UNIVERSITÀ DEGLI STUDI
DI GENOVA

Department of Computer Science, Bioengineering, Robotics and
System Engineering

A Hierarchical Architecture for Flexible Human-Robot Collaboration

Kourosh Darvish

advisor:

Giuseppe Casalino

co-advisors:

Fulvio Mastrogiovanni

Enrico Simetti

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

April 30, 2019

Acknowledgements

First and foremost, I would like to thank my supervisor and co-supervisors. Giuseppe Casalino, who gave me a unique view of Control Theory and in specific for controlling the robots. Whom, his enthusiasm to discuss fundamental problems always was always a source of motivation for me. Fulvio Mastrogiovanni who lightened the way to do this project. His advice and guidance throughout this shared journey was an invaluable support to me. Enrico Simetti for opportunities and knowledge that has provided me during this research, for his availability and patience, during all research phases.

I would like to thank Barbara Bruno for her availability along this research, for the knowledge which shared with me.

I would like to thank all my lab mates in EmaroLab and GRAAL who helped me to do this thesis.

A special thank to all my family that has always supported and encouraged me during my journey. To my parents, to whom I owe so much of what I am and what I am becoming. I thank them with all my heart. To my sister, Leila, and my brothers, whom I have always been close with even if distant and for their unconditional love.

I thank all my friends who have been close to me and helped me to do this work.

A special thanks is for *azizam* Chiara, who believed in me and supports me always. Thank you for all your invaluable inspiration and patience which made this work possible.

To my beloved Mom and Dad,

lontani chilometri ma neanche ad un passo dal cuore

Abstract

This thesis is devoted to design a software architecture for Human-Robot Collaboration (HRC), to enhance the robots' abilities for working alongside humans. We propose FlexHRC, a hierarchical and flexible human-robot cooperation architecture specifically designed to provide collaborative robots with an extended degree of autonomy when supporting human operators in tasks with high-variability. Along with FlexHRC, we have introduced novel techniques appropriate for three interleaved levels, namely *perception*, *representation*, and *action*, each one aimed at addressing specific traits of human-robot cooperation tasks.

The Industry 4.0 paradigm emphasizes the crucial benefits that collaborative robots could bring to the whole production process. In this context, a yet unreached enabling technology is the design of robots able to deal at all levels with humans' intrinsic variability, which is not only a necessary element to a comfortable working experience for humans but also a precious capability for efficiently dealing with unexpected events. Moreover, a flexible assembly of semi-finished products is one of the expected features of next-generation shop-floor lines. Currently, such flexibility is placed on the shoulders of human operators, who are responsible for product variability, and therefore they are subject to potentially high stress levels and cognitive load when dealing with complex operations. At the same time, operations in the shop-floor are still very structured and well-defined. Collaborative robots have been designed to allow for a transition of such burden from human operators to robots that are flexible enough to support them in high-variability tasks while they unfold.

As mentioned before, FlexHRC architecture encompasses three perception, action, and representation levels. The perception level relies on wearable sensors for human action recognition and point cloud data for perceiving the object in the scene. The action level embraces four components, the robot execution manager for decoupling action planning from robot motion planning and mapping the symbolic actions to the robot controller command interface, a task Pri-

ority framework to control the robot, a differential equation solver to simulate and evaluate the robot behaviour on-the-fly, and finally a random-based method for the robot path planning. The representation level depends on AND/OR graphs for the representation of and the reasoning upon human-robot cooperation models online, a task manager to plan, adapt, and make decision for the robot behaviors, and a knowledge base in order to store the cooperation and workspace information.

We evaluated the FlexHRC functionalities according to the application desired objectives. This evaluation is accompanied with several experiments, namely collaborative screwing task, coordinated transportation of the objects in cluttered environment, collaborative table assembly task, and object positioning tasks.

The main contributions of this work are: (i) design and implementation of FlexHRC which enables the functional requirements necessary for the shop-floor assembly application such as task and team level flexibility, scalability, adaptability, and safety just a few to name, (ii) development of the task representation, which integrates a hierarchical AND/OR graph whose online behaviour is formally specified using First Order Logic, (iii) an in-the-loop simulation-based decision making process for the operations of collaborative robots coping with the variability of human operator actions, (iv) the robot adaptation to the human on-the-fly decisions and actions via human action recognition, and (v) the predictable robot behavior to the human user thanks to the task priority based control frame, the introduced path planner, and the natural and intuitive communication of the robot with the human.

Contents

List of Figures

Chapter 1

Introduction

History of humankind encountered a number of universal revolutions which have affected the human life forever, such as *Scientific Revolution* and *Industrial Revolution*. The industrial revolution has been allowing the mass and cheap production, which paved the way for the current progress of science to continue the scientific revolution Harari (2014).

The advent of the computers, allowed the examinations of many theories in *Artificial Intelligence* (AI) and Robotics Buchanan (2005). In specific, the Turing's 1950th seminal paper Turing (1950) and Shakey robot (1966-1972) Nilsson (1984) were a number of landmarks in the AI and robotics history just to name. The developments both in methodological and hardware in last decades have fostered a new revolution in the industry, namely *Industry 4.0* or *Smart Factories*; in which the humans and robots work alongside and share their workspace (*collaborative robot*). One of the objectives of Industry 4.0 is to enable the mass customization Esmaeilian *et al.* (2016); euRobotics aisbl (2014). This dissertation is meant to meet some of the requirements for these new paradigms, by introducing a new software architecture for the collaborative robot which allows the cooperation between human and robot to produce customized products.

The study of collaborative robots falls into the field of Human-Robot Interaction(HRI). As defined by Goodrich & Schultz (2007), Human-Robot Interaction (HRI) is "a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with human"; whereas in a Human-Robot Collaboration (HRC) [1] scenario, the human and robot work together to reach a common goal Bratman (1992); Grosz (1996); Hoffman & Breazeal (2004). Therefore, a human and a robot which collaborate together also interact together in a certain way; and we benefit from the literature of the interactive robots to design a collaborative robot.

[1] In this thesis, we consider the words *collaboration* and *cooperation* have identical meaning.

To benchmark a good HRI design, we should set common metrics to compare our findings with others Steinfeld *et al.* (2006). In the field of HRI, these metrics are at human, robot, and system level through the application space. Therefore, to have a good evaluation, the design of HRI should be human-centred Kiesler & Goodrich (2018); Norman (2013). In this work, we have designed and evaluated a HRC architecture based on *functional requirements* reference to the application, human, robot, and system level requirements and considerations. Do so, we study the taxonomy and features of the HRI limited to collaborative robots in industrial and shop-floor environment.

In shop-floor scenarios, the collaborative robot has a proximate interaction with the human, such that they share the space and time together. Proximate or remote interaction affects the human-robot communication as a principal part of the interaction. The two primary dimensions of the information exchange are communication medium (e.g., seeing, hearing, touch, physiological signals, etc.) and format (e.g., natural language, symbolic, etc.) Goodrich & Schultz (2007). In a shop-floor environment, some of these communication mediums and formats may prefer to others because of the peculiar working conditions or the application scenarios. To design collaborative robots, the designer should take into account the physical aspects as well as the social aspects for the long-term interaction Ajoudani *et al.* (2018); Goodrich & Schultz (2007).

In the context of mass customization, the collaborative robot is a peer for the human, with limited autonomy and authority; such that the human does not supervise or intervene to the robot's action constantly. Vernon (2014); Vernon *et al.* (2007) defines autonomy as the degree of self-determination of a system. In a human-robot interaction scenario, the autonomy rather than being an end is a medium to support productive interaction Goodrich & Schultz (2007). Authority determines the responsible for making certain decisions; and as an organizational question in human-robot interaction scenarios, it is correlated to the autonomy and the team structure. The designer of interaction should define the level of autonomy and authority for a scenario.

For some shop-floor scenarios, we may require a different level of autonomy, authority, communication medium or format, the physical or the social interactions, team organization, or the learning and adaptation of the collaborative robot or the training of the human user. This demand entails the dynamic interaction, in which interaction features changes on the basis of the time or the task Goodrich & Schultz (2007).

1.1 Motivation

According to the Industry 4.0 paradigm, manufacturing is expected to undergo an important paradigm shift involving the nature of shop-floor environments. One of the main ideas put forth in smart factories is *getting closer* to customers, increasing their satisfaction through a high degree of personalization and just in time goods delivery. The paradigm of consumer- and demand-driven manufacturing introduces the need for customised, high quality production at lower prices, and with faster delivery times Esmaeilian *et al.* (2016); euRobotics aisbl (2014). Demand-driven manufacturing and high product customisation are better managed in small-scale production, which is typical of medium-sized manufacturing enterprises (SMEs). This poses serious challenges to shop-floor operators, in so far as work stress, fatigue and eventually alienation are concerned, with repercussions also on work quality and faulty semifinished products.

Among the recommendations to reduce such drawbacks on human operators, collaborative robots have been proposed to work alongside humans to perform a series of tasks traditionally considered stressful, tiring or difficult Lenz (2011). However, while large-scale manufacturing benefits from robots already, small-scale production does not fully exploit the benefit of robot-based manufacturing euRobotics aisbl *et al.* (2017); Kock *et al.* (2011). Consumer- and demand-driven manufacturing requires robots characterised by high flexibility, fast reconfiguration and installation, as well as low maintenance costs. Collaborative robots are expected to meet such demands, decrease manufacturing costs, and therefore increase products variability and customisation Esmaeilian *et al.* (2016); euRobotics aisbl (2014); Kock *et al.* (2011). In fact, they are considered key enabling factors to automate small-scale production when operations to be carried out are highly dynamic and partially unstructured International Federation of Robotics (2018).

Recently, many authors argued that consumer- and demand-driven manufacturing can benefit from the introduction of human-robot cooperation (HRC) processes. Clearly, this proposal implies a number of challenges related to human-robot interaction both at the physical and the cognitive levels of the cooperation DeSantis *et al.* (2008); Hayes & Scassellati (2016); Lemaignan *et al.* (2017), which depend also on their type Helms *et al.* (2002). Beside basic safety considerations, which are a necessary prerequisite Kuehn & Haddadin (2017), a number of key issues must be taken into account: sensing and human activity recognition Pedrocchi *et al.* (2013), definition of suitable cooperation models to reach certain goals Johannsmeier & Haddadin (2017); Kock *et al.* (2011); Shah *et al.* (2011), robot action planning and execution in the presence of humans Lemaignan *et al.* (2017), and the effect of robot's predictable behaviour on the operator well-being and performance Bortot *et al.* (2013), just to name a few.

Among the possible use cases where human-robot cooperation can be partic-

ularly relevant, we consider cooperative assembly as a motivating scenario. If we focus on assemblage tasks, typically involving a small number of semi-finished pieces, a number of difficult-to-model situations arise: the order of assemblage operations is often not strict, i.e., different sequences are possible and equally legitimate as far as the final result is concerned; an operator and a robot engage in a sort of turn-taking process, where the robot is expected to assist and adapt to human actions at run-time; for a fruitful cooperation to occur, the operator and the robot must understand each other actions and intentions.

1.2 Objectives and Innovations

The customised production advocated by consumer- and demand-driven manufacturing still relies on human operator cognitive capabilities since collaborative robots are largely unable to *efficiently* manage inter-tasks or intra-task variations International Federation of Robotics (2018). The ability of human operators to decompose complex tasks into simpler operations (e.g., assembling furniture parts to obtain other semifinished parts to be used later), or to *naturally* manage small variations (e.g., assembling furniture with parts of different size, like tables with differing flat top size or leg length), still poses a significant challenge for collaborative robots Garca *et al.* (2013).

To do so, in a human-robot cooperation scenario, the human operators and robots purposely interact to achieve a common objective, and they do so working together in a shared workspace. The design of collaborative robots should adhere to a number of *human-centric* principles so that cooperation can be effective, efficient, and natural. Human-centric design enforces such factors as the *explainablity* of robot decisions, the *usability* of robot interfaces (in a broad sense), the *awareness* of the cooperation process, and a fair *workload*, as well as *safety* requirements for human operators Adams (2005); Steinfeld *et al.* (2006).

These considerations can be synthesized in a number of *functional requirements* focusing on improving the operator's *working experience* Meneweger *et al.* (2015), effectiveness, efficiency, and applicability of the collaborative robot.

F_1 [*Task Level Flexibility*] The human and robot cooperate on similar tasks, such as the assembly of several table types, without requiring different representations. The architecture of the collaborative robot should handle these variations autonomously; and doing so by abstracting the *structure* of tasks from perceptual variabilities and uncertainties Darvish *et al.* (2018a); De-Santis *et al.* (2008); Kock *et al.* (2011).

F_2 [*Team Level Flexibility*] Operators should not be forced to follow a strict, predefined sequence of operations, but should be allowed to decide what

actions to perform *on the fly*, subject to their adherence to the overall cooperation goals. As a consequence, robots should trade-off between providing operators with optimal suggestions about next actions to perform and reacting appropriately when operators do not follow such instructions.

F_3 [*Intelligibility & Intuitiveness*] While the cooperation process unfolds, operators should be capable of intuitively understanding robot actions and *intentions*, and this may be achieved at a symbolic, *linguistic* level of communication. Therefore, collaborative robots should be able to decouple action planning (whose results are meaningful for operators) from motion planning and control, the latter hiding low-level complexities associated with robot motions also when the workspace is partially unknown.

F_4 [*Naturalness*] It has been demonstrated that a natural and efficient cooperation is possible only by a reasoned trade-off between the cooperation objective (e.g., the assemblage of a semifinished product), which must be achieved as a functional requirement, and the human or robot degrees of autonomy when the task is only partially well-defined (e.g., such assemblage can be done using different action sequences), which can be somewhat enforced or relaxed on a context-dependent basis Ferland *et al.* (2013); Goodrich & Schultz (2007).

F_5 [*Adaptability*] In order for a robot to detect and classify meaningful actions carried out by an operator, it should not be necessary that different operators undergo a specialised action modelling and adaptation process, i.e., the robot should adapt to them without requiring an operator-specific calibration process. In this sense, collaborative robots should be able to reactively adapt to human operator actions while retaining the capability of planning specific action sequences to meet the cooperation objective Argall *et al.* (2009); Darvish *et al.* (2018b); Valli (2008).

F_6 [*Decision Making*] The collaborative robot should plan for its future behaviour to reach the cooperation goal, estimate the execution of its planned future behaviours online, exploits such estimates to take decisions as far as cooperation is concerned, and allocates tasks to either humans or robots. Such a capability enhances the robot dexterousness in partially structured and uncertain environments.

F_7 [*Transparency*] Operators should not be required to limit their freedom as far as motions are concerned, e.g., being forced to stay in front of a collaborative robot all the time, to have their actions duly monitored during the cooperation process.

F_8 [*Safety*] Since it has been shown that the *effectiveness* and the overall performance of human operators is positively correlated with robot motion predictability Bortot *et al.* (2013), collaborative robots should prevent psychological discomfort, stress, and a high induced cognitive load on human operators, and furthermore ensures the physical safety of the human De-Santis *et al.* (2008); Kock *et al.* (2011); Lasota *et al.* (2017).

F_9 [*Scalability*] Collaborative robots should exhibit decision making capabilities grounded on hierarchical task representations and enforcing a scalable definition of action sequences able to map high-level complex tasks to low-level, simple robot operations. In a scalable system, the user combines simple tasks to perform a complex scenario Garca *et al.* (2013), which results in reducing the setup and reconfiguration time for different scenarios and increasing the efficiency.

In the literature, different approaches have been proposed to model HRC processes *as a whole*. While some of them are aimed at introducing aspects of social interaction Crandall *et al.* (2018); Lemaignan *et al.* (2017); Pineau *et al.* (2003); Shah *et al.* (2011), others focus explicitly on HRC processes for collaborative manipulation or assembly Capitanelli *et al.* (2018); Darvish *et al.* (2018b); Hawkins *et al.* (2014); Johannsmeier & Haddadin (2017); Levine & Williams (2014); Michalos *et al.* (2014); Toussaint *et al.* (2016). Among them, it is possible to discriminate between those architectures allowing for an online adaptation of robot action sequences on the basis of human operator actions Caccavale & Finzi (2017); Darvish *et al.* (2018b); Hawkins *et al.* (2014); Levine & Williams (2014); Sebastiani *et al.* (2017), and approaches limiting plan adaptation to time constraints Johannsmeier & Haddadin (2017); Karpas *et al.* (2015); Shah *et al.* (2011). Proactive planning or online decision making is considered in the frameworks described in Capitanelli *et al.* (2018); Darvish *et al.* (2018a); Lemaignan *et al.* (2017); Levine & Williams (2014); Müller *et al.* (2007); Nikolaidis *et al.* (2017), whereas the others perform it offline. Different task representation and planning methods have been adopted to model the cooperation process, namely Markov Decision Processes (MDPs) Claes & Tuyls (2014); Crandall *et al.* (2018); Toussaint *et al.* (2016), Task Networks (TNs) Lemaignan *et al.* (2017); Levine & Williams (2014); Shah *et al.* (2009), satisfiability-based planners Capitanelli *et al.* (2018), and AND/OR graphs Darvish *et al.* (2018b); Hawkins *et al.* (2014); Johannsmeier & Haddadin (2017).

In this dissertation, we present an integrated architecture for flexible HRC processes, which we refer to as FlexHRC, to address a number of important functional requirements as identified above. FlexHRC enables online human-robot decision making and adaptation, flexible execution of HRC tasks, a scalable representation of such tasks enforcing modularity and reuse. FlexHRC

can adapt the behaviour of collaborative robots to human operator actions reactively, while proactively taking decisions aimed at meeting cooperation goals. The innovations of the current work at functional levels are:

- *Human-robot cooperation level.* A hybrid reactive-deliberative architecture for online, flexible and scalable HRC processes is proposed, characterised by proactive decision making and reactive adaptation to the peculiar cooperation state, i.e., a perceived sequence of human operator actions, also when certain robot operations cannot be successfully executed. FlexHRC supports all the functional requirements defined above.

- *Task representation level.* An integrated hierarchical representation of HRC processes employing First Order Logic (FOL) and AND/OR graphs to model static and dynamic aspects of HRC-related tasks. The proposed method enables the necessary scalability by introducing the hierarchical task representation, intuitiveness using the linguistic communication at this level with the human, and optimality of the action sequences for the collaborative robot at representation level to deal with F_9, F_3, and F_4. The Task Manager enables the F_1, F_2, F_5, F_6 functional requirements by integrating the representation, perception, and action levels, and predicting the future robot behaviour online.

- *Action Level.* Although robot operations are well-defined in terms of motion trajectories and, above all, intended effects, reactive behaviors allow for dealing with partially unknown or dynamic workspaces, e.g., to perform obstacle avoidance, without the need for whole trajectory re-planning Simetti & Casalino (2016); Srivastava *et al.* (2014). FlexHRC control and path planning framework decouple human-robot action planning from robot motion planning and control Simetti & Casalino (2016), and avoid the obstacles Karaman & Frazzoli (2011) therefore addressing F_3 and F_8.

- *Perception Level.* FlexHRC benefits from the human action recognition and workspace object perception using *wearable sensors* and *RGB-d data*, which do not pose any constraint on operator motions, to address F_7; it exploits statistical techniques for action modeling Bruno *et al.* (2013) to take F_5 into account; and it integrates the different perceptive information at symbolic level to support the F_1, F_2, F_4, F_6, F_8.

1.3 Dissertation Outline

The dissertation is organized as follows. Chapter 2 describes relevant state-of-the-art approaches in HRC processes and introduces the main traits of FlexHRC.

We describe the representation level, including the *Task Representation*, *Task Manager*, and *Knowledge Base* in Chapter 3. In Chapter 4, we explain the *Human Action Recognition* and *Object and Scene Perception*, where we integrate the perceptive information at the symbolic level. The portray of the action level, including the *Controller*, *Path Planning*, and *Robot Execution Manager* is presented in Chapter 5. In Chapter 6, we describe the experimental scenarios and discuss relevant results. Conclusions follow.

Chapter 2

Software Architecture for Flexible Human-Robot Cooperation

Summary

In this Chapter, we portray the software architecture for the flexible human-robot cooperation which responds to the functional requirements described in Chapter 1. First, we present the state of the art for the collaborative robots at architecture, representation, action, and perception level. Then, we analyze the literature according to their methodologies and approaches for answering to the introduced functional requirements. Finally, we describe the FlexHRC architecture, and the three levels of the architecture, namely representation, action, and perception level.

2.1 State of the Art

During the past few years, human-robot interaction gained much attention in the research literature. Whilst approaches focused on cooperation consider aspects related to natural interaction with robots, e.g., targeting human-robot coordination in joint action Huber *et al.* (2013); Sebanz *et al.* (2006); Valdesolo *et al.* (2010), this analysis focuses on the human-robot cooperation process from the perspective of the functional specifications discussed in Chapter 1.

The problem of allowing humans and robots to perform open-ended cooperation by means of coordinated activity did not receive adequate attention so far. An approach highlighting the challenge is presented in Shah *et al.* (2011), in which an execution planning and monitoring module adopts two teamwork

modes, i.e. when humans and robots are equal partners and when humans act as leaders. On the one hand, a reference shared plan is generated off line, and actions are allocated to a human or a robot according to their capabilities. On the other hand, coordination is achieved by an explicit step-by-step, speech-based, human to robot communication, which makes the user experience cumbersome and unnatural in most cases.

The ability of robots to mediate between high-level planning and low-level reactive behaviours has been subject of huge debates in the past three decades. When it comes to human-robot cooperation, the need arises to balance the requirements of reaching a well-defined goal (e.g., a joint assembly) and providing human co-workers with as much freedom as possible. A number of conceptual elements for joint and coordinated operations are identified in Vesper *et al.* (2010). The authors propose a *minimalistic* architecture to deal with aspects related to agents cooperation. In particular, a formalism to define goals, tasks and their representation, as well as the required monitoring and prediction processes, is described. The work discussed in Lemaignan *et al.* (2017) significantly extends the notions introduced in Vesper *et al.* (2010) to focus on *social* human-robot interaction aspects. The architecture makes an explicit use of *symbol anchoring* to reason about human actions and cooperation states. An approach sharing some similarities with FlexHRC is described in Johannsmeier & Haddadin (2017). As in the proposed approach, AND/OR graphs are used to sequence actions for the cooperation process. However, unlike FlexHRC, action sequences cannot be switched at runtime, but are determined off line in order to optimize graph-based metrics. As a matter of fact, the possibility of multiple cooperation models is provided for, although off line: optimal paths on the AND/OR graph are converted to *fixed* action sequences, and then executed without any possible variation. In a similar way, multiple cooperation models are considered in Hawkins *et al.* (2014), where an AND/OR graph is converted to a nondeterministic finite state machine for representation, and later to a probabilistic graphical model for predicting and monitoring human actions, as well as their timing. An architecture for flexible execution of the concurrent tasks in the context of HRI is adopted in Caccavale & Finzi (2017); Caccavale *et al.* (2016). The architecture embeds a hierarchical planning approach, i.e. Hierarchical Task Networks (HTNs) Erol *et al.* (1995), to progress toward the interaction goal using task-oriented and stimuli-driven attentional processes. Likewise, Sebastiani *et al.* (2017) presented an offline conditional plan generation for HRI scenarios. Online, it clarifies the plan by negotiating with the human. Karpas *et al.* (2015) extended online executive to unify the intention recognition and plan adaptation for temporally flexible plans. It takes as input a temporal plan network with uncertainties, handles uncontrollable action duration by enforcing strong temporal controllability, and in case of exceeding the limits starts negotiating with the humans to relax the temporal constraints.

Task representation and action planning for HRC scenarios are traditionally considered challenging research efforts, specifically when the task to be carried out is characterised by high-variability, and human operators are allowed to autonomously decide how to proceed. Approaches in the literature focus on specific aspects of the cooperation process. A general approach for encoding such complex scenarios is a hierarchical representation of the abstract behaviours or actions. The symbolic level representation of the behaviours allows the learning representation of general tasks and automatic generation of the behavioural system Nicolescu & Matarić (2002).

Representation level. The work described in Darvish *et al.* (2018b); Levine & Williams (2014) is aimed at recognising the peculiar sequence of actions performed by human operators online, and to provide robots with methods to reactively adapt to those actions. The recognition of action sequences carried out by human operators assumes actions to be completed *before* they can be properly recognised, which is expected to introduce possibly unacceptable delays in the cooperation process, and therefore jeopardise efficiency and naturalness. A slightly different approach, pursued in Hawkins *et al.* (2014), employs methods to *predict* rather than recognise human operator actions, thereby trading-off recognition performance and prediction accuracy. However, in case of misclassifications, the overall cooperation process can be negatively affected, above all in so far as effectiveness is concerned. Rather than providing human operators with high freedom in how to carry out a given cooperative task, the approach proposed in Nikolaidis *et al.* (2017) envisions a sort of *dyadic*, mutual adaptation between human operators and robots, where robots act as leaders guiding human operators towards an efficient task execution strategy. It is no mystery that such an approach can lead to a lack of naturalness in the cooperation process. While human action prediction, recognition, and adaptation are forms of implicit human-robot *communication*, an explicit, speech-based communication is adopted instead in Lemaignan *et al.* (2017); Shah *et al.* (2011). Aspects related to naturalness are enforced using speech-based communication only in principle. In fact, this is done at the detriment of effectiveness and efficiency, since speech recognition can yield to dramatically poor results in industrial scenarios. MDPs have been used in Claes & Tuyls (2014); Crandall *et al.* (2018) to enforce adaptation to human operator behaviours online. Such an approach leads to purely reactive approaches, which are considered indeed natural, but neither effective nor efficient. Classical planning integrated into a hierarchical reactive/deliberative architecture has been proposed in Capitanelli *et al.* (2018) as a trade-off between the two contrasting requirements of efficiency and naturalness. While such a trade-off has been demonstrated in a specific domain, there is no evidence that such a balance can be easily replicated in other HRC scenarios.

It is noteworthy that naturalness in HRC scenarios should not be treated

independently from the perception of *natural* robot behaviours, i.e., human-like morphology and robot motions. However, when goal-oriented cooperation tasks are involved, it is necessary to pursue a trade-off between optimality in robot motions and action sequences, and human-like behaviours Bortot *et al.* (2013), so that human operators can understand, explain and predict robot behaviours Adams (2005); Steinfeld *et al.* (2006). Among different task representation methods, only those approaches based on AND/OR graphs and TNs explicitly consider explainability and predictability at the representation level.

In order to model action planning in HRC scenarios, hierarchical approaches have been used in Hayes & Scassellati (2016); Lemaignan *et al.* (2017). As mentioned above, a hierarchical representation allows for the modelling of complex cooperation tasks efficiently, and it enforces modularity and scalability in the representation. With the sole exception of the use of HTNs in Lemaignan *et al.* (2017), hierarchical approaches do not explicitly consider the interplay between efficiency and naturalness in the cooperation process.

TNs and approaches based on classical planning are characterised by a natural description layer based on FOL Capitanelli *et al.* (2018); Cashmore *et al.* (2015); Lemaignan *et al.* (2017), which is expected to enforce effectiveness since it constitutes a close-to-human language used to associate semantics to each robot action Russell & Norvig (2010). However, traversal and planning algorithms cannot guarantee explainable nor predictable robot action sequences, unless severe constraints are posed. FOL-based MDPs Boutilier *et al.* (2001); Yoon *et al.* (2002), also referred to as *relational* MDPs, are a tentative solution to trade-off these aspects, and indeed have been adopted to model cooperative assembly tasks in Toussaint *et al.* (2016). In a similar perspective, AND/OR graphs naturally yield explainable results Luger (2009), and are amenable to be represented using FOL for the purpose of cooperative task representation.

Action level. One of the main challenges to address in HRC scenarios is deciding how to allocate actions, either to the human operator, the robot, or in principle to both Goodrich & Schultz (2007). Action allocation is a necessary modelling choice relating task representation to task and motion planning combined together, and obviously, it has a great impact on effectiveness and efficiency. When a human operator is given the freedom of autonomously deciding how to accomplish a task, action allocation cannot be defined beforehand and must be resolved online to be effective Arai *et al.* (2002); Chen *et al.* (2014); Miyata *et al.* (2002). In order to schedule resources and to allocate tasks to human operators or robots, a multi-objective optimisation problem is typically formulated Chen *et al.* (2014); Tsarouchi *et al.* (2017). In particular, the allocation problem is addressed from the perspective of a *utility* measure, which estimates the overall team performance as a function of such parameters as *quality* of an action result, its *cost*, its associated *cognitive load*, and the *resources* needed for its completion Gerkey

& Matarić (2004). Such optimisation problem is then resolved online for dynamic task allocation Chen *et al.* (2014); R. A. Giele *et al.* (2015); Shah *et al.* (2009); Wilcox *et al.* (2012), while other approaches are limited to offline solutions Johannsmeier & Haddadin (2017); Tsarouchi *et al.* (2017). As described in Darvish *et al.* (2018a), resolving action allocation online can be done only if the relevant parameters of the employed utility measure are either estimated beforehand or it is safe to assume they can be quantified while the cooperation process unfolds. For example, in the approach described in Tsarouchi *et al.* (2017), expected action completion times are estimated offline, whereas robot workspace reachability is assumed to be a function of the Euclidean distance between current and goal robot's end-effector poses. The approach discussed in Darvish *et al.* (2018a) proposes a fast in-the-loop simulation to estimate relevant utility parameters online, whereas in Müller *et al.* (2007) it is suggested that simulations should be done *a priori*. It is noteworthy that offline task allocation is limited as far as the reliability is concerned, due to uncertainties and changes in the workspace and unpredictable human decisions.

In the literature, among different task planning approaches have been proposed for HRC scenarios; the approaches described in Johannsmeier & Haddadin (2017); Lemaignan *et al.* (2017) generate different plans offline, and optimize a utility function to find the optimal sequence of actions and to perform task allocation. During online execution, humans and robots are constrained to follow the optimal plan while executing the cooperation task. Major issues arise when the cooperation process cannot be described in its entirety, or some relevant features of the environment are uncertain, or cannot be properly perceived by the robot or represented. In all these cases finding an optimal sequence of actions for allocated tasks cannot be but sub-optimal. Other approaches Darvish *et al.* (2018b); Hawkins *et al.* (2014); Levine & Williams (2014) allow for tuning the cooperation process as it unfolds, by enabling human operators only to decide how to progress, while a robot reactively adapts to human actions.

It is noteworthy that the integration of task representation, online task planning, task allocation, and motion planning is expected to enhance the robustness to failures and the overall HRC process efficiency Darvish *et al.* (2018a). Furthermore, it has been shown in Darvish *et al.* (2018a) that in-the-loop robot motion predictions are beneficial to a natural interaction, as opposed to the prediction of human operator motions Hawkins *et al.* (2014); Koppula & Saxena (2013); Mainprice *et al.* (2015).

The development of sensing and control architectures able to integrate and coordinate action planning with motion planning and control is an active research topic. However, the challenge is typically addressed to deal with cases where planning cannot be guaranteed to be *monotone*, i.e., when sensory information must be used to validate the plan during execution Agrawal *et al.* (2016). Its

application to human-robot cooperation tasks has not been fully addressed in the literature. An approach in that direction is described in Toussaint *et al.* (2016), where an integrated approach to Monte Carlo based action planning and trajectory planning via Programming by Demonstration is adopted in a scenario of toolbox assembly. Concurrent activities are formalized using a Markov decision process, which determines when to initiate and terminate each human or robot action. A multi-objective optimization approach for solving the subtask allocation for the project scheduling problem of HRC is introduced in Chen *et al.* (2014), where an evolutionary algorithm takes care of real-time subtask allocation. The proposed framework considers both parallel and sequential features and logic restrictions as well as given objectives for human and robot action time and cost, idle time, etc.

Perception level. Finally, a few approaches consider the issue of allowing human operators to retain a certain freedom of motion or action when interacting with a robot, but at the price of introducing a few assumptions in the process Bauer *et al.* (2008); Cartmill *et al.* (2011). A Bayesian framework is used in Huber *et al.* (2013) to track a human hand position in the workspace with the aim of predicting an action's time-to-completion. The hand must be clearly visible for the estimate to be accurate, which limits certain motions. The opposite approach is adopted in Shah *et al.* (2011), where an extended freedom of motion is obtained resorting to speech-based communication to indicate performed actions to the robot, as well as action start and end times. The obvious drawback of this approach relies on the fact that such a communication act must be voluntary, and therefore human stress and fatigue may jeopardize the will to do it. A more comprehensive approach is described in Lemaignan *et al.* (2017), which integrates human body position (determined by an external sensory system, e.g., motion capture), deictic gestures, gaze and verbal communication to determine a number of human actions. A gesture lexicon for giving commands to other partners in industrial environments is studied in Gleeson *et al.* (2013). The work investigates the gestures commonly performed by humans to communicate with each other about part acquisition, manipulation, and operation tasks. In the experimental evaluation, such gestures were replicated by an industrial robot and the understanding of human operators was measured. Both solutions rely on an *external* system for human activity recognition, which may be of difficult deployment in a shop-floor environment, and occlusions may occur nonetheless.

From this focused analysis, it emerges that although a number of approaches have been discussed, which take the identified functional specifications into account, they do so only partially. FlexHRC attempts to provide a holistic and integrated solution to these heterogeneous challenges.

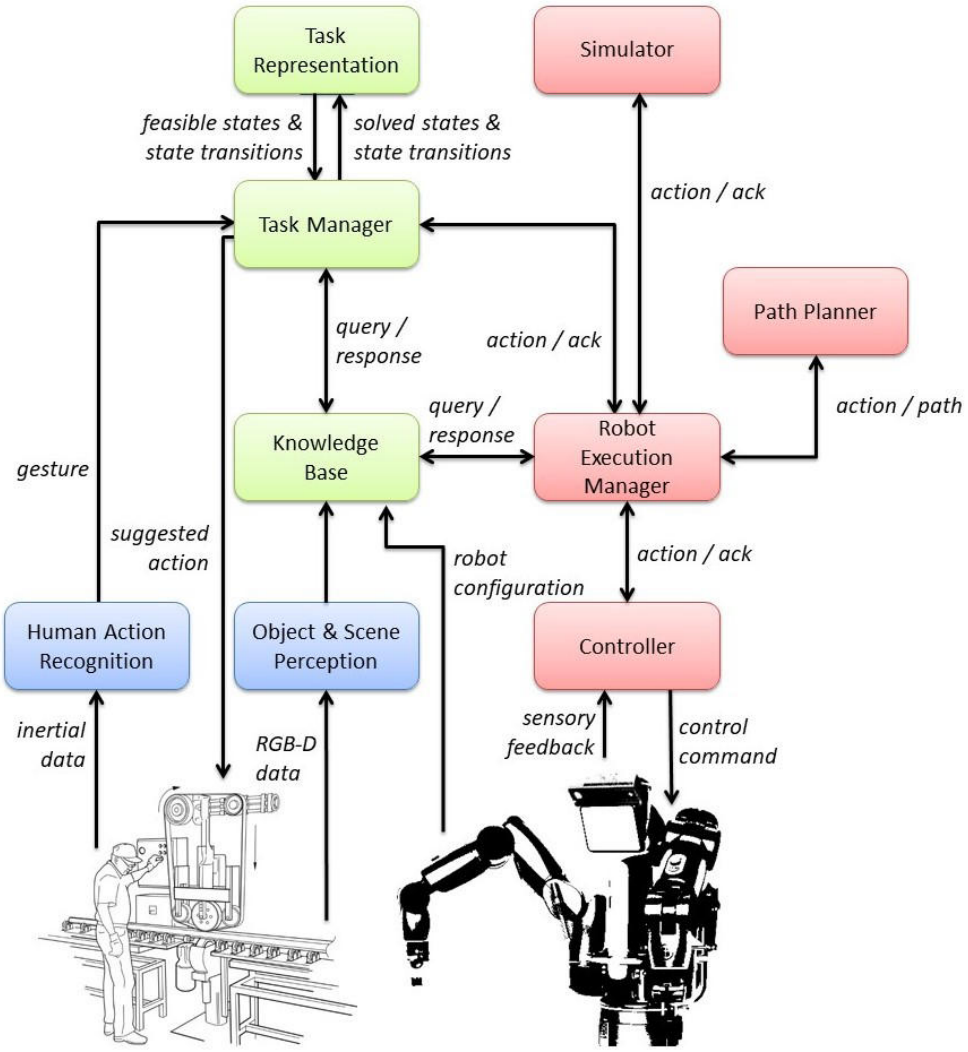


Figure 2.1: FlexHRC's architecture: in *green* the representation level, in *blue* the perception level, in *red* the action level.

2.2 FlexHRC System Architecture

FlexHRC is based on a distributed hybrid reactive-deliberative architecture, which is conceptually described in Figure 2.1. FlexHRC is a cognitive architecture as it holds all the attributes of a cognitive architecture including autonomy, perception, learning, anticipation, action, and adaptation Vernon (2014). Some of these attributes are emphasized highly as the core of FlexHRC, while others are less.

The architecture of FlexHRC is organised in three levels, namely the representation level (in *green* in Figure 2.1), the perception level (in *blue*), and the action level (in *red*). The representation level maintains all the relevant information related to cooperative tasks via the *Task Representation* module, and to the shared workspace in the *Knowledge Base* module. The representation level performs action planning and decision making for task execution, as well as for action allocation via the *Task Manager* module. The perception level acquires information about the workspace in terms of objects and other entities therein using *Object and Scene Perception*, and it is responsible for detecting and classifying actions performed by human operators as done in the *Human Action Recognition* module. Starting from raw data, the perception level updates the representation level with relevant semantic information about objects and human operator actions. On the basis of such information, the action level serialises the execution of robot actions (via the *Robot Execution Manager* module), plans a path for the robot end-effector, performs in-the-loop robot action simulations in the *Simulator* module, and controls online all robot motions using the *Controller* module.

2.2.1 Representation level

The *Task Representation* module maintains knowledge about all possible states and state transitions modelling cooperative tasks. The module also defines how the HRC process can progress by providing suggestions to the human operator or the robot about what to do next. As anticipated above, we apply AND/OR graphs to represent cooperative tasks, as better described in the next Section Darvish *et al.* (2018b); de Mello & Sanderson (1990). The module receives in input the current cooperation status from the *Task Manager*, and therefore provides it with next action suggestions. The *Task Manager* module maps cooperation states as represented in the AND/OR graph structure to either human or robot actions, and informs the human for such actions through the robot display. The module plans for suggested states receiving appropriate information from the *Task Representation* module, it grounds action parameters to actual values, and it performs action assignments on the basis of incoming perceptual information Darvish *et al.* (2018a,b). The *Knowledge Base* module maintains and explicitly represents the cooperation state and workspace-related perceptual information using custom data structures Darvish *et al.* (2018a).

2.2.2 Perception level

The *Human Action Recognition* module provides FlexHRC with information about actions performed by human operators. The module models action templates using Gaussian Mixture Models and Gaussian Mixture Regression starting

from a relevant dataset of inertial data obtained using operator-worn sensors, and applies online statistical distance metrics to determine the likelihood of the occurrence of an action. It applies a pattern matching algorithm to detect and recognise meaningful actions as performed by human operators. To do so, the module receives an inertial data stream, and informs *Task Manager* about the detected and recognised human operator actions Bruno *et al.* (2014); Darvish *et al.* (2018b). The *Object and Scene Perception* module provides information about objects in the robot workspace and models them using a set of primitive shapes characterised by their geometrical characteristics. The module simply applies Euclidean distance to cluster a point cloud originating from an RGB-D sensor located on the robot body, and it applies the Random Sample Consensus (RANSAC) algorithm to model those clusters as primitive shapes, and to determine their relevant features. Additionally, Principal Component Analysis (PCA) is used to find complementary object features for manipulation purposes. Recognised objects and their features are maintained in the *Knowledge Base* module Buoncompagni & Mastrogiovanni (2015); Fischler & Bolles (1981); Wold *et al.* (1987).

2.2.3 Action level

The *Robot Execution Manager* maps action commands issued by the representation level to simple commands fed to the *Simulator* or the *Controller* module. The module receives workspace-related information from the *Knowledge Base* for manipulation purposes. The *Path Planner* plan a path for the manipulator robot end-effector such that does collide with the obstacles of the workspace. The in-the-loop *Simulator* module simulates closed-loop robot kinematics and control to predict the robot behaviour online. It receives reference information from the *Robot Execution Manager*, and provides it with the results of the simulation (e.g., failure/success of a given command, action execution time, final robot pose, or estimated energy consumption) Darvish *et al.* (2018a). The *Controller* module controls robot motions using a Task Priority framework at the kinematic level, taking into account multiple control objectives, both of *equality* and *inequality* type, and their priorities. The reference robot velocity is found solving a sequence of prioritized optimization problems Simetti & Casalino (2016); Simetti *et al.* (2018).

