Copyright Information

Flexible Human-Robot Cooperation Models for Assisted Shop-floor Tasks

Kourosh Darvish, Francesco Wanderlingh, Barbara Bruno[1], Enrico Simetti, Fulvio Mastrogiovanni, Giuseppe Casalino

Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, via Opera Pia 13, 16145 Genoa, Italy.

Abstract

The Industry 4.0 paradigm emphasizes the crucial benefits that collaborative robots, i.e., robots able to work alongside and together with humans, 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. In this paper, a sensing, representation, planning and control architecture for flexible human-robot cooperation, referred to as FlexHRC, is proposed. FlexHRC relies on wearable sensors for human action recognition, AND/OR graphs for the representation of and the reasoning upon human-robot cooperation models online, and a Task Priority framework to decouple action planning from robot motion planning and control.

Keywords: Human-Robot Cooperation; Smart Factory; AND/OR graph; Task Priority control; Wearable Sensing.

1. Introduction

According to the Industry 4.0 paradigm, manufacturing is expected to undergo an important paradigm shift involving the nature of shop-floor environ-

[1]Corresponding author. *Email address*: barbara.bruno@unige.it

ments. 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. 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 [1]. Clearly, this proposal implies a number of challenges related to human-robot interaction both at the physical and the cognitive levels of the cooperation [2, 3, 4], which depend also on their type [5]. Beside basic safety considerations, which are a necessary prerequisite [6], a number of key issues must be taken into account: sensing and human activity recognition [7], definition of suitable cooperation models to reach certain goals [8, 9, 10], robot action planning and execution in the presence of humans [4], and the effect of robot's predictable behavior on the operator well-being and performance [11], just to name a few.

Among the possible use cases where human-robot cooperation can be particularly relevant, we consider cooperative assembly as a motivating scenario. If we focus on assemblage tasks, typically involving a small number of semifinished 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. These considerations can be synthesized in four functional specifications focusing on improving the operator's *working experience* [12].

F_1 [*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_2 [*Intelligibility*] 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_3 [*Adaptability*] In order for a robot to detect and classify meaningful actions carried out by an operator, it should not be necessary 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.

F_4 [*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.

In this paper, a sensing, representation, planning and control architecture for flexible human-robot cooperation, referred to as FlexHRC, is proposed. FlexHRC deals with the specifications outlined above by design, in particular enforcing flexibility at two different – yet related – levels.

R_1 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 [13], while the robot reacts to operators and plans for the next action accordingly [14, 15, 16].

R_2 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 [17, 18].

To this aim, FlexHRC implements a hybrid, reactive-deliberative human-robot cooperation architecture for *assisted cooperation* [5, 19] integrating different modules, namely: (i) human action recognition using *wearable sensors*, which do not pose any constraint on operator motions, to address F_4, and exploiting statistical techniques for action modeling [20] to take F_3 into account; (ii) representation of human-robot cooperation models and online reasoning using *AND/OR graphs* [21, 13, 10] to deal with F_1; (iii) control schemes based on a *Task Priority* framework to decouple human-robot action planning from robot motion planning and control [18], therefore addressing F_2.

The paper is organized as follows. Section 2 discusses related work. Cooperation models and the associated sensing, reasoning and robot motion processes are described in Section 3. Experimental results are presented and discussed in Section 4. Conclusions follow.

2. Background

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 [22, 23, 24], this analysis focuses on the human-robot cooperation process from the perspective of the functional specifications discussed above.

The problem of allowing humans and robots to perform open-ended cooperation by means of coordinated activity (F_1) did not receive adequate attention so far. An approach highlighting the challenge is presented in [8], where 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 offline, 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 behaviors 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 [15]. 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 [4] significantly extends the notions introduced in [15] to focus on *social* human-robot interaction aspects (F_2). 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 [10]. 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 offline in order to optimize graph-based metrics. As a matter of fact, the possibility of multiple cooperation models is provided for, although offline: 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 [13], 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.

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 [25]. Its application to human-robot cooperation tasks (F_3) has not been fully addressed in the lit-

erature. An approach in that direction is described in [26], 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 [27], 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.

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 (F_4), but at the price of introducing a few assumptions in the process [28, 29]. A Bayesian framework is used in [24] 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 [8], 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 [4], 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 [30]. 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

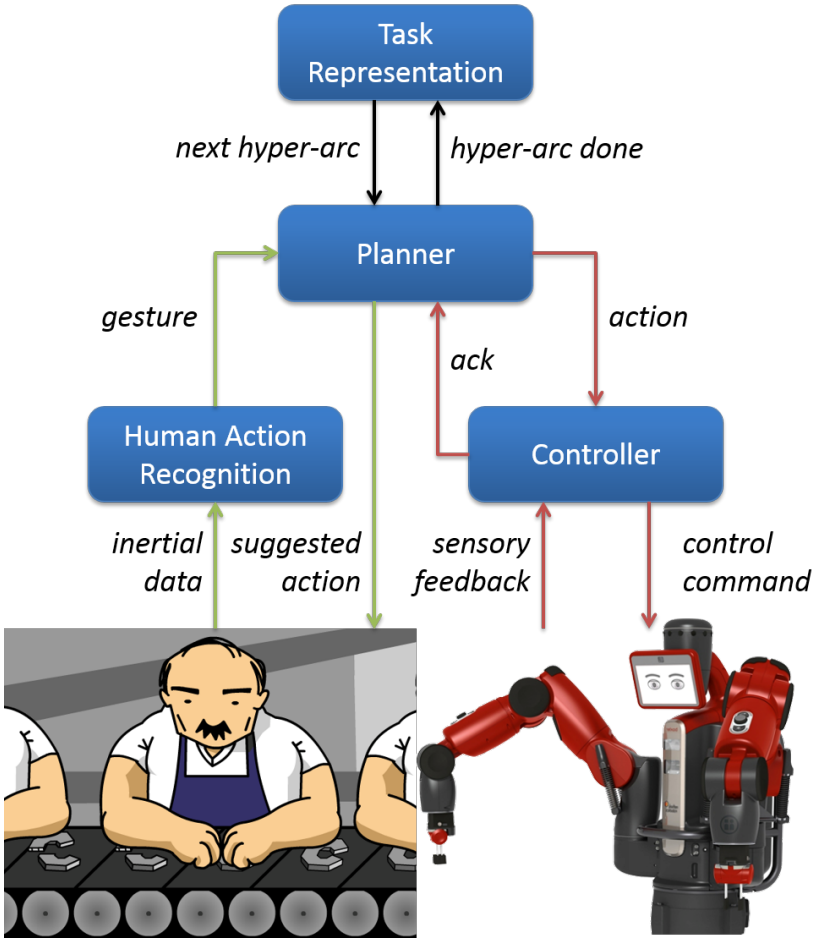


Figure 1: The FlexHRC system's architecture: modules and data flow.

