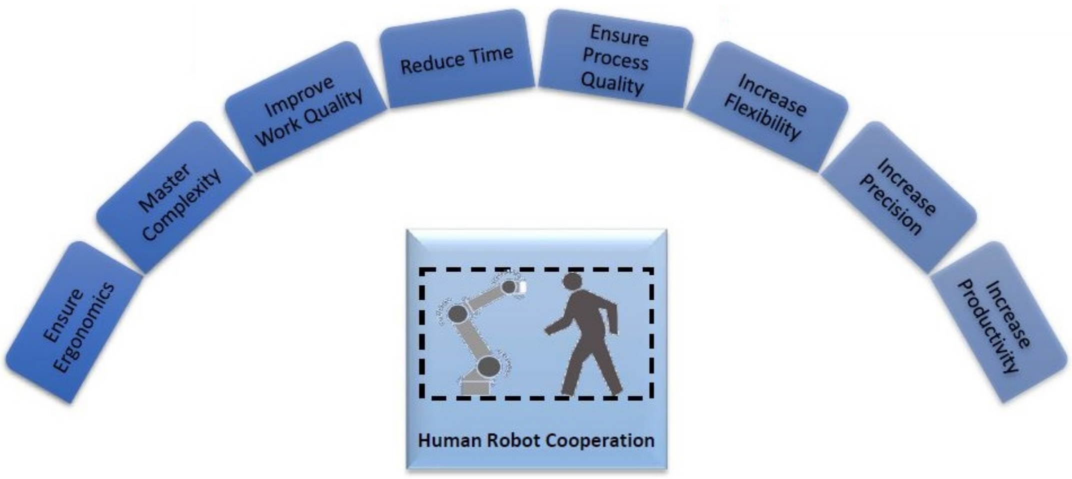


Figure 2.5: Benefits of human-robot collaboration

Moreover, most cobots are fully compatible with Industry 4.0 design principles. In 2015, McKinsey defined Industry 4.0 as “the next phase in the digitization of the manufacturing sector, driven by four disruptions:

- The astonishing rise in data volumes, computational power, and connectivity, especially new low-power wide-area networks;
- The emergence of analytics and business-intelligence capabilities;
- New forms of human-machine interaction such as touch interfaces and augmented-reality systems;
- Improvements in transferring digital instructions to the physical world, such as advanced robotics and 3D printing."

As stated by McKinsey, the role of “human-machine interaction” is of utmost importance for the current developments of automation. Cobots could represent a first important step towards a future which, with reference to Figure 2.6, can be referred to as “Fourth Robotic Revolution”.

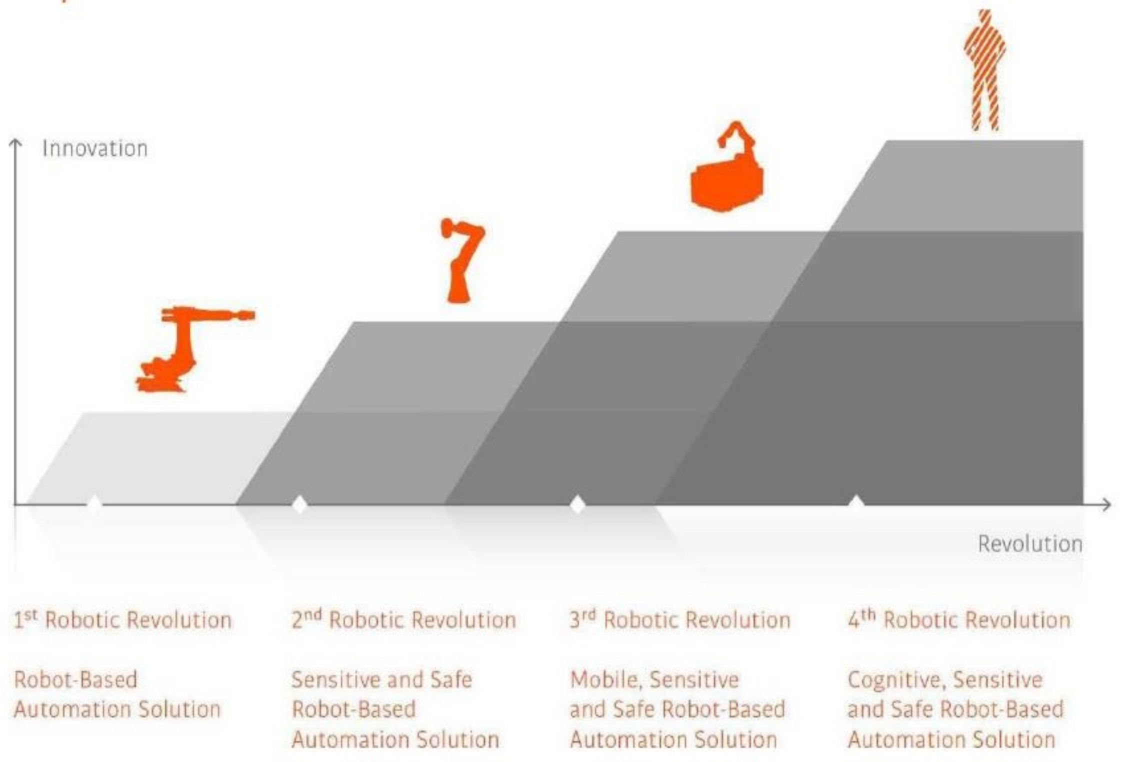

Innovation
Revolution
1st Robotic Revolution
Robot-Based Automation Solution
2nd Robotic Revolution
Sensitive and Safe Robot-Based Automation Solution
3rd Robotic Revolution
Mobile, Sensitive and Safe Robot-Based Automation Solution
4th Robotic Revolution
Cognitive, Sensitive and Safe Robot-Based Automation Solution

Figure 2.6: Features of the four Robot Revolutions

Beyond their innate Industry 4.0 compatibility, cobots have been the main actors in enabling small and medium-sized enterprises (SMEs) to start automating their processes. Because they are versatile, easy to program, small, lightweight and affordable, cobots are being deployed at companies not yet ready to build a state-of-the-art Industry 4.0 facility, and at companies of all sizes in developing countries. These characteristics also make cobots perfectly suited for deployment in processes that were not previously automated. It can be therefore said that cobots are helping companies everywhere to join the latest wave of automation, even if they are not ready to go all the way to Industry 4.0. Furthermore, because cobots are programmed, configured and controlled locally in the factory, companies deploying this solution on their factory floors are able to retain the expertise over their automated processes. This results in greater operational agility and flexibility, and greater competitive power in world markets [17].

2.3 State of the art of collaborative robotics

Collaborative operation is ripe for exploration and innovation, opening the possibility for widespread adoption of robots into new fields for which they may currently be perceived as unsuitable. Even with the current state-of-the-art in skill acquisition and execution, robots have difficulty performing at their full potential when removed from the typically well-controlled environment of the lab. As explained so far, collaborative robot systems could represent a solution to this problem, but many challenges still have to be addressed. The first aspect to be taken into consideration is that robots are already working side by side with human operators inside many factory floors, but different levels of interaction can be identified, as suggested in the paper [18].

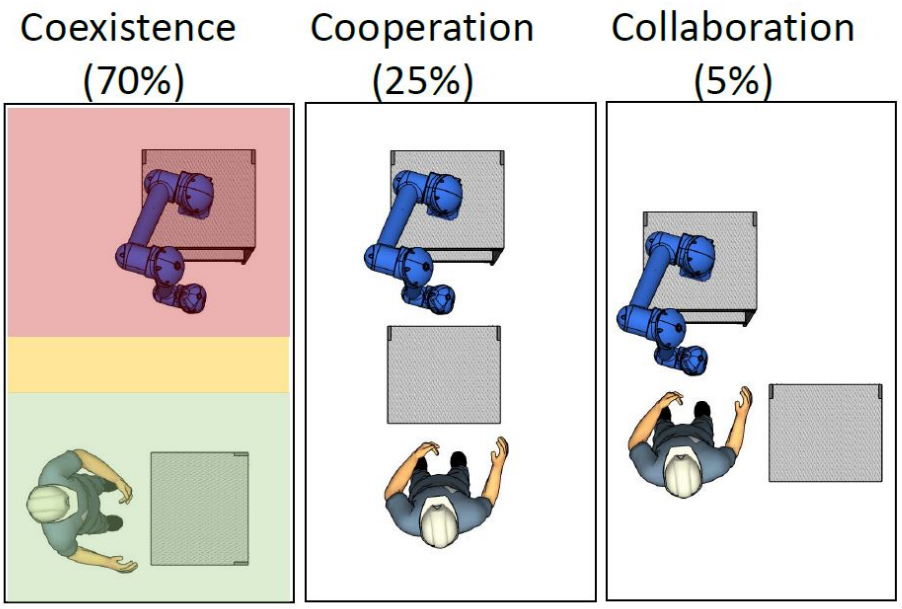


Figure 2.7: Levels of human-robot interaction

With reference to Figure 2.7, most industrial robots are simply “coexisting” with the human operator since their workspaces are completely separated and no direct interaction is ever allowed to happen. In many cases this separation is even enforced by placing the robots inside cages or creating physical/virtual fences around their workspace in order to stop any motion if an unexpected access is detected. A lower percentage of the industrial robots are already entering the world of the so called “fenceless robots”, capable of “cooperating” with the operator in a tighter space and sharing the same workspace, but either on a sequential basis or in parallel. This second level of interaction represents a big step towards the concept of cobots, but still retains a series of limitations. For instance, the workspace of the two entities is the same, but is either accessed by only one of the two at a time (sequential) or at the same time but with no contact intended (parallel). Finally, representing a small niche of today's deployed industrial automation, a small percentage of robots is actually capable of "collaborating" with the human as a second teammate would. In this case both the human and the manipulator are allowed to work in the same workspace simultaneously with a series of situations in which direct contact is intended. Achieving this final level of interaction seamlessly and safely while still maintaining high levels of productivity, represents the current challenge of collaborative robotics as well as the goal of this thesis and of the long-term project to which it belongs.

As just stated, the first issue to be tackled when trying to develop this kind of collaborative system is the operational safety of human operator, robot and equipment. In [19], a categorization of the different levels of human-robot interaction similar to the one just presented is proposed. Additionally, a nested framework is used to clarify how a higher level of human-robot interaction inherits the requirements of the lower levels and introduces new necessary features that make the system significantly more complex, as reported in Figure 2.8 below.

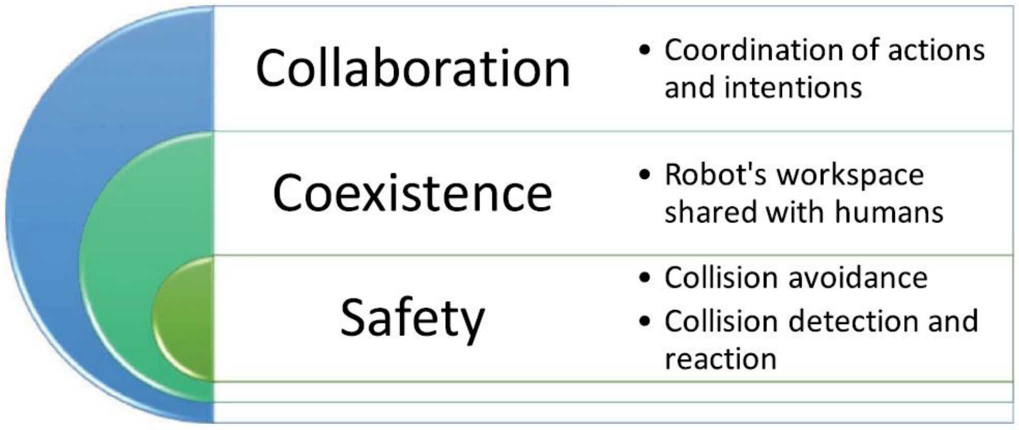

Collaboration
• Coordination of actions and intentions
Coexistence
• Robot's workspace shared with humans
Safety
• Collision avoidance
• Collision detection and reaction

Figure 2.8: Nested levels of safety for human-robot collaboration, [2]

In order to guarantee all these features, a series of solutions have been proposed in the past, but success is limited due to the disruption of system production, generated while ensuring this level of safety. Safety standards provide unified requirements and design guidelines which help and simplify the development of new systems. From a formal point of view, compliance to standards is not mandatory to demonstrate the safety of a system but it conveniently reduces the effort in safety compliance and certification with respect to Machinery Directive, which is the main European legislation for health and safety requirements for machinery, and at the same time it speeds up the commissioning of new systems. According to these standards, four main collaboration modes can be distinguished [2]:

- **Safety-rated Monitored Stop (SMS)**: The operator performs manual tasks inside a collaborative area in which both the human and the robot can work. Safety is guaranteed by ensuring that, if the operator is occupying the shared space, the robot is not allowed to move. When the human enters the collaborative area, the robot switches to a "safe standstill" mode, as indicated in [20]: its movement is paused maintaining the automatic cycle active so that the program can restart from where it was interrupted as soon as the worker leaves the collaborative area. Applications for this type of cooperation can be found in literature [21], for instance manual placement of objects to the robot's end-effector, finishing operations or other complex operations that require the presence of a human, or situations in which the robot can help the operator to position heavy components.
- **Hand Guiding (HG)**: In this collaborative mode the operator teaches the robot positions directly moving it for example by means of a teach pendant, while the weight of the robot arm is compensated in order to hold its position. Examples of this collaborative approach in an industrial assembly scenario can be found in [22] and the more recent [23]. This collaborative solution is much more complicated to implement since the robot must be equipped with both safety-rated monitored stop and safety-rated monitored speed capabilities. The robot executes its program inside the collaborative area automatically until the operator approaches it. This situation triggers the robot's program interruption and, as the operator activates the hand guiding device, the robot state switches to safety-rated monitored speed functionality. This allows direct contact with the manipulator that follows the operator's guidance. When the hand guiding device is released, the robot returns in safety-rated monitored stop and any previously interrupted program can restart as soon as the operator leaves the collaborative area.
- **Speed and Separation Monitoring (SSM)**: In this collaborative mode, the human presence inside the same workspace of the robot is allowed through safety-rated monitoring sensors. With reference to Figure 2.9, the robot operates at different speeds depending on the zone occupied by the operator: full speed for the green zone and reduced speed for the yellow zone, while the manipulator's motion is completely stopped as the human moves into the red zone. Metrics that take into account safety and productivity of this approach are provided in [24]. These areas are constantly inspected by a vision system that should also monitor areas out of the reach of the manipulator, where the operator does not get in contact with the robot, but that can represent a potential danger due to dropped manipulated objects. As in SMS, when the operator moves away from the red zone the manipulator's motion can be restarted either at reduced or maximum speed depending on the position of the operator. An analysis of the current equation for SSM implementation is discussed in [25], while a practical application of this approach can be found in [26].
- **Power and Force Limiting (PFL)**: This collaborative approach allows the worker to operate side-by-side with the robot by limiting the motor power and force. Dedi-

cated equipment and control models are required in order to ensure that collisions between the robot and the human do not have any harmful consequence for the latter. An overview of human–robot physical interaction control has been proposed in [27], where a classification of the different types of contact, their related injuries and a series of collision handling methods are presented. In the paper, four robot reactions in response to the contact are also proposed. A straight-forward solution can be achieved by activating the robot's brakes after collision with immediate stop while torque control with gravity compensation, torque and admittance reflex are more advanced strategies leading to a safer behavior (e.g. lower impact energy thanks to a countermotion in the opposite direction).

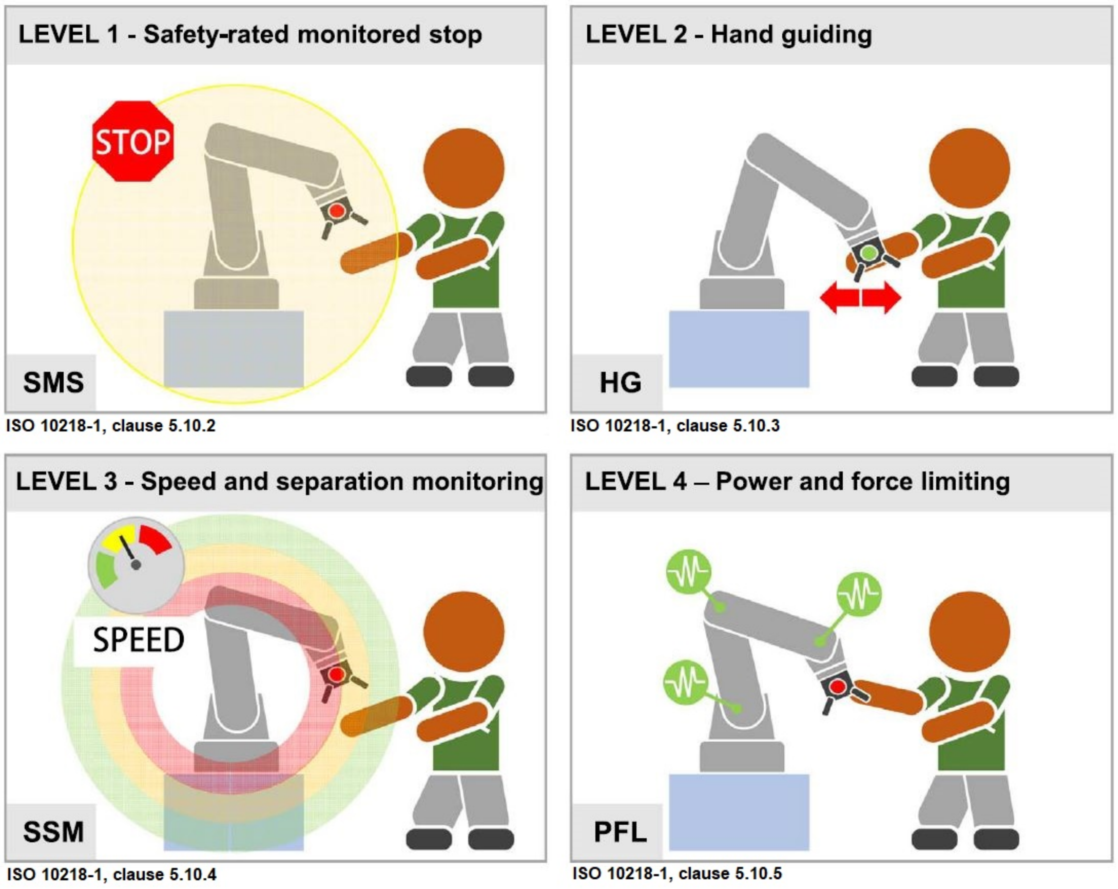

LEVEL 1 - Safety-rated monitored stop
STOP
SMS
ISO 10218-1, clause 5.10.2
LEVEL 2 - Hand guiding
HG
ISO 10218-1, clause 5.10.3
LEVEL 3 - Speed and separation monitoring
SPEED
SSM
ISO 10218-1, clause 5.10.4
LEVEL 4 – Power and force limiting
PFL
ISO 10218-1, clause 5.10.5

Figure 2.9: Representation of the four human-robot collaboration modes

A very interesting approach to collaborative robotic systems has been proposed in [3]. The authors of the paper envision "complex robot behaviors to emerge in real time from the interplay of several concurrently running elemental controllers". In order to accomplish complex tasks, a solution to provide proper sequencing of skills in a reactive manner is found in Behavior Trees [28], used to assign appropriate success/failure conditions to the various elemental controllers. A pre-defined library of skills optimized beforehand is made available by the authors of the paper so that a user has free choice to achieve completion of the task at hand. A schematic representation of the concept proposed in the paper is presented in Figure 2.10. One benefit of this approach highlighted by the authors of the paper is that “existing controllers can be leveraged also for learning new ones and that controllers can be implemented in arbitrary operational spaces”.

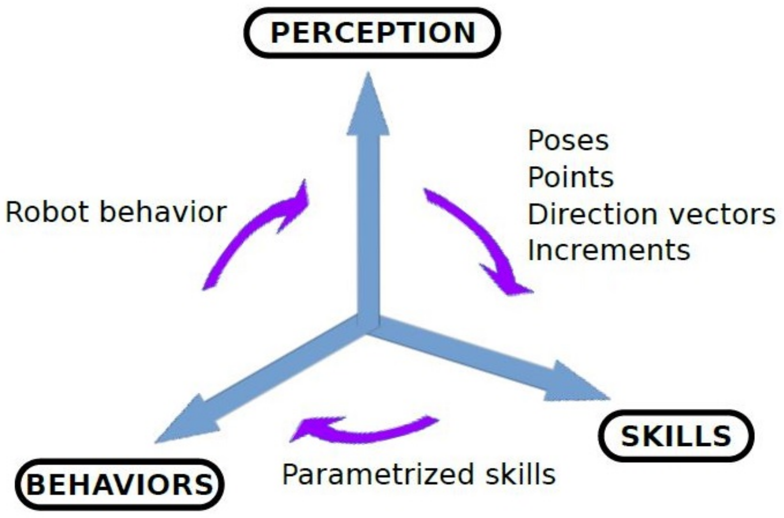


Figure 2.10: Schematic representation of the concept of behavior in [3]

Additional robot controller approaches are available in literature. For example, a deformation-tracking impedance control method was validated with a robot performing a cooperative assembly tasks with the human worker acting as the time-varying environment [29]. Using a parallel combination of a baseline time-invariant controller and a safety controller enforcing a time varying safety constraint, U.C. Berkeley researchers are working to establish a set of design principles for a safe and efficient robot collaboration system (SERoCS). Their approach consists of: efficiency and safety goals treated separately, modularized structure, compatible with existing robot motion control algorithms, online safety controller, robot motion confined to safe regions according to predicted human motion, reduced computations by modeling humans as single or multiple sphero-cylinders with portions of the control policy solved offline [30]. A collision avoidance strategy is presented in [31] for on-line re-planning of robot motion and creates a safe network for unsafe devices (distributed layers of data cross-checking and validation of sensors, PLCs, PCs) as an infrastructure for achieving functional safety. An optimal control problem formulated for physical-HRC based robot motion control is augmented with a social-HRC in [32], improving interactions in assembly tasks by increasing the human worker’s trust in his/her robot partner. It was also shown that an optimal velocity could be determined which reduced human workload while maintaining the overall performance of the human–robot team. However, because the generation of safe robot trajectories is limited due to the inherent uncertainty of robot trajectory execution time (i.e., a need to modulate robot speed according to the distance between the robot and the worker), CNR-STIIMA estimated a confidence interval on robot trajectory execution time for scenarios in which human–robot space sharing is required [5].

Today's research on cobots is not solely focused on the development of new methodologies for real-time motion planning and control that ensures safety and productivity of the system. Another relevant research topic for the field of collaborative robotics is the way cobots are managed and taught. It is of fundamental importance to provide human operators with intuitive interfaces that ease the process of communication and therefore leave the operator free to concentrate on the task and goals at hand. Traditional non collaborative robots often require expert engineers for the programming operations on the robot. This obviously means higher cost of personnel and possible disruption of the company's expertise, since in many cases external consultants are needed for the management of the automated system. In the case of cobots, this is not a suitable scenario since the operator works side-by-side with the manipulator and must be able to easily communicate with the it and receive feedback from it, as would happen with a human collaborator, in order to achieve safe and productive levels of collaboration. In general, a distinction is made between online and offline programming. Online programming approaches require the interruption of the robot's production cycle since the user will "teach" the new task directly to the robot itself. Many methodologies are available in literature and are currently being investigated: *lead-through programming*, in which the robot learns the new trajectory by storing a series of positions defined by jogging through the use of a teach pendant [33] [34], *walk-through programming*, in which the operator is allowed to physically move the end-effector of the robot through the desired positions freely therefore creating a direct intuitive language between the robot and the human operator [35], *programming by demonstration*, in which the robot is not purely reproducing the motion of the operator as in walk-through programming but is also able to generalize them in new scenarios [36] and other new interaction modes that exploit sensors in order to alleviate the burden of communication with the robot (speech, gestures, eye tracking, facial expression, haptics). *Offline programming*, instead, uses software tools to virtually replicate the shop floor on a computer in order to minimize the downtimes of the robot. In this case many intuitive interfaces have been developed [37], but typically each manufacturer has its own specific software protected by very expensive licenses. Another aspect to take into consideration is the mental strain of the operator during the interaction with the robot, currently under research in a framework called "affective robotics" [38]. The goal is once again to relieve the cognitive burden of the user, when the task to be accomplished overloads his/her mental capabilities, by adapting the behavior of the robot and implementing a sufficient level of autonomy [39]. Finally, only recently much interest in robot interfaces has been devoted to augmented and virtual reality for manufacturing applications. The first examples of application have shown that these new approaches can lead to improved productivity of the system and enhanced human safety [40]. Figure 2.11 provides a quick summary of all the aspects just presented.

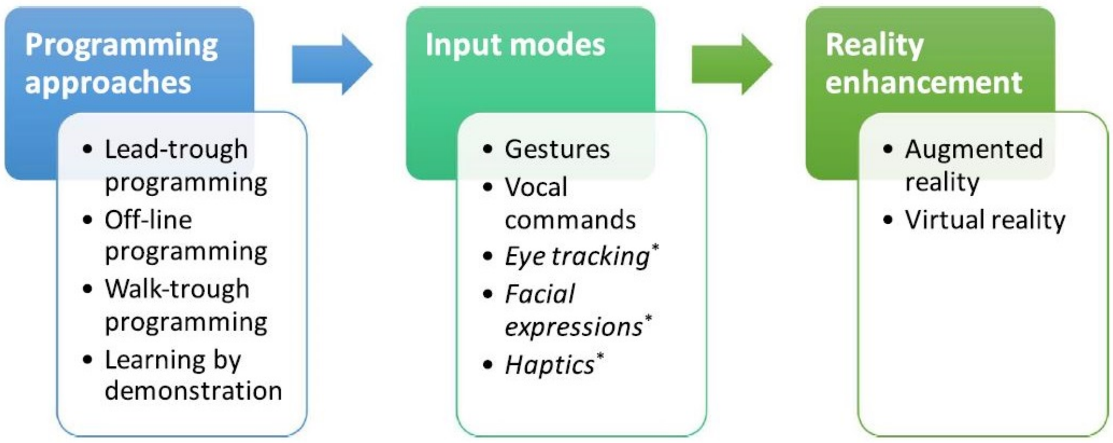


Figure 2.11: Programming approaches, input modes and reality enhancement listed in [2]

The main research topics for collaborative robotics, that have been presented up to this point, represent the starting point for the development of the system envisioned in the long-term project of which this thesis is a part of. Therefore, starting from the state of the art defined in this section, the authors' research attempts to combine all the benefits of the solutions already published in literature while at the same time overcoming the limitations that still slow down the deployment of this technology in future "smart factories". Obviously, the complete envisioned system is wide and complex and, as the project moves towards maturity, will require more detailed studies focused on specific aspects of human-robot collaboration, that are not addressed here due to the early stages at the time of the authors' research.

Chapter 3

Materials and methods

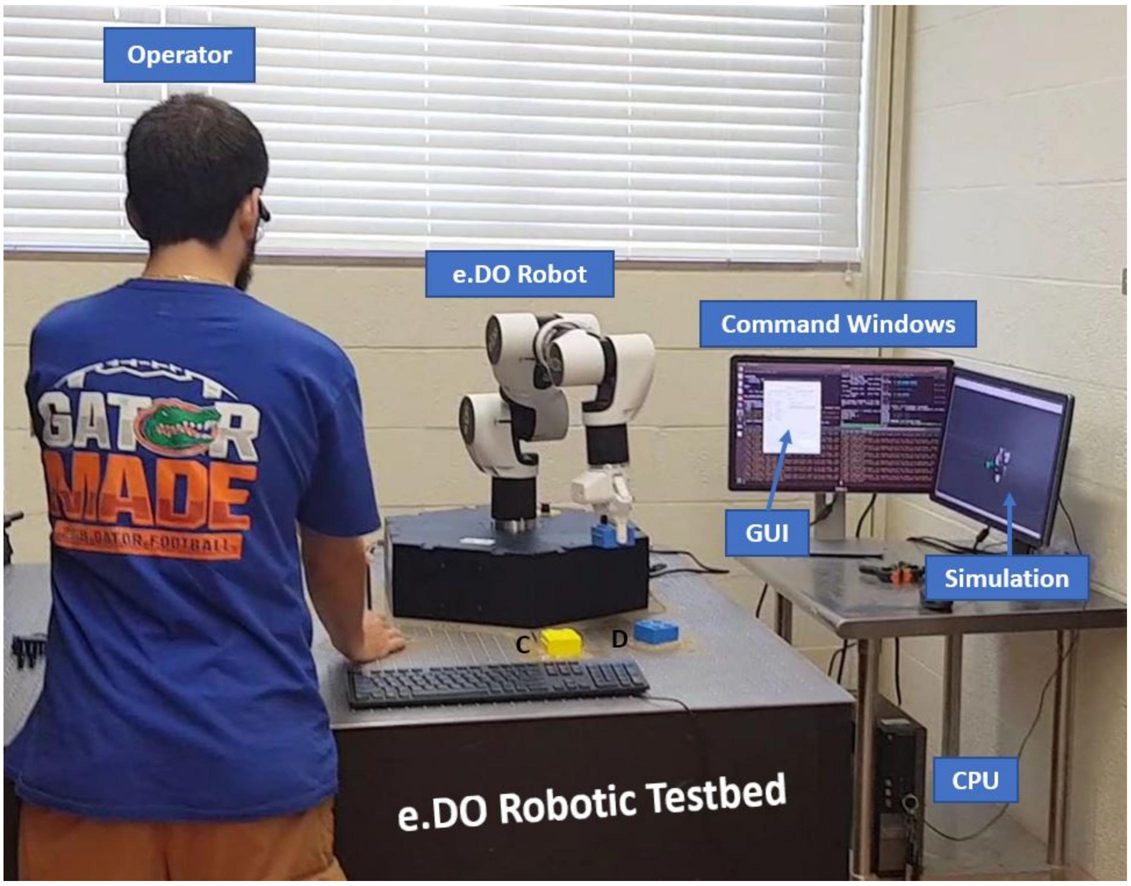


Figure 3.1: Representation of the main software and hardware components of the testbed

Figure 3.1 is used to represent the main software and hardware components exploited to build the robotic testbed. As shown, a human operator works in the same workspace of a physical e.DO robot in order to complete a collaborative assembly scenario, representative of common industrial applications. The robot is wired to a computer, represented on the right in Figure 3.1, on which the developed robotic system used to control the manipulator is running. One screen shows the command windows related to the underlying ROS and MoveIt! operations in play, together with the main window of the GUI, used to program the collaborative operation offline. The second screen is, instead, showing the simulation according to which the robot is controlled. As said, due to the early stages at the time of the authors' research, the sensor fusion input envisioned was not available yet. Therefore, in order to allow system testing and validation, perception of the human operator's presence have been emulated using ROS, and graphically displayed on the second screen using the RViz 3D visualizer. A keyboard is also connected to the computer and directly accessible by the operator for manual data input and quick confirmations signals. In this chapter, the tools just introduced and exploited by the authors for the development of the robotic system are briefly presented.

3.1 ROS

In order to develop a robotic system characterized by a high level of scalability and in line with the state of the art, the authors of this thesis chose to use ROS Melodic Morenia running on an Ubuntu 18.04 Bionic Beaver system. The acronym ROS [41] stands for Robotic Operating System, a free open-source platform for robotic systems. Many software packages and libraries for communication, motion planning and control, perception, navigation and mapping have been developed for use with ROS. This tool allows the development of a modular and scalable robotic system, in line with the state of the art of industrial robotics. Because of all these benefits, it is strongly supported by the academic, research and industrial communities. The ROS environment is composed of independent units called "nodes" which communicate between each other through links called "topics", based on a subscription/publication protocol. Basically, the ROS framework is a common interface between different nodes corresponding to different hardware or software components. The functionalities of a manipulator may be divided between several nodes, each one with a different function and goal, thus guaranteeing an easy debugging operation. Nodes can be written in different programming languages, the most common ones are C++ and Python (regarding this thesis, all the codes have been written in C++). In Figure 3.2 an example of communication between four nodes through one topic is shown.

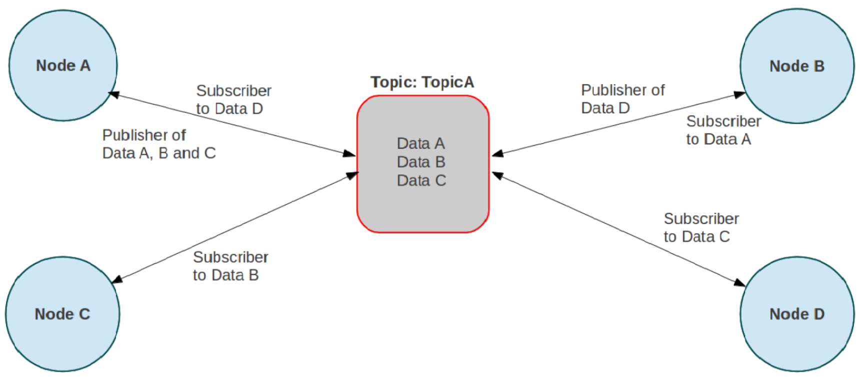


Figure 3.2: Schematic example of the subscription/publication protocol

Topics can be considered as buses used to send messages between different nodes. Nodes can use the publish function to publish data to a topic and the subscribe function to read information from the topic in which they are interested. A node may publish or subscribe to multiple topics and topics may have multiple publishers and subscribers. Each publisher can publish only one type of message and the same rule is applied for subscribers, which can read only the type of message that characterizes the topic they are interested in. Some examples of types of message are: boolean, float, integer or string. These are some of the basic message types provided by ROS, but users may use custom types as well.

3.2 MoveIt!

MoveIt! [42] is an open-source mobile robot manipulation library developed by Ioan A. Sucan and Sachin Chitta. It provides solutions for mobile manipulation related problems, such as kinematics, motion planning and control, 3D perception, collision checking and navigation. It is widely used in robotic systems as it can be easily configured and adapted to any kind of robot. Moveit! is built inside the ROS environment and uses a main node called "move_group" which provides actions and services to the user. The structure [43] of the MoveIt! package is represented in Figure 3.3.

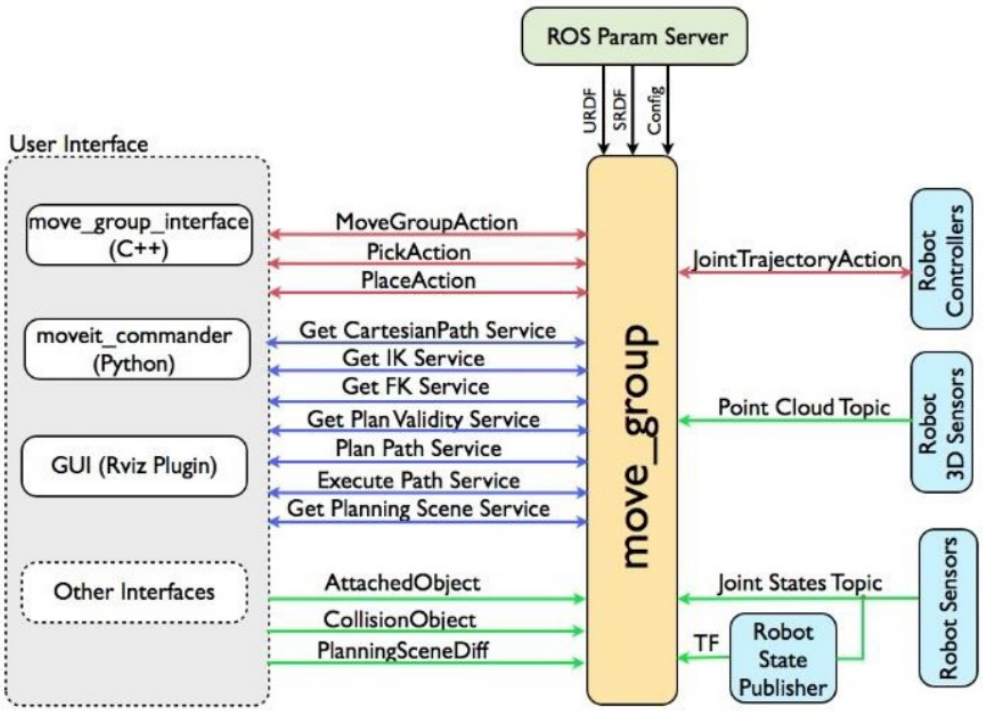


Figure 3.3: Functionalities and services offered by the "move_group" node

As depicted on the left of Figure 3.3, a user interface supporting C++ and Python programming languages as well as a plugin for graphical simulation called Rviz, is made available to the user. Through this intuitive tool, the user can exploit actions and services provided by the "move_group" node, represented as the core of the scheme in Figure 3.3.

Thanks to a parameter server, the “move_group” node is also capable of analyzing and storing a URDF model (Unified Robot Description Format) containing the robot kinematic representation, and a SRDF model (Semantic Robot Description Format) carrying all the preset robot parameters, such as position, speed and acceleration limits for the joints. Always with reference to Figure 3.3, the “move_group” node has the ability to communicate with the manipulator through a topic called */joint_states*, carrying all the data measured by its available sensors. Direct commands to the robot controllers can also be sent exploiting the JointTrajectoryAction server. Moreover, external sensors can be used to collect information about the external environment. In particular, the so called “Planning Scene Monitor”, represented in Figure 3.4, is the component of the MoveIt! package in charge of collecting all the information in order to plan and execute trajectories correctly. As shown, the “Planning Scene Monitor” receives information about the robot state and builds a 3D representation of the surrounding environment taking into account data from external sensors and geometry models uploaded by the user.