Chapter 3

Human-Robot Cooperation at Representation level

Summary

In this chapter, we describe the theory and implementation of FlexHRC at the representation level. First, we explain the *Task Representation* module and in specific the AND/OR graph. We introduce the elements and theory of the *Propositional Logic* AND/OR graph. Later, we extend the AND/OR graph notion to a First-Order-Logic *Task Representation* method, and found the hierarchical AND/OR graph *Task Representation*. In Section 3.2, we establish the theory and implementation of *Task Manager* which includes proactive decision making and reactive adaptation. Finally, we describe the customized *Knowledge Base* developed for FlexHRC.

3.1 Task Representation Model

In order to formalise and to explicitly represent in FlexHRC the human-robot cooperation process we adopt AND/OR graphs. An AND/OR graph allows for an easy representation of problems to solve or procedures to follow, which can be clearly decomposed in subproblems (as parts of the graph), as well as the logic relationships among those subproblems (the interconnectivity of the graph). Since the root node conventionally represents the solution, solving the problem means traversing the graph from leaf nodes to the root note according to the graph structure. Therefore, AND/OR graphs can take limited forms of non-determinism or uncertainty into account de Mello & Sanderson (1990); Luger (2009); Russell & Norvig (2010) via the availability of various branches possibly leading to the solution. On the basis of previous work Darvish *et al.* (2018a,b),

Table 3.1: Definition of symbols related to propositional logic AND/OR graphs

Symbol	Definition
n	A node in the AND/OR graph
N	The set of all nodes in the AND/OR graph
h	An hyper-arc in the AND/OR graph
H	The set of all hyper-arcs in the AND/OR graph
$G(N, H)$	An AND/OR graph composed of the $\|N\|$ nodes and $\|H\|$ hyper-arcs
$c(h_i)$, $p(h_i)$	Hyper-arc h_i connects child nodes $c(h_i)$ to parent node $p(h_i)$
w_{n_i}	Weight of node n_i
w_{h_i}	Weight of hyper-arc h_i
n_r	The root node of the AND/OR graph
a	An action associated with one or more hyper-arcs in G
A_i	The set of actions associated with hyper-arc h_i
$e(a_j)$	Action a_j is *finished* when performed by an agent
$d(h_i)$	Hyper-arc h_i is *done* when all the actions in A_i are finished
$s(n_k)$	Node n_k is *solved* if there is at least one hyper-arc $h_i \in H$ such that $p(h_i) = n_k$ and $d(h_i)$ holds
$s(G)$	The AND/OR graph G is solved if $s(n_r)$ holds
$f(n_k)$	Node n_k is *feasible* if there is at least one hyper-arc $h_i \in H$ such that $p(h_i) = n_k$ and for all nodes $n_l \in c(h_i)$, $s(n_l)$ holds
$a(h_i)$	Hyper-arc h_i is *active* if $p(h_i) = n_k$ and $f(n_k)$ holds
H_a	The set of all active hyper-arcs at a given time
S_G	The set of all feasible nodes and active hyper-arcs in the AND/OR graph G at a given time
P	A cooperation path traversing G
$cost(P)$	Cost of path P
M	The ordered sequence of actions corresponding to P
P_c	The cooperation path followed at a given time
M_c	The sequence of actions followed at a given time

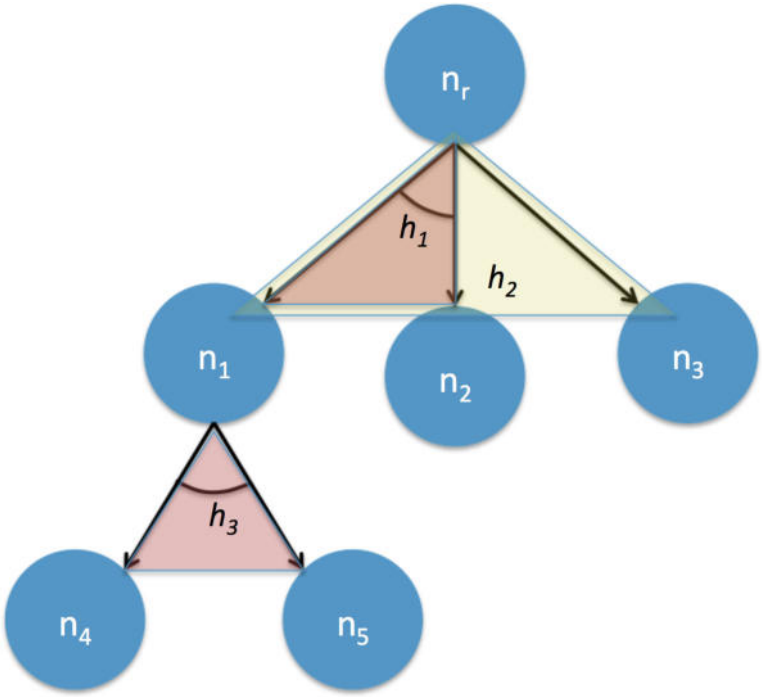


Figure 3.1: A generic AND/OR graph with six nodes and three hyper-arcs: h_1 and h_3 are *and* hyper-arcs, whereas h_2 is a *or* hyper-arc.

where an online use of such a representation has been adopted to model simple cooperation processes, we systematise and extend the original formulation along two directions: first, we provide a conceptualisation of AND/OR graphs compatible with a FOL-based task representation framework; second, we introduce and analyse the benefits of hierarchical AND/OR graphs to support scalable, modular and flexible task representation. Table 3.1 recaps the definitions of the symbols related to the *Representation Level* for ease of reference by the reader.

3.1.1 Propositional logic AND/OR graph

An AND/OR graph $G(N, H)$ is defined as a data structure where N is a set of $n_1, \ldots, n_{|N|}$ nodes and H is a set of $h_1, \ldots, h_{|H|}$ hyper-arcs. Nodes in N define reachable *states*, whereas hyper-arcs in H define *transition* relationships among states. Each hyper-arc $h_i \in H$ defines a *many-to-one* transition relationship between a set of $|c|$ child nodes $c(h_i) = (n_{h_{i,1}}, \ldots, n_{h_{i,|c|}})$ and a parent node $p(h_i) = n_k$. The child nodes of a hyper-arc are in logical *and*, while different hyper-arcs with the same parent node are in logical *or*. Both nodes and hyper-arcs are associated with costs, namely $w_{n_1}, \ldots, w_{n_{|N|}}$ and $w_{h_1}, \ldots, w_{h_{|H|}}$. Figure 3.1 shows an example of AND/OR graph with six nodes termed $n_r, n_1, \ldots, n_5$

and three hyper-arcs, called h_1, h_2 and h_3. Node n_r is called *root* node. Hyper-arc h_1 establishes an *and* relationship between n_1 and n_2 towards n_r, i.e., in order to reach state n_r it is necessary to have reached both n_1 and n_2. A similar condition holds true for h_3, which connects n_4 and n_5 to n_1. Hyper-arc h_2 is an *or* relationship between the *and* relationship involving n_1 and n_2, node n_3 and n_r. The semantics associated with h_2 is such that in order to reach n_r either the couple n_1 and n_2 (via h_1), or alternatively n_3 must be reached first.

In FlexHRC, each hyper-arc h_i models a set A_i of actions $a_1, \ldots, a_{|A_i|}$, and an action $a_j \in A_i$ can be assigned either to a human or a robot. If the order in which to execute actions in A_i is important, A_i is defined as an *ordered* set such that $A_i = (a_1, \ldots, a_{|A_i|}; \preceq)$, i.e., a *temporal* sequence is assumed in the form $a_1 \preceq a_2 \preceq \ldots \preceq a_{|A_i|}$. Initially, all actions $a_j \in A_i$ are labeled as *unfinished*, i.e., $\neg e(a_j)$. When an action a_j has been executed, it is labeled as *finished*, i.e., $e(a_j)$. For all actions in A_i, if $e(a_j)$ holds then h_i is *done*, and the notation $d(h_i)$ is used. If an ordering is induced, $d(h_i)$ holds if and only if also the temporal execution sequence is satisfied.

A node can be either solved or unsolved. A node $n_k \in N$ is *solved*, specified with $s(n_k)$, if there is at least one hyper-arc $h_i \in H$ such that $p(h_i) = n_k$, $d(h_i)$ holds, and for all $n_l \in c(h_i)$ it holds that $s(n_l)$. A node n_k is *unsolved* otherwise, and specified with $\neg s(n_k)$. *Leaves* in G, i.e., nodes n_k for which there is no hyper-arc $h_i \in H$ such that $p(h_i) = n_k$ (as is the case of n_2, n_3, n_4 and n_5 in Figure 3.1), are initialized as solved or unsolved, depending on the initial state of the cooperation. An AND/OR graph G is traversed from leaves to the *root* node $n_r \in N$. When $s(n_r)$ holds, then G is *solved*, i.e., $s(G)$. During the traversal procedure, a node $n_k \in N$ is *feasible*, i.e., $f(n_k)$ holds if there is at least one hyper-arc $h_i \in H$ such that $p(h_i) = n_k$ and for all nodes $n_l \in c(h_i)$ it holds that $s(n_l)$. In this case h_i is labeled as *active*, i.e., $a(h_i)$. Otherwise, n_k is *unfeasible*, i.e., $\neg f(n_k)$. Leaves that are not solved at the start of the cooperation are initialized as feasible. While the cooperation unfolds, there is a set of active hyper-arcs $H_a \subset H$ in G.

We define the *graph representation state* S_G as the set of all feasible nodes and active hyper-arcs in G, i.e., possible action alternatives for the human or the robot. A cooperation path P in G is defined as a sequence of visited nodes and hyper-arcs. Each cooperation path is associated with a *traversal* cost, namely $cost(P)$, which defines how effortful following P is, on the basis of the involved nodes and hyper-arcs weights. A cooperation model M is a ordered sequence of $|M|$ actions, such that $M = (a_1, \ldots, a_{|M|}; \preceq) \subset S_G$, corresponding to an allowed cooperation path in G. At any given time instant, there is one current cooperation model M_c as well as one current cooperation path P_c.

All available cooperation models and cooperation paths are maintained by the *Task Representation* module; which is composed of two online and offline phases.

Algorithm 1 Setup()

Require: A description of an AND/OR graph $G = (N, H)$
Ensure: A data structure encoding G
1: $G \leftarrow$ loadDescription()
2: $s(G) \leftarrow$ *false*
3: **for all** $n \in N$ **do**
4: updateFeasibility(G, n)
5: **end for**
6: $\mathcal{P} \leftarrow$ generateAllPaths(G)
7: $n^* \leftarrow$ findSuggestion($\mathcal{P}$)

3.1.1.1 Offline phase

Algorithm 1 starts the process, loading the description of a cooperative task to create the corresponding AND/OR graph G and set it as unsolved. Then, all node feasibility states are determined (line 4), the set $\mathcal{P}$ of all possible cooperation paths $P_1, \ldots, P_{|\mathcal{P}|}$ are generated (line 6) and the first node to solve n^* is determined (line 7). With reference to the loadDescription() function, it is noteworthy that each description is made up of three data *chunks*, respectively encoding: (i) the structure of the AND/OR graph in terms of the sets N and H, (ii) the set A_i with all actions associated with a hyper-arc h_i, as well as their temporal constraints (if any), and (iii) a number of action-specific parameters, e.g., whether the action must be executed by the operator or the robot, the symbolic action name (for humans) and associated planning and control parameters (for robots).

Feasibility check is performed on each node in N. The process is described in Algorithm 2. Feasible nodes are ignored (line 2) and leaves are set as feasible (line 5). For all other nodes, the algorithm looks for active hyper-arcs (lines 8 to 20): if there is at least one hyper-arc h for which $p(h_i) = n$ and all child nodes are solved, then the hyper-arc is active (line 16) and n is feasible (line 17); otherwise, the hyper-arc is ignored (lines 9 to 14). If a node has no associated active hyper-arcs, then it is not feasible (line 21).

When Algorithm 2 is complete, the graph representation state S_G is available, and it is possible to determine all available cooperation paths. This is done by Algorithm 3, which is a variation of a depth-first traversal procedure for AND/OR graphs. The set of cooperation paths $\mathcal{P}$ and an empty path P are defined (lines 1 to 3). All nodes are initially marked as unexplored (line 5) and the root node n_r is added to path P (line 7). Then, the procedure iterates calling Algorithm 4 on the unexplored nodes (lines 12 and 13) until all nodes are explored and all paths defined (lines 9 and 10).

Algorithm 4 proceeds along a single cooperation path P, starting from the

Algorithm 2 updateFeasibility()

Require: An AND/OR graph $G = (N, H)$, a node $n \in N$
Ensure: $f(n)$ or $\neg f(n)$

```
 1: if f(n) = true then
 2:     return
 3: end if
 4: if c(n) = ∅ then
 5:     f(n) ← true
 6:     return
 7: end if
 8: for all h ∈ H such that p(h) = n do
 9:     allChildNodesSolved ← true
10:     for all m ∈ c(h) do
11:         if s(m) = false then
12:             allChildNodesSolved ← false
13:         end if
14:     end for
15:     if allChildNodesSolved = true then
16:         a(h) ← true
17:         f(n) ← true
18:         return
19:     end if
20: end for
21: f(n) ← false
22: return
```

current node. The cost of P is updated and the node is marked as explored (lines 1 and 2). If the node does not have child nodes, the exploration of the path P from node n is completed (lines 4 and 5); otherwise, if all the child nodes of n belong to the same hyper-arc h, the hyper-arc is added to P (line 7) and the child nodes are added to P for later exploration (line 9); finally, if the child nodes of n belong to more than one hyper-arc, a new path is created for each hyper-arc, as a copy of the current path (lines 15 and 16). The different hyper-arcs and the corresponding child nodes are added to the new paths (lines 17 and 19) for later exploration.

When these procedures end, FlexHRC is ready for online cooperation. Depending on the optimal cooperation path P^*, i.e., the one minimizing the overall cost depending on node and hyper-arc weights, the robot may start moving or waiting for operator actions.

Algorithm 3 generateAllPaths()

Require: An AND/OR graph $G = (N, H)$
Ensure: The set $\mathcal{P}$ of all cooperation paths
1: $\mathcal{P} \leftarrow \emptyset$
2: $P \leftarrow$ initNewPath()
3: $\mathcal{P} \leftarrow \mathcal{P} \cup P$
4: **for all** $n \in N$ **do**
5: $\quad e(n) \leftarrow \mathit{false}$
6: **end for**
7: addNode(P, n_r)
8: **while** *true* **do**
9: $\quad$ **if** $\forall P \in \mathcal{P}$ it holds that $\forall n \in P$, $e(n) = \mathit{true}$ **then**
10: $\quad\quad$ *return* $\mathcal{P}$
11: $\quad$ **else**
12: $\quad\quad$ $P, n \leftarrow$ getUnexploredNode($\mathcal{P}$)
13: $\quad\quad$ generatePath(G, n, P)
14: $\quad$ **end if**
15: **end while**

3.1.1.2 Online phase

Whenever a node is solved and the graph state S_G is updated, the next node to solve n^* in the current cooperation path P_c (which might have changed due to the operator's actions) can be defined.

This is done by Algorithm 5. The Algorithm loops indefinitely until the root node n_r is reached, and therefore $s(G)$ holds (lines 4, 5 and 14). In the meantime, it updates all feasibility states (lines 7-9) as well as cooperation paths (line 10), and provides a suggested next node n^* to solve (line 11).

Whilst feasibility updates are managed by Algorithm 2, cooperation path updates are done as described in Algorithm 6. The set P^u of all paths to update is determined (line 2) as those containing the last solved node n. For each path P, its associated cost is updated as:

$$cost(P) = cost(P) - (w_n + h_n^m - w_h), \tag{3.1}$$

where w_n is the weight associated with n, h_n^m is the maximum weight of the hyper-arcs connecting any parent node to n, and w_h is the weight of the hyper-arc connecting any parent node to n in P.

Suggestions for the next node n^* are determined by the procedure in Algorithm 7. The optimal cooperation path P^* is determined, such that it is characterized by the minimum cost (line 1). Then, for all nodes in P^*, the first node n is found such that $f(n)$ and $\neg s(n)$ hold, which is labeled as n^*.

Algorithm 4 generatePath()

Require: An AND/OR graph $G = (N, H)$, the current node n, a cooperation path P
Ensure: A valid cooperation path P

```
1: updatePathCost(P, n)
2: e(n) ← true
3: for all h ∈ H such that p(h) = n do
4:     if |h| = 0 then
5:         return
6:     else if |h| = 1 then
7:         addArc(P, h)
8:         for all m ∈ c(h) do
9:             addNode(P, m)
10:        end for
11:        return
12:    else if |h| > 1 then
13:        P' ← P
14:        for all h ∈ H such that p(h) = n do
15:            P ← initNewPath(P')
16:            𝒫 ← 𝒫 ∪ P
17:            addArc(P, h)
18:            for all m ∈ c(h) do
19:                addNode(P, m)
20:            end for
21:        end for
22:        return
23:    end if
24: end for
```

The hyper-arcs with n^* as parent (i.e., within the current cooperation model), together with all hyper-arcs with other, currently feasible nodes as parent (i.e., within other, admissible cooperation models) are marked as *active* and constitute the new *Action-State* table, which the *Planner* uses to monitor and drive the suggestions for actions, by giving highest priority to the actions in the hyper-arc with minimum cost.

3.1.2 First order logic AND/OR graph

An AND/OR graph G in FOL can be formally defined as a tuple $\langle N, H \rangle$ where N is a set of $|N|$ nodes, and H is a set of $|H|$ hyper-arcs. An hyper-arc $h \in H$

Algorithm 5 NextSuggestedNode()

Require: An AND/OR graph G, the last solved node $n \in N$
Ensure: An updated AND/OR graph G, the next node to solve n^*

```
1: loop ← true
2: s(n) ← true
3: while loop = true do
4:     if n = n_r then
5:         loop ← false
6:     end if
7:     for all m ∈ N do
8:         updateFeasibility(m)
9:     end for
10:    updateAllPaths(P)
11:    n* ← findSuggestion(P)
12:    return
13: end while
14: s(G) ← true
15: return
```

Algorithm 6 updateAllPaths()

Require: The set $\mathcal{P}$ of all cooperation paths, the last solved node $n \in N$
Ensure: An updated set $\mathcal{P}$

```
1: P^u ← ∅
2: P^u ← findPathsToUpdate(n)
3: for all P ∈ P^u do
4:     cost(P) ← updateCost()
5: end for
```

Algorithm 7 findSuggestion()

Require: The set $\mathcal{P}$ of all cooperation paths
Ensure: The next node to solve n^*

```
1: P* ← findOptimalPath(P)
2: for all n ∈ P* do
3:     n* ← findOptimalNode()
4: end for
5: return
```

induces two sets of nodes, namely the set $N_c(h) \subset N$ of its *child* nodes, and the

singleton $N_p(h) \subset N$ made up of a *parent* node, such that:

$$h : N_c(h) \rightarrow N_p(h). \tag{3.2}$$

For the scenarios considered in this paper, at the semantic level each node $n \in N$ represents a peculiar *state* related to the cooperation between a human operator and a collaborative robot, whereas each hyper-arc $h \in H$ represents a (possibly) many-to-one *transition* among states, i.e., activites performed by human operators or robots that make the cooperation move forward. In FlexHRC, a node $n \in N$ is associated with a conjunction $S(n)$ of literals, such that:

$$S(n) = s_1 \wedge \ldots \wedge s_k \wedge \ldots \wedge s_{|S(n)|}, \tag{3.3}$$

where each literal s_k may consist of variables, constants or logic predicates, also negated. As it will be described later, each literal can be considered a representation *fragment* related to the cooperation state defined by n. As a consequence, we will refer to $S(n)$ as the *cooperation state* represented in n. It is noteworthy that a given S does not have to necessarily include only grounded literals, i.e., constants or grounded predicates, but it can include variables as well, and as such the corresponding node can be treated as a *class* of states at the *Task Representation* level Russell & Norvig (2010).

Using the definition of states in (3.3), it is possible to better specify the state transition in (3.2) as a relationship induced on the hyper-arc h between a set of requirements made up by joining all the literals defining states $S(n_i)$ associated with all the nodes $n_i \in N_c(h)$, and a set of effects made up by the state $S(n)$ associated with the single node $n \in N_p(h)$, such that:

$$h : S(n_1) \wedge \ldots \wedge S(n_k) \wedge \ldots \wedge S(n_{|N_c|}) \rightarrow S(n). \tag{3.4}$$

It can be observed that the relation among child nodes in hyper-arcs is the logical *and*, whereas the relation between different hyper-arcs inducing on the same parent node is the logical *or*, i.e., different hyper-arcs inducing on the same parent node represent alternative ways for a cooperation process to move on. Furthermore, we define $n \in N$ as a *leaf* node if n is not acting as a parent node for any set of nodes, i.e., if $h \in H$ does not exist such that $n \in N_p(h)$, or as a *root* node if it is the only node that is not a child node for any other node, i.e., if $h \in H$ does not exist such that $n \in N_c(h)$.

Each hyper-arc $h \in H$ implements the transition in (3.4) by checking the truth values associated with all requirements defined by relevant nodes, executing a number of actions associated with h, and generating effects compatible with the cooperation state of the parent node. In particular, each hyper-arc $h \in H$ is responsible for executing an ordered set $A(h)$ of *actions*, such that:

$$A(h) = (a_1, \ldots, a_k, \ldots, a_{|A|}; \preceq), \tag{3.5}$$

where the precedence operator $\preceq$ defines the pairwise expected order of action execution. Before an hyper-arc h is executed, all actions $a \in A(h)$ are marked as *undone*, and we refer to this using a predicate $\mathsf{done}(a) \leftarrow false$. When one action a is executed either by the human operator or the robot, its status changes to *done* as $\mathsf{done}(a) \leftarrow true$. An hyper-arc $h \in H$ is marked as *solved*, i.e., $\mathsf{solved}(h) \leftarrow true$ *iff* all actions $a \in A(h)$ are done in the expected order. In a similar way, nodes $n \in N$ may be associated with a (possibly ordered) set of *processes* $P(n)$, i.e.,

$$P(n) = (p_1, \ldots, p_k, \ldots, p_{|P|}; \preceq). \tag{3.6}$$

Diffently from actions, which are instrumental to perform transitions among states and must be necessarily executed by human operators or robots, processes are relevant *within* states as *fluents*, and model physical or other non-functional variations of some quantity over time, without leading to qualitatively different states from the human-robot cooperation state perspective. An example may be a robot behaviour aimed at keeping a certain object in a given pose or configuration using two grippers, the effects of external forces notwithstanding. In this case, such a process does not lead to a different cooperation state, and its effects are limited to the current state. When a node $n \in N$ is reached, all of its processes are *activated*, i.e., $\mathsf{activated}(p) \leftarrow true$ for each $p \in P(n)$. A process $p \in P(n)$ can be *deactivated* when certain process-specific termination conditions are met, i.e., $\mathsf{activated}(p) \leftarrow false$. A node n is marked as *met*, i.e., $\mathsf{met}(n) \leftarrow true$, if all the associated processes are deactivated if necessary in the prescribed order, or $P(n)$ is an empty set.

Using these definitions, it is possible to introduce the notion of feasibility for nodes and hyper-arcs. A node $n \in N$ is *feasible*, which we refer to as $\mathsf{feasible}(n) \leftarrow true$, *iff* a solved hyper-arc $h \in H$ exists, for which $n \in N_p(h)$, and $\mathsf{met}(n) \leftarrow false$, i.e.,

$$\exists h \in H. \left(\mathsf{solved}(h) \cap n \in N_p(h) \cap \neg\mathsf{met}(n)\right). \tag{3.7}$$

All leaf nodes in an AND/OR graph are usually feasible at the beginning of the human-robot cooperation process, which means that the cooperation itself can be performed in many ways and is not constrained to follow certain sequences of operations. In a similar way, an hyper-arc $h \in H$ is *feasible*, i.e., $\mathsf{feasible}(h) \leftarrow true$, *iff* for each node $n \in N_c(h)$, $\mathsf{met}(n) \leftarrow true$ and $\mathsf{solved}(h) \leftarrow false$, i.e.,

$$\forall n \in N_c(h). \left(\mathsf{met}(n) \cap \neg\mathsf{solved}(h)\right). \tag{3.8}$$

Once an hyper-arc $h_i \in H$ is solved, all other feasible hyper-arcs $h_j \in H \setminus \{h_i\}$, which share with h_i at least one child node, i.e., $N_c(h_i) \cap N_c(h_j) \neq \emptyset$, are marked as unfeasible, in order to prevent the cooperation process to consider alternative ways to cooperation that have become irrelevant.

The human-robot cooperation process is therefore modelled as a *graph traversal* procedure which, starting from a set of leaf nodes, must reach the root node of the graph by selecting hyper-arcs and reaching states in one of the available sequences, depending on the feasibility status of nodes and hyper-arcs. To this aim, each node $n \in N$ is associated with a *weight* $w(n)$, and each hyper-arc $h \in N$ is similarly associated with a weight $w(h)$. Weights depend on a number of parameters, which may be related to the number, difficulty or time-to-completion of actions/processes, and on other more qualitative metrics related to human operator preferences Buoncompagni & Mastrogiovanni (2018). Then, a *cooperation path* cp induced by G is a set of nodes and hyper-arcs, such that

$$cp = (n_1, \ldots, n_k, h_1, \ldots, h_l), \tag{3.9}$$

which represents a peculiar way to connect leaf nodes to the root node. We refer to the set of cooperation pathes induced by G as $CP(G)$, where each element $cp \in CP$ is in the form described by (3.9). According to the structure of the modelled human-robot cooperation task, multiple cooperation paths may exist, meaning that multiple ways to solve the task may be equally legitimate. Each cooperation path $cp \in CP$ can be associated with an overall cost $c(cp)$, such that:

$$c(cp) = \sum_{j=1}^{k} w(n_j) + \sum_{j=1}^{l} w(h_j). \tag{3.10}$$

The different cooperation paths in CP can be ranked according to their overall costs. Two cooperation paths cp_i and $cp_j \in CP$ are *equal iff* the corresponding sets of nodes and hyper-arcs are the same, and are *equivalent iff* their corresponding overall costs are the same.

The traversal procedure dynamically follows the cooperation path that at any time is characterised by the lowest cost. As a consequence, the traversal procedure suggests to human operators actions in the hyper-arcs that are part of the path, and sends to robots actions they must execute. However, human operators can override at any time suggestions, executing different actions, which may cause the system to be in a cooperation state not part of the current cooperation path. When this situation is detected, FlexHRC tries to progress from that state onwards Darvish *et al.* (2018a,b). This mechanism enables FlexHRC to pursue an optimal path leading to the solution, while it allows human operators to choose alternative paths when they deem it fit.

As long as the human-robot cooperation process unfolds, and the AND/OR graph is traversed, we refer with N_f and H_f to the sets of *currently* feasible nodes and hyper-arcs, respectively. In fact, the actual members of these two sets depend on the particular evolution of the cooperation process. We say that an AND/OR graph G is *solved*, and we write $\mathsf{solved}(G) \leftarrow true$, *iff* its root node $r \in N$ is

Algorithm 8 offlinePhase()

Require: An AND/OR graph $G = \langle N, H \rangle$
Ensure: A data structure encoding G
1: $G \leftarrow$ loadDescription()
2: $\mathsf{solved}(G) \leftarrow \mathit{false}$
3: updateGraphFeasibility(G)
4: $CP \leftarrow$ generateAllPaths(G)
5: $\{\langle x_i, c(x_i)\rangle\} \leftarrow$ findSuggestions(G)

met, i.e., $\mathsf{met}(r) \leftarrow \mathit{true}$. Otherwise, if the condition $N_f \cup H_f = \emptyset$ (i.e., there are no feasible nodes nor hyper-arcs) then the human-robot cooperation process is failed, because there is no feasible cooperation path leading to the root.

It is noteworthy that representations based on AND/OR graphs, when updated online, do not require the full knowledge of the robot workspace, nor actions that are irrelevant for the current cooperation path. In fact, while a given cooperation path is followed, the traversal algorithm only needs knowledge about feasible nodes and hyper-arcs for making the task progress.

3.1.3 Single-layer AND/OR Graph Traversal Procedure

The single-layer AND/OR graph traversal procedure is composed of two phases, the first being offline and the second online. The offline phase is described in Algorithm 8. The AND/OR graph structure presented in this paper is based on the one introduced in Darvish *et al.* (2018b), with notable differences such as the possibility of allowing for multiple hyper-arcs connecting the same children nodes to a parent node, ensuring the minimum cost returned from each hyper-arc or node, and supporting the FOL-based representation of the cooperation task. The first feature allows the AND/OR graph to model different possible state transitions from one cooperation state to another, the second one ensures an optimal and therefore a predictable robot behaviour, whereas the last one increases the overall expressive power of the representation structure.

The functions *loadDescription()* and *generateAllPaths()* in Algorithm 8 are described with great detail in Darvish *et al.* (2018b). In summary, *loadDescription()* generates the data structure G (line 1), the graph G is setup as unsolved (line 2), all feasibility statuses for nodes and hyper-arcs are checked (line 3), the set CP of cooperation paths is generated (line 4), and suggestions for next actions (in terms of nodes and hyper-arcs) are computed on the basis of path costs defined as in (3.10).

It is worth discussing the behaviour of function *updateGraphFeasibility(G)*, described by Algorithm 9, which updates the feasibility statuses of all involved

Algorithm 9 updateGraphFeasibility()

Require: An AND/OR graph $G = \langle N, H \rangle$
Ensure: The feasibility sets N_f and H_f
1: $N_f = \emptyset$
2: $H_f = \emptyset$
3: **for all** $n \in N$ **do**
4: $\quad (N_f, H_f) \leftarrow$ updateNodeFeasibility(n, N_f, H_f)
5: **end for**
6: **for all** $h \in H$ **do**
7: $\quad (N_f, H_f) \leftarrow$ updateHyperarcFeasibilty(h, N_f, H_f)
8: **end for**

nodes and hyper-arcs, and populates the corresponding sets N_f and H_f. The Algorithm works simply by iteratively invoking two functions, namely *updateFeasibilityNode()* (line 4) and *updateFeasibiltyHyperarc()* (line 7). The two functions are further developed in Algorithm 10 and Algorithm 11. Given a node n and a hyperarc h the two Algorithms use such predicative knowledge on n and h as the values of $\mathsf{feasible}(n)$, $\mathsf{feasible}(h)$, $\mathsf{met}(n)$, and $\mathsf{solved}(h)$ to update the feasibility of graph nodes and hyper-arcs, respectively, therefore producing updated sets N_f and H_f. In Algorithm 10, lines 3-19 update the feasibility status of a relevant hyper-arc h when $n \in N_c(h)$ and met(n) holds true. In case node met(n) holds false (lines 20-32), lines 21-23 change the node feasibility when it does not have any child nodes, i.e., if n is not a parent of any hyper-arc, whereas lines 24-31 check for a solved hyper-arc connected to node n, and in case at least one of such hyper-arcs exist, then it is marked as feasible. In Algorithm 11, lines 2-13 update the feasibility statuses when the hyper-arc h is solved. The feasibility of the h's parent node (line 3) is updated in lines 4-7. The feasibility of all the hyper-arcs that have a common set of child nodes with h is updated in lines 8-13. Lines 14-27 check the feasibility of the unsolved hyper-arc h; if a child node of h (line 17) is not met (lines 18-21) or there is another solved hyper-arc h' with a common set of child nodes with h (lines 22-25), the hyper-arc h becomes infeasible. Finally, *findSuggestions()* in Algorithm 12 determines the set of feasible nodes and hyper-arcs, generically indicated using x, and their associated cost $c(x)$. There might be different paths from the feasible nodes N_f or hyper-arcs H_f to the root of G. Therefore, the AND/OR graph is expected to provide the *minimum cost* among all these cooperation paths such that the optimality of the cooperation process is ensured. The cost $c(x)$ for a node or hyper-arc is the minimum cost of the cooperation path cp which the node or the hyper-arc belongs to, and therefore the Algorithm guarantees the optimality of the AND/OR graph because for all

Algorithm 10 updateNodeFeasibility()

Require: A node n
Ensure: The feasibility sets N_f and H_f

```
1: feasible(n) ← false
2: N_f ← N_f\{n}
3: if met(n) then
4:     for all h s.t. n ∈ N_c(h) do
5:         if solved(h) then
6:             feasible(h) ← false
7:             H_f ← H_f\{h}
8:         else
9:             feasible(h) ← true
10:            H_f ← H_f ∪ {h}
11:            for all n' s.t. n' ∈ N_c(h) do
12:                if met(n') then
13:                    feasible(h) ← false
14:                    H_f ← H_f\{h}
15:                    break
16:                end if
17:            end for
18:        end if
19:    end for
20: else
21:    if N_c(n) = ∅ then
22:        feasible(n) ← true
23:        N_f ← N_f ∪ {n}
24:    else
25:        for all h s.t. n ∈ N_p(h) do
26:            if solved(h) then
27:                feasible(h) ← true
28:                H_f ← H_f ∪ {h}
29:            end if
30:        end for
31:    end if
32: end if
```

nodes or hyper-arcs in N_f or H_f, respectively, it holds that:

$$\min_{x \in cp} c(cp). \tag{3.11}$$

In Algorithm 12, lines 3-11 return feasible nodes and the minimum cost of the

Algorithm 11 UpdateFeasibiltyHyperarc()

Require: A hyper-arc h
Ensure: The feasibility sets N_f and H_f

```
1: feasible(h) ← false
2: if solved(h) then
3:     n ← N_p(h)
4:     if ¬met(n) then
5:         feasible(n) ← true
6:         N_f ← N_f ∪ {n}
7:     end if
8:     for all n s.t. n ∈ N_c(h) do
9:         for all h' s.t. n ∈ N_c(h') do
10:            feasible(h') ← false
11:            H_f ← H_f \ {h'}
12:        end for
13:    end for
14: else
15:    feasible(h) ← true
16:    H_f ← H_f ∪ {h}
17:    for all n s.t. n ∈ N_c(h) do
18:        if ¬met(n) then
19:            feasible(h) ← false
20:            H_f ← H_f \ {h}
21:            break
22:        else if ∃h' s.t. solved(h') ∧ n ∈ N_c(h') then
23:            feasible(h') ← false
24:            H_f ← H_f \ {h'}
25:        end if
26:    end for
27: end if
```

cooperation paths which include them. The same applies to hyper-arcs in lines 12-20.

During online execution, when a node is met or a hyper-arc is solved, the *Task Manager* module queries the AND/OR graphs to get updated N_f and H_f, as well as the associated costs, in order to make the cooperation progress. This is done by Algorithm 13. In the Algorithm, the two sets of met nodes and solved hyper-arcs are referred to as N_m and H_s, respectively. Upon the reception of the *Task Manager*'s query, the Algorithm updates node and hyper-arc statuses (in terms of solved, met and feasible predicates) in lines 2-9. Later, the solved

Algorithm 12 findSuggestions()

Require: An AND/OR graph $G = \langle N, H \rangle$
Ensure: A set $\Phi = \{\langle x_i, c(x_i) \rangle\}$

```
1:  Φ = ∅
2:  cost ← 0
3:  for all n ∈ N s.t. feasible(n) do
4:      cost ← inf
5:      for all cp ∈ CP(G) s.t. n ∈ cp do
6:          if cost < c(cp) then
7:              cost ← c(cp)
8:          end if
9:      end for
10:     Φ ← Φ ∪ ⟨n, cost⟩
11: end for
12: for all h ∈ H s.t. feasible(h) do
13:     cost ← inf
14:     for all cp ∈ CP(G) s.t. h ∈ cp do
15:         if cost < c(cp) then
16:             cost ← c(cp)
17:         end if
18:     end for
19:     Φ ← Φ ∪ ⟨h, cost⟩
20: end for
```

status for the whole AND/OR graph is checked. If the root node is met, then the graph is marked as solved (line 11). Otherwise, line 14 updates all the path weights, which include nodes in N_m and hyper-arcs in H_s. Finally in line 15 the new feasible nodes and hyper-arcs, and their associated costs, are made available. In the Algorithm, the functions *metNode(n, G)* and *solvedHyperarc(h, G)* check first if feasible(n) or feasible(h) hold true, then update G by met$(n) \leftarrow true$ and solved$(h) \leftarrow true$. In particular, Algorithm 14 updates the cooperation path costs at each query. For a given cooperation path $cp \in CP$, the path cost $c(cp)$ at each moment is the cost of traversing it from the current to the root state. Initially, all the path costs are computed from the leaves to the root using (3.10). When a node or hyper-arc belonging to a given cooperation path is met or solved, its overall cost is reduced of an amount related to its weight (lines 3 and 8).

Algorithm 13 onlinePhase()

Require: An AND/OR graph $G = \langle N, H \rangle$, the feasibility sets N_f and H_f, the met set N_m, the solved set H_s
Ensure: An updated AND/OR graph G, updated feasibility sets N_f and H_f, the associated costs c

```
1: n_r ← getRoot(G)
2: for all n ∈ N_m do
3:     metNode(n, G)
4:     updateNodeFeasibility(n, N_f, H_f)
5: end for
6: for all h ∈ H_s do
7:     solvedHyperarc(h, G)
8:     updateHyperarcFeasibility(h, N_f, H_f)
9: end for
10: if met(n_r) then
11:     solved(G) ← true
12:     return
13: end if
14: updateAllPaths(G, N_m, H_s)
15: {⟨x, c(x)⟩} ← findSuggestions(G), where x ∈ N_f ∪ H_f
16: return
```

Algorithm 14 updateAllPaths()

Require: An AND/OR graph $G = \langle N, H \rangle$, the met set N_m, the solved set H_s
Ensure: An updated set CP

```
1: for all n ∈ N_m do
2:     for all cp ∈ CP s.t. n ∈ cp do
3:         c(cp) = c(cp) − w(n)
4:     end for
5: end for
6: for all h ∈ H_s do
7:     for all cp ∈ CP s.t. h ∈ cp do
8:         c(cp) = c(cp) − w(h)
9:     end for
10: end for
```

3.1.4 Hierarchical AND/OR graph

In order to deal with HRC tasks, the use of hierarchical AND/OR graphs has two motivations, the first related to the computational complexity of single-layer

AND/OR graphs, the second by flexibility and scalability requirements. On the one hand, it has been shown that AND/OR graphs are characterised by a polynomial time complexity in the number of nodes and hyper-arcs Laber (2008), whereas the problem of determining whether a solution in terms of a path from the set N_L of leaf nodes to the root node is NP-hard Sahni (1974). In the online phase of HRC tasks, being able to quickly determine and select an alternative cooperation path to take into account human operator preferences is of the utmost importance to enhance the overall usability of the collaborative robot. On the computational side, this means reducing the number of nodes and hyper-arcs which the *Task Manager* module must reason upon. On the other hand, different real-world operations are structured as mandatory or alternative sets of human or robot actions, which can be seen as *atomic*. Being able to identify and re-use the same sub-sequences of operations in different parts of the same HRC process or as part of different processes is expected to enhance flexibility, because such sub-sequences can be easily substituted if needed, and scalability, since the overall complexity can be increased maintaining a manageable representation overhead.

Analogously to single-layer AND/OR graphs, a hierarchical AND/OR graph $\mathcal{H}$ can be defined as a tuple $\langle \Gamma, \Theta \rangle$ where Γ is an ordered set of $|\Gamma|$ AND/OR graphs, such that:

$$\Gamma = \left(G_1, \ldots, G_{|\Gamma|}; \preceq\right), \tag{3.12}$$

and Θ is a set of $|\Theta|$ transitions between couples of AND/OR graphs. In (3.12), the AND/OR graphs are pairwise ordered according to their *depth* level. With a slight abuse of notation, we associate a depth level l to an AND/OR graph G and we indicate it with G^l, the highest level being $l = 0$. AND/OR graphs with increasing depth levels are characterised by a decreasing level of abstraction, i.e., deeper graphs model HRC more accurately. Transitions in Θ define how different AND/OR graphs in Γ are connected, and in particular model the relationship between any G^l and a deeper connected graph G^{l+1}.