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.

3. An Architecture for Flexible Human-Robot Cooperation

3.1. System's Architecture

FlexHRC is based on a distributed hybrid reactive-deliberative architecture, which is conceptually described in Figure 1. The architecture integrates recognition and classification of operator activities (handled by the *Human Action Recognition* module in the Figure), action planning for both operators and robots (jointly handled by the *Task Representation* and *Planner* modules in the Figure), as well as robot motion planning and execution (handled by the *Controller* module in the Figure). In particular:

- *Task Representation* maintains a set of models for the cooperation tasks to be carried out, and interacts with the *Planner* to decide which action an operator or a robot should execute next. Cooperation models are represented using AND/OR graphs, described in Section 3.2.

- The *Planner* operates on the AND/OR graph via an *ad hoc* online graph traversal procedure to determine the most appropriate sequence of actions to ground the cooperation on, based on the graph structure. These action sequences are encoded within graph edges, referred to as hyper-arcs. The *Planner* interacts with the *Human Action Recognition* module and the human operator (green loop in Figure 1) to assess and suggest human actions, while it interacts with the *Controller* (along the red loop in the Figure) to assess and suggest robot actions.

- Adopting the approach described in [20, 31], the *Human Action Recognition* module retrieves data from wearable devices and statistically classifies inertial data streams according to a number of predefined *gesture models*. As briefly discussed in the Introduction, the use of wearable devices to be worn by human operators is expected to allow them to move freely in the environment (in accordance with F_4), whereas the use of statistical classifiers should enforce the availability of gesture models not specifically tailored for different operators (F_3).

- The *Controller* is tasked with robot motion control and execution, and integrates the Task Priority control framework first discussed in [18]. The *Planner* does not have access to the *Controller*'s internal parameters and inner working, thereby enforcing F_1.

There is an important difference between the loop involving the *Planner* and the human, and the one involving the *Planner* and the robot. In the first case, the *Planner* simply *suggests* the next action to the operators, leaving them free to execute it or not and therefore taking into account functional requirement F_1, whereas in the second case it *imposes* the next action for the robot to perform. In both cases, the *Planner* receives a feedback, namely a gesture label corresponding to the action carried out by the operator or an acknowledge about the successful execution of an action by the robot.

It should be noted that FlexHRC assumes that only one operator interacts with the robot at a time and that he/she is cooperative and unwilling to *cheat* on the system. Since, as it will be evident reading the next Sections, 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.

3.2. Representation of Cooperation Models

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

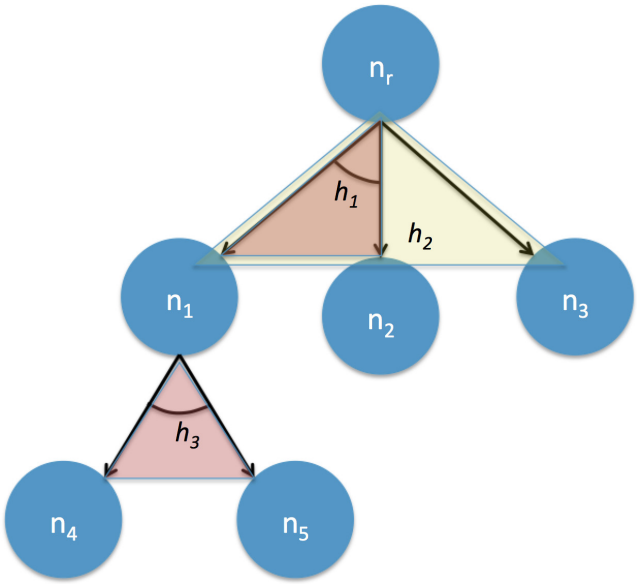


Figure 2: A generic AND/OR graph with six nodes and three hyper-arcs: h_1 and h_3 are *and* hyper-arcs, whereas h_2 is an *or* hyper-arc.

2 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 $d(h_i)$

Table 1: Definition of symbols related to 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	A 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}	Cost of node n_i
w_{h_i}	Cost 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

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 2), 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.

Table 1 recaps the above definitions for the reader's ease of reference.

All available cooperation models and cooperation paths are maintained by the *Task Representation* module. 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

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: $s(G) \leftarrow \mathit{false}$
3: **for all** $n \in N$ **do**
4: updateFeasibility(G, n)
5: **end for**
6: $\mathcal{P} \leftarrow$ generateAllPaths(G)
7: $n^* \leftarrow$ findSuggestion($\mathcal{P}$)

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[2]: (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).

[2]In the current version of FlexHRC, we do not use such standard formalisms as – for example – `xml` to represent this information. However, current work is devoted to integrate it with an ontology-based structure, which can be used as an appropriate formalism for data interoperability.