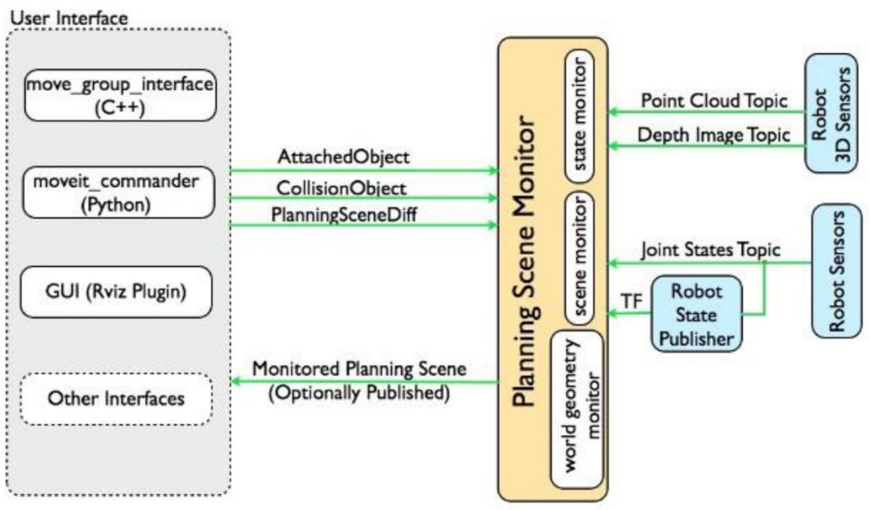


Figure 3.4: Functionalities and services offered by the "planning_scene_monitor" node

As introduced before, MoveIt! is an extremely rich package, offering several solutions for many different robotic-related tasks. For instance, direct and inverse kinematics, planning algorithms and collision detection capabilities are some of the features already available in the package and immediately accessible for use. In the following pages, a brief introduction of the MoveIt! plugins exploited for this research is reported.

3.2.1 OMPL library

MoveIt exploits the Open Motion Planning Library (OMPL) [44] which makes available a variety of sampling-based planning algorithms. The developer can choose to plan a certain trajectory either using a pre-existing algorithm offered by the OMPL Library or a custom one, which can be easily added into the software. Below, a list of planners offered by this library together with a brief description of their characteristics is reported.

Multi-query Planners

These planners build a roadmap of the entire environment that can be used for multiple queries.

1. *Probabilistic Roadmap Method (PRM)*
   This is a sampling-based algorithm. The OMPL's implementation uses one thread to construct a roadmap while a second thread checks whether a path exists in the roadmap between a start and goal state. A number of variants of PRM are available in the library:
   - LazyPRM
     This planner is similar to regular PRM but checks the validity of a vertex or edge "lazily," i.e. only when it is part of a candidate solution path.
   - PRM*
     While regular PRM attempts to connect states to a fixed number of neighbors, PRM* gradually increases the number of connection attempts as the roadmap grows in a way that provides convergence to the optimal path.
   - LazyPRM*
     A version of PRM* with lazy state validity checking.
2. *SPArse Roadmap Spanner algorithm (SPARS)*
   SPARS is a planner that provides asymptotic near-optimality (a solution that is within a constant factor of the optimal solution) and includes a meaningful stopping criterion. Although it does not guarantee optimality, its convergence rate tends to be much higher than PRM*.
3. *SPARS2*
   SPARS2 is variant of the SPARS algorithm that works through similar mechanics, but uses a different approach to identifying interfaces and computing shortest paths through said interfaces.

Single-query Planners

These planners typically grow a tree of states connected by valid motions. They differ in the heuristics they use to control where and how the tree is expanded. Some tree-based planners grow two trees: one from the start and one from the goal. Such planners will attempt to connect a state in the start tree with another state in the goal tree.

1. *Rapidly-exploring Random Trees (RRT)*
   This is one of the first single query planners. Many variants of RRT have been proposed. The OMPL library contains several RRT variants:
   - RRT Connect
     This planner is a bidirectional version of RRT (i.e. it grows two trees). It usually outperforms the original RRT algorithm.

- RRT*
  An asymptotically optimal version of RRT: the algorithm converges on the optimal path as a function of time. Since its publication, several other algorithms have appeared that improve on RRT*'s convergence rate, such as RRTX.
- Lower Bound Tree RRT (LBTRRT)
  LBTRRT is a asymptotically near-optimal version of RRT: it is guaranteed to converge to a solution that is within a constant factor of the optimal solution.
- Sparse Stable RRT
  SST is an asymptotically near-optimal incremental version of RRT.
- Transition-based RRT (T-RRT)
  T-RRT does not give any hard optimality guarantees, but tries to find short, low-cost paths.
- Vector Field RRT
  VF-RRT is a tree-based motion planner that tries to minimize the so-called upstream cost of a path. The upstream cost is defined by an integral over a user-defined vector field.
- Parallel RRT (pRRT)
  Many different parallelization schemes have been proposed for sampling-based planners, including RRT. In the OMPL's implementation, several threads simultaneously add states to the same tree. Once a solution is found, all threads terminate.
- Lazy RRT
  This planner performs lazy state validity checking (similar to LazyPRM). It is not experimental, but it does not seem to outperform other planners by a significant margin on any class of problems.

2. *Expansive Space Trees (EST)*
   This planner was published around the same time as RRT. It is not as sensitive to having a good distance measure, which can be difficult to define for complex high-dimensional state spaces. There are actually three versions of EST: the original version that is close to the first publication, a bidirectional version, and a projection-based version. The low-dimensional projection is used to keep track of how the state space has been explored. Most of the time OMPL can automatically determine a reasonable projection. A few planners that are not necessarily simple variants of EST, but do share the same expansion strategy, have been implemented:

   - Single-query Bi-directional Lazy collision checking planner (SBL)
     This planner is essentially a bidirectional version of EST with lazy state validity checking.
   - Parallel Single-query Bi-directional Lazy collision checking planner (pSBL)
     This planner grows the two trees in SBL with multiple threads in parallel.

3. *Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE)*
   KPIECE is a tree-based planner that uses a discretization (multiple levels, in general) to guide the exploration of the (continuous) state space. The OMPL's implementation is a simplified one, using a single level of discretization. The grid is imposed on a projection of the state space. When exploring the space, preference is given to the boundary of that part of the grid that has been explored so far. The boundary is defined to be the set of grid cells that have fewer than 2n non-diagonal non-empty neighboring grid cells in an n-dimensional projection space. There are two variants of KPIECE:
   - Bi-directional KPIECE (BKPIECE)
   - Lazy Bi-directional KPIECE (LBKPIECE)

4. *Search Tree with Resolution Independent Density Estimation (STRIDE)*
   This planner was inspired by EST. Instead of using a projection, STRIDE uses a Geometric Near-neighbor Access Tree to estimate sampling density directly in the state space. STRIDE is useful for high-dimensional systems where the free space cannot easily be captured with a low-dimensional (linear) projection.

5. *Path-Directed Subdivision Trees (PDST)*
   PDST is a planner that has entirely removed the dependency on a distance measure, which is useful in cases where a good distance metric is hard to define. PDST maintains a binary space partitioning such that motions are completely contained within one cell of the partition. The density of motions per cell is used to guide expansion of the tree.

6. *Fast Marching Tree algorithm (FMT*)*
   The FMT* algorithm performs a "lazy" dynamic programming recursion on a set of probabilistically-drawn samples to grow a tree of paths, which moves outward in cost-to-come space. Unlike all other planners, the number of valid samples needs to be chosen beforehand.

7. *Bidirectional Fast Marching Tree algorithm (BFMT*)*
   Executes two FMT* trees, one from the start and another one from the goal resulting in a faster planner as it explores less space.

As will be explained in Chapter 4, all the planning algorithms listed above can be leveraged by the user during the offline planning process of the developed robotic system. However, some further planning operations have been hardcoded by the authors and exploit specific preset algorithms. In particular, the authors chose to exploit RRT Connect and RRT* algorithms for those operations because of their efficiency in finding either feasible or optimal trajectories in a very short time.

3.2.2 RRT and RRT* algorithms

Rapidly exploring random tree (RRT) is an algorithm developed to efficiently search nonconvex, high-dimensional spaces by randomly creating a space-filling tree. This tree is expanded incrementally from samples drawn randomly from the search space and is inherently biased to direct towards large unsearched areas of the problem. In other words, when the planner creates a suitable path for the manipulator, points are randomly generated and connected to the closest available node. Each time a new vertex is identified, the code has to perform a check that guarantees that the vertex is not in correspondence of an obstacle. Furthermore, chaining the vertex to its closest neighbor must also avoid obstacles. The algorithm finds its solution when a node is created within the goal region, or a computational limit is reached. The RRT makes the planning process very fast and with a low computational cost, although the solution is not necessarily the optimal one. For these reasons, the authors of this thesis decided to use this specific planner only when the robot needs to generate a trajectory in the shortest time possible, and, more in general, in all the cases in which the path planning has to be performed online. As will be analyzed later (Section 5.2), the RRT algorithm allows the robot to perform an online, quasi-real-time replanning to avoid an obstacle in the workspace. A more detailed explanation of the algorithm and of its implementation can be found in [45].

RRT* is an optimal version of the RRT algorithm, capable of delivering the shortest possible path from a start position of the robot to the goal. Its basic principle is the same as RRT, but with two key additions to the algorithm that guarantee a significantly different result. First, RRT* checks the distance that each vertex has traveled from its parent vertex. This distance is called "cost" of the vertex. After the closest node is detected in the graph, a neighborhood of vertices at a fixed distance from the new node is analyzed. If a node with a "cost" lower than the proximal node is found (smaller distance) , it is used to replace the proximal node. The other difference between the two algorithms is that RRT* also performs the rewiring of the tree: when a new vertex is connected to the cheapest neighbor, the other vertex in the neighborhood are again examined by checking if being rewired to the newly added vertex will make their cost decrease. If it happens, the neighbor is rewired to the newly added vertex, also leading to the generation of a smoother path. For a more detailed explanation of RRT* algorithm and implementation please refer to [46].

3.2.3 Inverse kinematics

The Inverse Kinematics problem aims to find a suitable set of joint values that moves the robot's end-effector to a user-specified pose, defined in position and orientation. The equations that describe this problem are in general non-linear and, for this reason, numerical methods are more feasible approaches to find a solution. The most commonly used numerical IK implementation in the robotics community today is the so called Orocos Kinematics and Dynamics Library (KDL), an open-source kinematics framework [47] used for example by MoveIt! planning library. KDL implementation exploits the Inverse Jacobian method together with the Newton's iteration method in order to try to find a set of joint values that brings the end-effector "close enough" to the desired Cartesian solution. Inverse Jacobian methods generally start from a seed value of the joints q_{seed}, for example their current position. This seed is then used to calculate its corresponding pose by a process of simple forward kinematics. Then, the Cartesian error p_{err} between the seed pose and the target pose and the Jacobian J, which comprises the partial derivatives in Cartesian space with respect to the current joint values are also calculated. At this point the Jacobian can be inverted in order to obtain the partial derivatives in joint space with respect to the Cartesian pose and the kinematic inversion is then performed by the following iteration:

$$q_{next} = q_{previous} + J^{-1} \cdot p_{err}$$

When all the elements of p_{err} satisfy a certain stopping criterion, the last computed vector of joints is returned as solution to the IK problem. After a first analysis of the performances of this solver, however, the authors noticed that a series of failures would occur even when giving exact poses known reachable by the manipulator. Further studies on this problem led the authors to the decision of using a different plugin for the inverse kinematic problem. In particular the authors chose the TRAC-IK solver, another open-source framework described in [4]. The paper highlights the following issues for the KDL's IK implementation:

- Frequent convergence failures for robots with joints limits;
- No actions taken when the search becomes "stuck" in local minima.

The first problem is explained by the author of [4] by stating that the iterative algorithm seems to be making progress towards a feasible solution but encounters joint limits in the robot configuration. This issue cannot be overcome by simply improving the KDL's algorithm since it rises from a limitation of the applied method itself: the inverse Jacobian mathematics want to push the joints over their limits in order to reduce the Cartesian error, therefore causing the IK solver to fail. Therefore, a different method, called Sequential Quadratic Programming, is presented in the paper as a solution to this problem. SQP is an iterative algorithm for nonlinear optimization that can be applied successfully in the presence of constraints such as, in this case, joint limits.

$$min\,(q_{seed} - q)^T\,(q_{seed} - q) \qquad \text{and} \qquad f_j(q) < b_i$$

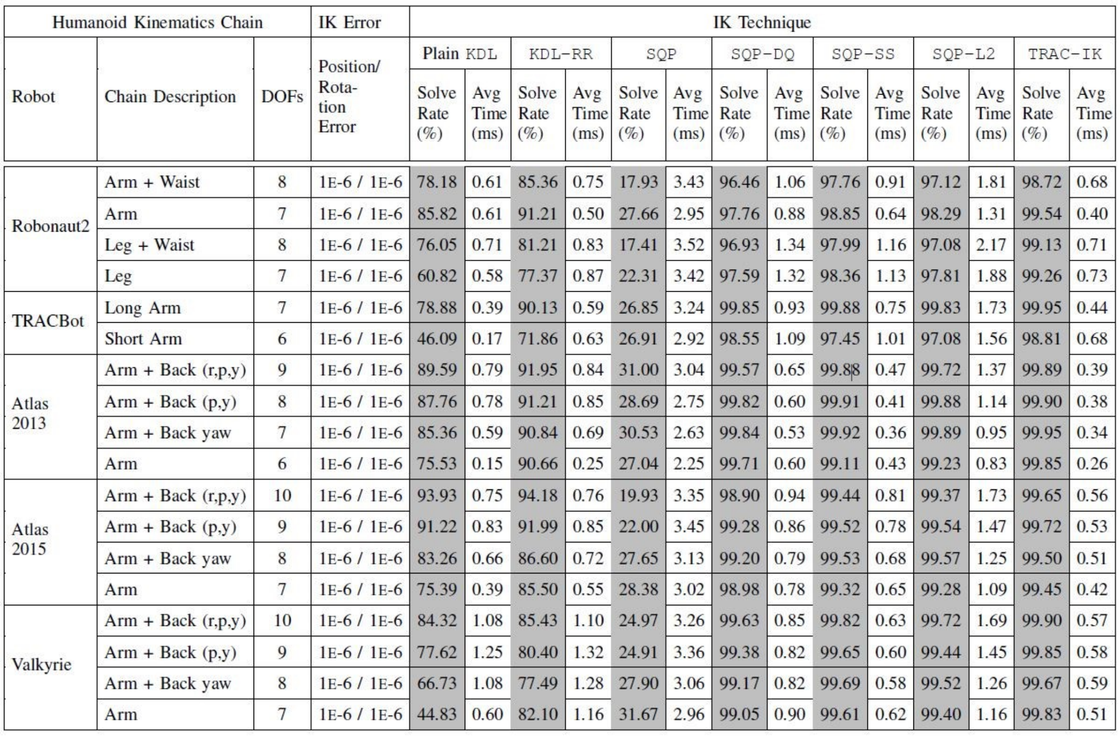
In particular, the author proposes to exploit the SQP-SS variant of the algorithm, where SS indicates Sum of Squares as metrics used for the error calculation, for which the objective function being minimized is formulated as follows:

$$\varphi_{SS} = p_{err} \cdot {p_{err}}^T$$

The second problem, on the other hand, is solved in [4] thanks to a simple re-implementation of the KDL's IK solver, leading to a new version of the algorithm called KDL-RR. From the name itself we can understand how this proposed solution works: RR means Random Restarts, meaning that this improved version of the algorithm is able to detect a local minimum as

$$q_{next} - q_{previous} \simeq 0$$

and mitigate it by providing a new randomly generated seed, so that the solver can restart its iterations avoiding the local minimum "trap".
An analysis of the performances of these two new methods is then presented in [4], with the results shown in Figure 3.5:



Humanoid Kinematics Chain			IK Error	IK Technique													
				Plain KDL		KDL-RR		SQP		SQP-DQ		SQP-SS		SQP-L2		TRAC-IK	
Robot	Chain Description	DOFs	Position/ Rotation Error	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)	Solve Rate (%)	Avg Time (ms)
Robonaut2	Arm + Waist	8	1E-6 / 1E-6	78.18	0.61	85.36	0.75	17.93	3.43	96.46	1.06	97.76	0.91	97.12	1.81	98.72	0.68
	Arm	7	1E-6 / 1E-6	85.82	0.61	91.21	0.50	27.66	2.95	97.76	0.88	98.85	0.64	98.29	1.31	99.54	0.40
	Leg + Waist	8	1E-6 / 1E-6	76.05	0.71	81.21	0.83	17.41	3.52	96.93	1.34	97.99	1.16	97.08	2.17	99.13	0.71
	Leg	7	1E-6 / 1E-6	60.82	0.58	77.37	0.87	22.31	3.42	97.59	1.32	98.36	1.13	97.81	1.88	99.26	0.73
TRACBot	Long Arm	7	1E-6 / 1E-6	78.88	0.39	90.13	0.59	26.85	3.24	99.85	0.93	99.88	0.75	99.83	1.73	99.95	0.44
	Short Arm	6	1E-6 / 1E-6	46.09	0.17	71.86	0.63	26.91	2.92	98.55	1.09	97.45	1.01	97.08	1.56	98.81	0.68
Atlas 2013	Arm + Back (r,p,y)	9	1E-6 / 1E-6	89.59	0.79	91.95	0.84	31.00	3.04	99.57	0.65	99.88	0.47	99.72	1.37	99.89	0.39
	Arm + Back (p,y)	8	1E-6 / 1E-6	87.76	0.78	91.21	0.85	28.69	2.75	99.82	0.60	99.91	0.41	99.88	1.14	99.90	0.38
	Arm + Back yaw	7	1E-6 / 1E-6	85.36	0.59	90.84	0.69	30.53	2.63	99.84	0.53	99.92	0.36	99.89	0.95	99.95	0.34
	Arm	6	1E-6 / 1E-6	75.53	0.15	90.66	0.25	27.04	2.25	99.71	0.60	99.11	0.43	99.23	0.83	99.85	0.26
Atlas 2015	Arm + Back (r,p,y)	10	1E-6 / 1E-6	93.93	0.75	94.18	0.76	19.93	3.35	98.90	0.94	99.44	0.81	99.37	1.73	99.65	0.56
	Arm + Back (p,y)	9	1E-6 / 1E-6	91.22	0.83	91.99	0.85	22.00	3.45	99.28	0.86	99.52	0.78	99.54	1.47	99.72	0.53
	Arm + Back yaw	8	1E-6 / 1E-6	83.26	0.66	86.60	0.72	27.65	3.13	99.20	0.79	99.53	0.68	99.57	1.25	99.50	0.51
	Arm	7	1E-6 / 1E-6	75.39	0.39	85.50	0.55	28.38	3.02	98.98	0.78	99.32	0.65	99.28	1.09	99.45	0.42
Valkyrie	Arm + Back (r,p,y)	10	1E-6 / 1E-6	84.32	1.08	85.43	1.10	24.97	3.26	99.63	0.85	99.82	0.63	99.72	1.69	99.90	0.57
	Arm + Back (p,y)	9	1E-6 / 1E-6	77.62	1.25	80.40	1.32	24.91	3.36	99.38	0.82	99.65	0.60	99.44	1.45	99.85	0.58
	Arm + Back yaw	8	1E-6 / 1E-6	66.73	1.08	77.49	1.28	27.90	3.06	99.17	0.82	99.69	0.58	99.52	1.26	99.67	0.59
	Arm	7	1E-6 / 1E-6	44.83	0.60	82.10	1.16	31.67	2.96	99.05	0.90	99.61	0.62	99.40	1.16	99.83	0.51

Figure 3.5: Summary of the results obtained in [4] for the performance of the analyzed plugins

In particular:

- SQP-SS outperforms all the tested algorithms in terms of success rate, with a final result, averaged on 180000 samples, of 99.21%;
- KDL-RR has the lowest runtime for all the cases in which a solution is found without hitting the joint limits.

Given these results, the TRAC-IK solver proposed in [4], tries to overcome the limitations of the analyzed algorithms by running both of them in separate threads. As soon as one of the two methods finishes with a solution, both threads are stopped, and the resulting solution is returned. As expected, TRAC-IK not only outperforms all the other IK methods tested in terms of solve rate, but also reduces the overall runtime.

Regarding this research, the switch from KDL's IK plugin to TRAC-IK plugin proved to be very helpful in all the processes in which the solver is exploited, by overcoming the limits previously experienced and yielding higher performances.

3.2.4 Collision detection

For this research, the collision checking process is carried out exploiting the FCL (Flexible Collision Library) package, a powerful library for collision checking explained in detail in [48]. As described in the article, a collision detection process between two different bodies A and B, given their configurations in space q_A and q_B as groups of points, is carried out by computing every separation distance in space between the two body configurations:

$$distance = inf\left\{\|x-y\|_2 : x \in A\left(q_A\right), y \in B\left(q_B\right)\right\}$$

If at least one of the computed distances is negative or equal to zero, it means that the two bodies are in collision between each other, while, on the contrary, if all the distances are greater than zero, it means that A and B are two non-overlapping objects. In the latter case (A and B not in collision), it could be useful to know the value of the minimum distance between the two bodies:

$$min_{x \in A(q_A), y \in B(q_B)} \|x-y\|_2$$

The biggest limit of the collision checking process is the huge amount of computational expense (often close to the 90% of the whole motion planning process [43]). Therefore, to reduce the computational burden, Moveit! allows the user to define an Allowed Collision Matrix (ACM). This matrix basically encodes a binary value related to the need of checking for collision between pairs of bodies, either belonging to the robot or to the external environment. In particular, if the value in the ACM corresponding to two objects is set to 1, this specifies that collision check between the two bodies is not needed. This, for instance, is convenient for cases in which it is guaranteed that two entities will never collide with each other.
In this thesis, every possible collision between the moving parts of the robot and any other object in the workspace must be avoided. However, the computational burden can be significantly reduced, while still maintaining this safety condition valid, by disabling a series of collision checks, as displayed in Figure 3.6. As can be seen, the authors have disabled the collision check between all the couples of adjacent links and between all the couples of links that for hardware limits can never be in contact with each other. Moreover, an additional collision check can be avoided: since the base of the robot is fixed inside the environment, it is not considered as one of the moving parts of the robot and therefore there is no risk in allowing contacts between it and the objects in the workspace.

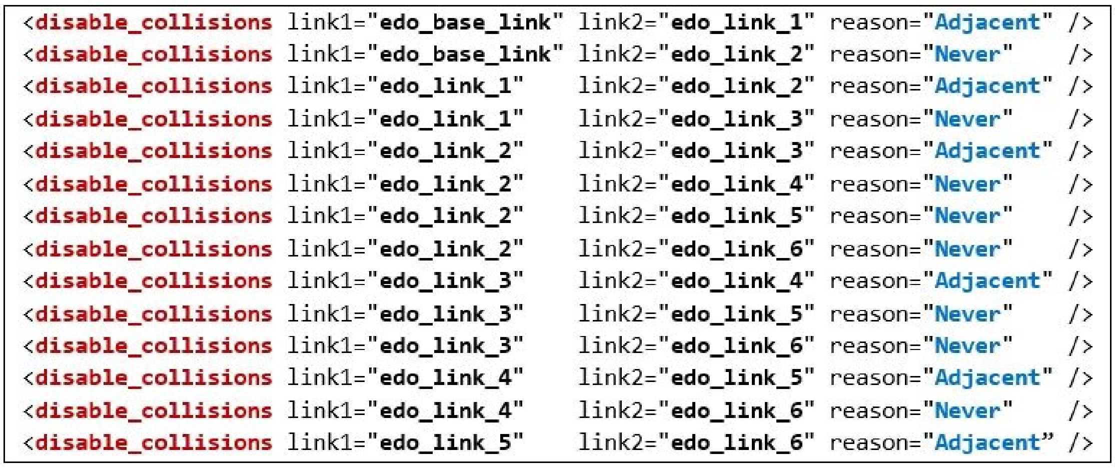

```
<disable_collisions link1="edo_base_link" link2="edo_link_1" reason="Adjacent" />
<disable_collisions link1="edo_base_link" link2="edo_link_2" reason="Never"    />
<disable_collisions link1="edo_link_1"    link2="edo_link_2" reason="Adjacent" />
<disable_collisions link1="edo_link_1"    link2="edo_link_3" reason="Never"    />
<disable_collisions link1="edo_link_2"    link2="edo_link_3" reason="Adjacent" />
<disable_collisions link1="edo_link_2"    link2="edo_link_4" reason="Never"    />
<disable_collisions link1="edo_link_2"    link2="edo_link_5" reason="Never"    />
<disable_collisions link1="edo_link_2"    link2="edo_link_6" reason="Never"    />
<disable_collisions link1="edo_link_3"    link2="edo_link_4" reason="Adjacent" />
<disable_collisions link1="edo_link_3"    link2="edo_link_5" reason="Never"    />
<disable_collisions link1="edo_link_3"    link2="edo_link_6" reason="Never"    />
<disable_collisions link1="edo_link_4"    link2="edo_link_5" reason="Adjacent" />
<disable_collisions link1="edo_link_4"    link2="edo_link_6" reason="Never"    />
<disable_collisions link1="edo_link_5"    link2="edo_link_6" reason="Adjacent” />
```

Figure 3.6: Summary of the disabled collision checks

3.2.5 Joint limits and time parametrization

Limits for position, velocity and acceleration can be specified for each joint in the kinematic chain in the URDF (Unified Robot Description Format) file, which is a textual CAD description of the robot itself. Additionally, a file called *joint_limits.yaml* allows the dynamic properties specified in the URDF file to be overwritten or augmented as needed. For this project, the limit values in the URDF file have been set to the hardware ones, reported in the technical sheets of the e.DO Robot. On the other hand, slightly lower values have been specified in the *joint_limits.yaml* file in order to guarantee a certain safety margin. Figure 3.7 reports an extract of the values set in the file. As explained later in Section 3.3, the first three joints of the e.DO robot are equipped with big motion units, while the last three are moved by smaller motors. As depicted in Figure 3.7, in which only the first and fourth joints are reported, different motors have been assigned different values of maximum allowed speed.

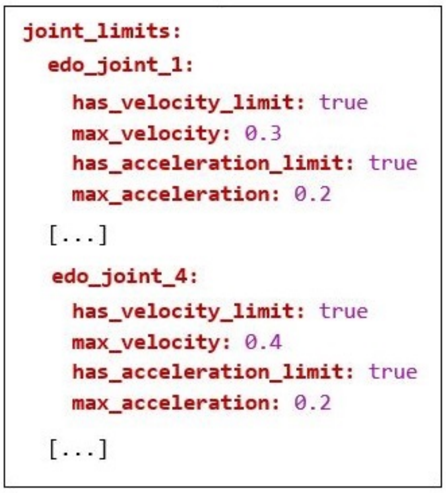

```
joint_limits:
  edo_joint_1:
    has_velocity_limit: true
    max_velocity: 0.3
    has_acceleration_limit: true
    max_acceleration: 0.2
  [...]
  edo_joint_4:
    has_velocity_limit: true
    max_velocity: 0.4
    has_acceleration_limit: true
    max_acceleration: 0.2
  [...]
```

Figure 3.7: Extract of acceleration and speed limits set for the robot’s joints

However, MoveIt! is primarily a kinematic motion planning framework. This means that the planning routines that it uses take into account the position of the joints or of the end effector, but not their velocity or acceleration. For this reason, post-processing is needed to time parametrize the planned trajectories, in order to be sure that the specified limits are respected. The default post-processing algorithm used by MoveIt! is called Iterative Parabolic Time Parametrization and works by assigning time stamps, properly defined, to each waypoint of the trajectory, as explained in [49].
First, the velocity constraint is taken into account by iteratively analyzing a couple of consequent points, until all the points of the trajectory have been considered. For each couple of points the algorithm calculates their distance in terms of joints position and uses that information to define the delta time that ensures a velocity within constraint.

$$\Delta time = max\frac{abs(q_{j,n}-q_{j,n+1})}{v_{max}} \qquad \text{for} \qquad j = \mathit{1,\ ...,\ joints\ number}$$

In the formula above, n and n+1 are used to distinguish between the two consecutive waypoints. After this procedure has been performed on the whole trajectory, the acceleration constraint has to be taken into account. In order to do that, three consecutive points need to be considered per iteration so that v_1 (speed to go from the first to the second point) and v_2 (speed to go from the second to the third point) can be defined. Then, the required acceleration can be calculated as:

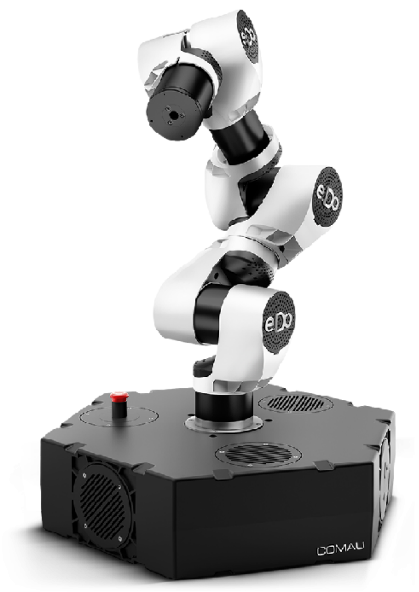
$$a = 2 \cdot \frac{v_2 - v_1}{\Delta t_1 + \Delta t_2}$$

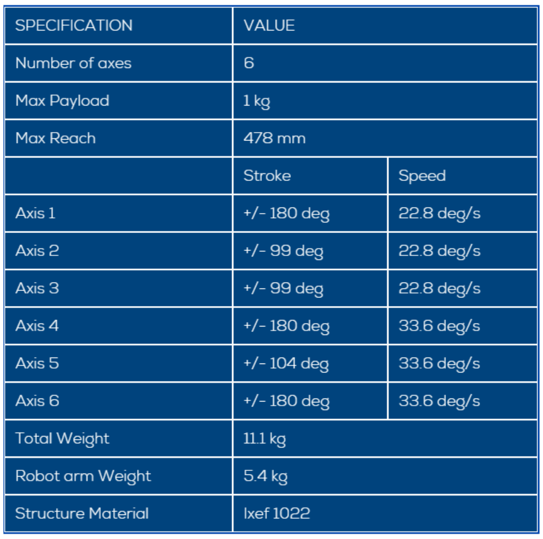
At this point, the algorithm keeps incrementing the deltas and recalculating the values of speed until it is assured that $a < a_{max}$. Finally, on top of the limit values reported in the robot model, additional indexes can be defined at runtime: *max_velocity_scaling_factor* and *max_acceleration_scaling_factor*. These indexes can be set to be equal to values between 0 and 1 and they are used to further reduce the limits written in the configuration files.

3.3 e.DO robot

The hardware-in-the-loop used to validate the collaborative outcome of this research is represented by a six axis robot called e.DO [50], manufactured and provided by the Italian company Comau, partner of the NSF/NRI project. The e.DO robot is equipped with six articulated axes that are capable of interfacing between themselves in a modular, easy and independent way. Each motorized unit is provided with an autonomous mechanical and electronic control that can be configured by the operator, allowing e.DO to pick up, move, manipulate and rotate parts, in order to accomplish the desired tasks. It must be noted that this robot has been created mainly for educational purposes. The authors believe that, thanks to the low investment required, such a platform could represent a perfect tool for the validation and evaluation of newly developed algorithms, before the implementation on the actual industrial robots.

As shown in Figure 3.8, the robot is equipped with a black hexagonal base, which ensures the full stability of the manipulator and contains its integrated open-source hardware and software platforms. In particular the base contains a Raspberry Pi motherboard, running the ROS framework, and a SD memory card, on which the pre-installed e.DO Control Logic, Infratask and the ISO have been uploaded.

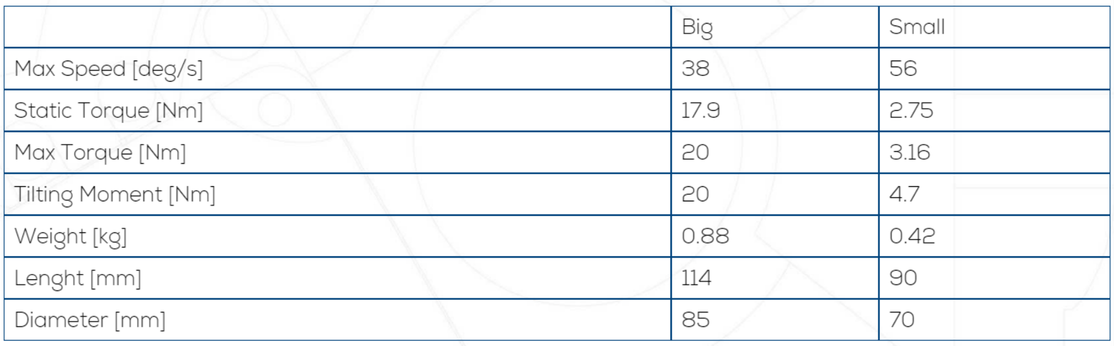




SPECIFICATION	VALUE	
Number of axes	6	
Max Payload	1 kg	
Max Reach	478 mm	
	Stroke	Speed
Axis 1	+/- 180 deg	22.8 deg/s
Axis 2	+/- 99 deg	22.8 deg/s
Axis 3	+/- 99 deg	22.8 deg/s
Axis 4	+/- 180 deg	33.6 deg/s
Axis 5	+/- 104 deg	33.6 deg/s
Axis 6	+/- 180 deg	33.6 deg/s
Total Weight	11.1 kg	
Robot arm Weight	5.4 kg	
Structure Material	Ixef 1022	

Figure 3.8: e.DO representation and general specification

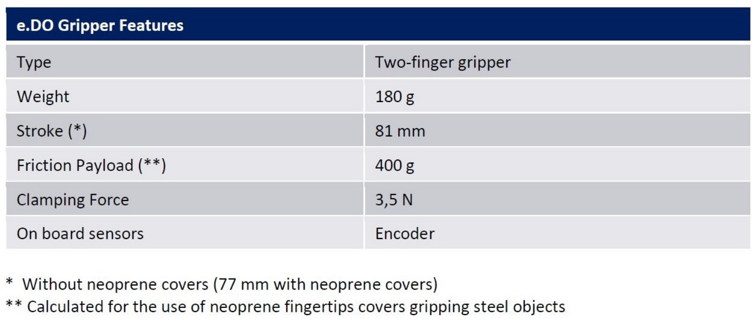
The manipulator has three big motion units and three small motion units, each of them represented by a DC motor. The table reported in Figure 3.9 contains all the specifications for these motors.



	Big	Small
Max Speed [deg/s]	38	56
Static Torque [Nm]	17.9	2.75
Max Torque [Nm]	20	3.16
Tilting Moment [Nm]	20	4.7
Weight [kg]	0.88	0.42
Lenght [mm]	114	90
Diameter [mm]	85	70

Figure 3.9: Motors specification for e.DO robot

In order to perform pick and place operations, the e.DO used for this research was equipped with a mechanical two-prongs gripper, shown in Figure 3.10.



e.DO Gripper Features	
Type	Two-finger gripper
Weight	180 g
Stroke (*)	81 mm
Friction Payload (**)	400 g
Clamping Force	3,5 N
On board sensors	Encoder

* Without neoprene covers (77 mm with neoprene covers)
** Calculated for the use of neoprene fingertips covers gripping steel objects

Figure 3.10: Gripper specification for e.DO robot

Chapter 4

Offline module: Task segmentation and planning

4.1 Introduction

Due to smaller lot sizes of customized products, the demand for increased flexibility and adaptability to changing production environments is rising exponentially. This requirement leads both to the need for a system that is reprogrammable in a fast and intuitive way and also to the necessity of a manipulator able to adapt to very different scenarios.
A general observation of real industrial cases led the authors to the realization that common industrial robotic applications can be thought of as composed of a series of discrete subtasks. These subtasks can represent a movement of the robot between two configurations in space, a specific action of the end-effector performed by the robot in a certain position or a combination of the two. Therefore, the focus of this work is to develop a system specifically built to allow the user to describe a scenario as a collection of consequent subtasks. In this project, the division process will be referred to as "segmentation" and each subtask will be called a "segment". This approach introduces the possibility to independently manage each segment in a different way, both in terms of how the robot's motion trajectory is planned offline and of how the robot reacts online to unexpected obstacles (e.g., human encounters) during the execution of the mentioned trajectory. In particular, this chapter deals with the offline planning process by developing a series of planning techniques aimed at addressing the main needs of common industrial scenarios. For instance, considering a welding scenario, the programming approach that would be more suited for the segment in which the robot is approaching the welding area from its original position is different from the segment in which the robot performs the welding on the workpiece. The two segments are part of the same task and are performed continuously one after the other, but they have completely different features. In the first case, the robot simply has to succeed in its motion from start to goal position, avoiding possible obstacles without constraints on its trajectory. In the second case, a predefined trajectory must be followed, and any stop or disturbance of the execution must be avoided. For this reason, having a set of available offline planning

techniques that can be assigned independently to each segment could represent a great source of flexibility and adaptability for the robotic system. Moreover, the fact that each scenario is described as a collection of consecutive segments simplifies reprogramming of the robot motion when considering cases of customized production. Supposing that the production cycle for a specific product has already been programmed, reprogramming the whole scenario from scratch for a customized version of the same product would be very time-consuming. Thanks to the segmentation process the new program can be obtained by simply modifying the segments of the original scenario according to the customized production process, therefore saving time in the offline programming step.

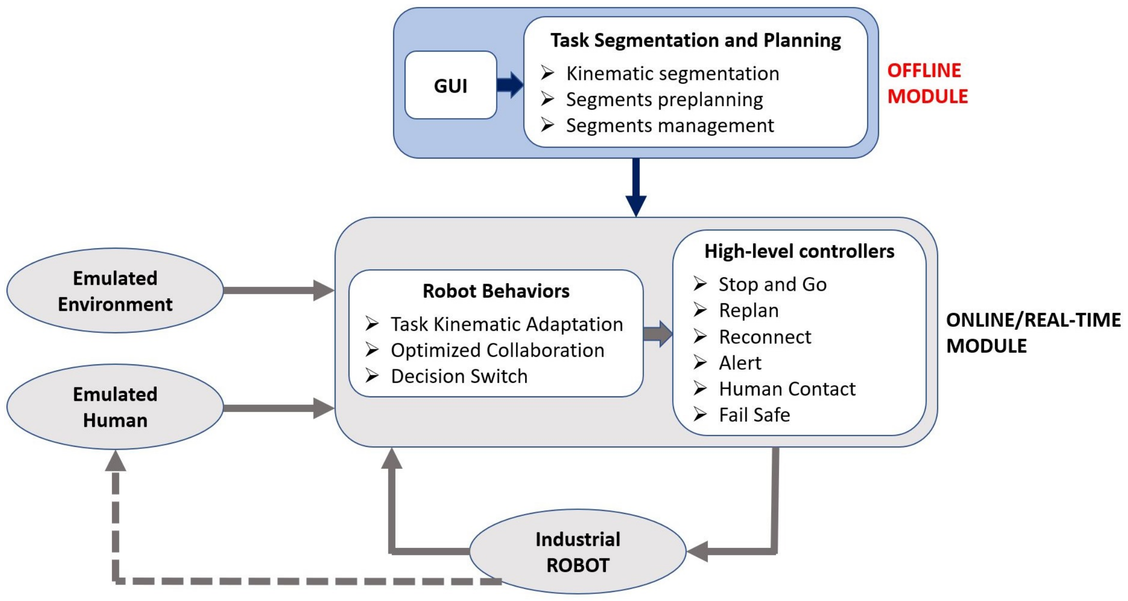


Figure 4.1: Focus on the module of the control scheme addressed by Chapter 4

With reference to the control structure represented in Figure 4.1, this chapter is focused on the Offline Module, implemented by the authors through a dedicated code, structured in four main parts:

1. **Input reading**: the first section of the code is responsible to read the input given by the user in an intuitive language and translate it into efficient coded information;
2. **Offline planning**: the second part of the code is in charge of generating the trajectory of each segment according to the choice of offline planning technique specified by the user;
3. **Segment connection**: the code then analyzes the computed trajectories to ensure a proper connection between consecutive segments;
4. **Segment execution**: this final section of the code is used to make sure that each segment is executed with the correct timing.

