University of Cassino and Southern Lazio

Department of Electrical and Information Engineering

Doctoral Course in Methods, Models and Technologies for Engineering
Cycle XXXVIII

Control and Scheduling Frameworks for Multi-Human–Multi-Robot Collaboration

---

SSD: ING-INF/04

Supervisor:
Prof. Alessandro Marino
Prof. Paolo Augusto Di Lillo

Author:
Jozsef Palmieri

Coordinator:
Prof. Fabrizio Marignetti

UNIVERSITÀ DEGLI STUDI DI CASSINO E DEL LAZIO MERIDIONALE
CORSO DI DOTTORATO IN METODI, MODELLI E TECNOLOGIE PER L'INGEGNERIA

Author: Jozsef Palmieri
Title: Control and Scheduling Frameworks for Multi-Human–Multi-Robot Collaboration
Department: Department of Electrical and Information Engineering
Degree: PHILOSOPHIAE DOCTOR

"*To my family, thank you for your patience, understanding, and unwavering confidence in my ability to successfully complete this journey.*"

Abstract

Recent technological advances and the increasing demand for production flexibility have driven the widespread adoption of complex automated systems, including redundant and mobile manipulators, in both industrial and everyday life scenarios. These systems are increasingly deployed in unstructured and dynamic environments, where they must operate alongside humans and address complex manipulation and coordination challenges. In this context, ensuring safe, efficient, and minimally invasive human–robot coexistence remains a critical issue, particularly given the limitations of traditional safety mechanisms. This thesis investigates key aspects of human–robot collaboration and interaction in real-world environments, ranging from advanced control methodologies for redundant robotic platforms to task allocation and scheduling strategies for heterogeneous multi-agent systems composed of humans and robots. The proposed contributions include control frameworks that integrate hierarchical optimization and safety constraints, human-safety-oriented control strategies, adaptive shared autonomy approaches, and human-in-the-loop scheduling methods that account for human workload, preferences, and behavioral variability. Furthermore, a hybrid framework combining Large Language Models and constraint-based optimization is introduced to bridge high-level natural language instructions with feasible and optimized multi-agent execution plans. Overall, this work aims to balance task effectiveness and human safety while leveraging the complementary capabilities of human and robotic agents to enhance collaboration, efficiency, and system robustness.

Contents

List of Abbreviations

DoFs	**D**eegres **o**f **F**reedom
HRI	**H**uman–**R**obot **I**nteraction
pHRI	**p**hysical **H**uman–**R**obot **I**nteraction
MAS	**M**ulti-**A**gent **S**ystems
HQP	**H**ierarchical **Q**uadratic **P**rogramming
CBFs	**C**ontrol **B**arrier **F**unctions
IK	**I**nverse **K**inematics
CLIK	**C**losed-**L**oop **I**nverse **K**inematic
QP	**Q**uadratic **P**rogramming
COP	**C**onstraint **O**ptimization **P**roblem
LP	**L**inear **P**rogramming
ILP	**I**nteger **L**inear **P**rogramming
MILP	**M**ixed-**I**nteger **L**inear **P**rogramming
CP	**C**onstraint **P**rogramming
CSP	**C**onstraint **S**atisfaction **P**roblem
LLM	**L**arge **L**anguage **M**odel
AI	**A**rtificial **I**ntelligence
CLF	**C**ontrol **L**yapunov **F**unction
NUC	**N**ext **U**nit of **C**omputing
IMU	**I**nertial **M**easurement **U**nit
GPS	**G**lobal **P**ositioning **S**ystem
RTK	**R**eal **T**ime **K**inematic
SSM	**S**peed and **S**eparation **M**onitoring
F/T	**F**orce/**T**orque
PDDL	**P**lanning **D**omain **D**efinition **L**anguage

List of Symbols

$\mathbb{R}$	set of real numbers
$\mathbb{Z}$	set of integers
$\boldsymbol{q}$	joint position (configuration) vector
$\boldsymbol{q}_d$	desired joint position (configuration)
$\dot{\boldsymbol{q}}$	joint velocity vector
$\dot{\boldsymbol{q}}_d$	desired joint velocity
$\boldsymbol{\sigma}_x$	task function associated with task x
$\boldsymbol{\sigma}_{x,d}$	desired value of task x
$\tilde{\boldsymbol{\sigma}}_x$	task x error
$\dot{\boldsymbol{\sigma}}_x$	task x time derivative
$\dot{\boldsymbol{\sigma}}_{x,d}$	desired task x time derivative
$\ddot{\boldsymbol{\sigma}}_x$	task x second time derivative
$\ddot{\boldsymbol{\sigma}}_{x,d}$	desired task x second time derivative
$\boldsymbol{J}_x$	Jacobian matrix associated with task x
t_k	time instant at step k
Δt	integration time step
$\boldsymbol{J}_x^{-1}$	inverse of the task x Jacobian matrix
$\boldsymbol{K}_x$	positive-definite gain matrix associated with task x
$\boldsymbol{J}_x^{\dagger}$	Moore-Penrose pseudoinverse of $\boldsymbol{J}_x$
$\boldsymbol{J}_x^{\star}$	complex-conjugate of $\boldsymbol{J}_x$
$\mathcal{R}(\cdot)$	range space
$\mathcal{N}(\cdot)$	null-space
$\boldsymbol{N}_x$	null-space projector of $\boldsymbol{J}_x$
$\boldsymbol{I}$	identity matrix
$\boldsymbol{I}_n$	identity matrix of size $(n \times n)$
k_0	scalar gain
$\boldsymbol{J}^A$	augmented Jacobian matrix
$\boldsymbol{N}^A$	null-space projector of $\boldsymbol{J}^A$
$\boldsymbol{\xi}$	system state vector

$\boldsymbol{u}$	control input vector
$\boldsymbol{u}_{(\cdot)}$	nominal input vector
h_k	control barrier function
$\boldsymbol{h}_x$	control barrier functions vector of task x
$\boldsymbol{w}_x$	slack variables vector of task x
$\boldsymbol{b}_x$	barrier terms vector of task x
$\boldsymbol{Q}_x$	positive-definite weight matrix for decision variables vector $\boldsymbol{q}_x$
$\boldsymbol{Q}_{\boldsymbol{w}_x}$	positive-definite weight matrix for slack variables vector $\boldsymbol{w}_x$
$\boldsymbol{w}_x^\star$	optimal solution of the task x QP problem
$\boldsymbol{q}_b$	joint position (configuration) vector of the mobile base
$\dot{\boldsymbol{q}}_b$	joint velocity vector of the mobile base
$\boldsymbol{q}_t$	joint position (configuration) vector of the torso
$\dot{\boldsymbol{q}}_t$	joint velocity vector of the torso
$\boldsymbol{q}_l$	joint position (configuration) vector of the left arm
$\dot{\boldsymbol{q}}_l$	joint velocity vector of the left arm
$\boldsymbol{q}_r$	joint position (configuration) vector of the right arm
$\dot{\boldsymbol{q}}_r$	joint velocity vector of the right arm
n_b	number of DoFs of the mobile base
n_t	number of DoFs of the torso
n_a	number of DoFs of the arm
n	number of DoFs of the entire system
$i \in \{l, r\}$	index identifying left (l) and right (r) arms
SE(3)	Special Euclidean group of rigid-body transformations in $3D$
$\boldsymbol{x}_i$	Cartesian configuration vector of the end-effector i
$\boldsymbol{x}$	Cartesian configuration vector of the dual-arm system
$\boldsymbol{p}_i$	position vector of the end-effector i
$\boldsymbol{R}_i$	rotation (orientation) matrix of the end-effector i
$\boldsymbol{o}_i$	unit quaternion of the end-effector i
κ_{o_i}	scalar part of the unit quaternion $\boldsymbol{o}_i$
$\boldsymbol{\varrho}_{o_i}$	vector part of the unit quaternion $\boldsymbol{o}_i$
$\boldsymbol{v}_i$	generalized (Cartesian) velocity vector of the end-effector i
$\boldsymbol{v}$	generalized (Cartesian) velocity vector of the dual-arm system
$\dot{\boldsymbol{p}}_i$	linear velocity vector of the end-effector i

$\boldsymbol{\omega}_i$	angular velocity vector of the end-effector i
$\mathbf{0}_x$	zero vector of size x
$\boldsymbol{O}_{x\times y}$	zero matrix of size $(x \times y)$
$\boldsymbol{J}_i$	Jacobian matrix of the end-effector i
$\boldsymbol{J}_\gamma$	Jacobian matrix of the dual-arm system
$\boldsymbol{J}_{b,i}$	Jacobian matrix relating $\dot{\boldsymbol{q}}_b$ to $\boldsymbol{v}_i$
$\boldsymbol{J}_{t,i}$	Jacobian matrix relating $\dot{\boldsymbol{q}}_t$ to $\boldsymbol{v}_i$
$\boldsymbol{J}_{a,i}$	Jacobian matrix relating $\dot{\boldsymbol{q}}_i$ to $\boldsymbol{v}_i$
$\underline{h}_{jp,i}$	lower position limit barrier function of joint i
$\bar{h}_{jp,i}$	upper position limit barrier function of joint i
$\underline{\phi}_{jp,i}$	positive gain of barrier function $\underline{h}_{jp,i}$
$\bar{\phi}_{jp,i}$	positive gain of barrier function $\bar{h}_{jp,i}$
$\underline{\boldsymbol{h}}_{jp}$	lower joint position limit barrier function vector
$\bar{\boldsymbol{h}}_{jp}$	upper joint position limit barrier function vector
$\underline{\boldsymbol{\Phi}}_{jp}$	diagonal gain matrix of barrier function vector $\underline{\boldsymbol{h}}_{jp}$
$\bar{\boldsymbol{\Phi}}_{jp}$	diagonal gain matrix of barrier function vector $\bar{\boldsymbol{h}}_{jp}$
$\boldsymbol{J}_{jp}$	Jacobian matrix of the joint position constraints
$\underline{\boldsymbol{b}}_{jp}$	lower bound vector of the joint position constraints
$\bar{\boldsymbol{b}}_{jp}$	upper bound vector of the joint position constraints
$\boldsymbol{J}_{jv}$	Jacobian matrix of the joint velocity constraints
$\underline{\boldsymbol{b}}_{jv}$	lower bound vector of the joint velocity constraints
$\bar{\boldsymbol{b}}_{jp}$	upper bound vector of the joint velocity constraints
$\boldsymbol{p}_j$	position vector of generic point j on the robot structure
$\boldsymbol{p}^k$	position vector of an arbitrary point k on the virtual plane
$\hat{\boldsymbol{n}}$	unit normal vector
$\sigma_{\text{vw,j}}$	virtual-wall task function for point $\boldsymbol{p}_j$
$\boldsymbol{J}_{p,j}$	position Jacobian matrix for point $\boldsymbol{p}_j$
$\boldsymbol{J}_{\text{vw},j}$	Jacobian matrix of the virtual-wall task for point $\boldsymbol{p}_j$
$\underline{\sigma}_{\text{vw,j}}$	minimum admissible distance between $\boldsymbol{p}_j$ and the virtual wall
$\underline{h}_{\text{vw},j}$	virtual-wall task barrier function for point $\boldsymbol{p}_j$
$\underline{\phi}_{\text{vw},j}$	positive gain of barrier function $\underline{h}_{\text{vw},j}$
$\underline{\boldsymbol{h}}_{\text{vw}}$	virtual-wall task barrier function vector
$\boldsymbol{J}_{\text{vw}}$	overall Jacobian matrix of the virtual-wall task
$\boldsymbol{\Phi}_{\text{vw}}$	diagonal gain matrix of barrier function vector $\underline{\boldsymbol{h}}_{\text{vw}}$
$\sigma_{\text{sc},j,l}$	self-collision task function for points $\boldsymbol{p}_j$ and $\boldsymbol{p}_l$

$\boldsymbol{J}_{\mathrm{sc},j,l}$	self-collision task Jacobian matrix associated with points $\boldsymbol{p}_j$ and $\boldsymbol{p}_l$
$\hat{\boldsymbol{n}}_{j,l}$	unit normal vector connecting $\boldsymbol{p}_j$ to $\boldsymbol{p}_l$
$\underline{\sigma}_{\mathrm{sc},j,l}$	minimum admissible distance between $\boldsymbol{p}_j$ and $\boldsymbol{p}_l$
$\underline{h}_{\mathrm{sc},j,l}$	self-collision task barrier function for points $\boldsymbol{p}_j$ and $\boldsymbol{p}_l$
$\underline{\phi}_{\mathrm{sc},j,l}$	positive gain of barrier function $\underline{h}_{\mathrm{sc},j,l}$
$\boldsymbol{J}_{\mathrm{sc},j,l}$	self-collision task Jacobian matrix for points $\boldsymbol{p}_j$ and $\boldsymbol{p}_l$
$\boldsymbol{J}_{\mathrm{sc},j}$	overall self-collision task Jacobian matrix for point $\boldsymbol{p}_j$
$\underline{\boldsymbol{h}}_{\mathrm{sc},j}$	overall self-collision task barrier function vector for point $\boldsymbol{p}_j$
$\underline{\boldsymbol{\Phi}}_{\mathrm{sc},j}$	diagonal gain matrix of the barrier function vector $\underline{\boldsymbol{h}}_{\mathrm{sc},j}$
$\boldsymbol{J}_{\mathrm{sc}}$	overall self-collision task Jacobian matrix
$\underline{\boldsymbol{h}}_{\mathrm{sc}}$	overall self-collision task barrier function vector
$\underline{\boldsymbol{\Phi}}_{\mathrm{vw}}$	diagonal gain matrix of the barrier function vector $\underline{\boldsymbol{h}}_{\mathrm{sc}}$
$\underline{\boldsymbol{b}}_{\mathrm{sc}}$	lower bound vector of the self-collision constraints
$\boldsymbol{\sigma}_{\gamma}$	uncooperative motion task function
$\boldsymbol{J}_{\gamma}$	Jacobian matrix of the uncooperative motion task
$\boldsymbol{\sigma}_{a,r}$	cooperative motion task function
$\dot{\boldsymbol{\sigma}}_{a,r}$	time derivative of the cooperative motion task function
$\boldsymbol{J}_{a,r}$	Jacobian matrix of the cooperative motion task
$\boldsymbol{p}_a$	position vector of the absolute frame
$\boldsymbol{p}_r$	position vector of the relative frame
$\boldsymbol{o}_a$	unit quaternion of the absolute frame
$\boldsymbol{o}_r$	unit quaternion of the relative frame
$\boldsymbol{R}_a$	rotation matrix of the absolute frame
$\boldsymbol{R}_r$	rotation matrix of the relative frame
$\boldsymbol{R}^l_{l,r}$	rotation matrix expressing the orientation of the right end-effector frame w.r.t. the left one in the left end-effector frame
$(\boldsymbol{r}^l_{l,r}, \theta_{l,r})$	axis–angle representation of the rotation matrix $\boldsymbol{R}^l_{l,r}$
$(\boldsymbol{r}_l, \theta_l)$	axis–angle representation of the rotation matrix $\boldsymbol{R}_l$
$\boldsymbol{\sigma}_a$	pose vector of the absolute frame
$\boldsymbol{\sigma}_{a_p}$	position component of $\boldsymbol{\sigma}_a$
$\boldsymbol{\sigma}_{a_o}$	orientation component of $\boldsymbol{\sigma}_a$
$\boldsymbol{\sigma}_r$	pose vector of the relative frame
$\boldsymbol{\sigma}_{r_p}$	position component of $\boldsymbol{\sigma}_r$

$\boldsymbol{\sigma}_{r_o}$	orientation component of $\boldsymbol{\sigma}_r$
$\boldsymbol{J}_\sigma$	Jacobian matrix of the operational task
$\boldsymbol{b}_\sigma$	right-hand-side term of the operational task constraints
$\boldsymbol{p}_{i,e}$	position vector of the elbow of arm i
$\boldsymbol{p}_{i,w}$	position vector of the wrist of arm i
$\boldsymbol{p}_{i,s}$	position vector of the shoulder of arm i
$\mathrm{sgn}(\cdot)$	sign function
φ_i	swivel-angle of arm i
$\varphi_{i,d}$	desired swivel-angle for arm i
$\boldsymbol{J}_{\mathrm{sw}}$	Jacobian matrix of the swivel-angle task
$\boldsymbol{b}_{\mathrm{sw}}$	right-hand-side term of the swivel-angle task constraints
$\boldsymbol{R}(\phi(t), \theta(t), \psi(t))$	rotation matrix derived from roll-pitch-yaw angles
$n_\hbar$	number of relevant points in the human skeleton
$\boldsymbol{p}_\hbar$	position vector of point $P_\hbar$ on the human operator
d	Euclidean distance between two arbitrary points
$f(\boldsymbol{p}, \boldsymbol{p}_\hbar)$	local safety index (field) between a robot point $\boldsymbol{p}$ and a human point $\boldsymbol{p}_\hbar$
$\boldsymbol{p}_\ell^0$	position vector of the starting point of robot link ℓ
$\boldsymbol{p}_\ell^1$	position vector of the ending point of robot link ℓ
$s \in [0, 1]$	curvilinear abscissa
$\boldsymbol{p}_\ell^s$	position vector of a point on robot link ℓ at abscissa s
$F_\ell\left(\boldsymbol{p}_\ell^0, \boldsymbol{p}_\ell^1, \boldsymbol{p}_\hbar\right)$	safety field between the robot link ℓ and a human point $\boldsymbol{p}_\hbar$
$F^j\left(\boldsymbol{q}, \boldsymbol{p}_{\hbar,j}\right)$	safety field between the robot and human point $\boldsymbol{p}_{\hbar,j}$
F	overall safety field between the robot and all $n_\hbar$ human points
$\underline{F}$	safety threshold
$\dot{f}$	derivative of f
$d_{\ell,\hbar}^s$	Euclidean distance between $\boldsymbol{p}_\ell^s$ and $\boldsymbol{p}_\hbar$
$\dot{d}_{\ell,\hbar}^s$	time derivative of $d_{\ell,\hbar}^s$
$\boldsymbol{J}_\ell^s$	position Jacobian matrix of point $\boldsymbol{p}_\ell^s$
$\dot{\boldsymbol{p}}_\ell^s$	generalized velocity vector of point $\boldsymbol{p}_\ell^s$
t_0	initial time
t_f	final time
$c(t)$	time-scaling parameter
$\dot{c}(t)$	time derivative of $c(t)$