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** $f(n) = \mathit{true}$ **then**
2: *return*
3: **end if**
4: **if** $c(n) = \emptyset$ **then**
5: $f(n) \leftarrow \mathit{true}$
6: *return*
7: **end if**
8: **for all** $h \in H$ such that $p(h) = n$ **do**
9: allChildNodesSolved $\leftarrow \mathit{true}$
10: **for all** $m \in c(h)$ **do**
11: **if** $s(m) = \mathit{false}$ **then**
12: allChildNodesSolved $\leftarrow \mathit{false}$
13: **end if**
14: **end for**
15: **if** allChildNodesSolved $= \mathit{true}$ **then**
16: $a(h) \leftarrow \mathit{true}$
17: $f(n) \leftarrow \mathit{true}$
18: *return*
19: **end if**
20: **end for**
21: $f(n) \leftarrow \mathit{false}$
22: *return*

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)

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**

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 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

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
```

17 and 19) for later exploration.

When these procedures end, FlexHRC is ready for online cooperation. De-

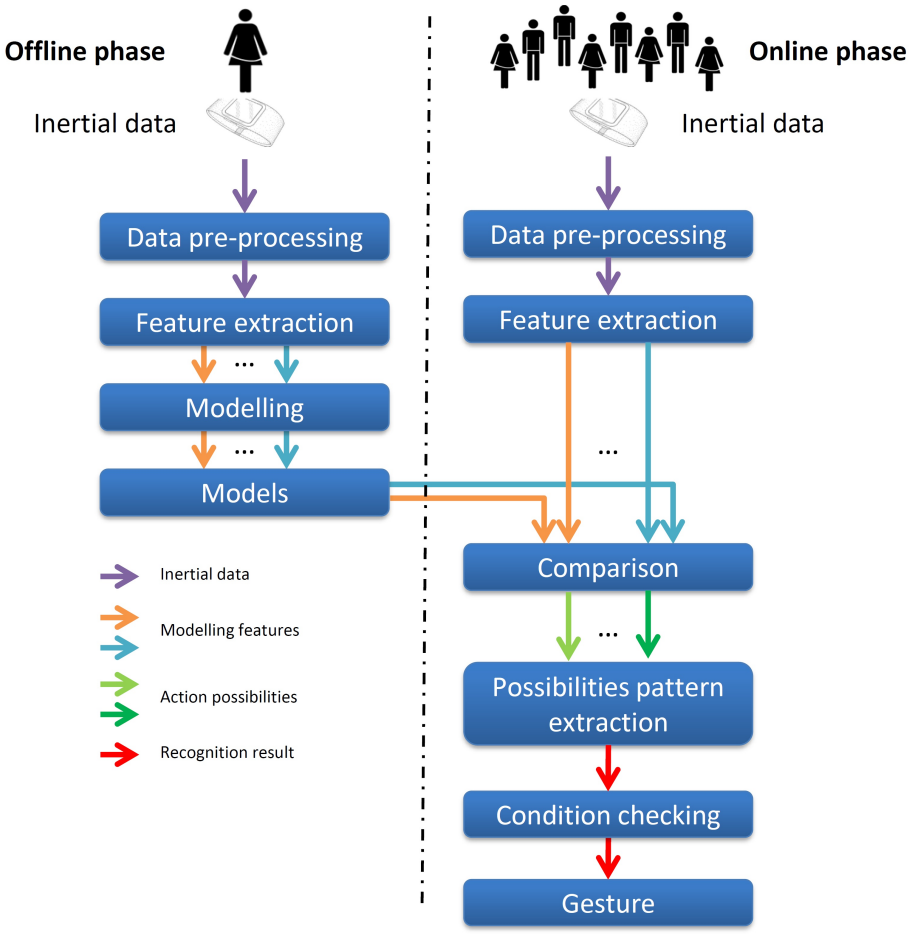


Figure 3: A schematic description of the *Human Action Recognition* module.

pending 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.

3.3. Recognition and Classification of Operator Actions

As shown in Figure 1 (green loop), operator actions affect both the *Human Action Recognition* and the *Planner* 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 3) employs a system for gesture recognition and classification first described in [20, 31]. 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 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.

While the cooperation process unfolds, *Human Action Recognition* executes a number of steps, in part similar to the procedure in the offline phase. Online, 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 [31]. 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 [20], 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

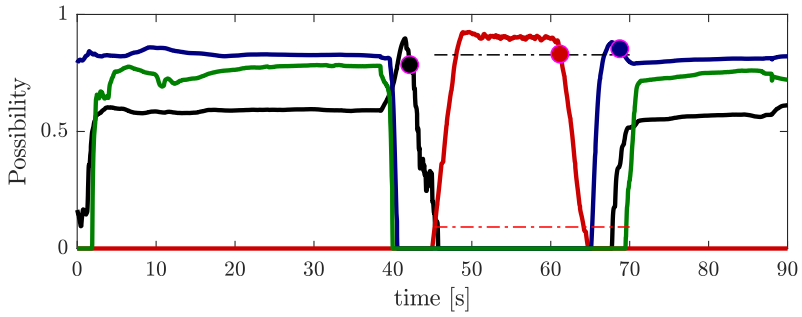


Figure 4: An example of action possibility evolutions for a cooperation task.

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 [31].

In FlexHRC, possibilities are used to determine which gesture has been executed [31]. 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 shows possibility values for four gestures, using different colors (see Section 4 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 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.

Once a gesture has been detected and classified, the gesture label is forwarded to the *Planner* module, which identifies the corresponding action a_j and checks whether it appears in the set of actions associated with the currently active hyper-arcs (anywhere or as next-in-sequence according to the ordering constraints). The set of currently active hyper-arcs is called *Action-State* table; for each hyper-arc in the *Action-State* table:

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

On the basis of the number of active hyper-arcs after the above check, the *Planner* updates the status of the cooperation:

- If there is only one active hyper-arc, 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 active hyper-arc, 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 active hyper-arcs, 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 active hyper-arcs, FlexHRC enters a *null* mode, inferring that an unexpected action occurred, and ends the cooperation.

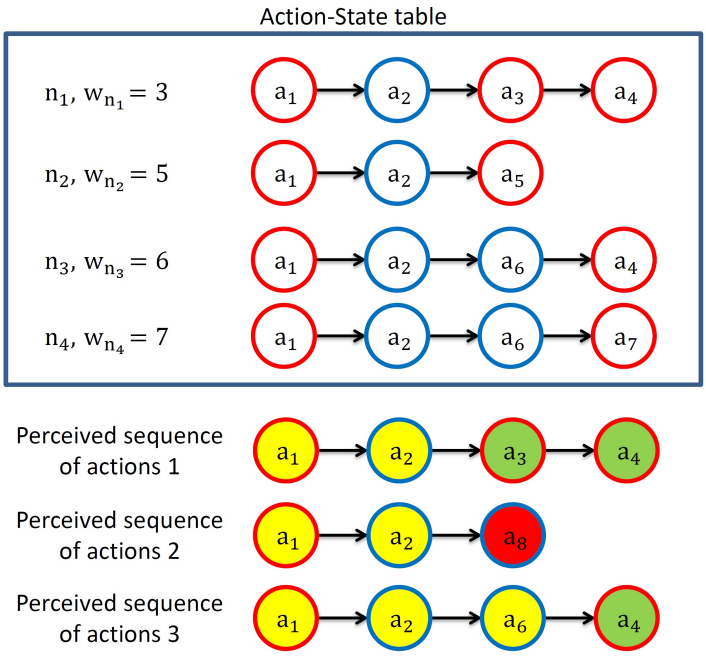


Figure 5: Action-State table search and update example: red circles denote actions for which the robot is responsible, while blue circles denote actions for which the human is responsible. Yellow, red and green filling colors denote, respectively, ambiguous, null and clear mode of the table search. $a_1 - a_8$ are labels of actions; $n_1 - n_4$ are labels of feasible nodes; and $w_{n_1} - w_{n_4}$ denote the weight of each state.

The same updating mechanism occurs for actions executed by the robot, as discussed in Section 3.5.