It is necessary to better define transitions. If we recall (3.4) and we contestualise for an AND/OR $G^l = \langle N^l, H^l \rangle$, we observe that a given hyper-arc in H^l represents a mapping between the set of its child nodes and the singleton parent node. We can think of a generalised version of such mapping to encompass a whole AND/OR graph $G^{l+1} = \langle N^{l+1}, H^{l+1} \rangle$, where the set of child nodes is constituted by the set N_L^{l+1} of leaf nodes, and the singleton parent node by the graph's root node $r^{l+1} \in N^{l+1}$, such as:

$$G^{l+1} : S(n_1^{l+1}) \wedge \ldots \wedge S(n_k^{l+1}) \wedge \ldots \wedge S(n_{|N_L^{l+1}|}^{l+1}) \rightarrow S(r^{l+1}). \tag{3.13}$$

As a consequence, a transition T can defined between a hyper-arc $h^l \in H^l$ and an entire deeper AND/OR graph G^{l+1}, such that:

$$T : h^l \rightarrow G^{l+1}, \tag{3.14}$$

subject to the fact that appropriate mappings can be defined between the set of child nodes of h^l and the set of leaf nodes of the deeper graph, i.e.,

$$M_1 : N_c(h^l) \rightarrow N_L \in N^{l+1}, \tag{3.15}$$

and the singleton set of parent nodes of h^l and the root node of the deeper graph, i.e.,

$$M_2 : N_p(h^l) \rightarrow r^{l+1} \in N^{l+1}. \tag{3.16}$$

Mappings M_1 and M_2 must be such that the conjunction of literals of nodes in $N_c(h^l)$ and the conjunction of literals of leaves in G^{l+1} should be *semantically equivalent*, i.e., they should be the same or representing the same information with a different depth of representation, for example each literal of nodes in $N_c(h^l)$ may correspond to one or more literals of nodes in N_L^{l+1}. The same applies for the root of G^{l+1} and $N_p(h^l)$. Once these mappings are defined, it easy to see that $\mathcal{H}$ has a tree-like structure, where graphs in Γ are nodes and transitions in Θ are edges.

As far as the overall workflow is concerned, an AND/OR graph G^l is feasible, and we refer to it as $\mathsf{feasible}(G^l)$ if it has at least one feasible node or hyper-arc. If a transition T exists in the form (3.14), a hyper-arc $h^l \in H^l$ is feasible *iff* the associated deeper AND/OR graph G^{l+1}, is feasible:

$$\forall T. \left(\mathsf{feasible}(G^{l+1}) \leftrightarrow \mathsf{feasible}(h^l)\right). \tag{3.17}$$

As a consequence, when hyper-arc h^l becomes feasible in G^l, the nodes in N_L^{l+1} of G^{l+1} become feasible as well. Furthermore, hyper-arc h^l is solved *iff* the associated deeper AND/OR graph G^{l+1} is solved:

$$\forall T. \left(\mathsf{solved}(G^{l+1}) \leftrightarrow \mathsf{solved}(h^l)\right). \tag{3.18}$$

For all hyper-arcs in H^l for which a transition T towards G^{l+1} exists, we must define how to compute the related weight. In particular, if we define $cp^{l+1,*}$ the cooperation path in G^{l+1} characterised by the lowest cost, we easily define:

$$w(h^l) = c\left(cp^{l+1,*}\right). \tag{3.19}$$

It is noteworthy that in this case the weight is attributed using an optimistic strategy, because as per change of the cooperation path in G^{l+1} it may happen that the actual $w(h^l)$ is underestimated.

Similarly to the single-layer case, hierarchical AND/OR graphs are used in two phases, first offline and then online. From a computational standpoint, a transition T is modelled using a function in the form $G^{l+1} = \textsc{lowerGraph}(h^l)$, whereas the inverse relationship is obtained using $h^l = \textsc{upperHyperarc}(G^{l+1})$.

The offline phase, similar to what already described for the single-layer case in Algorithm 8, first loads the description of the highest-level AND/OR graph G^0. Considering any nesting level l, if a hyper-arc $h \in H^l$ is associated with a deeper AND/OR graph description G^{l+1} by a transition, the Algorithm recursively calls the function *OfflinePhase*() on G^{l+1} to build it before going on with G^l.

Algorithm 15 describes the workflow associated with the hierarchical AND/OR graph during online execution. Whenever the status of the HRC process needs updating, the graph representation is updated starting from all sets $N_{i,m}$ of met nodes, and the sets $H_{i,s}$ of solved hyper-arcs, for all AND/OR graphs in Γ (lines 3-12). In particular, after node statuses are updated in lines 5-6, the Algorithm checks whether any graph is solved (line 7): if this holds true and the solved graph is not the root graph of $\mathcal{H}$, then the associated higher-level hyper-arc is labelled as solved (line 9) and then included in the corresponding set of solved hyper-arcs $H_{i,s}$. Lines 14-19 update the feasibility statues for all solved hyper-arcs. Then, if the root node of the root graph is met, then the whole graph is solved (line 21) and the Algorithm terminates (line 22). Otherwise, all cooperation paths are updated (line 25), and the set Φ^H of next suggestions is found (line 30), as better described in Algorithm 16. It is noteworthy that Φ^H includes Φ and adds to each triplet the graph label containing the node or hyper-arc.

The Algorithm 16 finds first feasible nodes part of an optimal cooperation path (lines 2-12), as well as the associated cost and graph. A similar operation is done in lines 13-32 for hyper-arcs. However, in this case it is necessary to check whether a transition exists towards a deeper AND/OR graph. If this is not the case, the hyper-arc is stored as a suggestion. Otherwise, the associated graph is determined and the function is recursively called on it. Finally, the minimum cost from the parent node of a hyper-arc to the root node of the corresponding graph is computed in line 27, and the suggestion updated accordingly.

3.2 Task Manager

Task Manager receives the set of feasible states or state transitions from the *Task Representation*, determines the sequences of actions for all the states or state transitions, and grounds the parameters of the actions and assigns actions to humans or robots such that the cooperation utility maximizes. The module reactively adapts to the human online decisions.

3.2.1 Task manager formalization

The principal attribute in the *Task Manager* is the notion of *action*. An *action* a is a transition applied on the parameters of the action from a set of physical

Algorithm 15 onlineHierarchicalPhase()

Require: A hierarchical AND/OR graph $\mathcal{H} = \langle \Gamma, \Theta \rangle$, feasibility sets $N_{i,f}$ and $H_{i,f}$, the met set $N_{i,m}$, the solved set $H_{i,s}$ for each $G_i \in \Gamma$

Ensure: An updated hierarchical AND/OR graph $\mathcal{H}$, updated feasibility sets $N_{i,f}$ and $H_{i,f}$ for each $G_i \in \Gamma$, a set $\Phi^H = \{\langle x, c(x), g(x)\}$ of suggestions

1: $G^0 \leftarrow$ getRootGraph(Γ)
2: $r^0 \leftarrow$ getRoot(G^0)
3: **for all** $G_i \in \Gamma$ **do**
4: **for all** $n \in N_{i,m}$ **do**
5: metNode(n, G_i)
6: updateNodeFeasibility(n, G_i)
7: **if** solved(G_i) and $G_i \neq G^0$ **then**
8: $h \leftarrow$ upperHyperarc(G_i)
9: solved(h) $\leftarrow$ *true*
10: $H_{i,s} \leftarrow H_{i,s} \cup \{h\}$
11: **end if**
12: **end for**
13: **end for**
14: **for all** $G_i \in \Gamma$ **do**
15: **for all** $h \in H_{i,s}$ **do**
16: solvedHyperarc(h, G_i)
17: updateHyperarcFeasibility(h, $N_{i,f}$, $H_{i,f}$)
18: **end for**
19: **end for**
20: **if** met(r^0) **then**
21: solved(G^0) $\leftarrow$ *true*
22: **return**
23: **end if**
24: **for all** $G_i \in \Gamma$ **do**
25: updateAllPaths(G_i, $N_{i,m}$, $H_{i,s}$)
26: **end for**
27: $N_f \leftarrow N_{1,f} \cup \ldots \cup N_{|\Gamma|,f}$
28: $H_f \leftarrow H_{1,f} \cup \ldots \cup H_{|\Gamma|,f}$
29: $\Phi^H = \emptyset$
30: $\{\langle x, c(x), g(x)\rangle\} \leftarrow$ findSuggestions($\mathcal{H}$, Φ^H)
31: **return**

states (conjunction of literals, preconditions) to the new states (goal), $a : s \rightarrow s'$.

Algorithm 16 findSuggestions()

Require: A hierarchical AND/OR graph $\mathcal{H} = \langle \Gamma, \Theta, \rangle$, a set $\Phi^H = \{\langle x, c(x), g(x)\rangle\}$ of suggestions
Ensure: An updated set Φ^H

```
1: cost ← 0
2: for all G_i = ⟨N_i, H_i⟩ ∈ Γ do
3:     for all n ∈ N_i s.t. feasible(n) do
4:         cost ← inf
5:         for all cp ∈ CP(G_i) s.t. n ∈ cp do
6:             if cost < c(cp) then
7:                 cost ← c(cp)
8:             end if
9:         end for
10:        Φ^H ← Φ^H ∪ {⟨n, cost, G_i⟩}
11:    end for
12: end for
13: for all G_i = ⟨N_i, H_i⟩ ∈ Γ do
14:    for all h ∈ H_i s.t. feasible(h) do
15:        cost ← inf
16:        for all cp ∈ CP(G_i) s.t. h ∈ cp do
17:            if cost < c(cp) then
18:                cost ← c(cp)
19:            end if
20:        end for
21:        if lowerGraph(h) = null then
22:            Φ^H ← Φ^H ∪ {⟨h, cost, G_i⟩}
23:        else
24:            G_j ← lowerGraph(h)
25:            Φ^H ← findSuggestion(G_j, Φ^H), with x ∈ N_{j,f} ∪ H_{j,f}
26:            for all ⟨x, c(x), g(x)⟩ ∈ Φ do
27:                cost ← cost − w(h) + c(x)
28:                Φ^H ← Φ^H ∪ {⟨x, c(x), g(x)⟩}
29:            end for
30:        end if
31:    end for
32: end for
33: return Φ^H
```

Using PDDL-like formalization, an action is defined as:

$$\begin{aligned} &action : a(Parameters := \{par_1, ..., par_n\}, Agents), \\ &PreCondition : (pc_1 \wedge ... \wedge pc_m) \\ &Effect : (eff_1 \wedge ... \wedge eff_l) \end{aligned} \tag{3.20}$$

where $Parameter$, $PreCondition \subset s$, and $PostCondition \subset s'$ are conjunction of literals which their interpretations are known. $Agents(a) = Ag_1 \times ... \times Ag_z$ are the responsible agents for performing the action a. If the number of agents performs an action a_i is bigger than one, we call it a *joint action*. In this formalization, although the *interpretation* of the literals are known they may not be grounded. We assume the grounding of the parameters does not affect the planning problem, otherwise, we solve a new planning problem.

Given the feasible state transitions from the *Task Representation*, all the initial states and goals are defined as well the domain of the actions. Taking into account the interpretation of the variables in *Task Manager* we define a planning problem to find the ordered sequence of actions $(a_1, ..., a_n)$ from the initial states to the goal such that the state transition is executed. We assume the sequence of these actions is provided.

Given the set of feasible states or state transitions and the associated costs, the *Task Manager* either proactively decides for the state transition to follow (*proactive decision making*), or follows the human preferred decision through the planning recognition (*reactive adaptation*). We call the selected state or state transition *optimal state*, which has the minimum cost according to Equation (3.10) to reach the cooperation goal.

3.2.2 Proactive decision making

It is the process of selecting the state with the minimum costs (optimal state), grounding the literals, assigning the actions to the human or robot, and the examination of the optimal state execution. Given the optimal state, *Task Manager* gets the information of the workspace from the *Knowledge Base*, finds all the possible grounding of the parameters of the actions and the possible agents (human or robot) who can perform all the actions of the optimal state. Using all the possible combination of parameters grounding and assigned agent, we create a data structure so-called *decision tree* to make decisions for future actions of the cooperation process Darvish *et al.* (2018a), which is different from the classical decision tree introduced in Russell & Norvig (2010). Then, we examine the execution of all the branches by simulating the robot actions online. We use the results of the simulations to compute the *utility value* of the decision tree branches, which is a metric to estimate the performance and the quality of the execution of a state transition Gerkey & Matarić (2004). In this work, we define the utility function as:

$$J = unit(success, failure) \times \frac{1}{\sum_{k=1}^{k=K} t_k} \tag{3.21}$$

where K is the number of actions in the optimal states, t_k is the execution time of the action k in the simulation, and the unit function equals to one if all the

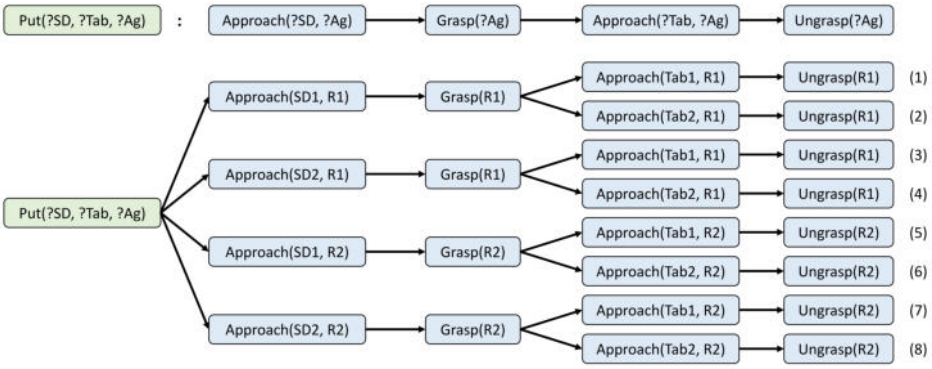


Figure 3.2: The decision tree used by the *Simulator* module for online task allocation and parameter grounding. SD stands for a screwdriver, Ag stands for an agent, and Tab stands for a table.

actions in the simulation executed successfully, otherwise, it is zero. Finally, we ground the parameters of the optimal state and assign the actions to the agent based on the branch with the maximum utility value.

Consider the action *Put down screwdriver on table*, which does not specify which screwdriver and table are involved; in principle, there may exist more than one screwdriver and table in the robot workspace, which may ground such a command. On the one hand, the robot should consider all the possible instances of *screwdriver* and *table* to find the optimal and complete plan to address such a command. On the other hand, there might be more than one agent *capable* of performing the associated *put down* action, and therefore they should coordinate to determine which one executes it. In fact, if the number of instances of screwdrivers and tables are m and n, respectively, and k agents (human operators or robots) can perform the action, there are $m \times n \times k$ possible realization of the *put down* action.

An example of the analysis done by the *Simulator* in the case of action *Put down screwdriver on table* associated with the optimal path P_i is shown in Figure 3.2, where it is assumed that in the workspace there are two tables $\{Tab1, Tab2\}$, two screwdrivers $\{SD1, SD2\}$, and two robot agents $\{R1, R2\}$ that can perform all the steps required by the action. In the case of Figure 3.2, there are $2 \times 2 \times 2 = 8$ possible combinations to ground the parameters and allocate the tasks to the agents, represented as 8 branches of the decision tree. We select the branches to simulate using Breadth-first search algorithm. When all the branches are simulated, we compare them according to the utility value of each branch. In this work, utility value is simply a function of execution time as shown in (3.21).

On the basis of (3.21), the *Task Manager* module determines the most appropriate and realistic combination of screwdriver, table and robot maximizing the

utility among all the branches, to finally allocate it to one agent or the other.

If an action execution fails in the decision tree, the *Task Manager* deletes the branch associated to that action. For example, if $Approach(SD1, R1)$ fails in the simulation, the *Task Manager* deletes branches (1) and (2). If $Approach(Tab1, R1)$ fails to perform, it deletes only branch (1). If all the branches are deleted, the simulations imply the robots cannot execute the state transition. Therefore the *Task Manager* sets the feasibility value of the state transition associated to the optimal path P_i to $false$. Later the *Task Manager* get updated, and generates new decision tree for the feasible state transition associated with a new optimal path P_o. In other words, if the utility value of all the decision tree branches becomes zero, the *Task Manager* makes the optimal state infeasible and finds a new optimal state. With this method, the *Task Manager* proactively avoids the cooperation from failure and increases the robustness of the architecture to the failure.

Failures occur whenever a robot does not succeed in executing a given command because of unexpected conditions of the workspace, uncertainty, or the impossibility to meet specific kinematics constraints or safety requirements. In this case, the *Simulator* module can proactively determine the feasibility of actions before they are actually executed and it can suggest the *Task Manager* to allocate actions to human operators if their outcome is uncertain, or inefficient, given the robot capabilities and the current status of the workspace.

we estimate the time and failure/success of the actions in Equation 3.21 by simulating online the robot kinematic behaviour. If the disturbances are low and the robot model is precise, the simulation and real robot behaviour will be similar. Therefore, the estimates in Equation 3.21 is realistic and similar to what will happen in reality. One may simulate the robot in dynamic level, but this option may not provide more information to the user because of dynamic uncertainties in interaction with the environment.

3.2.3 Reactive adaptation

Using the sequences of the actions for all the feasible state transitions with their associated costs, we create a data structure called *Action-State Table* Darvish *et al.* (2018b). This table keeps the information of the grounded optimal state, the progression of the optimal state, and execution of the actions. This table simply chooses the minimum cost cooperation path to follow if possible. However, if the human decides to follow another feasible state of the *Action-State Table*, the *Task Manager* gives priority to the human decision, therefore reactively adapts to the human decisions. For simplicity, in this section we call both hyper-arcs and nodes as states.

Moreover, if the robot/human cannot perform a given action in a certain

amount of time or with a pre-defined quality despite the successful simulation, the *Task Manager* reactively stops the collaborative human/robot from their current responsibilities, makes the optimal state infeasible, and finds a new optimal state among the available ones, so that the system becomes robust to the failures and environment's uncertainties reactively.

Once an executed action has been detected or classified, the executed action label is forwarded to the *Task Manager* module, which identifies the corresponding action a_j and checks whether it appears in the set of actions associated with the currently feasible state. For each state in the *Action-State* table:

- if action a_j appears in the set of actions associated with the state, predicate $done(a_j)$ is set to true and the state is kept;
- if not, the state is marked at *infeasible* and removed from the set of feasible hyper-arcs.

On the basis of the number of feasible states after the above check, the *Task Manager* updates the status of the cooperation:

- If there is only one feasible state, along the current cooperation model M_c, FlexHRC enters a *clear* mode, inferring that the cooperation is proceeding along the optimal path.
- If there is only one feasible state, along a different cooperation model with respect to the current one, FlexHRC enters a *clear* mode and it is inferred that the operator switched to another cooperation model.
- If there are two or more feasible states, FlexHRC enters an *ambiguous* mode and waits for further inputs (i.e., other completed actions) to repeat the check and determine which cooperation path P in G is followed.
- If there are no feasible state, FlexHRC enters a *null* mode, inferring that an unexpected action occurred, and ends the cooperation.

Once all the actions associated with a state have been performed, the state is marked as *solved/met*. Then, the *Task Manager* informs the *Task Representation* module, which updates the AND/OR graph and determines the next suggested action for operators or robots.

Figure 3.3 illustrates the procedure. In the Figure, each row corresponds to an feasible state and the set of actions associated with the states are shown in circles on the right-hand side. Actions assigned to the human operator are shown as red circles, while actions assigned to the robot are shown as blue circles. As an example, the top row shows that actions a_1, a_2, a_3, a_4 compose the set A_1 of

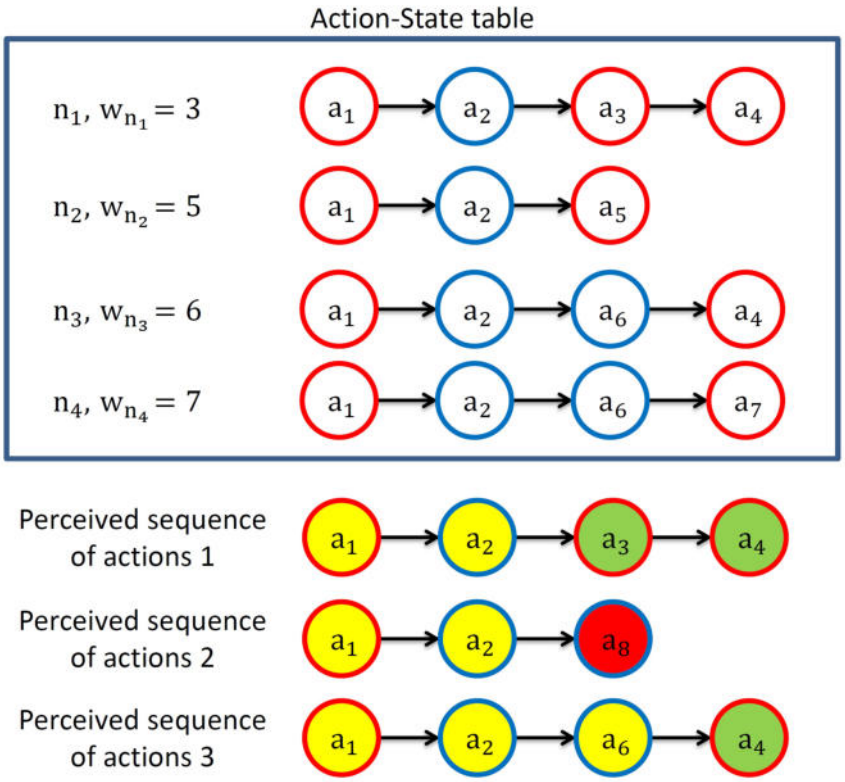


Figure 3.3: Action-State table search and update example: red circles denote actions for which the robot is responsible, while blue circles denote actions for which the human is responsible. Yellow, red and green filling colors denote, respectively, ambiguous, null and clear mode of the table search. $a_1 - a_8$ are labels of actions; $n_1 - n_4$ are labels of feasible states; and $w_{n_1} - w_{n_4}$ denote the weight of each state.

actions associated with state n_1, that operator and robot should perform to solve it.

Let us assume that, at the beginning of the cooperation, the feasible states are those shown in Figure 3.3 and that the state with minimum cost is the top one.

Depending on the sequence of operator and robot actions perceived by the *Human Action Recognition* and *Robot Execution Manager* modules, we have different scenarios, as outlined hereafter:

1. If the sequence corresponds to sequence 1, upon the recognition of operator action a_2, the *Task Manager* commands the robot to execute actions a_3 and a_4, and once the latter is completed the state is marked as *solved/met*. Upon the completion of action a_3, FlexHRC is in *clear* mode and it is

Algorithm 17 PlanningOffline()

Require: A description of actions definition, offline action-state table, agents information, and cooperation task name
Ensure: Create a data structure for online Execution
1: $G \leftarrow \{cooperationTaskName\}$
2: loadAgents()
3: loadActions()
4: loadOfflineAction-State(G)
5: *return*

inferred that the cooperation has followed the optimal cooperation model.

2. If the sequence corresponds to sequence 2, after action a_2, the operator performs action a_8, which does not appear in any of the feasible states. FlexHRC thus enters the *null* mode and ends the cooperation.

3. If the sequence corresponds to sequence 3, after action a_2, the *Task Manager* commands the robot for the execution of action a_3, along the top state, but the operator interrupts its execution by performing action a_6. The first two states become infeasible, and the *Task Manager* switches to state n_3 (which has a lower cost than state n_4) to command the execution of a_4. Upon its completion, FlexHRC is in *clear* mode and it is inferred that the operator has switched cooperation model.

3.2.4 Task manager algorithm

The *Task Manager* has two offline and online phases. Algorithm 17 shows offline phase. Line 3 of the Algorithm 17 loads the description of the actions and possible robots or humans who can perform each action. Line 4 loads the set of sequences of actions for all the states or state transitions of the cooperation. Using the loaded information offline, we create the necessary data structures for the fast online execution. Offline, the user may not provide the grounded parameters of the actions or the responsible individuals to perform the actions.

Figure 3.4 shows the flowchart of the online phase. When the response of the query from the *Task Representation* arrives with the set of feasible states or state transition; *Task Manager* checks first if the cooperation graph is solved successfully. Later, it generates the *Action-State Table* data structure and checks for a met state or a solved state transition in `Check state execution` function; recalling that there might be some state with an empty process set. Afterwards, among the feasible states, we find the optimal state by function `Find optimal state`; we check if the actions in the optimal state are grounded or assigned and

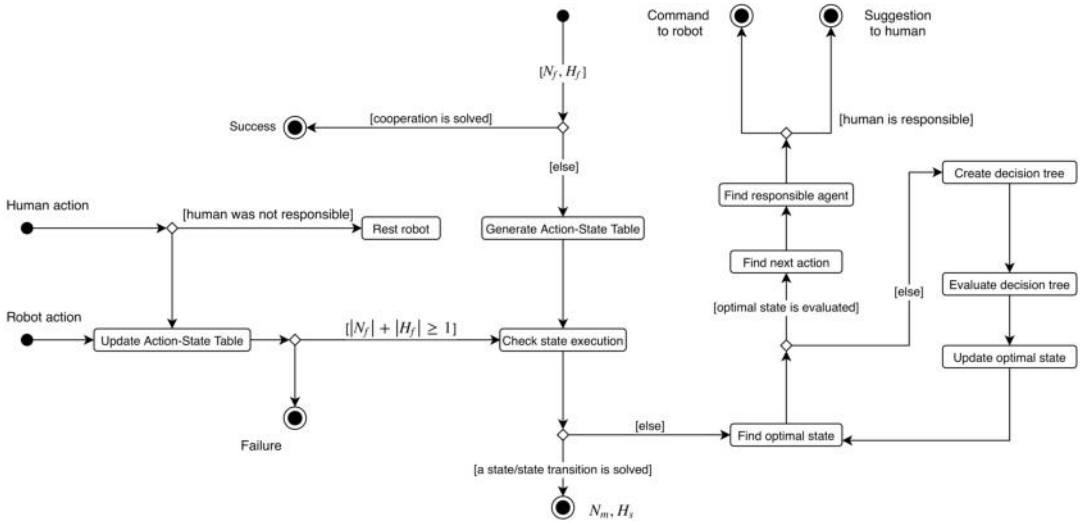


Figure 3.4: The *Task Manager* online phase flowchart.

if the robot can successfully execute the actions in the simulation. To ground the optimal state actions' parameters and assign the actions to the agents, we generate the decision tree in `Create decision tree`, simulate all the actions of the branches, and finally find the utility value for all the branches in `Evaluate decision tree`. Function `Update optimal state` checks for the maximum utility value, grounds the parameters actions, and assign the actions to the agents in the optimal state. If the maximum utility value is zero, `Update optimal state` makes the current optimal state infeasible. Eventually, `Find next action` finds the first action in the optimal state which is not done; and `Find responsible agent` gives the command to the assigned agent.

When the acknowledgement of an action execution arrives (either successful or failed execution), `Update Action-State Table` updates the representation in the Action-State table reactively. In case the human performs an action which *Task Manager* did not give it a priory; we send a command to the robot to drop the current command and go to its *resting configuration*. Finally, if the Action-State table does not hold any feasible state or state transition, the cooperation is failed.

3.3 Knowledge Base

The *Knowledge Base* module maintains and explicitly represents the cooperation state and workspace-related perceptual information using custom data structures Darvish *et al.* (2018a). The module is a data structure that stores information related to the status of the workspace (i.e., the objects therein) and the robot and

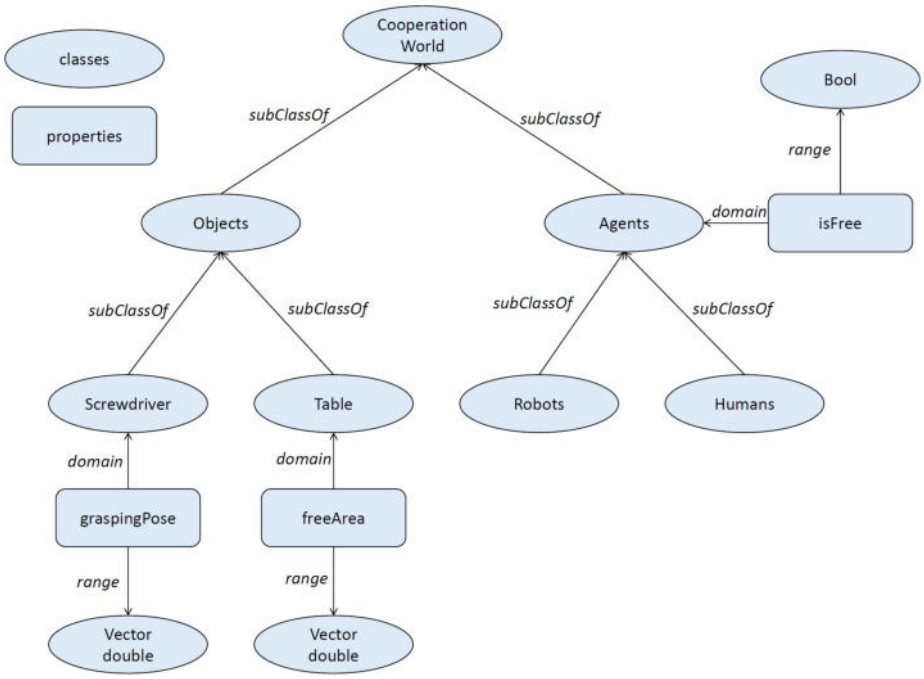


Figure 3.5: The graph of ontology for placing the screwdriver on the table associated with Figure 3.2

the human. Similar to standard knowledge representation, the *Knowledge Base* is consist of classes and properties. The customized knowledge base presented in this work is a simplified ontology in First-Order-Logic for the cooperation process Krötzsch *et al.* (2012). It allows the definition of the concepts or *classes*, individuals or *instances*, and *properties* for the description of the classes features in the cooperation domain Noy *et al.* (2001). The knowledge base tool developed here does not allow for reasoning, while the standard ones such as protégé platform allow Noy *et al.* (2001); Protege (2018).

The ontology of the example shown in Figure 3.2 is presented in Figure 3.5. As shown in this figure, each class can be a subclass of another class. The *domain* of a property defines the type of the input class, while the *range* defines the output type. The graph of ontology is defined offline, whereas the instances of the classes and the value of the range of properties are determined online, coming from the perception level. Different modules may query *Knowledge Base* in two different ways; either they query the instances of the classes (for example in case of decision tree generation), or they query for the values of the properties of some instances.

Chapter 4

Human-Robot Cooperation at Perception Level

Summary

In this chapter, first, we introduce the *Human Action Recognition* module. We demonstrate the pipeline of the module, and we describe the necessary steps to successfully recognize the human actions. Section 4.2 describes the software pipeline for *Object and Scene Perception*. We provide the theoretical description of the steps, in which it enables the robot to manipulate the objects in the workspace.

4.1 Human Action Recognition

As shown in Figure 2.1 (green loop), operator actions affect both the *Human Action Recognition* and the *Task Manager* modules. The former performs *gesture recognition* using inertial data collected at the operator's wrist, whereas the latter determines if the recognized gesture corresponds to an action in hyper-arcs and assesses its effects on the overall cooperation.

The *Human Action Recognition* module (Figure 4.1) employs a system for gesture recognition and classification first described in Bruno *et al.* (2013, 2014). The approach assumes two phases: an offline training phase, where a set $\mathcal{G}$ of $g_1, \ldots, g_{|\mathcal{G}|}$ gesture models are created from a training set of inertial data, and an online phase, where operator motions are classified on the basis of the gesture models in $\mathcal{G}$. After a data filtering step to isolate *gravity* and *body acceleration* as features, the modeling process adopts Gaussian Mixture Modeling (GMM) and Gaussian Mixture Regression (GMR) to compute an *expected* regression curve and the covariance matrix for each $g \in \mathcal{G}$. Once the regression curve for a model

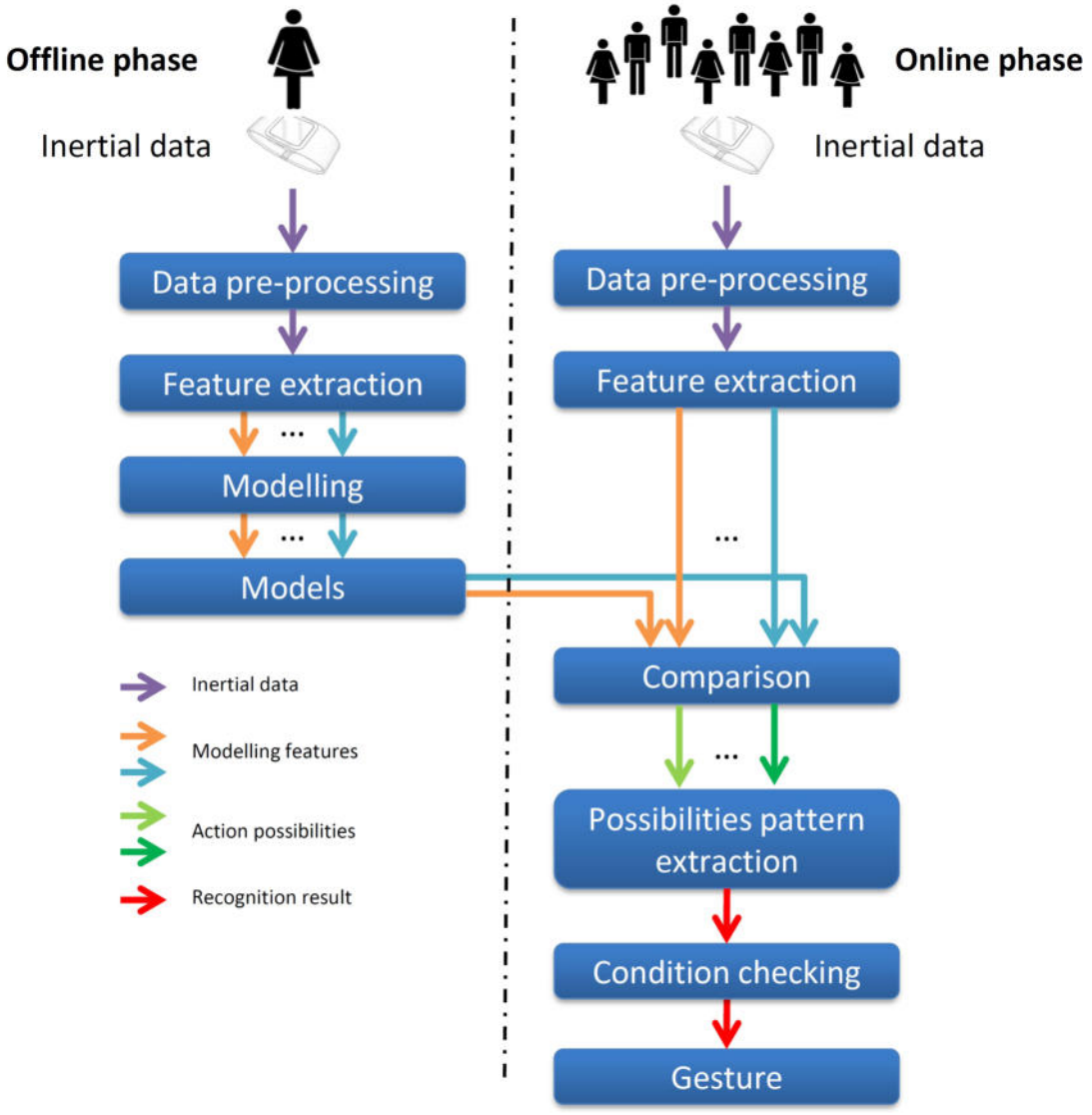


Figure 4.1: A schematic description of the *Human Action Recognition* module.

g is obtained, the number of data points in it needs not to be the same as that in the trials in the training set, which is of the utmost importance to cope with computational requirements in the online phase.

4.1.1 Probabilistic modeling for human action recognition

While the cooperation process unfolds, *Human Action Recognition* executes a number of steps, in part similar to the procedure in the offline phase. Online,

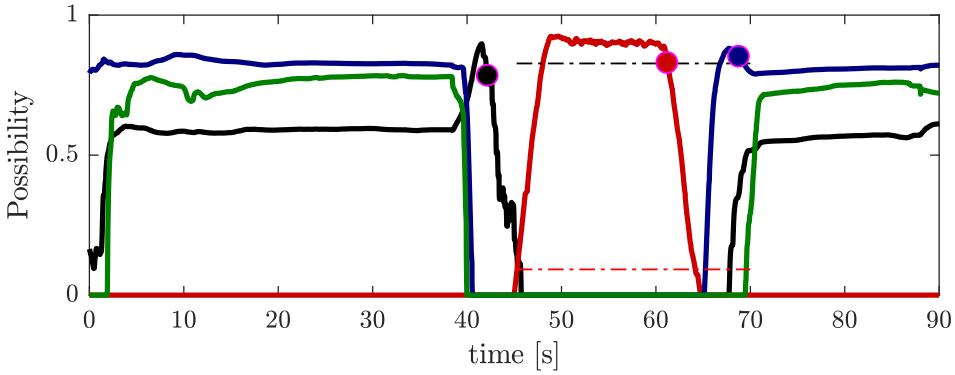


Figure 4.2: An example of action possibility evolutions for a cooperation task.

once inertial data are processed to extract gravity and body acceleration features (typically focusing on a time window depending on gesture model lengths), *gesture recognition* is performed by comparing those features against the models in $\mathcal{G}$, thereby labeling data with a gesture symbol. It is noteworthy that such an approach assumes the operator does not artificially hesitate in performing the gesture. Two distance metrics are adopted, i.e., the well-known Mahalanobis distance and the maximization of the so-called *possibilities*, to take into account the variability associated with gesture models Bruno *et al.* (2014). The Mahalanobis distance is a statistical measure comparing a current data stream and models represented using a regression curve and the associated covariance matrices for each point; however, it does not explicitly take into account the temporal variability associated with gesture execution. A state of the art approach to consider temporal variabilities is Dynamic Time Warping. In previous work Bruno *et al.* (2013), we proposed a metric for gesture classification integrating the Mahalanobis distance and Dynamic Time Warping. Our experiments showed that the increased computational time needed to warp the two signals (the computational complexity of Dynamic Time Warping being polynomial in the data window size) does not provide substantial classification improvements, and therefore we decided to adopt only the Mahalanobis distance in FlexHRC to reduce delays introduced by gesture recognition and classification. As the experiments show, the implicit encoding of temporal differences in the covariance matrices of the gesture models is robust to small variations in the execution of the gestures to the point, in particular, of retaining good recognition performance for the modelled gestures even with users who did not provide recordings for the training set. Possibilities are computed on the basis of Mahalanobis distances, as described in Bruno *et al.* (2014).

In FlexHRC, possibilities are used to determine which gesture has been ex-

ecuted Bruno *et al.* (2014). At a given time instant, a time window contains an inertial data pattern related to a gesture model g. The correlation between the time window and the correct gesture model is maximum when the former is in perfect overlap with the model. Accordingly, the possibility value tends to increase, reaches a peak and decrease afterwards. Figure 4.2 shows possibility values for four gestures, using different colors (see Chapter 6 for a more detailed description). Focusing on the *red* pattern, the associated possibility value is zero for the first 45 seconds, it jumps to reach almost 1, and afterwards it decreases reaching 0 again. There might be small oscillations in possibility values, which might cause local maxima and minima. In our case, a threshold is introduced to find the (semi-global) maximum of the possibility pattern. When, after the peak, the possibility reaches the threshold value (currently set at 90% of the peak possibility value, red dot in the Figure 4.2 at 62 seconds), the corresponding gesture g is considered as executed, subject to the fact that it corresponds to the highest value among all other model possibilities.

Following we describe each of the steps presented in Figure 4.2.

4.1.2 Data pre-processing

For filtering the noise of the acceleration signal a median filter is applied as it outperforms the linear filter in our scenario since the signal-to-noise ratio is high Arias-Castro *et al.* (2009); Bruno *et al.* (2013).

Besides, to model the time series data we need to truncate and synchronize the raw data in offline phase. This process becomes important when we have big data and it is not possible to handle them manually. Also, the synchronization and truncation of the data affects the model we want to generate, and therefore the delay and the precision of the *Human Action Recognition*. The procedure we benefit is valid when the signal-to-noise ratio is high, and data acquisition is done with the same sample rate.

To do so, for an action/gesture g the user finds and truncates a reference data series $ref_g = (g^1, ..., g^T)$ among the available raw data series such that ref_g data series visually is a good representation of the action. The window size of the action T is defined by the user. Later, for all the raw time series data, we find the Euclidean distance between the moving horizon window of the raw data trial and ref_g. We assume the place in which the distance reaches its minimum, the raw time series data is more similar to ref_{g_i} and therefore action g_i is performed. We truncate the raw series data at this point.