This code represents a node in the ROS environment and, as such, it is able to exchange information with the other available nodes. A more in-depth presentation of the tools used for the implementation can be found in Chapter 3, where all the background knowledge exploited for this project has been reported.

4.2 Offline planning

One of the key goals of collaborative robotics is the ability to reprogram the task with a great degree of freedom and in an intuitive and fast way. The relevant needs for these features are driven by the application of this type of robot, mainly concerned with flexible and rapidly changing environments, like the production of highly customized parts. Therefore the operator should be able to easily reconfigure the robotic task without the need of any programming specialization and without losing focus on the task to be performed [2]. With this goal in mind, the authors propose the implementation of a series of high-level programming methods, fast and intuitive, but general enough to not limit the operator and the robot in their abilities. The first step in this direction is an analysis of the different needs of a user that has to plan a trajectory for a real industrial scenario.

In the process of creating a segment, three main possibilities have been identified by the authors. The user may be interested only in the definition of start and goal configurations of the robot. Other situations may, instead, require the definition of the whole trajectory to be followed by the manipulator. Moreover, the definition of an action to be performed by the end-effector in a specific position of the workspace could be required. If the user is not interested in the specific path, but only in start and goal configurations of the robot for the segment, two possibilities arise. First, the authors imagine a situation in which the user does not have any a priori knowledge of the task (in the sense of human presence probability, average task execution time, synchronization with the human, etc.). Therefore the generation of the path that connects the two configurations only depends on the choice of planning algorithm, which can be based on the type of optimization that is required, the computational burden that can be sustained or many other aspects. On the other hand, a scenario in which a priori knowledge of the task is available to the user, must also be considered. In that case, the trajectory could be defined taking into account a series of stochastic data related to the human presence for the specific task, collected by means of experiments. By leveraging the availability of these data, it is possible to develop an offline planning approach able to generate a high-performance trajectory offline on the basis of a-priori knowledge of the task. Considering now the case in which the user has to define the whole trajectory of the segment, a different approach would be needed. The authors propose the development of an offline planning technique that enables the user to generate the segment's path point by point. This approach is particularly useful for segments in which the end-effector is not simply moving from point A to point B, but it is also performing a certain operation along with its motion (e.g. welding, pouring molten metals. . .). Finally, there are many cases in which the segment does not represent a movement of the robotic arm, but simply an action performed by the end effector. Also in this case, the authors propose a different offline planning technique, specifically developed to address this need. On the basis of the analysis just presented, Table 4.1 summarizes the four main needs identified by the authors, together with the specific offline planning technique that have been developed by the authors to address them.

OFFLINE PLANNING TECHNIQUE	ADDRESSED NEED
User-defined Algorithm	The user is interested only in the definition of start and goal configurations of the robot for the segment and a-priori knowledge of the task is not available.
Human Occupancy Volumes	The user is interested only in the definition of start and goal configurations of the robot for the segment and stochastic data about the human presence for the task have been collected.
Relevant Trajectory	The user wants to define the whole trajectory to be followed by the manipulator for the segment point by point.
Tool Operation	The user has to plan an action of the end-effector to be performed in a specific position inside the workspace.

Table 4.1: Developed offline planning techniques and application field

In the following pages a detailed description of each of these planning techniques is reported, considering both their specific application field and the logic that has been used to implement them.

4.2.1 User-defined Algorithm

This first offline planning technique addresses the problem of planning a trajectory given start and goal configurations of the robot, specified by the user, when no a-priori knowledge of the task is available. In order to leave as much freedom as possible, the authors' implementation gives the user the possibility of defining these two configurations either in joint space (defining the angular position of each joint of the robot) or in operational space (specifying the corresponding poses of the end-effector). Since the planning algorithms generally work inside the joint space, the difference between these two options mainly lays in the computational burden. If the user knows the angular position of all of the joints corresponding to the two configurations, the planner can directly work with them giving out the result in a shorter time and with a unique solution. If, instead, the user specifies the pose of the end-effector, before being able to generate a trajectory, the planner has to perform inverse kinematics in order to extract a feasible set of corresponding joint values (Section 3.2.3). This additional operation increases the computational effort and the obtained solution is not unique, since other configurations of the joints that bring the end-effector to the same pose may exist. Additionally, the developed code enables the user to specify a planning algorithm, choosing among the ones offered by the OMPL Library (Section 3.2.1). Limits related to maximum speed and acceleration for the specific robot have been hardcoded by the authors and are automatically respected by the planning algorithm. However, in order to give more control to the user, two additional scaling factors for these parameters can be specified so that it is always possible to further reduce these limits when needed (Section 3.2.5). Given all this data, the developed code exploits the capabilities of MoveIt! to compute a feasible trajectory that connects the two specified configurations of the robot using the chosen planning algorithm and respecting all the limits set for the robot. This trajectory is generated in the form of a "path object", schematically represented in Figure 4.2.

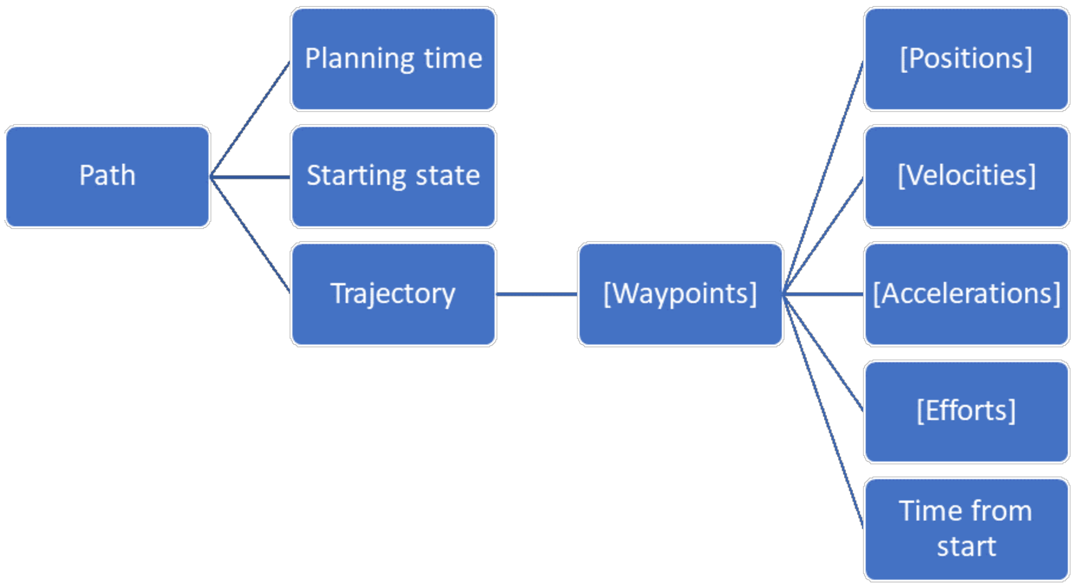


Figure 4.2: Structure of a Path object

As depicted in Figure 4.2, the path object is composed of three items: "planning time", reporting the amount of time it took to generate the plan, "start state", containing the full start state used for planning and "trajectory", made up of a series of waypoints. Each waypoint represents an intermediate state that the robot will assume during the execution of the trajectory. Their number mainly depends on the "*maximum_waypoint_distance*" parameter set for the planning algorithm, defining the discretization of robot motion for collision checking. This is one of the aspects that makes MoveIt! a very powerful tool: the planning algorithm automatically takes into account any obstacle present in the planning scene in order to trace a trajectory that avoids collisions with them (Section 3.2.4). For this specific project, the obstacles are modelled as STL volumes and represent the static environment in which the task takes place. Finally, each waypoint contains the information about position, velocity, acceleration and effort of each of the joints that make up the robot, together with the "*time_from_start*" value. This last parameter indicates how much time, from the beginning of the execution of the segment, is needed to reach that particular intermediate state and is calculated in post-processing through a time parametrization that considers both the preset hardware limits and the user-defined scaling factors. For a more detailed description of the plugin used for the process of inverse kinematics, the available path planners, the collision detection method and the time parametrization performed, the reader can refer to Section 3.2.

4.2.2 Human Occupancy Volumes

This second approach is inspired by the paper [5]. It has been developed to address the case in which the user is still interested only in start and goal configurations of the robot for the segment but, thanks to an experimental campaign, has access to vast amounts of data related the specific task at hand. In this section, both the methodology proposed in the article (the same notation is used here for clarity) and the authors' implementation of an offline planning approach exploiting said method are presented and discussed.

The authors of [5] propose to describe a collaborative operation as a spatial and statistical distribution of human occupancy volumes related to the execution of the task, where task is defined as "the arm (robot or human) movements necessary to reach for a goal and to locally execute an action". Looking at Figure 4.3, it is immediately clear that a trajectory passing far from the usual location of the human operator ("human working volume") yields a higher execution time, because a longer path is needed to circumnavigate the whole volume, but a smaller variability since, ideally, the probability of encountering the operator is bound to zero. On the other hand, a trajectory crossing the worker's workspace is shorter and therefore faster but, at the same time, more likely to undergo variations due to stops or changes in speed needed to avoid collisions.

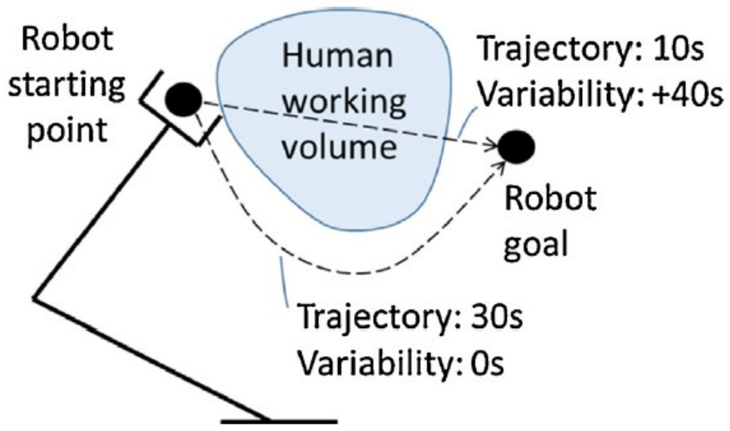


Figure 4.3: Comparison between two possible trajectories of the robot, [5]

For each point in space, the probability to be occupied by the human during his/her movements is calculated and assigned to its corresponding probability range. All the points belonging to a certain probability range are then grouped to form a volume, called “Human Occupancy Volume” (HOV). Each HOV represents the part of the workspace in which there is the corresponding probability of intersecting the operator during the execution of the task. It must be noticed that both the dimension of the probability intervals and the resolution with which the workspace is approximated to a grid are a trade-off between accuracy and computational time. For instance, if (0-20%, 20-40%, 40-60%, 60-80%, 80-100%) are the probability ranges considered, five volumes, each one related to one of these ranges, will be generated. Now, considering one of these volumes at a time, a robot trajectory that does not intersect it is defined. This trajectory, however, may intersect other volumes with lower probability indices, as we can see from the example in Figure 4.4, in which a trajectory avoiding $HOV_{99,100}$ is represented. The path avoids the mentioned volume, but is in collision both with $HOV_{80,99}$ (dotted lines) and with $HOV_{60,80}$ (solid lines).

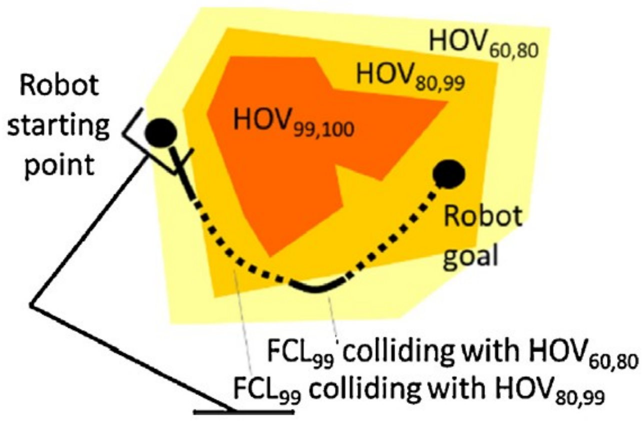


Figure 4.4: Graphical representation of the HOVs, [5]

It is essential to estimate the probability for the manipulator to interfere with the human operator during the trajectory execution, which would cause a stop or a decrease of the robot’s speed and therefore a loss in performance. The procedure proposed by the article is the following. First, an optimal free trajectory avoiding $HOV_{99,100}$ is generated. Then, the probability of collision with the volumes of lower index (indicated as $CP_{99,100}$) is calculated as the average of the human occupancy probabilities (HOP) associated with each of these volumes, weighted on the number of collisions ($NC_{k,j}$) detected with each one of them.

$$CP_{HOP(k)} = \frac{\sum_{j<k} 0.5\,(HOP\,(j+1) + HOP\,(j)) \cdot NC_{k,j}}{\sum_{j} NC_{k,j}}$$

The maximum delay, that could be caused by the collision with each volume, is represented by the time of human stay ($HST_{HOP(j),HOP(j+1)}$) inside that volume. This value can be used to estimate the maximum robot time ($MRT_{HOP(k}$) needed to complete its task. This parameter is calculated as the sum of the duration of the free trajectory ($RT_{HOP(k)}$) and of the time of human stay of each volume intersected.

$$MRT_{HOP(k)} = RT_{HOP(k)} + \sum_{j<k} HST_{HOP(j),HOP(j+1)}$$

Finally, from the information of “maximum robot time” and “collision probability”, the “likely robot time” ($LRT_{HOP(k)}$) can be estimated. If $RT_{HOP(k)}$ is the robot time related to the execution of the undisturbed trajectory and $MRT_{HOP(k)}$ is the maximum time required to complete the motion if all the collisions actually occur, $LRT_{HOP(k)}$ represents a more realistic estimated time, recalibrated on the basis of the probability that the delay will happen.

$$LRT_{HOP(k)} = RT_{HOP(k)} + \left(\sum_{j<k} HST_{HOP(j),HOP(j+1)} \right) \cdot CP_{HOP(k)}$$

Once this value is calculated, the volume used for the generation of the free trajectory is removed and the process is repeated iteratively for each of the computed HOVs. This means that for the next iteration the first volume is not considered anymore, and the free trajectory is computed around the next (in terms of probability range) HOV. When considering the last volume, which is the one related to the lowest probability range, no delay can be induced since the trajectory is already avoiding any possible interaction with the operator. This means that the last “likely robot time” computed coincides both with the “maximum robot time” and with the duration of the free trajectory.

In [5] these results are used to define a confidence interval for the robot execution time, relevant data for a scheduling problem. Regarding the specific objectives of this thesis, the authors believe that the “likely robot time” can also be considered a valuable indicator of the performance of the trajectory. In particular, among all the plans that have been computed during the process, the one with the shortest LRT is selected. In fact, on the basis of the available a priori knowledge of the task, it is the fastest path considering both its length and possible delays induced by human-robot interference. The thesis now focuses on the authors’ implementation of the approach just presented.

It is assumed that the following required inputs have been received: the STL files of the volumes obtained experimentally, the corresponding probability ranges, the total task execution time, start and goal configurations defined by the user and maximum allowed speed and acceleration. First, the HOV related to the highest probability range is added to the planning scene and an optimal trajectory is computed around it using the optimal planning algorithm RRT* (Section 3.2.2) offered by MoveIt!. Once the optimization process has finished and the trajectory has been created, the code removes the volume under analysis, adds the next HOV with lower probability index and checks if any collision exists with the waypoints composing the trajectory. This process repeats iteratively for each HOV. For each waypoint the authors assign "0" if there is no collision or "1" if a collision has been detected, therefore obtaining a vector of booleans for each of the lower probability volumes. In order to compute the number of collisions ($NC_{k,j}$) that is needed to use the formulas proposed by the article, the relevant data is not the number of colliding waypoints, but the number of trajectory portions in collision. These portions are identified inside the vector of booleans by a certain number of consecutive colliding waypoints, as shown in Figure 4.5.

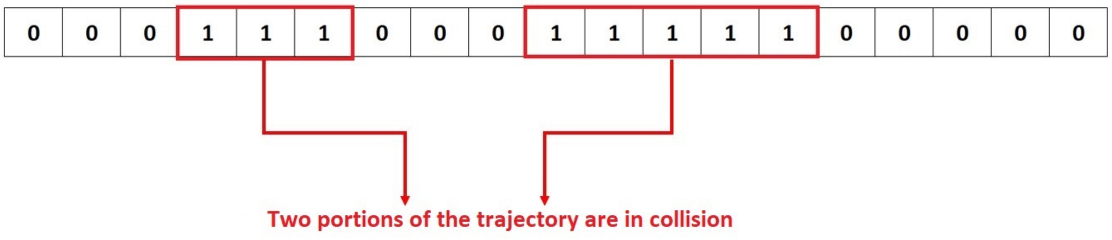


Figure 4.5: Example of vector of booleans with two portions of the trajectory in collision

To obtain the number of collisions, the code counts the number of changes from "0" to "1" or from "1" to "0" and divides the total count by a factor 2, supposing that there is no collision at the beginning and at the end of the trajectory. Finally, $RT_{HOP(k)}$ is extracted from the trajectory by looking at the "*time_from_start*" of its last waypoint, as explained in Figure 4.2. At this point, all the data required to compute the LRT of this first trajectory are available. Once the value of LRT has been computed, the volume around which the trajectory has been generated can be removed from the planning scene. The same process is iteratively repeated starting from the following volume with the highest probability, thus obtaining a number of paths equivalent to the number of HOVs given as input and a corresponding LRT for each one of them. Among all the available paths, the code outputs the one associated with the lowest value of "likely robot time". This path is the one that will be used during task execution since, based on the statistical analysis performed, it satisfies the user requests in the shortest time. The logic that has been used to implement this approach can be analyzed in detail with the flowchart in Figure 4.6.

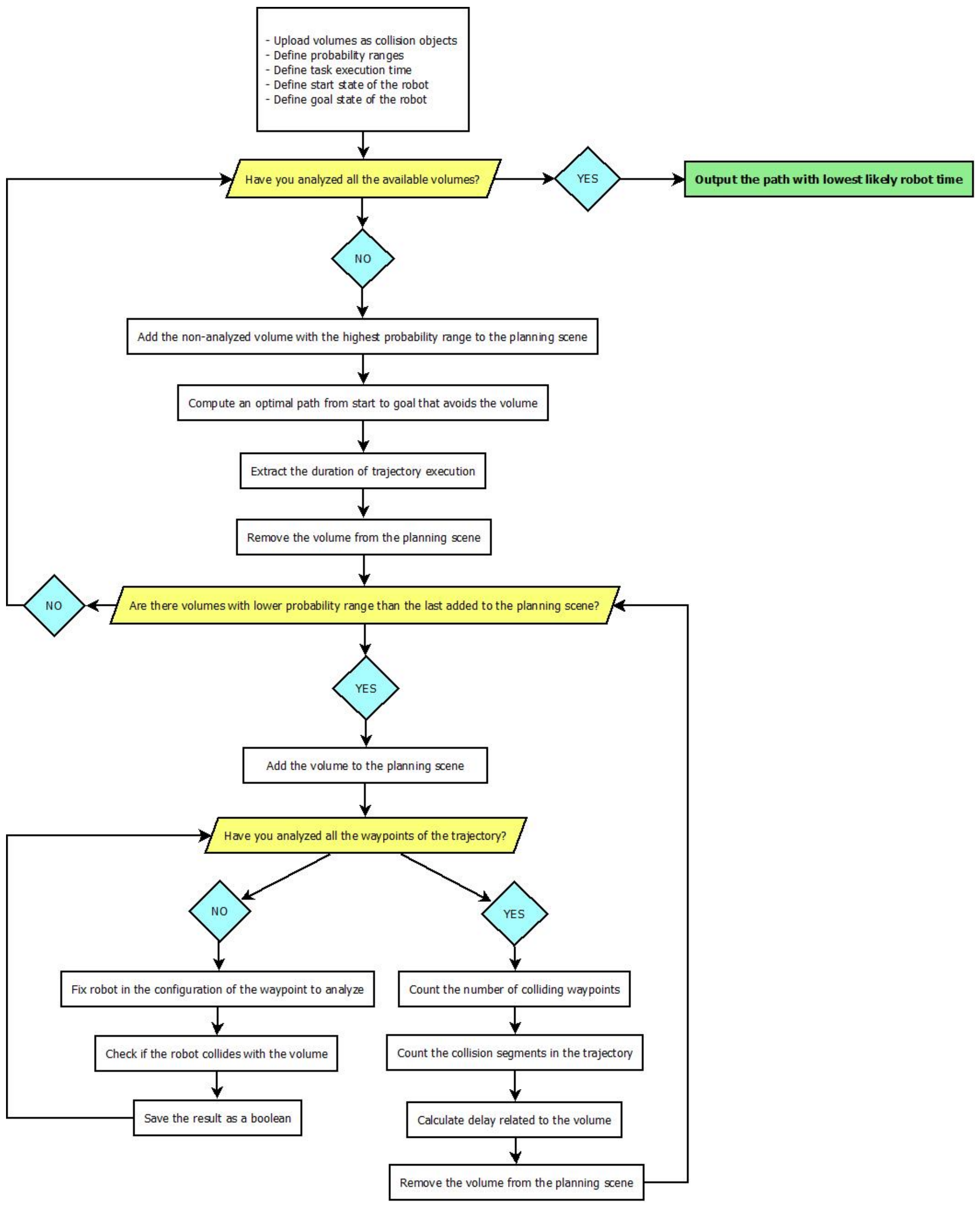

- Upload volumes as collision objects
- Define probability ranges
- Define task execution time
- Define start state of the robot
- Define goal state of the robot
Have you analyzed all the available volumes?
YES
Output the path with lowest likely robot time
NO
Add the non-analyzed volume with the highest probability range to the planning scene
Compute an optimal path from start to goal that avoids the volume
Extract the duration of trajectory execution
Remove the volume from the planning scene
NO
Are there volumes with lower probability range than the last added to the planning scene?
YES
Add the volume to the planning scene
Have you analyzed all the waypoints of the trajectory?
NO
YES
Fix robot in the configuration of the waypoint to analyze
Check if the robot collides with the volume
Save the result as a boolean
Count the number of colliding waypoints
Count the collision segments in the trajectory
Calculate delay related to the volume
Remove the volume from the planning scene

Figure 4.6: Flowchart of the HOV offline planning technique

4.2.3 Relevant Trajectory

The third offline planning technique proposed by the authors deals with the case in which the user has to define the whole trajectory of the segment. In general, this happens because the end-effector is not simply moving in space, but is also performing some kind of activity, like for example welding, cutting, pouring molten metals, etc. In a collaborative scenario, these operations are typically assigned to the robot because they are either dangerous or require a high degree of precision. With the approach proposed by the authors, first the user has to define the trajectory point by point in terms of position and orientation of the end-effector. Then, the code takes this collection of points as input and generates a corresponding trajectory that respects the limits of speed and acceleration set for the robot.

As just stated, the user is asked to input the trajectory point by point, either in joint space or operational space. Once the user has defined all the desired waypoints, the algorithm processes them in order to assign to each point of the trajectory a time stamp calculated so that all the limits of speed and acceleration can be respected. As shown in Figure 4.2, the mentioned time stamp is called "*time_from_start*" and represents the time from the beginning of the execution needed to reach the particular intermediate state. In order to assign correct values of time to each waypoint, the authors followed a method similar to the one explained in Section 3.2.5. The first waypoint is automatically assigned a *time_from_start* equal to 0 as it is the starting point of the trajectory. For the remaining waypoints, the code calculates the joint distance between all sets of consecutive points. Next, the minimum acceptable delta time is calculated on the basis of the distances just evaluated, the speed limit set in the configuration files and the scaling factor defined by the user. The *time_from_start* of a generic waypoint of the trajectory is equal to the sum of the *time_from_start* of the previous one and the calculated delta time. Regarding the last waypoint of the trajectory, an increased delta time is used in order to obtain a smooth stop of the robot at goal position. A representation of the logic used to implement this procedure is reported in Figure 4.7.
At this point, a trajectory that satisfies all the joint limits has been created. As for every other trajectory commanded to the robot, MoveIt! is able to take the path object as input, interpolate between the waypoints and execute it using its built-in position controller. Obviously, if for a given trajectory only a few waypoints are specified, the computational burden is low, but the path followed between each couple of consecutive waypoints is less predictable. If, on the other hand, the same trajectory is defined using a higher number of waypoints the user has more control on the exact path that will be followed, at the cost of a higher computational burden for the interpolation process.

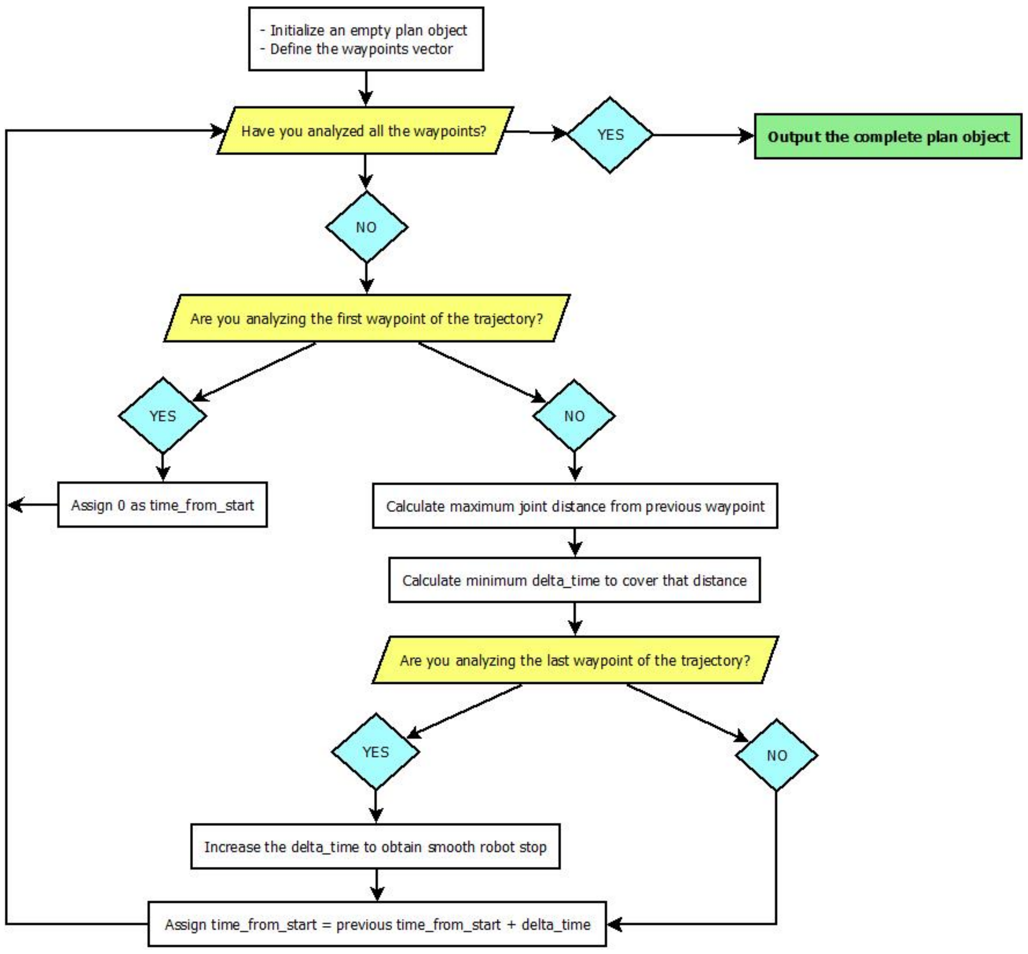
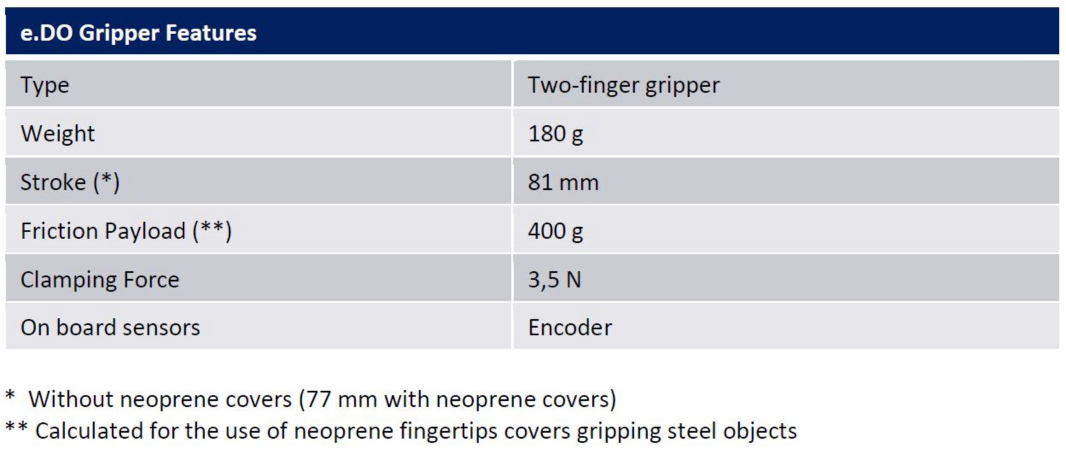

- Initialize an empty plan object
- Define the waypoints vector
Have you analyzed all the waypoints?
YES
Output the complete plan object
NO
Are you analyzing the first waypoint of the trajectory?
YES
NO
Assign 0 as time_from_start
Calculate maximum joint distance from previous waypoint
Calculate minimum delta_time to cover that distance
Are you analyzing the last waypoint of the trajectory?
YES
NO
Increase the delta_time to obtain smooth robot stop
Assign time_from_start = previous time_from_start + delta_time

Figure 4.7: Flowchart of the Relevant Trajectory offline planning technique

4.2.4 Tool operation

Differently from the previous cases, the aim of this last offline planning technique, called Tool Operation, is not to plan a movement of the robotic arm, but to plan an action to be performed by the end-effector in a certain pose inside the workspace. Regarding this specific research, the robot was equipped with a simple two-prongs mechanical gripper. Therefore, pick and place operations can be performed by controlling the opening and closing distance of the end-effector's fingers. In Figure 4.8 the features of the specific gripper used for this research and its specifications are reported.



e.DO Gripper Features	
Type	Two-finger gripper
Weight	180 g
Stroke (*)	81 mm
Friction Payload (**)	400 g
Clamping Force	3,5 N
On board sensors	Encoder

* Without neoprene covers (77 mm with neoprene covers)
** Calculated for the use of neoprene fingertips covers gripping steel objects

Figure 4.8: Technical sheet of the e.DO gripper

In terms of code, the user is simply required to input a choice of action and the position in which it has to be performed, either in joint space or in operational space. Starting from the capabilities of the mentioned end-effector, the author's decided to implement the three actions depicted in Figure 4.9: "Open fingers", "Close fingers" and "Wait". The first two possibilities represent simply the open and close functionalities of the gripper performed in the position specified by the user. For all the previously introduced offline planning approaches this code is in charge of providing the correct timing for the segment execution, as will be explained in the following sections. In this case, no movement of the robotic arm is required, and the code has to behave differently. A ROS-message containing the information about the desired action is published on an available topic, named */open_gripper*, that allows the authors to control the end-effector in a very fast and intuitive way. The third option, instead, is an operational mode in which the robot remains stationary in a predefined configuration and direct contact with the operator is allowed. This mode applies to all the cases in which, for instance, inspection of the end-effector or of the carried workpiece is needed. Since the robot is not allowed to move for all the time needed to perform the inspection, no harm can be done to the operator. The robot is allowed to restart its motion, and therefore execute the following segment, only when the operator gives a confirmation signal. Regarding the authors' implementation, this signal is sent simply by pressing the "ENTER" key on the keyboard.

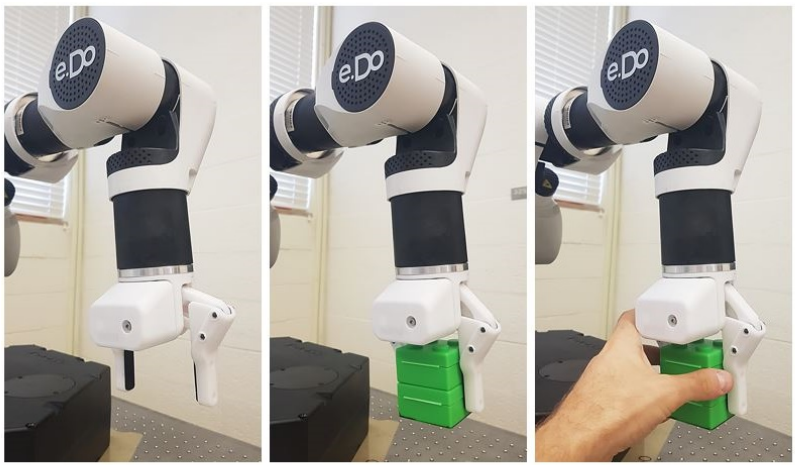
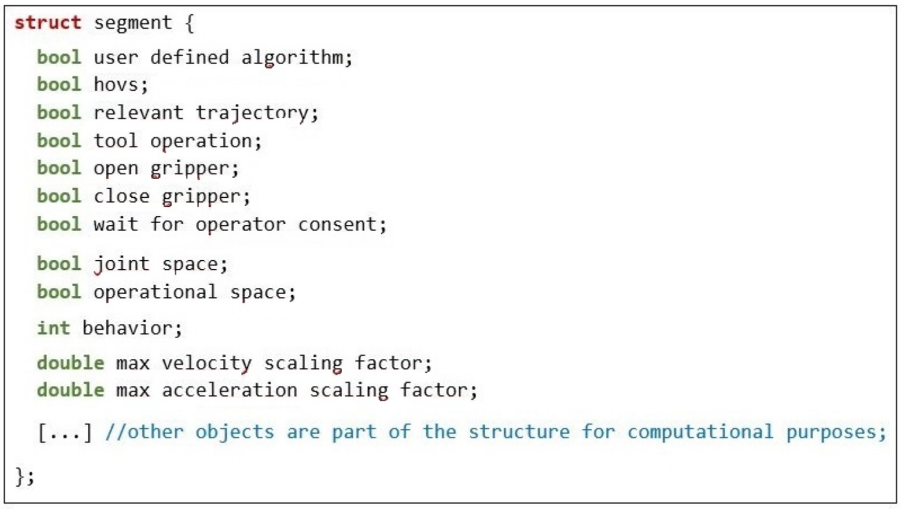


Figure 4.9: Implemented actions for the Tool Operation offline planning technique

4.2.5 Code modularity

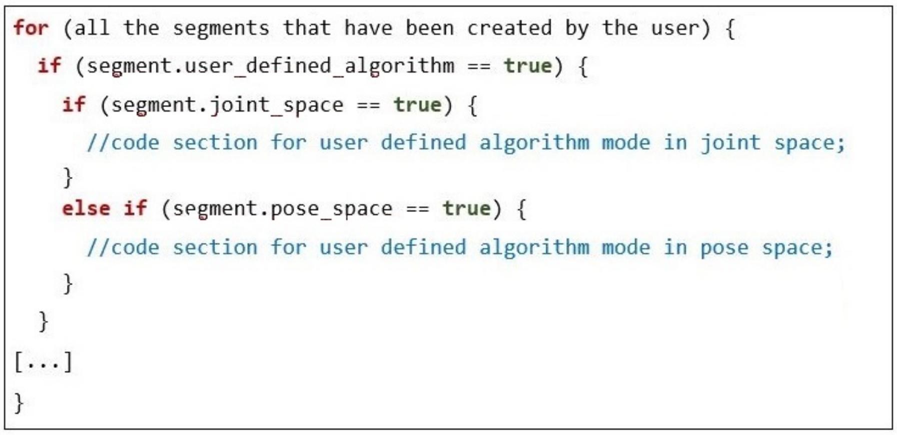
All the coding necessary to implement the robotic features presented up to now has been structured with the goal of enhancing the modularity of the system. This means that for any new offline planning technique required, a future developer can simply add the new implementation to the list of possibilities. In order to do that, each segment created by the user is translated into a C++ "segment object" characterized by a series of attributes, represented in Figure 4.10.



```
struct segment {
  bool user defined algorithm;
  bool hovs;
  bool relevant trajectory;
  bool tool operation;
  bool open gripper;
  bool close gripper;
  bool wait for operator consent;

  bool joint space;
  bool operational space;

  int behavior;

  double max velocity scaling factor;
  double max acceleration scaling factor;

  [...] //other objects are part of the structure for computational purposes;

};
```

Figure 4.10: Pseudo-code of a Segment object

As shown, each segment object contains a series of booleans that are used to activate the corresponding offline planning technique and to select a choice between joint space or operational space. The same object is used to store the scaling factors defined by the user, together with the behavior preference, that will be addressed in Chapter 5. In order to obtain a modular structure, each offline planning mode is implemented inside a dedicated C++ *if* statement. For each offline planning technique, the authors' implementation offers a choice between joint space and operational space, once again introduced in the code as an additional *if* statement. In Figure 4.11, a pseudocode representing an example of the practical implementation of this structure is reported. The *if* statement related to the User-defined Algorithm mode is represented and, inside its structure, two additional *if* statements are used to distinguish between joint space and operational space. Therefore, the code implementing the offline mode is written inside the specific corresponding section. The same idea is used for all the offline planning techniques developed by the authors. In practice, when a segment is created, the booleans related to the choices of offline planning technique and computational space are set to "true" so that the corresponding section of code can be accessed. This code structure simplifies the process of introducing new features in the robotic system. In fact, a future developer can simply create a new boolean variable inside the segment object and implement the new approach inside a corresponding *if* statement. Thanks to this approach, the process of discarding or updating obsolete features and introducing new ones in the robotic system can be performed quickly and in a more intuitive manner.



```
for (all the segments that have been created by the user) {
  if (segment.user_defined_algorithm == true) {
    if (segment.joint_space == true) {
      //code section for user defined algorithm mode in joint space;
    }
    else if (segment.pose_space == true) {
      //code section for user defined algorithm mode in pose space;
    }
  }
[...]
}
```

Figure 4.11: Pseudo-code of the modular structure of the system

4.3 Segment processing

Up to this point, this chapter has presented the different offline planning techniques made available by the authors for the generation of the trajectory of the segments. As stated at the beginning of the chapter, though, the code under analysis is also responsible for additional offline steps. First, this section deals with the analysis of the connection of the segments. Then, the authors' solution to achieve correct timing in the execution of the segments is presented.