Once all the actions associated with a hyper-arc have been performed, the hyperarc is marked as *done* and its parent node is marked as *solved*. Then, the *Planner* informs the *Task Representation* module, which updates the AND/OR graph and determines the next suggested action for operators or robots.

Figure 5 illustrates the procedure. In the Figure, each row corresponds to an active hyper-arc, its parent node is defined on the left-hand side, and the set of actions associated with the hyper-arc 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 actions associated with hyper-arc h_1, that operator and robot should perform to solve node n_1.

Let us assume that, at the beginning of the cooperation, the active hyper-arcs are those shown in Figure 5 and that the hyper-arc with minimum cost is the top one.

Depending on the sequence of operator and robot actions perceived by the *Human Action Recognition* and *Controller* 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 *Planner* commands the robot to execute actions a_3 and a_4, and once the latter is completed the hyper-arc is marked as *done* and node n_1 is *solved*. Upon the completion of action a_3, FlexHRC is in *clear* mode and it is 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 active hyper-arcs. FlexHRC thus enters the *null* mode and ends the cooperation.
3. If the sequence corresponds to sequence 3, after action a_2, the *Planner* commands the robot for the execution of action a_3, along the top hyper-arc, but the operator interrupts its execution by performing action a_6. The first two hyper-arcs become inactive, and the *Planner* switches to hyper-arc h_3 (which has a lower cost than hyper-arc h_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.

3.4. Planning the Next Operator or Robot Action

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

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: $loop \leftarrow true$
2: $s(n) \leftarrow true$
3: **while** $loop = true$ **do**
4: **if** $n = n_r$ **then**
5: $loop \leftarrow false$
6: **end if**
7: **for all** $m \in N$ **do**
8: updateFeasibility(m)
9: **end for**
10: updateAllPaths($\mathcal{P}$)
11: $n^* \leftarrow$ findSuggestion($\mathcal{P}$)
12: *return*
13: **end while**
14: $s(G) \leftarrow true$
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: $P^u \leftarrow \emptyset$
2: $P^u \leftarrow$ findPathsToUpdate(n)
3: **for all** $P \in P^u$ **do**
4: $cost(P) \leftarrow$ updateCost()
5: **end for**

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

Algorithm 7 findSuggestion()

Require: The set $\mathcal{P}$ of all cooperation paths

Ensure: The next node to solve n^*

1: $P^* \leftarrow$ findOptimalPath($\mathcal{P}$)
2: **for all** $n \in P^*$ **do**
3: $\quad n^* \leftarrow$ findOptimalNode()
4: **end for**
5: *return*

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:

$$cost(P) = cost(P) - (w_n + h_n^m - w_h), \tag{1}$$

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^*.

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.

3.5. Robot Control and Action Execution

This Section describes the Task Priority framework integrated within the *Controller* module (Figure 6), and elaborates on its decoupling of action planning, as performed by the *Planner* module, and control. The framework is

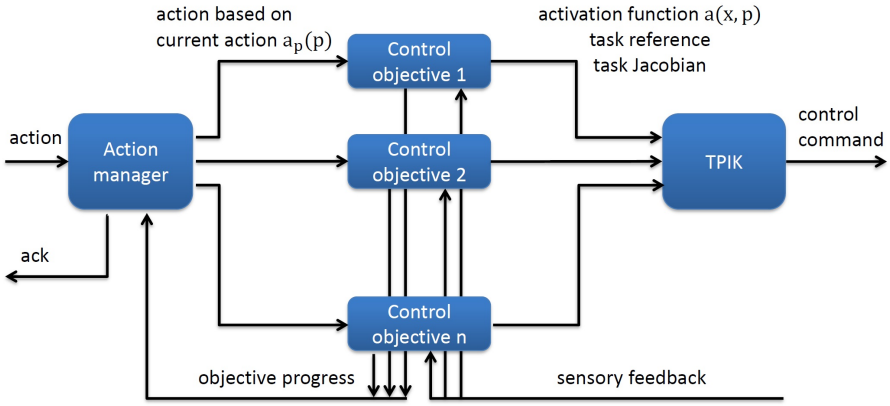


Figure 6: A sketch of the *Controller* internal structure.

general and can handle multi-arm mobile manipulators, but it can be scaled down to be used with fixed-base robots such as dual-arm manipulators. In the following paragraphs, the robot configuration vector is referred to as $\boldsymbol{c} \in \mathbb{R}^n$ and contains the robot DOFs, e.g., 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.

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.

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

$$\dot{\tilde{x}}_o(x_o) \triangleq \gamma(x_o^* - x_o),\ \gamma > 0, \tag{2}$$

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:

$$\dot{x}_o = \boldsymbol{J}_o(\boldsymbol{c})\dot{\boldsymbol{y}}, \tag{3}$$

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 $\dot{\tilde{x}}_o$.