$\boldsymbol{\sigma}_d(t)$	nominal trajectory
$\boldsymbol{\nu}_d(t)$	time derivative of $\boldsymbol{\sigma}_d(t)$
$\boldsymbol{\sigma}_d(c(t))$	time-scaled nominal trajectory
$\boldsymbol{\sigma}_s(t)$	safe trajectory
$\boldsymbol{\nu}_s(t)$	time derivative of $\boldsymbol{\sigma}_s(t)$
$\Delta\boldsymbol{\sigma}$	vector of spatial deviations from the nominal path
$\Delta\boldsymbol{\nu}$	deviation velocities vector
$\Delta\boldsymbol{\epsilon}$	vector of position and orientation deviations
$\boldsymbol{T}$	diagonal gain matrix of vector $\Delta\boldsymbol{\epsilon}$
$\boldsymbol{\zeta}$	diagonal gain matrix of vector $\Delta\boldsymbol{\epsilon}$
$\boldsymbol{\Psi}$	matrix defining admissible deviation directions
$\boldsymbol{u}_\alpha$	vector modulating deviation from the nominal path

Chapter 1

Introduction

1.1 Motivations

Recent advances in technology, together with the growing demand for production flexibility, have fostered the adoption of complex automated systems, including redundant manipulators, across a wide range of scenarios [1]. These technological progresses have also enhanced the systems' capabilities for contextual awareness and reasoning, facilitating their deployment not only in traditional industrial domains but also in everyday life settings. Industrial applications include logistics and manufacturing, where automated systems handle picking operations for diverse objects [2], ground structure assembly [3], and the management of high-altitude platforms [4]. In everyday life, applications span from factories [5] and operating rooms [6] to agricultural fields [7, 8] and domestic environments [9], where they contribute to task execution and improve the overall quality of life. Unlike traditional industrial manufacturing scenarios, where robots are typically used to operate in structured environments performing repetitive tasks, recent years have seen their increasing deployment in unstructured environments to address complex manipulation problems. Representative examples include the maintenance and operation of Oil & Gas underwater structures [10], underground search and rescue missions [11], and applications involving mobile manipulators operating over large workspaces. A remarkable example is provided by mobile manipulators, which integrate the mobility of a robotic platform with the dexterous manipulation capabilities of an articulated arm. This combination enables them to navigate and interact with complex and dynamic environments that can not be fully modeled in advance. Nevertheless,

the employment of such systems introduces several challenges, including the management of multiple control-level constraints. A representative example involves avoiding collisions between robot components, such as those that may occur between the arms or between the arms and the mobile base, as well as with the surrounding environment. Addressing this issue entails the development of control algorithms, based on a comprehensive model of the system, that are capable of ensuring the effective coordination of the components motion. Beyond the increased number of degrees of freedom (DoFs), the overall complexity of such systems is frequently augmented by sophisticated sensor suites mounted on them. For example, in contexts involving human–robot interaction, these systems incorporate dedicated hardware and software modules designed to support and enhance the implementation of physical human–robot interaction (pHRI) paradigms. [12]. It is worth emphasizing that the increasing implementation of such systems has been driven not only by recent technological advancements but also by the acknowledgment that humans and robots exhibit distinct yet complementary abilities that can be leveraged to enhance task performance and optimize productivity [13], as well as by the demonstrated potential of human–robot collaboration and interaction across diverse application domains [14, 15, 16]. This collaboration can occur in different forms, such as: *i*) sharing of the workspace, with humans and robots working in parallel on different tasks; or *ii*) engaging in collaborative tasks, with humans providing their cognitive skills while intentionally exchanging forces with robots.

However, as robots become more integrated into society, an important question arises: how can humans coexist in a workspace shared with robots in a way that is safe and minimally invasive for everyone involved? Regarding safety, current collaborative robots are typically equipped with several safety features, e.g., lightweight or emergency stop procedures. Despite these features, ensuring that robots do not pose any risk to human workers within their workspace remains challenging, particularly in unstructured and dynamic settings, where such features may not be sufficient to ensure safe coexistence. Thus, to foster the adoption of such systems and fully exploit their capabilities, additional safety strategies must be designed, in particular to enable reliable assessment of human safety level under all possible operating conditions involving both humans and robots, and to enforce appropriate interventions when safety is

at risk. However, at the same time, it is essential to recognize the value of allowing robots to successfully carry out the assigned tasks while adhering to their constraints. This implies that an appropriate balance must be established between preserving the execution of robotic activities and safeguarding human safety, as an excessively conservative approach may lead to intrusive and unnecessary interventions that could undermine the effectiveness of the robots' missions.

The analysis of existing paradigms for single-human–single-robot collaboration and interaction, along with the identification of strategies for improving their efficiency, constitutes an initial and essential step toward the deployment of multi-agent systems (MAS). The integration of such systems, particularly those involving heterogeneous agents such as humans and robots, into everyday human environments aims to leverage the complementary capabilities of these agents, namely the cognitive abilities of humans and the physical operational capacities of robots. The primary objective of this approach is to exploit such complementarities to complete complex missions and/or achieve additional objectives, such as improving task efficiency, optimizing quantitative metrics like the makespan (i.e., total mission completion time) and human workload, and enhancing human comfort and preferences. Multi-agent scenarios are typically characterized by a finite set of tasks that must be accomplished. Thus, to effectively leverage the aforementioned complementarities, it is essential to devise robust strategies for task distribution and scheduling across human and robotic agents [17]. The optimization of human-specific aspects is crucial to ensure socially acceptable and effective interactions [18, 19, 20], aspects that are central to fulfilling the objectives outlined above. Therefore, the devised strategies should also integrate descriptive models of human behavior, which are typically shaped by factors such as fatigue, attentional fluctuations, and changing task requirements [21].

1.2 Thesis Outline and Contribution

The challenges outlined in Section 1.1 shaped the structural organization of this study, which systematically investigates the key aspects of human–robot collaboration and interaction in real-world environments. The analysis evolves from the examination of advanced control methodologies for redundant mobile manipulators to the development of efficient

management strategies for multi-agent systems that integrate human and robotic agents. In particular, this thesis is divided into 6 chapters:

- **Chapter 2 - Mathematical Background**: this chapter aims to introduce the theoretical background and techniques required for the comprehension of the topics discussed in the subsequent chapters of this thesis. Specifically, it provides a concise yet comprehensive overview of: *i*) the main aspects of kinematic control of robotic systems; *ii*) techniques commonly adopted to address multiple tasks and constraints characterized by different priority levels in the control of redundant robotic systems; and *iii*) optimization-based techniques commonly utilized either for activity allocation or scheduling.
- **Chapter 3 - A Unified HQP–CBF Framework for Hierarchical Control of Redundant Robotic Platforms**: this chapter focuses on the control of robotic platforms characterized by complex kinematic structures when operating in dynamic and unstructured environments, such as those encountered in agricultural domains. In these scenarios, several tasks are required to be performed concurrently, including the management of kinematic constraints, the achievement of operational objectives, and the processing of external inputs. To address these challenges, these systems must rely on a flexible and effective architecture that integrates perception with control. The chapter, therefore, introduces a comprehensive framework that combines Hierarchical Quadratic Programming (HQP) and Control Barrier Functions (CBFs) to manage both equality and inequality constraints across different priority levels.
- **Chapter 4 - A Human-Safety-Oriented Control Framework for Redundant Robotic Systems**: this chapter addresses the challenge of guaranteeing human safety in dynamic and unstructured environments where robots and humans operate side by side. It introduces a control architecture designed to facilitate the effective operation of complex robotic systems, such as dual-arm mobile robots, under such conditions. The proposed architecture endows these systems with the capability to dynamically adjust their trajectory either by reducing their velocity or deviating from a nominal path, thus ensuring the safeguard of human operators while fulfilling

the assigned operational objectives.

- **Chapter 5 - Adaptive Shared Control Framework for Enhancing Human–Robot Collaboration**: this chapter introduces a shared control framework, based on the architecture presented in Chapter 3, for complex kinematic systems operating in real-world scenarios, such as dual-arm mobile robots engaged in harvesting procedures. The proposed strategy seeks to facilitate task execution by dynamically adjusting the level of autonomy of the system.
- **Chapter 6 - A Human-in-the-Loop Scheduling and Task Allocation Framework for Multi-Human–Multi-Robot Collaboration**: this chapter addresses methodologies for coordinating heterogeneous teams composed of humans and robots. The proposed framework, introduced by considering a scenario in which mobile service robots perform assistance activities for human and robotic working agents, handles the allocation and scheduling of tasks while accounting for three key aspects: *i*) the different characteristics of the agents, *ii*) their inherent variability due to changing environmental conditions and dynamic human behavior, captured through chance-constrained programming, and *iii*) human preferences and feedback provided in real time.
- **Chapter 7 - From Natural Language to Optimal Plans: A Hybrid LLM and Constraint Programming Approach for Multi-Agent Systems**: this chapter presents a hybrid framework for task planning, allocation, and scheduling in heterogeneous multi-agent systems. The proposed two-layer architecture is designed to bridge the gap between high-level human instructions and optimized execution plans. It achieves this by synergistically combining *i*) the semantic reasoning and flexibility of Large Language Models (LLMs) for interpreting natural language commands and generating a high-level plan structure, with *ii*) the mathematical rigor of Constraint Programming (CP) for ensuring that the final task allocation and schedule is feasible, optimal, and compliant with all operational constraints.

The activities described in this thesis led to the publication of the following papers:

- J. Gallou, M. Lippi, J. Palmieri, A. Gasparri, A. Marino, 2025, "A human-centered task allocation and scheduling framework for multi-human-multi-robot collaboration in precision agriculture settings", IEEE Transactions on Automation Science and Engineering, IEEE, USA, DOI: 10.1109/TASE.2025.3595413.
- J. Palmieri, P. Di Lillo, S. Chiaverini, A. Marino, 2024, "A Hierarchical Quadratic Programming control architecture based on Control Barrier Functions including admittance constraints for velocity-controlled dual-arm systems", Control Engineering Practice, Elsevier, Netherlands, DOI: 10.1016/j.conengprac.2025.106394.
- J. Palmieri, P. Di Lillo, M. Lippi, S. Chiaverini, A. Marino, 2024, "A Control Architecture for Safe Trajectory Generation in Human-Robot Collaborative Settings", IEEE Transactions on Automation Science and Engineering, IEEE, USA, DOI: 10.1109/TASE.2024.3350976.
- J. Palmieri, M. Lippi, A. Marino, 2025. "LLM-Enhanced Constraint Programming for Task Planning in Heterogeneous Multi-Agent Systems". In 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025) - Workshop Multi-Agent Cooperative Systems and Swarm Robotics in the Era of Generative AI, Hangzhou, 19/10/2025 - 25/10/2025
- J. Palmieri, P. Di Lillo, S. Chiaverini, A. Marino, 2025. "A QP-based control framework for safe human-robot collaboration in real-world field environments". In 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025) - Workshop Building Safe Robots, Hangzhou, 19/10/2025 - 25/10/2025
- J. Palmieri, P. Di Lillo, S. Chiaverini, A. Marino, 2025. "A Shared Control Approach for Semi-Autonomous Agricultural Harvesting Robots". In 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025) - Workshop On Agricultural Robotics: Advances in Design, Perception and Control to Deal With the Complex Agricultural Environment, Hangzhou, 19/10/2025 - 25/10/2025
- J. Palmieri, P. Di Lillo, A. Marino, 2025. "Shared Control and Task Planning for Semi-Autonomous Robots in Agricultural Harvesting

Tasks". In Automatica.it, Società Italiana Docenti e Ricercatori in Automatica (SIDRA), Perugia, 03/09/2025 - 05/09/2025

- J. Palmieri, P. Di Lillo, A. Marino, 2025. "A human-centered task allocation and scheduling framework for multi-human-multi-robot collaboration in precision agriculture settings". In 18th International Workshop on Human-Friendly Robotics (HFR 2025), Capri, 15/06/2025 - 16/06/2025
- J. Palmieri, P. Di Lillo, A. Marino, 2024. "A control architecture for semi-autonomous robots in agriculture settings". In 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2024) - Workshop on Agricultural Robotics for a Sustainable Future, Abu Dhabi, 14/10/2024 - 18/10/2024
- J. Palmieri, P. Di Lillo, A. Sanfeliu, A. Marino, 2024. "Perception-Driven Shared Control Architecture for Agricultural Robots Performing Harvesting Tasks". In 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2024), Abu Dhabi, 14/10/2024 - 18/10/2024
- L. Ferrini, J. Palmieri, A. Marino, D. Lee, S. Lemaignan, 2024. "reMap: Spatially-Grounded and Queryable Semantics for Interactive Robots". In 17th International Workshop on Human-Friendly Robotics (HFR 2024), Lugano, 30/09/2024 - 01/10/2024
- L. Ferrini, J. Palmieri, A. Marino, D. Lee, S. Lemaignan, 2024. "reMap: Spatially-Grounded and Queryable Semantics for Interactive Robots". In 21th International Conference on Software and Systems Reuse (ICSR 2024), Odense, 23/10/2024 - 26/10/2024
- J. Palmieri, P. Di Lillo, A. Marino, 2024. "A Simple and Effective Human-Robot Shared Control Approach for Agricultural Robots Employed in Table-Grape Vineyards". In 32nd Mediterranean Conference on Control and Automation (MED 2024), Creta, 11/06/2024 - 14/06/2024
- M. Lippi, J. Gallou, J. Palmieri, A. Gasparri, A. Marino, 2023. "Human-Multi-Robot Task Allocation in Agricultural Settings: A

Mixed Integer Linear Programming Approach". In 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN 2023), Busan, 28/08/2023 - 31/08/2023

- J. Palmieri, 2023. "A safety planner based on trajectory scaling and path deviation for human-robot interaction". In 40th IEEE International Conference on Robotics and Automation (ICRA 2023) - 3rd Annual Workshop on Robot Teammates in Dynamic Unstructured Environments (RT-DUNE), Londra, 29/05/2023 - 02/06/2023

Chapter 2

Mathematical Background

2.1 Inverse Kinematic Problem

In a generic robotic system, the state is mathematically represented by the joint vector $\boldsymbol{q} = [q_1, q_2, \ldots, q_n]^T \in \mathbb{R}^n$, where n denotes the number of Degrees of Freedom (DoFs). In this context, a generic task is typically defined as an m-dimensional function of the system state, $\boldsymbol{\sigma}_x(\boldsymbol{q}) \in \mathbb{R}^m$. The inverse kinematics (IK) problem, therefore, consists in finding the configuration vector $\boldsymbol{q}$ that yields:

$$\boldsymbol{\sigma}_x(\boldsymbol{q}) = \boldsymbol{\sigma}_{x,d}, \tag{2.1}$$

where $\boldsymbol{\sigma}_{x,d}$ denotes the desired task value.