4.3.1 Connection analysis

Supposing that the user has already defined all the segments that compose the robot's task at hand and that their trajectory has been computed offline by the algorithm, an additional step is required in order to ensure the correct connection of the sections. In this thesis, two consequent segments are considered properly connected if the configuration of the robot at the end of the first one coincides, within a certain tolerance, with the configuration prescribed for the beginning of the following one. The code first checks the difference of the value of each joint between the end position of a segment and the start position of the following one. If this difference is bigger than a certain tolerance, set to 0.05 rad, a new segment connecting the two configurations is created. Thanks to this analysis, the algorithm is therefore able to automatically compensate for any mistake or inaccuracy of the user. The connection path is planned using the RRT algorithm offered by MoveIt!, an efficient and fast planner addressed in detail in Section 3.2.2. The reason behind this choice of algorithm is that the authors were not interested in creating a connecting trajectory with particular features. Therefore, a planner able to find a feasible solution in a very short time represents an ideal solution since it has very limited impact on the offline performances of the system.

With reference to Figure 4.12, an example of the process just presented, simplified in two dimensions, is reported. As depicted, the user has created a first segment going from A to B and a second one going from C to D, but due to inaccuracy or a mistake they are not properly connected. The algorithm takes as input the segments created by the user and checks the distance between B and C. Since the value of distance between the two positions is bigger than the predefined tolerance, a third segment going from B to C, called "connection segment" in the example, is generated. It is of utmost importance in this phase to maintain the correct order of the segments, since it also represents the sequence with which the code will command the execution. In this example, the code takes in input the vector <*Segment_1* , *Segment_2*> and, after the connection process, expands it to become the vector <*Segment_1* , *Connection_Segment* , *Segment_2*>.

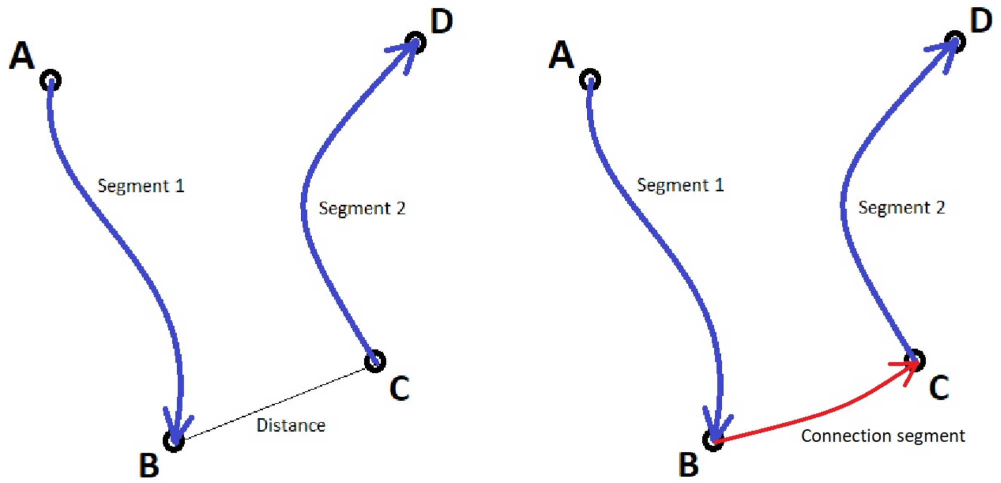


Figure 4.12: 2D example of the connection analysis process

The flowchart of the logic used to analyze the collection of segments and create connections where needed is reported in Figure 4.13.

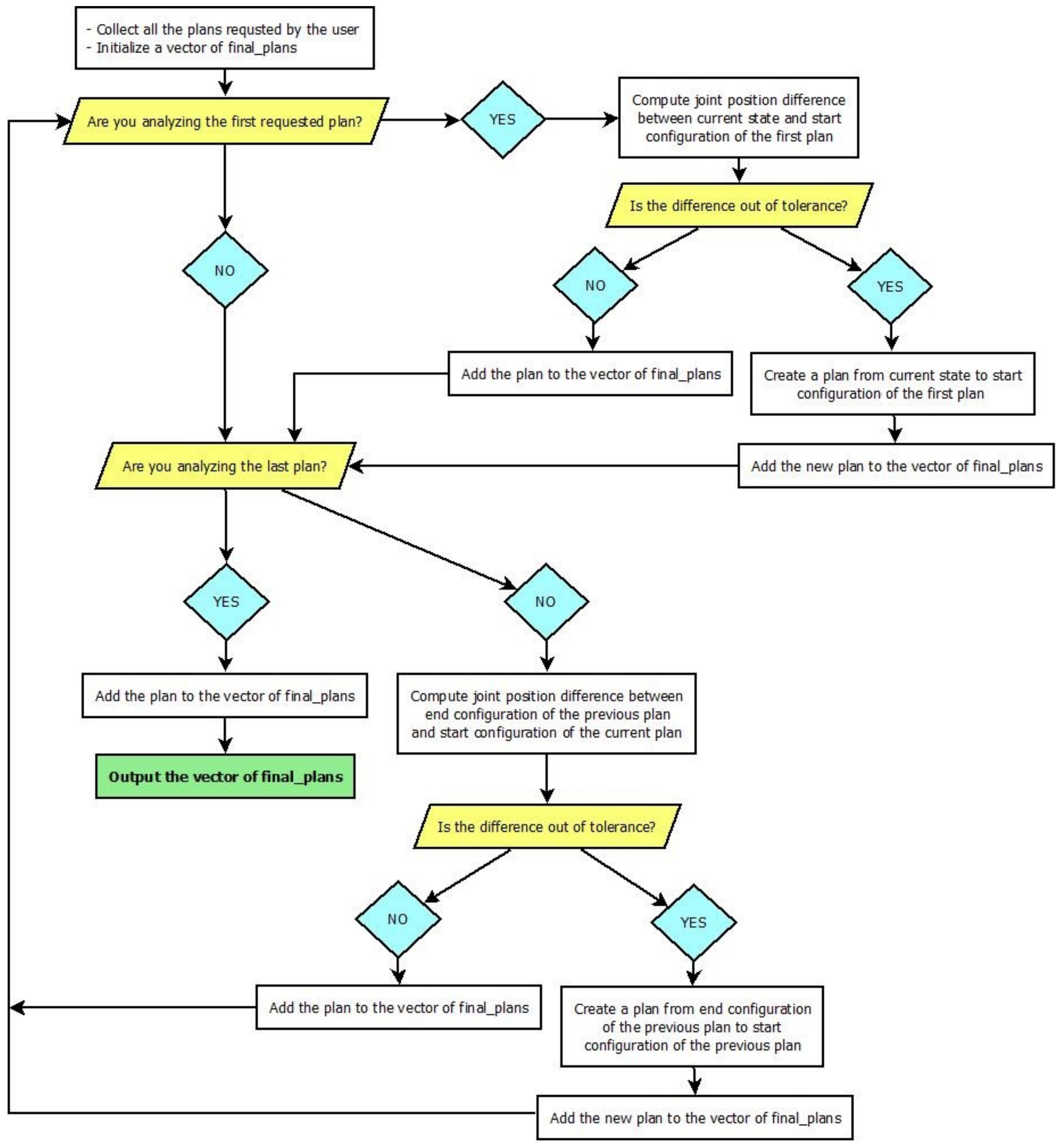

- Collect all the plans requsted by the user
- Initialize a vector of final_plans
Are you analyzing the first requested plan?
YES
Compute joint position difference between current state and start configuration of the first plan
Is the difference out of tolerance?
NO
YES
Add the plan to the vector of final_plans
Create a plan from current state to start configuration of the first plan
NO
Are you analyzing the last plan?
Add the new plan to the vector of final_plans
YES
NO
Add the plan to the vector of final_plans
Output the vector of final_plans
Compute joint position difference between end configuration of the previous plan and start configuration of the current plan
Is the difference out of tolerance?
NO
YES
Add the plan to the vector of final_plans
Create a plan from end configuration of the previous plan to start configuration of the previous plan
Add the new plan to the vector of final_plans

Figure 4.13: Flowchart of the logic used for the connection analysis

4.3.2 Segment execution

At this point, the collection of properly connected trajectories that composes the robotic task has been created. The last critical step assigned to the code under analysis, is to make sure that the execution of the segments is commanded in a safe manner and with the correct timing. As already stated, the code presented up to this point also represents a node in the ROS environment and, as such, is capable of communicating with the other components of the robotic system. Exploiting the ROS functionalities of publisher and subscriber, this node is responsible for the communication of the online characteristics of the segments to the other nodes in play. In the authors' implementation, the actual execution command for a path can be sent only when two conditions are satisfied. The first condition is that a feedback, carrying the information about the readiness of the controller, is received. Thanks to this safety check, the authors are able to guarantee that no uncontrolled, and therefore unsafe, movements of the robot are allowed to happen. Furthermore, an additional feedback is collected by the code and represents the second condition to be satisfied: for all the duration of the task, it must be ensured that the segment under execution has been successfully completed before commanding the execution of the following one. In Figure 4.14, a schematic representation of the ROS framework developed by the authors is reported.

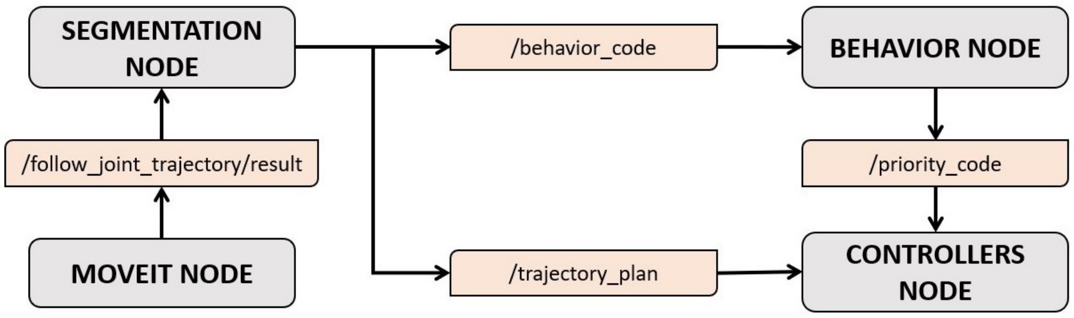


Figure 4.14: Diagram showing the ROS structure of the implemented system

Always with reference to Figure 4.14, the "Segmentation Node" represents the offline code presented in this chapter. The "Behavior Node" and the "Controllers Node" are two additional codes, developed by the authors and addressed in Chapter 5, related to the online behavior of the robot. The scheme also represents the connections created between these nodes and the messages that are exchanged. In particular, the Segmentation Node publishes two messages: one on a topic called */behavior_code*, containing the user preference for the online behavior of the robot for the specific segment, and another one through the topic named */trajectory_plan*, carrying the trajectory that has been computed by the offline planning process. The first condition to be satisfied before commanding the execution of a segment, is therefore to make sure that both the Behavior Node and the Controllers Node are effectively subscribed to their corresponding topic. In fact, only if this happens the messages can be received and therefore used to take care of the segment execution according to the user requests. The remaining branch of the graph represents the subscription of the Segmentation Node to another topic, called */follow_joint_trajectory/result*, on which a message, containing the second feedback needed, is published by a node created by MoveIt!. Among all the information within that message, a parameter containing the status of the trajectory under execution can be found, as shown in the Table 4.2. The second condition for the execution of a new segment is therefore satisfied only when the status of the execution of the previous path becomes “Succeeded”.

STATUS	CODE	DESCRIPTION
Pending	0	The goal has yet to be processed by the action server.
Pending	1	The goal is currently being processed by the action server.
Preempted	2	The goal received a cancel request after it started executing and has since completed its execution.
Succeeded	3	The goal was achieved successfully by the action server.
Aborted	4	The goal was aborted during execution by the action server due to some failure.
Rejected	5	The goal was rejected by the action server without being processed, because the goal was unattainable or invalid.
Preempting	6	The goal received a cancel request after it started executing and has not yet completed execution.
Recalling	7	The goal received a cancel request before it started executing, but the action server has not yet confirmed that the goal is canceled.
Recalled	8	The goal received a cancel request before it started executing and was successfully cancelled.
Lost	9	An action client can determine that a goal is LOST. This should not be sent over the wire by an action server.

Table 4.2: List of status codes related to the execution of a trajectory

A robotic system based on the envisioned segmentation process is now available. Thanks to the developed framework, the user has the possibility to divide a robotic application into sequential segments and to characterize each one of them independently. The code automatically takes care of the problem of planning, connecting, sequencing and commanding the execution these segments properly. Moreover, the system ensures that each segment is executed with the right timing and only when under the supervision of the desired controller, so that operational safety can be guaranteed.

4.4 Graphical User Interface

In order to enhance the accessibility of the robotic system and lower the skill threshold required to program the robot, an intuitive Graphical User Interface (GUI) has been developed. The GUI makes it possible to rapidly create, open, edit or save groups of segments by completely avoiding the need of “hard-coding” any movement of the robot. Thanks to this tool, the user is able to program the robot in a fast and easy way with no need of complex trainings or advanced knowledge. With this approach, the task segmentation is performed in an iterative offline process guided by the GUI, that leads the user through the full definition of the robot's task, specifying the levels of human-robot interaction allowed for each segment.

Upon opening the GUI, the main window is simple yet provides a quick overview of the capabilities of the system, as show in Figure 4.15. The scenario is defined segment by segment by clicking on the "Create New Segment" button. Any segments that have already been created are displayed in a dedicated white box, along with a brief summary of their characteristics. The "File" tab, located at the top left corner of the window, allows the user to save the scenario for future use or to open an already existing one. It should be noted that the convenience of this feature has a great impact on the degree of customizability that this robotic system provides. For instance, if the production of a product has already been programmed and a new customization is needed, the user has the possibility to open the existing scenario, modify the segments according to the new customization and save the new program instead of having to re-define the scenario from scratch. Moreover, the "Edit" tab can be used for further basic functionalities like "Undo", "Redo" or "Copy/Paste" operations on the segments. Once the whole scenario has been defined, the algortithm automatically takes care of all the remaining offline steps: segments planning, connection and management.

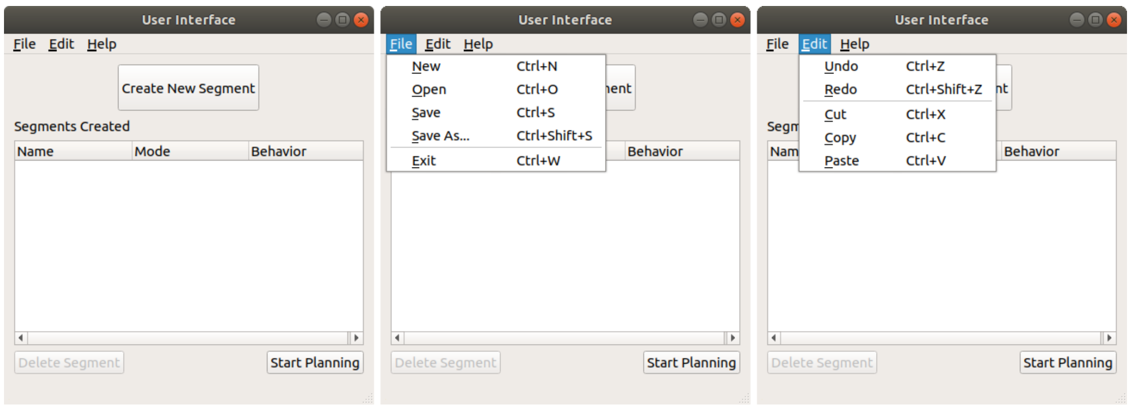
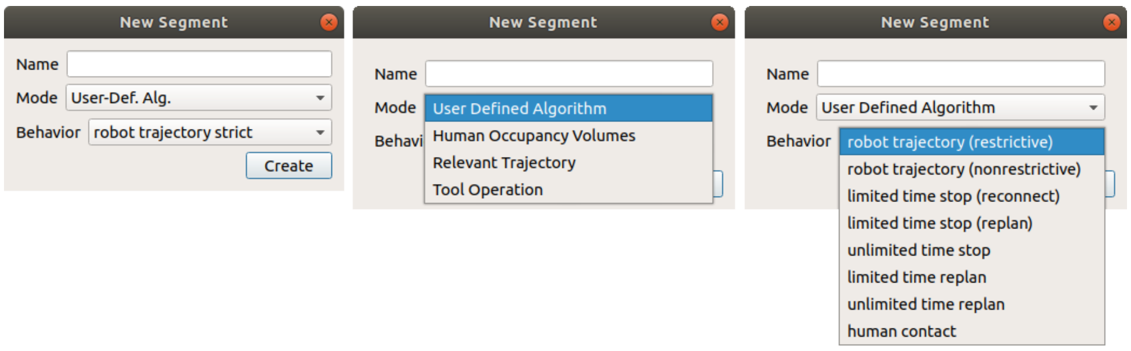


Figure 4.15: Screenshot of the main window of the GUI

As previously stated, the scope of the segmentation approach is the independent characterization of each segment both in terms of offline planning technique, main content of this chapter, and of online robot behavior, that will be addressed in detail in Chapter 5. In line with this concept, every time that a segment is created with the GUI, a new window pops up in order to request the characterization of the segment desired by the user. Each segment is assigned a "Name", a "Mode" (offline planning technique), and a "Behavior", both selectable from a list of available choices, as shown in Figure 4.16. Once the user has properly characterized the segment, the "Create" button shown in Figure 4.16 can be clicked. At this point, a new window is generated in order to allow the manual input of more detailed information about the segment, according to the assigned characteristics. The windows relative to each available offline planning technique are now presented.



New Segment

Name

Mode: User-Def. Alg.

Behavior: robot trajectory strict

Create

New Segment

Name

Mode: User Defined Algorithm

- User Defined Algorithm
- Human Occupancy Volumes
- Relevant Trajectory
- Tool Operation

New Segment

Name

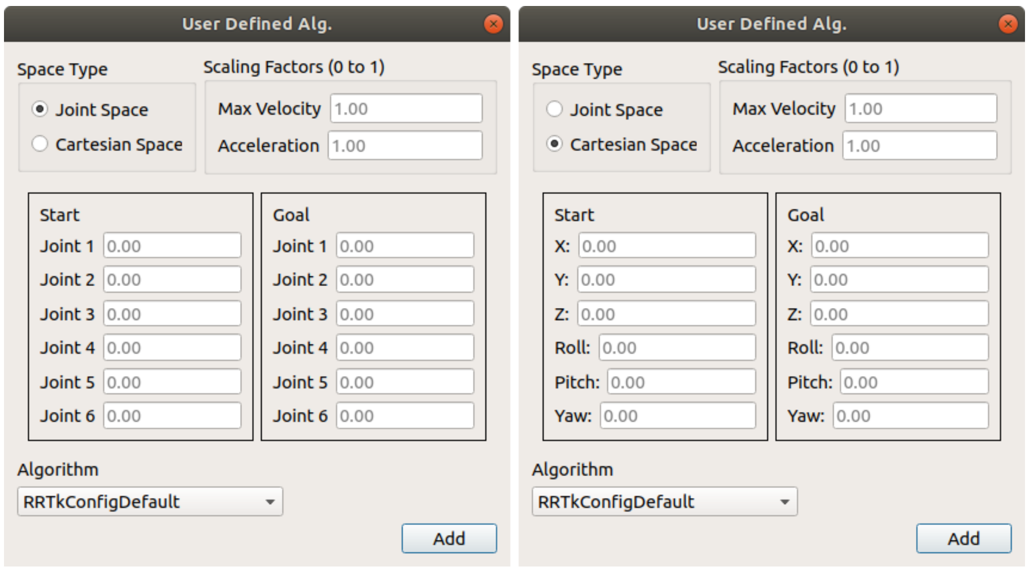
Mode: User Defined Algorithm

Behavior: robot trajectory (restrictive)

- robot trajectory (restrictive)
- robot trajectory (nonrestrictive)
- limited time stop (reconnect)
- limited time stop (replan)
- unlimited time stop
- limited time replan
- unlimited time replan
- human contact

Figure 4.16: Creation of a new segment with the GUI

In Figure 4.17, the window related to "User Defined Algorithm" mode is reported. First, the user can select a preference between joint space or cartesian space for the definition of start and goal configurations of the robot for the segment. A list, provided at the bottom of the window, can then be used to choose the desired planning algorithm among the ones offered by the OMPL plugin (Section 3.2.1). Finally, two scaling factors can be specified in order to further reduce the preset maximum allowed values of speed and acceleration.



User Defined Alg.

Space Type: (● Joint Space) ○ Cartesian Space

Scaling Factors (0 to 1): Max Velocity 1.00; Acceleration 1.00

Start		Goal	
Joint 1	0.00	Joint 1	0.00
Joint 2	0.00	Joint 2	0.00
Joint 3	0.00	Joint 3	0.00
Joint 4	0.00	Joint 4	0.00
Joint 5	0.00	Joint 5	0.00
Joint 6	0.00	Joint 6	0.00

Algorithm: RRTkConfigDefault

Add

User Defined Alg.

Space Type: ○ Joint Space (● Cartesian Space)

Scaling Factors (0 to 1): Max Velocity 1.00; Acceleration 1.00

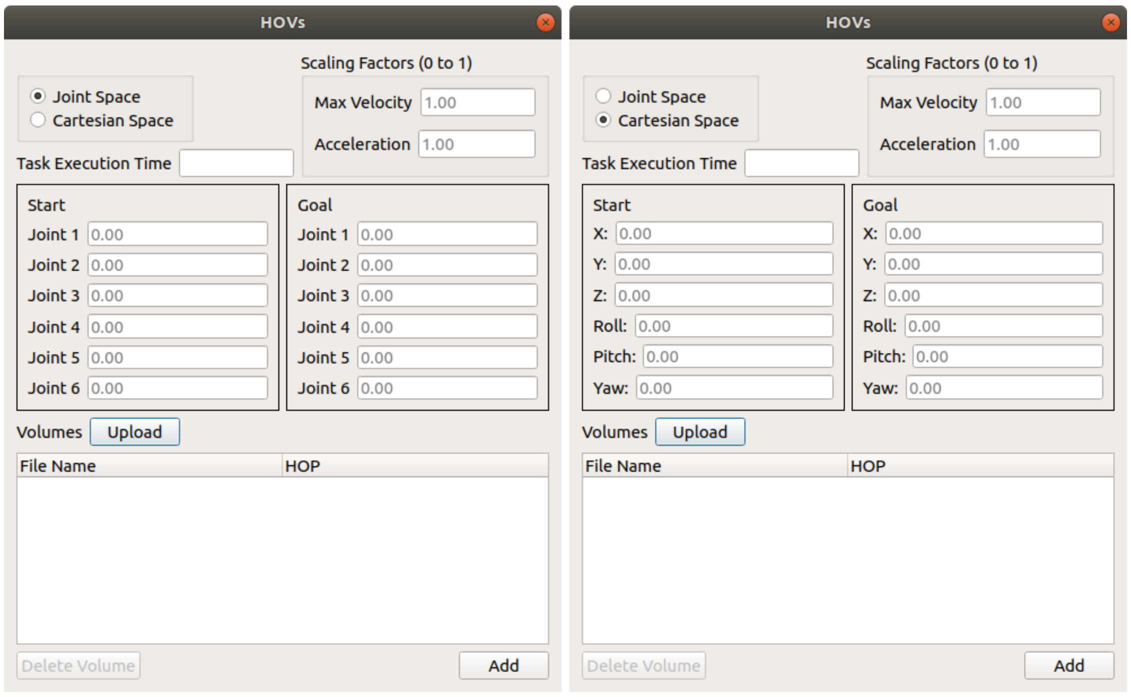
Start		Goal	
X:	0.00	X:	0.00
Y:	0.00	Y:	0.00
Z:	0.00	Z:	0.00
Roll:	0.00	Roll:	0.00
Pitch:	0.00	Pitch:	0.00
Yaw:	0.00	Yaw:	0.00

Algorithm: RRTkConfigDefault

Add

Figure 4.17: Screenshot of the User Defined Algorithm offline planning mode

The window related to the "Human Occupancy Volumes" mode, shown in Figure 4.18, maintains all the features of the previous one but, instead of asking the user for a choice of planning algorithm, a box for the upload of the probability volumes is available. The user has to click on the "Upload" button in order to select the STL files of the volumes experimentally created and assign to each one of them the corresponding human occupancy probability range. Moreover, a value of "Task Execution Time" is required in order to completely define the task to be analyzed, as explained in detail in the previous section dealing with this specific offline planning technique.



HOVs

Scaling Factors (0 to 1)

[✓ Joint Space] Cartesian Space

Max Velocity 1.00

Acceleration 1.00

Task Execution Time

Start		Goal	
Joint 1	0.00	Joint 1	0.00
Joint 2	0.00	Joint 2	0.00
Joint 3	0.00	Joint 3	0.00
Joint 4	0.00	Joint 4	0.00
Joint 5	0.00	Joint 5	0.00
Joint 6	0.00	Joint 6	0.00

Volumes Upload

File Name	HOP

Delete Volume Add

HOVs

Scaling Factors (0 to 1)

Joint Space [✓ Cartesian Space]

Max Velocity 1.00

Acceleration 1.00

Task Execution Time

Start		Goal	
X:	0.00	X:	0.00
Y:	0.00	Y:	0.00
Z:	0.00	Z:	0.00
Roll:	0.00	Roll:	0.00
Pitch:	0.00	Pitch:	0.00
Yaw:	0.00	Yaw:	0.00

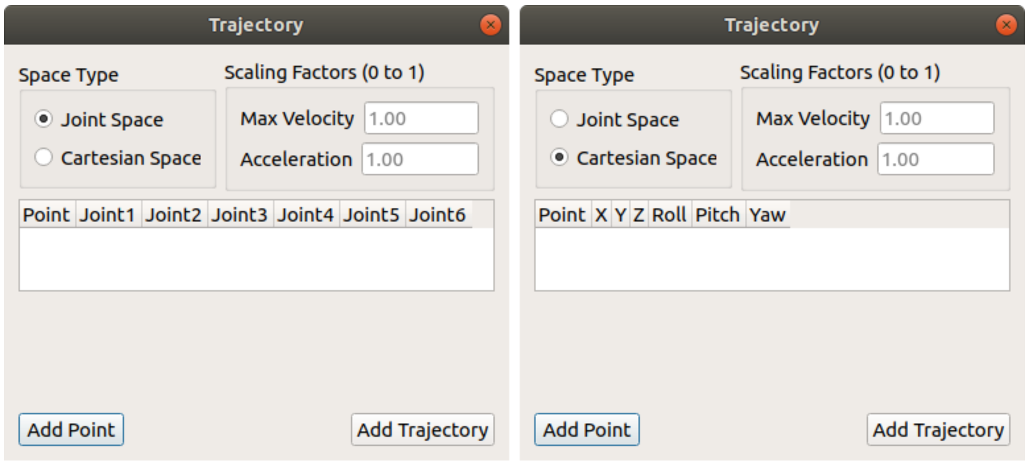
Volumes Upload

File Name	HOP

Delete Volume Add

Figure 4.18: Screenshot of the Human Occupancy Volume offline planning mode

The "Relevant Trajectory" window, presented below in Figure 4.19, is built using a different logic with respect to the previous ones. Again, the user chooses between joint space and pose space and, optionally, specifies scaling factors for maximum velocity and acceleration of the robot. At this point, the trajectory can be created point by point by clicking the "Add Point" button. Every time, a small additional window, like the ones in Figure 4.20, pops up in order to store the values for the configuration of the robot in that specific point. Once all the points have been stored, the user can click on "Add Trajectory" in order to confirm the creation of the segment.



Trajectory

Space Type

[✓ Joint Space] Cartesian Space

Scaling Factors (0 to 1)

Max Velocity 1.00

Acceleration 1.00

Point	Joint1	Joint2	Joint3	Joint4	Joint5	Joint6

Add Point Add Trajectory

Trajectory

Space Type

Joint Space [✓ Cartesian Space]

Scaling Factors (0 to 1)

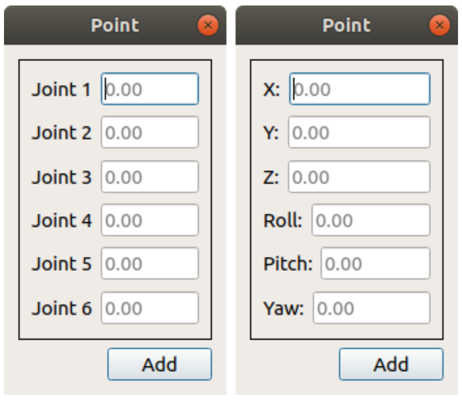
Max Velocity 1.00

Acceleration 1.00

Point	X	Y	Z	Roll	Pitch	Yaw

Add Point Add Trajectory

Figure 4.19: Screenshot of the Relevant Trajectory offline planning mode

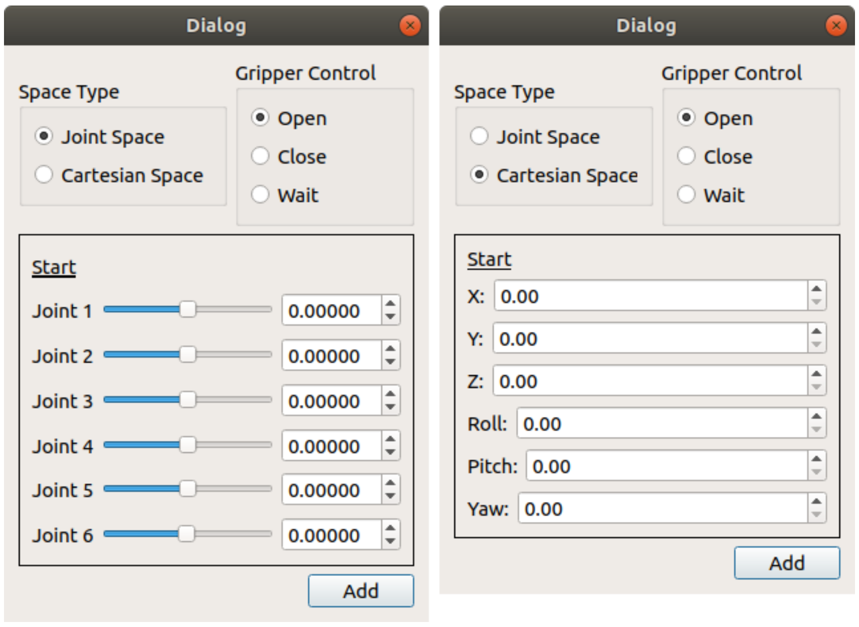


Point	
Joint 1	0.00
Joint 2	0.00
Joint 3	0.00
Joint 4	0.00
Joint 5	0.00
Joint 6	0.00

Point	
X:	0.00
Y:	0.00
Z:	0.00
Roll:	0.00
Pitch:	0.00
Yaw:	0.00

Figure 4.20: Window for the definition of the points of a trajectory

Finally, the "Tool Operation" mode window shown in Figure 4.21 allows the user to select a preference between joint space and cartesian space for the definition of the position in which the robot has to perform the operation. The specific action to be performed can be selected among the available ones (Open, Close and Wait). As a further notice, for all the cases in which the operator selects the "Wait" operation, the system automatically assigns the "Human Contact" behavior to the segment in order to make the robot act as desired.



Dialog

Space Type: [✓ Joint Space] Cartesian Space

Gripper Control: [✓ Open] Close Wait

Start

Joint 1	0.00000
Joint 2	0.00000
Joint 3	0.00000
Joint 4	0.00000
Joint 5	0.00000
Joint 6	0.00000

Dialog

Space Type: Joint Space [✓ Cartesian Space]

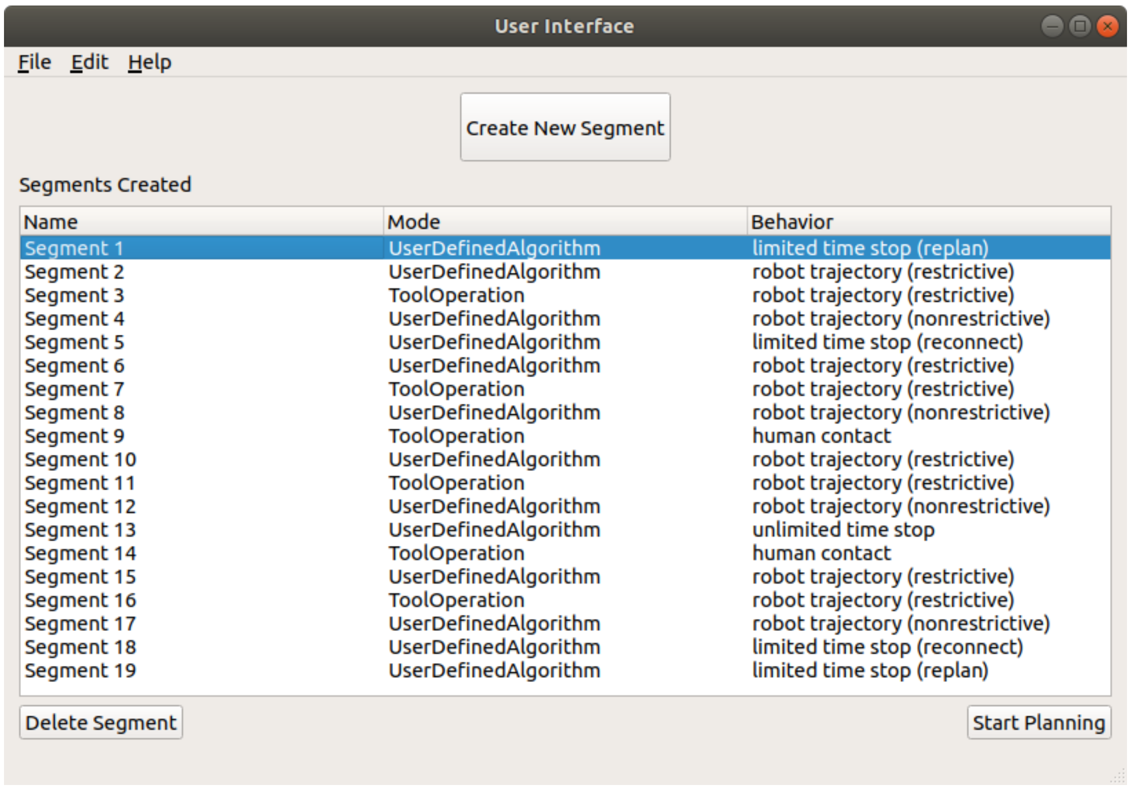
Gripper Control: [✓ Open] Close Wait

Start

X:	0.00
Y:	0.00
Z:	0.00
Roll:	0.00
Pitch:	0.00
Yaw:	0.00

Figure 4.21: Screenshot of the Tool Operation offline planning mode

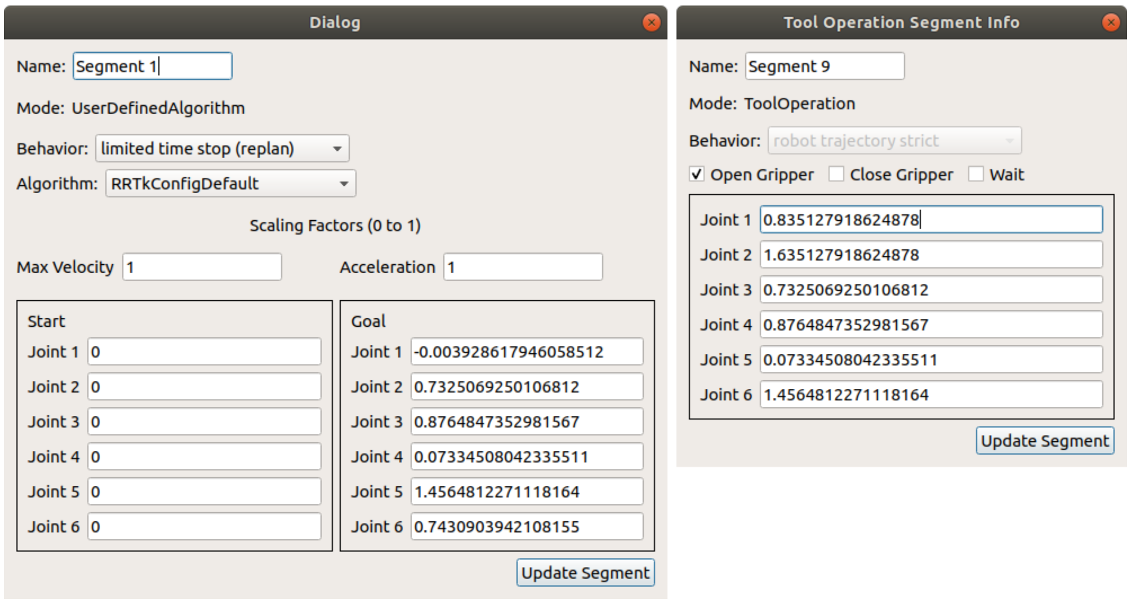
In order to provide the reader with a deeper understanding of the convenience of the GUI, Figure 4.22 shows a fully defined scenario (note that the scenario displayed on the GUI is the same used in Chapter 6 for the experimental validation of the system). As previously stated, a brief summary of the characteristics of each segment is displayed in a very intuitive manner in order to give the user a general overview of the programmed task. Once the scenario has been saved, it can be re-opened at any time and each segment can be modified by simply double-clicking on it. In Figure 4.23 two examples of the window that pops up during this modification process are reported: a summary of all the information assigned to the specific segment is provided and immediately accessible to the user for any change.