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:

$$\alpha(x_o) = \alpha_o(x_o), \tag{4}$$

where $\alpha_o(x_o) \in [0, 1]$ is a continuous sigmoid function of a scalar objective variable x_o, whose value is zero within the validity region of the associated control objective o. When a new action to execute is received, the *Action Manager* block (shown in Figure 6) activates and deactivates the proper activation functions for each control objective.

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

- $\dot{\tilde{\boldsymbol{x}}}_k \triangleq \left[\dot{\tilde{x}}_{1,k}, \ldots, \dot{\tilde{x}}_{m,k}\right]^T$ 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 $\dot{\boldsymbol{y}}$.

- $\boldsymbol{A}_k \triangleq \mathrm{diag}(\alpha_{1,k}, \ldots, \alpha_{m,k})$ is the diagonal matrix of all the activation functions in the form of (4).

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 [18]. 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:

$$S_k \triangleq \left\{ \arg \text{R-} \min_{\dot{\tilde{\boldsymbol{y}}} \in S_{k-1}} \left\| \boldsymbol{A}_k (\dot{\tilde{\boldsymbol{x}}}_k - \boldsymbol{J}_k \dot{\tilde{\boldsymbol{y}}}) \right\|^2 \right\}, \tag{5}$$

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 are duly reported in [18] and will be omitted here for the sake of brevity.

Control actions. From the robot control standpoint, an action a in input to the *Controller* module in Figure 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, and 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, and 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 [32, 33], which handle only equality control objectives, or more recent frameworks such as [34], 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 (4) are modified as:

$$\alpha(x, \boldsymbol{p}) = \alpha_o(x)\alpha_p(\boldsymbol{p}), \tag{6}$$

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.

Whenever a robot action is successfully executed, the *Planner* is invoked to update the *Action-State* table and define the next action.

4. Experimental Evaluation

4.1. Scenario

In order to provide a quantitative and qualitative assessment of FlexHRC, the experimental setup comprises a dual-arm Baxter manipulator to perform cooperative operations and an LG G watch R (W110) smartwatch worn at the operator's right wrist to acquire inertial data.

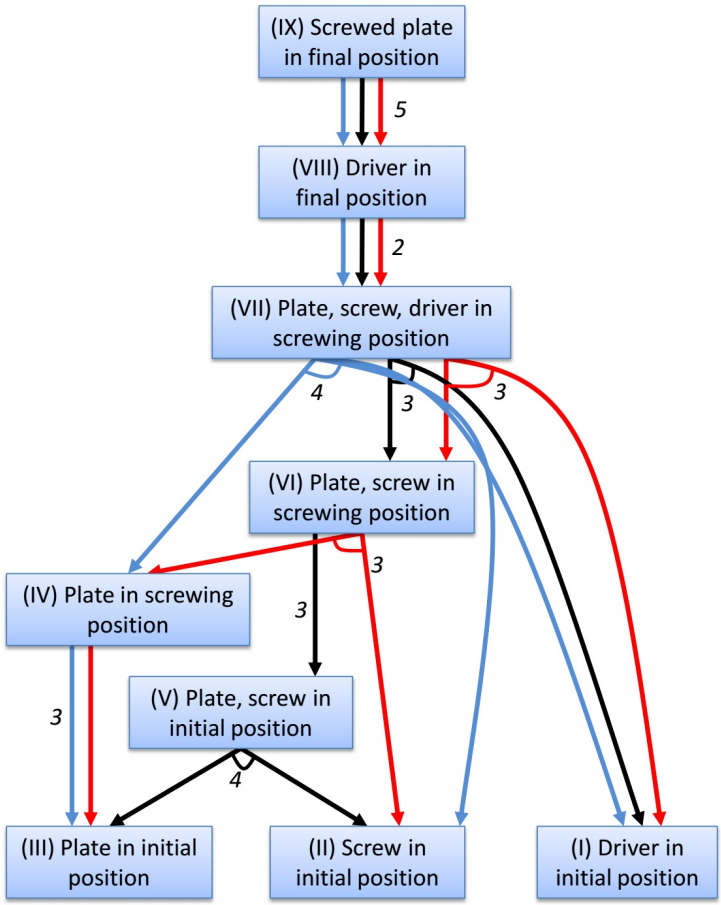


Figure 7: The AND/OR graph representation of the screwing task: different colors (*blue*, *black* and *red*) indicate different action sequences the cooperation can unfold in. Hyper-arcs costs appear beside the hyper-arcs they refer to.

Smartwatch data are collected via an LG G3 smartphone, which is paired with the smartwatch using Bluetooth and communicates with a workstation using WiFi. The smartphone acquires inertial measurements from the smartwatch at approximately 40 Hz via a Java application, and then forwards them to the workstation, which computes at best effort. The ROS Indigo based architecture 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.

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 7). In this case, cooperation models are structured in advance. It is noteworthy that current work is devoted to 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 7 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 8 shows snapshots of an actual human-robot cooperation process for the screwing task described above, which starts with M_{blue}, but switches to M_{black} when the operator performs the first action (Figures 8b and 8c), which is different from the one expected in M_{blue}. For this task, modeled human and robot actions are:

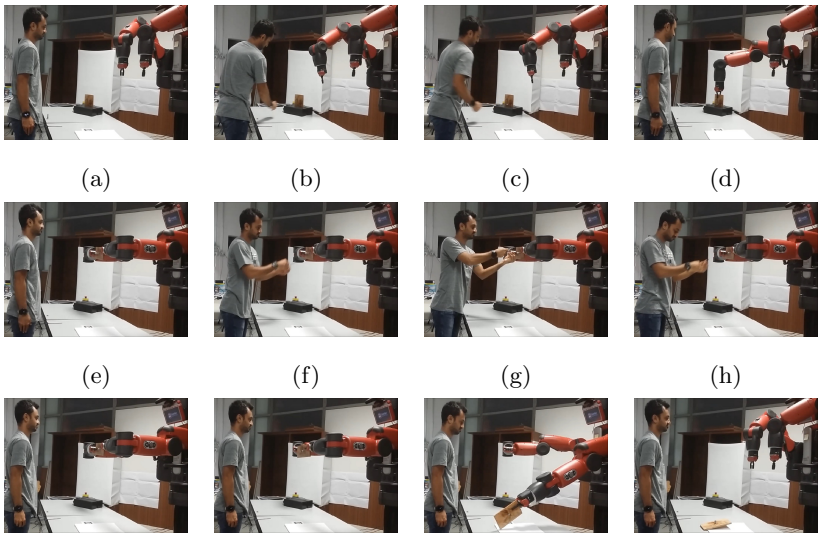


Figure 8: The sequence of actions associated with M_{black}, chosen after the operator decided not to follow M_{blue} by performing the action *initial bolt sink*.

- *initial bolt sink*: the human sinks the bolt in the wooden plate's countersink while the plate is still located on the table (Figures 8a to 8c);
- *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 8d and Figure 8e);