Since Eq. (2.1) is non-linear, its solution cannot be determined by simply inverting it. Differentiating Eq. (2.1) with respect to time, the following expression can be derived:

$$\dot{\boldsymbol{\sigma}}_x(\boldsymbol{q}) = \boldsymbol{J}_x(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{2.2}$$

where $\boldsymbol{J}_x(\boldsymbol{q}) = \partial\boldsymbol{\sigma}_x(\boldsymbol{q})/\partial\boldsymbol{q} \in \mathbb{R}^{m\times n}$ represents the task Jacobian matrix, whereas $\dot{\boldsymbol{q}}$ denotes the system velocity vector. The derived formulation establishes a linear relationship between velocities in the task space and those in the joint space. Consequently, for a given initial configuration, if $m = n$, that is, the number of DoFs of the system is equal to the task dimension, the joint increment needed to bring the task value closer to the desired one can be computed by simply inverting Eq. (2.2) [22]:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_x^{-1}\dot{\boldsymbol{\sigma}}_x(\boldsymbol{q}). \tag{2.3}$$

In the discrete-time formulation of Eq. (2.3), the joint velocities related to sampling instant k are obtained using the inverse of the Jacobian matrix derived from the joint values of the previous step, that is, the step $k-1$. The joint positions $\boldsymbol{q}$ are then recovered by means of a numerical integration procedure:

$$\boldsymbol{q}(t_k) = \boldsymbol{q}(t_{k-1}) + \boldsymbol{J}_x^{-1}(\boldsymbol{q}(t_{k-1}))\dot{\boldsymbol{\sigma}}_x(t_{k-1})\Delta t, \tag{2.4}$$

where $t_k = t_{k-1} + \Delta t$, while Δt represents the integration time step. Although the above numerical integration allows the reconstruction of joint positions, it causes a progressive drift of the solution. This issue is effectively addressed by the Closed-Loop Inverse Kinematics (CLIK) algorithm [23], which relates the system velocity to the task error. According to this algorithm, the system velocity vector required for accomplishing the task can be computed as follows:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_x^{-1}(\boldsymbol{q})(\dot{\boldsymbol{\sigma}}_{x,d} + \boldsymbol{K}_x\tilde{\boldsymbol{\sigma}}_x), \tag{2.5}$$

where $\dot{\boldsymbol{\sigma}}_{x,d}$ denotes the desired task velocity, $\tilde{\boldsymbol{\sigma}}_x$ represents the task error, and $\boldsymbol{K}_x$ represents a positive-definite gain matrix. As for the task error $\tilde{\boldsymbol{\sigma}}_x$, it is worth noting that its mathematical formulation is directly influenced by the specific representation method adopted for describing the end-effector orientation. For instance, under the Euler angles convention, it can be expressed as follows:

$$\tilde{\boldsymbol{\sigma}}_x = \boldsymbol{\sigma}_{x,d} - \boldsymbol{\sigma}_x. \tag{2.6}$$

2.2 Redundant Systems

A robotic system is said to be kinematically redundant when $n > m$, i.e., when the number of its DoFs is greater than those strictly needed to accomplish a given task. Under this condition, the Jacobian matrix becomes

rank-deficient and, consequently, non-invertible, which implies the existence of an infinite number of joint configurations satisfying Eq. (2.2). A widely employed method for addressing this situation involves formulating a constrained optimization problem, whose cost function is expressed as:

$$g(\dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}}, \tag{2.7}$$

to select, among the infinite possible solutions, the one that minimizes the norm of the joint velocities. Consequently, Eq. (2.3) can be rewritten as:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_x^T(\boldsymbol{J}_x\boldsymbol{J}_x^T)^{-1}\dot{\boldsymbol{\sigma}}_x(\boldsymbol{q}) = \boldsymbol{J}_x^\dagger\dot{\boldsymbol{\sigma}}_x(\boldsymbol{q}) \tag{2.8}$$

where $\boldsymbol{J}_x^\dagger$ is the Moore-Penrose pseudoinverse of $\boldsymbol{J}_x$, which satisfies the following conditions:

$$\boldsymbol{J}_x\boldsymbol{J}_x^\dagger\boldsymbol{J}_x = \boldsymbol{J}_x, \quad \boldsymbol{J}_x^\dagger\boldsymbol{J}_x\boldsymbol{J}_x^\dagger = \boldsymbol{J}_x^\dagger, \quad (\boldsymbol{J}_x\boldsymbol{J}_x^\dagger)^\star = \boldsymbol{J}_x\boldsymbol{J}_x^\dagger, \quad (\boldsymbol{J}_x^\dagger\boldsymbol{J}_x)^\star = \boldsymbol{J}_x^\dagger\boldsymbol{J}_x,$$

where $\boldsymbol{J}_x^\star$ denotes the complex-conjugate of $\boldsymbol{J}_x$. Combining Eq. (2.8) with the general formulation of the CLIK algorithm, Eq. (2.5) can be reformulated as:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_x^\dagger(\dot{\boldsymbol{\sigma}}_{x,d} + \boldsymbol{K}_x\tilde{\boldsymbol{\sigma}}_x). \tag{2.9}$$

In general, the solution of Eq. (2.3) lies within the range space of $\boldsymbol{J}_x$ ($\mathcal{R}(\boldsymbol{J}_x)$). For redundant systems, the orthogonal complement of $\mathcal{R}(\boldsymbol{J}_x)$, i.e., the null-space of $\boldsymbol{J}_x$ ($\mathcal{N}(\boldsymbol{J}_x)$), is non-empty and can be utilized to encode secondary objectives, namely additional motion components that do not affect the accomplishment of the primary task. Thus, the general solution can be rewritten as follows:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_x^\dagger\dot{\boldsymbol{\sigma}}_x + \boldsymbol{N}_x\dot{\boldsymbol{q}}_0, \tag{2.10}$$

where $\boldsymbol{N}_x = \boldsymbol{I} - \boldsymbol{J}_x^{\dagger}\boldsymbol{J}_x$ denotes the null space projector, and $\dot{\boldsymbol{q}}_0$ is an arbitrary joint velocity vector, typically chosen to optimize (minimize or maximize) a given a scalar performance index. This is achieved by defining it as follows:

$$\dot{\boldsymbol{q}}_0 = k_0 \left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} \right)^T,$$

where k_0 is a scalar gain and $w(\boldsymbol{q})$ is the secondary objective function.

2.3 Set-Based Control

To effectively control a complex robotic system, the simultaneous execution of multiple tasks is required. These tasks are usually grouped into three categories [24]:

1. Safety tasks (e.g., maintaining joint motions within kinematic limits);
2. Operational tasks (e.g., maintaining or reaching a specified end-effector pose);
3. Optimization tasks (e.g., controlling the swivel angle).

A common approach to solve potential conflicts that may arise among these tasks is to establish a hierarchy, that is, to assign different priority levels to each of them (as shown in Figure 2.1).

For a hierarchy made up of h tasks, h priority levels i should be specified and ordered from the highest priority ($i = 1$) to the lowest ($i = h$). In this situation, the reference system velocity can be determined recursively as [25]:

$$\dot{\boldsymbol{q}}_h = \sum_{i=1}^{h} \left(\boldsymbol{J}_i \boldsymbol{N}_{i-1}^A \right)^{\dagger} \left(\dot{\boldsymbol{\sigma}}_{i,d} + \boldsymbol{K}_i \tilde{\boldsymbol{\sigma}}_i - \boldsymbol{J}_i \dot{\boldsymbol{q}}_{i-1} \right), \tag{2.11}$$

where $\boldsymbol{N}_0^A = \boldsymbol{I}$, $\dot{\boldsymbol{q}}_0 = \boldsymbol{0}$, $\dot{\boldsymbol{q}}_i$ corresponds to the reference system velocity satisfying all tasks from the first to the i-th, whereas $\boldsymbol{N}_i^A$ denotes the null

Figure 2.1: Example of hierarchy with tasks grouped into three categories. Priority decreases from left to right, with safety tasks having higher priority than operational and optimization tasks.

space projector of the augmented Jacobian matrix, obtained by stacking the Jacobian matrices of all tasks from the first through the i-th:

$$\boldsymbol{J}_i^A = \begin{bmatrix} \boldsymbol{J}_1^T & \boldsymbol{J}_2^T & \dots & \boldsymbol{J}_i^T \end{bmatrix}^T, \tag{2.12}$$

and is defined as:

$$\boldsymbol{N}_i^A = \boldsymbol{I} - \left(\boldsymbol{J}_i^A\right)^\dagger \boldsymbol{J}_i^A. \tag{2.13}$$

The control strategy described in Eq. (2.11), like the other main redundancy resolution methods, has been developed to address equality-based tasks, i.e., cases in which the objective is to bring the task variable $\boldsymbol{\sigma}_x$ to a prescribed value (e.g., moving the arm end-effector to a target position). However, several tasks may require their value to lie within an interval, i.e., between a lower and an upper threshold. As a result, classical task-priority frameworks have been extended to accommodate these types of tasks as well [26, 27], which are typically identified in the literature as *set-based* tasks or inequality constraints. This improvement relies on the concept that a *set-based* task can be seen as an equality-based one whose activation is determined by its current value. To formalize this concept, a

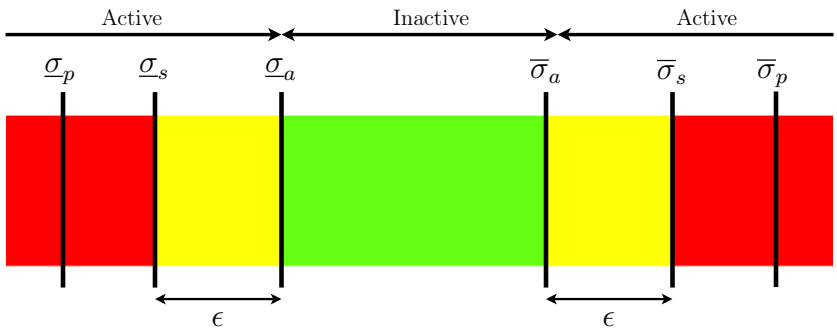


Figure 2.2: Activation and physical thresholds of a set-based task.

set of reference thresholds must be defined for each set-based task, including physical thresholds $\overline{\boldsymbol{\sigma}}_p$ ($\underline{\boldsymbol{\sigma}}_p$), safety thresholds $\overline{\boldsymbol{\sigma}}_s < \overline{\boldsymbol{\sigma}}_p$ ($\underline{\boldsymbol{\sigma}}_s > \underline{\boldsymbol{\sigma}}_p$), and activation thresholds $\overline{\boldsymbol{\sigma}}_a = \overline{\boldsymbol{\sigma}}_s - \boldsymbol{\epsilon}$ ($\underline{\boldsymbol{\sigma}}_a = \underline{\boldsymbol{\sigma}}_s + \boldsymbol{\epsilon}$). Figure 2.2 depicts all the aforementioned thresholds.

Whenever the value of a task exceeds its activation threshold, it must be incorporated into the hierarchy as an equality-based task, with its desired value determined as follows:

$$\boldsymbol{\sigma}_d = \begin{cases} \overline{\boldsymbol{\sigma}}_s & \text{if } \boldsymbol{\sigma} \geq \overline{\boldsymbol{\sigma}}_a, \\ \underline{\boldsymbol{\sigma}}_s & \text{if } \boldsymbol{\sigma} \leq \underline{\boldsymbol{\sigma}}_a. \end{cases} \tag{2.14}$$

A task can be deactivated when the solution of the hierarchy, derived by considering all tasks except the one in question, tends to drive its value toward the feasible set. Specifically, for a set-based task $\boldsymbol{\sigma}_A$, let $\dot{\boldsymbol{q}}$ the solution obtained from the hierarchy excluding it, and let $\boldsymbol{J}_A$ denote its Jacobian matrix. If $\boldsymbol{J}_A\dot{\boldsymbol{q}} > 0$ ($\boldsymbol{J}_A\dot{\boldsymbol{q}} < 0$) the obtained solution would increase (decrease) the set-based task value. Accordingly, the set-based task $\boldsymbol{\sigma}_A$ can be deactivated if:

$$\boldsymbol{\sigma}_A \geq \overline{\boldsymbol{\sigma}}_a \ \wedge \ \boldsymbol{J}_A\dot{\boldsymbol{q}} < 0, \tag{2.15}$$

or

$$\boldsymbol{\sigma}_A \leq \underline{\boldsymbol{\sigma}}_a \ \wedge \ \boldsymbol{J}_A\dot{\boldsymbol{q}} > 0. \tag{2.16}$$

It is important to emphasize that the inequalities presented above hold element-wise.

2.4 Control Barrier Functions

In this section, the basic theory of Control Barrier Functions (CBFs) is provided, which will be used in the following chapters to formalize several control objectives aimed at assuring constraints such as the safety of the robot [28]. Consider a general system having the following :

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(t, \boldsymbol{\xi}) + \boldsymbol{g}(\boldsymbol{\xi})\boldsymbol{u}, \tag{2.17}$$

where $\boldsymbol{f}$ and $\boldsymbol{g}$ are Lipschitz-continuous vector fields, $\boldsymbol{\xi} \in \mathcal{D} \subset \mathbb{R}^l$ and $\boldsymbol{u} \in \mathcal{U} \subset \mathbb{R}^q$ are the state and the input of the system, respectively.

Let the k-th generic constraint be expressed in the following general form: $h_k(\boldsymbol{\xi}) \geq 0$, where $h_k(\cdot)$ is a continuous differentiable function in the domain $\mathcal{D}$. According to the CBF framework, let $\mathcal{C}_k \subset \mathcal{D}$ be defined as:

$$\begin{aligned} \mathcal{C}_k &= \{\boldsymbol{\xi} \in \mathbb{R}^l : h_k(\boldsymbol{\xi}) \geq 0\}, \\ \partial\mathcal{C}_k &= \{\boldsymbol{\xi} \in \mathbb{R}^l : h_k(\boldsymbol{\xi}) = 0\}, \\ \mathrm{Int}(\mathcal{C}_k) &= \{\boldsymbol{\xi} \in \mathbb{R}^l : h_k(\boldsymbol{\xi}) > 0\}, \end{aligned} \tag{2.18}$$

implying that the state $\boldsymbol{\xi}$ is required to belong to the set $\mathcal{C}_k$ in order to satisfy constraint k. Function h_k is a CBF if an extended class $\mathcal{K}_\infty$ function α_k exists such that, for a dynamic system represented as in Eq. (2.17), it holds:

$$\sup_{\boldsymbol{u} \in \mathcal{U}} [L_f h_k(\boldsymbol{\xi}) + L_g h_k(\boldsymbol{\xi})\boldsymbol{u}] \geq -\phi_k \alpha_k(h_k(\boldsymbol{\xi})), \tag{2.19}$$

where $\phi_k > 0$, and $L_f h_k$ and $L_g h_k$ are the Lie derivatives of function h_k with respect to f and g, respectively. Then, the following theorem holds [28].

Theorem 1. *Let function $h_k : \mathcal{D} \subset \mathbb{R}^l \to \mathbb{R}$ be a continuously differentiable function and the corresponding set $\mathcal{C}_k$ defined as in Eq. (2.18). If h_k*

is a CBF on $\mathcal{D}$ *and* $\frac{\partial h_k}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}) \neq 0 \ \forall \boldsymbol{\xi} \in \partial \mathcal{C}_k$*, then any Lipschitz continuous controller* $u(\boldsymbol{\xi})$ *for system in Eq.* (2.17) *satisfying Eq.* (2.19) *renders the set* $\mathcal{C}_k$ *asymptotically stable.*

Proof. The proof can be found in [28]. □

Remarkably, since Eq. (2.19) is affine in the control input $\boldsymbol{u}$, the latter can be computed as the result of a convex optimization problem subject to the constraint:

$$\begin{aligned} \boldsymbol{u}^\star = \arg\min_{\boldsymbol{u}} \ & \frac{1}{2} \left(\boldsymbol{u} - \boldsymbol{u}_{(\cdot)}\right)^T \boldsymbol{Q} \left(\boldsymbol{u} - \boldsymbol{u}_{(\cdot)}\right) \\ \text{s.t.} \quad & \sup_{\boldsymbol{u} \in \mathcal{U}} \left[L_f h_k(\boldsymbol{\xi}) + L_g h_k(\boldsymbol{\xi}) \boldsymbol{u}\right] \geq -\phi_k h_k(\boldsymbol{\xi}), \quad \forall\, k \end{aligned} \tag{2.20}$$

where $\boldsymbol{u}_{(\cdot)}$ is any nominal input for the system and $\boldsymbol{Q} \in \mathbb{R}^{q \times q}$ is a positive definite weight matrix.

2.5 Hierarchical Quadratic Programming

The Hierarchical Quadratic Programming (HQP) control framework computes the control signal that optimally fulfills a given task hierarchy, minimizing the error of lower-priority tasks when full accomplishment is constrained by higher-priority ones. The HQP formulation converts this hierarchy into a series of Quadratic Programming (QP) problems [29]. In this setup, the solution to the i-th problem (associated with the task of priority i) is obtained by treating the solutions of the QP problems for higher-priority tasks as additional constraints. It is worth noticing that in [29], the result of this procedure is proven to be equal to the traditional null-space projection techniques in [30, 31], at least for task hierarchies containing only equality constraints.

More in-depth, consider a task $\boldsymbol{\sigma}_1$ that is related to the system velocity vector $\dot{\boldsymbol{q}}$ through the Jacobian matrix $\partial \boldsymbol{\sigma}_1 \backslash \partial \boldsymbol{q} = \boldsymbol{J}_1(\boldsymbol{q})$, and the minimum and maximum desired task velocity $\underline{\boldsymbol{b}}_1$ and $\bar{\boldsymbol{b}}_1$, respectively. The associated QP problem that computes the solution that fulfills the task can be expressed as:

$$\begin{aligned} \min_{\boldsymbol{w}_1, \dot{\boldsymbol{q}}} \quad & \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{Q}_1 \dot{\boldsymbol{q}} + \boldsymbol{w}_1^T \boldsymbol{Q}_{w_1} \boldsymbol{w}_1 \\ \text{s.t.} \quad & \underline{\boldsymbol{b}}_1 \leq \boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_1 \leq \bar{\boldsymbol{b}}_1, \end{aligned} \tag{2.21}$$

where $\boldsymbol{w}_1$ is a *slack variable* vector associated with the task $\boldsymbol{\sigma}_1$ to relax its constraint in case of non-feasibility of the task, $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_{w_1}$, instead, are the cost matrices associated with the decision variables vector $\dot{\boldsymbol{q}}$ and the slack variables vector $\boldsymbol{w}_1$, respectively. It is worth noticing that in case $\underline{\boldsymbol{b}}_1 = \bar{\boldsymbol{b}}_1 = \boldsymbol{b}_1$, the bilateral constraint in Eq. (2.21) is equivalent to an equality constraint (that is $\boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_1 = \boldsymbol{b}_1$).