Segments Created

Name	Mode	Behavior
Segment 1	UserDefinedAlgorithm	limited time stop (replan)
Segment 2	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 3	ToolOperation	robot trajectory (restrictive)
Segment 4	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 5	UserDefinedAlgorithm	limited time stop (reconnect)
Segment 6	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 7	ToolOperation	robot trajectory (restrictive)
Segment 8	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 9	ToolOperation	human contact
Segment 10	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 11	ToolOperation	robot trajectory (restrictive)
Segment 12	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 13	UserDefinedAlgorithm	unlimited time stop
Segment 14	ToolOperation	human contact
Segment 15	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 16	ToolOperation	robot trajectory (restrictive)
Segment 17	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 18	UserDefinedAlgorithm	limited time stop (reconnect)
Segment 19	UserDefinedAlgorithm	limited time stop (replan)

Figure 4.22: Overview window of a scenario programmed with the GUI



Name: Segment 1

Mode: UserDefinedAlgorithm

Behavior: limited time stop (replan)

Algorithm: RRTkConfigDefault

Scaling Factors (0 to 1)

Max Velocity 1 Acceleration 1

	Start		Goal
Joint 1	0	Joint 1	-0.003928617946058512
Joint 2	0	Joint 2	0.7325069250106812
Joint 3	0	Joint 3	0.8764847352981567
Joint 4	0	Joint 4	0.07334508042335511
Joint 5	0	Joint 5	1.4564812271118164
Joint 6	0	Joint 6	0.7430903942108155

Name: Segment 9

Mode: ToolOperation

Behavior: robot trajectory strict

[✓ Open Gripper] Close Gripper Wait

Joint 1	0.835127918624878
Joint 2	1.635127918624878
Joint 3	0.7325069250106812
Joint 4	0.8764847352981567
Joint 5	0.07334508042335511
Joint 6	1.4564812271118164

Figure 4.23: Example of the windows available for the modification process

As introduced at the beginning of the chapter, the first duty of this code is a step of “input reading”. In fact, all the information provided by the user through the GUI is written in a TXT file in an intuitive form, in order to enhance the accessibility of the system as much as possible. However, this language is not efficient in terms of coding and, therefore, it is necessary to translate all the data in C++ programming language. In order to do that, the authors developed a method to quickly read the information provided by the user through the GUI and store it inside the “segment objects” that have been presented in the previous sections. Thanks to this step of translation, the algorithm can exploit all the data in an efficient way and, at the same time, the original program file can be saved creating a library of preprogrammed collaborative scenarios, always available for future use.

Chapter 5

Online module: High-level controllers and robot behaviors

5.1 Introduction

So far, this work has introduced the idea of programming an industrial application as a composition of segments. A detailed explanation of the methodology used to create, manage and execute segments can be found in Chapter 4. In the first part of this chapter, instead, the research will focus on the development of a series of high-level controllers, providing the robot with a variety of capabilities that can be used to react in real-time to different changes in the environment (e.g., obstacles, human encounters). Then, the concept of "robot behavior" is introduced. As proposed in [3], a behavior consists in the interplay of several concurrently running elemental controllers, properly sequenced. By joining the segmentation process and the concept of robot behaviors, a user that divides the robotic application into segments, can assign a particular behavior to each one of them independently. This approach is particularly useful for all the cases in which the segments composing an industrial application are characterized by different needs and therefore require different online behaviors.

This can be shown by referring to the same example used in Chapter 4. When considering a welding scenario the robot has to behave differently while approaching the welding area and when performing the welding on the workpiece. According to previous explanation, this scenario can be described as the composition of two segments: the first one represents the movement of the robot from its original position to the welding area, while the second one is the path that the end effector has to follow while performing the welding. Thanks to the segmentation process, the user is not only able to plan the two trajectories using two different approaches, but also to independently assign to each one of them an appropriate online behavior. Considering the first segment of the example, the user could decide that the path followed by the robot to reach its goal is not relevant, and therefore may want to allow the system to be able to modify what has been computed offline if necessary. Unexpected changes to the environment therefore trigger the system to adapt the robot movement in order to reach the goal ensuring safety and good performances. On the other hand, the trajectory defined for the second segment is of absolute importance when trying to obtain good welding results. Speed and positions that have been decided offline cannot be modified without risking a damage to the workpiece, meaning that the system should try to avoid any of those changes.

Before presenting the implementation of this system, another important aspect must be taken into account. The target application field of this work is characterized by a rapidly changing environment that requires high levels of flexibility and, therefore, robot behaviors able to adapt to every possible requirement. As already stated in the previous chapter, a modular system could represent a great solution. In fact, it would allow developers to keep up with the rapid changes of the industry by improving the online capabilities of the robot or by adding new features, without discarding the whole structure of the system. This concept of modularity also enables reusability of code, incremental design of functionality and efficient testing as highlighted in [28] and [3]. With reference to Figure 5.1, showing the control architecture of the robotic system, this chapter addresses the Online/Real-time Module.

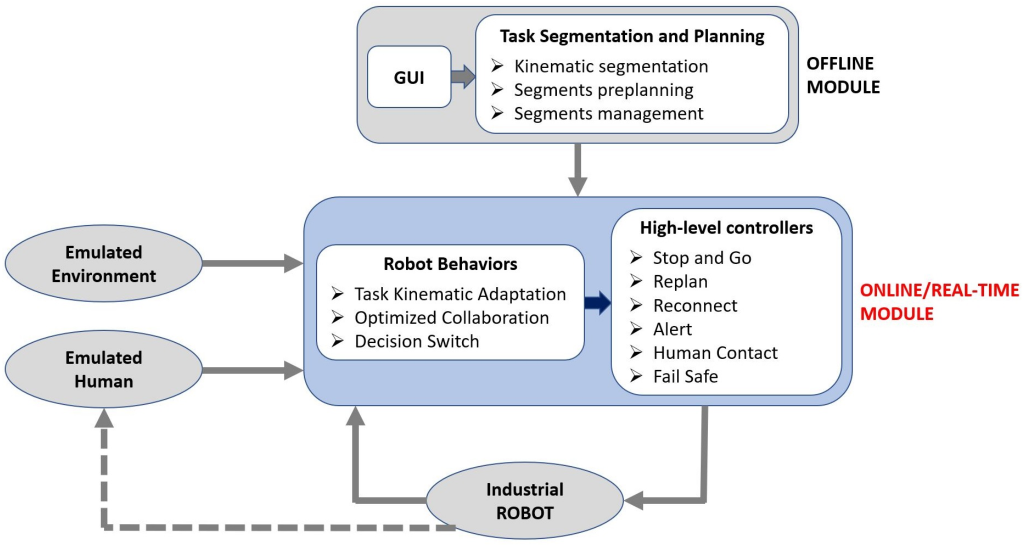


Figure 5.1: Focus on the module of the control scheme addressed by Chapter 5

In particular, this chapter presents in detail two codes:

1. **High-level controllers**: containing all the controllers that have been developed;
2. **Robot behaviors**: containing all the implemented reactive behaviors.

As already introduced in Figure 4.14, each one of these codes represents a node in the ROS environment and can, therefore, communicate with all the other available nodes. They are connected both to the segmentation code, developed in Chapter 4, in order to gather information about the plans under execution, and between each other to exchange information about the controllers' activation.

5.2 High-level controllers

In Chapter 2, the existing approaches to human-robot collaboration have been presented. Starting from the need of flexibility and adaptability of today's industry, though, selecting a single mode of interaction and developing an entire system around it would be limiting. Therefore, the idea of this project is to develop a series of approaches to human-robot interaction so that the most suitable one can be activated for every specific situation. In this thesis, these approaches are called "High-level Controllers" and represent the different modalities that are available for the robot to interact with both the environment and the human operator. Moreover, having a series of different controllers allows the robotic system to be able to adapt to a great variety of situations typical of an industrial environment, making the system flexible while always ensuring safety. Here is the list of the controllers that have been developed:

- Stop and Go;
- Replan;
- Reconnect;
- Alert;
- Human Contact;
- Fail Safe.

In the following pages, a detailed analysis of each one of them is reported, in order to understand their functionality and the logic behind their implementation.

5.2.1 Stop and Go

This first high-level controller developed by the authors constantly monitors the distance between the robot and any obstacle in the environment during the execution of a trajectory. If this distance becomes smaller than a certain predefined threshold, the robot comes to a complete stop in order to avoid a possible collision. When this happens, the whole system remains active so that the value of distance can be constantly updated and no information about the commanded trajectory is lost. Then, as soon as the obstacle moves away from the manipulator and the distance becomes once again bigger than the given threshold, the robot restarts its motion from the position in which it was stopped, following the remaining part of the original trajectory. The threshold is decided at code level and represents the minimum distance between the moving robot and the environment/operator that can be considered acceptable in terms of safety. Figure 5.2 provides a representation of the functionalities of this controller.

As long as the Stop and Go controller is active, the algorithm cycles inside a C++ *while* loop where the value of distance from the closest obstacle is constantly updated. Every obstacle present in the workspace is also represented in the planning scene so that the robotic system can take it into account. This allows MoveIt! to calculate the minimum distance between the robot and the obstacles using the FCL library, presented in greater detail in Section 3.2.4. A safety check is performed every time the distance is updated. If the distance is bigger than the predefined threshold the robot is allowed continue on its path. However, if the value of distance is smaller, the controller considers the situation as hazardous and therefore calls a MoveIt! function that immediately stops the robot, overriding the execution of the remaining part of the trajectory.

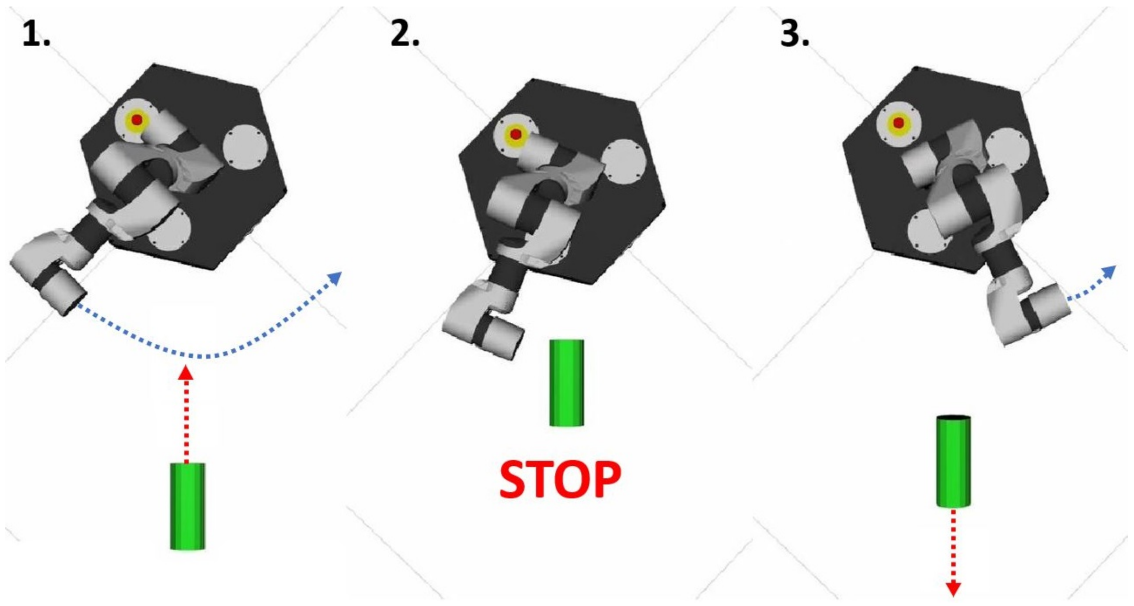


Figure 5.2: Representation of the functionality of the Stop and Go controller

As soon as the obstacle has moved away far enough and the value distance has become greater than the threshold, the algorithm is triggered to perform an analysis on the original trajectory. First, it cycles through all the waypoints of the trajectory, looking for the minimum distance between the state in which the robot was stopped and the intermediate states corresponding to each waypoint. This information allows the authors to detect the waypoint of the preempted path closest to the stop position. This point is used to extract, from the original trajectory, the portion that still has to be executed and to copy it inside a new path object. Moreover, in order to avoid any problem related to the tolerance on the start position, the code adds, at the beginning of the new plan, a point corresponding to the exact position in which the robot has been stopped. Before commanding the execution of this new trajectory, the algorithm has to modify the time stamps of all the points. The first waypoint is assigned a *time_from_start* equal to zero, the following ones are labeled with a time value calculated on the basis of the distance between them and the imposed limits of speed and acceleration, while the time stamp of the last waypoint is modified in order to make the robot slow down and smoothly stop at goal position. The structure of this controller is reported in Figure 5.3.

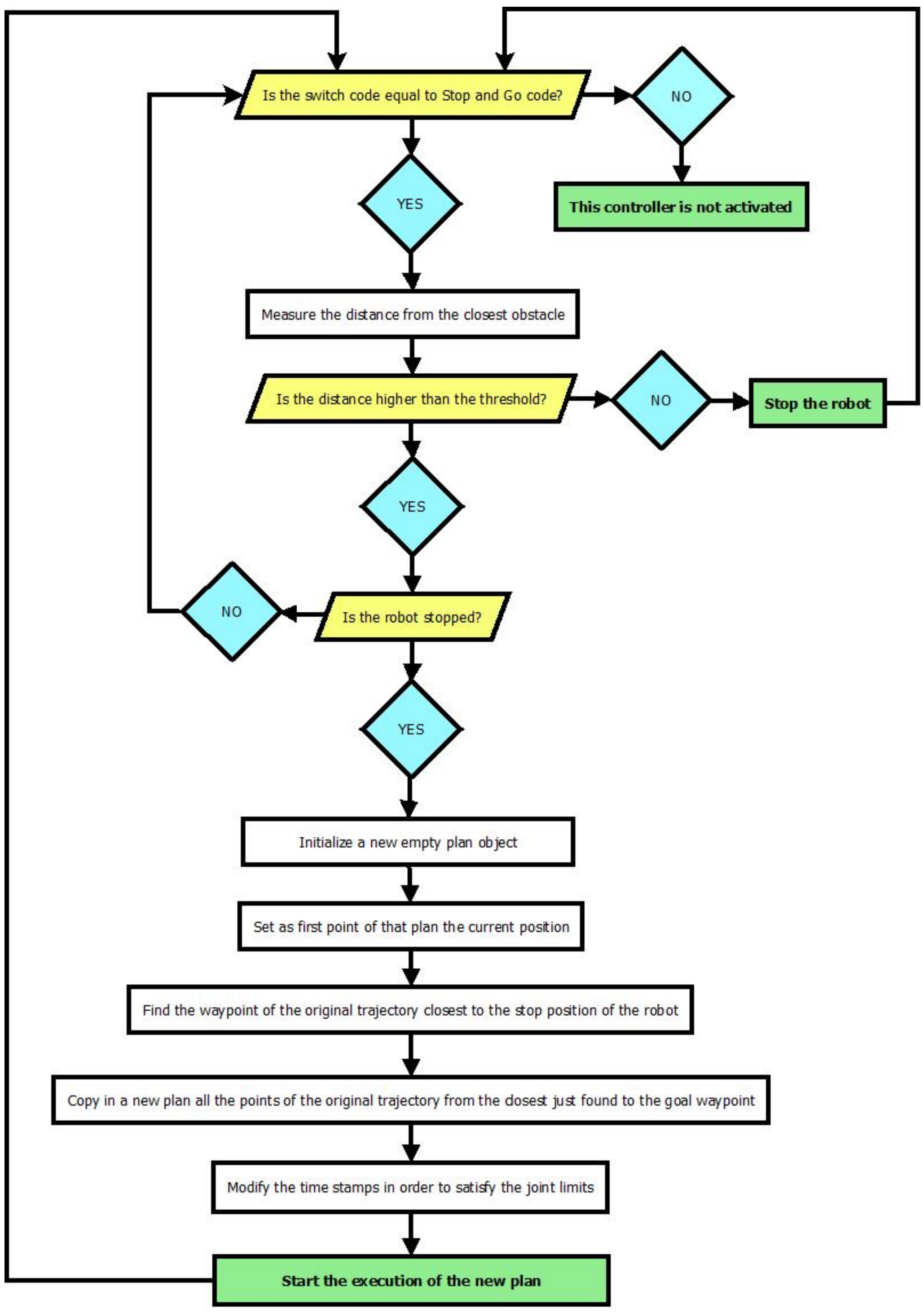

Is the switch code equal to Stop and Go code?
NO
This controller is not activated
YES
Measure the distance from the closest obstacle
Is the distance higher than the threshold?
NO
Stop the robot
YES
NO
Is the robot stopped?
YES
Initialize a new empty plan object
Set as first point of that plan the current position
Find the waypoint of the original trajectory closest to the stop position of the robot
Copy in a new plan all the points of the original trajectory from the closest just found to the goal waypoint
Modify the time stamps in order to satisfy the joint limits
Start the execution of the new plan

Figure 5.3: Flowchart of the Stop and Go controller

5.2.2 Replan

The high-level controller presented here introduces a different approach to obstacle avoidance with respect to the one used for “Stop and Go”. The “Replan” controller differs from the “Stop and Go” controller in the fact that the first one allows the manipulator to modify its path in order to avoid the obstacle and reach its goal. In particular, when the value of distance becomes smaller than a predefined threshold, the robot gets stopped using the built-in MoveIt! function and the algorithm immediately starts looking for a new path that brings the robot from the position in which it was stopped to the same goal of the original trajectory, without colliding with the obstacle. In order to ensure the safety of this operation, a key feature of this controller is the ability to replan around the object with a certain value of clearance. The built-in planner used by MoveIt! does not allow the user to select a desired minimum distance from the obstacle. Instead, it considers as feasible every configuration of the robot that is not in a state of collision, even if it brings the manipulator extremely close to the object. To overcome this limitation, before calling the planner, the code developed by the authors increases the size of the virtual obstacle in a process referred to as “virtual inflation” Thanks to this process, zero clearance from the “inflated” object actually represents a value of clearance for the real one equal to the difference between the two sizes. As soon as the planner has generated the new path around the virtually enlarged obstacle, the algorithm reduces its size to the original one and, only after that, the execution can be commanded. As a result, by modifying the percentage of size increase at code level, the algorithm is able to guarantee a value of clearance that satisfies the required safety constraints. Once again, in Figure 5.4 a representation of the functionalities of this high-level controller is provided.

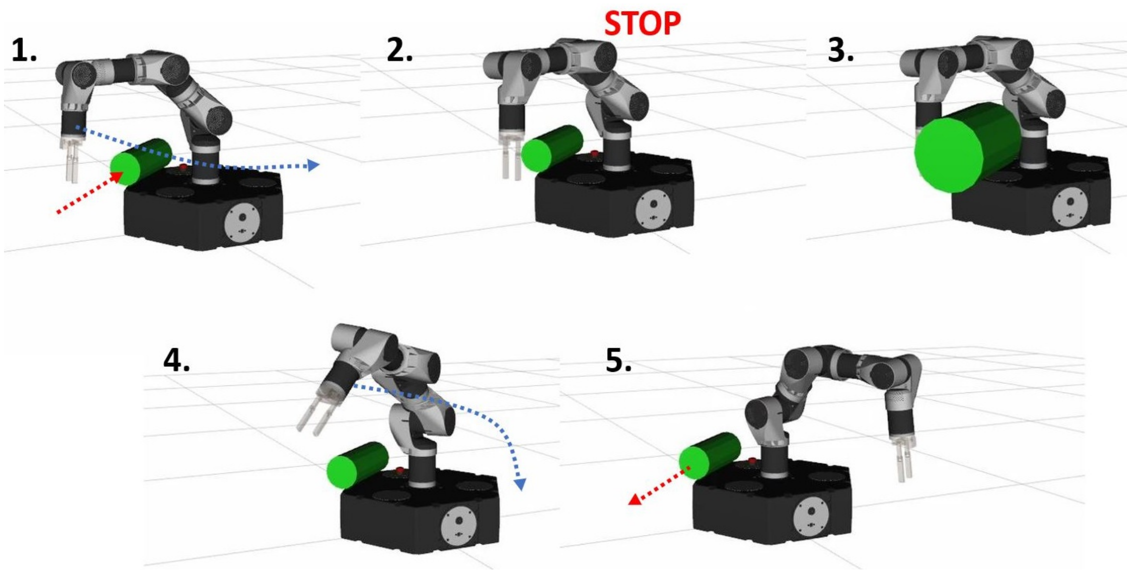


Figure 5.4: Representation of the functionality of the Replan controller

In terms of code, the activation of the controller means entering a *while* loop in which the system continuously updates the value of distance between the robot and the closest obstacle. If this value is bigger than a predefined threshold, no action is required and the robot can continue on its path undisturbed. If, instead, the distance becomes smaller than the minimum allowed, the algorithm immediately stops the execution of the trajectory. At this point, in order to increase the dimensions of the virtual obstacle, a message containing the information of the new size is published to the Planning Scene Monitor (Section 3.2) in order to modify the original value. Then, the planner computes a path that avoids the enlarged obstacle using as start position the current position, and as goal the original goal extracted from the last waypoint of the old trajectory. Finally, a second message is published in order to resize the object in the planning scene to its original real value, and the execution of the newly computed path is commanded. A flowchart of what has just been explained is reported in Figure 5.5.

Regarding threshold value, it should be noted that, if the execution of the replanned trajectory was commanded without modifying it, the algorithm would immediately stop the robot once again since the value of distance would still be too small. For this reason, before starting the execution, the threshold has to be reduced in order to allow the motion of the robot and then, as soon as the distance grows bigger than the original limit, its value can be restored. The code in charge of the constant communication of the threshold is “Robot Behaviors”, addressed in the next section of this chapter.

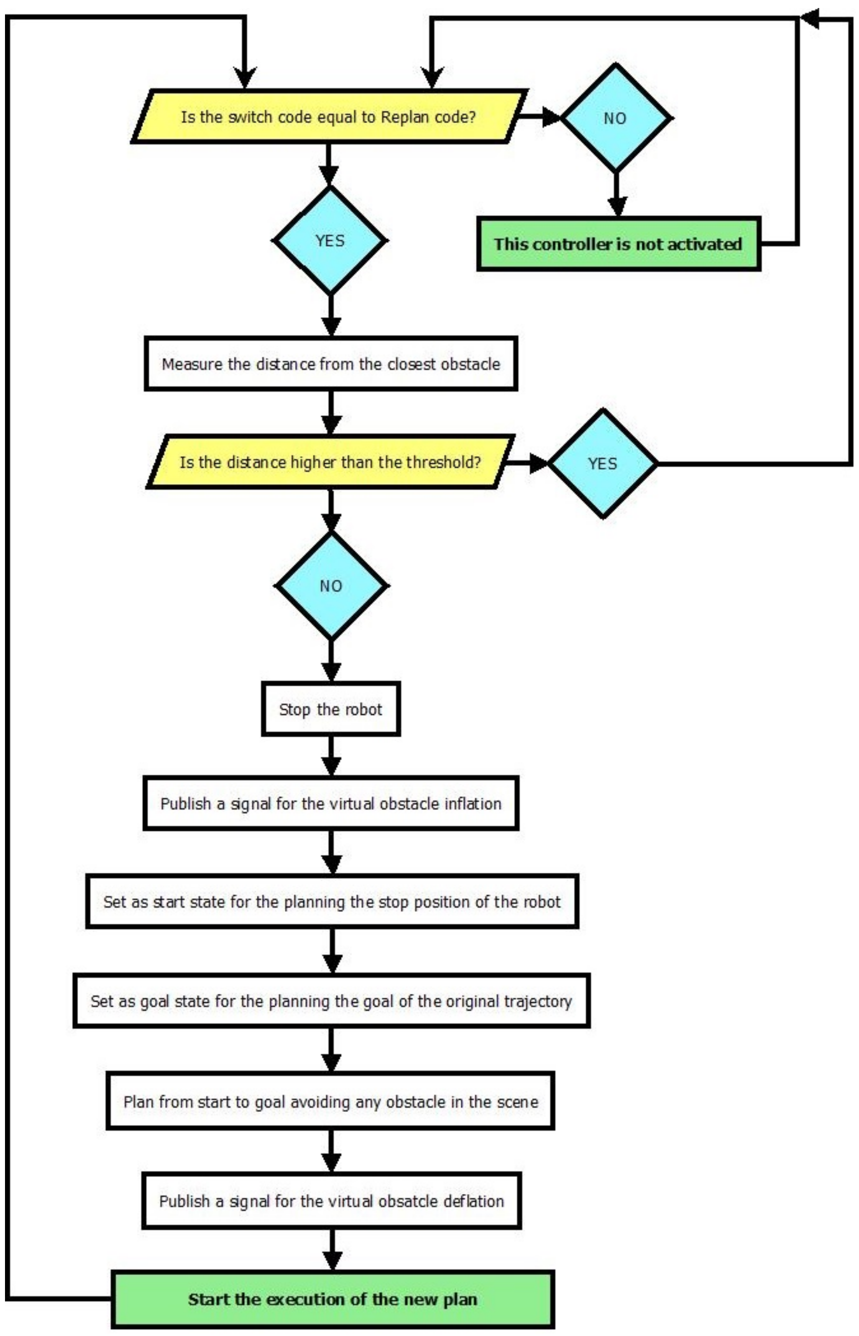

Is the switch code equal to Replan code?
NO
This controller is not activated
YES
Measure the distance from the closest obstacle
Is the distance higher than the threshold?
YES
NO
Stop the robot
Publish a signal for the virtual obstacle inflation
Set as start state for the planning the stop position of the robot
Set as goal state for the planning the goal of the original trajectory
Plan from start to goal avoiding any obstacle in the scene
Publish a signal for the virtual obsatcle deflation
Start the execution of the new plan

Figure 5.5: Flowchart of the Replan controller

5.2.3 Reconnect

The “Reconnect” controller is very similar to the previous one: if an obstacle is detected within a predefined threshold, the robot is stopped and the virtual object is enlarged in order to ensure a certain clearance. Then, the algorithm tries to replan a new trajectory around the object. The only difference between “Replan” and “Reconnect” is that, for the latter, the planner looks for a new path that goes from the position in which the robot was stopped to the first valid point of the original trajectory after the obstacle, instead of replanning all the way to the original goal. In the authors’ implementation, the controller considers “valid” the first waypoint of the portion of the original path that has not been executed not in collision with the obstacle and at a distance from the obstacle specified at code level. The only constraint in performing this operation, is that the chosen value of distance must be bigger than the threshold in order to make sure that the “first valid point” is detected after the obstacle. Finally, once the robot has reconnected with the original trajectory, it follows it until the goal of the segment is reached. The main steps representing the functionalities of this controller are depicted below in Figure 5.6,

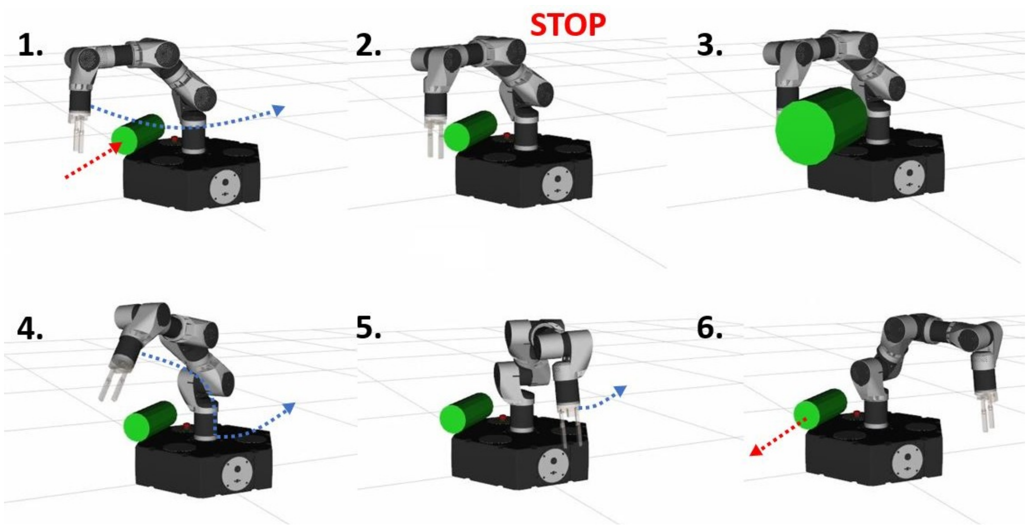


Figure 5.6: Representation of the functionality of the Reconnect controller

The code developed to implement this new controller is very similar to the previous one but requires further calculations in order to generate the reconnection capability. For every cycle, the value of distance is updated and compared to the threshold. Once again, if the distance is too small, the robot is immediately stopped to prevent any collision. As done for the “Stop and Go” controller, the code looks for the waypoint closest to the position where the robot was stopped by cycling through the whole original trajectory and saves its corresponding index. After that, the virtual obstacle inflation is triggered, and the algorithm starts looking for the “first valid point” of the preempted path by cycling through its intermediate positions from the saved index to the end of the segment. Now, the planner is leveraged to compute a path that avoids the enlarged obstacle and connects the current position of the robot with the “first valid point” just defined. This path is then copied into a new path object where it is merged with the portion of the original trajectory that goes from the “first valid point” to the end of the segment. As already done for the previous controllers, the time stamp of all the waypoints is modified in order to respect speed and acceleration limits of the robot and, finally, the execution can be commanded. The flowchart representing this whole procedure is shown in Figure 5.7.

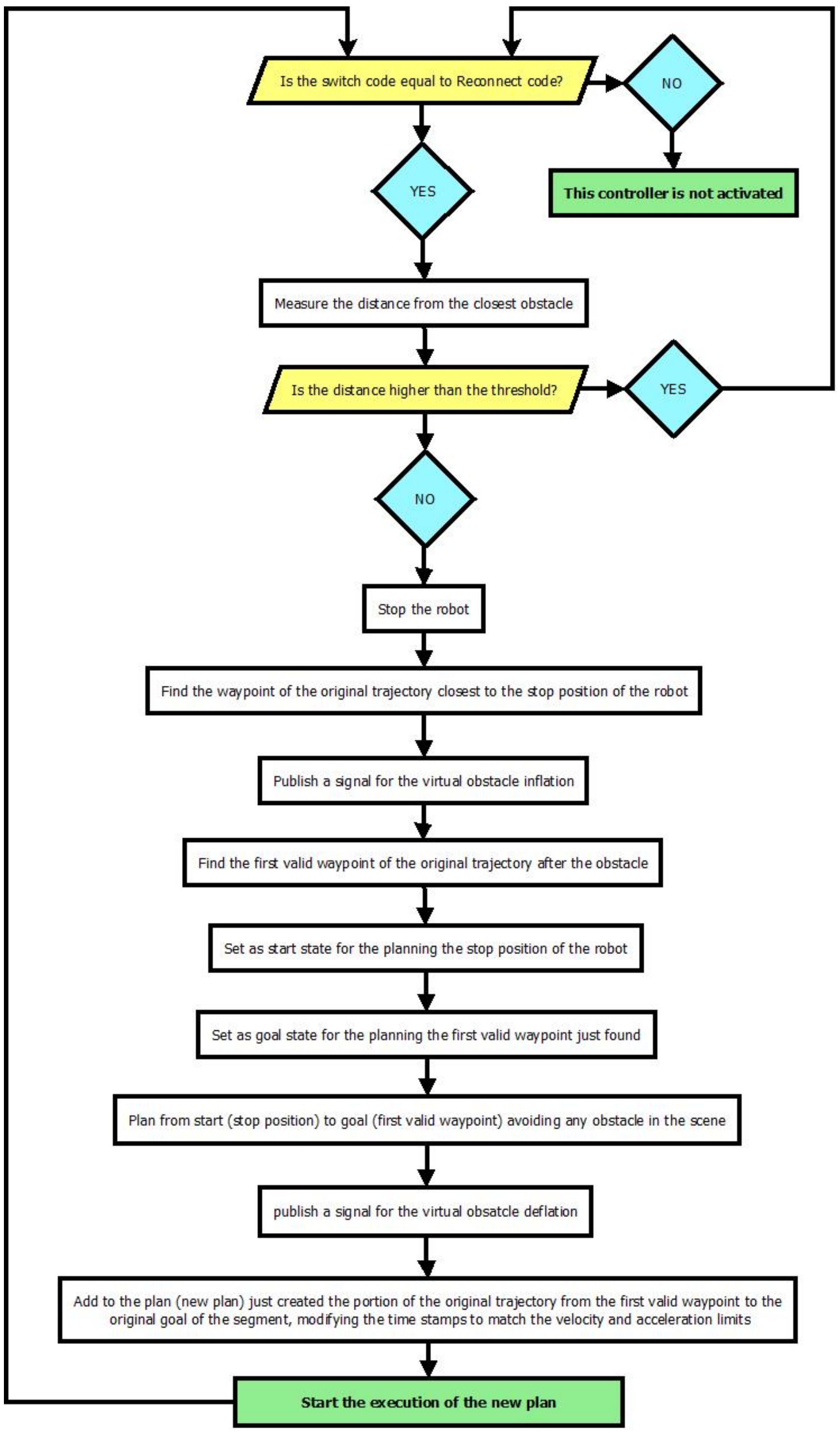

Is the switch code equal to Reconnect code?
NO
YES
This controller is not activated
Measure the distance from the closest obstacle
Is the distance higher than the threshold?
YES
NO
Stop the robot
Find the waypoint of the original trajectory closest to the stop position of the robot
Publish a signal for the virtual obstacle inflation
Find the first valid waypoint of the original trajectory after the obstacle
Set as start state for the planning the stop position of the robot
Set as goal state for the planning the first valid waypoint just found
Plan from start (stop position) to goal (first valid waypoint) avoiding any obstacle in the scene
publish a signal for the virtual obsatcle deflation
Add to the plan (new plan) just created the portion of the original trajectory from the first valid waypoint to the original goal of the segment, modifying the time stamps to match the velocity and acceleration limits
Start the execution of the new plan

Figure 5.7: Flowchart of the Reconnect controller

5.2.4 Alert

The “Alert” controller differs from the previous ones mainly in the robot reaction triggered when the distance from the closest obstacle becomes smaller than a predefined threshold. In the previous cases, the system was looking for the position of external obstacles in order to stop the robot if a collision was imminent. In this case, instead, the system still analyzes the distance from the obstacles, but the aim of the controller is to communicate to a human operator that the trajectory of the manipulator must not be disturbed. In practice, when this controller is activated, a *while* loop is accessed in order to constantly update the value of distance. Another threshold is used to define the minimum acceptable distance below which communication with the operator has to be activated. Therefore, if the user, or any other obstacle, gets too close to the manipulator while it is operating under this controller, a certain signal is emitted in order to state the detected possible disturbance and ask for a clear path. In the long run, the "Intelligent HRC for Smart Factory" project envisions the development of a new language between the human and the manipulator, able to allow a natural and intuitive way of communication between the two entities. In these early stages of the project, though, the authors' implementation simply relies on an alert sound that gets activated every time that an obstacle comes closer than a certain threshold to the robot. A flowchart of the logic of this controller is reported in Figure 5.8.

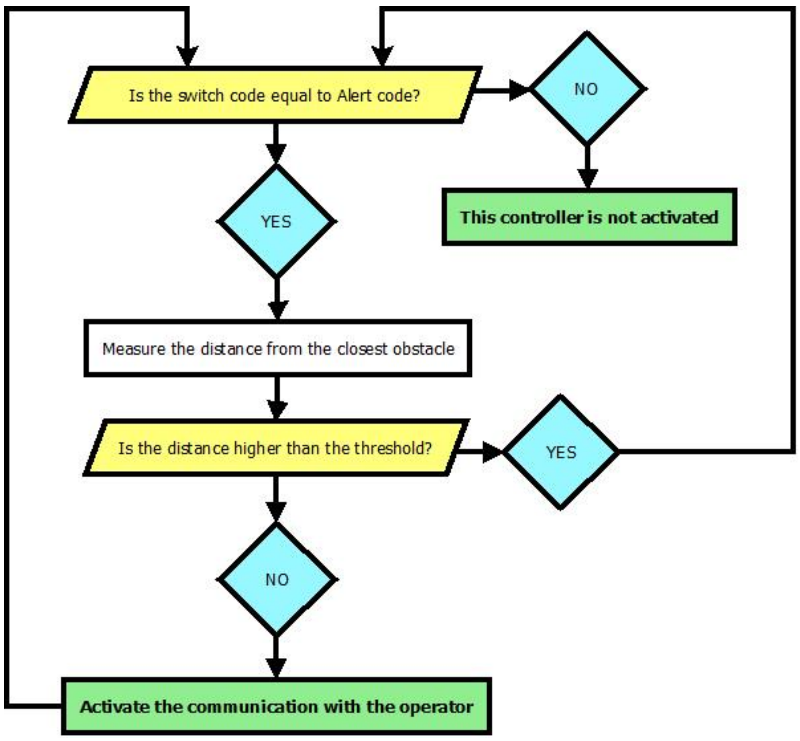

Is the switch code equal to Alert code?
NO
This controller is not activated
YES
Measure the distance from the closest obstacle
Is the distance higher than the threshold?
YES
NO
Activate the communication with the operator

Figure 5.8: Flowchart of the Alert controller

5.2.5 Human Contact

The “Human Contact” controller has been developed to allow direct contact between the robot and the external environment. For example, this functionality could be exploited every time that an operator has to perform a certain operation on the robot, such as inspection of the end-effector or of the carried workpiece, as represented in Figure 5.9. Since this controller basically allows collisions to happen, it can be activated only when the robot is completely still and therefore there is no risk of harming the operator. The “Human Contact” controller remains active for as long as the user needs to complete the operations on the robot. When the operator is done, the system waits for a confirmation signal, needed to trigger the execution of the next segment. In the authors’ case, this signal is given by the operator simply by pressing the ENTER key on the keyboard.