4.1.3 Feature extraction

To extract the features of the human movements, a low-pass filter is applied on raw acceleration data (a_x, a_y, a_z) in order to extract the body (b_x, b_y, b_z, t) and gravity accelerations (g_x, g_y, g_z, t) Bruno *et al.* (2012, 2013); Van Hees *et al.* (2013). After the low-pass filter $filter : (a_x, a_y, a_z) \rightarrow (g_x, g_y, g_z)$ is applied, body accelerations are computed as following:

$$(b_x, b_y, b_z) = (a_x, a_y, a_z) - (g_x, g_y, g_z), \tag{4.1}$$

4.1.4 Modeling

GMM and GMR produce a probabilistic model for each motion primitive. Consider we have N motion primitives. For a motion primitive $n \in N$ we have S_n number of training set trials with a window of $(1, ..., T_n)$ sample times. Let $\xi_m^{n,t} \in \mathbb{R}^3$ is a point related to the t'th sample time of trial m of the motion primitive n; and $\xi_m^n \in \mathbb{R}^4 = (\xi_m^{n,t}, t)$ be a generic feature among the ones we have with dimensionality $D = 4$, i.e. the body or gravity acceleration. Later, the set of all samples trails of the motion primitive n is generated as:

$$\Xi^{\xi,n} = \{\xi_1^1, ..., \xi_{T_n}^1, \xi_1^2, ..., \xi_{T_n}^{S_n}, \}, \tag{4.2}$$

as explained before, these trails are synchronized and truncated. Gaussian Mixture Modeling (GMM) and Gaussian Mixture Regression (GMR) methods return $\hat{\Xi}^{\xi,n} = (\hat{\xi}^n, \hat{\Sigma}^n)$; where $\hat{\xi}^n$ is the expected curve for modeling the feature ξ of motion primitive n and $\hat{\Sigma}^n$ is the covariance matrix associated with $\hat{\xi}^n$.

4.1.4.1 Gaussian Mixture Modeling

Assume the dataset ξ^n of all trials of motion primitive n, can be modelled by K Gaussian distributions with dimension D by Cohn *et al.* (1996):

$$p(\xi_t|k) = N(\xi_t; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}(\xi_t - \mu_k)^T \Sigma_k^{-1} (\xi_t - \mu_k)}, \tag{4.3}$$

where μ_k is the expected value and Σ_k is the covariance matrix of k'th Gaussian distribution. In (4.3), $p(\xi_t|k)$ represents the conditional probability of a variable ξ_t with respect to normal distribution $N(\mu_k, \Sigma_k)$. The number of Gaussian distributions K is found by Expectation Maximization (EM) algorithm Dempster *et al.* (1977). The prior probability of k'the normal distribution is defined by:

$$p(k) = \pi_k, \tag{4.4}$$

Therefore, the probability density functions is given by:

$$p(\xi_t) = \sum_{k=1}^{K} p(k)p(\xi_t|k), \tag{4.5}$$

Similarly, we can represent the mean value and covariance matrix by:

$$\begin{aligned} \mu_k &= \{\mu_{t,k}, \mu_{s,k}\}, \\ \Sigma_k &= \begin{pmatrix} \Sigma_{t,k} & \Sigma_{ts,k} \\ \Sigma_{st,k} & \Sigma_{s,k} \end{pmatrix}, \end{aligned} \tag{4.6}$$

where $\mu_{t,k}$ and $\mu_{s,k}$ are the time step and features or spatial mean values of k'th Gaussian model, and $\Sigma_{ts,k}$ is the cross-correlation of the temporal and spatial features.

4.1.4.2 Gaussian Mixture Regression

GMR returns a smooth generalized trajectory associated with K Gaussian distributions and covariance matrices Calinon *et al.* (2007). For each Gaussian distribution k, the conditional expectation and covariance of $\xi_{s,k}$ given ξ_t are found by:

$$\begin{aligned} \hat{\xi}_{s,k} &= \mu_{s,k} + \Sigma_{st,k}(\Sigma_{t,k})^{-1}(\xi_t - \mu_{t,k}), \\ \hat{\Sigma}_{s,k} &= \Sigma_{s,k} - \Sigma_{st,k}(\Sigma_{t,k})^{-1}(\Sigma_{ts,k}), \end{aligned} \tag{4.7}$$

To mix $\hat{\xi}_{s,k}$ and $\hat{\Sigma}_{s,k}$ associated k'th Gaussian distributions, prior β_k finds the probability that k'th Gaussian distribution is responsible for ξ_t:

$$\beta_k = \frac{p(\xi_t|k)}{\sum_{j=1}^{K} p(\xi_t|j)}, \tag{4.8}$$

As a result the conditional expectation and covariance of ξ_s given ξ_t is computed by:

$$\begin{aligned} \hat{\xi}_s &= \sum_{j=1}^{K} \beta_k \hat{\xi}_{s,k}, \\ \hat{\Sigma}_s &= \sum_{j=1}^{K} \beta_k^2 \hat{\Sigma}_{s,k}. \end{aligned} \tag{4.9}$$

4.1.5 Comparison

Online, when the new data arrives we perform the data pre-processing and feature extraction on them. We consider a *window* of moving horizon acceleration data

with length N_w equal to the longest modeled data $\arg\max_{n \in N} T_n$. Mahalanobis distance provides a similarity value or rank between the model and incoming acceleration data Later De Maesschalck *et al.* (2000); it is computed as the distance between the model $\hat{\Xi}^{\xi,n}$ and online features Ξ^{ξ}:

$$d_M^{\xi}(r,j) = \sqrt{(\hat{\xi}_r^n - \xi_{j,w})^T (\hat{\Sigma}_r^n)^{-1} (\hat{\xi}_r^n - \xi_{j,w})}, \tag{4.10}$$

where r is an element of the model and j is an element of online feature. When the distance is found for different elements, we compute the distance for the body and gravity features using the mean value of distances for each feature. The possibility of the execution of an action by the human is computed accordingly.

4.1.6 Possibilities pattern extraction

Algorithm 18 receives as input the modelled actions possibilities value; when an action has been recognized returns the tag of the action otherwise returns -1. Recognizing the global maximum in real-time is hard, as there are local maximum and minimums in the pattern of the possibilities as well as the correlation between the possibilities of different actions; in fact, in some cases, the possibilities arise and decline together as shown in Figure 4.2.

Line 1 in Algorithm 18 stores the highest possibility of action i in variable $highestPoss[i]$ starting from the last moment an action has been recognized. In line 2-3, the variables $MaxPoss$ and $MaxPossElement$ store the maximum possibility value among all the actions and the tag of the action which owns $MaxPoss$ from the last moment an has been recognized. Instead, lines 4-5 store similar information related to the last recognized action. In Algorithm 18, there are two parameters to tune for recognizing when the human has performed an action, i.e. when the possibility of an action i reaches its (semi) global maximum and the possibility value of action i is higher than other actions. The tuning parameters are $MaxThreshold[]$ and $MinThreshold[]$ which are defined at lines 6-7. These thresholds define when the possibility pattern reaches its (semi) global maximum or minimum. We call them (semi) global, as these extremum values are neither global nor the local extremums. As shown in Figure 4.2, there might be small oscillations in the possibility values that might cause local extrema. To overcome this issue, a threshold is introduced to find the (semi) global maximum of the possibility pattern. If this threshold is too low, the delay of human's action recognition will be high. In Figure 4.2, this threshold is shown by the black dotted line.

In Line 8, $ActionsNo$ is the number of offline modelled actions. Lines 9-23 update the highest possibility value of an action i. If an action i is not the same as the last recognized action, $PrevMaxPossElement$, it simply updates

the $highestPoss[i]$; otherwise the algorithm starts updating $highestPoss[i]$ if the possibility of i goes less than the minimum threshold or higher than the maximum threshold (lines 14-22). Once an action has been recognized $PrevMaxPossElement$, the current possibility value of this action might be higher than other actions possibility. To prevent multiple times the recognition of the same action, the highest possibility value of $highestPoss[PrevMaxPossElement]$ is maintained zero until either it is lower than a pre-defined threshold (red dotted horizontal line in Figure 4.2 for the action with red colour), or it is higher than $PrevMaxPoss$.

Later in lines 24-27, the algorithm updates the possibility value and the action tag with maximum possibility. Lines 28-35 check if possibility pattern related to action i reaches its maximum and starts decreasing more than $MaxPoss \times MaxThreshold[i]$; then it returns the tag i as the recognized action.

4.1.7 Condition checking

To improve the robustness of the human action recognition, the module exploits the knowledge of the sequence of actions that are allowed or expected to perform. For example, consider Action B that may be performed only if action A has been performed before. Online, if *Possibilities pattern extraction* recognize actions C and then B; the *Condition checking* filter out the recognized action B and does not return it to *Task Manager*.

4.2 Object and Scene Perception

Figure 4.3 shows the pipeline of the *Object and Scene Perception* module. The module receives the RGB-d data using a *Kinect* sensor; in *pre-processing* step, it performs downsampling to decrease the number of point clouds and enhance the computational efficiency. Moreover, a depth filter is applied in pre-processing step to remove the irrelevant data, such that it only keeps the point clouds inside a sphere centred at Kinect sensor and therefore decrease more the computational time in next steps of the module Buoncompagni & Mastrogiovanni (2015).

In the *clusterization* step, we find the support of the objects in the workspace. We assume the support is a horizontal plane in the workspace of the robot. To find the horizontal plane we apply iteratively Random Sample Consensus (RANSAC) method Fischler & Bolles (1981); Schnabel *et al.* (2007). Later, the point cloud belong to support is deducted from the set of point clouds; and a Euclidean distance clustering algorithm is applied to cluster the remained point cloud. The output of this step will be the geometrical information of the support plane and several clusters of point cloud associated with the objects in the workspace.

Algorithm 18 Possibilities pattern extraction()

Require: Vector of current possibility values $NewPoss[i], \forall i > 0 \in ActionsNo$
Ensure: Tag of an action when human has performed

$highestPoss[] \leftarrow 0$ ▷ *Poss* stands for possibility
$MaxPoss \leftarrow 0$
$MaxPossElement \leftarrow NULL$
$PrevMaxPoss \leftarrow 0$ ▷ *Prev* stands for previous
$PrevMaxPossElement \leftarrow NULL$
$MaxThreshold[]$ ▷ defined by user
$MinThreshold[]$ ▷ defined by user
for all $i \in ActionsNo$ **do**
  if $i \neq PrevMaxPossElement$ **then**
    if $NewPoss[i] > highestPoss[i]$ **then**
      $highestPoss[i] \leftarrow NewPoss[i]$
    end if
  else
    if $highestPoss[i] = 0$ **then**
      if $NewPoss[i] < MinThreshold[i] \times PrevMaxPoss$ Or $NewPoss[i] > PrevMaxPoss$ **then**
        $highestPoss[i] \leftarrow NewPoss[i]$
      end if
    else
      if $NewPoss[i] > highestPoss[i]$ **then**
        $highestPoss[i] \leftarrow NewPoss[i]$
      end if
    end if
  end if
  if $highestPoss[i] > MaxPoss$ **then**
    $MaxPoss \leftarrow highestPoss[i]$
    $MaxPossElement \leftarrow i$
  end if
  if $i = MaxPossElement$ **then**
    if $NewPoss[i] < MaxPoss \times MaxThreshold[i]$ **then**
      $PrevMaxPoss \leftarrow MaxPoss$
      $PrevMaxPossElement \leftarrow MaxPossElement$
      $highestPoss[] \leftarrow 0$
      return i
    end if
  end if
end for
return -1

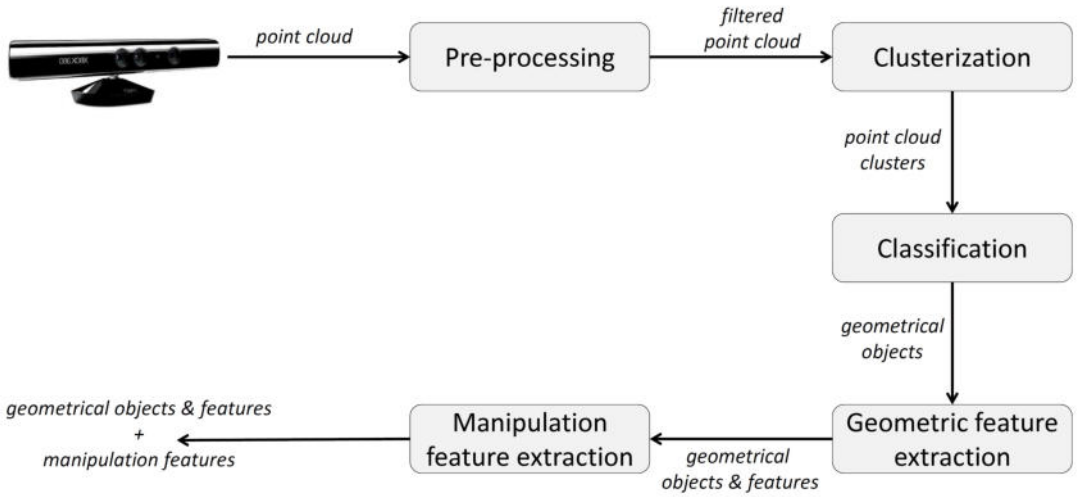


Figure 4.3: A schematic description of the *Object and Scene Perception* module.

In *classification* step, RANSAC method for different primitive objects including plate, sphere, cone, and cylinder is applied to point cloud clusters. In *geometric feature extraction* a Principal Component Analysis (PCA) Huang *et al.* (2009); Wold *et al.* (1987) is applied in order to attain other geometrical features of the objects in the workspace which is not possible with RANSAC method. Finally, in *manipulation feature extraction*, we use the geometrical features of the primitive objects found online and some offline information of the objects provided by the user to compute the frames of the objects for manipulation tasks such as grasping frames, connection frames. Moreover, we compute the bounding box and bounding sphere of the objects for path planning and motion planning.

Later in this section, we introduce the Euclidean distance clustering method, RANSAC method, and PCA method, and how we use the incoming information for manipulation scenarios.

4.2.1 Euclidean clustering

Let p be a point of the point cloud set P. Using Euclidean distance clustering method we generate a k-d tree to divide the P into the smaller sets of cluster C. It assigns a point $p \in P$ to a cluster $c \in C$ if the Euclidean distance between p and $\forall p_i \in c$ is less than the threshold r Rusu (2009); Tuerker (2018). In other words, the minimum distance between two points in distinct clusters c_1 and c_2 should be more than a given threshold.

4.2.2 RANSAC method for classification

In RANSAC method, we assume a point cloud belonging to a cluster $P_c = \{\forall p | p \in c\}$ is consist of inliers and outliers O'Leary (2018). Inliers $P_c^{inliers}$ fit a model with a set of unknown parameters considering the noises while the outliers $P_c^{outliers}$ do not fit the model.

Using RANSAC method we assume P_c belongs to one of the primitive geometrical object model set including $Models = \{cylinder, sphere, plane, cone\}$. If the number of inliers is less than a specific threshold it returns as *unknown* object. To fit the points of a cluster P_c to a model m with a set of n unknown parameters $w_1, ..., w_n$, we choose randomly the minimum number of necessary points to determine all the parameters of m. We assume all these randomly selected points belong to inliers and then we use them to fit a model to P_c. After finding the parameters, we examine how well the rest of points in P_c fit to the model we found. If a point $p^i \in P_c$ fits the model we hypothetically consider the point as an inlier. We try this for all the points in the P_c, and if the model m is a good fit of the cluster ($|P_c^{inliers}|\backslash|P_c|$ is high), we update the model parameters estimation with the set of all the inliers. Finally, among all the found parameters for model m, we choose the one with highest $|P_c^{inliers}|\backslash|P_c|$ value. We perform the same procedure for all the models inside the models set. The output of the classification step will be a model tag with its associated parameters which fits the best to the cluster. If the value $|P_c^{inliers}|\backslash|P_c|$ is less than a specific threshold for all the models, it returns *unknown* as the output.

4.2.3 Principal Component Analysis (PCA) method for feature extraction

The outputs of the classification problem are the geometrical object models and their associated parameters. In order to manipulate the objects, using only these parameters is not sufficient. For example, for a plane in space with the formula:

$$ax + by + cz + d = 0, \tag{4.11}$$

the classification outputs will be the parameters of the plane $\{a, b, c, d\}$, and the mean value of the inlier set P_c^{inlier}. To manipulate an object with a planar shape, we need to know at least the size of the plane and their directions or the vertices of a plane. In the case of cylinder and cone, we do not have the information about the height of them. To find, we apply the PCA method to the set of inliers of each classified cluster, considering their recognized geometrical features. In the case of the plate, we find the first and second principal components, map all the inliers on these two principal components to find the plate size, and accordingly their vertices.

To do so, we find the mean value $\mu \in \mathbb{R}^D$ and covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$ of the n inliers $X \in \mathbb{R}^{n \times D}$, where $D = 3$. Later, k eigenvectors of the Σ, which shows the axis with the maximum variability, is found by:

$$\Sigma v_i = \lambda_i v_i, \quad i \in 1, ..., k \tag{4.12}$$

where λ_i is the i'th eigenvalue of the of Σ and $k < D$. Finally, the data with reduced dimensionality is represented by $z = U_{red} \times x$, where U_{red} is:

$$U_{red} = \begin{bmatrix} v_1^T \\ v_2^T \\ ... \\ v_k^T \end{bmatrix} \in \mathbb{R}^{k \times n}, \tag{4.13}$$

and $x \in X$ and $z \in Z$ are the original data and reduced dimensional data.

4.3 Objects manipulation

To manipulate objects, such as transporting an object from one point to another one or screwing an object into another one, we need to know the position and orientation of some specific frames of the object being manipulated. However, the perception system is able to estimate the position and orientation of a frame attached to the object, which, in general, does not coincide with the ones needed for the manipulation task. For example, as shown in Figure 4.4, let us assume that for executing a manipulation task the robot needs to know the position of the reference frame $\langle M \rangle$ attached to the body of the object obj_1. However, the perception system can estimate only the position and orientation of the frame $\langle O \rangle$, attached to the object body.

To overcome this difficulty, the user provides the information of the manipulation frame $\langle M \rangle$ with respect to $\langle O \rangle$, therefore ${}^{O}_{M}T$ the homogeneous transformation matrix in the coordinate system attached to obj_1 body. As a result, if the robot can estimate the position and orientation of the frame $\langle O \rangle$ with respect to the world frame subject to obj_1, then the position and orientation of the manipulation frame $\langle M \rangle$ in the robot's world frame is computed by:

$${}^{W}_{M}T = {}^{O}_{M}T \times {}^{W}_{O}T. \tag{4.14}$$

Moreover, using the information coming from the *Geometric feature extraction* we define the bounding boxes and bounding spheres for all the objects, taking into account some user pre-defined safety factor, to ensure the collision avoidance.

In order to manipulate the objects, the robot should grasp them such that it can perform the given task. The problem of grasping depends on the object geometrical features, the manipulation task given to robot, the robot configuration,

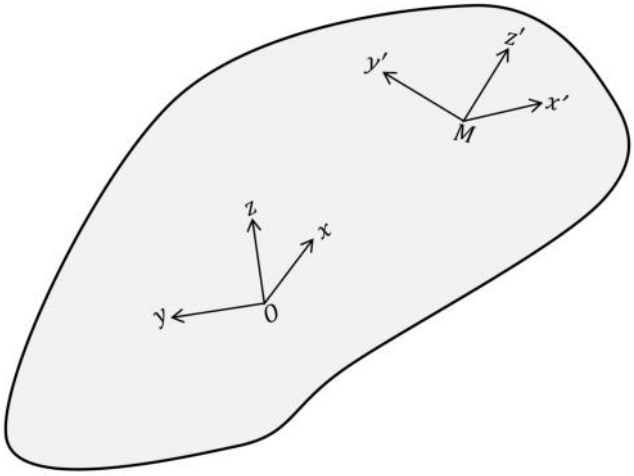


Figure 4.4: A schematic of object body frame $\langle M \rangle$ required for a specific manipulation task and center of volume of the object $\langle O \rangle$.

and the robot end-effector Miller & Allen (2004); Murray (2017). A simulation-based planning for computing the grasping pose is applied in Miller & Allen (2004). They reconstruct the simulated world using perception information; plan for the grasping pose, analyze, and finally execute it. Recently, learning methods are applied for scalable grasping pose computation, in which the robot learns how to grasp an object online, by interacting with objects in the environment during the training sessions Calandra *et al.* (2018); Levine *et al.* (2018); Morrison *et al.* (2018). After computing the grasping pose of an object, a visual servoing approach is applied widely in the literature Chaumette & Hutchinson (2006, 2007); Horaud *et al.* (1998) to successfully grasp the object.

Besides, to perform the visual servoing, the robot should track the object while approaching it. Successful visual servoing depends on the position of the camera on the robot and robot configuration, such that occurrence rate of camera occlusion is low.

In this work, we use a passive method for grasping the object; which means the visual perception is updated once at the beginning of approaching the object (using the *Object and Scene Perception*), we give the desired goal to *Task Execution Manager* and therefore *Controller*, and the error at each moment is getting updated in the *Controller* with this assumption that the object does not move while the robot is going to grasp.

Let us assume a manipulation scenario, where the robot grasp an object and transport it to another position to perform a given task. While transporting, the grasped object may slip in the robot end-effector, in case the holding force is not sufficient. On the other hand, the robot cannot exceed certain holding force

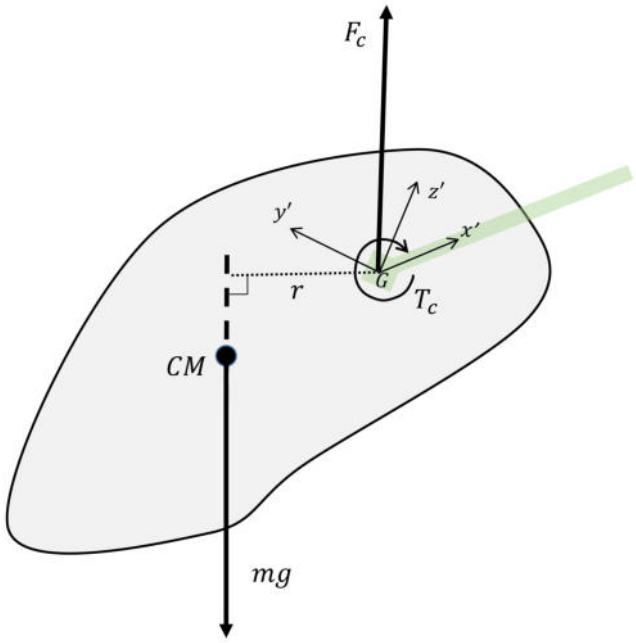


Figure 4.5: A schematic of a grasped object by a two-finger gripper (shown in green colour); CM is the centre of mass of the object with weight mg, G is the grasping position of the end-effector with contact force and torque F_c and T_c.

thresholds, because of the limited tension the object can afford. Therefore, the grasping pose should be computed in such a way that the object's local stability increases, i.e. the object does not move or slip involuntarily with respect to robot end-effector while grasped. As shown in Figure 4.5 by reducing r, the necessary object holding torque decreases. Conversely, the grasped object may act like an inverted pendulum, considering the robot end-effector movement and velocity to reach the desired goal, it becomes unstable while grasped.

While the robot transports the grasped object to another point, it may slip. To overcome this problem and ensure the manipulation task succeeds, it is necessary to compute the error between the grasped object manipulation frame $\langle M \rangle$ and the desired frame in workspace $\langle M_{des} \rangle$.

In our setup, we use a simple gripper with two fingers with a limited amount of holding force without the possibility to control it actively (therefore grasped object may slip while transporting). To minimize the slip and increase the rotational movement stability of the grasped object with single arm, we plan to grasp the object such that the arm length r (Figure 4.5) between the centre of contact (the position the end-effector grasp the object, point G in Figure 4.5) and the gravity vector passing by centre of mass CM minimized. We assume the centre of mass and the centre of volume of the objects are the same, therefore the object

mass distributed identically for the manipulating object. By grasping the object above the CM the object behaves like a pendulum while if G is below the CM the object will act like an inverted pendulum.

Chapter 5

Human-Robot Cooperation at Action Level

Summary

This chapter describes the FlexHRC at action level. The action level is consist of four modules, namely *Robot Execution Manager*, *Path Planner*, *Controller*, and *Simulator*. For each of these modules a section is dedicated accordingly.

5.1 Robot Execution Manager

The *Robot Execution Manager* enhances the scalability of the FlexHRC by a modularized implementation of different actions. When a new action is defined symbolically at the representation level, at the action level we should determine how the robot should perform the action. This module is responsible for turning high-level action commands, received from the *Task Manager*, into commands that the robot *Controller* can execute, i.e., it performs a symbolic-to-numerical mapping to the controller-level representation and checks the success/failure of an executed action. Some actions are simple and it is sufficient to call a control interface, e.g. *approaching an object*, while others might be more complicated, such as *screwing* a part to another part. The complicated actions should be chunked to a number of simple ones or should be learned via learning by demonstration or the robot interaction with the environment given a goal by the user Argall *et al.* (2009).

While the current implementation does not support the robot learning, it provides the necessary modularity for the scalability of FlexHRC. When a command arrives from *Task Manager*, it can be a command for calling the *Simulator*, *Controller*, or the *Path Planning*. In case it is a simulation command, it responds

the given command by success/failure and attributes of the executed action in simulation such as the time and configuration values of the robot after performing the given action. In case it is a path planning command, it calls the *Path Planner* module to find a path; if such a path exists, it stores the path in *Knowledge Base*. Finally, if the given command is an execution command, *Robot Execution Manager* calls the *Controller* after chunking the given action into simpler ones.

5.2 Robot Path Planning

The *Path Planner* computes a feasible path for the movements of a robot's end-effectors, mobile robot platform, or the transported objects carried by a robot; without caring for the robot kinematic or dynamic limitations. The path planner receives the working space region, the final goal region, the obstacle regions described as boxes or cubes, the transported object cube size, and the initial position of the path from the *Robot Execution Manager*.

The Rapidly-Exploring Random Tree Star (RRT*) algorithm presented by Karaman & Frazzoli (2011) finds the obstacle-free path for a point which represents a robot or mobile platform. In this work, we have extended the algorithm to Extended-RRT*, such that it finds the path for a box/cube which represents the robot or the mobile platform. There are two profits in Extended-RRT* with respect to its original version: i) when the object or robot size with respect to the workspace and obstacle size is considerable, it fits better the reality, i.e., considering the object/robot size instead of increasing the safety factor of all the obstacles for collision avoidance; ii) sometimes to find a path for reaching the goal region, the object/robot orientation may need to change. While we can define the orientation for a cube or box in space, it is not possible for a point in space.

The path for the object found by RRT* method converges to an asymptotically global optimal solution; such that it finds the minimum distance path from the object initial point to its goal region Karaman & Frazzoli (2011). The optimality of RRT* method is highly tied with the intuitiveness and intelligibility and safety of the robot behaviour as described in Chapters 1 and 2. Moreover, RRT* method provides *probabilistic completeness*; in which the failure probability of the algorithm for finding a solution if one exists goes to zero by increasing the number of samples to infinity Karaman & Frazzoli (2011).

The problem we want to solve in this section is to find the points for the center of the robot/object in the workspace region. As mentioned before, Extended-RRT* is an extension of the RRT* method Karaman & Frazzoli (2011); therefore its formulation of the solution and the algorithms is very similar to the seminal paper of Karaman & Frazzoli (2011) with some changes. Finally, the path planning module is embedded in ROS.

5.2.1 Path Planning Formulation

Let X be an open subset of $n \in \mathbb{N}$ dimension real space $\mathbb{R}^n$ subject to $2 \leq n \leq 3$ and $cl(X)$ is the closure of X. $X_{ws} \subset X$, $X_{obs} \subset X_{ws}$, $X_{free} \subset X_{ws}$, and $X_{goal} \subset X_{free}$ are the workspace region, obstacle region, obstacle-free region, and goal region respectively. Let $X_{rob} \subset X_{ws}$, $x_{rob} \in X_{ws}$, and $d \in \mathbb{R}^n$ be the robot/object region, position of the center of robot/object in the workspace, and the robot dimension which we want to find the path. Obstacle-free region is found by $X_{free} = \{\forall x_{rob} \in X_{ws} | X_{rob} \cap X_{obs} = \emptyset\}$; which makes the main difference with respect to the original RRT* method. Initially, the robot center x_{rob} is located in $x_{init} \in X_{free}$. The path planning objective is to find a collision free path for the robot which starts at x_{init} and finishes at X_{goal}. Such a path is presented by continious function $\sigma : [0, s] \rightarrow X_{free}$; such that $\sigma(0) = x_{init}$ and $\sigma_s \in X_{goal}$.

A directed graph $G = (V, E)$ on X_{free} is composed of finite set of vertices $V \subset X_{ws}$ and edges $E \subset V \times V$. A directed path p on G is a sequence of vertices $(v_1, v_2, ..., v_m)$ on X_{free} such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq n-1$, and p_i is i'th element of the path. For vertex $v \in V$, the set $\{\forall u \in V | (u, v) \in E\}$ is called incoming neighbors, and the set $\{\forall u \in V | (v, u) \in E\}$ is called outgoing neighbors. The directed tree is a directed graph such that all the vertices but one has only one incoming neighbor. The excepted one does not have any incoming neighbor which is called the *root* vertex. The vertices with no outgoing vertex is called *leaves*. In edge $e = (u, v)$, vertex u is the *parent* of vertex v. The result of the Extended-RRT* is a tree G on X_{free} if such a path exists (*feasible path*). A feasible path $(v_1, ..., v_m)$ starts at root of the graph $v_1 = x_{init}$ and the leaf ends in $v_m \in X_{goal}$. Moreover, the Extended-RRT* ensures the optimality of returned path (if such a path exist) $\sigma^* : [0, s] \rightarrow cl(X_{free})$ where the cost of path will be minimum $c(\sigma^*) = min_{\sigma \in \Sigma_{cl(X_{free})}} c(\sigma)$ thanks to a procedure called *rewiring*.

5.2.2 Path Planning Algorithm

Algorithm 19 describes the overall procedure of the Extended-RRT* algorithm which is similar to the other incremental sampling-based path planning methods such as Rapidly-Exploring Random Tree (RRT) and Rapidly-Exploring Random Graph (RRG) Karaman & Frazzoli (2011); Kuffner & LaValle (2000); LaValle & Kuffner Jr (2001). Lines 1-2 generate the vertices and edges sets of the graph G. It initializes the vertex set V with the initial position of the robot x_{init} and the edge set as an empty set. In lines 4-5, P is the set of all the feasible paths and p is the feasible path with minimum cost; they are initialized as an empty set and an empty sequence. At each iteration, lines 6-11, the algorithm updates the graph G, samples an independent identically distributed random value x_{rand} on X_{free}, and extends the graph G with the samples value using

Algorithm 19 Body of Extended-RRT*()

Require: A workspace region X_{ws}, obstacle regions X_{obs}, goal region X_{goal}, robot initial region X_{rob} and x_{init}
Ensure: A path p
1: $V \leftarrow \{x_{init}\}$
2: $E \leftarrow \emptyset$
3: $i \leftarrow 0$
4: $P \leftarrow \emptyset$
5: $p \leftarrow ()$
6: **while** $i < n$ **do**
7: $\quad G \leftarrow (V, E)$
8: $\quad x_{rand} \leftarrow \mathsf{Sample}(i)$
9: $\quad i \leftarrow i + 1$
10: $\quad (V, E) \leftarrow \mathsf{Extend}(G, x_{rand})$
11: **end while**
12: $G \leftarrow (V, E)$
13: $P \leftarrow \mathsf{feasiblePaths}(G)$
14: $p \leftarrow \mathsf{optimalPath}(P)$
15: *return* p

function $\mathsf{Extend}(G, x_{rand})$. When the number of iteration reaches its limit n, graph G is updated in line 12. Algorithm 20 describes function $\mathsf{Extend}(G, x_{rand})$. In line 13, function $\mathsf{feasiblePaths} : (G) \rightarrow P$ finds all the feasible paths:

$$\mathsf{feasiblePaths}(G) = \{\forall p \in G | p_1 = x_{init} \text{ and } p_{|p|} \subset X_{goal}\}, \tag{5.1}$$

and line 14 finds the path with minimum cost among them:

$$\mathsf{optimalPath}(P) = argmin_{p \in P}\ \mathsf{Cost}(p_{|p|}), \tag{5.2}$$

Function $\mathsf{Cost}(v)$ is the cost of the unique path from the root vertex x_{init} to vertex v; and for the root vertex it is equal to zero $\mathsf{Cost}(x_{init}) = 0$. Finally, if the returned path p at line 15 is an empty sequence, it means the path planning fails to find a feasible path by n iterations.

In Algorithm 20, lines 1-2 store the vertex and edges values of graph G on a new variables V' and E'. In Line 3, function $\mathsf{Nearest} : (G, x) \rightarrow v$ finds the closest vertex $x_{nearest} \in V$ to the sampled random variable x. The distance measurement in this work is the Euclidean distance, and therefore:

$$\mathsf{Nearest}(G = (V, E), x) = argmin_{v \in V} ||x - v||, \tag{5.3}$$

Line 4 calls the *steering* function to find x_{new}. Basically, the function $\mathsf{Steer} : (x, y) \rightarrow z$ returns a point $z \in \mathbb{R}^d$ such that z is closer to y than x. In this work, the function is:

$$\mathsf{Steer}(x, y) = argmin_{z \in \mathbb{R}^d, ||z-x|| < \eta} ||z - y||, \tag{5.4}$$

where $\eta > 0$ is a predefined value. Function $\mathsf{ObstacleFree} : (x, y) \rightarrow bool$ in line 5 tests the collision between the line connecting the two points with the obstacles in space. It returns *true* iff the line segment between the two points are in X_{free}, i.e., $[x, y] \subset X_{free}$. Lines 6 adds the new point x_{new} as a vertex of the graph to V'; and line 7 stores $x_{nearest}$ value on a new variable called x_{min}. Later, x_{min} will be updated as the parent vertex of x_{new}. Function $\mathsf{Near} : (G, x_{new}, |V|) \rightarrow X_{near}$ in line 8 returns a set of $|V|$ vertices close to x_{new}:

$$\mathsf{Near}(G, x, n) = \{v \in V |\ ||v - x|| < r(n)\}, \tag{5.5}$$

where $r(n)$ is the radius of a ball with the center x.

Lines 9-16 update the parent vertex of x_{new} using the vertices X_{near} found before. Line 10 checks for collision between $[x_{near}, x_{new}]$; and line 11 finds the cost c' of vertex x_{new} arriving from x_{near}. Function $\mathsf{Line}(x, y) : [0, s] \rightarrow X_{free}$ returns the path between x and y, and $s = ||x-y||$. In line 11, $c(\mathsf{Line}(x_{near}, x_{new}))$ returns the cost between the two points x_{near} and x_{new}. Finally, lines 12-14 find the parent vertex (x_{min}) of vertex x_{new} by minimizing cost of c'. Line 17 adds the new edge (x_{min}, x_{new}) to the set of edges E'.

Lines 18-25 perform the *rewiring* procedure; in which it checks the connections of the vertices in the set of $X_{near} \backslash x_{new}$ in order to minimize the cost of reaching a vertex x_{near}. To do so, it checks if the connection between x_{near} and x_{new} is collision free and the cost of going to x_{near} by connecting it to x_{new} is less than cost of x_{near}. If so, it returns the parent vertex x_{parent} of the x_{near} using function $\mathsf{Parent}(x_{near}) : v \in V \rightarrow v \in V$; then eliminates the edge (x_{parent}, x_{near}) from the edges set E', and adds (x_{new}, x_{near}) to E'. Finally, line 27 returns the updated graph G'.

After Extended-RRT* returns the sequence of way-points between initial position and goal region of the robot, we interpolate between the initial orientation and desired orientation of the robot for finding the corresponding object orientation at each way-point to relax for a smooth transition.

5.3 Robot Controller

This Section describes the Task Priority Inverse Kinematics framework Simetti & Casalino (2016); Simetti *et al.* (2018) integrated within the *Controller* module

Algorithm 20 Extend()

Require: A workspace region X_{ws}, obstacle regions X_{obs}, goal region X_{goal}, a graph G, and a random sample x
Ensure: An updated tree G

$V' \leftarrow V$
$E' \leftarrow E$
$x_{nearest} \leftarrow \mathsf{Nearest}(G, x)$
$x_{new} \leftarrow \mathsf{Steer}(x_{nearest}, x)$
if $\mathsf{ObstacleFree}(x_{nearest}, x_{new})$ **then**
    $V' \leftarrow V' \cup x_{new}$
    $x_{min} \leftarrow x_{nearest}$
    $X_{near} \leftarrow \mathsf{Near}(G, x_{new}, |V|)$
    for all $x_{near} \in X_{near}$ **do**
        if $\mathsf{ObstacleFree}(x_{near}, x_{new})$ **then**
            $c' \leftarrow \mathsf{Cost}(x_{near}) + c(\mathsf{Line}(x_{near}, x_{new})$
            if $c' < \mathsf{Cost}(x_{new})$ **then**
                $x_{min} \leftarrow x_{near}$
            end if
        end if
    end for
    $E' \leftarrow E' \cup (x_{min}, x_{new})$
    for all $x_{near} \in X_{near} \backslash \{x_{min}\}$ **do**
        if $\mathsf{ObstacleFree}(x_{new}, x_{near})$ and
$\mathsf{Cost}(x_{near}) > \mathsf{Cost}(x_{new}) + c(\mathsf{Line}(x_{new}, x_{near}))$ **then**
            $x_{parent} \leftarrow \mathsf{Parent}(x_{near})$
            $E' \leftarrow E' \backslash \{(x_{parent}, x_{near})\}$
            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
        end if
    end for
end if
return $G' = (V', E')$

(Figure 5.1), and elaborates on its decoupling of action planning, as performed by the *Task Manager* and *Robot Execution Manager* modules, and control. The framework is general and can handle multi-arm mobile manipulators, but it can be scaled down to be used with fixed-base robots such as Baxter. In the section, $x \in \mathbb{R}$ represents a scalar, $\mathbf{x} \in \mathbb{R}^n$ is a vector with n elements, and $\mathbf{X} \in \mathbb{R}^{m \times n}$ is a matrix with m rows and and n columns. In addition, $\bar{x}$ and $\dot{x}$ are the reference value and the rate of change of variable x respectively. The robot configuration vector is referred to as $\boldsymbol{c} \in \mathbb{R}^n$ and contains the robot DOFs, e.g.,

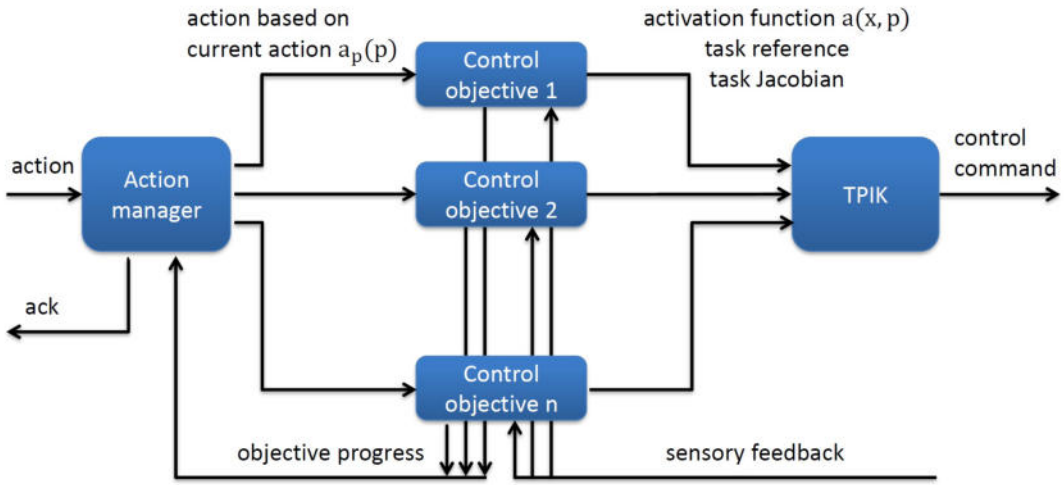


Figure 5.1: A sketch of the *Controller* internal structure.

joint positions and vehicle position, while the robot velocity vector is named $\dot{\boldsymbol{y}} \in \mathbb{R}^n$, and represents the controls to actuate the robot, e.g., joint velocities and vehicle velocities. The world frame in the cooperation scenario is attached to the robot torso and is fixed.

5.3.1 Control objectives

A control objective o expresses what the robot needs to achieve, e.g., reaching a desired position for its end-effectors. In mathematical terms, let us consider a scalar variable $x_o(\boldsymbol{c})$. For this variable, two broad classes of control objectives can be defined:

1. the requirement, for $t \to \infty$, that $x_o(\boldsymbol{c}) = x_{o,0}$ is called a *scalar equality control objective*,

2. the requirement, for $t \to \infty$, that $x_o(\boldsymbol{c}) < x_{o,max}$ or $x_o(\boldsymbol{c}) > x_{o,min}$ is called a *scalar inequality control objective*,

where $x_{o,0}$ is a given reference value, whereas $x_{o,min}$ and $x_{o,max}$ serve as lower and upper thresholds for the values scalar variables can assume. In the following, the dependency of x on $\boldsymbol{c}$ will be dropped to ease the notation. Each control objective updates its variables, its Jacobian and the associated activation function (see below) based on the robot's feedback.