Now, consider the case in which there is also a task $\boldsymbol{\sigma}_2$ (with related Jacobian $\boldsymbol{J}_2(\boldsymbol{q})$) to be performed with a strict lower priority than the task $\boldsymbol{\sigma}_1$. In this case, to obtain the solution is necessary to solve two separate QP problems. The first one will be structured as (2.21), the second one, instead, as follows:

$$\begin{aligned} \min_{\boldsymbol{w}_2, \dot{\boldsymbol{q}}} \quad & \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{Q}_2 \dot{\boldsymbol{q}} + \boldsymbol{w}_2^T \boldsymbol{Q}_{w_2} \boldsymbol{w}_2 \\ \text{s.t.} \quad & \underline{\boldsymbol{b}}_1 \leq \boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_1^* \leq \bar{\boldsymbol{b}}_1 \\ & \underline{\boldsymbol{b}}_2 \leq \boldsymbol{J}_2(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_2 \leq \bar{\boldsymbol{b}}_2, \end{aligned} \tag{2.22}$$

where $\boldsymbol{w}_1^*$ is the solution of the first QP problem, while $\boldsymbol{w}_2$ is the slack variables vector associated with the task $\boldsymbol{\sigma}_2$, allowing a solution to be found even when the two tasks are in conflict. The minimization of $\boldsymbol{w}_2$ in the objective function of Eq.(2.22) aims at reducing the error on the secondary task $\boldsymbol{\sigma}_2$ given the fulfillment of the constraints required by the primary task $\boldsymbol{\sigma}_1$.

The described approach can be generalized and employed to handle a hierarchy consisting of k arbitrary tasks. By iterating the method described above, it will be necessary to solve a cascade of k QP problems, with the i-th problem structured as follows:

$$\begin{aligned} \min_{\boldsymbol{w}_i, \dot{\boldsymbol{q}}} \quad & \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{Q}_i \dot{\boldsymbol{q}} + \boldsymbol{w}_i^T \boldsymbol{Q}_{w_i} \boldsymbol{w}_i \\ \text{s.t.} \quad & \underline{\boldsymbol{b}}_k \leq \boldsymbol{J}_k(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_k^* \leq \bar{\boldsymbol{b}}_k \quad \forall k \in 1, \ldots, i-1 \\ & \underline{\boldsymbol{b}}_i \leq \boldsymbol{J}_i(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_i \leq \bar{\boldsymbol{b}}_i, \end{aligned} \tag{2.23}$$

It is important to emphasize that the inequalities presented above hold element-wise.

2.6 Mixed-Integer Linear Programming

A significant part of real-world problems, including, but not limited to, path planning [32] and scheduling [33, 34], can be formulated as Constraint Optimization Problems (COPs), which are typically NP-hard. Linear Programming (LP), Integer Linear Programming (ILP), and Mixed-Integer Linear Programming (MILP) are among the most well-established mathematical optimization methodologies and are extensively employed to address these complex scenarios.

Within optimization theory, mathematical models describing the scenarios under investigation generally comprise an objective function, which specifies the quantity to be optimized, together with a set of constraints, which describe the limitations or requirements to be observed. In this context, Linear Programming (LP) constitutes a well-established mathematical optimization method for determining the optimal value, which can be either the minimum or the maximum, of a linear objective function subject to a set of linear constraints. In LP, the objective function and constraints may be formulated as follows

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & Z = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n \\ \text{s.t.} \quad & a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq (\geq, =)\ b_1 \\ & a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leq (\geq, =)\ b_2 \\ & \qquad\qquad \vdots \\ & a_{m1} x_1 + a_{m2} x_2 + \ldots + a_{mn} x_n \leq (\geq, =)\ b_m \end{aligned} \tag{2.24}$$

where $c_1, c_2, \ldots, c_n \in \mathbb{R}$ are the coefficients of the objective function,

$x_1, x_2, \ldots, x_n \in \mathbb{R}$ are the decision variables, $a_{1,1}, a_{1,2}, \ldots, a_{m,n} \in \mathbb{R}$ are the coefficients of the constraints, $b_1, b_2, \ldots, b_n \in \mathbb{R}$ are the right-hand side values of constraints, m is the number of constraints, and n is the number of decision variables.

Integer Linear Programming (ILP) is another well-established mathematical optimization technique to determine the optimal value of a linear objective function under a set of linear constraints. However, unlike the previously discussed technique, ILP relies on distinct resolution procedures (i.e., Branch-and-Bound and Branch-and-Cut) and to be implemented, it necessitates that all decision variables take only integer values (i.e., $x_1, x_2, \ldots, x_n \in \mathbb{Z}$).

Finally, Mixed-Integer Linear Programming (MILP) represents another widely acknowledged mathematical optimization technique for identifying the optimal value of a linear objective function under a set of linear constraints. Although it relies on resolution procedures analogous to those of ILP (i.e., Branch-and-Bound and Branch-and-Cut), MILP differs from ILP in that it permits decision variables to take both integer and continuous values, thereby enhancing its versatility and making it particularly suitable for addressing complex real-world scenarios. A typical MILP problem is generally structured as follows

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{y}} \quad & Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n + d_1y_1 + d_2y_2 + \ldots + d_my_m \\
\text{s.t.} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + a'_{11}y_1 + a'_{12}y_2 + \ldots + a'_{1m}y_m \\
& \quad \leq (\geq, =) b_1 \\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + a'_{21}y_1 + a'_{22}y_2 + \ldots + a'_{2m}y_m \\
& \quad \leq (\geq, =) b_2 \\
& \qquad\qquad\qquad \vdots \\
& a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kn}x_n + a'_{k1}y_1 + a'_{k2}y_2 + \ldots + a'_{km}y_m \\
& \quad \leq (\geq, =) b_k
\end{aligned}
\tag{2.25}
$$

where $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ are the integer decision variables, whereas $y_1, y_2, \ldots, y_m \in \mathbb{R}$ are the continuous decision variables [35, 36].

2.7 Constraint Programming

Constraint Programming (CP) is a logic-based approach that draws on a wide range of techniques from artificial intelligence, operations research, computer science, and mathematics [37]. Although mainly used for solving Constraint Satisfaction Problems (CSPs) [38], i.e., problems concerned with finding values for a set of variables subject to predefined constraints and/or rules, in recent decades, thanks to the development of more efficient CP solvers, such as IBM's CP Optimizer[1], its applicability has expanded to a broader range of domains. In particular, it has been successfully applied to a variety of Constraint Optimization Problems (COPs), including timetabling, factory scheduling, and the allocation of times or resources to events (e.g, assigning exams to timeslots or nurses to shifts) [39, 40].

A distinctive feature of CP is its ability to handle a wide range of decision variables, including integer, continuous, and interval variables. Interval variables are typically used to model time windows within which events may occur. The start time of these intervals, and possibly their duration, are among the unknowns that must be determined. Interval variables can also be *optional*, meaning that they may not be included in the solution and thus be ignored by any constraint or expression involving them. Such variables are commonly used to model optional tasks (i.e., tasks whose execution may not occur) or to represent alternatives in tasks and resources (i.e., cases where the same task can be performed in different ways). In CP, each decision variable is associated with a domain, that is, the set of all values it may assume. During the solving phase, infeasible values are progressively removed from these domains through an iterative mechanism known as constraint propagation. For this reason, domains are often regarded as dynamic or living data structures. In contrast to other approaches, CP is not limited to linear expressions but also supports non-linear cost functions and constraints. In addition, it natively handles a wide range of constraint types, including arithmetic expressions, logical disjunctions, conditional if–then rules, and global constraints. Global constraints are a special class of constraints that, unlike local or primitive constraints (e.g., arithmetic expressions),

[1] https://www.ibm.com/optimization-solver

are characterized by a non-fixed number of arguments and thus are capable of capturing relationships among an arbitrary number of variables. Such constraints are typically semantically redundant, in the sense that the same relation can also be expressed through the conjunction of multiple local (primitive) constraints. There are two main advantages resulting from the use of these constraints. First, they facilitate problem modeling, as they encapsulate some of those complex logical or combinatorial concepts commonly found in real-world scenarios (e.g., "*everything must be different*" or "*tasks must not overlap*"). Second, each of them comes with a specialized filtering algorithm capable of pruning the search space far more effectively than would occur if the same constraint were implemented using dozens or even hundreds of local (primitive) constraints. Two representative global constraints are allDifferent(·) and noOverlap(·) constraints. The allDifferent(·) constraint enforces that the values assigned to the specified variables must all be pairwise distinct, whereas the noOverlap(·) constraint requires that the tasks or events associated with the listed interval variables do not overlap. More details on CP can be found in [41].

2.8 Large Language Models

Large Language Models (LLMs) are advanced artificial intelligence systems built on deep learning architectures, such as the Transformer architecture, designed to perform diverse natural language processing tasks, including text generation, translation, question-answering, and summarization. In recent years, these systems have demonstrated significant capabilities in understanding and generating human-like text. These abilities derive mainly from the intensive training processes performed on vast quantities of diverse textual data, including books, articles, and other written materials [42].

In the generative artificial intelligence (AI) field, prompts are essential because they serve as the primary means of interacting with LLMs, which are designed to process and generate responses based on them [43]. In recent years, the research community has devoted increasing attention to prompts because they provide a lightweight and effective mechanism to control LLM behavior, thereby minimizing reliance on parameter-tuning procedures that are typically time-consuming and labor-intensive [44].

These are generally classified into two main categories: system prompts and user prompts. The system prompt is typically composed by the developers or deployers of the architecture and serves to outline operational conditions and parameters such as the model's role within the overall system, the objectives to be achieved, the constraints to be observed, and the expected output format. Conversely, the user prompt is generated by either end-users or connected modules and typically comprises questions and/or commands related to either the task to be executed or the mission to be achieved. As the quality of prompts strongly influences the accuracy of model outputs, their design has become a topic of significant focus. The discipline of prompt engineering specifically addresses this need by formulating methodologies, strategies, and best practices for the design, refinement, and optimization of prompts to improve model efficacy [42, 43, 44].

Due to their extensive world knowledge and capacity to comprehend and generate human-like text, LLMs are considered versatile tools applicable across numerous domains, including medicine and automation. Within healthcare contexts, for instance, these models are usually employed either as diagnostic support systems for disease identification and treatment planning or as virtual agents for providing essential health suggestions and addressing patients' inquiries regarding symptoms [45]. In contrast, in automation contexts, these models, in addition to being adopted to enhance the navigation and manipulation capabilities of robotic systems, are also employed to enhance and facilitate human-robot collaboration and interaction [46].

As outlined above, LLMs possess advanced reasoning and grounding capabilities, making them versatile tools that can be deployed in diverse scenarios. Although these models offer considerable advantages, their deployment also introduces several critical challenges that need to be addressed. First, LLMs operate as next token predictors; that is, their output is not the result of a logical reasoning process, but of a process based on statistical likelihood, namely, generating text by calculating the probability of which word is most likely to come next in a sequence, based on patterns learned from massive amounts of training data. Then, there is no guarantee about the correctness of the produced output. Moreover, in complex scenarios, such as those involving the control of robotic systems operating in dynamic and unstructured environments, an additional

training process might be necessary for enhancing the efficiency and capabilities of these models. However, these processes are demanding in terms of time and resources. Finally, they are highly prone to hallucination phenomena, that is, they have a pretty good chance of generating plausible yet incorrect responses [47, 48, 49].

Chapter 3

A Unified HQP–CBF Framework for Hierarchical Control of Redundant Robotic Platforms

3.1 Introduction

As outlined in Section 1.1, the growing need for flexibility across diverse production domains has, particularly in recent years, fostered the adoption of increasingly complex automated systems [1]. In contrast to traditional industrial manufacturing settings, where robots usually perform repetitive tasks in a controlled and structured environment, these contexts usually require the execution of advanced manipulation operations in unstructured and inherently unpredictable conditions. In more detail, working in such scenarios requires systems with more complex architectures and enhanced reasoning capabilities compared to those adopted in traditional manufacturing. This is particularly true in agriculture, where, owing to the rising global demand for food, more and more dual-arm systems, potentially mounted on a mobile base, are employed for pruning and harvesting procedures. Such systems are typically equipped with additional DoFs, which, although challenging to handle, play an essential role as they can be exploited to generate internal motions that improve the robot's posture. Nevertheless, to make an effective use of them, several constraints must be managed at the control level, especially for properly

coordinating the motion of the different parts of the robot and avoiding collisions among them, such as between arms or between arms and the mobile base itself.

As outlined in Section 2.2, when a robotic system has more DoFs than those strictly required to perform a given task, it is usually referred to as kinematically redundant for that task, and this redundancy must be properly handled in the computation of the control input, to make an effective use of the system. In the literature, this problem is typically referred to as redundancy resolution and has been extensively investigated for several decades. A widespread approach to address redundancy is to exploit the additional DoFs to allow the robotic system to perform multiple tasks simultaneously. Concerning potential conflicts that may occur among tasks, these are typically addressed by establishing a priority hierarchy (i.e., by assigning tasks to different priority levels) and computing the joint velocities that best satisfy the resulting task hierarchy. Some of the earliest works in this direction include [30, 31, 50], where the authors proposed the so-called *null-space projection* technique, which consists in the projection of the joint velocities of lower-priority tasks into the null space of the Jacobian matrices of the higher-priority ones. A similar approach, aimed at overcoming algorithmic singularities, was later proposed in [51] and then generalized to the multi-task case in [52]. More recently, the aforementioned class of control frameworks has been further extended to allow the inclusion, within the hierarchy, of control objectives aimed at keeping the task value within desired thresholds rather than enforcing it to take a specific value [53]. As stated in Section 2.3, these tasks are usually referred to as *inequality constraints* [54] or *set-based tasks* [55].

A well-established alternative to the null-space projection method is the HQP framework (see Section 2.5), in which the joint velocities that satisfy the hierarchy are obtained by solving a cascade of QP problems [29]. Due to the efficiency of modern QP solvers, this approach can now also be applied to highly redundant systems, such as humanoids [56, 57] and multi-robot systems [58], with good scalability of execution time with respect to the number of constraints considered, particularly when compared to other task-priority frameworks. In contrast to traditional null-space projection techniques, the HQP formulation offers a notable advantage of inherently addressing inequality constraints, thereby facilitating both the formulation and integration of safety constraints that the system

must satisfy. In this perspective, a rigorous and powerful mathematical approach for enforcing system safety are the Control Barrier Functions (CBF) [28, 59] (see Section 2.4), which are frequently employed in conjunction with Control Lyapunov Functions (CLFs) [60] within a single QP problem to realize the so-called *safety-critical control* [61, 62].

In response to the requirements imposed by modern society outlined at the beginning of this chapter, a comprehensive control architecture has been developed, capable of addressing multiple kinematic constraints while enabling the generation of internal motions to optimize diverse objective functions. Specifically, the devised architecture is an HQP-based control framework suitable for complex systems, such as mobile dual-arm robotic systems operating in precision agriculture scenarios, that integrates both equality and inequality constraints to guarantee robust task execution. In addition to the HQP framework, the proposed architecture also leverages the CBFs formulation as a formal mechanism for specifying and enforcing safety constraints, thereby preventing self-collisions and enhancing overall system reliability.

As previously mentioned, the development of the proposed architecture is framed within the domain of precision agriculture. Specifically, to validate the approach, a number of experiments have been conducted in scenarios inspired by the H2020 CANOPIES project[1], an EU-funded project involving robotic platforms engaged in table-grape harvesting procedures alongside human operators, and aiming to develop novel paradigms for human–multi-robot collaboration and interaction in agricultural contexts.

It is worth emphasizing that, although the proposed architecture was developed and validated in the context of precision agriculture, it relies on a general formulation and can therefore be effectively employed in a wide range of domains.

The rest of the chapter is organized as follows. Section 3.2 outlines the architecture and kinematics of a redundant robotic system, specifically a dual-arm tracked mobile robot. Section 3.3 presents an overview of the developed control framework and its main components. Section 3.4 discusses the concept of task, including its classification and mathematical formulation within the proposed framework. Finally, Sections 3.5 and 3.6

[1] www.canopies-project.eu

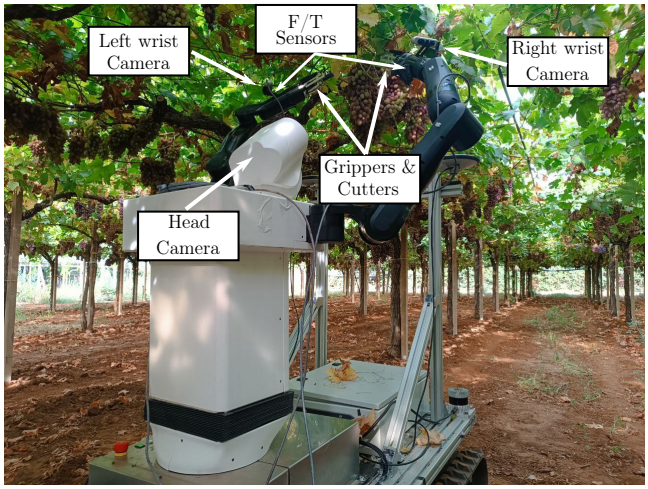


Figure 3.1: Dual-arm tracked mobile robot designed for the EU-funded project CANOPIES. In particular, it highlights: the head and the wrist cameras, the end-effectors equipped with gripper and cutter, and Force/Torque (F/T) sensors.

provide detailed descriptions of the simulated and real-world experiments conducted to validate the effectiveness of the proposed framework.