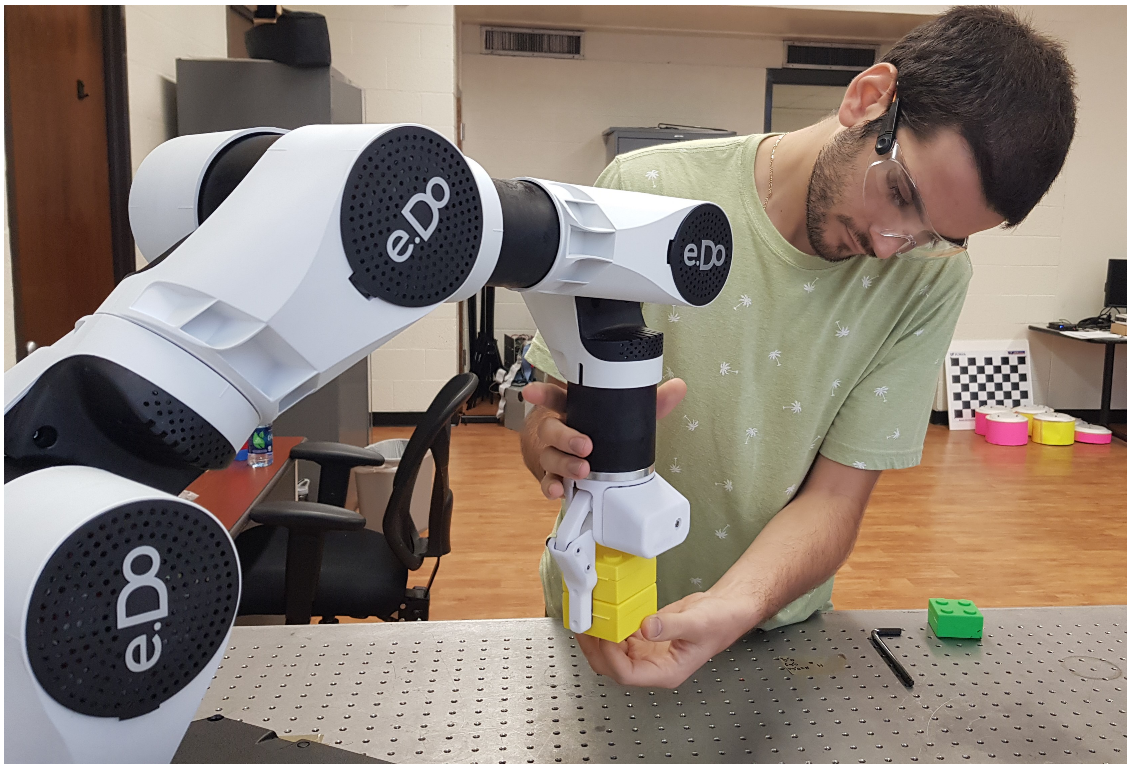


Figure 5.9: Example of an inspection operation performed under the Human Contact controller

In terms of code, the controller has been realized with the following simple logic: the system waits inside a while loop until the confirmation signal is given by the user. Only after that the signal is received, the controller sends a feedback to the segmentation code matching all the requirements needed for commanding the execution of the next segment, as explained in Chapter 4. Note that, for this type of controller, there is no need to monitor the distance from the closest collision. The flowchart of this controller is reported in Figure 5.10;

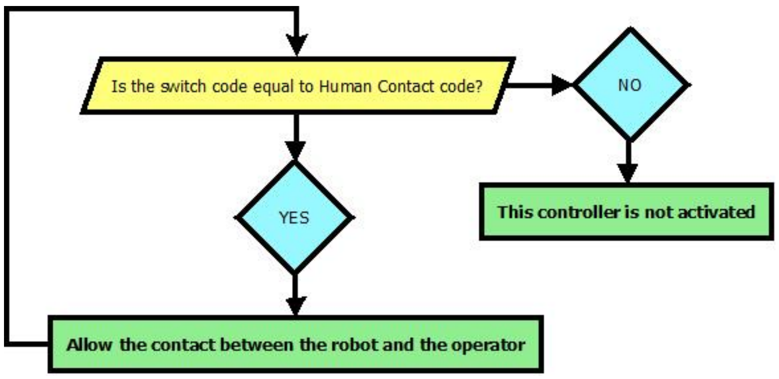

Is the switch code equal to Human Contact code?
NO
YES
This controller is not activated
Allow the contact between the robot and the operator

Figure 5.10: Flowchart of the Human Contact controller

5.2.6 Fail Safe

The last high-level controller, developed by the authors for this project, represents the base layer for the safety of the system. It equips the robot with a "Fail Safe" mode that can be called every time something unexpected happens. For instance, it is automatically activated if, for any reason, the distance between the manipulator and an external obstacle approaches zero, meaning that a collision is impending. In practice, whenever the "Fail Safe" gets activated, it overrides every other controller and brings the system to a complete shutdown. Obviously, this does not happen if the "Human Contact" controller is running, since the value of distance is not monitored and contact is allowed. A situation in which this mode has to be called represents an emergency. In this case, the only acceptable reaction is to interrupt whatever operation is being performed, so that maximum priority can be given to the safety of the operators. This means that if the "Fail Safe" is activated, since the whole system is shut down, a reboot will be needed before being able to restart it. Figure 5.11 reports a flowchart of the structure of the code used to implement the Fail Safe.

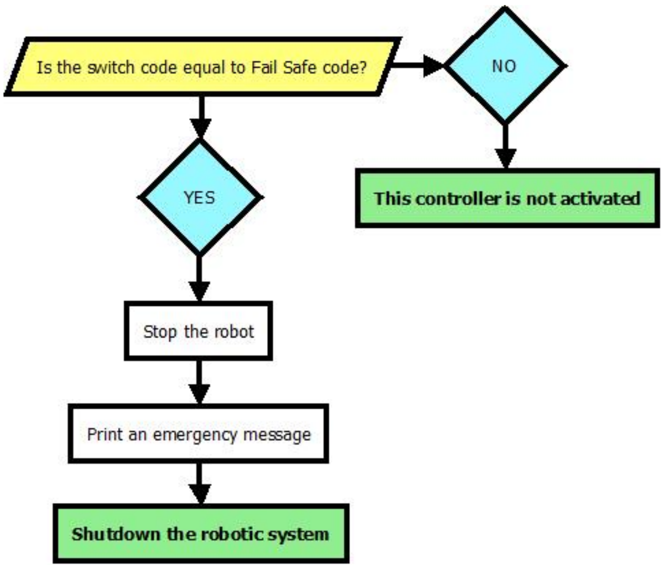
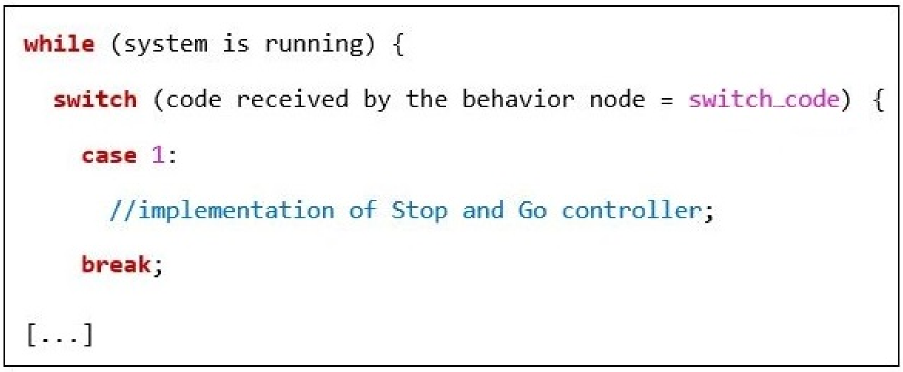

Is the switch code equal to Fail Safe code?
NO
YES
This controller is not activated
Stop the robot
Print an emergency message
Shutdown the robotic system

Figure 5.11: Flowchart of the Fail Safe controller

The high-level controllers have been developed starting from the recognition of the main needs that can show up in real industrial scenarios. Each one of them tries to solve the problem of human-robot interaction in a different way and for this reason, depending on the situation, there may be a particular one that is better suited than the others in terms of safety and performance. A more detailed analysis of when and why each controller could be preferred over the others will be presented in Section 5.3, dealing with the robot behaviors.

5.2.7 Code modularity

As already introduced for the offline planning techniques, a modular structure of the code would ease the process of adapting to the rapid changes of the modern industrial environment, enhance the reusability of the code and allow developers to add new features without the need of rearranging the whole system. Therefore, the solution proposed by the authors consists of a *switch* structure in which each case contains a high-level controller. A *switch* module is a simple C++ structure that can be exploited to activate different sections of the code on the basis of the value that it receives as input. Each section is labelled with a specific number and, every time that the switch function receives a new input, it activates the part of the algorithm that corresponds to that value. The code responsible for sending the correct input to this module is "Robot Behaviors", able to communicate with "High-level Controllers" exploiting the functionalities of the ROS environment. The two codes are both nodes composing the robotic network and therefore have the possibility of exchanging messages between each other. As a result of this approach, a new controller could be integrated in the system by simply adding it as a new case for the switch module. Moreover, an already existing controller can be modified in a simplified way, since it is confined in a specific part of the code, and an obsolete controller can be discarded by deleting its corresponding case. The pseudo-code representing this structure is shown in Figure 5.12.



```
while (system is running) {

  switch (code received by the behavior node = switch_code) {

    case 1:

      //implementation of Stop and Go controller;

    break;

[...]
```

Figure 5.12: Pseudo-code of the modular structure containing the developed High-level Controllers

5.3 Robot Behaviors

As explained in the first part of this chapter, the high-level controllers represent a series of possible approaches to human-robot interaction. The robotic system is now equipped with several online capabilities, but still lacks a smart decision-making module able to activate in real-time the best suited controller as a reaction to the data collected from the external environment. In order to guarantee the maximum flexibility, this module has been conceived as a collection of decision-making logics, called “robot behaviors”, each one capable of selecting in real-time the best controller to be activated on the basis of its own choosing algorithm. In particular, every behavior is associated to a specific group of high-level controllers and has the role of calculating a “cost of activation” for each one of them, by means of a cost function. This cost is used to quantify the impact that the activation of a specific controller would have on the safety and the productivity of the collaborative operation. On top of that, if any emergency situation is detected, the Fail Safe controller is immediately activated in order to guarantee operational safety.

The structure of each cost function is the same for all the implemented behaviors. The cost of activation of a controller, belonging to a specific behavior, is calculated as the sum of the following components:

$$Cost\ of\ Activation = Base\ cost + Distance\ cost + Delay\ cost$$

- **Base cost**: component of the cost that does not depend on any external parameter, simply equal to a constant value (C_{base});

$$Base\ cost = C_{base}$$

- **Distance cost**: component of the cost inversely proportional to the distance between the manipulator and the closest obstacle, calculated as the ratio between a constant coefficient (C_{dist}) and the value of distance measured by the collision detection library;

$$Distance\ cost = \frac{C_{dist}}{Distance}$$

- **Delay cost**: component of the cost proportional to the delay induced by any stop of the robot motion due to the presence of an obstacle, calculated as the product between a constant coefficient (C_{delay}) and the amount of delay measured in terms of number of code cycles spent while the robot is stopped;

$$Delay\ cost = C_{delay} \cdot Delay$$

The base cost is used to make sure that the desired controller is activated when the other two components are irrelevant. For instance, when no obstacle is detected close to the robot, the value of distance is big enough to make its corresponding cost very small and the delay cost is equal to zero since the robot is proceeding undisturbed on its path. The cost of activation is, therefore, almost equal to the base cost which becomes the main determinant for the choice of the controller to be activated. Considering the second component of the cost function, as the value of distance between the robot and external obstacles becomes smaller, its associated cost rises. This means that the switch logic can make a decision taking into account the safety level of the operation. Finally, the delay cost starts rising as soon as the robot motion is forced to stop in order to avoid a collision and, therefore, represents an indicator of the performance level for the task at hand.

$$Cost = C_{base} + \frac{C_{dist}}{Distance} + C_{delay} \cdot Delay$$

As shown in the final formula above, the total cost of activation for a controller is calculated on the basis of three coefficients. The value assigned to these coefficients are a specific characteristic of each decision algorithm. They have been defined experimentally in order to guarantee a specific timing and logic for the switch between the controllers triggered by each behavior. As depicted in the scheme in Figure 5.13, during the entire execution of a segment, the corresponding behavior chosen by the user is activated. This behavior constantly calculates the cost of activation for each controller belonging to its group and activates the least expensive one. This means that any change in the external environment influences the cost values and therefore triggers a switch between the controllers as soon as a different one becomes more convenient. If, for any reason, the safety of the operator is threatened, the system automatically switches to the "Fail Safe" mode, which immediately overrides the running controller in order to avoid any damage or injury cause by a collision with the robot.

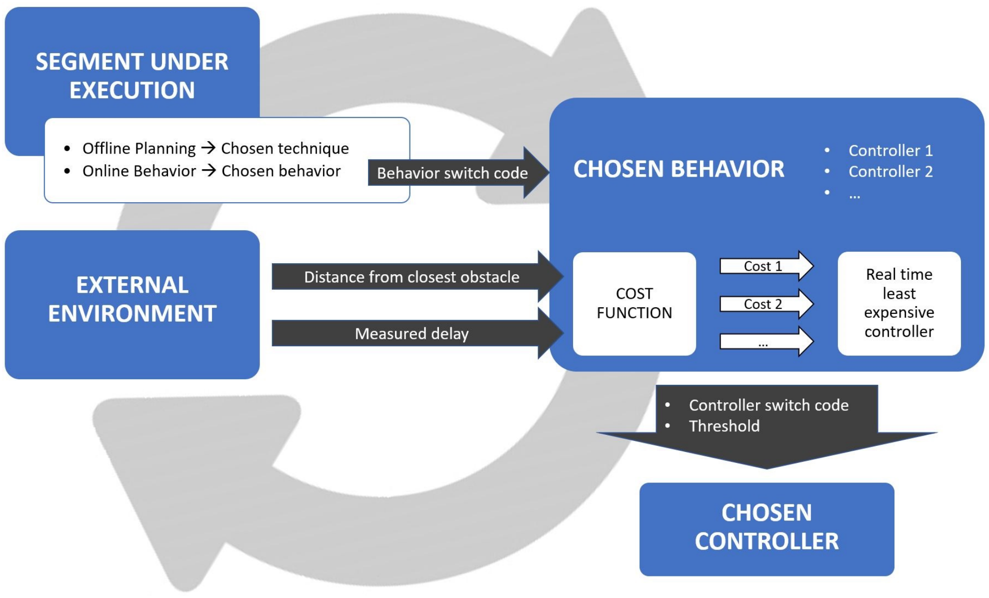


Figure 5.13: Schematic representation of the logic behind the Online/Real-time module

Once again this algorithm exploits the communication network provided by ROS. Every time that the “Segmentation” node commands the execution of a new segment, it publishes a message to “Robot Behavior” containing the code corresponding to the desired decision logic. This last node keeps the requested behavior running and, in turn, publishes a message to the “High-level Controllers” node with the information of the least expensive controller that has to be activated. Furthermore, “Robot Behavior” is also in charge of the communication of the values of the thresholds to be used by the involved controllers. In fact, every controller can be used by more than one behavior and for each case different values of the thresholds could be required. This means that they cannot be an intrinsic characteristic of the controller, but they must be updated every time according to the logic of the specific behavior.

On the basis of a general analysis of real industrial collaborative applications, the authors developed a series of robot behaviors aimed to solve the most common needs in terms of online smart capabilities:

- Limited Time Stop (Replan);
- Limited Time Stop (Reconnect);
- Unlimited Time Stop;
- Robot Trajectory (Non-restrictive);
- Robot Trajectory (Restrictive);
- Allowed Contact.

In the following pages a detailed analysis of the logic used to implement them and of their main benefits is presented.

5.3.1 Limited Time Stop (Replan)

The idea behind this first behavior is to address the problem of a convenient robot reaction in front of an obstacle obstructing the manipulator’s path. In some cases it would be convenient to stop the motion of the robot and simply wait for the obstacle to move away before restarting the original trajectory. On the other hand, for some situations it would be preferable to replan and avoid the detected obstacle. Given that, for the current state of the project, the duration of the obstruction cannot be predicted, the solution proposed by the authors consists in stopping the robot in front of the obstacle and taking a decision on the basis of the amount of delay that this stop causes. In particular, the robot is allowed to wait in its stationary state only for a time equal to a value specified by the user during the definition of the segments. If the obstacle moves away before the time limit, the execution of the original path can be restarted, otherwise a new plan is computed in order to avoid the obstacle and reach the goal of the segment. Obviously, this is possible only if the goal configuration is not obstructed by the obstacle itself. If this happens, the only feasible option is to let the robot wait in its position until the goal of the segment is cleared. If, during the execution of the segment, any emergency situation is detected, the “Fail Safe” controller immediately overrides any other command and the system is shut down. This behavior is particularly suited for all the cases in which the focus is not on the path followed by the manipulator, but mostly on its performance in reaching the goal. The two controllers selected for this behavior are “Stop and Go” and “Replan”. The coefficients of the function used to estimate their cost of activation are reported in Table 5.1.

High Level Controller	C_{base}	C_{dist}	C_{delay}
Stop and Go	0	0	1
Replan	$cycle\_frequency \cdot time\_limit$	0	0

Table 5.1: Table of Coefficients for the Limited Time Sop (Replan) behavior

As reported in Table 5.1, C_{delay} has been fixed to 1 for “Stop and Go” and to 0 for “Replan”, meaning that a delay would cause a cost increase only for the first controller. The value of C_{dist} is the same for both controllers because, for this behavior, the decision does not depend on the distance, but only on the amount of wasted time. In this case, the coefficients has been set to 0 in order to simplify the calculations. Regarding C_{base}, their values are calculated as a function of the maximum wait time defined by the user during the creation of the segments. In order to provide an example, the authors suppose a time limit equal to 4 seconds. Knowing that the cycle frequency is 10 Hz, the effect of the coefficients set by the authors is shown in Table 5.2.

Controller	**As soon as the robot stops**	**After 2s of delay**	**After 4.1s of delay**
$Cost_{StopAndGo}$	0	20	41
$Cost_{Replan}$	40	40	40
Active Controller	**Stop and Go**	**Stop and Go**	**Replan**

Table 5.2: Table with an example of cost values during the execution of a segment characterized by the Limited Time Stop (Replan) behavior

As shown, as soon as the robot stops due to the presence of an obstacle, the cost of the “Stop and Go” controller starts rising due to the contribution of the delay cost. After 2 seconds of delay, “Stop and Go” remains the most convenient controller, since the limit set by the user has not been reached. When the delay exceeds the limit, the cost of “Replan” becomes the lowest and therefore the switch between the controllers is triggered. When “Replan” gets activated, it plans a new path that avoids the obstacle and then commands its execution. The robot starts moving again and the value of delay is reset to 0, causing the “Stop and Go” controller to once again become the least expensive one, meaning that the new plan will be executed under its control.

As represented in the flowchart in Figure 5.14, this behavior has been implemented in the form of a *while* loop that remains active as long as the switch code, published by the “Segmentation” node, remains equal to the one corresponding to “Limited Time Stop

(Replan)”. Inside the loop, the values of the thresholds and of the coefficients of the cost function are defined and the measure of the distance from the closest obstacle is constantly updated. For every cycle during which the robot is stopped due to the presence of an obstacle not obstructing the goal of the segment, the value of delay is increased while, if the robot is moving, it is reset back to 0. Then, the cost of activation for both the controllers is calculated. If an emergency situation is detected, then Fails Safe is immediately activated. Otherwise, the switch code of the least expensive controller is published to High-level Controllers together with the corresponding value of threshold.

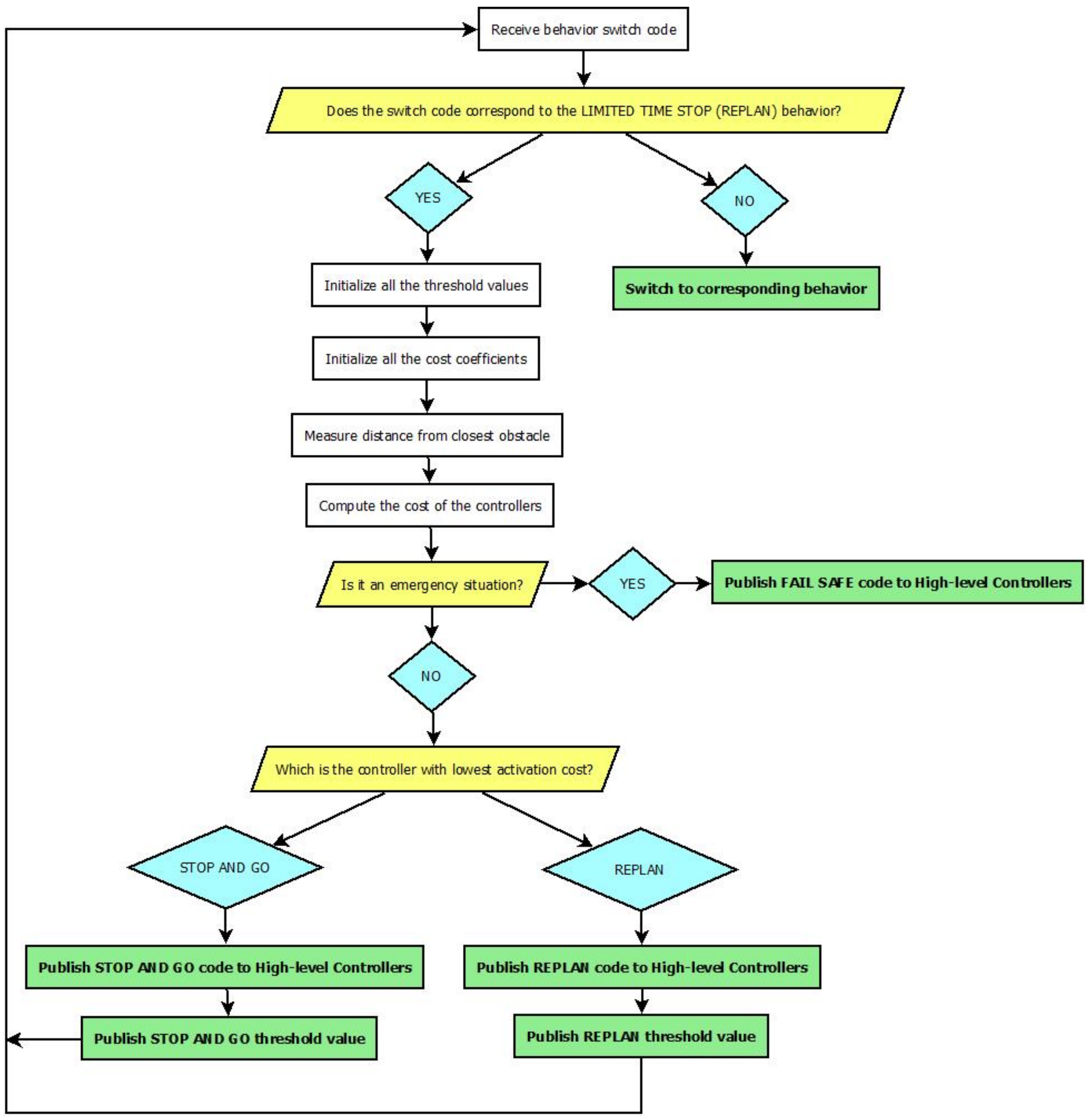

Receive behavior switch code
Does the switch code correspond to the LIMITED TIME STOP (REPLAN) behavior?
YES
NO
Initialize all the threshold values
Switch to corresponding behavior
Initialize all the cost coefficients
Measure distance from closest obstacle
Compute the cost of the controllers
Is it an emergency situation?
YES
Publish FAIL SAFE code to High-level Controllers
NO
Which is the controller with lowest activation cost?
STOP AND GO
REPLAN
Publish STOP AND GO code to High-level Controllers
Publish REPLAN code to High-level Controllers
Publish STOP AND GO threshold value
Publish REPLAN threshold value

Figure 5.14: Flowchart of the Limited Time Stop (Replan) behavior

5.3.2 Limited Time Stop (Reconnect)

The “Limited Time Stop (Reconnect)” behavior is very similar to the previous one. Also in this case, the robot is stopped in front of a detected obstacle by the “Stop and Go” controller and then a decision has to be made. If the delay limit set by the user is exceeded, then the “Reconnect” controller gets activated, in order to plan a path going from the current position of the manipulator to the first valid point of the original trajectory after the obstacle. Additionally, any emergency situation would trigger an immediate switch to the “Fail Safe” controller to ensure the safety of the operator. This behavior is suited for all the cases in which the focus is still on the performance of the task execution but, at the same time, the user is interested in keeping the manipulator as close as possible to the original trajectory. Table 5.3 reports the values of the coefficients of the cost function specific of this behavior, set by the authors to guarantee the desired reaction of the robot.

High Level Controller	C_{base}	C_{dist}	C_{delay}
Stop and Go	0	0	1
Reconnect	$cycle\_frequency \cdot time\_limit$	0	0

Table 5.3: Table of Coefficients for the Limited Time Sop (Reconnect) behavior

With reference to Table 5.3, this logic is very similar to the previous one. The table of coefficients for the cost function as well as the flowchart of implementation, presented in Figure 5.15, are almost identical to the previous behavior. The only difference that can be noticed is that, when the delay limit is exceeded, the algorithm publishes to “High-level Controllers” the switch code corresponding to the “Reconnect” controller, thus triggering its activation.

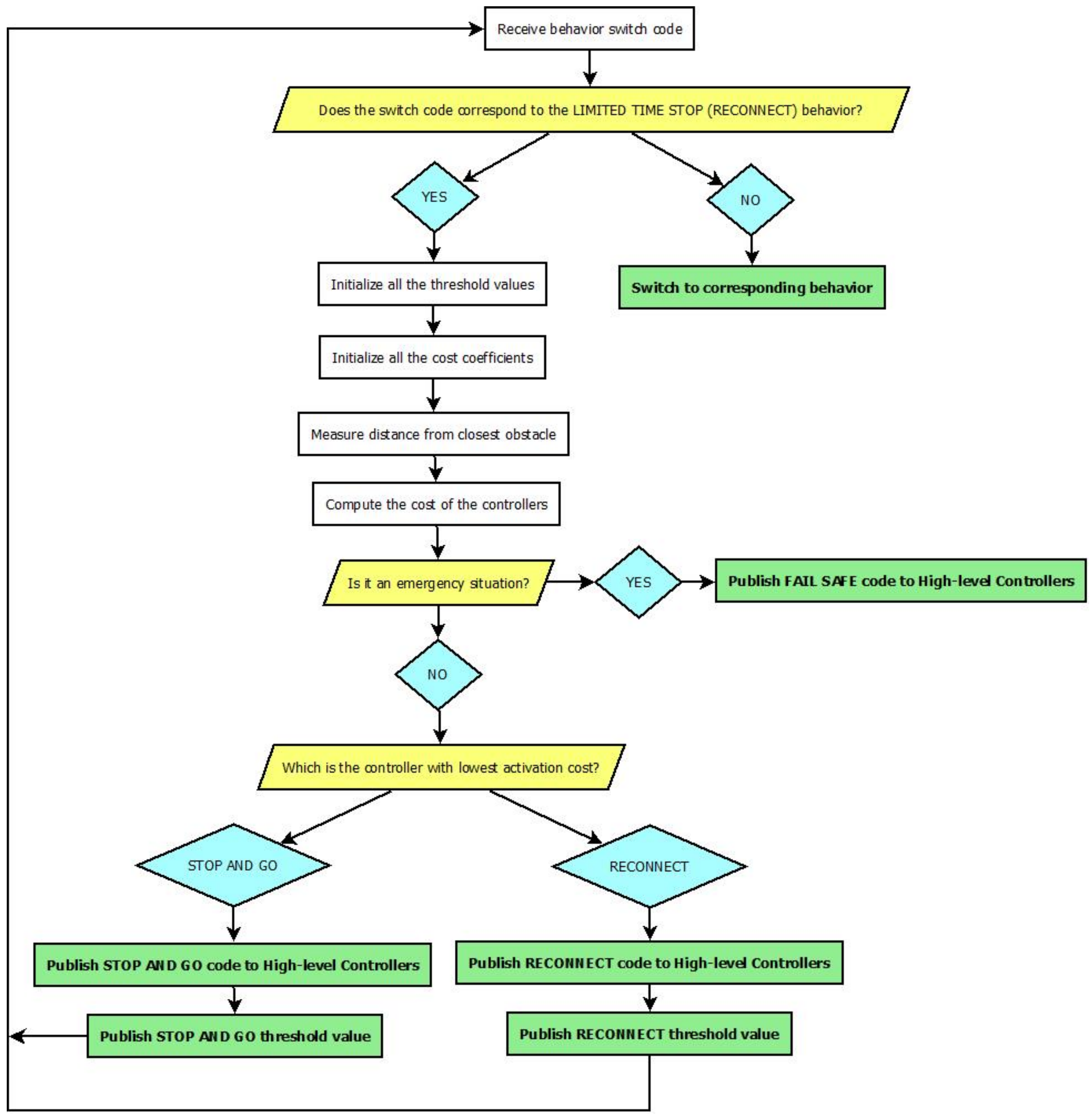

Receive behavior switch code
Does the switch code correspond to the LIMITED TIME STOP (RECONNECT) behavior?
YES
NO
Initialize all the threshold values
Switch to corresponding behavior
Initialize all the cost coefficients
Measure distance from closest obstacle
Compute the cost of the controllers
Is it an emergency situation?
YES
Publish FAIL SAFE code to High-level Controllers
NO
Which is the controller with lowest activation cost?
STOP AND GO
RECONNECT
Publish STOP AND GO code to High-level Controllers
Publish RECONNECT code to High-level Controllers
Publish STOP AND GO threshold value
Publish RECONNECT threshold value

Figure 5.15: Flowchart of the Limited Time Stop (Reconnect) behavior

5.3.3 Unlimited Time Stop

Considering a growing interest in keeping the trajectory close to the one calculated offline, the authors developed the so called "Unlimited Time Stop". With this behavior, the robot stops in front of a detected obstacle, but the only allowed reaction is to wait for the obstacle to move away before restarting the motion along the original path. For all the cases in which the offline planning technique used to compute the trajectory of the segment is very elaborate and time consuming, the user has now the possibility of choosing this behavior in order to make sure that it is never modified. For instance, "Unlimited Time Stop" can be applied to trajectories calculated using the "Human Occupancy Volumes" approach, explained in Chapter 4. In fact, the mentioned algorithm defines the best trajectory to follow, also considering the possible stops due to the interaction with an obstacle.

In this case, the only controller belonging to this behavior is the "Stop and Go", which is therefore kept active for the entire execution of the segment, unless an emergency occurs. This means that no cost function is required and, as can be seen from the flowchart in Figure 5.16, the behavior simply has to publish the switch code corresponding to "Stop and Go", together with the threshold to be used.

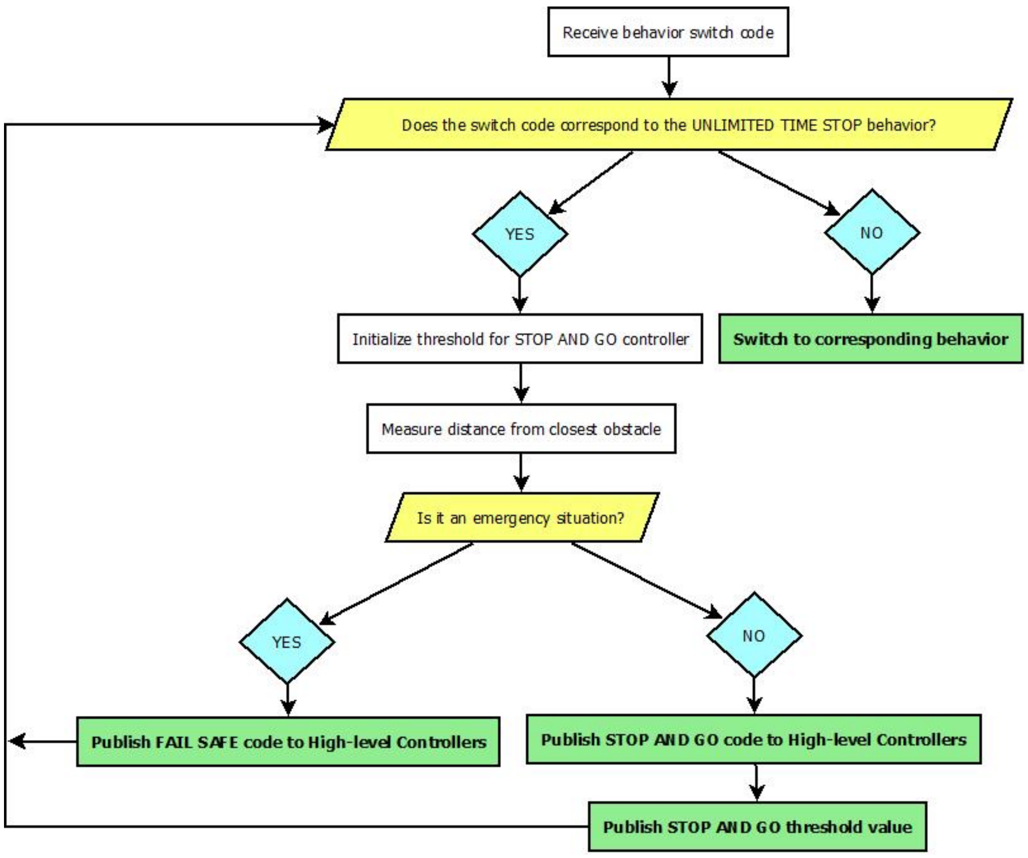

Receive behavior switch code
Does the switch code correspond to the UNLIMITED TIME STOP behavior?
YES
NO
Initialize threshold for STOP AND GO controller
Switch to corresponding behavior
Measure distance from closest obstacle
Is it an emergency situation?
YES
NO
Publish FAIL SAFE code to High-level Controllers
Publish STOP AND GO code to High-level Controllers
Publish STOP AND GO threshold value

Figure 5.16: Flowchart of the Unlimited Time Stop behavior

5.3.4 Robot Trajectory (Non-Restrictive)

The idea behind the “Robot Trajectory (Non-Restrictive)” behavior is to provide the user with a solution to all the cases in which the main interest is to let the robot perform its motion without being stopped. Regarding the behaviors presented before, the robot was adapting its movement according to the presence of the human operator. In this case, the aim is to obtain the opposite result: the motion of the manipulator is considered more important than the task of the operator and, therefore, the operator is the one that should adapt to the situation. Therefore, the problem is to develop an efficient way of communication with the human that allows the system to warn the operator whenever his/her movements are not compatible with the task of the manipulator. The solution proposed by the authors constantly monitors the position of the obstacles with respect to the robot. If the an obstruction is detected within a certain warning area, then an alert sound is activated in order to alert the operator of the possible interference. Regarding the name of the behavior, the expression “Non-Restrictive” is used because a stop in the robot motion is not desirable, but still acceptable. If the obstacle inside the warning area does not move away and the measured distance reaches a critical level set by the user, then the robot stops and waits for the obstacle to move away. Moreover, if any emergency situation occurs, the behavior triggers the immediate action of the “Fail Safe” controller.

The controllers that belong to this behavior are “Alert” and “Stop and Go” and their corresponding coefficients used in the cost function are listed in Table 5.4.

High Level Controller	C_{base}	C_{dist}	C_{delay}
Alert	0	1	0
Stop and Go	$\frac{1}{Critical\_distance}$	0	0

Table 5.4: Table of Coefficients for the Robot Trajectory (Non-Restrictive) behavior

With reference to Table 5.4, C_{dist} has been set to 1 for “Alert” and to 0 for “Stop and Go”. This means that, as the value of distance decreases, the distance cost rises only for the “Alert” controller. Moreover, the value of delay is not monitored. In this case, the decision process that triggers the switch is only based on the distance between the obstacle and the manipulator and therefore both the C_{delay} coefficients have been set to 0. C_{base}, instead, is calculated in order to trigger the switch at a critical value of distance, chosen by the user. This value represents the safety distance used to pause the robot motion if the operator has not followed the warning provided by the system. To provide an example of the effect of these coefficients, the authors selected the following values: the warning zone starts at a distance of 0.3 m from the robot, while the critical distance has been set equal to 0.1 m. A practical representation of the effect of these values is reported below in Table 5.5. As displayed, when the obstacle is still outside the warning area, the “Alert” controller is active, but no sound is produced. The second column shows an obstacle inside the warning zone, meaning that the active controller is still “Alert” but, in this case, a sound alarm is produced to force the operator to leave that area. Finally, as the distance reaches a value smaller than the critical one, the cost of “Stop and Go” makes it the most convenient controller, therefore triggering the switch to pause the robot movement.

Controller	**Distance = 0.31 m**	**Distance = 0.2 m**	**Distance = 0.09 m**
$Cost_{Alert}$	3.23	5	11.11
$Cost_{StopAndGo}$	10	10	10
Active Controller	**Alert (no sound)**	**Alert (sound)**	**Stop and Go**

Table 5.5: Table with an example of cost values during the execution of a segment characterized by the Robot Trajectory (Non-Restrictive) behavior

The flowchart in Figure 5.17, shows the logic used to implement this behavior. As represented, the structure of the code is very similar to “Limited Time Stop”. The only differences are that the value of delay is not measured, since it does not influence the costs of activation, and that the switch codes available for this behavior are related to its specific group of controllers (“Alert” and “Stop and Go”).