- *bolt or screwdriver pick up*: the operator picks up a bolt or a screwdriver from the table (Figure 8f);
- *bolt screw*: the operator sinks the bolt using the screwdriver (Figure 8g);
- *screwdriver put down*: the operator puts the screwdriver down on the table (Figure 8h and Figure 8i);
- *wooden plate put down*: the robot puts the wooden plate down in a predefined position on the table (Figure 8j and Figure 8k);
- *reset pose*: robot's pose is reset (Figure 8l).

Table 2: 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

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 *Planner* module.

4.2. 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 2 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 *Planner*'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

Table 3: 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

can be observed in the accompanying video[3]. Here, let us focus on the switch occurring between P_{blue} and P_{black} in Figures 8a to 8c. 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 8b) 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. This capability demonstrates requirement R_1 described in the Introduction.

4.3. Computational Performance

Table 2 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.

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

Table 3 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:

$$T_{ao} = \sum_{i=2}^{n} T_{next}(a_i) - T_{ack}(a_{i-1}). \tag{7}$$

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

$$\overline{T}_h = \sum_{i=1}^{m} T_{rec}(a_i) - T_{next}(a_i), \tag{8}$$

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:

$$T_h = \overline{T}_h + \sum_{i=1}^{k} T_{next}(a_{i+1}) - T_{start}(a_i). \tag{9}$$

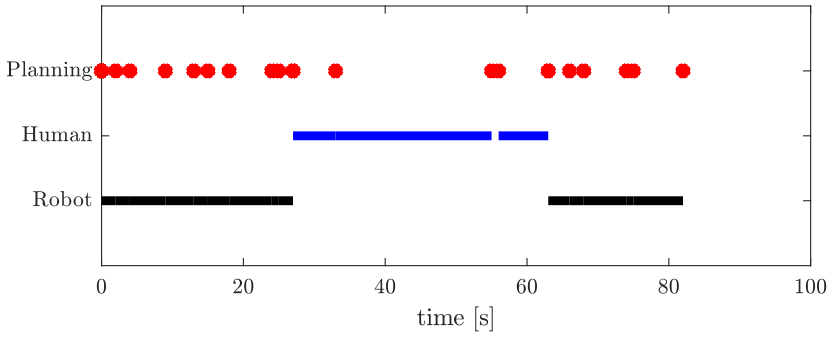


Figure 9: An example of time allocation in case of P_{blue}.

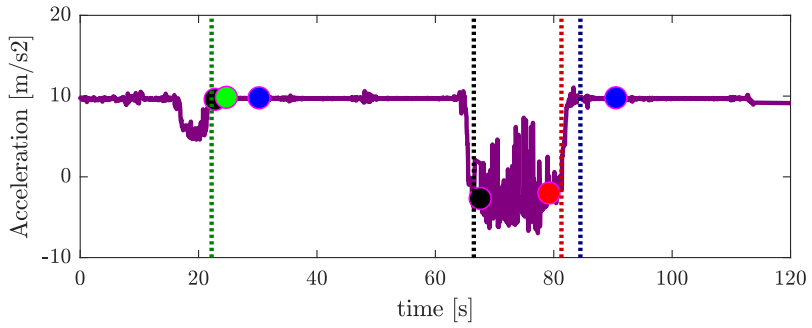
The average time percentage of AND/OR graph traversing is in all cases less than 0.1% of the total time. This is also due to the proposed control framework, which does not require a computationally intensive planning in the configuration space, thanks to its ability of reactively solving *local* obstacle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

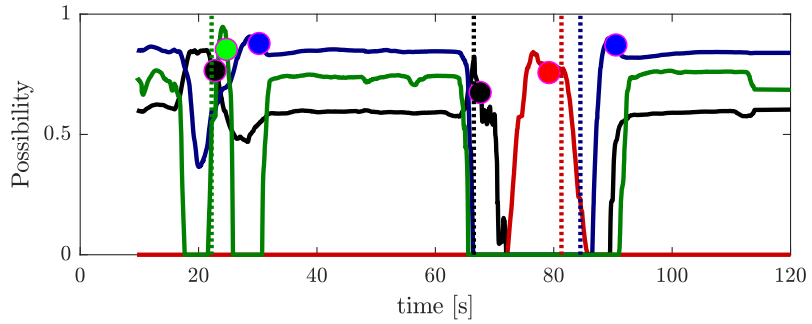
Figure 9 shows an example of time distribution for one task which follows P_{blue}. The Figure shows that the majority of the time is related to either operator or robot actions, and just a negligible part of the whole cooperation task is related to AND/OR reasoning. The total time to perform the assembly 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 *Planner* 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 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 10 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*



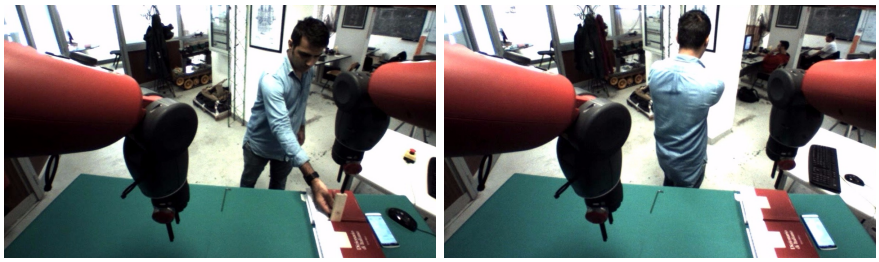
(a) Inertial data along x axis.



(b) Trend in possibilities.

Figure 10: 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.

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% of the overall cooperation time.



(a) (b)

Figure 11: 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.

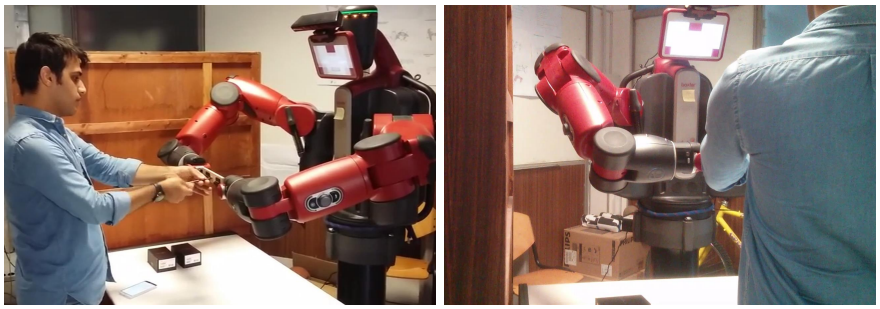
Table 4: Recap of the experiments to assess the performance of human action recognition with operators who did not contribute to the the training of the gesture models.

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

4.4. 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 4 shows the results of these experiments. In the Table, the second



(a) (b)

Figure 12: An activity part of P_{blue}, when an obstacle is detected and avoided by the robot's elbow joint.

column indicates the overall success rate, the third column the success rate of the 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 11.

4.5. Task Priority Control

In the experiments to evaluate the *Controller* module, which embeds the Task Priority control scheme described in Section 3.5, 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

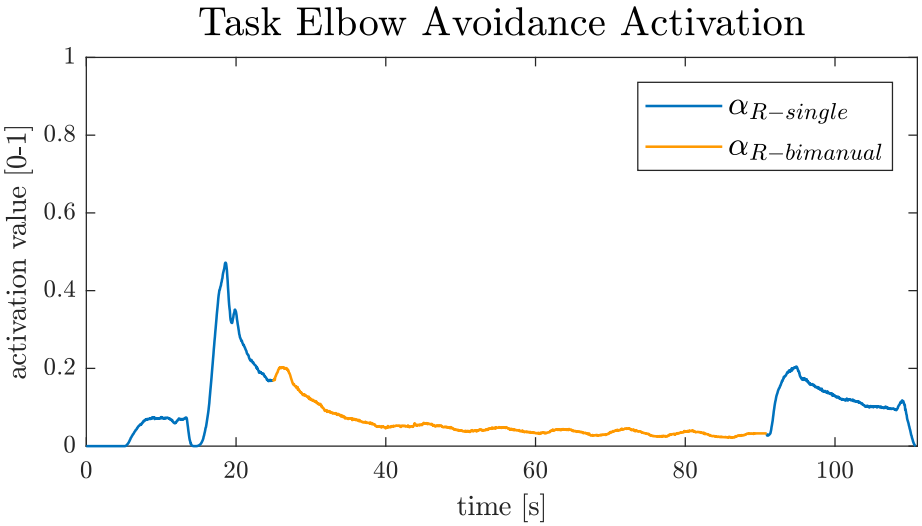