5.3.2 Control tasks

To achieve a control objective o, a desired *feedback reference rate* is defined as:

$$\dot{\tilde{x}}_o(x_o) \triangleq \gamma(x_o^* - x_o),\ \gamma > 0, \tag{5.6}$$

where γ is a positive gain proportional to the desired convergence rate for the considered variable, and x_o^* is a point inside the state region where o is satisfied. The mapping between the task velocity scalar $\dot{x}_o$ and the system velocity vector $\dot{\boldsymbol{y}}$ is given by the Jacobian relationship:

$$\dot{x}_o = \boldsymbol{J}_o(\boldsymbol{c})\dot{\boldsymbol{y}}, \tag{5.7}$$

where $\boldsymbol{J}_o(\boldsymbol{c}) \in \mathbb{R}^{1\times n}$ is the Jacobian matrix of the task. The *control task* τ_o associated with the objective o is defined as the need of minimizing the difference between the actual task velocity $\dot{x}_o$ and the feedback reference rate $\dot{\tilde{x}}_o$.

5.3.3 Activation and deactivation of control objectives

Control objectives may or may not be relevant in a given situation. For example, if we consider the problem of avoiding an obstacle with one of the robot's links, then the related task would be relevant only when the link is close to the obstacle. This task should not over-constrain the robot whenever the link is sufficiently far away from the obstacle. Therefore, let us define a prototype for activation functions such that:

$$\alpha(x_o) = \alpha_o(x_o), \tag{5.8}$$

where $\alpha_o(x_o) \in [0, 1]$ is a continuous sigmoid function of a scalar objective variable x_o in case of inequality control objective, whose value is zero within the validity region of the associated control objective o. In specific, for the inequality objective with upper threshold $\alpha_o(x_o)$ is defined as:

$$\alpha_o(x_o) = \begin{cases} 0 & x_o < x_{o,max} - \delta \\ s(x_o) & x_{o,max} - \delta \leq x_o \leq x_{o,max} \\ 1 & x_o > x_{o,max} \end{cases}, \tag{5.9}$$

where $s(x_o)$ is an increasing sigmoid function as shown in Figure 5.2.

In case, the control objective is equality type, $\alpha_o(x_o)$ equals to one.

5.3.4 Task priority inverse kinematics

The approach of Task Priority schemes is to define p priority levels so that: (i) each task is assigned to one priority level; (ii) low priority tasks are inhibited

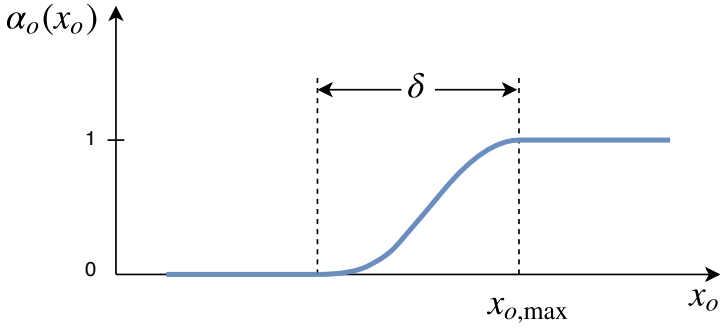


Figure 5.2: Activation function for inequality control objective with the upper threshold.

from interfering with high priority ones; (iii) different scalar objectives assigned to the same priority level can be grouped in a (possibly multidimensional) control objective. Assuming a priority level k with m scalar control tasks (and therefore m control objectives), the following vectors and matrices are defined.

- $\dot{\boldsymbol{x}}_k \triangleq \left[\dot{x}_{1,k}, \ldots, \dot{x}_{m,k}\right]^T$ is the stacked vector of all the reference rates, where the first index indicates control objectives $o_1, \ldots, o_m$ placed at the priority level k.
- $\boldsymbol{J}_k$ is the Jacobian relationship expressing the current rate of change of the k-th task vector $\left[\dot{x}_{1,k}, \ldots, \dot{x}_{m,k}\right]^T$ with respect to the system velocity vector $\dot{\boldsymbol{y}}$.
- $\boldsymbol{A}_k \triangleq \operatorname{diag}(\alpha_{1,k}, \ldots, \alpha_{m,k})$ is the diagonal matrix of all the activation functions in the form of (5.8).

With these definitions, the control problem is to find the system's velocity reference vector $\dot{\tilde{\boldsymbol{y}}}$ complying with the aforementioned priority requirements. In order to compute such a vector, a Task Priority Inverse Kinematics (TPIK) procedure has been proposed in Simetti & Casalino (2016). Here, it would suffice to describe the single regularization and optimization step, which unfolds iteratively taking into account all lower priority tasks. The manifold of solutions at the k level is:

$$S_k \triangleq \left\{ \arg \text{R-} \min_{\dot{\boldsymbol{y}} \in S_{k-1}} \left\| \boldsymbol{A}_k (\dot{\tilde{\boldsymbol{x}}}_k - \boldsymbol{J}_k \dot{\boldsymbol{y}}) \right\|^2 \right\}, \tag{5.10}$$

where S_{k-1} is the manifold of solutions of all the previous tasks in the hierarchy, with $S_0 \triangleq \mathbb{R}^n$. Since k is increased at each step, the recursion stops when $k = p$. The notation R- min is used to highlight the fact that the minimization must be

regularized to avoid algorithm singularities during transitions between pairwise control tasks. This regularization mechanism and the resulting TPIK algorithm as reported in Simetti & Casalino (2016) are initialized with:

$$\boldsymbol{\rho}_0 = \mathbf{0}, \boldsymbol{Q}_0 = \boldsymbol{I}, \tag{5.11}$$

and for $k = 1, ..., p$, where p is the total number of priorities:

$$\begin{aligned} \boldsymbol{W}_k &= \boldsymbol{J}_k \boldsymbol{Q}_{k-1} (\boldsymbol{J}_k \boldsymbol{Q}_{k-1})^{\#, \boldsymbol{A}_k, \boldsymbol{Q}_{k-1}} \\ \boldsymbol{Q}_k &= \boldsymbol{Q}_{k-1} (\boldsymbol{I} - (\boldsymbol{J_k} Q_{k-1})^{\#, \boldsymbol{A}_k, \boldsymbol{I}} \boldsymbol{J}_k \boldsymbol{Q}_{k-1}) \\ \boldsymbol{T}_k &\triangleq (\boldsymbol{I} - \boldsymbol{Q}_{k-1} (\boldsymbol{J}_k \boldsymbol{Q}_{k-1})^{\#, \boldsymbol{A}_k, \boldsymbol{I}} \boldsymbol{W}_k \boldsymbol{J}_k) \\ \boldsymbol{\rho}_k &= \boldsymbol{T}_k \boldsymbol{\rho}_{k-1} + \boldsymbol{Q}_{k-1} (\boldsymbol{J}_k \boldsymbol{Q}_{k-1})^{\#, \boldsymbol{A}_k, I} \boldsymbol{W}_k \dot{\tilde{\boldsymbol{x}}}_k \end{aligned} \tag{5.12}$$

and finally control output will be $\dot{\tilde{\boldsymbol{y}}} = \boldsymbol{\rho}_p$.

5.3.5 Control actions

. From the robot control standpoint, an action a in input to the *Controller* module in Figure 2.1 can be defined as a prioritized list of m control objectives $o_1, \ldots, o_m$ and the associated control tasks $\tau_1, \ldots, \tau_m$, to be managed *concurrently*.

Let us make a few examples to clarify the granularity and flexibility of the proposed approach, where each action is described in terms of its list of control objectives, in order of priority. A grasping action a_g for a single manipulator involves:

o_1 arm joint limits,

o_2 arm obstacle avoidance,

o_3 arm manipulability,

o_4 end-effector linear position control,

o_5 end-effector angular position control,

o_6 arm preferred pose.

A dual-arm object manipulation action a_d may involve:

o_1 object firm grasp kinematic constraint,

o_2 arms joint limits,

o_3 arms obstacle avoidance,

o_4 arms manipulability,

o_5 end-effectors linear position control,

o_6 end-effectors angular position control,

o_7 arms preferred pose.

Finally, a force regulation along a prescribed path action a_f for a single manipulator could involve:

o_1 force regulation,

o_2 arm joint limits,

o_3 arm obstacle avoidance,

o_4 alignment to the surface's normal,

o_5 arm manipulability,

o_6 end-effector path following, and

o_7 arm preferred pose.

Thanks to the proposed TPIK scheme, safety-oriented objectives such as *arm joint limits* and *arm obstacle avoidance* can be given high priority in the hierarchy, and they can be deactivated whenever irrelevant, through the use of properly defined activation functions. This is an important difference with respect to previous frameworks such as Chiaverini (1997); Siciliano & Slotine (1991), which handle only equality control objectives, or more recent frameworks such as Moe *et al.* (2016), where the management of inequality control objectives is not linear in the number of tasks. The TPIK output, i.e., the system reference velocity vector $\dot{\boldsymbol{y}}$, is given to the underlying dynamic control layers for execution and tracking.

Two remarks can be made. The first is that, in principle, an action a embeds an arbitrary number of m prioritized objectives $o_1, \ldots, o_m$, organized in different hierarchies. Typically, the main difference between any two actions is the set of objectives needed to achieve their goals and possibly other prerequisite objectives, whereas safety-oriented tasks are common to all actions. The second is that since actions are sequenced according to a cooperation model represented by an AND/OR graph, each action in hyper-arcs is defined with a few control objectives relevant to the action goal only.

Given the second remark, it is necessary to describe how transitions between two subsequent actions are implemented to achieve a safe – yet natural – robot

behavior. As discussed above, it is realistic to assume that actions are characterized by a common set of safety-oriented objectives, and differ only by a few action-specific objectives. For the sake of argument, let us imagine a unified list made up of all control objectives of two actions a_1 and a_2, e.g., $o_1, \ldots, o_{m_1+m_2}$. It is easy to imagine how, by a simple removal of some of the control objectives, the two initial sets can be easily determined. To do so, activation functions in the form of (5.8) are modified as:

$$\alpha(x, \boldsymbol{p}) = \alpha_o(x)\alpha_p(\boldsymbol{p}), \tag{5.13}$$

where $\alpha_p(\boldsymbol{p}) \in [0, 1]$ is a continuous sigmoid function of a vector of parameters $\boldsymbol{p}$ external to the control task itself. In particular, $\alpha_p(\boldsymbol{p})$ can be conveniently parametrized by the two subsequent actions, as well as the time elapsed from the start time T_{start} of the current action, to obtain the desired activation/deactivation smooth transition function between sets of objectives.

Once all the actions in cooperation models are defined, such a unified list of objectives can be easily built. Safety-oriented tasks are common to all actions, and they will be at the same (high) priority levels. As a consequence, for such tasks it will result that $\alpha_p(\boldsymbol{p}) = 1$. All other tasks will be instead managed by activation functions in the form $\alpha_p(\boldsymbol{p})$, to activate/deactivate action-specific and prerequisite objectives. Therefore, when a new action to execute is received, the *Action Manager* block (shown in Figure 5.1) activates and deactivates each control objective exploiting $\alpha_p(\boldsymbol{p})$.

Whenever a robot action is successfully executed, the *Robot Execution Manager* and consequently the *Task Manager* is invoked to update the *Action-State* table and the state of the cooperation.

5.4 Robot simulator

The *Robot Simulator* module predicts the robot behavior in its workspace by simulating the robot's closed loop kinematic model, thus allowing for predicting the outcome (in terms of success/failure) of any given action before its actual execution. A simulation is marked as successful if the robot reaches the given goal within a predefined time, and it is marked as failed otherwise. Although we simulate the robot's kinematic motion, during online execution there may be disturbances affecting robot motion; that may prevent the success of action eventually. For the work described here, we assume that the *Controller* module can compensate such disturbances. The output of the simulator includes a label indicating success or failure of an action, estimated execution time, suggested robot's trajectory, and the effort to perform the action.

The robot simulator solves a system of ordinary differential equations in the form:

$$\dot{X}(t) = F(X(t), U(X)). \tag{5.14}$$

from time t_0 and initial conditions X_0, to time t_{end}, where $X(t)$ is the vector of system states at each moment, and $U(X)$ is the control output vector, which is a function of states at each moment. To solve (5.14) we use the Runge-Kutta method, whereas to compute the control outputs we use the Task Priority based controller described above.

Chapter 6

Experimental Evaluation of the FlexHRC

Summary

In this chapter, we evaluate FlexHRC based on the functional requirements and taking into account the human, robot, interactions metrics. This evaluation is accompanied with several experiments, namely, collaborative screwing task, coordinated object transportation in a cluttered environment, task representation experiment, and finally collaborative table assembly. For all of these modules, the scenario is defined, the parts of FlexHRC used for the experiment is determined, the evaluation is performed and finally, the results are discussed based on functional requirements.

The equipments adopted for the evaluation of the proposed system includes a dual-arm seven DoF Baxter robot for cooperative manipulation, an LG G watch R (W110) smartwatch to acquire the acceleration data from the right wrist of the person, an LG G3 smartphone as a communication bridge between the smartwatch (Bluetooth) and the workstation (WiFi), and a Microsoft Kinect to acquire RGB-D data of the workspace. The whole FlexHRC architecture shown in Figure 2.1 is based on ROS Indigo and it runs on a workstation with Ubuntu 14.04 LTS 64-bit, 16 GB RAM, and Intel Core i7-4790, 3.60GHz CPU (for the collaborative screwing experiment, it runs on a 64 bit i5 2.3 GHz workstation, equipped with 4 Gb RAM, and Ubuntu 14.04.1 with kernel 3.19.0.).

6.1 Collaborative Screwing Task

6.1.1 Experiment objectives and scenario

In this experiment, a sensing, representation, planning and control architecture for flexible human-robot cooperation, shown in Figure 6.1 which is a part of the FlexHRC, is examined. It deals with the specifications outlined in Section 1.2 by design, in particular enforcing flexibility, intelligibility, adaptability, and transparency.

In this experiment, although robots suggest actions to perform based on *optimality* considerations and the goal to achieve, operators can choose an action without following robot's suggestions Hawkins *et al.* (2014), while the robot reacts to operators and plans for the next action accordingly Bertenthal (1996); Loehr *et al.* (2013); Vesper *et al.* (2010). Moreover, even if robot operations are well-defined in terms of motion trajectories and above all intended effects, reactive behaviors allow for dealing with partially unknown or dynamic workspaces, e.g., to perform obstacle avoidance, without the need for whole trajectory re-planning Simetti & Casalino (2016); Srivastava *et al.* (2014).

To this aim, the proposed architecture shown in Figure 6.1 implements a hybrid, reactive-deliberative human-robot cooperation architecture for *assisted cooperation* Helms *et al.* (2002); Krüger *et al.* (2009) integrating different modules, namely: (i) human action recognition using *wearable sensors*, which do not pose any constraint on operator motions, to address F_7, and exploiting statistical techniques for action modeling Bruno *et al.* (2013) to take F_5 into account; (ii) representation of human-robot cooperation models and on line reasoning using *AND/OR graphs* in propositional logic level Hawkins *et al.* (2014); Johannsmeier & Haddadin (2017); Sanderson *et al.* (1988) and *Task Manager* enjoying only the reactive adaptation to deal with F_2; (iii) control schemes based on a *Task Priority* framework and the *Task Manager* to decouple human-robot action planning from robot motion planning and control Simetti & Casalino (2016), therefore addressing F_3.

As a prototypical example, a screwing task has been considered. A human operator wearing a smartwatch and Baxter face each other on the opposite sides of a table, where bolts, wooden plates and screwdrivers are located in *a priori* known positions. The goal of the task is to sink a bolt inside a plate using a screwdriver, and place the assembled piece in a final position on the table.

Common sense and experience lead to the identification of three different cooperation models, referred to as M_{blue}, M_{black} and M_{red} in the following paragraphs, and represented in the corresponding AND/OR graph by three different paths, namely P_{blue}, P_{black} and P_{red} (Figure 6.2). In this case, cooperation models are structured in advance. It is noteworthy that current work is devoted to

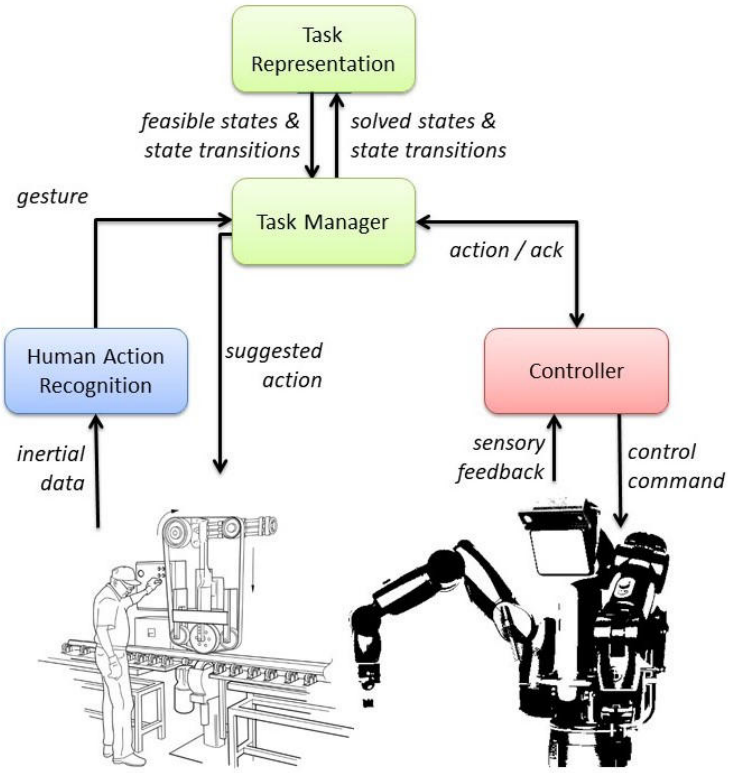


Figure 6.1: The software architecture for collaborative screwing task.

learning cooperation paths from open-ended observations of how humans would behave if not instructed about how to cooperate with the robot, in order to extract the most natural interaction sequences from a human perspective. However, this is out of the scope of the paper. The resulting AND/OR graph has $|N| = 9$ states and $|H| = 9$ hyper-arcs. The weight associated with each hyper-arc is specified by the estimated *effort* needed by an operator or a robot to complete the corresponding actions. As described above, such an effort does not necessarily consider only time, but may be a complex function taking into account also operator preferences, ergonomic aspects of the operation, as well as its intrinsic difficulty. However, such a function must be *monotonic*, i.e., actions with associated low weights are to be preferred to actions characterized by high weights within each cooperation model. For this validation scenario, values have been *a priori* determined through a test campaign with expert users. As Figure 6.2 shows, given this weights assignment, it follows that the optimal cooperation model is therefore M_{blue}, with a total expected cost of $cost(P_{blue}) = 14$.

As an example of a cooperation model, Figure 6.3 shows snapshots of an actual human-robot cooperation process for the screwing task described above,

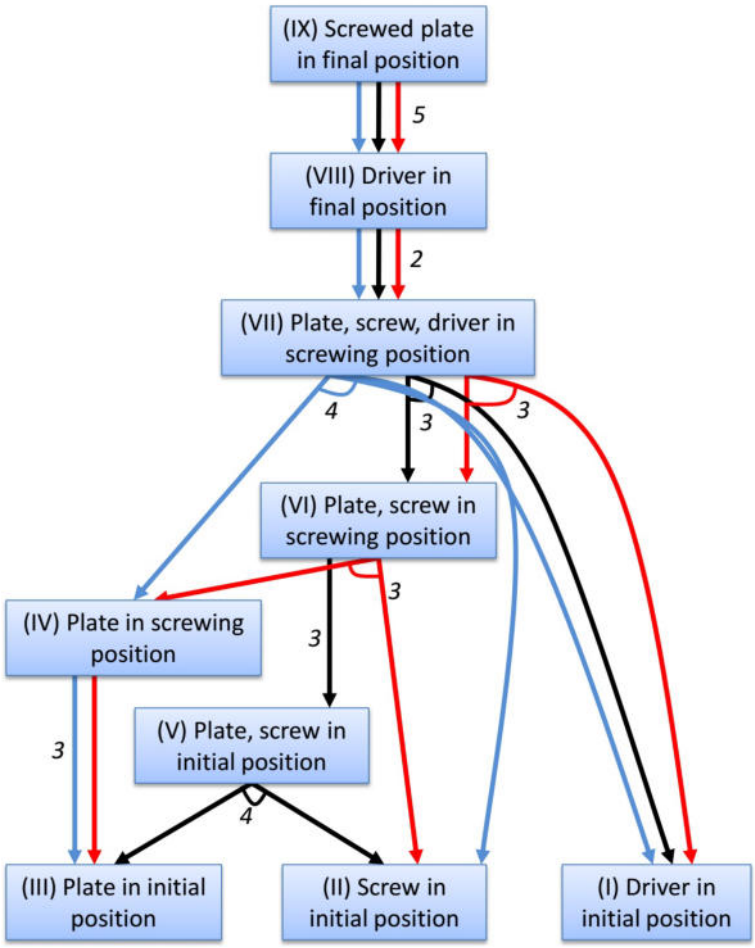


Figure 6.2: The AND/OR graph representation of the screwing task: different colors (*blue*, *black* and *red*) indicate different action sequences the cooperation can unfold in. Hyper-arcs costs appear beside the hyper-arcs they refer to.

which starts with M_{blue}, but switches to M_{black} when the operator performs the first action (Figures 6.3b and 6.3c), which is different from the one expected in M_{blue}. For this task, modeled human and robot actions are:

- *initial bolt sink*: the human sinks the bolt in the wooden plate's countersink while the plate is still located on the table (Figures 6.3a to 6.3c);
- *wooden plate pick up and positioning*: the robot picks the wooden plate from a predefined position on the table and keeps it firmly using both grippers (Figure 6.3d and Figure 6.3e);
- *bolt or screwdriver pick up*: the operator picks up a bolt or a screwdriver

Figure 6.3: The sequence of actions associated with M_{black}, chosen after the operator decided not to follow M_{blue} by performing the action *initial bolt sink*.

from the table (Figure 6.3f);

- *bolt screw*: the operator sinks the bolt using the screwdriver (Figure 6.3g);
- *screwdriver put down*: the operator puts the screwdriver down on the table (Figure 6.3h and Figure 6.3i);

Table 6.1: Recap of the reliability, robustness and flexibility experiments: number of trials (#) for each cooperation path P, success rate (S), average time (avg) and standard deviation (std) for completing the task successfully.

$\mathcal{P}$	#	S [%]	avg [s]	std [s]
P_{blue}	21	95.24	79.86	3.54
P_{black}	23	30.43	97.41	2.66
P_{red}	22	68.18	93.94	3.73

- *wooden plate put down*: the robot puts the wooden plate down in a predefined position on the table (Figure 6.3j and Figure 6.3k);
- *reset pose*: robot's pose is reset (Figure 6.3l).

Obviously enough, other human and robot actions can be modeled, as well as are other states in the cooperation models.

In the following, a number of cooperation experiments are discussed. In all experiments, a trial is considered *successful* if the operator and the robot reach the root state of the AND/OR graph, namely *screwed plate in final position*, by means of any of the allowed paths. The operator and the robot are not allowed to repeat the sequence of actions in a hyper-arc: if, at the first execution, actions are not successfully accomplished, the trial is considered *failed*. Experiments have been designed with three specific validation goals in mind:

- assessing reliability, robustness and flexibility of FlexHRC in terms of cooperation success rate and possible explanations for failures;
- quantifying computational performance, in terms of latency of action recognition and reasoning time;
- determining the *Controller* module's capabilities in solving constrained motion problems reactively, i.e., without burdening the *Task Manager* module.

6.1.2 Reliability, robustness and flexibility

In a first set of experiments, a total of 66 human-robot cooperation trials have been conducted with a single human operator. Table 6.1 shows, for each cooperation path P_{blue}, P_{black} and P_{red}, the number of trials, the success rate, the average completion time and the standard deviation for successful cases. It is possible to observe that not all cooperation models are equally difficult. P_{blue} is characterized

by a very high success rate, whereas the same does not occur for P_{black} and P_{red}. Both of them are characterized by a greater number of actions, which is reflected in a higher average time needed to complete the operations. Overall, there are 24 failures over 66 experiments, and in particular one failure for P_{blue}, sixteen failures for P_{black} and seven failures for P_{red}. As far as failures are concerned, two of them are due to human mistakes, e.g., misinterpretation of *Task Manager*'s suggestions, four of them are related to communication failures and temporary high latencies among software modules, one to a robot failure while executing a command, whereas seventeen of them have been caused by inaccurate recognition of the operator gestures.

A *trend* analysis on all experiments highlights a phenomenon related to how humans adapt their motions in order to facilitate gesture recognition over time. This means that inertial measurements become more correlated with gesture models encoded using GMM and GMR as the human progresses in performing them. The most direct consequence is that success rate increases, whereas average completion time and standard deviation decrease. In fact, if one looks only at the last 10 experiments per cooperation path, the success rate for P_{black} and P_{red} become 50% and 80%, respectively. Although adaptation can be expected, current work is devoted to better characterize this phenomenon. As a preliminary analysis, it can be noticed that it occurs especially for *initial bolt sink*. If one looks at this model, it can be observed that it shares similarity with other models to a high degree, such as *bolt or screwdriver pick up* or *screwdriver put down*, mainly in the first part. After some trials, operators are able to modify their motions to allow *Human Activity Recognition* to disambiguate between all models, with an average increase in successful recognition of about 20%.

As far as robustness considerations are concerned, the system proves to switch seamlessly among different cooperation models. Examples of such switches can be observed in the accompanying video[1]. Here, let us focus on the switch occurring between P_{blue} and P_{black} in Figures 6.3a to 6.3c. At the beginning of the cooperation, Baxter follows the optimal cooperation path, namely P_{blue}, by default. It moves its right arm to perform *wooden plate pick up and positioning* (Figure 6.3b) to reach the state *Plate in screwing position*. However, at the same time, the operator decides to perform *initial bolt sink*, i.e., the state *Plate, screw in initial position* is reached. This state is part of cooperation model M_{black}, and therefore FlexHRC sets P_{black} as the current context. At this point, the best solution involves reaching *Plate, screw in screwing position*, which means for the robot to perform *wooden plate pick up and positioning*. Then, M_{black} unfolds from this moment on. It is noteworthy that, from the operator perspective, this model switch does not imply any perceivable interruption in the operation workflow.

[1]Please refer to: https://youtu.be/MZv4fUuklq8.

Table 6.2: Required reasoning, human, and robot average time percentages for successful tasks.

$\mathcal{P}$	avg T_{ao} [%]	avg T_h [%]	avg T_r [%]
P_{blue}	0.09	44.04	55.86
P_{black}	0.09	45.93	53.97
P_{red}	0.09	51.94	47.95

This capability demonstrates requirement R_1 described in the Introduction.

6.1.3 Computational performance

Table 6.1 shows, in the last two columns, the average time required to complete cooperation models and the associated standard deviations. Overall, P_{blue} outperforms P_{black} and P_{red} in terms of required time, while the three cooperation models are comparable in terms of determinism in execution.

Table 6.2 reports, for each successfully executed cooperation path, the percentages of average time related to FlexHRC reasoning (T_{ao}), and the time needed for human operators (T_h) and robot actions (T_r). The first percentage is a measure of the time needed by the AND/OR graph traversal algorithm to suggest next actions to be performed either by the operator or the robot. Assuming a sequence of n actions, T_{ao} is defined as the sum of all such $n-1$ contributions (the first being set by default on the optimal path), and each contribution is given by the difference between the time T_{next} when the next action a_i suggestion is ready and the time T_{ack} when an acknowledge for a previous action a_{i-1} is received, such that:

$$T_{ao} = \sum_{i=2}^{n} T_{next}(a_i) - T_{ack}(a_{i-1}). \tag{6.1}$$

The second percentage refers to the time spent by operators in the cooperation, as well as the time required to detect their motion. It is noteworthy that this is a *greedy* estimate of human motions, since operators may perform other motions irrelevant for the cooperation. Let us assume that the operator performs m actions, and let us define $\overline{T}_h$ as the sum of all such m contributions, then each contribution depends on the sum of effective human motion time and the time needed by the *Human Action Recognition* module to recognize such a motion.

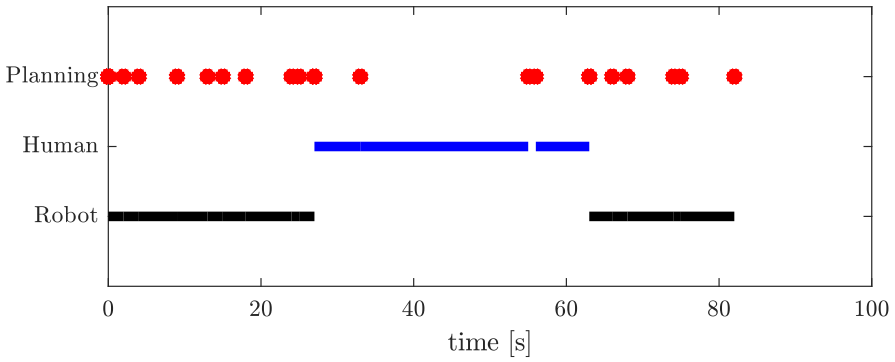


Figure 6.4: An example of time allocation in case of P_{blue}.

Therefore, $\overline{T}_h$ is given by:

$$\overline{T}_h = \sum_{i=1}^{m} T_{rec}(a_i) - T_{next}(a_i), \tag{6.2}$$

where T_{rec} is the recognition instant. However, it is also necessary to take into account cooperation model switches. Assuming to have k context switches during a single cooperation task, an additional term to $\overline{T}_h$ must be added, which considers the interval between the time T_{start} when the switching action a_i starts and the time T_{next} when the next action a_{i+1} in the new cooperation path is suggested, such as:

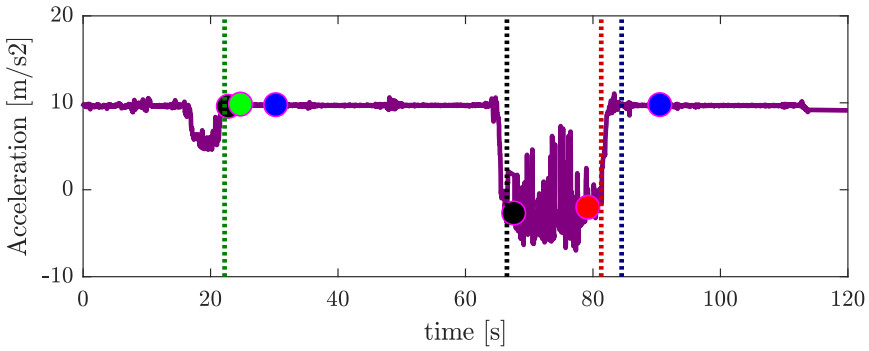
$$T_h = \overline{T}_h + \sum_{i=1}^{k} T_{next}(a_{i+1}) - T_{start}(a_i). \tag{6.3}$$

The average time percentage of AND/OR graph traversing is in all cases less than 0.1% of the total time. This is also due to the proposed control framework, which does not require a computationally intensive planning in the configuration space, thanks to its ability of reactively solving *local* obstacle avoidance constraints.

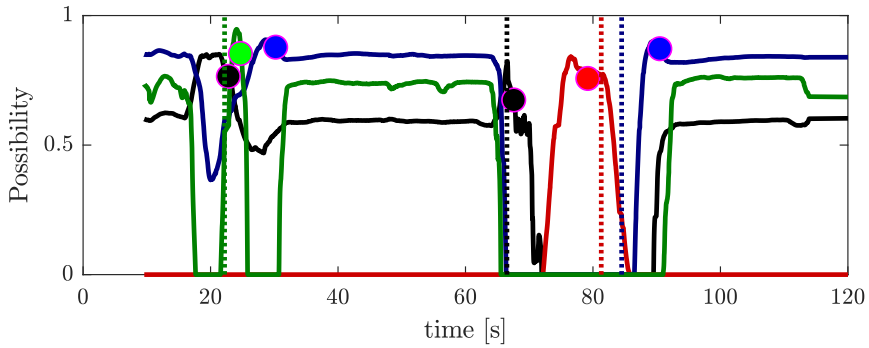
As far as T_h and T_r are concerned, it is possible to observe that they are comparable, with the contributions of operators more evident in P_{black} and P_{red}.

Figure 6.4 shows an example of time distribution for one task which follows P_{blue}. The Figure shows that the majority of the time is related to either operator or robot actions, and just a negligible part of the whole cooperation task is related to AND/OR reasoning. The total time to perform the assembly in this test is 82 seconds, of which 44.13% spent by the human, 55.56% spent by the robot, and 0.09% by the *Task Manager* module.

A specific analysis of the delay introduced by the *Human Action Recognition* module during cooperation tasks has been performed as well. In particular, it



(a) Inertial data along x axis.

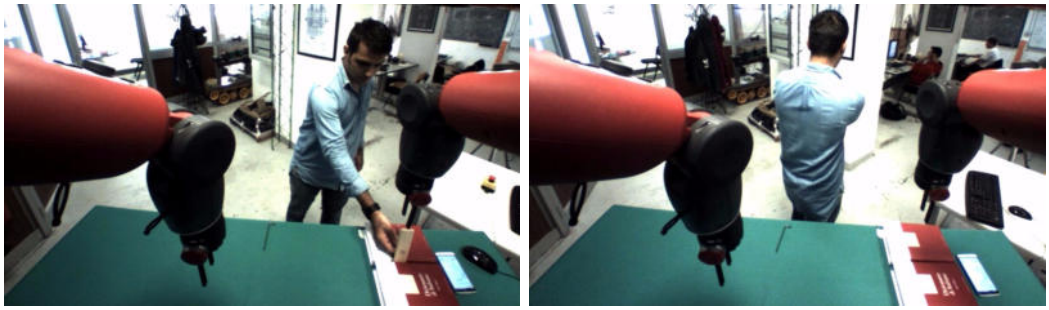


(b) Trend in possibilities.

Figure 6.5: Delays introduced in human action recognition in one trial. Dots represent recognition times; vertical dotted lines mark the moments in which human gestures actually end.

is necessary to characterize the interval between the time T_{rec} the action a_i is recognized by the module and the time T_{end} the action truly ends.

Figure 6.5 shows an example of P_{black}. On the top, acceleration data along the x axis are presented, and on the bottom the corresponding possibility trends are shown. Operator actions are represented using different colors (*black* for *bolt or screwdriver pick up*, *red* for *bolt screw*, *blue* for *screwdriver put down*, *green* for *initial bolt sink*). Colored circles represent the time instants T_{rec} at which *Human Action Recognition* assesses actions, whereas vertical dotted lines show the actual completion instants T_{end}, which have been determined by manually inspecting acceleration data. In this case, the time required to recognize *bolt or screwdriver pick up*, *bolt screw*, *screwdriver put down* and *initial bolt sink* are approximately 1.2, 0.1, 6.1 and 2.6 seconds, respectively. Overall, around 10 out of 120 seconds of cooperation time are due to the human action recognition delay. In all the tests, *Human Activity Recognition* introduces a delay accounting for about 10%



(a) (b)

Figure 6.6: A human operator does not have to necessarily perform actions in front of the robot for those actions to be recognized and classified: (a) the operator performs the *initial bolt sink* action in sight of the robot, (b) the operator performs the *initial bolt sink* action out of sight of the robot.

Table 6.3: Recap of the experiments to assess the performance of human action recognition with operators who did not contribute to the training of the gesture models.

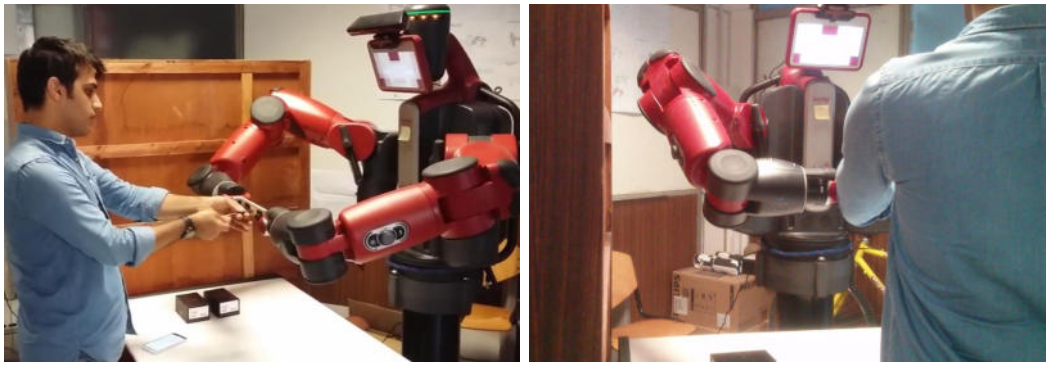
Action	S [%]	S (first) [%]	S (last) [%]
bolt or screwdriver pick up	100	100	100
screwdriver put down	100	100	100
bolt screw	100	100	100
initial bolt sink	53.19	55.55	30

of the overall cooperation time.

6.1.4 Performance of human action recognition

In a second set of experiments, the system has been tested with 10 people who did not participate in the training phase to evaluate whether gesture models obtained using a specific training set are general enough to be used with more than one operator. The age of all participants (two females, eight males) ranges between 21 and 30 years. Each participant is required to perform 5 trials. Before the trials, the participant is verbally instructed on how to cooperate with the robot, an example cooperation model is shown by an experimenter, and the participant is allowed to practice with the *initial bolt sink* action.

Table 6.3 shows the results of these experiments. In the Table, the second column indicates the overall success rate, the third column the success rate of the



(a) (b)

Figure 6.7: An activity part of P_{blue}, when an obstacle is detected and avoided by the robot's elbow joint.

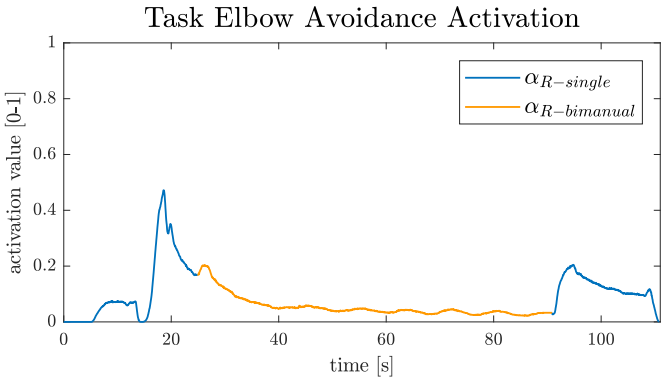


Figure 6.8: Activation function of the robot's elbow avoidance task for the right arm: in *blue* for a single-arm operation, in *orange* for a dual-arm operation.

first trial, and the fourth column the success rate of the last trial. Among a total number of 50 trials, in one case the participant did not follow the instructions, two times the communication stopped, and 22 times the cooperation failed because of incorrect action recognition, always, specifically, of *initial bolt sink*. Indeed, *initial bolt sink* is not characterized by any improvement as far as the different trials are concerned. Other actions seem more natural and are always recognized correctly without a specific training.

As far as naturalness is concerned, it is noteworthy that FlexHRC detects and classifies operator actions even if those are not executed within the robot workspace or field of view. An example is shown in Figure 6.6.

6.1.5 Task priority control

In the experiments to evaluate the *Controller* module, which embeds the Task Priority control scheme described in Section 5.3, the evaluation objective is two-fold: on the one hand, assessing the *Controller* capabilities in dealing with situations requiring reactive control, e.g., obstacle avoidance, which does not need planning in the operational space; on the other hand, assessing its capability of doing so without jeopardizing the overall cooperation context process, in so far as cooperation time is concerned. To this aim, the same operator of the first set of experiments performed an additional number of trials after the introduction of obstacles in the robot's workspace.

As an example, Figure 6.7 shows a cooperation task where a lateral obstacle has been located on the right hand side of the robot. Such an obstacle would impede the robot to perform *wooden plate pick up and positioning*. Therefore, beside action-specific control objectives and tasks, an *arm obstacle avoidance* objective has been introduced in the tasks hierarchy. In this particular case, we focused on the robot's elbow avoidance, but the proposed technique can be employed for any frame inside the rigid body space of the manipulator. The robot is equipped with a Kinect sensor mounted on the head to perceive the environment. The acquired point cloud is processed by a perception module (first described in Buoncompagni & Mastrogiovanni (2015); Buoncompagni *et al.* (2017)), whose output is a lumped representation of the plane approximating the wall. The control objective variable x for the arm obstacle avoidance is defined as the norm of the vector between the origin of the elbow frame and the closest point on the plane, which is required to be maintained above a minimum safe threshold.