3.2 Robot Kinematics

Consider a robotic system similar to the one shown in Figure 3.1, that is, a redundant mobile robot composed of: *i*) a mobile base, *ii*) a movable torso, and *iii*) a dual-arm system, with n_b, n_t, and $2n_a$ DoFs, respectively. For a system of this kind, the state is defined as the vector stacking all the joint position vectors, which can be expressed as:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_b^T & \boldsymbol{q}_t^T & \boldsymbol{q}_l^T & \boldsymbol{q}_r^T \end{bmatrix}^T \in \mathbb{R}^n, \tag{3.1}$$

where $\boldsymbol{q}_b \in \mathbb{R}^{n_b}$ is the vector gathering joint variables related to the mobile base, $\boldsymbol{q}_t \in \mathbb{R}^{n_t}$ is the vector gathering joint variables related to the torso, $\boldsymbol{q}_l \in \mathbb{R}^{n_a}$ is the vector of joint variables related to the left arm, $\boldsymbol{q}_r \in \mathbb{R}^{n_a}$ is the vector of the joint variables related to the right arm, and $n = n_b + n_t + 2n_a$ is the number of DoFs of the entire system.

The velocity of the system, on the other hand, is the vector stacking all joint velocity vectors. It can be computed by differentiating with respect to time the state vector of the system in Eq. (3.1) and can be represented as:

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\boldsymbol{q}}_b^T & \dot{\boldsymbol{q}}_t^T & \dot{\boldsymbol{q}}_l^T & \dot{\boldsymbol{q}}_r^T \end{bmatrix}^T \in \mathbb{R}^n, \tag{3.2}$$

where $\dot{\boldsymbol{q}}_b \in \mathbb{R}^{n_b}$ is the vector gathering the linear and angular velocities of the mobile base.

Building on the above considerations, consider the configuration and generalized velocity vectors of the end-effector i. The former can be represented as:

$$\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{p}_i^T & \boldsymbol{o}_i^T \end{bmatrix}^T \in \mathrm{SE}(3), \tag{3.3}$$

where $i \in \{l, r\}$ denotes the quantities associated with left and right arms, respectively, $\boldsymbol{p}_i \in \mathbb{R}^3$ represents the end-effector position vector, and $\boldsymbol{o}_i = \begin{bmatrix} o_{i,1} & o_{i,2} & o_{i,3} & o_{i,4} \end{bmatrix}^T = \begin{bmatrix} \kappa_{o_i} & \boldsymbol{\varrho}_{o_i}^T \end{bmatrix}^T \in \mathrm{SO}(3)$ denotes the end-effector unit quaternion, i.e., the four-component vector describing the end-effector orientation, with κ_{o_i} and $\boldsymbol{\varrho}_{o_i} \in \mathbb{R}^3$ denoting its scalar and vector components, respectively. In contrast, the latter can be represented as:

$$\boldsymbol{v}_i = \begin{bmatrix} \dot{\boldsymbol{p}}_i^T & \boldsymbol{\omega}_i^T \end{bmatrix}^T \in \mathbb{R}^6, \tag{3.4}$$

where $\dot{\boldsymbol{p}}_i \in \mathbb{R}^3$ denotes the end-effector linear velocity vector while $\boldsymbol{\omega}_i \in \mathbb{R}^3$ represents the end-effector angular velocity vector.

By stacking the configuration vectors of the two end-effectors, the following expression is obtained:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_l^T & \boldsymbol{x}_r^T \end{bmatrix}^T \in \mathrm{SE}(3) \times \mathrm{SE}(3), \tag{3.5}$$

which represents the Cartesian configuration of the dual-arm system. On the other hand, by stacking the generalized velocity vectors of both the end-effectors, the following expression is obtained:

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_l^T & \boldsymbol{v}_r^T \end{bmatrix}^T \in \mathbb{R}^{12}, \tag{3.6}$$

which represents the Cartesian velocity of the dual-arm system.

The differential relationship between the generalized velocity vector $\boldsymbol{v}$, describing the Cartesian velocity of the dual-arm system, and the system velocity vector $\dot{\boldsymbol{q}}$, describing its joint velocities, can be expressed as

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_l \\ \boldsymbol{v}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_l(\boldsymbol{q}) \\ \boldsymbol{J}_r(\boldsymbol{q}) \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{J}_\gamma(\boldsymbol{q})\ \dot{\boldsymbol{q}}, \tag{3.7}$$

where $\boldsymbol{J}_i(\boldsymbol{q}) \in \mathbb{R}^{6\times n}, i \in \{l, r\}$ is the Jacobian matrix relating the joint velocities of the base ($\dot{\boldsymbol{q}}_b$), torso ($\dot{\boldsymbol{q}}_t$), and arm i ($\dot{\boldsymbol{q}}_i$) to the generalized velocity vector $\boldsymbol{v}_i$, which describes the linear and angular velocities of end-effector i.

Based on the above considerations, the overall Jacobian matrix $\boldsymbol{J}_\gamma(\boldsymbol{q}) \in \mathbb{R}^{12\times n}$, namely the matrix relating the generalized velocity vectors of both end-effectors to the system velocity vector, can be partitioned as follows

$$\boldsymbol{J}_\gamma(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{J}_{b,l}(\boldsymbol{q}_b) & \boldsymbol{J}_{t,l}(\boldsymbol{q}_t) & \boldsymbol{J}_{a,l}(\boldsymbol{q}_l) & \boldsymbol{O}_{6\times n_a} \\ \boldsymbol{J}_{b,r}(\boldsymbol{q}_b) & \boldsymbol{J}_{t,r}(\boldsymbol{q}_t) & \boldsymbol{O}_{6\times n_a} & \boldsymbol{J}_{a,r}(\boldsymbol{q}_r) \end{bmatrix}, \tag{3.8}$$

where $\boldsymbol{J}_{b,i}(\boldsymbol{q}_b), i \in \{l, r\}$ is the Jacobian matrix relating the joint velocities of the base ($\dot{\boldsymbol{q}}_b$) to the generalized velocity vector $\boldsymbol{v}_i$, $\boldsymbol{J}_{t,i}(\boldsymbol{q}_t), i \in \{l, r\}$ is the Jacobian matrix relating the joint velocities of the torso ($\dot{\boldsymbol{q}}_t$) to the generalized velocity vector $\boldsymbol{v}_i$, $\boldsymbol{J}_{a,i}(\boldsymbol{q}_i), i \in \{l, r\}$ is the Jacobian matrix relating the joint velocities of the arm i ($\dot{\boldsymbol{q}}_i$) to the generalized velocity vector $\boldsymbol{v}_i$, while $\boldsymbol{O}_{x\times y}$ is a zero matrix of size $(x \times y)$. The extended form highlights that the base and torso joint velocities influence the velocities of both the end-effectors, whereas $\dot{\boldsymbol{q}}_l$ and $\dot{\boldsymbol{q}}_r$ affect only the velocities of the left and right end-effectors, respectively.

3.3 Control Framework

Mobile manipulators, regardless of their complexity, are usually equipped with a built-in velocity interface that allows sending velocity reference

commands to the robot at a given frequency (e.g., 100, Hz). Building on this interface, the control scheme shown in Figure 3.2 has been devised, whose objective is to provide such systems with manipulation functionalities and allow them to exhibit various behaviors. As illustrated in the figure, it consists of four modules: *i*) the Perception System; *ii*) Trajectory Generation; *iii*) the High-Level Layer; and *iv*) the Low-Level Layer.

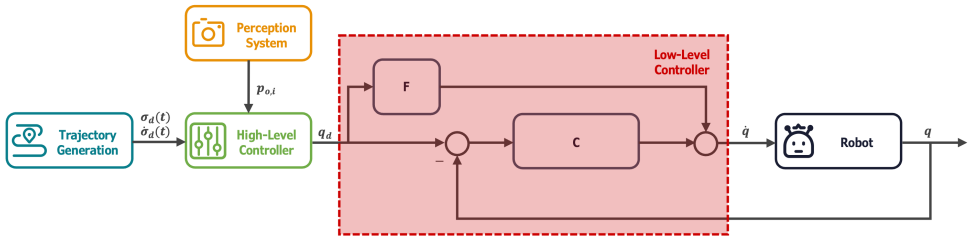


Figure 3.2: Block diagram showing the schematic of the robot control.

In this section, an overview of the proposed control architecture is provided.

3.3.1 Low-Level Layer

The low-level layer is designed to track the reference signal generated by the high-level layer. It employs a proportional (C) and derivative (F) control action, resulting in the following velocity reference for the built-in controller:

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_d + \boldsymbol{K}_q(\boldsymbol{q}_d - \boldsymbol{q}), \tag{3.9}$$

where $\boldsymbol{K}_q$ is a diagonal gain matrix. Although structurally simple, this controller has shown good performance in joint configuration tracking, as highlighted by the results in Section 3.5.

3.3.2 High-Level Layer

The high-level layer is responsible for combining all elementary control objectives according to their priority levels, thereby providing the system with the necessary manipulation functionalities and, in particular, enabling it to exhibit various behaviors. To this end, multiple

control schemes were investigated, in particular the Set-Based Closed-Loop Inverse Kinematics and the Hierarchical Quadratic Programming approaches, detailed in Sections 2.3 and 2.5, respectively. However, the HQP was selected because, in comparison with conventional null-space projection techniques, it has the advantage of intrinsically handling inequality constraints, facilitating the formulation of safety constraints that the system has to respect.

At each time step, the controller computes the reference signals for the low-level layer that best satisfy all prioritized tasks in accordance with the given lexicographic order.

3.4 Tasks

Consider a generic task $\boldsymbol{\sigma}_x$ of the robotic system expressed as:

$$\boldsymbol{\sigma}_x = \boldsymbol{f}_x(\boldsymbol{q}_b, \boldsymbol{q}_t, \boldsymbol{q}_l, \boldsymbol{q}_r) = \boldsymbol{f}_x(\boldsymbol{q}). \tag{3.10}$$

As outlined in Section 2.1, the relationship between the system velocity $\dot{\boldsymbol{q}}$ and the task velocity $\dot{\boldsymbol{\sigma}}_x$, similarly to that between the system velocity $\dot{\boldsymbol{q}}$ and the generalized velocity vector $\boldsymbol{v}$ shown in Eq. (3.7), can be expressed through a Jacobian matrix:

$$\dot{\boldsymbol{\sigma}}_x = \boldsymbol{J}_x(\boldsymbol{q})\dot{\boldsymbol{q}}, \quad \boldsymbol{J}_x(\boldsymbol{q}) = \begin{bmatrix} \overbrace{\frac{\partial \boldsymbol{f}_x(\boldsymbol{q})}{\partial \boldsymbol{q}_b}}^{\text{base}} & \overbrace{\frac{\partial \boldsymbol{f}_x(\boldsymbol{q})}{\partial \boldsymbol{q}_t}}^{\text{torso}} & \overbrace{\frac{\partial \boldsymbol{f}_x(\boldsymbol{q})}{\partial \boldsymbol{q}_l}}^{\text{arm left}} & \overbrace{\frac{\partial \boldsymbol{f}_x(\boldsymbol{q})}{\partial \boldsymbol{q}_r}}^{\text{arm right}} \end{bmatrix}. \tag{3.11}$$

As shown in Eq. (3.11), the columns of the Jacobian matrix can be divided into blocks, each associated with a specific set of DoFs of the system and describing the relationship between the task velocity ($\dot{\boldsymbol{\sigma}}_x$) and the DoFs corresponding to that block. Such a decomposition makes it possible to activate or deactivate selected degrees of freedom according to the context, allowing, for example, the base and/or torso to be kept still when one or more human operators are in proximity to the robot. Specifically, this is achieved by setting all columns associated with the DoFs to be deactivated to zero.

Each task results in an elementary control objective that can be combined, at different priority levels, with other elementary control objectives to achieve various robot behaviors. As discussed in Section 2.3, according to its specific control objective, each task may impose either an inequality or an equality constraint.

Regarding tasks that impose an inequality constraint, they are typically characterized by a minimum and/or a maximum admissible value, denoted by $\underline{\boldsymbol{\sigma}}_x$ and $\bar{\boldsymbol{\sigma}}_x$, respectively. Before being incorporated into the hierarchy arising from the combination of the elementary objectives that produce the desired behavior, the constraints resulting from these tasks are first encoded in proper CBFs $h_x(\boldsymbol{q})$ following the approach recalled in Section 2.4, and are then included as an inequality constraint according to Eq. (2.23) and the formulation in Section 2.5.

Regarding tasks that impose equality constraints, it is assumed that a desired trajectory for the task value $\boldsymbol{\sigma}_{x,d}$ is available (and potentially for its velocity $\dot{\boldsymbol{\sigma}}_{x,d}$ and acceleration $\ddot{\boldsymbol{\sigma}}_{x,d}$).

3.4.1 Safety Task: Joint Position and Velocity Limits

In addition to the robot kinematic model introduced in Section 3.2, it is worth highlighting that all joints of the system are subject to physical limits in both position and velocity. These kinematic constraints can be expressed as follows:

$$\begin{cases} \underline{q}_i \leq q_i \leq \bar{q}_i, & i = 1, \cdots, n \\ \underline{\dot{q}}_i \leq \dot{q}_i \leq \bar{\dot{q}}_i, & i = 1, \cdots, n, \end{cases} \tag{3.12}$$

where $\underline{q}_i$ ($\underline{\dot{q}}_i$) denotes the minimum position (velocity) that the i-th joint can achieve, whereas $\bar{q}_i$ ($\bar{\dot{q}}_i$) denotes the corresponding maximum value. Since these bounds are imposed by the electromechanical structure of the robot and must therefore be satisfied, this task has to be assigned the highest priority in the HQP formulation. However, to be included in the HQP formulation at the desired priority level, as in Eq. (2.23), these constraints must first be appropriately reformulated and expressed as functions of the joint velocities.

Concerning the joint position limits, it is necessary to exploit the CBF framework detailed in Section 2.4 to formulate a constraint that involves the joint velocities. By defining the task function for the i-th joint position as $\sigma_{\text{jp},i} = q_i$, with minimum and maximum admissible values $\underline{\sigma}_{\text{jp},i} = \underline{q}_i$ and $\bar{\sigma}_{\text{jp},i} = \bar{q}_i$, the following functions can be formulated:

$$\begin{cases} \underline{h}_{\text{jp},i}(q_i) = \sigma_{\text{jp},i} - \underline{\sigma}_{\text{jp},i} & \geq 0, \quad i = 1, \cdots, n \\ \bar{h}_{\text{jp},i}(q_i) = \bar{\sigma}_{\text{jp},i} - \sigma_{\text{jp},i} & \geq 0, \quad i = 1, \cdots, n. \end{cases} \tag{3.13}$$

Based on Eq. (2.19), the constraint ensuring the satisfaction of the joint position limits can be formulated as:

$$\begin{cases} \dot{q}_i \geq & -\underline{\phi}_{\text{jp},i}\ \underline{h}_{\text{jp},i}(q_i) \quad i = 1, \cdots, n \\ -\dot{q}_i \geq & -\bar{\phi}_{\text{jp},i}\ \bar{h}_{\text{jp},i}(q_i) \quad i = 1, \cdots, n. \end{cases} \tag{3.14}$$

where $\bar{\phi}_{\text{jp},i}$ and $\underline{\phi}_{\text{jp},i}$ are positive scalar gains.