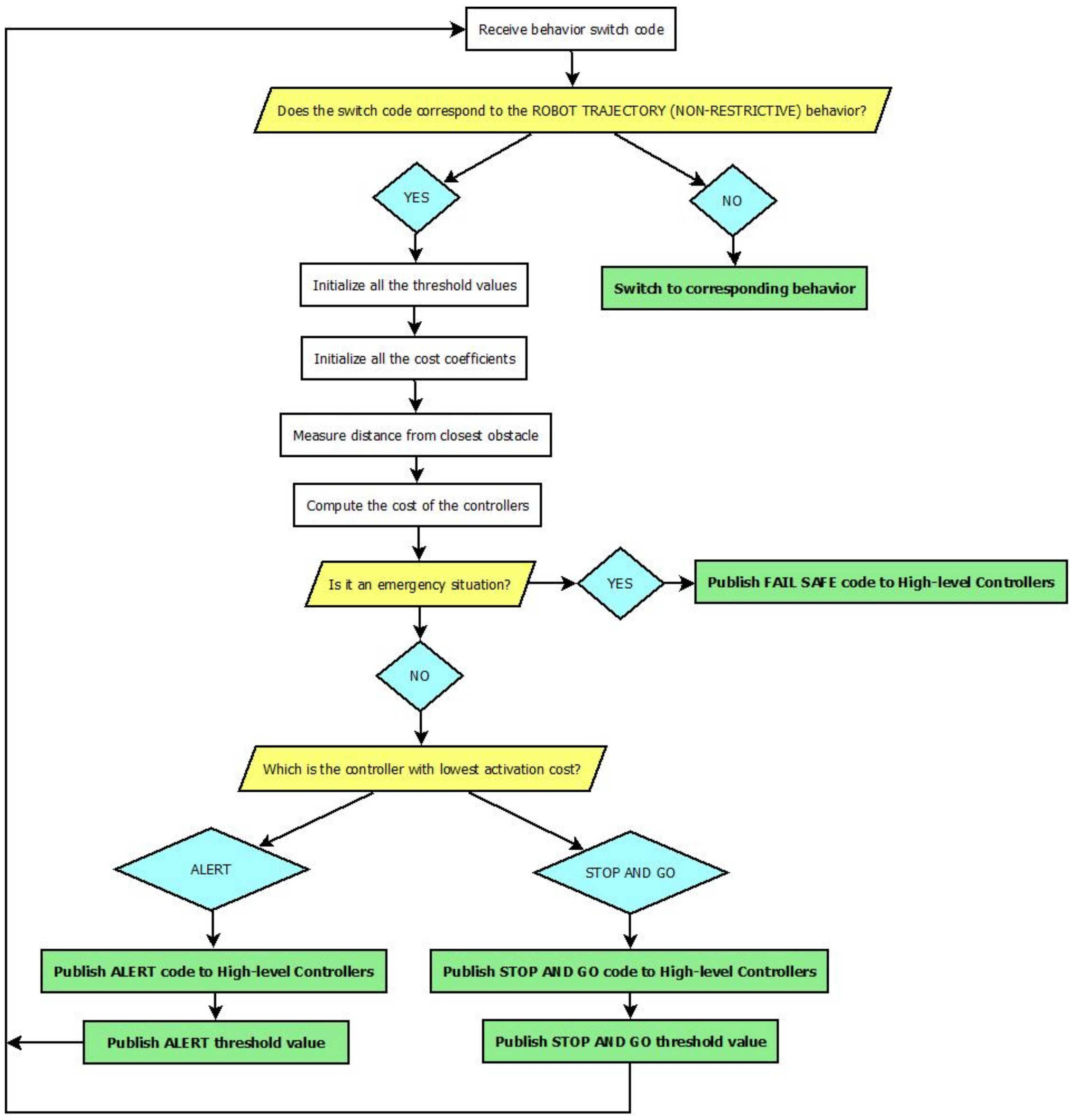

Receive behavior switch code
Does the switch code correspond to the ROBOT TRAJECTORY (NON-RESTRICTIVE) behavior?
YES
NO
Initialize all the threshold values
Switch to corresponding behavior
Initialize all the cost coefficients
Measure distance from closest obstacle
Compute the cost of the controllers
Is it an emergency situation?
YES
Publish FAIL SAFE code to High-level Controllers
NO
Which is the controller with lowest activation cost?
ALERT
STOP AND GO
Publish ALERT code to High-level Controllers
Publish STOP AND GO code to High-level Controllers
Publish ALERT threshold value
Publish STOP AND GO threshold value

Figure 5.17: Flowchart of the Robot Trajectory (Non-Restrictive) behavior

5.3.5 Robot Trajectory (Restrictive)

The behavior presented here is very similar to the previous one but represents a more ”restrictive” version of it. Again, an obstacle inside the warning area triggers the activation of a sound alarm to notify the operator of the situation. However, in this case, any interruption of the robot motion must be avoided without exceptions. This means that the robot is not allowed to enter any pause condition. Therefore, even if the obstacle does not react accordingly to the warning, the robot keeps moving, unless the situation becomes so critical that the “Fail Safe” controller is triggered. This behavior is therefore suited for all the cases in which any disturbance of the robot trajectory execution would lead to a damage of the workpiece and, therefore, must be avoided unless it is due to an emergency.

As in “Unlimited Time Stop”, only one controller is involved in the implementation of this behavior and there is no need for a cost function. The switch code related to “Alert” and its value of threshold are published to the “High-level Controllers” node for the entire execution of the segment, unless an emergency requires the intervention of the “Fail Safe” controller, as represented in the flowchart in Figure 5.18.

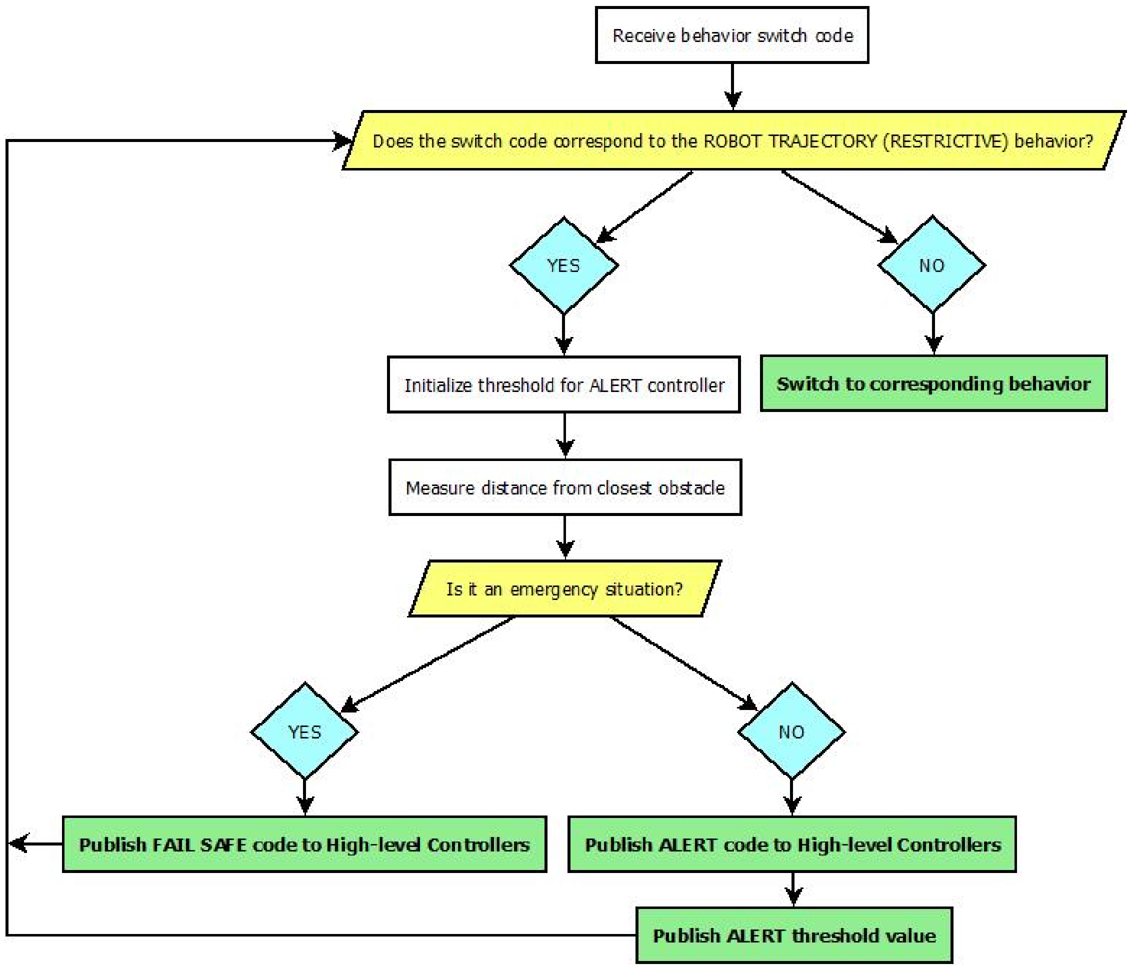

Receive behavior switch code
Does the switch code correspond to the ROBOT TRAJECTORY (RESTRICTIVE) behavior?
YES
NO
Initialize threshold for ALERT controller
Switch to corresponding behavior
Measure distance from closest obstacle
Is it an emergency situation?
YES
NO
Publish FAIL SAFE code to High-level Controllers
Publish ALERT code to High-level Controllers
Publish ALERT threshold value

Figure 5.18: Flowchart of the Robot Trajectory (Restrictive) behavior

5.3.6 Allowed Contact

There are many situations in which the user needs to perform manual operations directly on the robot. For instance, the manipulator may be stopped in a certain position in order to allow the inspection of the end-effector or of the carried workpiece. In all those cases, contact with the operator must be permitted, and not detected as a collision, until the robotic system receives a confirmation signal from the user. The signal means that the operator has finished its operations on the robot and the execution of the next segment can be commanded. This problem has already been solved by the "Human Contact" controller but, up to now, none of the implemented behaviors included it in their group of available controllers. The role of the "Allowed Contact" behavior is therefore to simply publish the switch code that activates "Human Contact". As soon as this behavior is requested, the robot is stationary in the required position and, therefore, there is no need to monitor for emergency situations (in terms of distance from the manipulator). The simple logic needed for the implementation of this behavior is represented below by the flowchart in Figure 5.19.

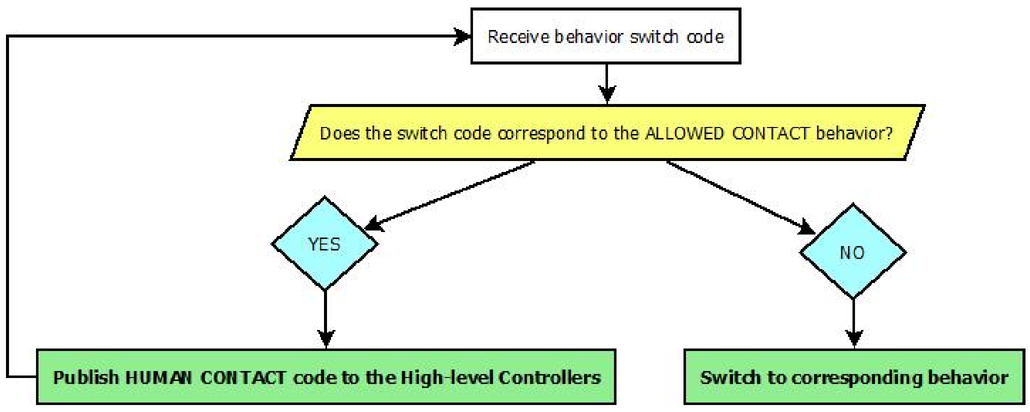
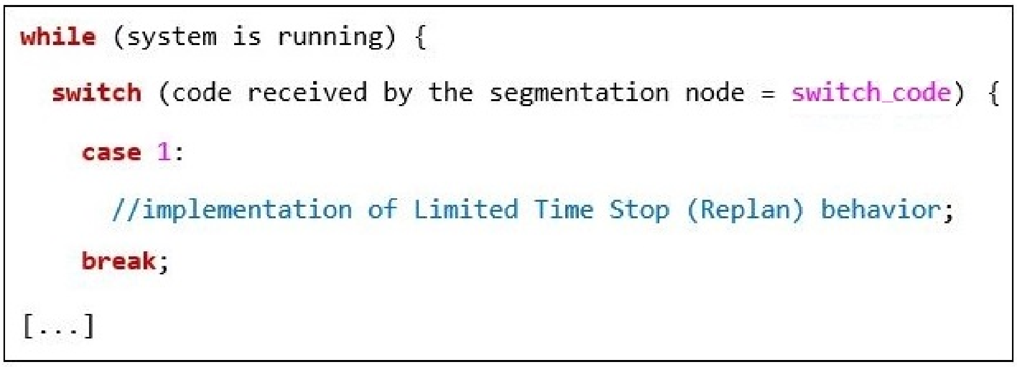

Receive behavior switch code
Does the switch code correspond to the ALLOWED CONTACT behavior?
YES
NO
Publish HUMAN CONTACT code to the High-level Controllers
Switch to corresponding behavior

Figure 5.19: Flowchart of the Allowed Contact behavior

5.3.7 Code modularity

Finally, in order to obtain a modular structure, the authors approached the structure of this code similarly to the code related to the High-level Controllers. Each behavior is implemented inside individual cases of a *switch* module, as shown in the Figure 5.20. Every time the execution of a new segment starts, the "Segmentation" node publishes the code corresponding to the behavior chosen by the user for that particular segment. The switch code is received by the "Robot Behaviors" node and used to activate the correct section of code. This approach, as already said for "High-level Controllers", guarantees a high level of modularity of the system since a behavior can be added, discarded or modified by simply adding, discarding or modifying the corresponding case of the switch structure.



```
while (system is running) {
  switch (code received by the segmentation node = switch_code) {
    case 1:
      //implementation of Limited Time Stop (Replan) behavior;
    break;
[...]
```

Figure 5.20: Pseudo-code of modular structure containing the developed Robot Behaviors

Chapter 6

Case study: Results and discussion

In order to validate the control architecture developed and presented in the previous chapters, a hardware-in-the-loop testbed setup is used. An industrial collaborative scenario is proposed by the authors in order to show and test the capabilities of the system and its performance in terms of safety and productivity. The experimental activity is performed by a physical e.DO robot equipped with a two-prongs gripper able to perform pick and place operations. The robot, together with a human operator, executes a series of predefined tasks in order to complete a collaborative assembly operation. Due to the early stages of the project at the time of the authors' research, sensing of a real-life operator was not possible and therefore all the perception data needed to represent the human actions during the execution of the scenario have been emulated. In particular, the original goal of the authors was the dynamic emulation of the whole body of the operator, but software limitations led to unexpected crashes of the simulation platform. For this reason, the authors chose to dynamically emulate only the forearm of the operator, since it is the main body part actively interacting with the robot. The presence of the remaining body parts was still taken into account as a static obstacle inside the workspace. The implementation of a more stable simulation, capable of achieving the original goal, is therefore left for future developments.

6.1 Scenario description and robot task segmentation

In this section, a detailed presentation of the collaborative scenario under analysis is reported. The authors chose to validate the robotic system by means of a simple assembly task, representative of common real industrial scenarios. In the example, manipulator and human share the same workspace and the assembly of the product is carried out in a collaborative manner. In Figure 6.1, the product taken into account is represented: it is made up of four components, in the example colored blocks, which are placed in different locations of the workspace. Some of the operations required for its production are assigned to the human, others are responsibility of the robot and some direct interaction between the two is also required. During this collaboration, human intervention may intersect the motion of the manipulator, generating the need for smart robot reactions in order to ensure the safety of the operator while at the same time attempting to minimize the disruption of production efficiency. Always with reference to Figure 6.1, a customization of the same product, used to test the flexibility of the system, is represented. The difference between the two versions lays in the location of their components, meaning that a reprogramming of the robotic task is needed, while the operator can easily adapt to the new duties. In the example, the outcome of the customization is simply a product composed of blocks of different colors, but it is used as a generic representation of real industrial scenarios.

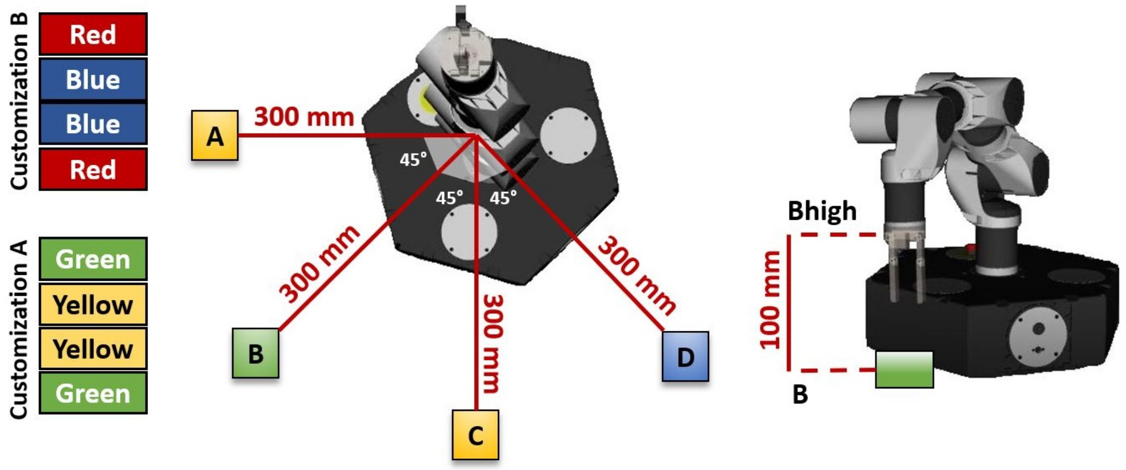


Figure 6.1: Schematic representation of the two product customizations and of the testbed setup

As depicted in Figure 6.2, the robot, starting from its stand-by vertical position, moves to grasp the first component in location A (1) and then transfers it towards the second component located in C. At the same time, the operator prepares the common assembly area (2), located in B between the two parts. The robot performs the assembly of the two components (3), moves away and waits for the operator to ensure the solidity of the subassembly (4). After the inspection, the operator gives a confirmation signal by pressing the ENTER key on the keyboard and triggers the manipulator to start its next task. The subassembly is moved by the robot towards B, where the operator has placed the third component with the correct orientation (5). As the end-effector reaches the predefined location where the assembly will take place, it stops and allows the operator to check the correct alignment of the three parts (6). After another confirmation signal is given, the robot performs the next step of the assembly operation by joining the subassembly with the third component (7). At this point, the robot has completed successfully all its tasks and therefore moves back to its vertical stand-by position, while the operator assembles the fourth and last component and retrieves the finished product (8). In Figure 6.2, the production of Customization A is depicted. Being the first component a yellow block (location A), the robot is required to move to location C in order to assemble it with the component of the same color, while the human, knowing the composition of the specific product, automatically inserts green parts. Regarding the production of Customization B, the process to be followed would be very similar to the one just presented. The first component fed to the robot (location A) would be a blue block and therefore the manipulator would have to move to location D in order to assemble it with a second blue component. Once again, the human would naturally adapt to the new production cycle by inserting red components in the assembly.

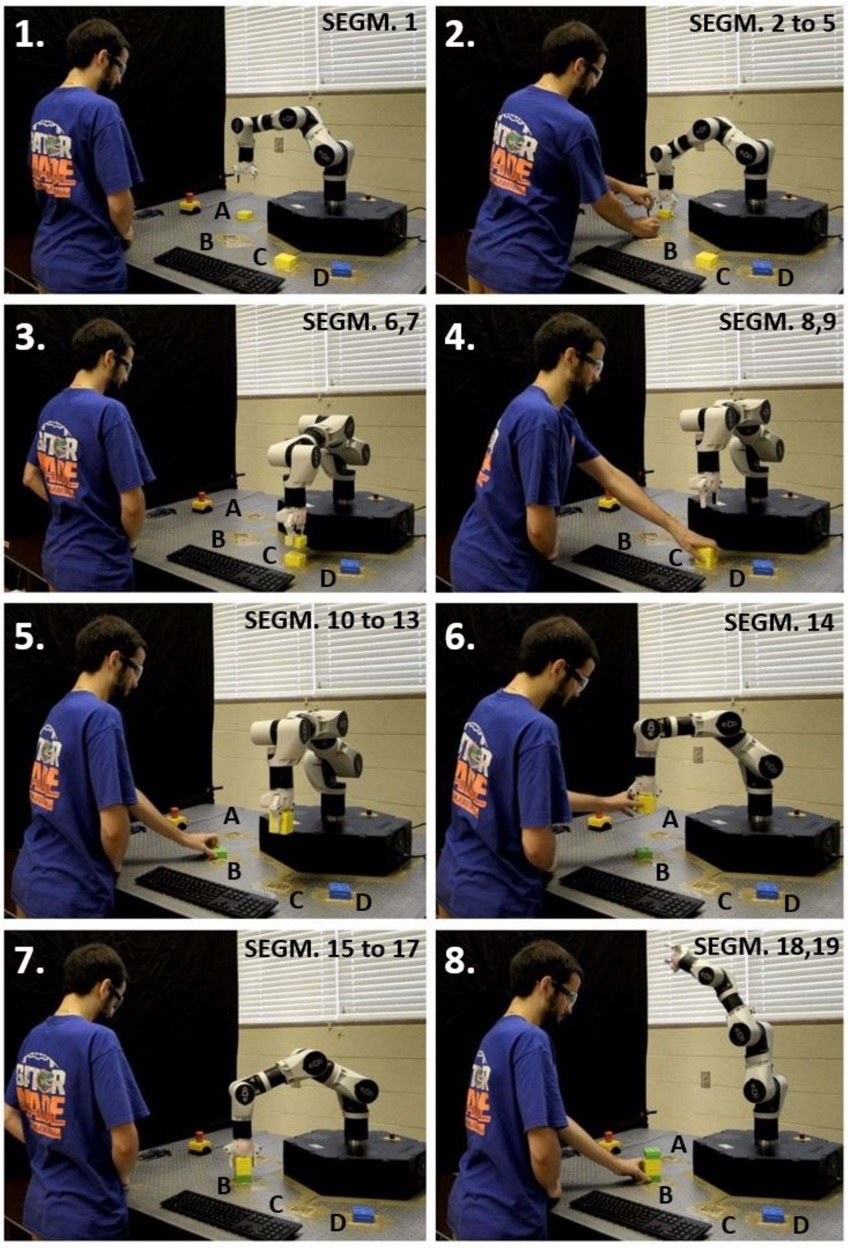


Figure 6.2: Main steps of the assembly operation under analysis (Customization A)

Now that the collaborative scenario has been presented, it is duty of the user to correctly program the robot. According to the segmentation approach introduced in Chapter 4, the robot motion is divided into consecutive segments and each segment is assigned a certain offline planning technique (Chapter 4) and online behavior (Chapter 5), based on the specific requirements of each individual segment. In this example, the logic used to perform the segmentation process is strongly task-oriented, meaning that each segment is either a movement between two key points of the task or an action of the end-effector. This segmentation logic is the most intuitive one, but many different ways of segmenting the same scenario could be chosen, depending on the needs of the user. In Table 6.1, a brief description of the manipulator's motion corresponding to each segment together with the assigned characteristics is reported (note that for each location A, B, C and D indicated in Figures 6.1 and 6.2 a corresponding position just above them has been defined as A_{high}, B_{high}, C_{high} and D_{high} respectively):

Segment	**Robot motion**	**Offline Planning Technique**	**Online Robot Behavior**
1	Move from stand-by vertical position to A_{high}	UDA	LTS
2	Move from A_{high} down to A	UDA	RT(R)
3	Close the gripper to grasp first component	TO	RT(R)
4	Move from A up to A_{high}	UDA	RT(NR)
5	Move from A_{high} to C_{high}	UDA	LTS
6	Move from C_{high} down to C	UDA	RT(R)
7	Open the gripper to assemble first and second components	TO	RT(R)
8	Move from C up to C_{high}	UDA	RT(NR)
9	Wait for operator consent in C_{high}	UDA	HC
10	Move from C_{high} down to C	UDA	RT(R)
11	Close the gripper to grasp subassembly	TO	RT(R)
12	Move from C up to C_{high}	UDA	RT(NR)
13	Move from C_{high} to B_{high}	UDA	UTS
14	Wait for operator consent in B_{high}	TO	HC
15	Move from B_{high} down to B	UDA	RT(R)
16	Open the gripper to assemble the third component	TO	RT(R)
17	Move from B up to B_{high}	UDA	RT(NR)
18	Move from B_{high} to A_{high}	UDA	LTS
19	Move from A_{high} to the vertical stand-by position	UDA	LTS

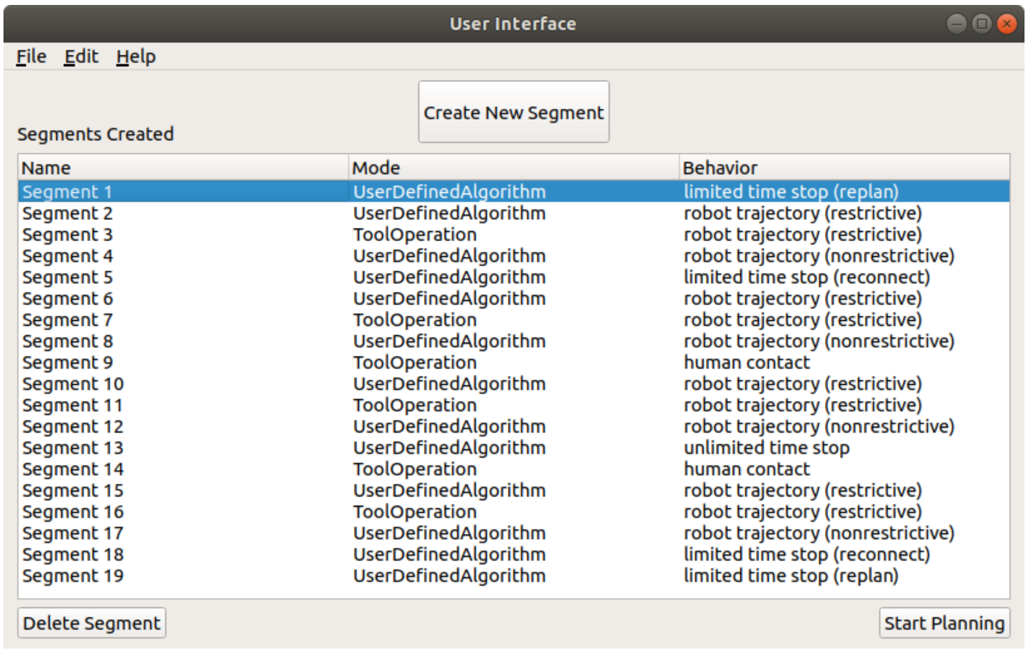
Legend:

UDA	User Defined Algorithm
TO	Tool Operation
LTS	Limited Time Stop
RT(R)	Robot Trajectory (Restrictive)
RT(NR)	Robot Trajectory (Non-Restrictive)
HC	Human Contact
UTS	Unlimited Time Stop

Table 6.1: Robot motion, offline planning technique and robot behavior assigned to each segment

According to the segmentation approach, once the segments have been defined, they can be independently characterized in terms of how their trajectory will be planned offline (Chapter 4) and of how the robot will react online when executing them (Chapter 5). Table 6.1 summarizes the characteristics assigned to each segment. Regarding the offline planning, for all the segments in which a movement of the robotic arm is required, the User Defined Algorithm mode has been selected since no a-priori knowledge of the task is available. For the segments in which the robot has to open/close the gripper or wait for the operator's consent, the dedicated Tool Operation mode has been used. In terms of online robot behaviors, the authors made the following choices:

- For all the segments in which a wide motion of the robotic arm has to be performed, the Limited Time Stop behavior has been selected, since interactions with the human operator may occur (segments 1, 5, 18 and 19);
- For all the segments in which the manipulator has to be extremely precise, like when it is approaching, grasping or releasing a component, the Robot Trajectory (Restrictive) behavior has been chosen to make sure that no external disturbances are allowed during the execution of these tasks (segments 2, 3, 6, 7, 10, 11, 15 and 16);
- For all the segments in which the manipulator moves away from a component, the required precision of the task is lower and therefore the Robot Trajectory (Non-restrictive) behavior can be used (segments 4, 8, 12 and 17);
- When the robot has to remain stationary in one position, while the human operator performs a certain activity, until the confirmation signal is received, the Human Contact behavior has been selected (segments 9 and 14);
- Segment 13 has been assigned the Unlimited Time Stop behavior because the most likely interaction with the operator would happen at the goal of the segment. Therefore no replanning is possible, since the goal of the segment is obstructed, and the only solution is to wait for the path to be cleared.



Name	Mode	Behavior
Segment 1	UserDefinedAlgorithm	limited time stop (replan)
Segment 2	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 3	ToolOperation	robot trajectory (restrictive)
Segment 4	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 5	UserDefinedAlgorithm	limited time stop (reconnect)
Segment 6	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 7	ToolOperation	robot trajectory (restrictive)
Segment 8	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 9	ToolOperation	human contact
Segment 10	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 11	ToolOperation	robot trajectory (restrictive)
Segment 12	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 13	UserDefinedAlgorithm	unlimited time stop
Segment 14	ToolOperation	human contact
Segment 15	UserDefinedAlgorithm	robot trajectory (restrictive)
Segment 16	ToolOperation	robot trajectory (restrictive)
Segment 17	UserDefinedAlgorithm	robot trajectory (nonrestrictive)
Segment 18	UserDefinedAlgorithm	limited time stop (reconnect)
Segment 19	UserDefinedAlgorithm	limited time stop (replan)

Figure 6.3: GUI overview window for the scenario under analysis

This scenario has been programmed offline using the dedicated GUI, introduced in Chapter 4. Figure 6.3 shows the overview window of the interface, in which the segments specified in Table 6.1 and their assigned characteristics are reported.

In order to further test the reactive behaviors of the robot during the execution of the scenario, three different level of human-robot interaction have been analyzed:

1. **CASE 1**: Since the scenario have been defined offline with the aim of achieving the maximum productivity level, if everything goes as expected, the human presence does not induce any delay in the motion of the manipulator. In this case, the robot is able to move through its predefined trajectories continuously without any stop caused by the operator's forearm.
2. **CASE 2**: In this variant of the scenario, fixturing of the assembly area performed by the operator takes a little longer than expected. Consequently, during the execution of segment 5, the robot finds the operator's forearm obstructing its way and has to stop in order to avoid a collision. The same happens again during the execution of segment 13.
3. **CASE 3**: This third variant represents the case in which a certain problem occurs during the preparation of the common assembly area. For this reason, the operator, in charge of solving the problem, remains in the way of the robot (segment 5) for a much longer period of time. Therefore, the robot has to react in order to mitigate the loss in performance by accomplishing its task while the operator still remains on the original path. As in the second variant, a short delay also occurs during the execution of segment 13.

6.2 Performance of the robotic system

6.2.1 Offline performance

The scenario under analysis have been programmed using the dedicated Graphical User Interface, developed for this project and introduced in Chapter 4. Once the step of manual data input has been carried out by the user, the GUI triggers the system to start the planning process according to the user's requests. Timers have been implemented inside the developed codes in order to measure the performance of the system in its offline duties. In particular the authors are interested in four values: duration of the process needed to read all the information provided by the user through the GUI, duration of the offline planning process used to generate all the predefined robot trajectories, duration of the analysis of the segments aimed to check their correct connection and total duration of the processes. In Table 6.2, the collected results for the two customizations are reported:

	Customization A [ms]	**Customization B [ms]**
Reading process	0.6	0.6
Planning process	765.9	772.7
Connection process	5.1	5.9
Total duration	771.6	779,2

Table 6.2: Measured times for the offline performance of the robotic system

Analyzing the terms reported in Table 6.2, it can be stated that the most relevant value is the one related to the planning process. The reading process duration, measured for the example under analysis, is almost irrelevant if compared to the other terms. Regarding the connection process, the small values measured for both Customization A and B are justified by the fact that the segments had already been correctly created by the user and therefore no connections had to be planned. In general, it must be said that the duration of the three process is strictly related to the complexity of the programmed task. The more complex the scenario to be programmed, the higher the three values and therefore the total duration of the offline computations. Considering the example under analysis, the total duration of the processes carried out by the robotic system is less than 1 s for both Customization A and Customization B. Comparing this value with the time needed by the user for the manual input of all the characteristics of the segments (not precisely measurable, but in the order of minutes), it is clear that the duration of the offline processes above do not impact the overall offline performances of the system. Given the fact that more complex scenarios would require longer times both for the manual data input and for the offline computations, it is safe to say that the most time-consuming offline step is represented by the manual user data input. Thanks to the GUI, this step is required only the first time that a scenario is introduced in the production system. Once a robotic task has been defined, the GUI offers the possibility to save it, therefore creating a library of scenarios that can be retrieved every time that the specific production process is required. This means that, supposing that a library of programmed scenarios is already available, the setup required to start performing one of them is limited, in terms of offline computation, to total duration values like the ones reported in Table 6.2. Consequently, the developed robotic system would be perfectly suited for constantly changing "Pull Manufacturing" environments, as it provides high levels of flexibility and adaptability. In fact, every time that a different product is requested, if its production process is already available in the library, the manipulator is able to quickly adapt to the situation by simply retrieving the existing information. Moreover, the ability to quickly modify an existing scenario, in order to accommodate customized versions of a product, has been validated with the definition of Customization B. Supposing that the program for Customization A has already been created by the user and saved in the library of programmed tasks, defining the program of Customization B from scratch would represent a waste of time since it is only slightly different from the first one. Thanks to the GUI, the user can simply reopen the program of Customization A, modify the segments according to the customized process and save it as a new independent scenario to be added to the library, therefore minimizing the time required for the manual input of all the information. Finally, the user interface was demonstrated to be very effective in enabling non expert users to intuitively interact with the robotic system with minimal needed technical knowledge.

6.2.2 Online performance

As previously introduced, the robotic system has been tested in its online performances with three different levels of human-robot interaction for each product customization. A summary of the results related to Customization A is presented in Figure 6.4, while Figure 6.5 refers to Customization B. The two graphs can be used to compare the online performance of the system depending on the different levels of human-robot interaction. On the x-axis the specific segment under execution is indicated, while the y-axis shows the time passed from the beginning of the task.

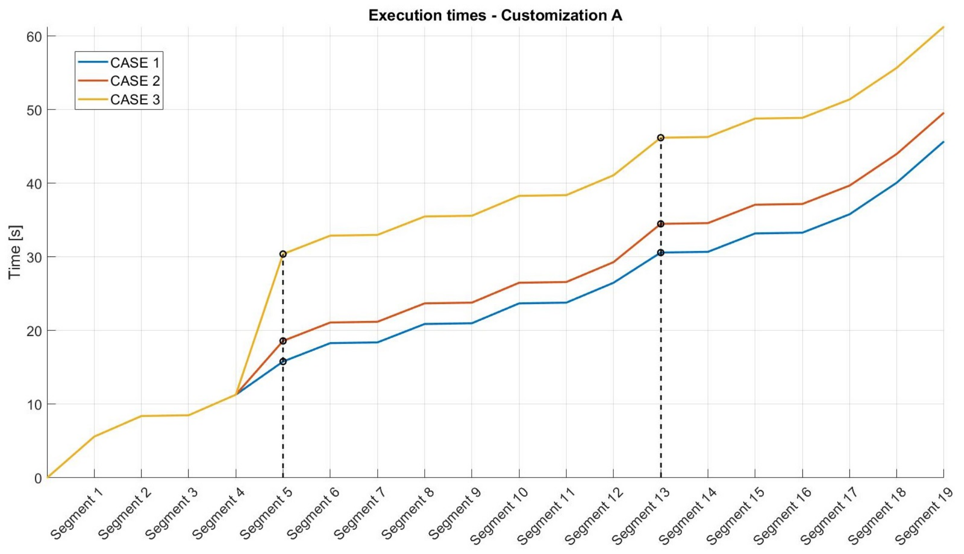


Figure 6.4: Execution times for the segments of Customization A with three different levels of human-robot interaction

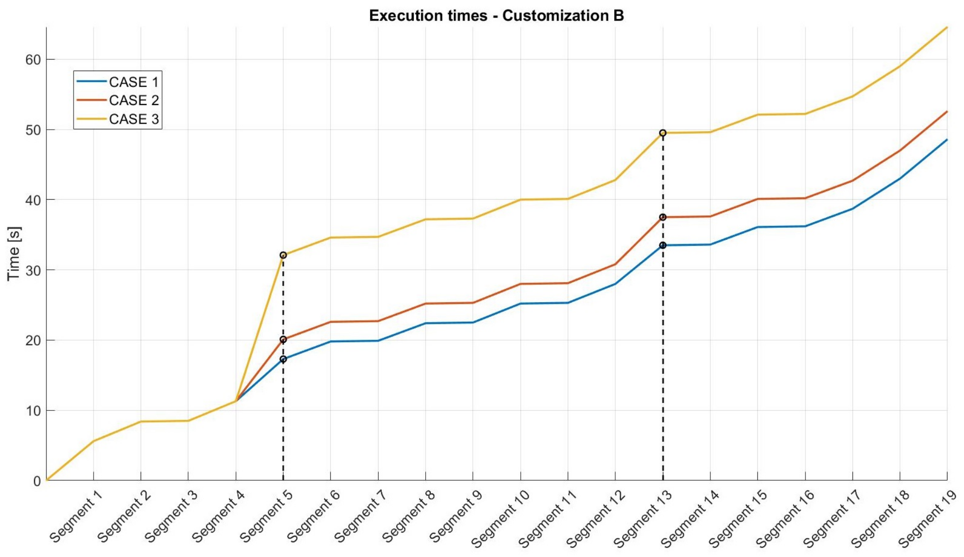


Figure 6.5: Execution times for the segments of Customization B with three different levels of human-robot interaction

As shown in both Figure 6.4 and 6.5, the lines for the three cases analyzed coincide until the end of Segment 4. In fact, up to that point (when the robot has lifted the first component), no human interference occurs for any of the cases and therefore the same execution times are measured. Segment 5 is still undisturbed for Case 1, while a delay in the robot's motion is introduced for both Case 2 and 3 because of the human presence on the path of the manipulator. As can be seen, the delay measured for Case 2 is shorter than the one related to Case 3 because, as previously stated, in the latter case the operator is experiencing some problems in the preparation of the assembly area and therefore obstructs the robot's path for a much longer time. From Segment 6 till Segment 12 the three lines develop in parallel, since no further delay is introduced. Segment 13 is again undisturbed for Case 1, while a short delay is induced for Cases 2 and 3 by the human intervention.

Furthermore, Figure 6.6 can be used to compare the results obtained for the two customizations. Customization A is represented with solid lines, while dashed lines are used for Customization B and, for each of them, the three cases under analysis are reported. A vertical dashed line is used in order to point out the segments where a difference between the two customizations exists. As can be seen from Figure 6.6, higher times have been registered for Segments 5 and 13 of Customization B. The reason for this simply lays in the fact that those two segments are the only ones that have been modified in order to accommodate the production process of the second customization. Since Segments 5 and 13 of Customization B are characterized by longer trajectories than the same segments for Customization A, longer execution times were expected by the authors. Regarding all the other segments, the same values have been measured for the two scenarios and, for this reason, couples of lines (solid and dashed) related to the same case are generally parallel to each other.