Figure 13: Activation function of the robot's elbow avoidance task for the right arm: in *blue* for a single-arm operation, in *orange* for a dual-arm operation.

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 12 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 [35, 36]), 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 13 shows the trend of the corresponding activation function (4) 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 *Planner* 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 13, 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.

5. Conclusions

In this paper, a novel architecture for human-robot cooperation is proposed, which is aimed at addressing a few challenges in shop-floor environments. FlexHRC 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.

FlexHRC 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 online learning capabilities to perform co-adaptation, as discussed in [37, 38, 39, 40].

2. 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 [41]. An on the fly assignment to the operator or the robot would increase to a great extent the flexibility of the cooperation process.
3. Safety considerations in FlexHRC are considered only to a limited extent. Different safety strategies including the detection of sudden, unwanted contacts, as well as active or adaptive safety measures are investigated extensively in the literature [42, 43]. Currently, the adoption of safety measurements in FlexHRC is limited to the possible adoption of specific tasks with high priority in the control framework.
4. 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.
5. 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 [17, 25].

References

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

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

[3] B. Hayes, B. Scassellati, 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, 2016.

[4] S. Lemaignan, M. Warnier, E. A. Sisbot, A. Clodic, R. Alami, Artificial cognition for social humanrobot interaction: An implementation, Artificial Intelligence 247 (2017) 45–69.

[5] E. Helms, R. D. Schraft, M. Hagele, 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, 2002.

[6] J. Kuehn, S. Haddadin, 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 (1) (2017) 72–79.

[7] N. Pedrocchi, F. Vicentini, M. Matteo, L. M. Tosatti, Safe human-robot cooperation in an industrial environment, International Journal of Advanced Robotic Systems 10 (1) (2013) 27.

[8] J. Shah, J. Wiken, B. Williams, C. Breazeal, 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, 2011.

[9] S. Kock, T. Vittor, B. Matthias, H. Jerregard, M. Kllman, I. Lundberg, R. Mellander, M. Hedelind, 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, 2011.

[10] L. Johannsmeier, S. Haddadin, A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes, IEEE Robotics and Automation Letters 2 (1) (2017) 41–48.

[11] D. Bortot, M. Born, K. Bengler, 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, 2013, pp. 89–90.

[12] T. Meneweger, D. Wurhofer, V. Fuchsberger, M. Tscheligi, 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, 2015.

[13] K. P. Hawkins, S. Bansal, N. N. Vo, A. F. Bobick, 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, 2014.

[14] B. I. Bertenthal, Origins and early development of perception, action, and representation, Annual Review of Psychology 47 (1) (1996) 431–459.

[15] C. Vesper, S. Butterfill, G. Knoblich, N. Sebanz, A minimal architecture for joint action, Neural Networks 23 (89) (2010) 998–1003.

[16] J. D. Loehr, N. Sebanz, G. Knoblich, Joint action: From perception-action links to shared representations, in: W. Prinz, M. Beisert, A. Herwig (Eds.), Action Science: Foundations of an Emerging Discipline, The MIT Press, Cambridge, Massachusetts, 2013, Ch. 13, p. 333.

[17] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, P. Abbeel, 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, 2014.

[18] E. Simetti, G. Casalino, A novel practical technique to integrate inequality control objectives and task transitions in priority based control, Journal of Intelligent & Robotic Systems 84 (1) (2016) 877–902.