By gathering the functions $\underline{h}_{\text{jp},i}(q_i)$ and $\bar{h}_{\text{jp},i}(q_i)$, defined in Eq. (3.13), the following two function vectors can be defined:

$$\begin{aligned} \underline{\boldsymbol{h}}_{\text{jp}} &= [\underline{h}_{\text{jp},1},\ \underline{h}_{\text{jp},2} \cdots, \underline{h}_{\text{jp},n}]^T, \\ \bar{\boldsymbol{h}}_{\text{jp}} &= [\bar{h}_{\text{jp},1},\ \bar{h}_{\text{jp},2} \cdots, \bar{h}_{\text{jp},n}]^T. \end{aligned}$$

Similarly, by gathering the coefficients $\underline{\phi}_{\text{jp},i}$ and $\bar{\phi}_{\text{jp},i}$, defined in Eq. (3.14), the following two diagonal coefficient matrices can be defined:

$$\begin{aligned} \underline{\boldsymbol{\Phi}}_{\text{jp}} &= \text{diag}\{\underline{\phi}_{\text{jp},1}, \underline{\phi}_{\text{jp},2}, \cdots, \underline{\phi}_{\text{jp},n}\} \\ \bar{\boldsymbol{\Phi}}_{\text{jp}} &= \text{diag}\{\bar{\phi}_{\text{jp},1}, \bar{\phi}_{\text{jp},2}, \cdots, \bar{\phi}_{\text{jp},n}\}. \end{aligned}$$

It is straightforward to verify that Eq. (3.14) can be expressed as:

$$\underline{\boldsymbol{b}}_{\text{jp}} = \underline{\boldsymbol{\Phi}}_{\text{jp}}\ \underline{\boldsymbol{h}}_{\text{jp}}(\boldsymbol{q}) \leq \boldsymbol{J}_{\text{jp}}\dot{\boldsymbol{q}} \leq \bar{\boldsymbol{\Phi}}_{\text{jp}}\ \bar{\boldsymbol{h}}_{\text{jp}}(\boldsymbol{q}) = \bar{\boldsymbol{b}}_{\text{jp}}\ , \tag{3.15}$$

with $\boldsymbol{J}_{\text{jp}} = \boldsymbol{I}_n$, where $\boldsymbol{I}_n$ denotes the $(n \times n)$ identity matrix.

Regarding the joint velocity limits, they can be straightforwardly reformulated. In fact, referring to Eq. (2.23) and setting $\underline{\boldsymbol{b}}_{\text{jv}} = \underline{\dot{\boldsymbol{q}}}$ and $\bar{\boldsymbol{b}}_{\text{jv}} = \bar{\dot{\boldsymbol{q}}}$, the constraint to be incorporated in the HQP formulation can be formulated as:

$$\underline{\boldsymbol{b}}_{\text{jv}} \leq \boldsymbol{J}_{\text{jv}}\, \dot{\boldsymbol{q}} \leq \bar{\boldsymbol{b}}_{\text{jv}}\ , \tag{3.16}$$

with $\boldsymbol{J}_{\text{jv}} = \boldsymbol{I}_n$.

3.4.2 Safety Task: Virtual Walls

This task allows maintaining the distance between a point along the robot structure and a given virtual plane above a certain threshold, in order to avoid self-collisions and/or further constrain the workspace of the robot.

Given a point $\boldsymbol{p}_j(\boldsymbol{q})$ along the robot structure and three non-aligned points $\boldsymbol{p}^1$, $\boldsymbol{p}^2$, and $\boldsymbol{p}^3$ belonging to a generic virtual plane, the task function can be expressed as:

$$\sigma_{\text{vw,j}} = \hat{\boldsymbol{n}}^T(\boldsymbol{p}_j - \boldsymbol{p}^1), \tag{3.17}$$

where $\sigma_{\text{vw,j}}$ is a scalar task function that measures the distance between the point $\boldsymbol{p}_j$ and the virtual plane, while $\hat{\boldsymbol{n}}$ denotes the unit normal vector to the plane, which is defined as:

$$\hat{\boldsymbol{n}} = \frac{(\boldsymbol{p}^2 - \boldsymbol{p}^1) \times (\boldsymbol{p}^3 - \boldsymbol{p}^1)}{\|(\boldsymbol{p}^2 - \boldsymbol{p}^1) \times (\boldsymbol{p}^3 - \boldsymbol{p}^1)\|}.$$

Concerning the Jacobian matrix associated with this task, it is defined as:

$$\boldsymbol{J}_{\text{vw},j} = -\hat{\boldsymbol{n}}^T \boldsymbol{J}_{p,j},$$

where $\boldsymbol{J}_{p,j}$ denotes the positional Jacobian associated with the point $\boldsymbol{p}_j$, i.e., the Jacobian relating the joint velocities with the linear velocities

of point $\boldsymbol{p}_j$ belonging to the robot structure. Denoting the minimum admissible distance between the point $\boldsymbol{p}_j$ and the virtual wall as $\underline{\sigma}_{\text{vw},j}$, the following CBF can be formulated:

$$\underline{h}_{\text{vw},j} = \sigma_{\text{vw},j} - \underline{\sigma}_{\text{vw},j} \ , \tag{3.18}$$

where $\underline{h}_{\text{vw},j}$ represents the barrier function enforcing the maximum admissible distance from the virtual wall for point $\boldsymbol{p}_j$.

The constraint needed to keep the distance above the minimum threshold is:

$$\boldsymbol{J}_{\text{vw},j} \ \dot{\boldsymbol{q}} \geq -\underline{\phi}_{\text{vw},j} \ \underline{h}_{\text{vw},j}(\boldsymbol{q}) \ , \tag{3.19}$$

with $\underline{\phi}_{\text{vw},j}$ being a positive scalar gain.

When the number of points along the serial chain to be kept at a minimum distance from the same virtual plane is L, with $L > 1$, the constraints can be grouped by stacking the Jacobian matrices $\boldsymbol{J}_{\text{vw},j}$ as:

$$\boldsymbol{J}_{\text{vw}} = \begin{bmatrix} \boldsymbol{J}_{\text{vw},1}^T & \boldsymbol{J}_{\text{vw},2}^T & \cdots & \boldsymbol{J}_{\text{vw},L}^T \end{bmatrix}^T ,$$

and the CBFs as:

$$\underline{\boldsymbol{h}}_{\text{vw}}(\boldsymbol{q}) = \begin{bmatrix} \underline{h}_{\text{vw},1}, \underline{h}_{\text{vw},2}, \cdots, \underline{h}_{\text{vw},L} \end{bmatrix}^T \ .$$

By defining the matrix:

$$\underline{\boldsymbol{\Phi}}_{\text{vw}} = \text{diag}\{\underline{\phi}_{\text{vw},1}, \underline{\phi}_{\text{vw},2}, \cdots, \underline{\phi}_{\text{vw},L}\} \ ,$$

the overall constraint can be expressed as:

$$\boldsymbol{J}_{\text{vw}} \ \dot{\boldsymbol{q}} \geq -\underline{\boldsymbol{\Phi}}_{\text{vw}} \ \underline{\boldsymbol{h}}_{\text{vw}}(\boldsymbol{q}) = \underline{\boldsymbol{b}}_{\text{vw}} \ , \tag{3.20}$$

and included in the HQP framework as in Eq. (2.23).

3.4.3 Safety Task: Self-Collision Avoidance

This task allows avoiding collisions between two generic points of the robot, e.g., between a point $\boldsymbol{p}_j(\boldsymbol{q})$ belonging to one arm and any point $\boldsymbol{p}_l(\boldsymbol{q})$ belonging to any other part of the robot. Therefore, the control objective is to keep the distance between the points $\boldsymbol{p}_j(\boldsymbol{q})$ and $\boldsymbol{p}_l(\boldsymbol{q})$ above a certain threshold. The corresponding task function, in this case, can be expressed as:

$$\sigma_{\mathrm{sc},j,l} = \|\boldsymbol{p}_j - \boldsymbol{p}_l\| \,. \tag{3.21}$$

Concerning the Jacobian matrix associated with this task, it is defined as:

$$\boldsymbol{J}_{\mathrm{sc},j,l} = -\hat{\boldsymbol{n}}_{j,l}^T \left(\boldsymbol{J}_{p,j} - \boldsymbol{J}_{p,l}\right)$$

where $\hat{\boldsymbol{n}}_{j,l}^T$ is defined as:

$$\hat{\boldsymbol{n}}_{j,l}^T = \frac{\boldsymbol{p}_j - \boldsymbol{p}_l}{\|\boldsymbol{p}_j - \boldsymbol{p}_l\|} \,,$$

and represents the unit vector connecting the two points, while $\boldsymbol{J}_{p,j}$ and $\boldsymbol{J}_{p,l}$ represent the positional Jacobians of the points $\boldsymbol{p}_j$ and $\boldsymbol{p}_l$, respectively. Denoting the minimum admissible distance between the points $\boldsymbol{p}_j(\boldsymbol{q})$ and $\boldsymbol{p}_l(\boldsymbol{q})$ as $\underline{\sigma}_{\mathrm{sc},j,l}$, the following CBF can be defined:

$$\underline{h}_{\mathrm{sc},j,l} = \underline{\sigma}_{\mathrm{sc},j,l} - \sigma_{\mathrm{sc},j,l} \,. \tag{3.22}$$

The corresponding constraint, ensuring that the distance remains above the minimum threshold, can be formulated as:

$$\boldsymbol{J}_{\mathrm{sc},j,l}\, \dot{\boldsymbol{q}} \geq -\underline{\phi}_{\mathrm{sc},j,l}\, \underline{h}_{\mathrm{sc},j,l} \,, \tag{3.23}$$

with $\underline{\phi}_{\mathrm{sc},j,l}$ being a positive scalar gain.

In case there are n_c constraints related to the same point $\boldsymbol{p}_j$, they can be grouped by stacking the Jacobian matrices $\boldsymbol{J}_{\mathrm{sc},j,i}$ as:

$$\boldsymbol{J}_{\text{sc},j} = \begin{bmatrix} \boldsymbol{J}_{\text{sc},j,1}^T & \boldsymbol{J}_{\text{sc},j,2}^T & \cdots & \boldsymbol{J}_{\text{sc},j,n_c}^T \end{bmatrix}^T ,$$

the CBFs as:

$$\underline{\boldsymbol{h}}_{\text{sc},j} = \begin{bmatrix} \underline{h}_{\text{sc},j,1}, \underline{h}_{\text{sc},j,2}, \cdots, \underline{h}_{\text{sc},j,n_c} \end{bmatrix}^T ,$$

and by defining the matrix:

$$\underline{\boldsymbol{\Phi}}_{\text{sc,j}} = \text{diag}\{\underline{\phi}_{\text{sc},j,1}, \underline{\phi}_{\text{sc},j,2}, \cdots, \underline{\phi}_{\text{sc},j,n_c}\} .$$

Similarly, if there are M constraints related to other points of the serial chain, piling up the corresponding overall task Jacobian matrices as:

$$\boldsymbol{J}_{\text{sc}} = \begin{bmatrix} \boldsymbol{J}_{\text{sc},1}^T & \boldsymbol{J}_{\text{sc},2}^T & \cdots & \boldsymbol{J}_{\text{sc},M}^T \end{bmatrix}^T ,$$

the CBFs as:

$$\underline{\boldsymbol{h}}_{\text{sc}} = \begin{bmatrix} \underline{\boldsymbol{h}}_{\text{sc},1}^T & \underline{\boldsymbol{h}}_{\text{sc},2}^T & \cdots & \underline{\boldsymbol{h}}_{\text{sc},M}^T \end{bmatrix}^T ,$$

and the gain matrices as:

$$\underline{\boldsymbol{\Phi}}_{\text{sc}} = \text{diag}\{\underline{\boldsymbol{\Phi}}_{\text{sc},1}, \underline{\boldsymbol{\Phi}}_{\text{sc},2}, \cdots, \underline{\boldsymbol{\Phi}}_{\text{sc},M}\} ,$$

all the constraints can be expressed in a more compact form and incorporated within the HQP framework in Eq. (2.23), as:

$$\boldsymbol{J}_{\text{sc}}\, \dot{\boldsymbol{q}} \geq -\underline{\boldsymbol{\Phi}}_{\text{sc}}\, \underline{\boldsymbol{h}}_{\text{sc}}(\boldsymbol{q}) = \underline{\boldsymbol{b}}_{\text{sc}} . \tag{3.24}$$

3.4.4 Operational task: Cooperative and Uncooperative Motion

In order to enhance system flexibility in performing the required operations, two kinds of control tasks have been designed.

The first, referred to as *uncooperative motion*, aims at controlling the pose (i.e., position and orientation) of the two end-effectors independently. The corresponding Jacobian is represented by the $\boldsymbol{J}_\gamma(\boldsymbol{q})$ matrix defined in Eq. (3.7), which establishes the relationship between the linear and angular velocities of the two end-effectors and the system velocity. The task value is the vector obtained by stacking the position and orientation vectors of the two end-effectors:

$$\boldsymbol{\sigma}_\gamma = \begin{bmatrix} \boldsymbol{p}_l^T & \boldsymbol{o}_l^T & \boldsymbol{p}_r^T & \boldsymbol{o}_r^T \end{bmatrix}^T, \tag{3.25}$$

where $\boldsymbol{o}_l$ and $\boldsymbol{o}_r$ denote the unit quaternions expressing the orientation of the left and right end-effectors, respectively.

In contrast, the second task, referred to as *cooperative motion*, is designed to achieve a cooperative control of the two end-effectors through the regulation of the poses of the *absolute* and *relative* frames [63, 64]. The associated task variable is defined as:

$$\boldsymbol{\sigma}_{a,r} = \begin{bmatrix} \boldsymbol{p}_a^T & \boldsymbol{o}_a^T & \boldsymbol{p}_r^T & \boldsymbol{o}_r^T \end{bmatrix}^T, \tag{3.26}$$

where $\boldsymbol{p}_a$ an $\boldsymbol{p}_r$ denote, respectively, the absolute and relative position vectors, whereas $\boldsymbol{o}_a$ and $\boldsymbol{o}_r$ represent the quaternions corresponding to the absolute and relative rotation matrices, $\boldsymbol{R}_a$ and $\boldsymbol{R}_r$. The absolute position and orientation are defined as:

$$\boldsymbol{p}_a = \alpha \boldsymbol{p}_l + (1-\alpha)\boldsymbol{p}_r, \tag{3.27}$$

$$\boldsymbol{R}_a = \boldsymbol{R}_l \boldsymbol{R}_{l,r}^l(\boldsymbol{r}_{l,r}^l, (1-\alpha)\vartheta_{l,r}), \tag{3.28}$$

where $\boldsymbol{r}_{l,r}^l$ and $\vartheta_{l,r}$ denote the axis-angle representation of the orientation matrix $\boldsymbol{R}_{l,r}^l$, describing the orientation of the right end-effector frame with respect to the left one in the left end-effector frame. As previously mentioned, $\boldsymbol{p}_a$ represents the position vector of the absolute frame, i.e., the reference frame located between the two end-effectors. According to Eq. (3.27), this frame coincides with the left end-effector frame when

$\alpha = 0$, with the right end-effector frame when $\alpha = 1$, and with the frame centered at the midpoint between the two end-effectors when $\alpha = 0.5$. Conversely, the relative position and orientation are defined as:

$$\boldsymbol{p}_r = \boldsymbol{p}_r - \boldsymbol{p}_l, \tag{3.29}$$

$$\boldsymbol{R}_r = \boldsymbol{R}_l^T(\boldsymbol{r}_l, \vartheta_l)\boldsymbol{R}_r, \tag{3.30}$$

and can be regarded as the position and orientation of the right end-effector frame expressed with respect to the left end-effector frame. Figure 3.3 provides a graphical representation of the absolute and relative frames for different values of α.

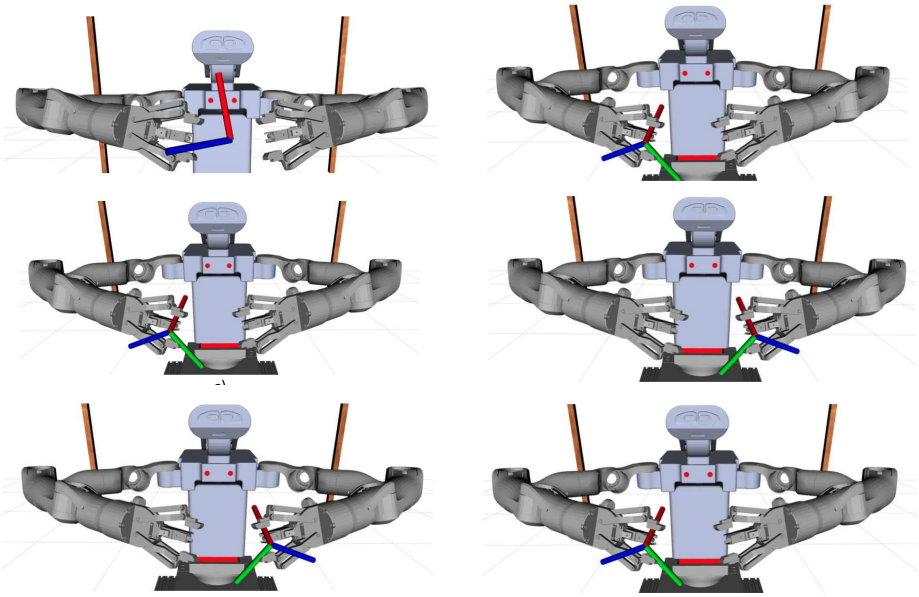


Figure 3.3: Absolute and Relative frames representation. top-left) Graphical representation of the absolute frame ($\alpha = 0.5$); top-right) Graphical representation of the relative frame ($\alpha = 0.5$); mid-left) Graphical representation of the absolute frame ($\alpha = 0$); mid-right) Graphical representation of the relative frame ($\alpha = 0$); bottom-left) Graphical representation of the absolute frame ($\alpha = 1$); bottom-right) Graphical representation of the relative frame ($\alpha = 1$).