Figure 6.8 shows the trend of the corresponding activation function (5.8) for the arm obstacle avoidance objective, during both single-arm and dual-arm operations. It is possible to observe that the activation never reaches its maximum value, and the avoidance task is completed within the established thresholds. Such an example shows how the *Task Manager* can focus on the generation of Cartesian trajectories for the end-effectors or for the object being manipulated, once grasped by both robot grippers, without planning in the operational space, as the underlying robot controller has the reactive capabilities to deal with (local) avoidance of obstacles that were not taken into account in the original trajectory planning. There is no observable difference on actual cooperation context execution times due to the effect of reactive tasks. For instance, in the trial referring to Figure 6.8, avg $T_{ao} = 0.06$ seconds, avg $T_h = 54.76$ seconds and avg $T_r = 44.96$ seconds. Although an exhaustive experimental campaign varying obstacle size and number has not been carried out, these preliminary results allow us to conclude that FlexHRC complies with requirement R_2 as described in the

Introduction.

6.1.6 Discussion

In this experiment, a novel architecture for human-robot cooperation is proposed, which is aimed at addressing a few challenges in shop-floor environments. The proposed architecture supports a natural, intuitive, assisted and direct cooperation. On the one hand, operator gestures implicitly drive the cooperation by executing meaningful actions, and the robot flexibly and seamlessly adapts to those actions via a number of allowed cooperation models. On the other hand, the robot controller deals with all low-level complexities, e.g., to perform obstacle avoidance in a full reactive fashion, without the need for re-planning in most cases. The proposed architecture in this experiment has obviously a number of limitations, which are considered as challenges in current research activities.

1. Inertial data models obtained via GMM and GMR can be very similar to each other, depending on the action. This may lead to false positives, and requires processing as much data as possible before action recognition can occur. To solve these ambiguities, work is currently carried out to integrate different sensing modalities, such as RGB-D sensors and wearable suits, as well as investigating different classification techniques. However, an adaptation trend on the human side has been observed: people naturally tend to move in such a way as to maximize the likelihood for their actions to be properly recognized. This leads to a possible extension, i.e., to include on line learning capabilities to perform co-adaptation, as discussed in Fragkiadaki *et al.* (2015); Hadfield-Menell *et al.* (2016); Nikolaidis *et al.* (2014); Saveriano *et al.* (2015).

2. Human activity recognition is based only on the detection and classification of certain gestures, no guarantees about the use of specific tools can be given. For example, an operator manually operating a screwdriver to sink a bolt, and the same operator *mimicking* the gesture without holding a screwdriver appear as indistinguishable to the system.

3. Actions are *a priori* assigned to operators or robots, depending on their different capabilities as far as object manipulation is concerned, in a way maybe similar to what has been done in Mastrogiovanni *et al.* (2013). An on the fly assignment to the operator or the robot would increase to a great extent the flexibility of the cooperation process.

4. Safety considerations in FlexHRC are considered only to a limited extent. Different safety strategies including the detection of sudden, unwanted con-

tacts, as well as active or adaptive safety measures are investigated extensively in the literature Makris *et al.* (2016); Michalos *et al.* (2015). Currently, the adoption of safety measurements in FlexHRC is limited to the possible adoption of specific tasks with high priority in the control framework.

5. FlexHRC does not consider activities where physical cooperation and purposive contacts are chiefly needed. Currently, cooperation models take a form of turn-taking, with a few implicit turns where physical interaction is necessary. However, an explicit account of such tasks is of the utmost important in a whole range of real-world shop-floor activities.

6. The use of AND/OR graphs limits the allowed cooperation paths to a few ones, which are designed through common sense and human experience, whereas more versatile action planning techniques may be adopted. On the one hand, an AND/OR graph leaves to the operator the responsibility to decide which cooperation path to pursue among the allowed ones, which makes sense in shop-floor scenarios. On the other hand, planning techniques are in principle more flexible, but at the expense of being unpredictable to a large extent, and leading to non intuitive cooperation paths, which are solely determined on the basis of plan optimality.

Finally, it is noteworthy that an important aspect to be addressed is the intersection between Task Priority control, motion planning and execution, as well as the AND/OR graph. In particular, determining what to do when gestures are not properly recognized, or detecting when the motion controller is stuck in a local minimum, and the development of motion or whole task re-planning for error recovery constitute an important research topic, as demonstrated by a number of contributions in the field Agrawal *et al.* (2016); Srivastava *et al.* (2014).

6.2 Coordinated Object Transportation in Cluttered Environment

6.2.1 Experiment objectives and scenario

In this work, to demonstrate the capabilities of our proposed architecture shown in Figure 6.10, we have chosen as a test case an "hybrid" object bi-manual manipulation task, performed using a bimanual robot and an RGB-D optical sensor. In the experiments, the dual arm robot needs to move the object to a required position, avoiding obstacles that interfere both with its end-effectors and its elbows, whilst maintaining the rigid grasp of the object. The obstacle avoidance

problem, in particular, is tackled using two different strategies at the same time: (a) a path planner is computing in real time a feasible path for the movements of the end effectors, without caring for the rest of the arm links, and (b) a control task is reactively avoiding the obstacles for the elbow joints. The results achieved show the capabilities of the underlying control architecture in solving local problems, in the perspective of simplifying the amount of work of higher levels that can therefore focus better on high level objectives.

Indeed, this work is framed within an integration effort between AI techniques and control Darvish *et al.* (2016), with the goal of moving towards a smoother combination of the two that could possibly make the difference in terms of performance, flexibility and dependability, especially for factory of the future environments.

Figure 6.9 shows the coordinated transportation of an object in the workspace in existence of the static obstacles. There are two balls in the scene playing the role of the obstacles and a plate as the object to transport. The robot arms should transport the plate coordinately using a bimanual control task (coordination task); and while the object is avoiding the obstacles thanks to the software architecture depicted in Figure 6.10, the elbow joints of the robot arms should avoid the obstacle reactively at the kinematic control level.

(a) (b) (c) (d)

Figure 6.9: A series of successive still frames from one of the coordinated transportation experiments. As it can be seen in the last two frames 6.9c and 6.9d, the elbow is gradually raising its height to stay out of the bounding box defined for the obstacle.

6.2.2 Experimental results

We have done two experiments to show the interaction between the software architecture shown in Figure 6.10 and the *Controller* module; and together, how they solve the avoidance task through planning and in a reactive manner, respectively. In particular, in the first experiment the elbow obstacle avoidance task has not been included while it is in the second one.

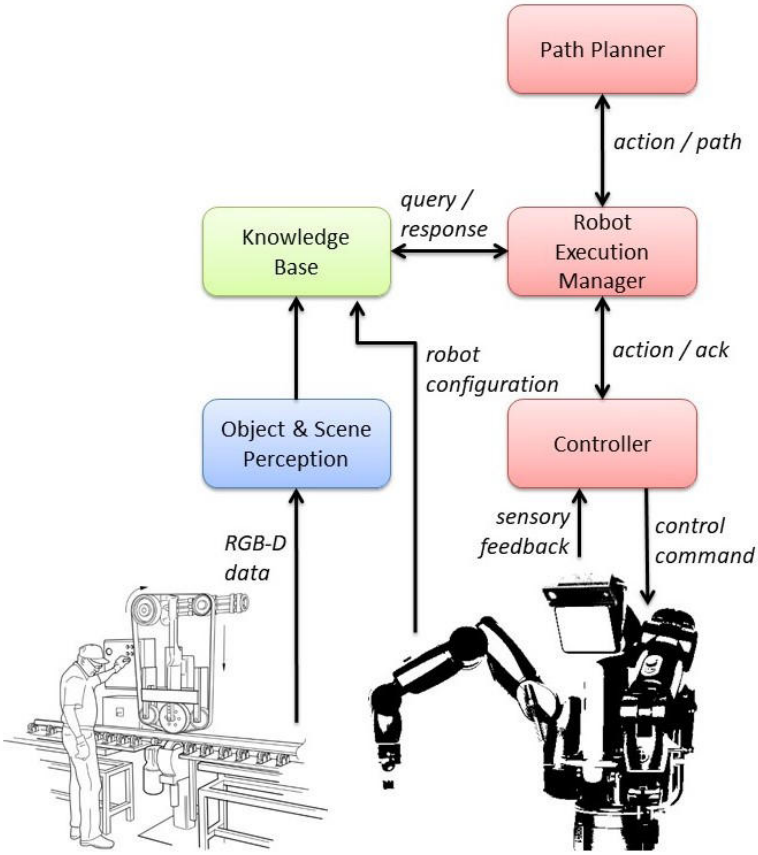


Figure 6.10: The software architecture for the coordinated transportation of an object in cluttered environment.

Figures 6.11b and 6.11d show that, in both experiments, the grasped object (object frame) is avoiding the obstacles within the workspace and reaching its final goal by following the path generated by the *Path Planner*. Figure 6.11c shows the distance between the closer obstacle to the left arm elbow joint of the robot. The object is following its path, but the elbow is getting closer to an obstacle in the working space. Figures 6.11e and 6.11f depict the distance between the elbow and the obstacle and the activation function. When he elbow joint distance get close to the threshold of the obstacle bounding box the task is activated, and while the object is following its path, the elbow is avoiding the obstacle within the defined threshold.

As the results of the experiments show, the kinematic control layer of the *Controller* controls the robot arms to reactively avoid the elbow's obstacle, freeing the higher level from having to plan a path taking into account all the possible

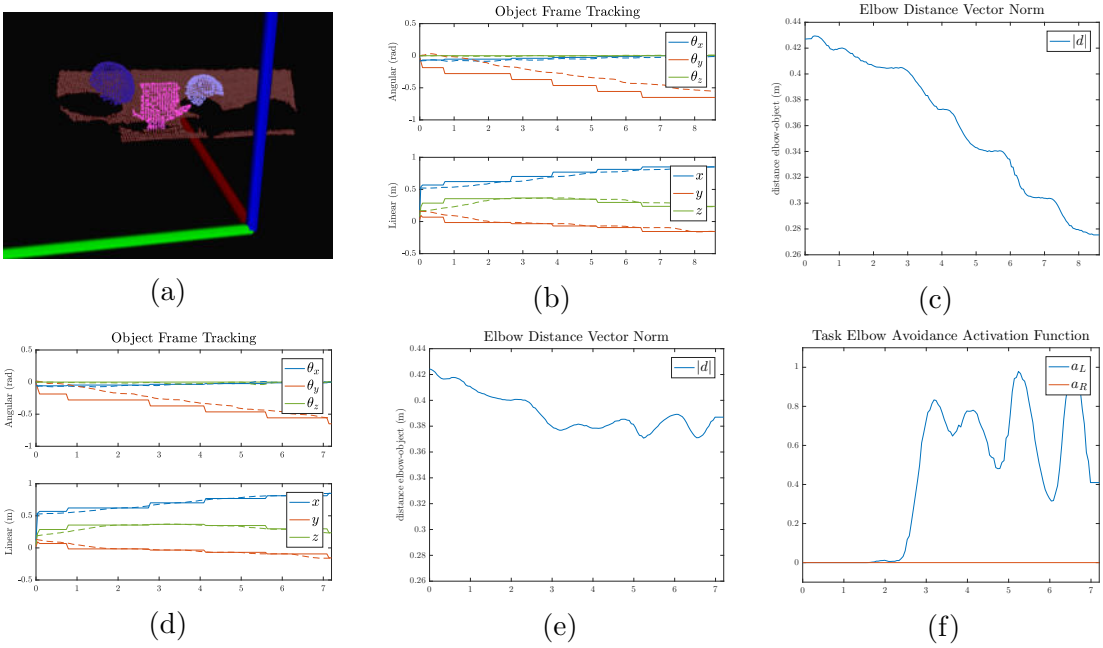


Figure 6.11: In figure 6.11a scene and objects recognized by *Object and Scene Perception* and relative frame of the vision system, each color is identifying a different object. Figures 6.11b and 6.11c are for the experiments with obstacle avoidance of the grasped object at the path planning level and figures 6.11d, 6.11e, and 6.11f are for the experiment related to obstacle avoidance both at *Path Planner* and *Controller*.

collisions.

6.2.3 Discussion

The experiments show the benefit of chopping the execution of complex behaviors into a number of different modules and as a consequence translating the desired system abilities into a set of control objectives. The proposed *Controller* can enforce safety measures autonomously and exposes a rich interface which releases the *Robot Execution manager* and *Path Planner* from dealing with the complexity of low-level problems. Moreover, the robot end-effector or the object will follow a asymptotically global optimal path and does not trap in local minimum.

Beside the promising results we attain by the integration of the RRT* based *Path Planner* and the *Controller* as shown in Figure 6.10; there are a number of challenges in the proposed architecture: (i) the path planner finds a path for a point in the workspace. In order to avoid the collision between the transporting object and the obstacles in the workspace, we consider the object size to the

obstacle safety margins. With this approach, if the object size with respect to obstacles and workspace size is not negligible; it causes problem for finding the path for the object and we may not find a path for the object to reach the goal (although there might be a solution for path planning); (ii) The *Path Planning* time increases proportional to the workspace size and the number of obstacles in the workspace, and therefore the RRT* method might not be the best option for workspaces where there exist human; (iii) the RRT* method can not ensure the human safety and avoiding the collision with the human which is moving in the robot workspace (dynamic obstacle).

6.3 Task Representation Experiments

Two types of experiments have been performed to evaluate the novelties of the *Task Representation*: I) stand-alone (robot only) tests have been executed to show how the FOL AND/OR graph outperforms the standard propositional-logic (PL) AND/OR graph; II) further stand-alone tests have been executed to show the scalability of the proposed *Task Representation* module by comparing the benefits of the Hierarchical AND/OR graph with respect to the standard single-layer one.

6.3.1 Propositional Logic and First Order Logic AND/OR Graph Performance Comparison

To compare the two versions of the FOL and PL state and state transition representation of the AND/OR graph, let us consider placing two identical balls (`A` and `B`) into two identical boxes (`C` and `D`). Even if the objects are identical, their location in the workspace is different. Therefore, in the execution level the `ball(?)` and `box(?)` predicates should be grounded to either `A` or `B` and `C` or `D`, so that their locations are known when giving a command to the robot controller.

In the representation level, it does not matter which ball and box is chosen. Fig. 6.12 on the left, shows all the possible ways to place the two balls into the two boxes when they are grounded in the *Task Representation* level. The placement task is composed of 11 nodes, seven hyper-arcs and two paths to follow. Instead, Fig. 6.12 on the right shows the placement task when the `ball(?)` and `box(?)` predicates are used. The placement task is composed of seven nodes, three hyper-arcs, and one path to follow. By comparing the two graphs, it is evident how the PL AND/OR graph is more complex to design and requires an higher computational time to be solved online.

We have performed the ball placement task shown in Figure 6.12 ten times. As expected, the FOL AND/OR graph outperforms the PL AND/OR in terms

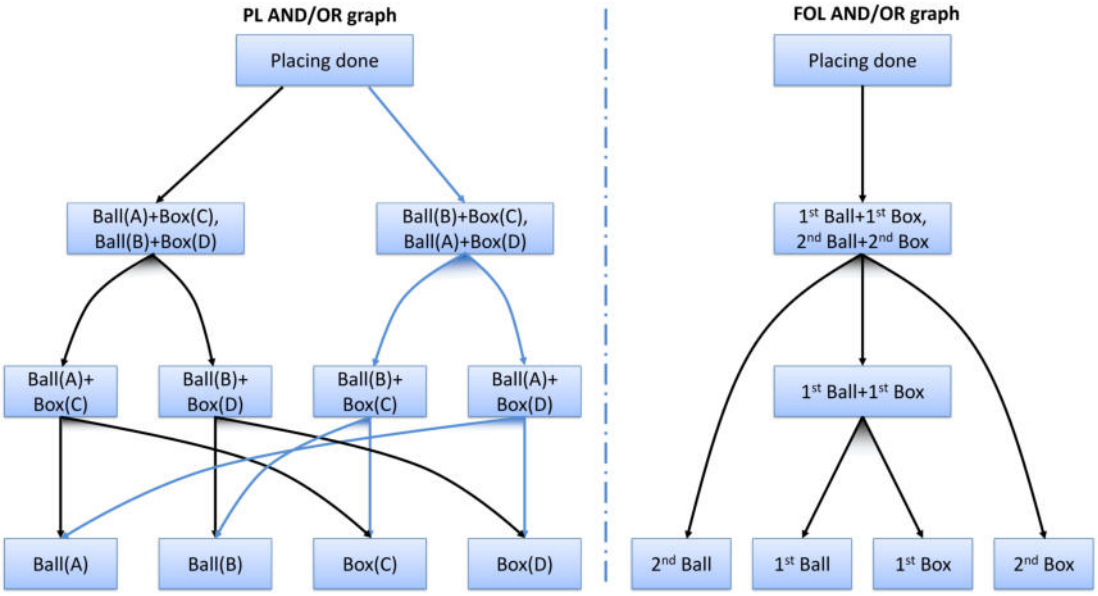


Figure 6.12: The PL (Propositional Logic) and the FOL AND/OR graph for representation of placement of two identical balls (`A` and `B`) into two identical boxes (`C` and `D`) (left: PL AND/OR graph, right: FOL AND/OR graph).

of computational time mean and standard deviation values for both online and offline phases. The offline phase takes 2.23×10^{-4} seconds on average for computation of FOL, while in case of PL it takes 2.91×10^{-4} seconds. The standard deviations are 1.66×10^{-5} seconds and 5.09×10^{-5} seconds respectively. For the online phase, the mean computational time and the standard deviations are 3.26×10^{-4} seconds and 2.74×10^{-5} seconds for FOL AND/OR graph, while for the PL are 3.50×10^{-4} seconds and 6.34×10^{-5} seconds.

6.3.2 Single-layer and Hierarchical AND/OR Graph Performance Comparison

The second set of tests was conceived to compare the benefits of using an hierarchical AND/OR graph with respect to the standard single-layer one. To this aim, we have performed several table assembly tasks, gradually increasing the number of tables legs to be assembled from one to nine.

Figure 6.13 shows the hierarchical graph of a table assembly with two legs. We used the FOL to represent this assembly task. To connect a leg to the tabletop there are four paths that can be followed: I) the robot connects the leg and the tabletop directly (blue path, cost of one); II) the human connects the leg and the tabletop directly (red path, cost of three); III) the robot places the leg in a

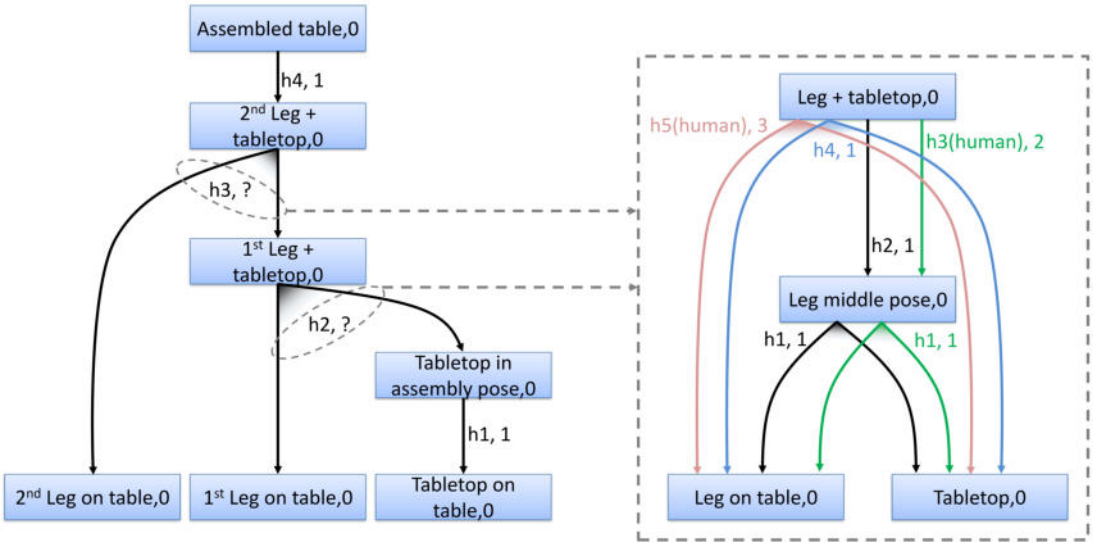


Figure 6.13: The hierarchical AND/OR graph for the table assembly with two legs (left: high level AND/OR graph for table assembly, right: low level AND/OR graph for connecting a leg to a tabletop).

new position in the workspace and later connects the leg to the tabletop (black path, cost of two); IV) after the leg is in the new position, the human connects the leg to the tabletop (green path, cost of three). The reason for introducing a temporary new position for the table leg in the assembly scenario is that, in case of a dual-arm robot, there might be situations where one arm can reach the initial leg position, but only the other arm can transport the leg to its final position. Although the cost of the red and green paths are equal (three); if the robot can perform hyper-arc *h1*, it follows the green path. Because in the green path, the first feasible hyper-arc *h1* is common in black and green paths; therefore solving this hyper-arc can lead to the black path, which has a lower cost (two), as well as the green path. Before solving the *Leg middle pose* node, the robot does not know if it can execute the hyper-arc *h2* or not (using simulation); therefore it will follow the green path if possible. It is noteworthy that human can traverse to the red path online if he decides.

Figure 6.14 depicts the average computational time of the assembly task both for the offline phase (top) and online phase (bottom). For each number of legs, we have performed the table assembly task ten times, and reported the average time. The figure represents the time on a logarithmic scale. As it can be seen, the standard AND/OR graph has an exponential complexity in the number of legs (both online and offline phase) while the hierarchical representation computational time grows linearly. The standard deviation of the computational time

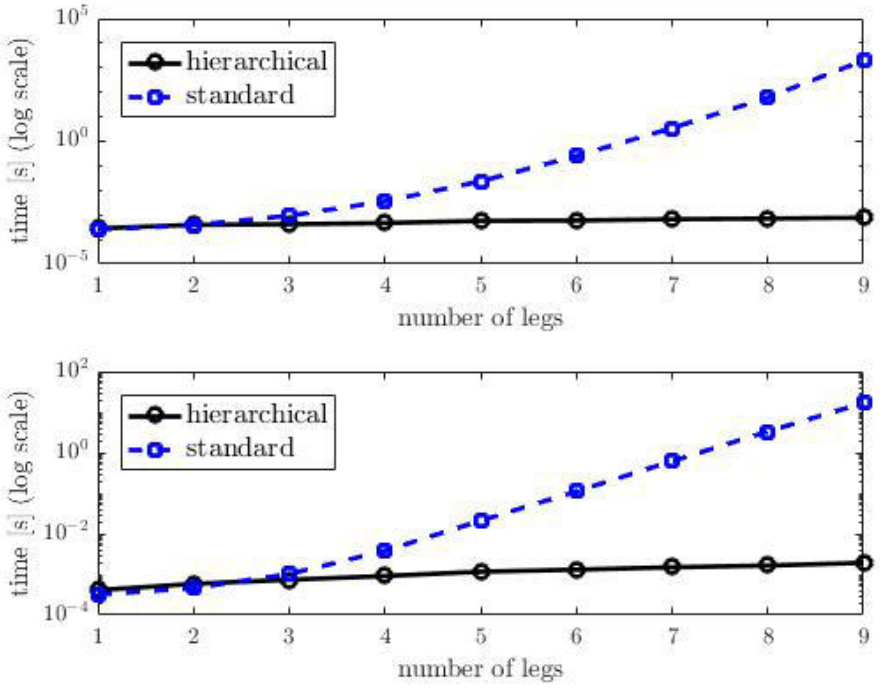


Figure 6.14: The mean computational time (logarithmic scale) of the hierarchical (solid black line) and standard (dashed blue line) *Task Representation* of the collaborative table assembly task with different number of legs. top: offline phase of the AND/OR graph, bottom: online phase of the AND/OR graph.

of the hierarchical AND/OR graph does not increase meaningfully both for the online and offline phase. The mean of the computational time for offline and online phases are 4.60×10^{-5} seconds and 9.81×10^{-5} seconds respectively, while in the standard AND/OR graph, the computational time increases from 1.22×10^{-5} seconds (one leg) to 18.40 seconds (nine legs) for the offline phase, and from 2.08×10^{-5} seconds to 1.13 seconds for the online phase.

Furthermore, as shown in Figure 6.13, using the Hierarchical AND/OR Graph the user can easily represent a table assembly task with different number of legs without going into the details of how the robot and human connect the legs to the tabletop, with great advantages in terms of scalability of the problems that can be tackled.

In conclusion, these experiments show how the increased expressiveness of the Hierarchical AND/OR graph eases the design of AND/OR graph and also makes it computationally more efficient.

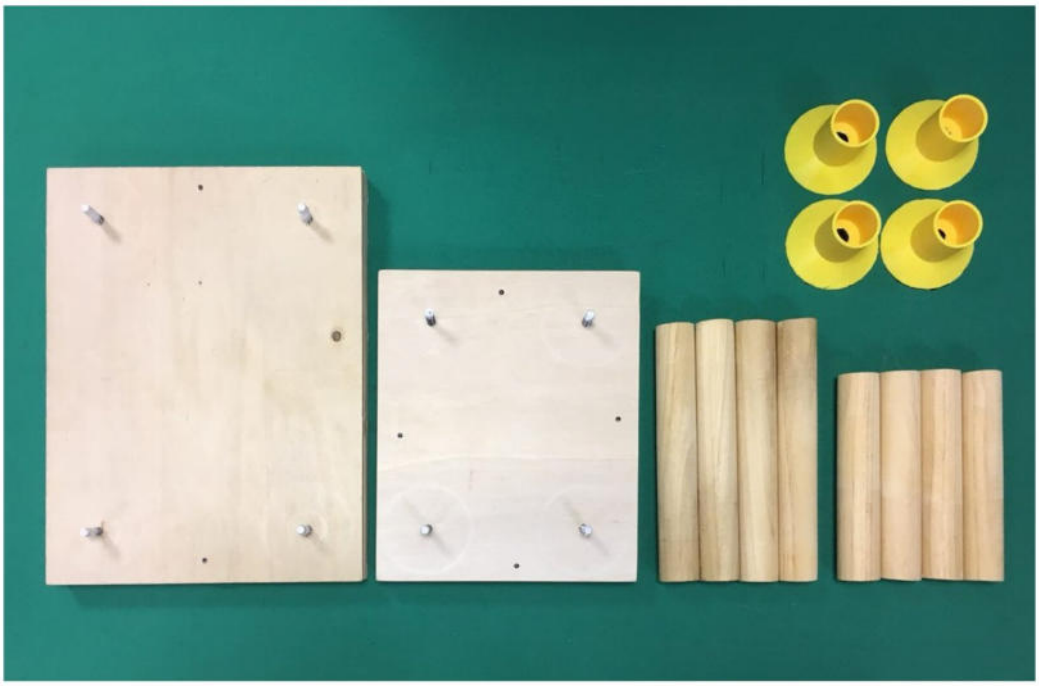


Figure 6.15: Different tabletops and legs for the table assembly task.

6.4 Collaborative Table Assembly Experiments

6.4.1 Scenario

In this set of tests, several human-robot cooperative table assembly tasks have been performed using different types of tables without any changes in the architecture, to show the architecture's flexibility at the task level. We have used two different rectangular tabletops, two sets of four legs, and four customized 3D print skirts (to guide the legs when placing them into the screws for precision compensation, and also to fix the legs to the tabletop), depicted in Figure 6.15. Hence, four different types of tables can be assembled which are not known a priori to the human or robot.

Initially, the legs and tabletop are located randomly in the human and robot shared workspace. The scenario representation encompasses all the possibilities, ranging from either the human or the robot assembling the whole table on their own, to all the combinations of human and robot cooperative assembly.

In these experiments, while the robot is manipulating the object, the human is standing in front of it and monitors the robot actions. In a real assembly line scenario, the human would monitor the task execution of *several* robots, moving between them and intervening whenever necessary.

The modeled human actions in this assembly scenario are:

- *pick up*, when the human initially stands and picks up one of the legs for manipulation;
- *put down*, when the human has performed the manipulation task and goes to his standing pose;
- *screwing*, when the human screws the leg to tabletop;

while the robot actions are:

- *approach*, when the robot approaches the grasping pose of an object;
- *transport*, when the robot transports an object to a desired goal position;
- *screw*, when the robot screws the leg to the tabletop;
- *unscrew*, when the robot rotates its end-effector in opposite direction of the screwing action;
- *grasp*, when the robot closes its gripper;
- *ungrasp*, when the robot opens its gripper;
- *update workspace*, when the robot updates its belief of the workspace using perception modules.

We model the table assembly task with four legs using a single-layer FOL AND/OR graph similar to the one shown in Fig. 6.13. The experiments demonstrate the architecture's flexibility at two levels:

1. Team level: the human and robot perform the same table assembly task with a different level of cooperation, from completely done by the robot to completely done by the human. This organizational flexibility is reached online, as the cooperation unfolds.
2. Task level: The human and robot cooperate on similar tasks, such as the assembly of several table types, without requiring different representations.

We reach these flexibilities through the proactive decision making and reactive adaptation of the *Task Manager*, the rich cooperation models of the *Task Representation* using first-order-logic AND/OR graph, the online object recognition and automatic computation of different object frames, and finally the action modules.

The software architecture to perform the table assembly is FlexHRC as shown in Figure 2.1 without including the *Path Planner* module. Examples of table assembly scenario can be viewed in the accompanying video[1].

[1]Please refer to: https://youtu.be/G7T1a_X9Dac.

Table 6.4: Computational performance of different modules in FlexHRC for the table assembly task.

Section	*avg time [s]*	*avg time [%]*	*std [s]*
Task Manager	0.01	0.00	0.00
Task Representation	0.57	0.23	0.01
Simulator	4.15	1.66	0.40
Robot Action	219.45	87.75	56.90
Human Action	25.82	10.32	5.67
TOTAL	250.09	100.00	51.87

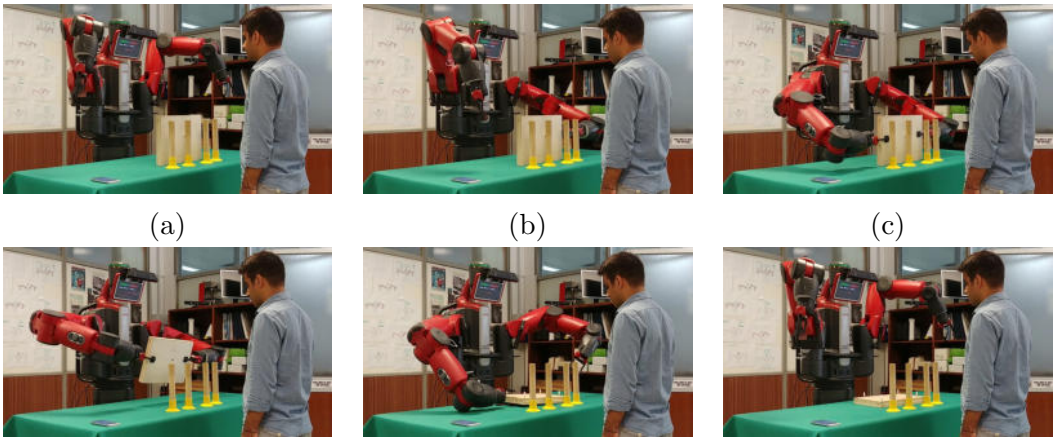
6.4.2 Computational performance

We have performed the table assembly test eight times with one volunteer. He was free to decide online how to proceed with the cooperation. The results are summarized in Table 6.4, and they show that the representation level and the decision making process, including the *simulator*, take less than 2% of the table assembly scenario, with a low standard deviation. The main sources of time consumption in these experiments are the human and robot actions (including the perception level). The high standard deviation of the experiments length is due to the human online decision, and significant difference between the human and robot speed to perform the given actions. In these experiments, the maximum robot joint angular velocity, and end-effector angular and linear velocities are 0.6 rad/s, 0.8 rad/s, and 0.4 m/s respectively.

6.4.3 Flexibility analysis

To avoid the repetition in showing the results and discussion, we subdivide the cooperative assembly task in three segments. In the first segment, the robot places the tabletop in the workspace (tabletop placement), the second segment is the connection of the legs to the tabletop (legs connection), and in the last segment the human monitors and controls the connections (monitoring).

Figure 6.16 shows the tabletop placement in the workspace. This task is always done only by the robot, and the robot can manage the task for different types of tabletops or in different poses autonomously. The left and right arms of the robot approach the tabletop (a-b), grasp it (c), change the orientation of the tabletop coordinately and place the tabletop on the table horizontally (d-e), and finally they return to the resting pose (f).



(a) (b) (c)

(d) (e) (f)

Figure 6.16: The sequence of actions associated with tabletop placement by the robot.

Figures 6.17, 6.18, 6.19, 6.20 show different situations in which the human and the robot connect the legs to the tabletop cooperatively. Depending on the workspace situation and the human online decisions, the human and robot define the amount of the cooperation online. The human and robot are free to choose which leg to pick up for manipulation, while the order of the bolts on the tabletop is fixed in our scenario.

Figure 6.17 shows the case in which the robot connects all the legs to the tabletop. At the beginning, the robot can choose between the four legs and its two arms. Hence, it evaluates the utility value for all eight options through the *Simulator* module. In this example, it chooses the second leg from the right and picks it up with its right arm (a-c). The human decides to not intervene in the assembly process, and since the robot can perform all the assembly tasks on its own, it does not ask the human for an intervention. Therefore, the robot follows the cooperation path with the blue color shown in Fig. 6.13.

Figure 6.18 shows the case where the human decides to connect all the legs to the tabletop. The robot tries, by default, to follow the minimum cost cooperation path (blue color in Figure 6.13), but as soon as it recognizes the human *Pick up* action, it adapts and follows the corresponding cooperation plan (red color in 6.13).

Figure 6.19 shows the case in which the human and robot cooperate together to connect all the legs to the tabletop. In particular, Figs. 6.19 (a-c) show how the robot approaches one of the legs, but the human intervenes in the middle and performs the first connection. The robot adapts to the human decision when it recognizes the human action. It updates its belief of the workspace using

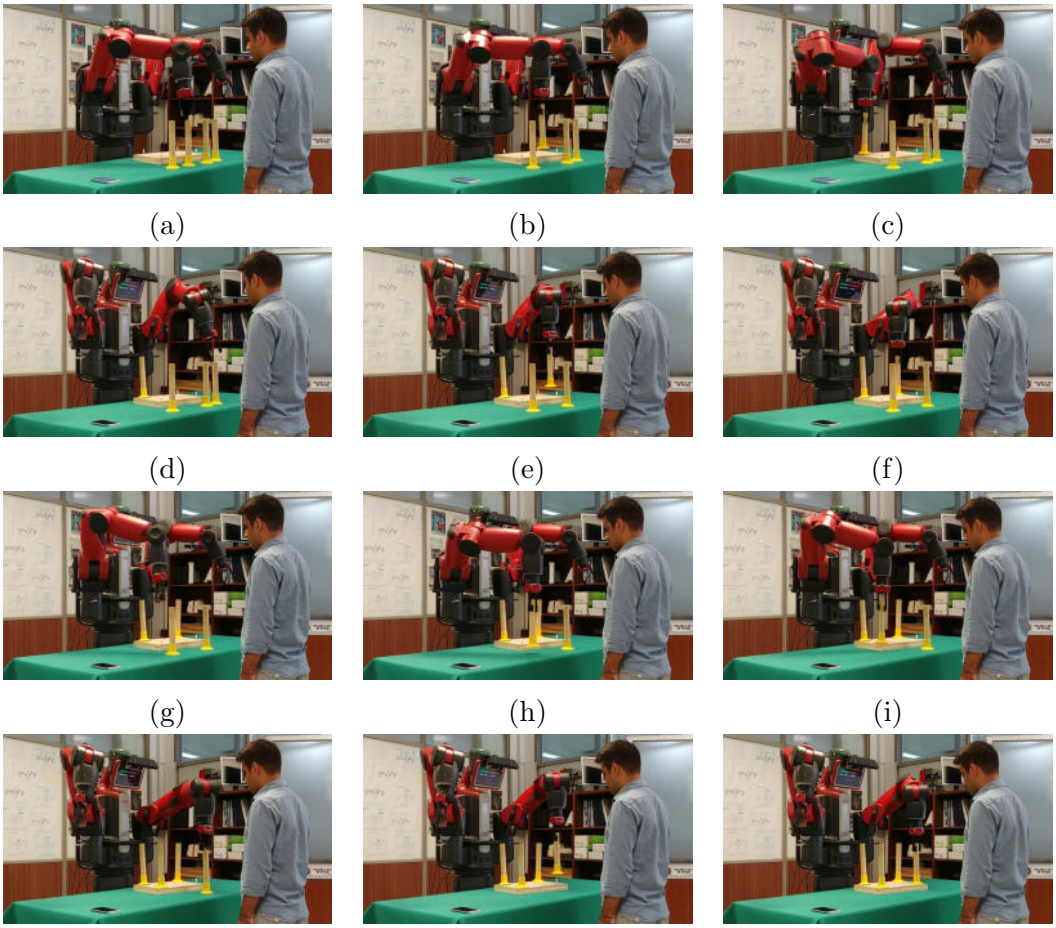


Figure 6.17: The robot connects all the legs to the tabletop.

the *Object and Scene Perception* module, it updates the AND/OR graph and determines the new feasible states. Later (Figs. 6.19 (d-f)), the robot decides to connect the leg on the left side of the workspace, grasping it with the left arm, to the second bolt of the tabletop. Then, the robot connects the third leg to the tabletop, and finally the human decides to perform the connection of the last leg.

Figure 6.20 demonstrates the robot reactive adaptation to the human decisions and proactive request to the human to perform a task which the robot cannot perform. In Figures (a-e) the robot connects the first two legs to the tabletop. Afterwards, (Figures (f-h)) the human decides to connect the third leg to the tabletop. Finally, Figures (i-l) show the situation where the robot cannot perform the given task and therefore it asks the human to connect the last leg to the tabletop. By following the green cooperation path in Figures 6.13, first the robot

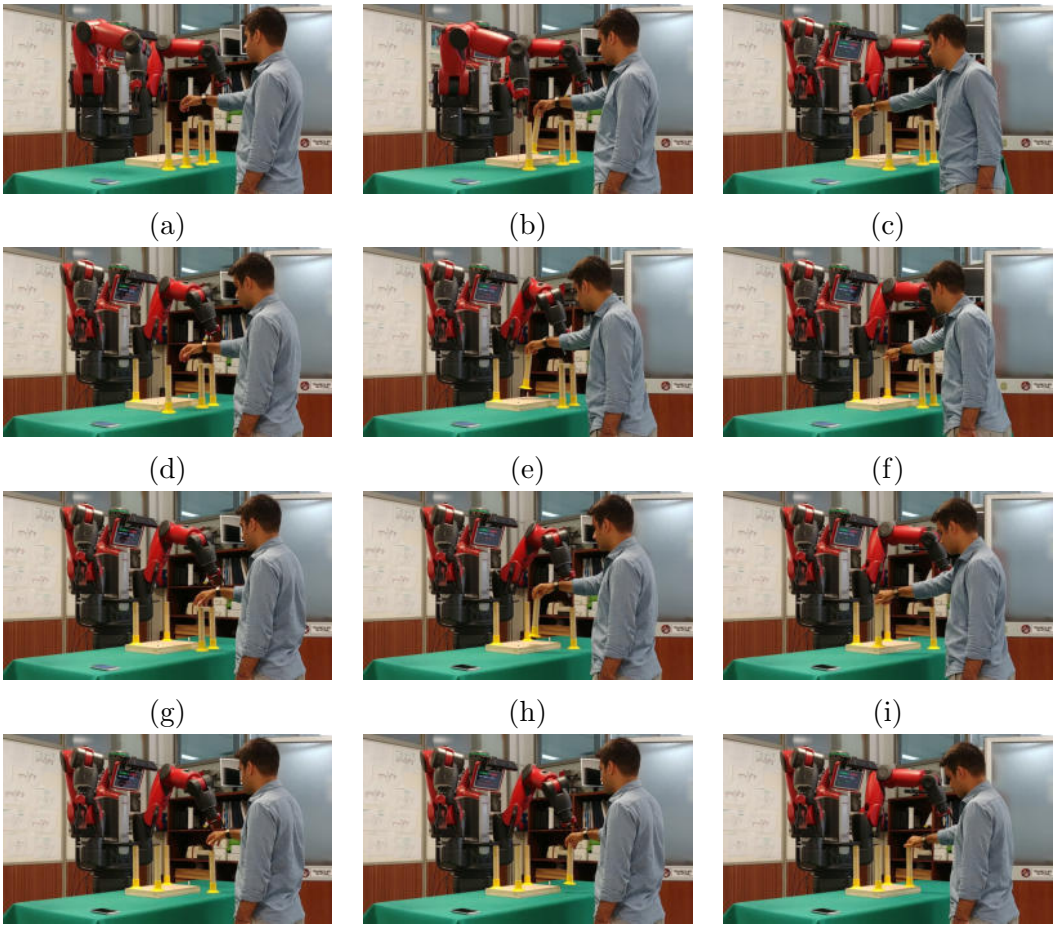


Figure 6.18: The human connects all the legs to the tabletop and the robot reactively adapts to the human online decision.

transports the leg in front of the human and finally the human connects it to the tabletop.