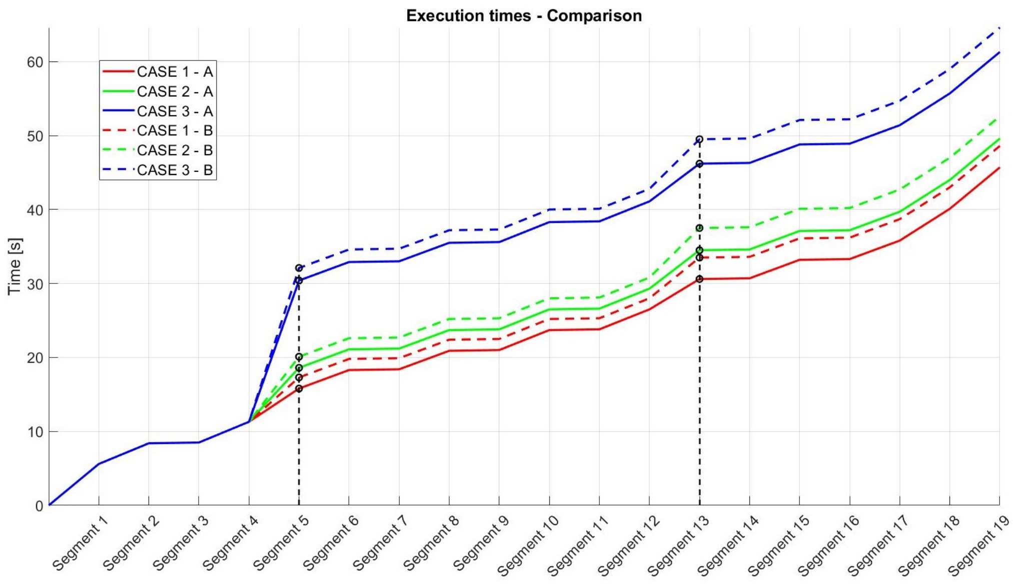


Figure 6.6: Comparison of the execution times for Customizations A and B

Now, a more detailed analysis is performed considering Segment 5 of Customization A, where the main interaction with the human happens. The authors aim to provide a deeper understanding of how the robot reacted online to the presence of the human for the three levels of interaction. An evaluation of the online performance, obtained through the smart decision logic implemented for the robotic system, is also presented in order to validate it as a suitable base framework for future developments.

Customization A, Case 1, Segment 5

As previously stated, Case 1 represents a situation of perfect synchronization of the individual tasks of human operator and robot and, for this reason, no interference is experienced. This example can be considered a reference measure of maximum performance for the specific task and is therefore used for a comparison with the other two cases, for which, instead, the movements of human and robot intersect. The times measured for this ideal case are reported in Table 6.3 below.

	Undisturbed execution	**Stopped robot**	**Disturbed execution**	**Computational time**	**Total time**
Actuated reaction [s]	4.5	-	-	-	4.5

Table 6.3: Reaction times for Customization A, Case 1, Segment 5

Customization A, Case 2, Segment 5

Segment 5 is characterized by the Limited Time Stop (Reconnect) behavior, which, as explained in Chapter 5, evaluates the best controller to be activated between Stop and Go and Reconnect on the basis of the values of obstacle proximity and induced delay. During the execution of this segment for Case 2, the robot finds the operator's forearm obstructing its way while moving from location A towards the location C. As it is performing this motion, the Stop and Go controller is active and, as the distance between the robot and the operator's forearm goes under the predefined threshold, it immediately stops the robot to ensure safety. As the operator clears the path, the manipulator is allowed to restart its motion along the original trajectory and reach its goal.
With reference to Figure 6.7, the trend of the cost of activation of the controllers and the values of the parameters considered to evaluate them during the execution of the segment can be analyzed. At the beginning of the segment, Stop and Go controller has the lowest cost of activation, the distance from the closest obstacle is around 280 mm and no delay has yet been induced. In this case, C_{dist} has been set to a value different from zero in order to see the effect of this parameter on the cost of the controllers. Therefore, as the value of distance decreases, the cost of both controllers rises until the proximity threshold (100 mm) is reached, around 1.2 s from the beginning. From the plots, it can be easily noticed that the cost of the Reconnect controller remains constant for about 2.8 s. On the other hand, the cost of the Stop and Go controller starts rising as soon as the stop is triggered,

since the robot motion has been paused and delay starts being measured. As the operator clears the path, the value of distance grows over the threshold once again and the Stop and Go controller commands the robot to restart its motion along the original trajectory. As soon as the robot starts moving again, the value of delay is reset to zero and therefore the cost of activation of the Stop and Go controller drops. In this case, the cost of activation of the Stop and Go controller did not increase enough to exceed the cost of Reconnect and therefore trigger a switch. For this reason, the Stop and Go controller remained active for the whole execution of the segment.

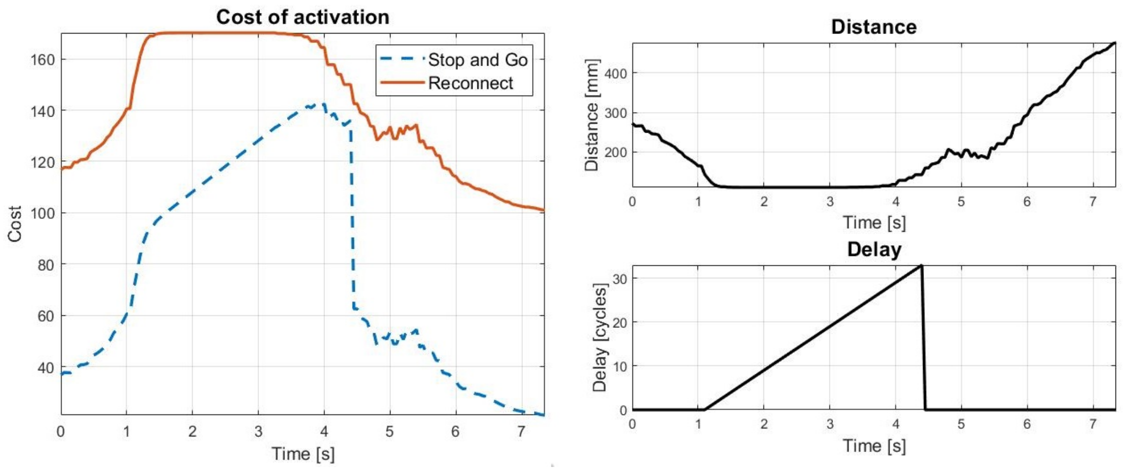


Figure 6.7: Trend of the cost of activation of the controllers in play and of the parameters used for the calculations in Case 2

The robot's reactive choice demonstrated to be effective in minimizing the disruption of the productivity of the system. As summarized in Table 6.4, the time needed by the robot to complete the segment is 7.3 s, which is basically the sum of the duration of the undisturbed trajectory (time to complete the segment if no interaction with the human occurs), equal to 4.5 s, and of the time during which the robot was paused, equal to 2.8 s. This result is not affected by the algorithm of the active controller since the process of updating the trajectory only takes around 0.002 s. If, instead, the control architecture would have immediately triggered a replanning operation around the obstacle, the total registered time to reach the goal would have been equal to 14.7 s, a much higher value than the previous one. It is therefore clear that, for all the cases in which the obstacle is going to clear the obstructed path quickly, the best choice is to wait and restart along the original trajectory, instead of immediately replanning around it.

	Undisturbed execution	**Stopped robot**	**Disturbed execution**	**Computational time**	**Total time**
Actuated reaction [s]	4.5	2.8	7.3	0.002	7.3
Discarded reaction [s]	4.5	-	14.6	0.14	14.7

Table 6.4: Reaction times for Customization A, Case 2, Segment 5

Customization A, Case 3, Segment 5

Considering Case 3, Segment 5 is still characterized by the Limited Time Stop (Reconnect) behavior. Once again, the robot moves from location A towards location B, but its path is obstructed by the operator that is experiencing some issues in the preparation of the common assembly area. The Stop and Go controller stops the robot as soon as the threshold (100 mm) is reached. After waiting for a certain period of time, the robot reacts to the induced delay by replanning its way around the operator's forearm and reaches its goal while the human is still working on the common assembly area. With reference to Figure 6.8, the cost of activation of the controllers can be analyzed in order to understand the logic behind the robot's reaction. As in the previous case, the execution of the segment starts under the control of Stop and Go and, as the robot approaches the human operator, the value of distance decreases till the threshold value (at 2.8 s). At that point, the robot stops and delay starts being measured. As in the previous case, the cost of Reconnect remains constant while the cost of Stop and Go grows. Around 6.8 s, the cost of Stop and Go becomes bigger than the cost of Reconnect and therefore the latter gets activated. As explained in detail in Chapter 5, the effect of this second controller is to trigger a virtual inflation of the obstacle (that can be seen in a sudden change in the distance plot), create a new plan that goes from the position in which the robot was stopped to the first valid point of the original trajectory and then follow it until the goal of the segment is reached. As soon as the robot starts the execution of the new plan, the delay is reset to zero and this causes the cost of the Stop and Go controller to become once again the most convenient one, causing a second switch. This means that the execution of the new computed plan is performed under the supervision of the Stop and Go controller.

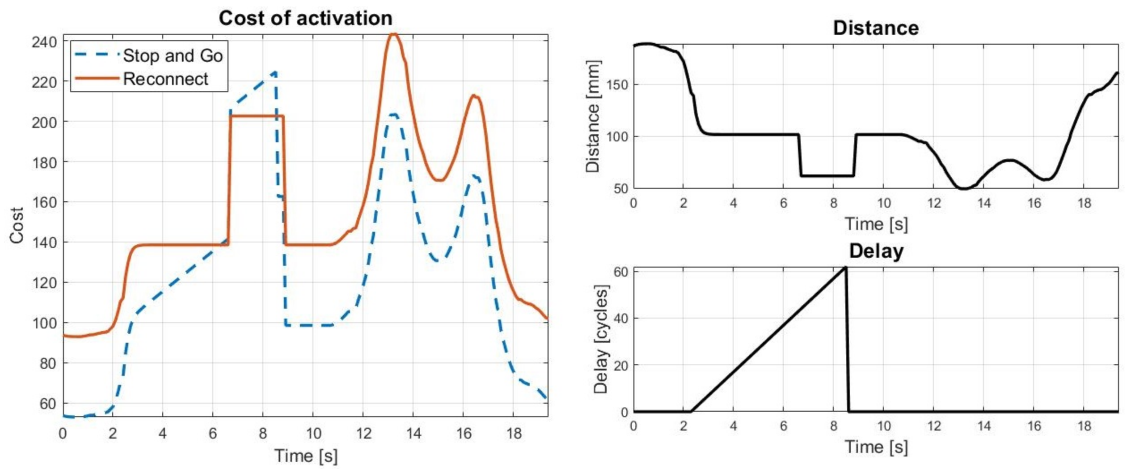


Figure 6.8: Trend of the cost of activation of the controllers in play and of the parameters used for the calculations in Case 3

Once again, an analysis of the performance of the system in this specific situation can be performed on the basis of the collected execution times, presented in Table 6.5. The time needed by the robot to complete the segment in this example is 19.1 s which can be decomposed as follows: execution of the first part of the segment till the stop is commanded, acceptable delay specified by the user, time required to compute the new plan (around 0.16 s) and its execution. Considering that the operator remains inside the common area for around 17 s and that the undisturbed execution of the whole segment takes 4.5 s, a solution in which the robot waits for the obstacle to move away instead of replanning around it would have let to a total completion time of around 21.5 s. It is clear that, if the obstruction stays in place for a long time, the best choice for the robot is to find a new way instead of waiting for the original one to be cleared.

	Undisturbed execution	**Stopped robot**	**Disturbed execution**	**Computational time**	**Total time**
Actuated reaction [s]	4.5	4.0	18.9	0.16	19.1
Discarded reaction [s]	4.5	17.0	21.5	0.002	21.5

Table 6.5: Reaction times for Customization A, Case 3, Segment 5

On a further notice, the long-term project envisions data input related to the predicted human actions. Therefore, in the future, it will be possible to augment the capabilities of switching logic and cost functions by taking into account new parameters related to these predictions. Thanks to this additional knowledge, the authors of this thesis expect a further minimization of the disruption of the productivity of the system. The robotic system was capable of guaranteeing the safety of the human operator, of the robot itself and of all the equipment around it, by avoiding any possible collision and by keeping a certain clearance from the objects of the external environment. At the same time, production times have been reduced by mitigating the sources of delay through smart and reactive robot behaviors and switching logic.

6.3 Reaction distance analysis

A further analysis has been performed by the authors in order to understand the impact of a certain set of parameters on safety and productivity of a segment in which the Replan controller is activated. In particular, the analysis is performed on Segment 5 of Customization B (Case 3), where, as previously presented, the robot's motion is disturbed by the presence of an obstacle that triggers a replanning operation in order to complete the execution. As explained in Chapter 5, the Replan controller is characterized by a distance threshold (t), used to stop the robot as it gets in the proximity of an obstacle, and a value of virtual inflation (R) of the obstacle itself, ensuring a certain clearance from the real obstacle when moving along the replanned path. The first parameter (t) can be considered as a sort of "reaction distance", since it is the distance within which the robot will react to the presence of an obstacle inside the workspace. The second parameter (R), on the other hand, represents the minimum distance acceptable for a robot trajectory that passes close to an obstacle and, for this reason, it can be considered an indicator of safety. The analysis consists in repeatedly executing the trajectory of Segment 5 with different values of reaction distance (t) and virtual inflation (R), measuring for each execution the time needed by the robot to move from the beginning (start) to the end (goal) of the segment. The execution time can be considered an indicator of productivity and the goal of the authors is to find a connection between the three mentioned parameters, in order to better understand how they influence each other. In order to have a clearer visualization of the parameters considered for this experimental campaign, Figure 6.9 can be used. The robot moves along a preplanned path, called “Original Trajectory”, obstructed by the presence of an object, indicated as “Real Obstacle”. The robot stops in front of the obstruction at a distance t, the obstacle is virtually inflated of a radius R to obtain the “Inflated Obstacle” and the trajectory recomputed in order to avoid it and reach the goal of the segment is indicated as “New Plan”. Always with reference to Figure 6.9, three cases are represented. Case 1 and Case 2 represent situations in which the same inflation radius, but different reaction distances are used. On the other hand, Case 2 and Case 3 are characterized by the same reaction distance but different inflation radii. It can be easily noticed that the shape of the replanned trajectory for the three cases is very different and an effect on the execution times is expected by the authors.

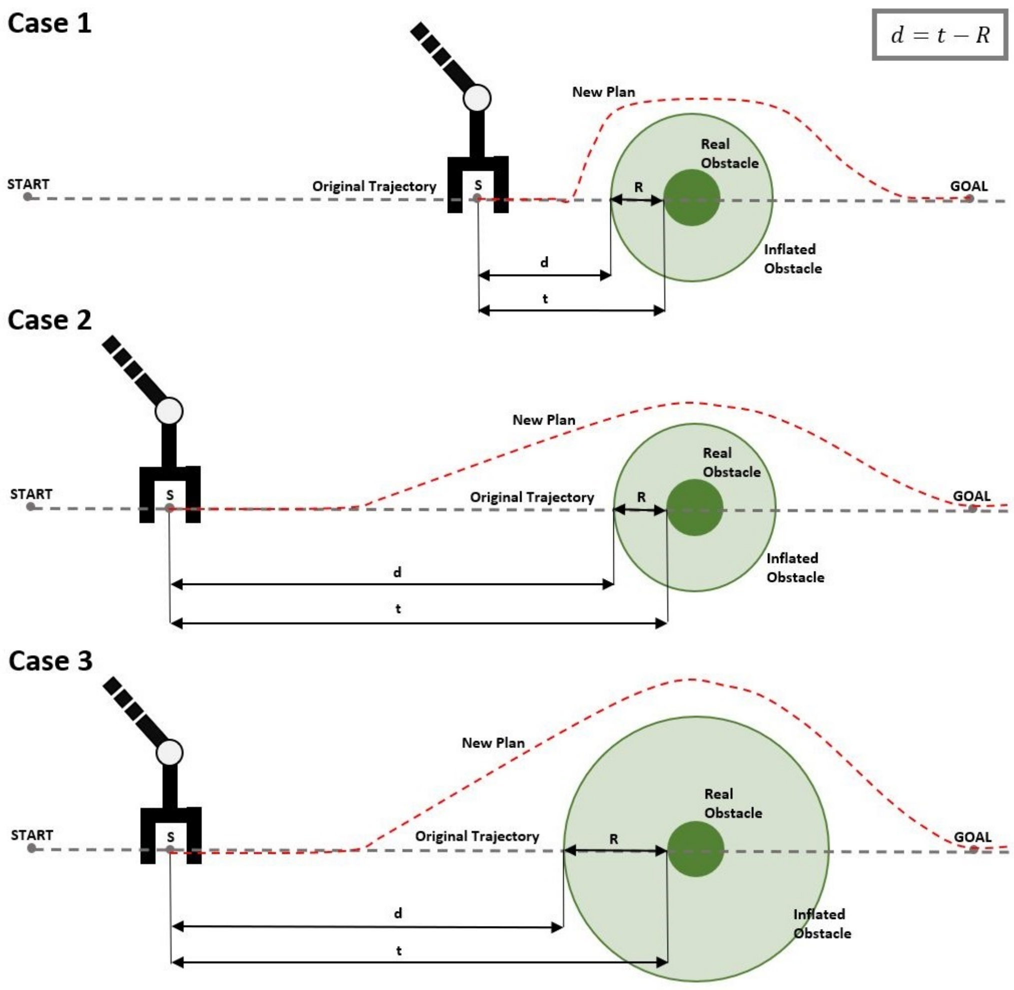


Figure 6.9: Schematic examples of the role of the different parameters

As can be inferred from Figure 6.9, the amount of virtual inflation of the obstacle is limited by the position in which the robot is stopped. In fact, an inflated obstacle overlapping with the body of the manipulator would represent a “virtual collision” for the robotic system and must therefore be avoided. In order to have a more intuitive parameter to evaluate this limit, the term d has been introduced. This new parameter is calculated with the formula reported in the top right corner of Figure 6.9 and it is directly dependent on t and R. Basically, d must always be greater than zero, since it represents the distance between the position in which the robot is stopped and the inflated obstacle.

Figure 6.10 represents the results of the experimental campaign. The plot on the left contains all the obtained curves, while the plot on the right only shows the two extreme ones together with the indication of the individual points used for their construction. The analysis has been performed by iteratively fixing a certain value of d while varying the value of virtual inflation R (and consequently the reaction distance t). For each fixed d, 9 different values of R have been considered and the obtained points have been interpolated to obtain each represented curve. Moreover, for each point (a set of d and R), multiple executions of the trajectory (Segment 5 Customization B) have been performed in order to average the intrinsic variability of the exploited planner. Given the computational burden of the simulation, the authors chose to perform 20 repetitions for each set of parameters in order to have a significative statistic sample.

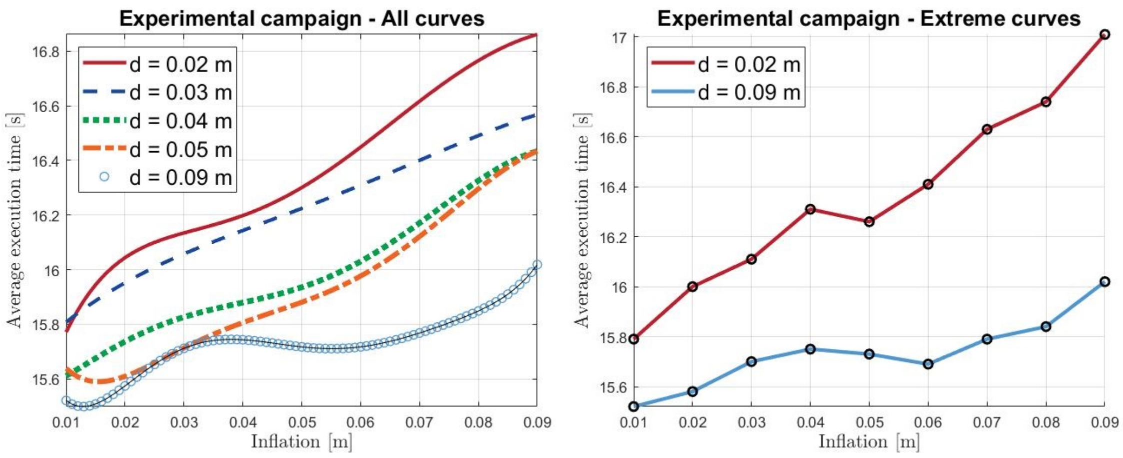


Figure 6.10: Plots of the results of the experimental campaign showing the impact of the parameters under analysis on safety and productivity

Analyzing the plots above curve by curve, it can be noticed that the general trend is to have an increasing execution time as the value of inflation grows. This is easily explained using Cases 2 and 3 in Figure 6.9 as reference: if the trajectory has to be replanned to avoid a bigger virtual obstacle (larger inflation), the new resulting path is longer and therefore more time is needed by the robot to complete it. On the other hand, considering the five plotted lines and a fixed value of inflation, it is clear that as d gets smaller, the average execution time rises. Once again, this behavior can be explained using the first two cases represented in Figure 6.9. For Case 1, the value of d is smaller and the

consequent replanned path is characterized by the presence of “sharp turns”. Since the manipulator has limits on the maximum acceleration that can be produced by its motors, these turns sensibly slow down the execution of the trajectory, leading to higher execution times. Regarding Case 2, thanks to a larger value of d, the new plan is much smoother and therefore the robot is able to execute it without slowing down and therefore achieving higher productivity.
The immediate consequence of these results is that, in order to minimize the robot execution times and therefore maximize the productivity of the system, a small inflation radius and a big reaction distance would be required. As already stated, though, the virtual inflation is an indicator of the safety of the operation and therefore its reduction would yield more discomfort and higher risks of collision between the human operator. On the other hand, a large reaction distance set for the robot would lead to a much more frequent disturbance of the trajectory, since the robot would attempt to modify its path according to the presence of obstacles that are still far from its body. Moreover, with reference to Figure 6.10, a non-linear relationship is observed. That is, the average execution time (inverse indicator of productivity) is not monotonically increasing with an increase in R. Consequently, further modelling is needed to arrive at an optimal set of the parameters under analysis able to maximize at the same time operational safety and productivity. For now, it is therefore duty of the user to select a set of parameters that represent an acceptable compromise between safety and productivity for the task at hand.

The analysis performed by the authors was successful in defining the influence of a series of parameters on the safety and productivity of the task, even if it was not possible to define an optimal set of parameters. Moreover, the authors believe that the results obtained with this experimental campaign could represent useful information for the future developments of the robotic system. The long-term project, in fact, envisions data related to predicted human trajectories as input to the control architecture. Knowing the impact that the robot’s reaction distance has on the performance of the system could, therefore, represent a powerful drive in the definition of an optimal value of the time span to be covered by the predicted data.

Chapter 7

Conclusions

To enable safe and effective human-robot collaboration in manufacturing for a smart factory, seamless integration of sensing, cognition and prediction into the robot controller is critical for real-time awareness, response and communication. Starting from these needs, this thesis proposed the implementation of a robot Proactive Adaptive Collaboration Intelligence (PACI) and switching logic within its control architecture. In particular, the end goal of this research was to give the robot the ability to optimally and dynamically adapt its motions given a priori knowledge and predefined execution plans for its assigned tasks, together with reliable sensing of human actions.

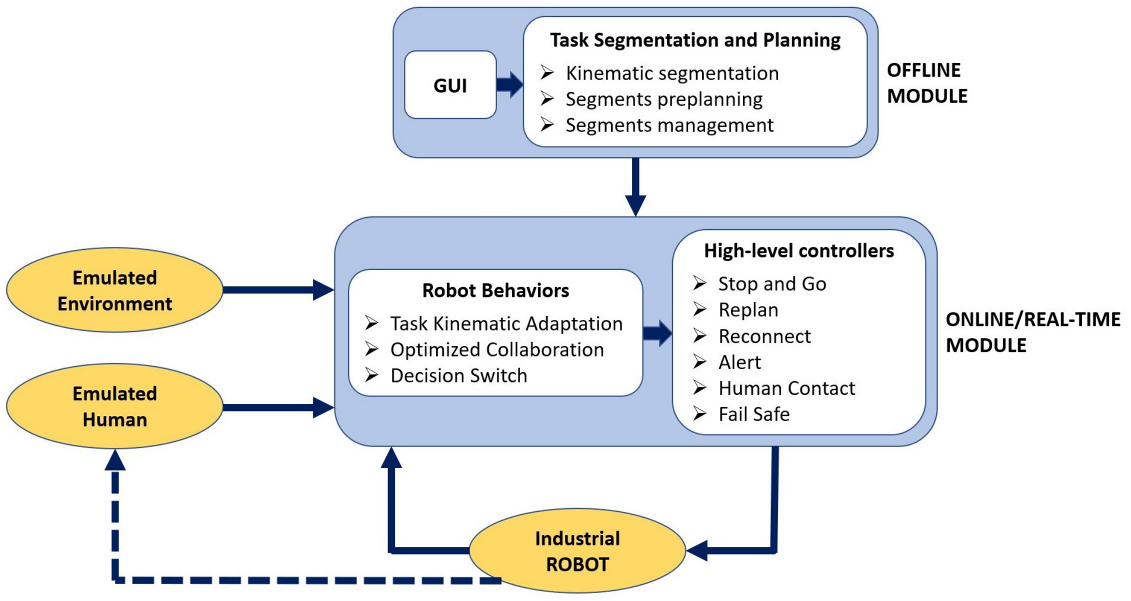


Figure 7.1: Successfully implemented control structure

The control architecture in Figure 7.1 was successfully developed on Ubuntu 18.04 Bionic Beaver inside the framework of ROS Melodic Morenia, a standard robotic platform for today's research in the field, well known for the high-level of scalability and maintainability that it provides. Moreover, open-source robotic libraries, such as MoveIt!, were exploited for their convenience in path planning and robot interface operations.

The GUI developed as part of the Offline Module was demonstrated to be very effective for the enhancement of the accessibility of the system, enabling non-experienced users to easily program the task at hand. Moreover, this tool yielded great flexibility and adaptability to the system by creating a library of preplanned scenarios, retrievable in any moment and useful both for "Pull Manufacturing" environments and for quick product customization. The Task Segmentation proposed for the same module was validated to be an additional source of flexibility for the robotic system. Thanks to this approach, each scenario can be decomposed in a series of segments making it much easier for the user to define the specific operations and allowing the independent characterization of each motion requested to the manipulator. In particular, the authors provided a list of offline planning techniques (Chapter 4), high-level controllers and robot behaviors (Chapter 5), briefly summarized below. The user is free to choose the best suited characteristics among the implemented ones, in order to optimize the human-robot interactive scenario. All the codes needed for the implementation of these capabilities have been written in C++, a high-level programming language that provides great reusability and reliability. Moreover, the structure used to write these codes augments the modularity of the system and allows easy updates and expansions for future needs.

Offline Planning Techniques

- **User Defined Algorithm**: allows the operator to create a segment by simply defining start and goal configurations of the robot together with the preferred planning algorithm;
- **Human Occupancy Volumes**: a-priori knowledge of a task can be exploited to define offline the best trajectory to be followed by the manipulator in terms of minimum "likely robot time";
- **Relevant Trajectory**: enables the user to generate a trajectory from scratch by defining a series of waypoints where the manipulator's passage is compulsory;
- **Tool Operation**: provides easy and intuitive definition of an action (open, close, wait) to be performed by the end effector in a specific position and orientation.

High-level Controllers

- **Stop and Go**: the robot stops in front of a detected obstacle and waits there for its path to be cleared before restarting the operation;
- **Replan**: the robot reacts to the presence of a detected obstacle by stopping and immediately replanning its way to the original goal of the segment;
- **Reconnect**: the robot reacts to the presence of a detected obstacle by stopping and immediately replanning its way to the first valid point of the remaining part of the original trajectory;
- **Alert**: if the robot senses the presence of an obstacle within a certain distance, it communicates the alert situation through a sound alarm;
- **Human Contact**: direct contact with the manipulator is allowed, provided that the robot is stationary in a certain position;

- **Fail Safe**: if an unexpected or hazardous situation occurs, the system immediately shuts down to give priority to the operator's safety.

Robot reactive behaviors were achieved via cost function-based switching logic activating the best suited high-level controller. The cost of activation has been defined as a function of safety (e.g., obstacle/human proximity) and productivity (e.g., induced time delays). Using a hardware-in-the-loop testbed setup, the performance of the developed control architecture subjected to different levels of human-robot interactions was validated in the University of Florida e.DO robot testbed, simulating perception of the human operator. In particular, a series of implemented behaviors were demonstrated to be very effective in triggering the correct reactions of the robot. Both safety and productivity were therefore achieved for the three human-robot interaction levels analyzed.

Further analysis, related to the Replan and Reconnect controllers, was also performed. The influence of two parameters (reaction distance and inflation radius) on the safety and productivity of the task was successfully evaluated, even if it was not possible to define their optimal values. However, knowing the impact that the robot's reaction distance has on the performance of the system could represent a powerful drive in the definition of an optimal value of the time span to be covered by the human action predicted data envisioned for the long-term project.

In general, the authors successfully developed a control structure having the desired features:

- **Flexibility**: the system can be adapted to a great variety of applications;
- **Accessibility**: an intuitive user interface allows non-expert users to easily program the tasks at hand;
- **Modularity**: the particular code structure and language used enable easy reusability and expansion of the various components of the system;
- **Safety**: the implemented high-level controllers guarantee that safety of operator, robot and equipment are always ensured;
- **Productivity**: intelligent reactive robot behaviors were demonstrated to be effective in the limitation of productivity disruption, while maintaining the required levels of safety.

A series of limitations were encountered during the development of the robotic system:

- Due to the early stages of the project at the time of the research, real sensor data was not available. The authors chose to emulate the workspace on a simulation platform in order to be able to validate the developed robotic system.
- Emulation of multiple dynamic obstacles led to unexpected crashed of the simulation platform. For this reason, the authors decided to validate the developed system by dynamically emulating only the operator's forearm and considering the rest of the body as a static obstacle.
- The absence of predicted human data at the time of the research limited the robot

behaviors to a level of intelligence that can be considered reactive, but not proactive. A framework upon which predicted trajectories can be taken as input has, on the other hand, been successfully established.
- Even if problems of this kind were not experienced, the stability of the switching logic has not yet been proven.

Finally, the authors propose a series of future developments, aimed to improve the overall system by overcoming its current limitations and moving towards greater maturity of the final product:

- Development of a dedicated planning algorithm capable of taking into account predicted trajectories of obstacles/operators in order to perform real-time replanning operations on the basis of sensor data;
- Improvement of the simulation robustness in order to sustain multiple dynamic obstacles, achievable by introducing a complete and controllable human model interacting with the manipulator;
- Expansion of the definition of the cost functions in order to take into account a greater set of parameters, such as more specific safety indicators weighted on danger levels related to different body parts and data related to the predicted motion of the operator;
- Prove of the stability of the implemented switching logic;
- Implementation of real-life sensor capabilities, together with additional and more advanced feedback methods to alert the human operator of hazardous impending situations, such as vision or haptic systems;
- Development of new offline planning techniques, high-level controllers and robot behaviors in order to further augment the adaptability of the system.

Bibliography

[1] International Federation of Robotics. *IFR Press Release*, 2019 (accessed October 18, 2019). https://ifr.org/ifr-press-releases/news/robot-investment-reaches-record-16.5-billion-usd.

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

[3] D. Kragic, J. Gustafson, H. Karaoguz, P Jensfelt, and R. Krug. Interactive, collaborative robots: Challenges and opportunities. *International Joint Conference on Artificial Intelligence*, pages 18–25, 2018.

[4] P. Beeson and B. Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. *IEEE-RAS* 15^{th} *International Conference on Humanoid Robots (Humanoids)*, 2015.

[5] S Pellegrinelli, F.L. Moro, N. Pedrocchi, L. Molinari Tosatti, and T. Tolio. A probabilistic approach to workspace sharing for human-robot cooperation in assembly tasks. *Robotics and Computer-Integrated Manufacturing*, 48:243–253, 2016.

[6] G.L. Streitmatter and G.J. Wiens. Multi-objective approach to HRC manufacturing environment. *ASME* 14^{th} *International Manufacturing Science and Engineering Conference.*

[7] ISO 8373:2012. Robots and robotic devices – Vocabulary. Standard, International Organization for Standardization, March 2012.

[8] MHI. *Industrial robots*, 2019 (accessed October 18, 2019). http://www.mhi.org/fundamentals/robots.

[9] Deloitte LLP. Made-to-order: The rise of mass personalization. In *The Deloitte Consumer Review*, 2015.

[10] PPMA. *When not to use robots*, 2019 (accessed October 18, 2019). https://www.ppma.co.uk/bara/expert-advice/robots/when-not-to-use-robots.html.

[11] ISO/TS 15066:2016. Robots and robotic devices – Collaborative robots. Standard, International Organization for Standardization, February 2016.

[12] ISO 10218-1:2011. Robots and robotic devices – Safety requirements for industrial robots – Part1: Robots. Standard, International Organization for Standardization, July 2011.

[13] ISO 10218-2:2011. Robots and robotic devices – Safety requirements for industrial robots – Part1: Robot systems and integration. Standard, International Organization for Standardization, July 2011.

[14] ISO 12100:2010. Safety of machinery – General principles for design – Risk assessment and risk reduction. Standard, International Organization for Standardization, November 2010.

[15] ISO 13850:2015. Safety of machinery – Emergency stop function – Principles for design. Standard, International Organization for Standardization, November 2015.

[16] ISO 13855:2010. Safety of machinery – Positioning of safeguards with respect to the approach speeds of parts of the human body. Standard, International Organization for Standardization, November 2010.

[17] Esben H. Østergaard. The role of cobots in industry 4.0. *White Paper*, 2017.

[18] R. Behrens, J. Saenz, C. Vogel, and N. Elkmann. Upcoming technologies and fundamentals for safeguarding all forms of human-robot collaboration. *International Conference Safety of Industrial Automated System*, 2015.

[19] A. De Luca and F. Fiacco. Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. *International Conference on Biomedical Robotics and Biomechatronics*, 2012.

[20] IEC 60204-1:2016. Safety of Machinery – electrical equipment of machines – Part 1: general requirements. Standard, International Electrotechnical Commission, October 2016.

[21] A. Vysocky and P. Novak. Human-robot collaboration in industry. *MM Science Journal*, pages 903–906, 2016.

[22] Y. Ogura, M. Fujii, K. Nishijima, H. Murakami, and M. Sonehara. Applicability of hand-guided robot for assembly-line work. 24:547–552, 2012.

[23] M. Fujii, H. Murakami, and M. Sonehara. Study on application of a human-robot collaborative system using hand-guiding in a production line. *Engineering Review*, 49(1):24–29, 2016.

[24] A. M. Jeremy. Performance metrics of speed and separation monitoring in shared workspaces. *Transactions on Automation Science and Engineering*, 10(2):405–414, 2013.

[25] J. Marvel and R. Norcross. Implementing speed and separation monitoring in collaborative robot workcells. *Robotics and Computer-Integrated Manufacturing*, 44:144–145, 2017.

[26] C. Vogel, C. Walter, and N. Elkmann. A projection-based sensor system for safe physical human-robot collaboration. *International Conference on Intelligent Robots and Systems*, 2013.

[27] S. Haddadin. Physical safety in robotics. In *Formal Modeling and Verification of Cyber-Physical Systems*, 2015.

[28] M. Colledanchise and P. Ogren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture and decision trees. *Transactions on Robotics*, 2016.

[29] L. Roveda, F. Vicentini, and L. Molinari Tosatti. Deformation-tracking impedance control in interaction with uncertain environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1992–1997, 2013.

[30] C. Liu and M. Tomizuka. Algorithmic safety measures for intelligent industrial co-robots. *IEEE International Conference on Robotics and Automation*, pages 3095–3102, 2016.

[31] N. Pedrocchi, F. Vicentini, M. Malosio, and L. Molinari Tosatti. Safe human-robot cooperation in industrial environment. *Transactions on Robotics*, 10:1–13, 2013.

[32] B. Sadrfaridpour and Y. Wang. Collaborative assembly in hybrid manufacturing cells: An integrated framework for human–robot interaction. *IEEE Transactions on Automation Science and Engineering*, 15(3):1178–1192, 2017.

[33] B. Hein, M. Hensel, and H. Worn. Intuitive and model-based on-line programming of industrial robots: A modular on-line programming environment. *International Conference on Robotics and Automation*, pages 3952–3957, 2008.

[34] B. Hein and H. Worn. Intuitive and model-based on-line programming of industrial robots: New input devices. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3064–3069, 2009.

[35] F. Ferraguti, C. Talignani Landi, C. Secchi, C. Fantuzzi, M. Nolli, and M. Pesamosca. Walk-through programming for industrial applications. *Procedia Manufacturing*, 11:31–38, 2017.

[36] A.G. Billard, S. Calinon, and R. Dillmann. Learning from humans. In *Springer Handbook of Robotics*, 2016.

[37] P. Neto and N. Mendes. Direct off-line robot programming via a common cad package. *Robotics and Autonomous Systems*, 61:896–910, 2016.

[38] C. Breazeal, K. Dautenhahn, and T. Kanda. Social robotics. In *Springer Handbook of Robotics*, 2016.

[39] S. Stork, C. StoBel, H. J. Muller, M. Wiesbeck, M. F. Zah, and A. Schubo. A neuroergonomic approach for the investigation of cognitive processes in interactive assembly environments. *The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pages 750–755, 2007.

[40] G Michalos, P. Karagiannis, S. Makris, Ö. Tokçalar, and G. Chryssolouris. Augmented reality (AR) applications for supporting human-robot interactive cooperation. *Procedia CIRP*, 41:370–375, 2016.

[41] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3, 2009.

[42] S. Chitta, I. Sucan, and S. Cousins. Moveit! *IEEE Robotics Automation Magazine*, 19(1):18–19, 2012.

[43] Moveit! *Basic concepts*, 2017 (accessed October 18, 2019). https://moveit.ros.org/documentation/concepts/.

[44] I.A. Sucan, M. Moll, and L.E. Kavraki. The open motion planning library. *IEEE Robotics Automation Magazine*, 19(4):72–82, 2012.

[45] J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation*, page 995–1001, 2000.

[46] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7), 2011.

[47] Github. *Orocos kinematics and dynamics*, 2019 (accessed October 18, 2019). https://github.com/orocos/orocos_kinematics_dynamics.

[48] J. Pan, S. Chitta, and D. Manocha. Fcl: A general purpose library for collision and proximity queries. *IEEE International Conference on Robotics and Automation*, 2012.

[49] Github. *Trajectory processing*, 2018 (accessed October 18, 2019). https://github.com/ros-planning/moveit/blob/kinetic-devel/moveit_core/trajectory_processing/src/iterative_time_parameterization.cpp.

[50] COMAU. *e.DO Robot*, 2018 (accessed October 18, 2019). https://edo.cloud/edo-robot/.