It should be emphasized that the two previously introduced operational tasks can also be represented as a single operational task. In particular, by appropriately embedding the mathematical definition of *uncooperative* motion into that of *cooperative* motion, the following formulation can be derived:

$$\boldsymbol{\sigma}_{a,r} = \begin{bmatrix} \boldsymbol{\sigma}_a \\ \boldsymbol{\sigma}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_{a_p} \\ \boldsymbol{\sigma}_{a_o} \\ \boldsymbol{\sigma}_{r_p} \\ \boldsymbol{\sigma}_{r_o} \end{bmatrix} = \begin{bmatrix} \alpha \boldsymbol{p}_l + (1-\alpha)\boldsymbol{p}_r \\ \boldsymbol{o}_a \\ \boldsymbol{p}_r - \beta \boldsymbol{p}_l \\ \boldsymbol{o}_r \end{bmatrix}, \tag{3.31}$$

which identifies the uncooperative motion as a special case of the cooperative one. Concerning the variables α and β, the first is defined within the interval $[0, 1]$ and serves as a balance parameter quantifying the level of symmetry in the arms' cooperative motion, while the latter is a binary coordination parameter that regulates the degree of coordination between the two arms. Specifically, for $\beta = 1$, the formulation in Eq. (3.31) reduces to that in Eq. (3.26), which corresponds to a cooperative motion. In this setting, the parameter $\alpha \in [0, 1]$ regulates the degree of symmetry between the two end-effectors. In particular, $\alpha = 0$ and $\alpha = 1$ give rise to leader–follower schemes (i.e., completely asymmetric behaviors), with the right end-effector leading when $\alpha = 0$ and the left end-effector leading when $\alpha = 1$, whereas $\alpha = 0.5$ results in a fully symmetric scheme. In contrast, when $\beta = 0$, the formulation in Eq. (3.31) reduces to that in Eq. (3.25), which corresponds to an uncooperative motion. In this case, $\alpha = 0$, and the arms result in mutually independent movements. It is worth emphasizing that the parameters α and β may vary over time, implying that both the symmetry level and the coordination degree may change over time.

Regarding the task velocity, it depends on the collective end-effector velocity vector introduced in Eq. (3.7) and can be derived as follows [65]:

$$\dot{\boldsymbol{\sigma}}_{a,r} = \begin{bmatrix} \alpha \boldsymbol{I}_6 & (1-\alpha)\boldsymbol{I}_6 \\ -\beta \boldsymbol{I}_6 & \boldsymbol{I}_6 \end{bmatrix} \boldsymbol{J}_\gamma(\boldsymbol{q})\, \dot{\boldsymbol{q}} = \boldsymbol{J}_{a,r}\, \boldsymbol{J}_\gamma(\boldsymbol{q})\, \dot{\boldsymbol{q}} = \boldsymbol{J}_\sigma(\boldsymbol{q})\, \dot{\boldsymbol{q}}, \tag{3.32}$$

where $\boldsymbol{J}_\sigma(\boldsymbol{q}) = \boldsymbol{J}_{a,r}\, \boldsymbol{J}_\gamma(\boldsymbol{q}) \in \mathbb{R}^{12 \times n}$ represents the task Jacobian matrix,

which can be employed to formalize a wide range of multi-robot tasks, such as cooperative transportation [66].

As regards the constraint imposed by this task to be incorporated into the HQP formulation, it can be expressed as follows:

$$\boldsymbol{J}_{\sigma}(\boldsymbol{q})\ \dot{\boldsymbol{q}} = \boldsymbol{b}_{\sigma}, \tag{3.33}$$

with

$$\boldsymbol{b}_{\sigma} = \dot{\boldsymbol{\sigma}}_{(\cdot),d} + \boldsymbol{K}_{\sigma}\tilde{\boldsymbol{\sigma}}_{(\cdot)}, \tag{3.34}$$

where $\dot{\boldsymbol{\sigma}}_d$ denotes the desired operational task velocity, $\boldsymbol{K}_{\sigma}$ is a positive-definite matrix of gains, and $\tilde{\boldsymbol{\sigma}}$ represents the task error.

3.4.5 Optimization Task: Swivel Angle

This task aims at controlling the so-called *swivel angle* of the manipulators, defined as the angle formed between the plane passing through the base, shoulder, and wrist of a manipulator, and the plane passing through the shoulder, elbow, and wrist [67]. The rationale behind the implementation of this task is related to what is outlined in Sections 1.1 and 3.1, i.e., robots are increasingly required to operate in collaboration with humans. Thus, regulating this angle enables robotic arms to adopt configurations resembling those of human arms, thereby improving the acceptability of the system and enhancing the predictability of their movements. Figure 3.4 provides a graphical representation of the swivel angle associated with the right manipulator of the system depicted in Figure 3.1.

Let $\boldsymbol{p}_{i,e}$, $\boldsymbol{p}_{i,w}$, and $\boldsymbol{p}_{i,s}$ denote, respectively, the position vectors of the elbow, wrist, and shoulder associated with side i. Considering these vectors together with the following ones:

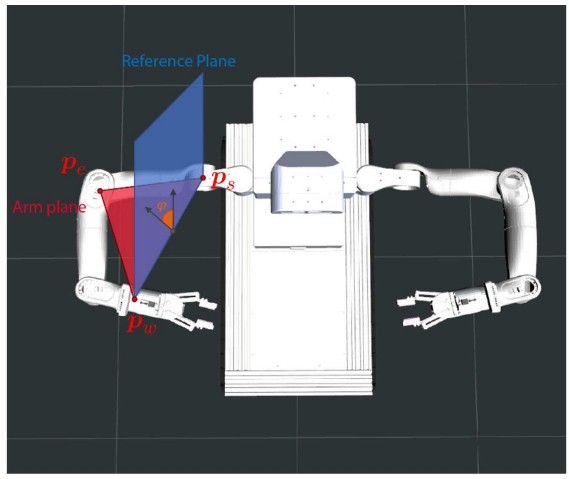


Figure 3.4: Graphical representation of the swivel angle φ_r for the right arm.

$$\boldsymbol{a}_i = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T, \tag{3.35}$$
$$\boldsymbol{b}_i = \boldsymbol{p}_{i,s} - \boldsymbol{p}_{i,e}, \tag{3.36}$$
$$\boldsymbol{c}_i = \boldsymbol{p}_{i,s} - \boldsymbol{p}_{i,w}, \tag{3.37}$$
$$\boldsymbol{d}_i = \boldsymbol{p}_{i,e} - \boldsymbol{p}_{i,w}, \tag{3.38}$$

with $i \in \{l, r\}$, the swivel angle associated with arm i can be obtained through the following relationship:

$$\varphi_i = \operatorname{sgn}(\boldsymbol{a}_i \times \boldsymbol{b}_i \cdot \boldsymbol{c}_i) \arccos\left(\frac{(\boldsymbol{a}_i \times \boldsymbol{c}_i)(\boldsymbol{b}_i \times \boldsymbol{d}_i)}{\|\boldsymbol{a}_i \times \boldsymbol{c}_i\| \, \|\boldsymbol{b}_i \times \boldsymbol{d}_i\|} \right). \tag{3.39}$$

In this case, the desired task velocity vector, to be included as an equality constraint in the HQP formulation, can be expressed as follows:

$$\boldsymbol{b}_{\text{sw}} = \boldsymbol{k}_{\text{sw}} \begin{bmatrix} \varphi_{l,d} - \varphi_l \\ \varphi_{r,d} - \varphi_r \end{bmatrix}, \tag{3.40}$$

where $\varphi_{i,d}$ denotes the desired swivel angle for arm i, while $\boldsymbol{k}_{\text{sw}}$ denotes the gain vector.

As regards the constraint imposed by this task to be incorporated into the HQP formulation, it can be expressed as follows:

$$\boldsymbol{J}_{\text{sw}}\dot{\boldsymbol{q}} = \boldsymbol{b}_{\text{sw}}, \tag{3.41}$$

where $\boldsymbol{J}_{\text{sw}}$ denotes the task Jacobian matrix, defined as:

$$\boldsymbol{J}_{\text{sw}} = \begin{bmatrix} \frac{\partial \varphi_l}{\partial \boldsymbol{q}} \\ \\ \frac{\partial \varphi_r}{\partial \boldsymbol{q}} \end{bmatrix}. \tag{3.42}$$

3.5 Validation

The control framework presented in this chapter has been extensively validated in both controlled laboratory and real-world conditions. The validation was conducted employing the robotic platform illustrated in Figure 3.1, a redundant mobile manipulator composed of a tracked mobile base and a dual-arm manipulator system. The mobile base is an Alitrak DCT-450P[2] equipped with an Nvidia Jetson Orin, which is a compact energy-efficient AI supercomputer belonging to the Nvidia Jetson Oring family, an Intel Next Unit of Computing (NUC), and several navigation sensors, in particular, two Ouster OS1-gen64 3D-Lidars, an Elipse-E Inertial Measurement Unit (IMU), and a Septentrio Real Time Kinematic dual antenna Global Positioning System (GPS-RTK). The dual-arm manipulator is installed on the top-tail part of the tracked mobile base, which is provided with unicycle kinematics ($n_b = 2$), and it is a bimanual robotic system completely designed and developed by PAL-Robotics[3]. It consists of a 2-DoF torso ($n_t = 2$) and two 7-DoF manipulators ($n_a = 7$). Regarding the torso, it includes a prismatic and a revolute joint that provides it with the capability of lifting and rotating. Moreover, it is featured with a speaker and a microphone aimed at promoting and facilitating human-robot collaboration and interaction. Along with the two manipulators, there is a head attached to the torso. It includes two joints, which provide it with the capability of performing pan and tilt movements, and an Intel

[2] www.alitrak.com/products/dct-450

[3] https://pal-robotics.com/

RealSense D435 RGB-D sensor. The arms, instead, are both equipped with seven revolute joints and are featured with a Force-Torque (FT) sensor at the wrist and an Intel RealSense D435 RGB-D sensor, which is mounted on a plastic support fixed up to the wrist next to the FT-sensor. Moreover, both manipulators are equipped with an end-effector tool consisting of a cutter and a two-finger gripper, which aims to provide the robot with the capability to achieve harvesting activities. Tables 3.1 and 3.2 report the robot kinematic constraints, expressed in terms of maximum and minimum positions and velocities, and the adopted values for each parameter and gain, respectively.

JOINT LIMITS
$\bar{\boldsymbol{q}}_b = [\infty,\ \infty,\ 3.14]^T$ $\underline{\boldsymbol{q}}_b = [-\infty,\ -\infty,\ -3.14]^T$
$\bar{\boldsymbol{q}}_t = [3.00,\ 0.30]^T$ $\underline{\boldsymbol{q}}_t = [-1.60,\ 0.0]^T$
$\bar{\boldsymbol{q}}_l = [1.44,\ 1.47,\ 2.41,\ 1.49,\ 2.42,\ 2.03,\ 1.28]^T$ $\underline{\boldsymbol{q}}_l = [-0.73,\ -1.45,\ -2.41,\ -2.30,\ -2.42,\ -2.03,\ -3.48]^T$
$\bar{\boldsymbol{q}}_r = [1.44,\ 1.47,\ 2.41,\ 1.49,\ 2.42,\ 2.03,\ 1.28]^T$ $\underline{\boldsymbol{q}}_r = [-0.73,\ -1.45,\ -2.41,\ -2.30,\ -2.42,\ -2.03,\ -3.48]^T$
$\bar{\dot{\boldsymbol{q}}}_t = [0.7,\ 0.2]^T$ $\underline{\dot{\boldsymbol{q}}}_t = [-0.7,\ -0.2]^T$
$\bar{\dot{\boldsymbol{q}}}_l = [1.95,\ 1.95,\ 1.95,\ 1.95,\ 1.95,\ 1.95,\ 1.95]^T$ $\underline{\dot{\boldsymbol{q}}}_l = [-1.95,\ -1.95,\ -1.95,\ -1.95,\ -1.95,\ -1.95,\ -1.95]^T$
$\bar{\dot{\boldsymbol{q}}}_r = [1.95,\ 1.95,\ 1.95,\ 1.95,\ 1.95,\ 1.95,\ 1.95]^T$ $\underline{\dot{\boldsymbol{q}}}_r = [-1.95,\ -1.95,\ -1.95,\ -1.95,\ -1.95,\ -1.95,\ -1.95]^T$

Table 3.1: Farming robot joint position limits (in [m] and [rad]) and joint velocity limits (in [m/s] and [rad/s]).

PARAMETER	VALUE	EQ.
$\underline{\phi}_{\text{jp},i}$,$\bar{\phi}_{\text{jp},i}$	10	Eq. (3.12)
$\underline{\phi}_{\text{vw},j}$	5	Eq. (3.19)
$\sigma_{\text{vw},j,i}$	$0.3m$	Eq. (3.18)
$\underline{\phi}_{\text{sc},j,i}$	10	Eq. (3.23)
$\sigma_{\text{sc},j,head}$	$0.5m$	Eq. (3.22)
$\sigma_{\text{sc},j,torso}$	$0.35m$	Eq. (3.22)
$\sigma_{\text{sc},j,arm}$	$0.2m$	Eq. (3.22)

Table 3.2: Value of the parameters used for the validation.

3.5.1 Laboratory Experimens

The following section reports the results of the extensive validation campaign conducted within a controlled laboratory environment. To offer a comprehensive overview of the validation of the proposed methodologies, the results are reported separately for the three task categories introduced in Section 3.4, namely safety, operational, and optimization. A video of the experiments is available at the following link[4].

3.5.1.1 Safety Tasks

In the following, the obtained results of the experiment conducted to validate the safety tasks described in Sections 3.4.1-3.4.3 are reported. The validation was conducted using a hierarchy with three priority levels and five tasks, namely four safety tasks and one operational task, organized as shown in Figure 3.5.

Figure 3.5: The task hierarchy employed to validate the safety tasks introduced in Sections 3.4.1-3.4.3.

For this purpose, a sinusoidal velocity profile was imposed to all joints through the introduction of an appropriate operational task within the task hierarchy. This was modeled as an equality constraint with the task Jacobian matrix $\boldsymbol{J} = \boldsymbol{I}$ and the desired task velocity vector $\boldsymbol{b}$ defined as a

[4] `https://www.youtube.com/watch?v=RbsWzz8vnMo`

vector of sinusoidal functions, each representing the desired velocity profile of a joint. Since the aim of this experiment was to validate the safety tasks proposed in Sections 3.4.1-3.4.3, the sinusoidal velocity profiles imposed to all joints were intentionally designed to violate all safety-task constraints with higher priority than the operational task, which was assigned to the lowest level of the hierarchy (i.e., the third level), and were computed according to the following equations:

$$\begin{cases} \boldsymbol{q}_d &= 3\sin\left(\frac{2\pi}{25}t\right)\begin{bmatrix}\mathbf{0}_{1x3}, & \mathbf{0}_{1x2}, & 1 & \dots & 1\end{bmatrix}^T \\ \dot{\boldsymbol{q}}_d &= 0.75\sin\left(\frac{2\pi}{25}t\right)\begin{bmatrix}\mathbf{0}_{1x3}, & \mathbf{0}_{1x2}, & 1 & \dots & 1\end{bmatrix}^T. \end{cases} \tag{3.43}$$

For this experiment, the joint position limits were chosen to match the actual actuator limits listed in Table 3.1, while the maximum and minimum joint velocities were intentionally restricted at $\bar{\dot{\boldsymbol{q}}} = 0.5$ rad/s and $\underline{\dot{\boldsymbol{q}}} = -0.5$ rad/s to prompt the activation of the associated constraint.

Concerning the virtual-wall task introduced in Section 3.4.2, it is a safety task typically employed to prevent collisions between the manipulators and the mobile base. Figure 3.6 provides a representation of the virtual wall introduced by this task through top-down and side views of the robot. In more detail, the blue surface over the mobile base represents the wall, while the blue spheres denote the manipulators' points constrained to remain at a distance from the wall greater than the specified one. For this experiment, the minimum safety distance was set at $\underline{\sigma}_{\mathrm{vw},j} = 0.5$m.

Regarding the self-collision avoidance task detailed in Section 3.4.3, it is, instead, a safety task aimed at preventing collisions between both the manipulators themselves and the manipulators and the other components of the bi-manual robotic system, including the torso and the head. For this experiment, three different thresholds were defined: $\underline{\sigma}_{\mathrm{sc},j,arm} = 0.2$ m for the manipulators, $\underline{\sigma}_{\mathrm{sc},j,head} = 0.5$ m for the head, and $\underline{\sigma}_{\mathrm{sc},j,torso} = 0.35$ m for the torso. Figure 3.7 provides a representation of points affected by this task through top-down and side views of the robot. In more detail, the red spheres and cylinders denote the components from which a safety distance must be maintained, while the blue spheres represent the components constrained to remain at a distance from the red ones greater than the specified one. It is worth noting that, for components

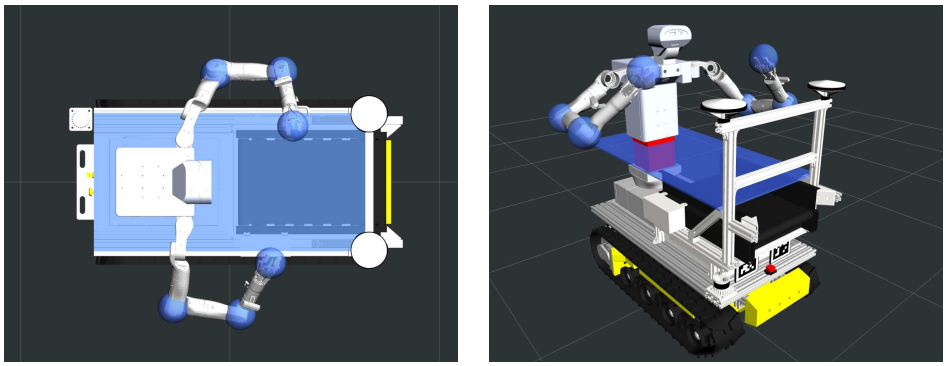


Figure 3.6: Top-down and side views of the virtual wall introduced by the safety task described in Section 3.4.2. The horizontal plane over the mobile base represents the wall, while the blue spheres denote the manipulators' points constrained to remain at a certain distance from the wall.

encapsulated in a red cylinder, compliance with the minimum-distance constraint is ensured by monitoring the position of the nearest blue sphere.