Figure 6.21 shows the human controlling the legs' connection to the tabletops at the end of the table assembly task. If the connections are loose, the human fixes them.

In Figures 6.17, 6.18, 6.20 the human and robot assemble the larger tabletop and longer legs, while in Figure 6.19 they assemble cooperatively the table with larger tabletop and smaller legs. These examples show the ability of the architecture to handle different parts for the same cooperation task without encoding new information. This feature not only increases the flexibility in task level but also supports the idea of the scalability.

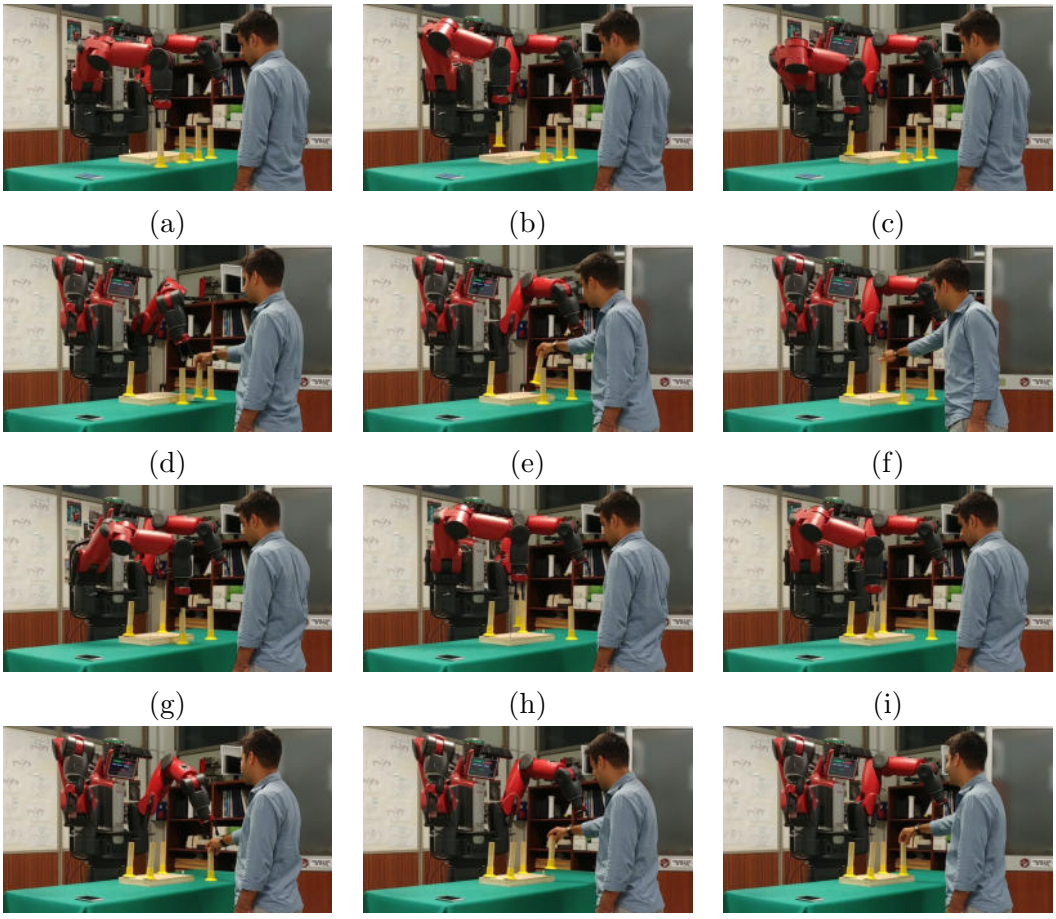


Figure 6.19: The human and robot connect the legs to the tabletop cooperatively; the robot adapts reactively to the human online decisions.

6.4.4 Decision making and simulation analysis

Figure 6.16 (a) shows the initial configuration of the workspace: the robot initializes the cooperation state, receives the first feasible state transitions from the *Task Representation* module and simulates them to allow the *Task Manager* to optimally allocate them to the agents. Figure 6.22 shows the configuration values in simulation (dashed lines) and reality (solid lines) for the *robot left arm* for the action *Approach(Tabletop)*, corresponding to scenes (a-c) of Figure 6.16. Figure 6.22 demonstrates that the simulation can reliably predict the robot behavior in case of small disturbances. Scenes (c-e) of Figure 6.16 show the placing of the tabletop in its final position. The tabletop weighs 1.4 Kg, and, to test a condition of *high disturbance*, this fact is purposefully not considered in the simulation.

Figure 6.20: The human and robot connect the legs to the tabletop cooperatively; the robot adapts reactively to the human decision online and asks proactively the human to perform a task.

Figure 6.23 shows the simulation and reality results for the *left arm* tool frame. As expected, the controller is able to compensate the high disturbances on the robot while performing the joint *transportation* action.

Once the tabletop is in its final position, the state of the cooperation is updated and the robot is given the new feasible states and state transitions, which all lead to the placement of a leg. The optimal path given by the *Task Representation*, is the blue path as shown in Figure 6.13. Therefore, the robot should try to connect a leg to the tabletop. Since this scenario assumes two robot agents and four legs in the workspace, the *Task Manager* and *Simulator* generates and tests eight simulation branches of the decision tree. On the basis of the simulation re-

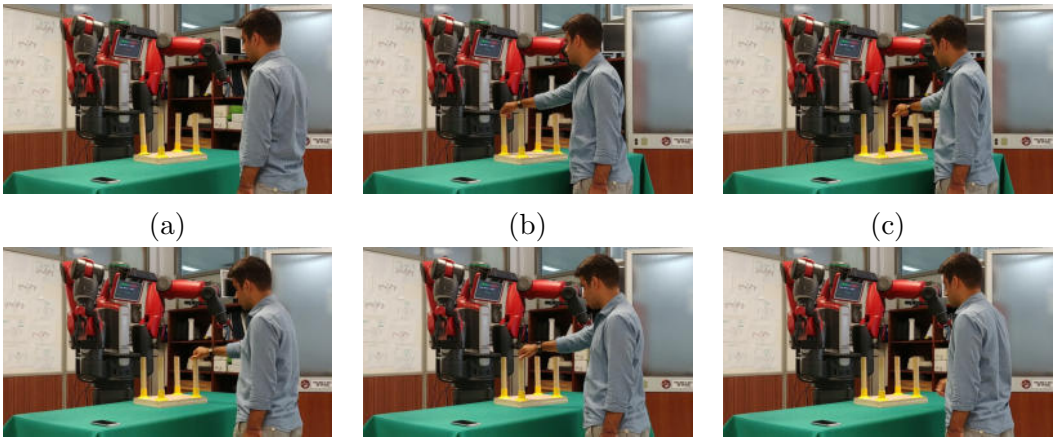


Figure 6.21: The sequence of actions the human performs to monitor the connections between the legs and the tabletop.

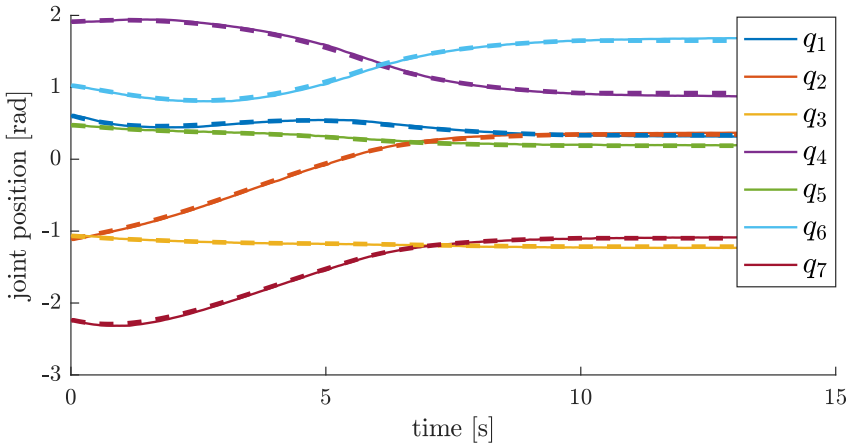


Figure 6.22: The real (solid line) and simulated (dashed line) joints positions of the *robot left arm* with low disturbance.

sults, the *Task Manager* assigns the action to the *robot left arm* or the *robot right arm* as shown in Figures 6.17, 6.18, 6.20, 6.19. Once, one leg and the tabletop are connected and the robot, upon analysing the workspace, again updates its *Task Representation*. The *Simulator* analyses the new feasible actions, generating six simulation branches (two agents and three remaining legs). This process is continued for the rest of the legs in the workspace that are not connected yet.

If the simulations fail: none of the robot arms is able to place any of the legs on the tabletop, given their current placement within the workspace (see Figure

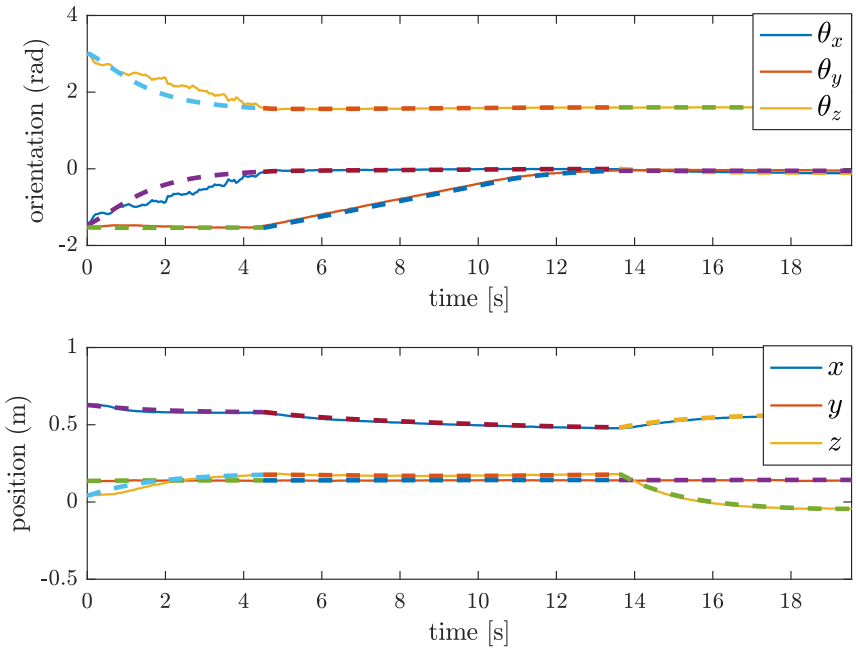


Figure 6.23: The real (solid line) and simulated (dashed line) tool frame orientation and position of the *robot left arm* with high disturbance.

6.24). This situation tests the ability of the FlexHRC architecture to recover from failures. As a result of the failure notification, the optimal state transition (hyper-arc in the AND/OR graph) becomes *unfeasible*, the path cost is updated and a new optimal path is identified by the *Task Manager* module using the Action-State table, which, in particular, requires the robot to move a leg in front of the human to let him or the other robot arm screw it to the tabletop , Figure 6.13 paths green or black. The new set of feasible states and state transitions found in *Task Manager* and, consequently, simulates. Since the simulation proves successful, the action is actually executed (scenes (j-k) of Figure 6.20). Having placed a leg in front of the human operator, in scene (k) the robot performs new simulations. As it fails (black path in Figure 6.13), the robot asks the person to connect a leg to the tabletop to proceed along the cooperative process (scene (l) of Figure 6.20 associated with green path in Figure 6.13).

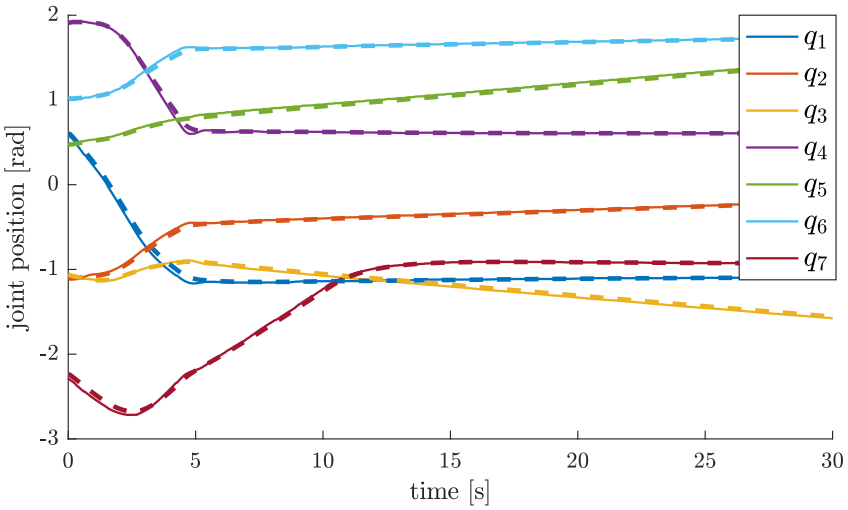


Figure 6.24: The real robot (solid line) and simulated robot (dashed line) joints position of the *left arm* for a failed action: some of the joint positions do not converge to constant values.

6.4.5 Symbolic fusion of object recognition and human action recognition modules

In this section we describe, how the *Object and Scene Perception* and the *Human Action Recognition* are integrated at the symbolic level so that we can recognize robustly the human actions and the cooperation state even if there are uncertainties and noises from the sensory information. Additionally, while the human and robot are cooperating, it is highly possible the field of view of the camera is occluded. To overcome this problem, some have proposed *active perception* strategies Bajcsy (1988); Bajcsy *et al.* (2018) or using multiple cameras Krumm *et al.* (2000); Wang (2013). In this work, we have proposed an integration between the wearable sensors and RGB-D information at the symbolic level, suitable for multi human-robot cooperation scenarios.

Figure 6.25 shows the shared workspace RGB-D data after clusterization step of Figure 6.25. As shown in Figure 6.16 the Kinect camera sensor is placed on top of the robot's display. Figure 6.25 (a-d) shows the workspace states associated with Figure 6.16 at the before and after placing the tabletop cooperation. When the robot is acting in the workspace the view of the robot is occluded, and therefore only by vision, FlexHRC is not able to recognize the robot places the tabletop horizontally in the workspace. In this case, we use the proprioceptive information of the robot via *Controller* to recognize the robot successful actions while executing the given actions, and later it updates the cooperation state using

the *Object and Scene Perception*. In Figure 6.25, scenes (e-f), (g-h), (i-j), and (k-l) show the cases when one, two, three, or all the legs are connected to the tabletop. Only using the vision-based camera, the robot can not detect if the legs are connected well to the tabletop, therefore at the end when all the legs are connected, the human monitors and fixes all the connections.

While we use the proprioceptive information of the robot to recognize its successful action execution, in the case of the human we use the wearable sensors and therefore the information coming from the *Human Action Recognition*.

As mentioned before, in this experiment, we have modeled three human actions, i.e., *pick up*, *put down*, and *screwing*. Among them we present the *screwing* action model generated offline. Figure 6.26 shows the raw inertial data along x, y, z direction of the sensor body frame associated with Figure 6.18 scene (c). The extracted features of the action by using the low-pass filter, i.e. body and gravity accelerations, is shown in Figure 6.27 and 6.28. Finally, the GMR model of the *screwing* action projected in 2D space is represented in Figure 6.29 and 6.30.

Figure 6.31 shows the raw online acceleration of the human's right wrist. These acceleration data are associated with the human's actions of Figure 6.18. Using the Mahalonis distance (4.10) between the online incoming data and the GMR model of different actions, the online possibility pattern of the human activity is represented in Figure 6.32.

Considering Equation (3.20) to formalise the actions, the human *screwing* action is defined as: *screwing*(`Leg(?)`, `Tabletop(?)`, `Human(?)`). Using the *Human Action Recognition* module, we can classify (recognize) the performed human action online, e.g. *screwing* among the modelled ones, using the ID of the smartwatch we can identify which human has performed the action (in our scenario there is only one human). As shown in Figure 6.32, there are not meaningful difference between the possibility patterns for doing different *pick up*, *screwing*, and *put down* actions of the human related to different legs. In order to fully recognize the human action, we profit from the *Object and Scene Perception* module. Therefore, in the case of the *screwing* action, the manipulated leg and tabletop are grounded using the information coming from RGB-D data.

6.4.6 Discussion

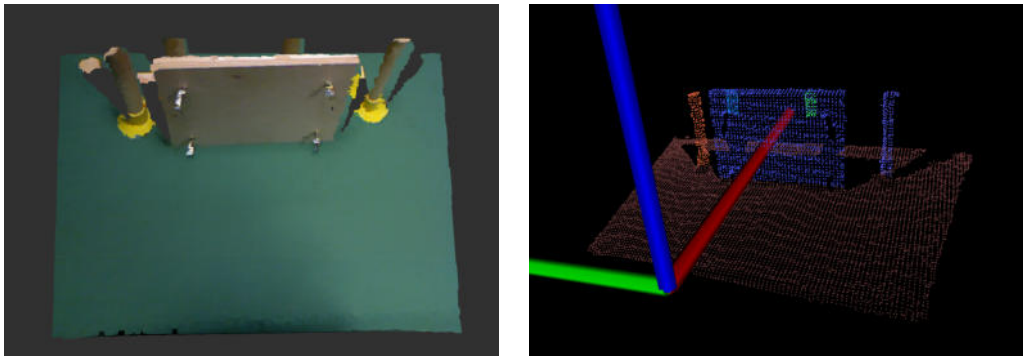
Regarding the functional requirement, in this paper we propose a simulation-based approach for the robot decision-making (F_6). The results of the simulation and decision-making include future robot trajectories, and human or robot actions. Communicating such information to the human results in increased effectiveness and safety of the collaboration, while reducing the cognitive load on the human operators (F_3, F_4). For a natural and intuitive interaction, the robot may

communicate future robot motions and actions (F_8) through the Augmented Reality (AR) technologies to the human Green *et al.* (2008); Michalos *et al.* (2016).

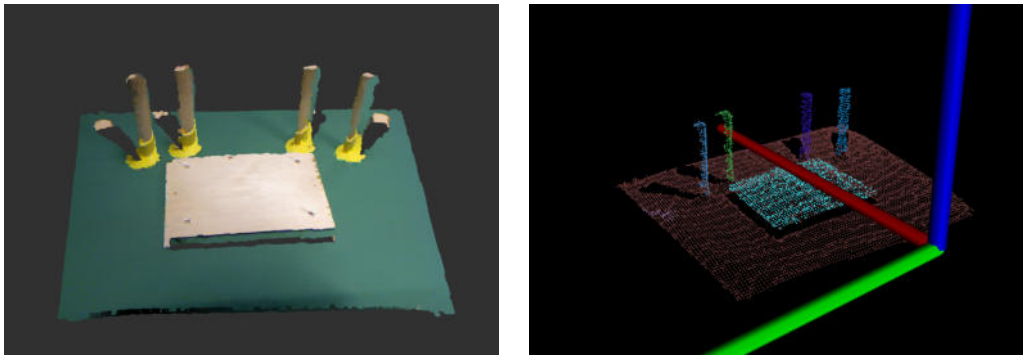
As mentioned in functional requirement, a collaborative robot architecture not only should provide the necessary autonomy for object manipulation, but it should also enable the collaborative robot to adapt reactively to the human behaviour online (F_5). This adaptation should be in the context of the available plans to reach the cooperation goal (F_2). In this paper, we propose an architecture which fulfills this requirement. FlexHRC equips the robot with the ability of autonomous object manipulation, exploiting the proposed rich perception and action modules. Using the AND/OR graph and the *Task Manager* modules, the architecture proposes all the possible plans, together with their associated cost, to the robot or the human throughout the cooperation. Among them, the robot always tries to follow the optimal plan option. If the human decides to follow another path, the robot reactively adapts to the human decision. These features of the FlexHRC allow to achieve a very high level of flexibility and autonomy.

Furthermore, the FlexHRC architecture separates the low-level uncertainties from the high-level structure of the task. The perception modules handle uncertainties in sensory data, while the uncertainties in action execution are handled in the *Robot Execution Manager* and *Controller* modules. These modules communicate to the *Task Manager* using high-level information at the semantic level (F_3 requirement).

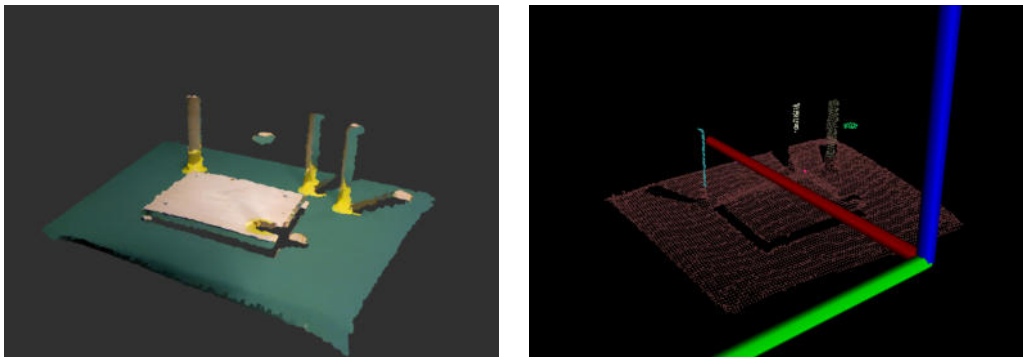
Finally, FlexHRC enables some scalability attributes for performing complex manufacturing and assembly tasks (F_9 requirement). In the *Task Representation* level, the introduction of the FOL and hierarchical AND/OR graph simplifies the definition of manipulation tasks and it greatly increases the computational efficiency in performing complex scenarios. This allows to build a library of simple tasks, which can be combined together to create a more advanced task, and so forth. Moreover, the *Task Manager* module provides the decision-making capabilities necessary in complex scenarios. Finally, by defining the primitive actions in *Robot Execution Manager*, we enable the robot to plan and perform a given task. In order to fully respond to the F_4 requirement, in a future implementation the robot should learn from the human demonstration and interaction with the environment both at the task and action levels.



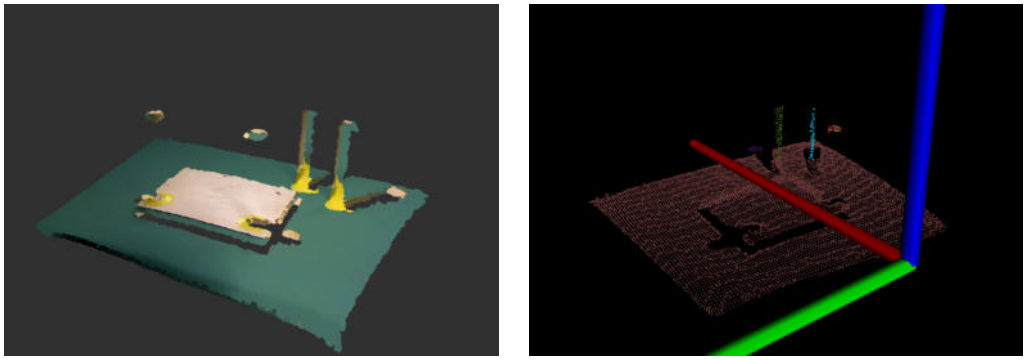
(a) (b)



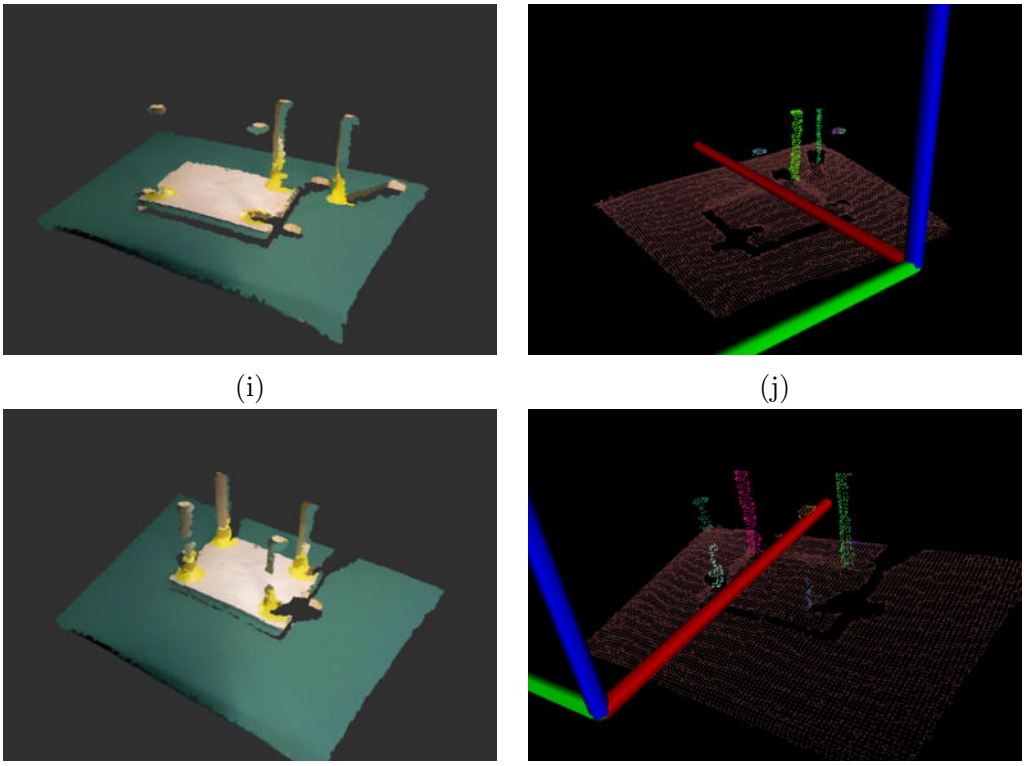
(c) (d)



(e) (f)



(g) (h)



(i) (j)

(k) (l)

Figure 6.25: The objects in the shared workspace using rgb-D data.

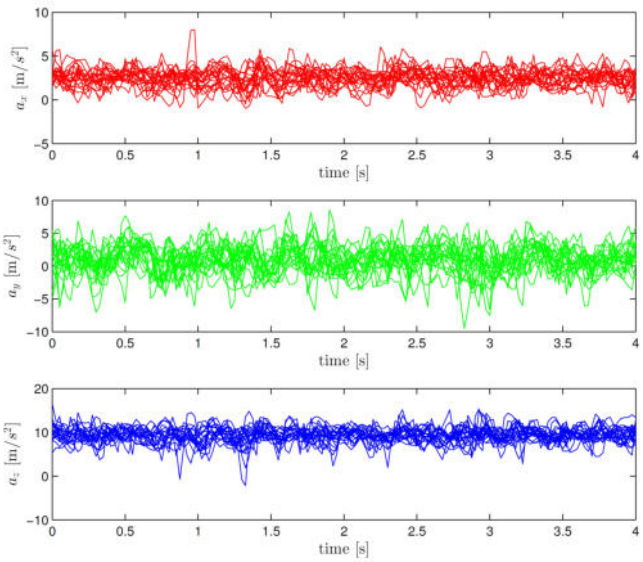


Figure 6.26: Raw inertial data associated with human *screwing* action of the table leg along x(red), y(green), z(blue) axis.

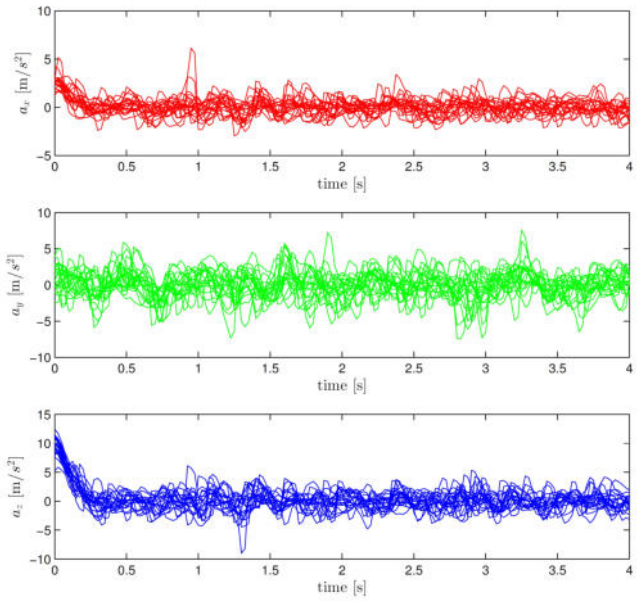


Figure 6.27: Body acceleration feature associated with human *screwing* action of the table leg along x(red), y(green), z(blue) axis.

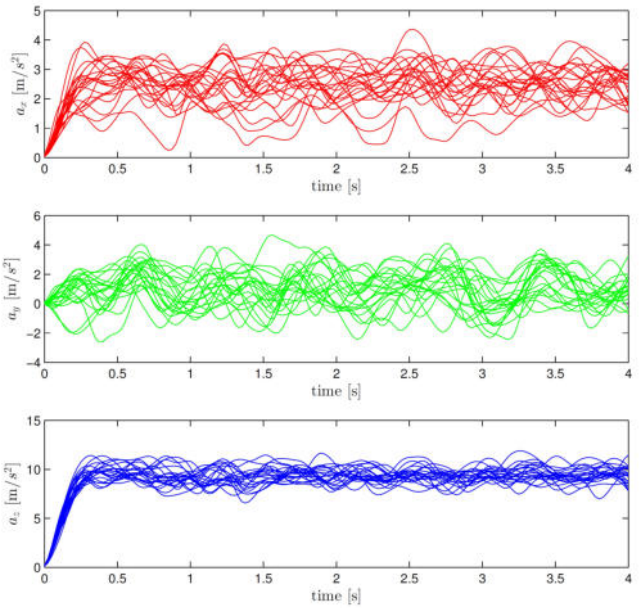


Figure 6.28: Gravity acceleration feature associated with human *screwing* action of the table leg along x(red), y(green), z(blue) axis.

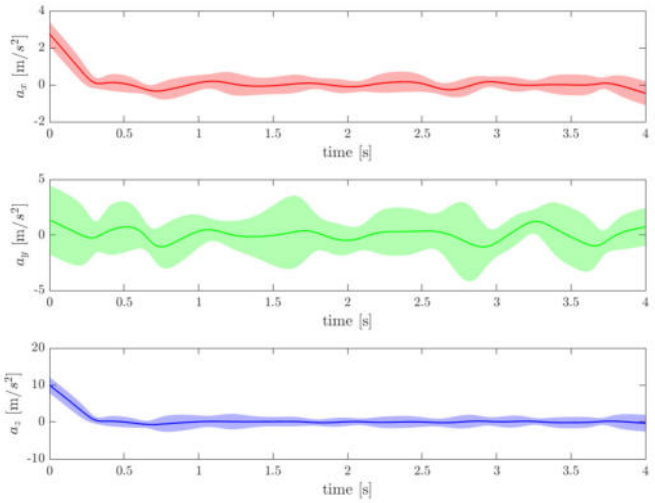


Figure 6.29: GMR model of the body acceleration associated with human *screwing* action of the table leg along x(red), y(green), z(blue) axis.

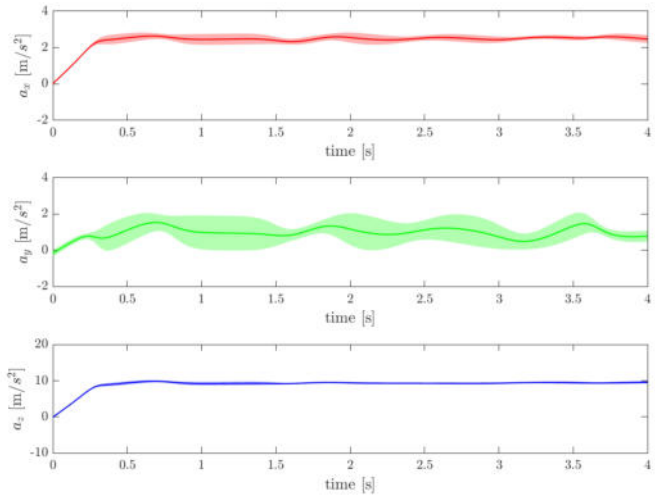


Figure 6.30: GMR model of the gravity acceleration associated with human *screwing* action of the table leg along x(red), y(green), z(blue) axis.

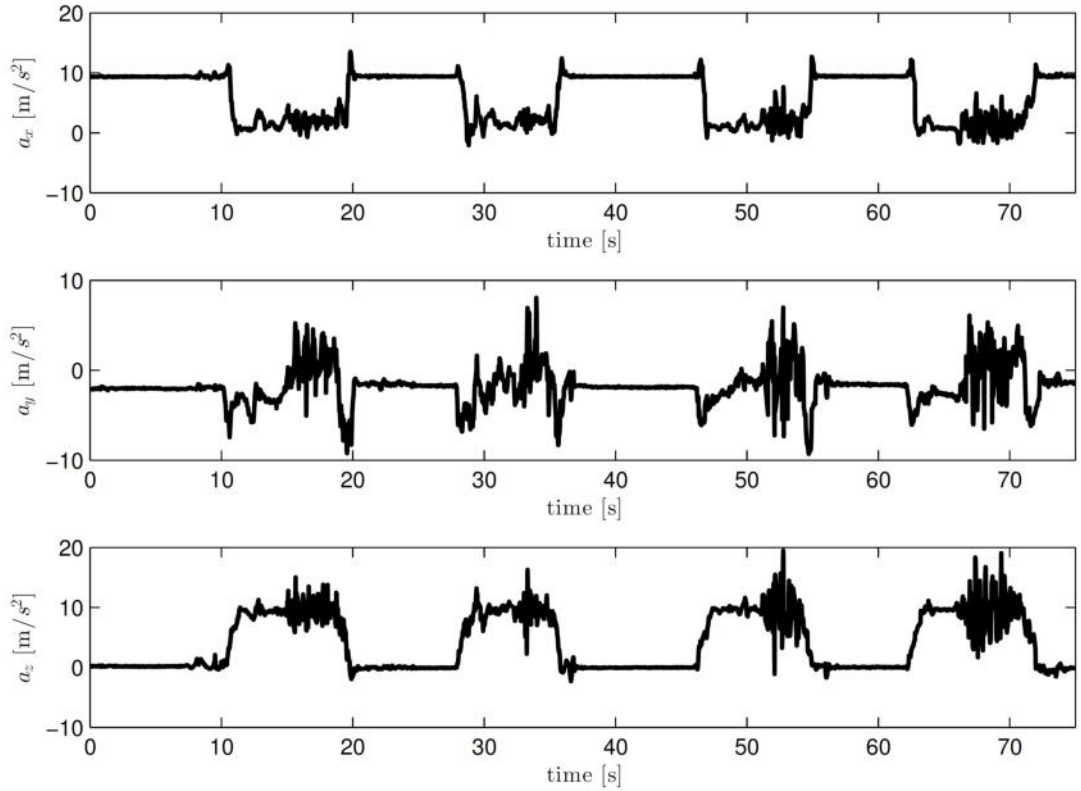


Figure 6.31: Online inertial data associated with the experiment of Figure 6.18.

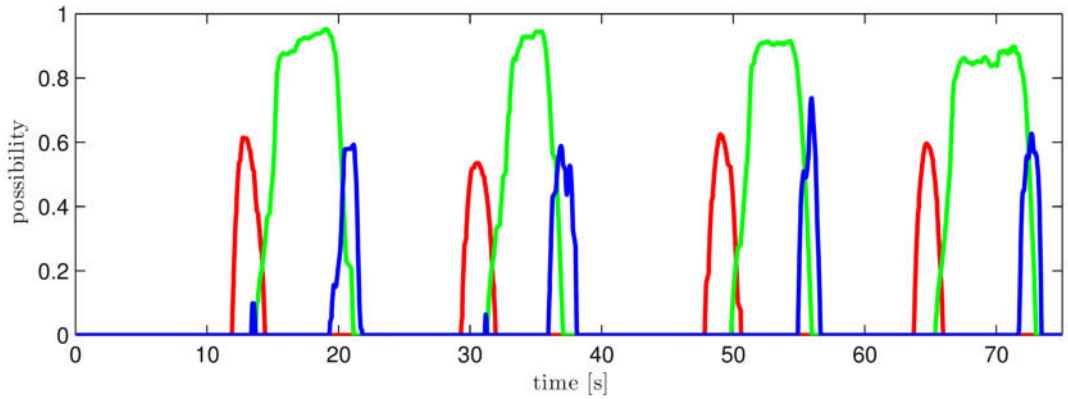


Figure 6.32: The trend of possibility of the human performing *pick up* (red color), *screwing*(green color), and *put down*(blue color) actions.

Chapter 7

Conclusions

7.1 Summary

In this dissertation, we have presented FlexHRC, a software architecture for flexible human-robot cooperation. FlexHRC is a novel approach to human-robot cooperation in shop-floor settings aimed at increasing the online flexibility, scalability, adaptation capabilities, and the controlled autonomy of collaborative robots. The architecture enables flexible and scalable manufacturing tasks in a partially structured and partially dynamic environment. FlexHRC estimates future robot behaviors online, exploits such estimates to take decisions as far as cooperation is concerned, allocates tasks to either humans or robots, and reactively adapts to human operator decisions. It is a hybrid reactive-deliberative architecture composed of three levels; namely *perception*, *representation*, and *action*. FlexHRC has been evaluated with several experimental scenarios including collaborative screwing task, coordinated object transportation in cluttered area, stand-alone task representation experiments, and finally collaborative table assembly task.

In Chapter 1, an introduction to human-robot cooperation has been provided. We described the motivation behind the FlexHRC, its principal applications in shop-floors and factories, and we outlined the specifications of such applications. According to this study, we explained the functional objectives that FlexHRC should enable.

Chapter 2 provides an overview and evaluation of the available architectures for collaboration of the human and robot in the literature. Using the functional requirements introduced in Chapter 1, this chapter determines the functional components of such architecture, the available methods to achieve such functionalities at the component level. According to this literature review, we introduce the software architecture for the collaborative robot, namely FlexHRC. Its com-

ponents are embraced in three levels, i.e. perception, action, and representation. We described interrelation among these levels and with the human to reach the desired goal.

In Chapter 3, we have addressed the representation level, where the cooperation process is defined, handled, and stored. It includes *Task Representation*, *Task Manager*, and *Knowledge Base* modules. The *Task Representation* module describes the cooperation task as an organized set of *states* and *state transitions*; online provides feasible states and state transitions to the *Task Manager*, in return the latter updates the *Task Representation* when the status of the cooperation changes. The change can be caused by the human operator, thus signalled to the *Task Manager* by the *Human Action Recognition* module or by the robot, therefore signalled by the *Robot Execution Manager*. *Task Representation* is founded on AND/OR graph, in which its notion is extended to first-order logic and hierarchical AND/OR graph. *Task Manager* allows the reactive adaptation to different situations through the action-state table, whereas the proactive decision making is endowed by the simulation-based decision tree. The shared workspace information is stored in the *Knowledge Base*, which is a data structure that maintains up-to-date the information related to the workspace (i.e., the objects therein) and the robot status.

Chapter 4 describes the perception level. It is composed of two modules, namely *Object and Scene Perception* and *Human Action Recognition* (HAR). The former provides information about the workspace states while the latter delivers the human actions. The *Human Action Recognition* module works in two stages. Offline, it models human actions using *Gaussian Mixture Modeling* and *Gaussian Mixture Regression* over inertial data provided by a wrist-worn inertial sensor. Online, it computes the Malahanobis distance between the continuous data stream and the modeled actions, therefore it provides the possibility of each modelled actions to match the data stream. Finally, it analyses the possibility patterns of all actions to recognize the execution of one of them. The *Object and Scene Perception* receives point cloud data corresponding to the robot's workspace using an RGB-D sensor mounted on top of the robot, applies a Euclidean distance metric for clustering the point cloud, and uses Random Sample Consensus (RANSAC) method to classify objects. Additionally, Principal Component Analysis (PCA) computes complementary features to manipulate the objects.