[19] J. Krüger, T. Lien, A. Verl, Cooperation of human and machines in assembly lines, CIRP Annals in Manufacturing Technology 58 (2) (2009) 628–646.

[20] B. Bruno, F. Mastrogiovanni, A. Sgorbissa, T. Vernazza, R. Zaccaria, 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, 2013.

[21] A. C. Sanderson, M. A. Peshkin, L. S. H. de Mello, Task planning for robotic manipulation in space applications, IEEE Transactions on Aerospace and Electronic Systems 24 (5) (1988) 619–629.

[22] N. Sebanz, H. Bekkering, G. Knoblich, Joint action: bodies and minds moving together, Trends in Cognitive Sciences 10 (2) (2006) 70–76.

[23] P. Valdesolo, J. Ouyang, D. DeSteno, The rhythm of joint action: Synchrony promotes cooperative ability, Journal of Experimental Social Psychology 46 (4) (2010) 693–695.

[24] M. Huber, C. Lenz, C. Wendt, B. Frber, A. Knoll, S. Glasauer, 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, 2013.

[25] P. Agrawal, A. Nair, P. Abbeel, J. Malik, S. Levine, Learning to poke by poking: Experiential learning of intuitive physics, arXiv preprint arXiv:1606.07419.

[26] M. Toussaint, T. Munzer, Y. Mollard, L. Y. Wu, N. A. Vien, M. Lopes, Relational activity processes for modeling concurrent cooperation, in: Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016.

[27] F. Chen, K. Sekiyama, F. Cannella, T. Fukuda, Optimal subtask allocation for human and robot collaboration within hybrid assembly system, IEEE Transactions on Automation Science and Engineering 11 (4) (2014) 1065–1075.

[28] A. Bauer, D. Wollherr, M. Buss, Human-robot collaboration: a survey, International Journal of Humanoid Robotics 05 (01) (2008) 47–66.

[29] E. A. Cartmill, S. Beilock, S. Goldin-Meadow, A word in the hand: action, gesture and mental representation in humans and non-human pri-

mates, Philosophical Transactions of the Royal Society B: Biological Sciences 367 (1585) (2011) 129–143.

[30] B. Gleeson, K. MacLean, A. Haddadi, E. Croft, J. Alcazar, 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, 2013.

[31] B. Bruno, F. Mastrogiovanni, A. Saffiotti, A. Sgorbissa, 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, Springer International Publishing, Montpellier, France, 2014, pp. 596–605.

[32] B. Siciliano, J.-J. E. Slotine, 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, 1991.

[33] S. Chiaverini, Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators, IEEE Transactions on Robotics and Automation 13 (3) (1997) 398–410.

[34] S. Moe, G. Antonelli, A. R. Teel, K. Y. Pettersen, J. Schrimpf, 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 (2016) 16.

[35] L. Buoncompagni, F. Mastrogiovanni, A software architecture for object perception and semantic representation, in: Proceedings of the Second Italian Workshop on Artificial Intelligence and Robotics (AIRO), Ferrara, Italy, 2015.

[36] L. Buoncompagni, F. Mastrogiovanni, A. Saffiotti, 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, 2017.

[37] S. Nikolaidis, K. Gu, R. Ramakrishnan, J. A. Shah, Efficient model learning for human-robot collaborative tasks, arXiv preprint arXiv:1405.6341.

[38] K. Fragkiadaki, S. Levine, P. Felsen, J. Malik, Recurrent network models for human dynamics, in: Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015.

[39] M. Saveriano, S. i. An, D. Lee, 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, 2015.

[40] D. Hadfield-Menell, S. J. Russell, P. Abbeel, A. Dragan, Cooperative inverse reinforcement learning, in: Proceedings of 2016 Conference on Advances in Neural Information Processing Systems, Barcelona Spain, 2016.

[41] F. Mastrogiovanni, A. Paikan, A. Sgorbissa, Semantic-aware real-time scheduling in robotics, IEEE Transactions on Robotics 29 (1) (2013) 118–135.

[42] G. Michalos, S. Makris, P. Tsarouchi, T. Guasch, D. Kontovrakis, G. Chryssolouris, Design considerations for safe human-robot collaborative workplaces, Procedia CIRP 37 (Supplement C) (2015) 248–253.

[43] S. Makris, P. Karagiannis, S. Koukas, A.-S. Matthaiakis, Augmented reality system for operator support in human-robot collaborative assembly, CIRP Annals - Manufacturing Technology 65 (1) (2016) 61–64.