In Chapter 5, the action level is detailed. It manages the robot actions and consists of four modules, i.e., *Robot Execution Manager*, *Controller*, *Simulator*, and *Path Planner*; the last three being related to actions defined by the former. The execution of the actions can be carried out by the real or the simulated robot; the simulation results are used for feasibility checking and decision makings in representation level. *Robot Execution Manager* turns a high-level command into

one or a sequence of commands that *Controller*, *Simulator*, or *Path Planner* can execute. On the basis of the feedback provided by the *Controller*, *Simulator*, *Path Planner*, *Robot Execution Manager* reports back to the *Task Manager* about the success or failure of an executed or a simulated action. The *Path Planner* employs an extension of RRT* method to find a feasible path for the robot end-effector or transporting an object in the cluttered environment. The *Controller* employs a Task Priority framework to control robot motions at the kinematic level, and the *Simulator* module predicts the robot's behaviour in its workspace by simulating the robot's closed-loop kinematic model.

Chapter 6 provides an evaluation of FlexHRC by several experiments in terms of functional requirements introduced in Chapter 1. In collaborative screwing task, the human and robot collaborate together in order to connect a bolt to a plane hole. In this experiment, a simplified version of FlexHRC has been evaluated with limited functionalities. In the second experiment, coordinated transportation of an object in a cluttered environment, mainly the action level modules have been evaluated. In the third experiment, the task representation module has been tested in terms of efficiency and scalability. Finally, the collaborative table assembly scenario, as a real-world complex assembly task, is given to FlexHRC and according to the functional requirements, the system has been evaluated.

7.2 Discussion on Functional Requirements

In this section, we provide an overview and discussion on the degree to which FlexHRC has achieved the functional requirements mentioned in Chapter 1. The discussion has been done for shop-floor scenarios but is not limited to that and in fact, FlexHRC can be extended for other applications.

The introduced FOL and hierarchical AND/OR graph make the task representation efficient enough to perform different and complex scenarios, simplify the design phase, and ensure a human-like and predictable behaviour of the robot. As shown in Section 6.3, the proposed novel task representation does not grow exponentially as the scenario's complexity increases. However, the user is still required to manually design the graph, which requires him/her to know the collaboration task and the robot capabilities very well. This process is prone to error and can be time-consuming. The design of *Task Manager* and *Simulator* allow the efficient online decision making and adaptability as shown in Sections 6.1, 6.4. Although *Task Manager* leads to promising features for effective and efficient cooperation, it does not acknowledge the concurrent execution of different actions in a collaboration scenario. This feature may highly enhance the

efficiency of manipulation tasks.

The effectiveness, robustness to the uncertainties, and the reproducibility of FlexHRC for a scenario in different conditions are other important factors to address which are highly connected together. Experiments presented in Chapter 6 show that at the representation level, FlexHRC can achieve these objectives. The main problem comes from the perception and action level. In the perception level, if the human modelled actions are similar, the rate of correct recognition will decrease. Applying other methods for *Object and Scene Perception* such as deep learning methods Qi *et al.* (2017a,b) may provide a richer perception necessary for manipulation task when the objects are partially visible and perception is not limited to the primitive objects on horizontal plane support. In action level, the kinematic level control of FlexHRC and the lack of skill learning is a limiting component for performing more complex manipulation scenarios. To do complex manipulation or assembly scenarios, the controller should allow both force and motion control in robot task space. Moreover, a visual servoing method enhances the accurate placement which is a very common task in assembly scenarios.

Task Level Flexibility. At the task level, FlexHRC allows the robot to manipulate objects in different poses, shapes and sizes autonomously. When a task is defined, FlexHRC performs it without asking for new information from the human; it handles the low-level dissimilarities autonomously. The task flexibility is achieved utilizing different modules of FlexHRC including *Object and Scene Perception* to recognize the manipulation features of the objects in the workspace, *Task Representation* by the first-order logic description of the tasks in the workspace, and *Task Manager* and *Simulator* using the proactive decision making feature of the cooperation.

Team Level Flexibility. At the cooperative level, *Task Representation* and *Task Manager* allow the human or robot to choose the degree of cooperation on-the-fly, as the manufacturing task unfolds. To achieve this level of flexibility in team level, the robot uses the *Human Action Recognition* module to adapt reactively to the human's decision on-the-fly; and by applying online simulations make decisions based on the workspace conditions. *Task Representation* gives the possibility to the human and robot to choose the cooperation model among different available ones; while the proactive decision making and reactive adaptation feature of the *Task Manager* enable the team level flexibility.

Intelligibility & Intuitiveness. FlexHRC makes possible both the symbolic and numerical communication of the robot with the human. It provides visual and textual feedback to the human operator; It communicates the currently executing action (symbolic feedback) and its following path (numerical feedback) with the human. In the experiments we have performed in Chapter 6, using the available hardware, the robot gives the visual feedback on the robot front screen. This information can be given to the human via the augmented reality technology as

well Makris *et al.* (2016). Moreover, the robot takes an optimal decision and follows the minimum distance path while avoiding the obstacles; therefore, it allows the human to easily recognize the robot's intention.

Naturalness. FlexHRC enables natural cooperation between the human and robot concerning the communication modalities and the cooperation models. We obtained to some degrees natural communication between the human and the robot. Although a natural language speech-based communication is missing, the robot communicates via visual interfaces with the human in semantic level close to the human natural language. Moreover, the human communicates with the robot about his actions, through the *Human Action Recognition* module without an explicit effort from the human to convey his intention or actions to the robot. As a result, the robot adapts to the human on-the-fly decisions and does not force the human for following a strict cooperation model to reach the cooperation task objective. In fact, FlexHRC empowers both the human and the robot to select a convenient choice of the cooperation model among the available ones.

Adaptability and Decision Making. Task Representation provides several cooperation models, and *Task Manager* gives the possibility to human or robot to follow one of them such that the robot adapts to the human behaviour and to the workspace peculiar situations. The robot adapts to the human on-the-fly decisions by recognizing his actions, and it adjusts to the partially uncertain or unstructured environment by proactive decision making. The ability of the robot to simulate its future actions on-the-fly given the online perception information coming from the workspace and the next task to follow, permits the robot to make decisions on-the-fly. This adaptation and decision making eventually avoid failure of the cooperation.

Transparency. FlexHRC allows monitoring the human actions in an intuitive and natural way, without forcing the human to be in a specific place or orientation in front of the camera. This transparency is achieved using the perception modules along with the *Task Manager*. In fact, the use of wearable devices allows monitoring the human's actions continuously all around the shop-floor. In some cases for collaborative manipulation as shown in Section 6.4, only using the wearable devices does not allow to fully recognize the human action. Therefore, *Task Manager* integrates the incoming information from *Human Action Recognition* and *Object and Scene Perception* at symbolic level. This integration can be extended to other perception information as well.

Safety. Ensuring the safety of the human in the workspace of the collaborative robot is one of the main challenges in the field human-robot cooperation Lasota *et al.* (2017). FlexHRC enhances the safety in a limited manner, by providing a collaborative robot with predictable behaviour to the human. FlexHRC ensures the predictable behaviour of the robot both at motion and (symbolic) action level. In the action level, it follows the cooperation model which has the common sense

with the human daily life. Before performing an action, it provides symbolic level information to the human. Moreover, in motion level, the robot follows a minimum distance path while avoiding the collision with the objects in the workspace. Using the results of *Simulator* or *Path Planner*, FlexHRC can provide the robot future motions to the human by augmented reality technology.

Scalability. In terms of scalability, the user can perform complex manipulation scenarios building from simple tasks. To this aim, a new first-order-logic hierarchical task representation has been developed and integrated within FlexHRC. Moreover, the design of *Task Manager* and *Robot Execution Manager* allows the user to define new actions for a collaboration scenario. As mentioned before in Chapter 1, an important feature of a scalable collaborative robot would be the learning capability both at the semantic action or task level and at the skill or action execution level Argall *et al.* (2009); Ekvall & Kragic (2008); Konidaris *et al.* (2012); Nikolaidis *et al.* (2015); Stopp *et al.* (2003). While at the task level the robot learns the sequences of semantic and discrete actions Ekvall & Kragic (2008); Munzer *et al.* (2017), at the skill level it learns controlling its movements to reach a given goal Argall *et al.* (2009); Kormushev *et al.* (2010). The robot may learn through the interaction with the world and/or human or through the human demonstration Argall *et al.* (2009). The desired scalability in an industrial or shop-floor environment is possible only if the robot learns new skills with a limited number of trials.

Appendix A

Software Implementation

In this appendix, to complement the theories described in Chapter 3, the offline description of the *Task Representation* and *Task Manager* are described.

A.1 Task Representation

Task Representation module is developed as a ROS *service* Quigley *et al.* (2009). Starting the process, it loads the AND/OR graph description from a *text* file, and generates the AND/OR graph G. Online, upon the request of the *Task Manager*, it updates the graph G, and returns new feasible nodes and hyper-arcs as described in Chapter 3.

To generate the AND/OR graph, we describe it with the following structure offline:

```
#1st line:

[AND/OR graph name] [No. nodes (N)] [root node name]

#list all the nodes in the AND/OR graph:

[node_1 name] [node_1 cost]

[node_2 name] [node_2 cost]

...

[node_N name] [node_N cost]
```

```
#After the list of nodes, list all the hyper-arcs (H
↪ hyper-arcs):
[hyper-arc_1 name] [No. child nodes(NC1)] [parent node name]
↪ [hyper-arc_1 cost]
[1st child node name]
...
[NC1'th child node name]
[hyper-arc_2 name] [No. child nodes(NC2)] [parent node name]
↪ [hyper-arc_2 cost]
[1st child node name]
...
[NC2'th child node name]
....
....
....
[hyper-arc_H name] [No. child nodes(NCH)] [parent node name]
↪ [hyper-arc_H cost]
[1st child node name]
...
[NCH'th child node name]
```

In the case of a hierarchical AND/OR graph, only the description of the hyper-arcs will change:

```
[hyper-arc_1 name] [No. child nodes(NC1)] [parent node name]
↪ [hyper-arc_1 cost] [hyper-arc_1 lower level graph file name]
```

```
[1st child node name]

...

[NC1'th child node name]
```

If a hyper-arc does not hold a lower level AND/OR graph, we simply put a dash ("-") character.

The description of the hierarchical AND/OR of a table assembly scenario with two legs, presented in Figure 6.13, is as following:

```
TableAssembly 7 Table_FinalPose

Plate_initialPose 0

Plate_assemblyPose 0

Leg1_initialPose 0

Leg2_initialPose 0

Leg1_Plate_connected 0

Leg2_Plate_connected 0

Table_FinalPose 0

h0 1 Plate_assemblyPose 1 -

Plate_initialPose

h1 2 Leg1_Plate_connected 1 basic_connection

Leg1_initialPose

Plate_assemblyPose

h2 2 Leg2_Plate_connected 1 basic_connection

Leg2_initialPose

Leg1_Plate_connected

h3 1 Table_FinalPose 1 -

Leg2_Plate_connected
```

In this description `basic_connection` is the lower level AND/OR file name which exists in the same path (folder) of the higher level AND/OR graph. This AND/OR graph describes how the human or the robot can cooperatively connect a leg to a plate (tabletop) using FOL AND/OR:

```
ConnectLegPlate 4 Leg_Plate_Connected

Leg_Plate_Connected 0

Leg_initialPose 0

Leg_middlePose 0

Plate 0

h1 2 Leg_middlePose 2 -

Leg_initialPose

Plate

h2 2 Leg_Plate_Connected 1 -

Leg_initialPose

Plate

h3 1 Leg_Plate_Connected 1 -

Leg_middlePose

h4_human 1 Leg_Plate_Connected 2 -

Leg_middlePose

h5_human 2 Leg_Plate_Connected 5 -

Leg_initialPose

Plate
```

The interpretation of nodes and hyper-arcs titles given in the description of the table assembly AND/OR graph is presented in Table A.1.

Table A.1: Interpretation of titles in hierarchical AND/OR graph for the table assembly.

Title	Interpretation
`TableAssembly`	the name of the higher level AND/OR graph G^0 (how to assemble a table)
`ConnectLegPlate`	the name of the lower level AND/OR graph G^1 (how to connect a leg to a plate or tabletop)
`Table_FinalPose`	the root node name of G^0 (the table is assembled and it is in its final pose)
`Plate_initialPose`	a node name (the tabletop or plate is in its initial position)
`Plate_assemblyPose`	a node name (the plate is in assembly position)
`Leg`(i)`_initialPose`	a node name (the leg i is in initial position)
`Leg`(i)`_Plate_connected`	a node name (the leg i is connected to the plate)
`h`i	a hyper-arc name (a state transition from the child nodes to the parent node)
`basic_connection`	the file name of the lower level AND/OR Graph G^1
`Leg_middlePose`	a node name of Graph G^1 (a leg is placed in front of the human)
`Plate`	a node name of Graph G^1 (a plate or tabletop)
`Leg_Plate_Connected`	a root node name of Graph G^1 (a leg is connected to the tabletop)
`h`(i)`_human`	a hyper-arc name of graph G^1 (a state transition from the child nodes to the parent node and done by human)

A.2 Task Manager

As mentioned in Algorithm 17, when the *Task Manager* process starts, it loads the information of the cooperative agents, the description of actions that agents can perform, and the offline Action-State list.

The description of collaborative agents are:

```
[First agent name] [First agent type]

[Second agent name] [Second agent type]

...

[N'th agent name] [N'th agent type]
```

At each row, the user first defines the agent's name, and later defines if the agent is a "Human" or a "Robot". The human-type agents are allowed to change the cooperation process to another model (even if the models with lower cost are possible), and the robot-type agents will reactively adapt.

We define the symbolic description of the actions as:

```
[First action name] [First parameter predicate] ... [Last
↪ parameter predicate] [powerset of agents who can perform the
↪ first action]

...

[N'th action name] [First parameter predicate] ... [Last
↪ parameter predicate] [powerset of agents who can perform the
↪ N'th action]
```

At each line, first we determine an action name, list all the predicates of the parameters of the action, and finally, determine the powerset of agents who can perform the action. The relation between the subsets of the powerset is logical `OR`, while the relation between the elements of inside each subset is logical `AND`. If several agents should cooperate together to perform an action, the convention or agreement between agents for performing the joint action is defined either in *Robot Execution Manager* or in *Controller*.

The offline Action-State list describes the sequence of actions in order to solve a hyper-arc or meet a node as follows:

```
[First node or hyper-arc name] [First action] ... [Last action]

...

[Last node or hyper-arc name] [First action] ... [Last action]
```

In this description, at each row, first we determine the node or hyper-arc name used in the description of the AND/OR graph in *Task Representation*, then we determine the sequence of actions to be performed. Actions predicates and their responsible agents are either defined offline by the user or they are grounded online using the information of the workspace stored in *Knowledge Base*.

References

Adams, J.A. (2005). Human-robot interaction design: Understanding user needs and requirements. In *Proceedings of the 2005 Annual Meeting on Human Factors and Ergonomics Society (HFES)*, vol. 49, Orlando, FL, USA. 4, 12

Agrawal, P., Nair, A., Abbeel, P., Malik, J. & Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*. 13, 91

Ajoudani, A., Zanchettin, A.M., Ivaldi, S., Albu-Schäffer, A., Kosuge, K. & Khatib, O. (2018). Progress and prospects of the human–robot collaboration. *Autonomous Robots*, 1–19. 2

Arai, T., Pagello, E. & Parker, L.E. (2002). Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, **18**, 655–661. 12

Argall, B.D., Chernova, S., Veloso, M. & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, **57**, 469–483. 5, 64, 122

Arias-Castro, E., Donoho, D.L. *et al.* (2009). Does median filtering truly preserve edges better than linear filtering? *The Annals of Statistics*, **37**, 1172–1206. 52

Bajcsy, R. (1988). Active perception. 109

Bajcsy, R., Aloimonos, Y. & Tsotsos, J.K. (2018). Revisiting active perception. *Autonomous Robots*, **42**, 177–196. 109

Bauer, A., Wollherr, D. & Buss, M. (2008). Human-robot collaboration: a survey. *International Journal of Humanoid Robotics*, **05**, 47–66. 14

Bertenthal, B.I. (1996). Origins and early development of perception, action, and representation. *Annual Review of Psychology*, **47**, 431–459. 78

Bortot, D., Born, M. & Bengler, K. (2013). Directly or on detours? how should industrial robots approximate humans? In *Proceedings of the 2013 ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Tokyo, Japan. 3, 6, 12

Boutilier, C., Reiter, R. & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the 2001 International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, USA. 12

Bratman, M.E. (1992). Shared cooperative activity. *The philosophical review*, **101**, 327–341. 1

Bruno, B., Mastrogiovanni, F., Sgorbissa, A., Vernazza, T. & Zaccaria, R. (2012). Human motion modelling and recognition: A computational approach. In *Proceedings of the 2012 IEEE International Conference on Automation Science and Engineering (CASE 2012)*, 156–161, Seoul, Korea. 53

Bruno, B., Mastrogiovanni, F., Sgorbissa, A., Vernazza, T. & Zaccaria, R. (2013). Analysis of human behavior recognition algorithms based on acceleration data. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany. 7, 49, 51, 52, 53, 78

Bruno, B., Mastrogiovanni, F., Saffiotti, A. & Sgorbissa, A. (2014). Using fuzzy logic to enhance classification of human motion primitives. In A. Laurent, O. Strauss, B. Bouchon-Meunier & R.R. Yager, eds., *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 596–605, Springer International Publishing, Montpellier, France. 17, 49, 51, 52

Buchanan, B.G. (2005). A (very) brief history of artificial intelligence. *AI Magazine*, **26**, 53–60. 1

Buoncompagni, L. & Mastrogiovanni, F. (2015). A software architecture for object perception and semantic representation. In *Proceedings of the 2015 Italian Workshop on Artificial Intelligence and Robotics (AIRO)*, Ferrara, Italy. 17, 56, 89

Buoncompagni, L. & Mastrogiovanni, F. (2018). Dialogue-based supervision and explanation of robot spatial beliefs: a software architecture perspective. In *Proceedings of the 2018 IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Nanjing and Tai'an, China. 29

Buoncompagni, L., Mastrogiovanni, F. & Saffiotti, A. (2017). Scene learning, recognition and similarity detection in a fuzzy ontology via human examples. In *Proceedings of the Fourth Italian Workshop on Artificial Intelligence and Robotics (AIRO)*, Bari, Italy. 89

Caccavale, R. & Finzi, A. (2017). Flexible task execution and attentional regulations in human-robot interaction. *IEEE Transactions on Cognitive and Developmental Systems*, **9**, 68–79. 6, 10

Caccavale, R., Cacace, J., Fiore, M., Alami, R. & Finzi, A. (2016). Attentional supervision of human-robot collaborative plans. In *Proceedings of the 2016 IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, New York City, NY, USA. 10

Calandra, R., Owens, A., Jayaraman, D., Lin, J., Yuan, W., Malik, J., Adelson, E.H. & Levine, S. (2018). More than a feeling: Learning to grasp and regrasp using vision and touch. *arXiv preprint arXiv:1805.11085*. 61

Calinon, S., Guenter, F. & Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **37**, 286–298. 54

Capitanelli, A., Maratea, M., Mastrogiovanni, F. & Vallati, M. (2018). On the manipulation of articulated objects in human-robot cooperation scenarios. *CoRR*, **abs/1801.01757**. 6, 11, 12

Cartmill, E.A., Beilock, S. & Goldin-Meadow, S. (2011). A word in the hand: action, gesture and mental representation in humans and non-human primates. *Philosophical Transactions of the Royal Society B: Biological Sciences*, **367**, 129–143. 14

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N. & Carreras, M. (2015). Rosplan: Planning in the robot operating system. In *Proceedings of the 2015 International Conference on Automated Planning and Scheduling (ICAPS)*, Jerusalem, Israel. 12

Chaumette, F. & Hutchinson, S. (2006). Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, **13**, 82–90. 61

Chaumette, F. & Hutchinson, S. (2007). Visual servo control, part ii: Advanced approaches. *IEEE Robotics and Automation Magazine*, **14**, 109–118. 61

Chen, F., Sekiyama, K., Cannella, F. & Fukuda, T. (2014). Optimal subtask allocation for human and robot collaboration within hybrid assembly system. *IEEE Transactions on Automation Science and Engineering*, **11**, 1065–1075. 12, 13, 14

Chiaverini, S. (1997). Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, **13**, 398–410. 74

Claes, D. & Tuyls, K. (2014). Human robot-team interaction. In *Proceedings of the 2013 International Symposium on Artificial Life and Intelligent Agents (ALIA)*, Bangor, Wales, UK. 6, 11

Cohn, D.A., Ghahramani, Z. & Jordan, M.I. (1996). Active learning with statistical models. *Journal of artificial intelligence research*, **4**, 129–145. 53

Crandall, J.W., Oudah, M., Chenlinangjia, T., Ishowo-Oloko, F., Abdallah, S., Bonnefon, J., Cebrián, M., Shariff, A., Goodrich, M.A. & Rahwan, I. (2018). Cooperating with machines. *Nature Communications*, **9**, 233. 6, 11

Darvish, K., Bruno, B., Simetti, E., Mastrogiovanni, F. & Casalino, G. (2016). An adaptive human-robot cooperation framework for assembly-like tasks. In *Proceedings of the 2016 Workshop on Artificial Intelligence and Robotics (AIRO)*, Genoa, Italy. 92

Darvish, K., Bruno, B., Simetti, E., Mastrogiovanni, F. & Casalino, G. (2018a). Interleaved online task planning, simulation, task allocation and motion control for flexible human-robot cooperation. In *Proceedings of the 2018 IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Nanjing and Tai'an, China. 4, 6, 13, 16, 17, 18, 29, 41, 47

Darvish, K., Wanderlingh, F., Bruno, B., Simetti, E., Mastrogiovanni, F. & Casalino, G. (2018b). Flexible human-robot cooperation models for assisted shop-floor tasks. *Mechatronics*, **51**, 97–114. 5, 6, 11, 13, 16, 17, 18, 29, 30, 43

De Maesschalck, R., Jouan-Rimbaud, D. & Massart, D.L. (2000). The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, **50**, 1–18. 55

de Mello, L.S.H. & Sanderson, A.C. (1990). And/or graph representation of assembly plans. *IEEE Transactions on Robotics and Automation*, **6**, 188–199. 16, 18

Dempster, A.P., Laird, N.M. & Rubin, D.B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, **39**, 1–22. 53

DeSantis, A., Siciliano, B., DeLuca, A. & Bicchi, A. (2008). An atlas of physical human-robot interaction. *Mechanism and Machine Theory*, **43**, 253–270. 3, 4, 6

Ekvall, S. & Kragic, D. (2008). Robot learning from demonstration: A task-level planning approach. *International Journal of Advanced Robotic Systems*, **5**, 33. 122

Erol, K., Hendler, J.A. & Nau, D.S. (1995). Semantics for hierarchical task-network planning. Tech. rep., Maryland University, Institute for Systems Research, College Park, MD, USA. 10

Esmaeilian, B., Behdad, S. & Wang, B. (2016). The evolution and future of manufacturing: A review. *Journal of Manufacturing Systems*, **39**, 79–100. 1, 3

euRobotics aisbl (2014). Strategic research agenda for robotics in europe 2014–2020. [Accessed 2018-08-04]. 1, 3

euRobotics aisbl, European Robotics Association & European Commission (2017). Robotics 2020 multi-annual roadmap for robotics in europe, horizon 2020 call ict-2017. [Accessed 2018-08-04, Release B, 02/12/2016]. 3

Ferland, F., Létourneau, D., Aumont, A., Frémy, J., Legault, M.A., Lauria, M. & Michaud, F. (2013). Natural interaction design of a humanoid robot. *Journal of Human-Robot Interaction*, **1**, 118–134. 5

Fischler, M.A. & Bolles, R.C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, **24**, 381–395. 17, 56

Fragkiadaki, K., Levine, S., Felsen, P. & Malik, J. (2015). Recurrent network models for human dynamics. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile. 90

Garca, P., Caamao, P., Duro, R.J. & Bellas, F. (2013). Scalable task assignment for heterogeneous multi-robot teams. *International Journal of Advanced Robotic Systems*, **10**, 105. 4, 6

Gerkey, B.P. & Matarić, M.J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, **23**, 939–954. 12, 41

Gleeson, B., MacLean, K., Haddadi, A., Croft, E. & Alcazar, J. (2013). Gestures for industry: Intuitive human-robot communication from human observation. In *Proceedings of the 2013 ACM/IEEE International Conference on Human-robot Interaction (HRI)*, Tokyo, Japan. 14

Goodrich, M.A. & Schultz, A.C. (2007). Human-robot interaction: a survey. *Foundations and Trends® in Human–Computer Interaction*, **1**, 203–275. 1, 2, 5, 12

Green, S.A., Chase, J.G., Chen, X. & Billinghurst, M. (2008). Evaluating the augmented reality human-robot collaboration system. In *Proceedings of the 2008 International Conference on Mechatronics and Machine Vision in Practice*, Auckland, New Zealand. 111

Grosz, B.J. (1996). Collaborative systems (aaai-94 presidential address). *AI magazine*, **17**, 67. 1

Hadfield-Menell, D., Russell, S.J., Abbeel, P. & Dragan, A. (2016). Cooperative inverse reinforcement learning. In *Proceedings of 2016 Conference on Advances in Neural Information Processing Systems*, Barcelona Spain. 90

Harari, Y.N. (2014). *Sapiens: A brief history of humankind*. Random House. 1

Hawkins, K.P., Bansal, S., Vo, N.N. & Bobick, A.F. (2014). Anticipating human actions for collaboration in the presence of task and sensor uncertainty. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China. 6, 10, 11, 13, 78

Hayes, B. & Scassellati, B. (2016). Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden. 3, 12

Helms, E., Schraft, R.D. & Hagele, M. (2002). rob@work: Robot assistant in industrial environments. In *Proceedings of the 2002 IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN)*, Berlin, Germany. 3, 78

Hoffman, G. & Breazeal, C. (2004). Collaboration in human-robot teams. In *AIAA 2004 Intelligent Systems Technical Conference*, Chicago, IL, USA. 1

Horaud, R., Dornaika, F. & Espiau, B. (1998). Visually guided object grasping. *Ieee Transactions on Robotics and Automation*, **14**, 525–532. 61

Huang, H., Li, D., Zhang, H., Ascher, U. & Cohen-Or, D. (2009). Consolidation of unorganized point clouds for surface reconstruction. *ACM transactions on graphics (TOG)*, **28**, 176. 58

Huber, M., Lenz, C., Wendt, C., Frber, B., Knoll, A. & Glasauer, S. (2013). Increasing efficiency in robot-supported assemblies through predictive mechanisms: An experimental evaluation. In *Proceedings of the 2013 IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Gyeongju, South Korea. 9, 14

International Federation of Robotics (2018). Robots and the workplace of the future. [Accessed 2018-08-04, positioning paper]. 3, 4

Johannsmeier, L. & Haddadin, S. (2017). A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters*, **2**, 41–48. 3, 6, 10, 13, 78

Karaman, S. & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, **30**, 846–894. 7, 65, 66

Karpas, E., Levine, S.J., Yu, P. & Williams, B.C. (2015). Robust execution of plans for human-robot teams. In *Proceedings of the 2015 International Conference on Automated Planning and Scheduling (ICAPS)*, Jerusalem, Israel. 6, 10

Kiesler, S. & Goodrich, M.A. (2018). The science of human-robot interaction. *ACM Transactions on Human-Robot Interaction (THRI)*, **7**, 9. 2

Kock, S., Vittor, T., Matthias, B., Jerregard, H., Kllman, M., Lundberg, I., Mellander, R. & Hedelind, M. (2011). Robot concept for scalable, flexible assembly automation: A technology study on a harmless dual-armed robot. In *Proceedings of the 2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, Tampere, Finland. 3, 4, 6

Konidaris, G., Kuindersma, S., Grupen, R. & Barto, A. (2012). Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, **31**, 360–375. 122

Koppula, H.S. & Saxena, A. (2013). Anticipating human activities for reactive robotic response. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan. 13

Kormushev, P., Calinon, S. & Caldwell, D.G. (2010). Robot motor skill coordination with em-based reinforcement learning. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan. 122

Krötzsch, M., Simancik, F. & Horrocks, I. (2012). A description logic primer. *arXiv preprint arXiv:1201.4089*. 48

Krüger, J., Lien, T. & Verl, A. (2009). Cooperation of human and machines in assembly lines. *CIRP Annals in Manufacturing Technology*, **58**, 628–646. 78

Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M. & Shafer, S. (2000). Multi-camera multi-person tracking for easyliving. In *Proceedings Third IEEE International Workshop on Visual Surveillance*, Dublin, Ireland. 109

Kuehn, J. & Haddadin, S. (2017). An artificial robot nervous system to teach robots how to feel pain and reflexively react to potentially damaging contacts. *IEEE Robotics and Automation Letters*, **2**, 72–79. 3

Kuffner, J.J. & LaValle, S.M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA. 66

Laber, E. (2008). A randomized competitive algorithm for evaluating priced and/or trees. *Theorical Computer Science*, **1**, 120–130. 36

Lasota, P.A., Fong, T. & Shah, J.A. (2017). A survey of methods for safe human-robot interaction. *Foundations and Trends® in Robotics*, **5**, 261–349. 6, 121

LaValle, S.M. & Kuffner Jr, J.J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, **20**, 378–400. 66

Lemaignan, S., Warnier, M., Sisbot, E.A., Clodic, A. & Alami, R. (2017). Artificial cognition for social humanrobot interaction: An implementation. *Artificial Intelligence*, **247**, 45–69. 3, 6, 10, 11, 12, 13, 14

Lenz, C. (2011). *Context-aware human-robot collaboration as a basis for future cognitive factories*. Ph.d. dissertation, Technische Universität München, München. 3

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. & Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, **37**, 421–436. 61

Levine, S.J. & Williams, B.C. (2014). Concurrent plan recognition and execution for human-robot teams. In *Proceedings of the 2014 International Conference on Automated Planning and Scheduling (ICAPS)*, Portsmouth, NH, USA. 6, 11, 13

Loehr, J.D., Sebanz, N. & Knoblich, G. (2013). Joint action: From perception-action links to shared representations. In W. Prinz, M. Beisert & A. Herwig, eds., *Action Science: Foundations of an Emerging Discipline*, chap. 13, 333, The MIT Press, Cambridge, Massachusetts. 78

Luger, G.F. (2009). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley Publishing Company, Boston, MA, USA, 6th edn. 12, 18

Mainprice, J., Hayne, R. & Berenson, D. (2015). Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA. 13

Makris, S., Karagiannis, P., Koukas, S. & Matthaiakis, A.S. (2016). Augmented reality system for operator support in human-robot collaborative assembly. *CIRP Annals - Manufacturing Technology*, **65**, 61–64. 91, 121

Mastrogiovanni, F., Paikan, A. & Sgorbissa, A. (2013). Semantic-aware real-time scheduling in robotics. *IEEE Transactions on Robotics*, **29**, 118–135. 90

Meneweger, T., Wurhofer, D., Fuchsberger, V. & Tscheligi, M. (2015). Working together with industrial robots: Experiencing robots in a production environment. In *Proceedings of the 2015 IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Kobe, Japan. 4

Michalos, G., Makris, S., Spiliotopoulos, J., Misios, I., Tsarouchi, P. & Chryssolouris, G. (2014). Robo-partner: Seamless human-robot cooperation for intelligent, flexible and safe operations in the assembly factories of the future. *Procedia CIRP*, **23**, 71–76. 6

Michalos, G., Makris, S., Tsarouchi, P., Guasch, T., Kontovrakis, D. & Chryssolouris, G. (2015). Design considerations for safe human-robot collaborative workplaces. *Procedia CIRP*, **37**, 248–253. 91

Michalos, G., Karagiannis, P., Makris, S., nder Tokalar & Chryssolouris, G. (2016). Augmented reality (AR) applications for supporting human-robot interactive cooperation. *Procedia CIRP*, **41**, 370–375. 111

Miller, A.T. & Allen, P.K. (2004). Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, **11**, 110–122. 61

Miyata, N., Ota, J., Arai, T. & Asama, H. (2002). Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *IEEE Transactions on Robotics and Automation*, **18**, 769–780. 12

Moe, S., Antonelli, G., Teel, A.R., Pettersen, K.Y. & Schrimpf, J. (2016). Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results. *Frontiers in Robotics and AI*, **3**, 16. 74

Morrison, D., Corke, P. & Leitner, J. (2018). Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *arXiv preprint arXiv:1804.05172*. 61

Müller, A., Kirsch, A. & Beetz, M. (2007). Transformational planning for everyday activity. In *Proceedings of the 2007 International Conference on Automated Planning and Scheduling (ICAPS)*, Providence, RI, USA. 6, 13

Munzer, T., Toussaint, M. & Lopes, M. (2017). Preference learning on the execution of collaborative human-robot tasks. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*. 122

Murray, R.M. (2017). *A mathematical introduction to robotic manipulation*. CRC press. 61

Nicolescu, M.N. & Matarić, M.J. (2002). A hierarchical architecture for behavior-based robots. In *Proceedings of 2002 International Joint Conference on Autonomous Agents and Multiagent Systems: part 1 (AAMAS)*, Bologna, Italy. 11

Nikolaidis, S., Gu, K., Ramakrishnan, R. & Shah, J.A. (2014). Efficient model learning for human-robot collaborative tasks. *arXiv preprint arXiv:1405.6341*. 90

Nikolaidis, S., Ramakrishnan, R., Gu, K. & Shah, J. (2015). Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. In *Proceedings of the 2015 Annual ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Portland, OR, USA. 122

Nikolaidis, S., Hsu, D. & Srinivasa, S. (2017). Human-robot mutual adaptation in collaborative tasks: Models and experiments. *The International Journal of Robotics Research*, **36**, 618–634. 6, 11

Nilsson, N.J. (1984). Shakey the robot. Tech. rep., SRI International, Menlo Park, CA, USA. 1

Norman, D. (2013). *The design of everyday things: Revised and expanded edition*. Constellation. 2

Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W. & Musen, M.A. (2001). Creating semantic web contents with protege-2000. *IEEE intelligent systems*, **16**, 60–71. 48

O'Leary, G. (2018). Point cloud library (pcl) documenation: How to use random sample consensus model. [Accessed 2018-11-24]. 59

Pedrocchi, N., Vicentini, F., Matteo, M. & Tosatti, L.M. (2013). Safe human-robot cooperation in an industrial environment. *International Journal of Advanced Robotic Systems*, **10**, 27. 3

Pineau, J., Montemerlo, M., Pollack, M., Roy, N. & Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, **42**, 271–281. 6

Protege (2018). Protege wiki. [Accessed 2018-11-11]. 48

Qi, C.R., Su, H., Mo, K. & Guibas, L.J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, **1**, 4. 120

Qi, C.R., Yi, L., Su, H. & Guibas, L.J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, 5099–5108. 120

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. & Ng, A.Y. (2009). Ros: an open-source robot operating system. In *Proceedings of the 2009 ICRA Workshop on open source software*, Kobe, Japan. 123

R. A. Giele, T., Mioch, T., A. Neerincx, M. & C. Meyer, J.J. (2015). Dynamic task allocation for human-robot teams. In *Proceedings of the 2015 International Conference on Agents and Artificial Intelligence (ICAART)*, Lisbon, Portugal. 13

Russell, S. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edn. 12, 18, 27, 41

Rusu, R.B. (2009). *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. Ph.D. thesis, Computer Science department, Technische Universitaet Muenchen, Germany. 58

Sahni, S. (1974). Computationally related problems. *SIAM Journal on Computing*, **4**, 262–279. 36

Sanderson, A.C., Peshkin, M.A. & de Mello, L.S.H. (1988). Task planning for robotic manipulation in space applications. *IEEE Transactions on Aerospace and Electronic Systems*, **24**, 619–629. 78

Saveriano, M., i. An, S. & Lee, D. (2015). Incremental kinesthetic teaching of end-effector and null-space motion primitives. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA. 90

Schnabel, R., Wahl, R. & Klein, R. (2007). Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, **26**, 214–226. 56

Sebanz, N., Bekkering, H. & Knoblich, G. (2006). Joint action: bodies and minds moving together. *Trends in Cognitive Sciences*, **10**, 70–76. 9

Sebastiani, E., Lallement, R., Alami, R. & Iocchi, L. (2017). Dealing with on-line human-robot negotiations in hierarchical agent-based task planner. In *Proceedings of 2017 International Conference on Automated Planning and Scheduling (ICAPS)*, Pittsburgh, PA, USA. 6, 10

Shah, J., Wiken, J., Williams, B. & Breazeal, C. (2011). Improved human-robot team performance using Chaski, a human-inspired plan execution system. In *Proceedings of the 2011 ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Lausanne, Switzerland. 3, 6, 9, 11, 14

Shah, J.A., Conrad, P.R. & Williams, B.C. (2009). Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *Proceedings of the 2009 International Conference on Automated Planning and Scheduling (ICAPS)*. 6, 13

Siciliano, B. & Slotine, J.J.E. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *Proceedings of the 1991 IEEE International Conference on Advanced Robotics 'Robots in Unstructured Environments' (ICAR)*, Pisa, Italy. 74

Simetti, E. & Casalino, G. (2016). A novel practical technique to integrate inequality control objectives and task transitions in priority based control. *Journal of Intelligent & Robotic Systems*, **84**, 877–902. 7, 17, 68, 72, 73, 78

Simetti, E., Casalino, G., Wanderlingh, F. & Aicardi, M. (2018). Task priority control of underwater intervention systems: Theory and applications. *Ocean Engineering*, **164**, 40 – 54. 17, 68

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S. & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China. 7, 78, 91

Steinfeld, A., Fong, T., Kaber, D., Lewis, M., Scholtz, J., Schultz, A. & Goodrich, M. (2006). Common metrics for human-robot interaction. In *Proceedings of the 2006 ACM SIGCHI/SIGART Conference on Human-robot Interaction (HRI)*, Salt Lake City, Utah, USA. 2, 4, 12

Stopp, A., Horstmann, S., Kristensen, S. & Lohnert, F. (2003). Toward interactive learning for manufacturing assistants. *IEEE Transactions on Industrial Electronics*, **50**, 705–707. 122

Toussaint, M., Munzer, T., Mollard, Y., Wu, L.Y., Vien, N.A. & Lopes, M. (2016). Relational activity processes for modeling concurrent cooperation. In *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden. 6, 12, 14

Tsarouchi, P., Michalos, G., Makris, S., Athanasatos, T., Dimoulas, K. & Chryssolouris, G. (2017). On a human-robot workplace design and task allocation system. *International Journal of Computer Integrated Manufacturing*, 1–8. 12, 13

Tuerker, S. (2018). Point cloud library (pcl) documenation: Euclidean cluster extraction. [Accessed 2018-11-22]. 58

Turing, A.M. (1950). Computing machinery and intelligence. *Mind*, **59**, 433–460. 1

Valdesolo, P., Ouyang, J. & DeSteno, D. (2010). The rhythm of joint action: Synchrony promotes cooperative ability. *Journal of Experimental Social Psychology*, **46**, 693–695. 9

Valli, A. (2008). The design of natural interaction. *Multimedia Tools and Applications*, **38**, 295–305. 5

Van Hees, V.T., Gorzelniak, L., Leon, E.C.D., Eder, M., Pias, M., Taherian, S., Ekelund, U., Renström, F., Franks, P.W., Horsch, A. *et al.* (2013). Separating movement and gravity components in an acceleration signal and implications for the assessment of human daily physical activity. *PloS one*, **8**, e61691. 53

Vernon, D. (2014). *Artificial cognitive systems: A primer*. MIT Press. 2, 15

Vernon, D., Metta, G. & Sandini, G. (2007). A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE transactions on evolutionary computation*, **11**, 151–180. 2

Vesper, C., Butterfill, S., Knoblich, G. & Sebanz, N. (2010). A minimal architecture for joint action. *Neural Networks*, **23**, 998–1003. 10, 78

Wang, X. (2013). Intelligent multi-camera video surveillance: A review. *Pattern Recognition Letters*, **34**, 3 – 19, extracting Semantics from Multi-Spectrum Video. 109

Wilcox, R., Nikolaidis, S. & Shah, J. (2012). Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. In *Proceedings of the 2012 Robotics: Science and Systems*, Sydney, Australia. 13

Wold, S., Esbensen, K. & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, **2**, 37–52. 17, 58

Yoon, S., Fern, A. & Givan, R. (2002). Inductive policy selection for first-order MDPs. In *Proceedings of the 2002 Conference on Uncertainty in Artificial Intelligence*, Alberta, Canada. 12