The implemented safety tasks introduced a total of 86 constraints within the HQP framework. Despite the high number of constraints involved, the control framework was successfully executed at 100 Hz, which represents the maximum control frequency supported by the robot actuators. Table 3.2 reports the values of each task parameter and gain.

Figure 3.8 reports the outcomes resulting from this experiment. In detail, Figures 3.8 top-left), 3.8 top-right), and 3.8 bottom-left) show the time evolution of the distances related to the self-collision avoidance and virtual-wall tasks (solid blue lines) with respect to the corresponding threshold values (dashed red lines). Figure 3.8 bottom-right), on the other hand, reports: in the top part, the time evolution of the normalized joint positions (solid lines) with respect to the corresponding normalized minimum and maximum threshold values (dashed red lines), and, in the bottom part, the time evolution of joint velocities (solid lines) with respect to their corresponding minimum and maximum threshold values (dashed red lines). Analyzing these figures, it can be observed that, throughout the entire experiment, both the joint position and velocity limits were successfully observed, and none of the distances related to the self-collision avoidance and virtual-wall tasks dropped below the corresponding threshold.

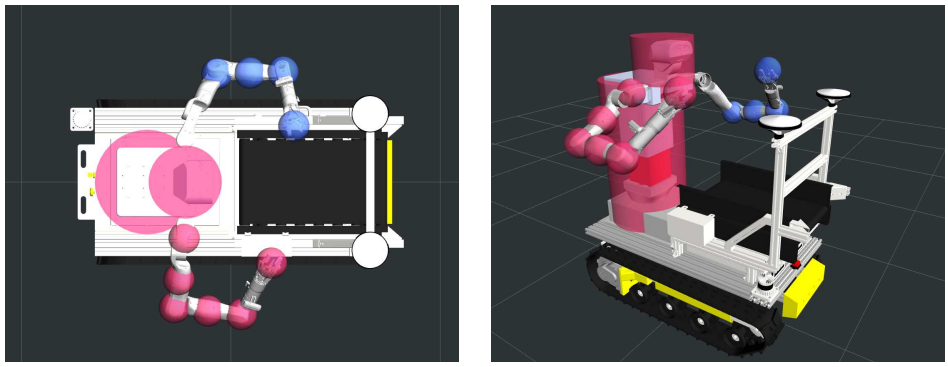


Figure 3.7: Top-down and side views of the self-collision tasks. The red spheres and cylinders denote points from which a safety distance must be maintained. The blue spheres, on the other hand, denote the components constrained to remain at a distance greater than the specified one.

3.5.1.2 Operational Tasks

In the following, the obtained results of the two experiments carried out to validate the operational tasks described in Section 3.4.4 are reported.

The first experiment was intended to validate the feasibility of controlling the manipulators in an uncooperative fashion. To this end, an uncooperative motion profile was designed for the two manipulators. In particular, the desired position of the left end-effector was set as:

$$\boldsymbol{p}_{l,d}(t) = \boldsymbol{p}_{l,i} + \begin{bmatrix} 0.1\left(1-\cos\left(\frac{\pi}{4}t\right)\right) \\ 0.2\left(1-\cos\left(\frac{\pi}{4}t\right)\right) \\ 0.05\left(1-\cos\left(\frac{\pi}{4}t\right)\right) \end{bmatrix}, \tag{3.44}$$

where $\boldsymbol{p}_{l,i}$ denotes the initial position of the left end-effector. The desired orientation, on the other hand, was set as follows:

$$\boldsymbol{R}_{l,d}(t) = \boldsymbol{R}_{l,i}\boldsymbol{R}(\phi(t), \theta(t), \psi(t)), \tag{3.45}$$

where $\boldsymbol{R}_{l,i}$ denotes the rotation matrix expressing the initial orientation of the left end-effector, while $\boldsymbol{R}(\phi(t), \theta(t), \psi(t))$ represents the rotation matrix derived from the following roll-pitch-yaw angles:

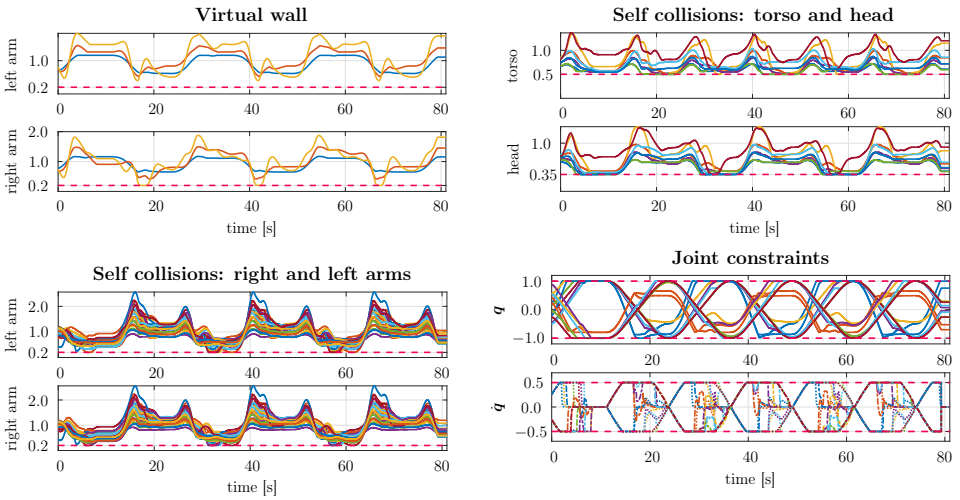


Figure 3.8: Safety tasks validation. Top-left) Distance of three points belonging to the arms from the virtual wall placed over the mobile base and imposed minimum threshold (red-dashed line). Top-right) Distance of three points belonging to the arms from the torso and the head of the robot, and imposed minimum threshold (red-dashed line). Bottom-left) Distance between three points of one of the arms with respect to three points of the other arm and imposed minimum threshold (red-dashed line). Bottom-right) Normalized joint positions for left (dotted lines) and right (dot-dashed lines) arms and velocity, and normalized imposed minimum and maximum thresholds (red-dashed lines).

$$\phi(t) = 0, \quad \theta(t) = -\frac{\pi}{8}\left(1 - \cos\left(\frac{\pi}{4}t\right)\right), \quad \psi(t) = 0. \tag{3.46}$$

As for the right arm, the desired end-effector position was set as:

$$\boldsymbol{p}_{r,d}(t) = \boldsymbol{p}_{r,i} + \begin{bmatrix} 0.1\cos\left(1 - \left(\frac{\pi}{4}t\right)\right) \\ 0.4\left(1 - \cos\left(\frac{\pi}{4}t\right)\right) \\ -0.05\left(1 - \cos\left(\frac{\pi}{4}t\right)\right) \end{bmatrix}, \tag{3.47}$$

where $\boldsymbol{p}_{r,i}$ denotes the initial position of the right end-effector. The desired orientation, on the other hand, was set as follows:

$$\boldsymbol{R}_{r,d}(t) = \boldsymbol{R}_{r,i}\boldsymbol{R}(\phi(t), \theta(t), \psi(t)), \tag{3.48}$$

where $\boldsymbol{R}_{r,i}$ denotes the rotation matrix expressing the initial orientation of the right end-effector, while $\boldsymbol{R}(\phi(t), \theta(t), \psi(t))$ represents the rotation matrix derived from the following roll-pitch-yaw angles:

$$\phi(t) = 0, \quad \theta(t) = 0, \quad \psi(t) = -\frac{\pi}{8}\left(1 - \cos\left(\frac{\pi}{4}t\right)\right). \tag{3.49}$$

The hierarchy considered for this experiment is the same as the one described in Section 3.5.1.1, but with the uncooperative motion task detailed in Section 3.4.4 in place of the random sinusoidal joint motion. The overall hierarchy is shown in Figure 3.9.

Figure 3.9: The task hierarchy used to validate the uncooperative motion operational task described in Section 3.4.4.

Figure 3.10 presents the results of this experiment. Specifically, it shows the temporal evolution of the position and orientation errors of the two end-effectors. The results demonstrate the effectiveness of both the proposed framework and the implemented task. In particular, they show that the end-effectors successfully tracked the desired trajectory in both position and orientation.

The second experiment, on the other hand, was intended to validate the feasibility of controlling the manipulators in a cooperative fashion. To this end, a cooperative motion profile was designed for the system in terms of absolute and relative pose, with the parameter α set to 0.5. In particular, the desired absolute position was set as

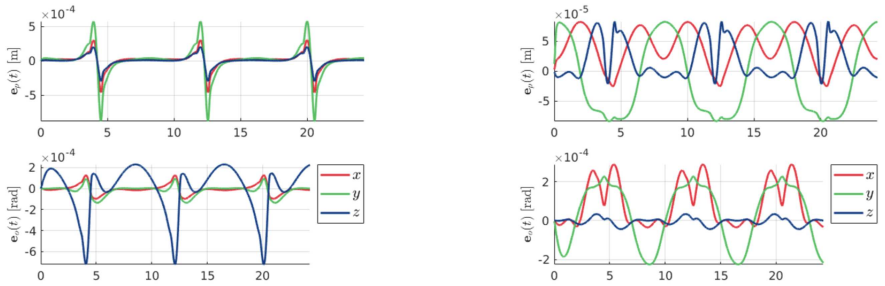


Figure 3.10: Uncooperative motion operational task validation. Top-left) Position error for the left arm. Bottom-left) Orientation error for the left arm. Top-right) Position error for the right arm. Bottom-right) Orientation error for the right arm.

$$\boldsymbol{p}_{a,d}(t) = \boldsymbol{p}_{a,i} + \begin{bmatrix} 0.05\left(1 - \cos\left(\frac{\pi}{2}t\right)\right) \\ 0.1\left(1 - \cos\left(\frac{\pi}{2}t\right)\right) \\ 0.0 \end{bmatrix}, \tag{3.50}$$

where $\boldsymbol{p}_{a,i}$ denotes the initial position of the absolute frame. The desired orientation, in contrast, was maintained constant at the initial value. As for the relative frame, the desired position was set as

$$\boldsymbol{p}_{r,d}(t) = \boldsymbol{p}_{r,i} + \begin{bmatrix} 0.1\left(1 - \cos\left(\frac{\pi}{2}t\right)\right) \\ 0.0 \\ 0.1\left(1 - \cos\left(\frac{\pi}{2}t\right)\right) \end{bmatrix}, \tag{3.51}$$

where $\boldsymbol{p}_{r,i}$ denotes the initial position of the relative frame. The desired orientation, on the other hand, was set as follows

$$\boldsymbol{R}_{r,d}(t) = \boldsymbol{R}_{r,i}\boldsymbol{R}(\phi(t), \theta(t), \psi(t)), \tag{3.52}$$

where $\boldsymbol{R}_{r,i}$ denotes the rotation matrix expressing the initial orientation of the relative frame, while $\boldsymbol{R}(\phi(t), \theta(t), \psi(t))$ represents the rotation matrix derived from the following roll-pitch-yaw angles

$$\phi(t) = 0, \quad \theta(t) = 0, \quad \psi(t) = -\frac{\pi}{8}\left(1 - \cos\left(\frac{\pi}{2}t\right)\right). \tag{3.53}$$

The hierarchy adopted in this experiment is the same as the previous one, but with the cooperative motion task detailed in Section 3.4.4 replacing the uncooperative one. The overall hierarchy is shown in Figure 3.11.

Figure 3.11: The task hierarchy used to validate the cooperative motion operational task described in Section 3.4.4.

Figure 3.12 presents the results of this experiment. Specifically, it shows the temporal evolution of the position and orientation errors of the absolute and relative frames. The results demonstrate, also in this case, the effectiveness of the proposed framework and the implemented task, in particular that the absolute and relative frames successfully tracked the desired trajectory in both position and orientation.

3.5.1.3 Optimization Task

In the following, the results of the experiment conducted to assess the effectiveness of the swivel angle task described in Section 3.4.5 are reported.

In this experiment, the manipulators were instructed to maintain their initial end-effector configurations, both in position and orientation, while a smooth reference trajectory for the swivel angles was imposed, evolving from the initial values to a final desired value of $\frac{\pi}{4}$ rad. The validation was conducted using a hierarchy with four priority levels and six tasks, namely four safety tasks, one operational task, and one optimization task, organized as shown in Figure 3.13.

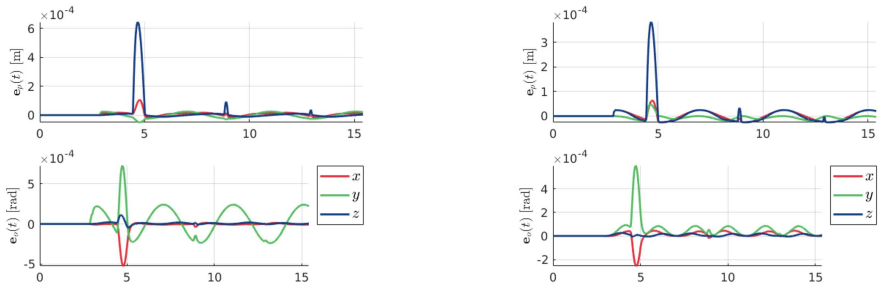


Figure 3.12: Cooperative motion operational task validation. Top-left) Position error for the absolute frame. Bottom-left) Orientation error for the absolute frame. Top-right) Position error for the relative frame. Bottom-right) Orientation error for the relative arm

Figure 3.14 presents the results of this experiment. In particular, it shows the temporal evolution of the swivel angle error for the two manipulators. The results show that the error converges to zero, demonstrating the capability of the proposed architecture to generate internal arm motions that enable the swivel angles to track the desired trajectory while preserving fixed end-effector configurations.

3.5.2 Real-World Experiments

This section presents the results of the validation campaign conducted under real-world conditions. Validation comprised a harvesting experiment conducted during the CANOPIES project's experimental campaigns. A video of the experiment is available at the following link[5].

Regarding the harvesting procedure, successful grape-bunch harvesting requires the robot to first detect nearby bunches and then precisely estimate the 3D position of the peduncle to be cut. This is accomplished in real time through perception software developed by one of the project partners, which makes use of data from the RGB-D camera (Figure 3.15) [68, 69].

Once the collected data have been analyzed and the peduncle position

[5] www.youtube.com/watch?v=V2dF5SXiKKE

Figure 3.13: The task hierarchy used to validate the Swivel angle task described in Section 3.4.5.

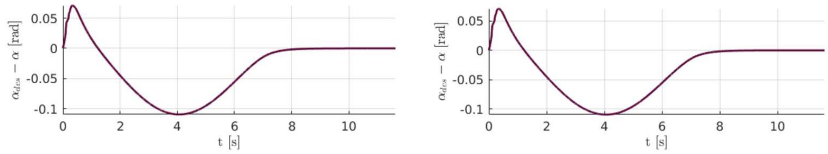


Figure 3.14: Optimization task validation. Left) Swivel angle error for the left arm. Right) Swivel angle error for the right arm.

has been estimated, the harvesting procedure begins. Specifically, a dedicated software module generates the desired end-effector position and orientation trajectories, using trapezoidal velocity profiles to connect a sequence of waypoints. This enables the robot to: *i*) move to a pre-grasp position (see Figure 3.16 top-right), i.e., a location at a predefined distance from the peduncle; *ii*) proceed to the grasp position, i.e., move onto the peduncle; *iii*) close the gripper to secure the bunch; *iv*) actuate the scissors to cut the peduncle; *v*) move to a pre-release position (see Figure 3.16 bottom-left), i.e., at a predefined distance from the box where the grapes must be placed; and *vi*) deposit the bunch into the designated container (see Figure 3.16 bottom-right). Once the grape has been

Figure 3.15: Output of the perception system. The highlighted zones represent the grapes (yellow zones) and the peduncles (light-blue zones) that the software has detected, with a confidence score above 0.9.

successfully released, the robot returns the tool to its initial configuration (see Figure 3.16 top-left). It is worth noticing that the success of this procedure strongly depends on the accuracy of the peduncle position estimation, which is particularly challenging to achieve due to the variability in distances between the cameras and the grape bunches, as well as the potential occlusions caused by leaves or other bunches. Moreover, it is also worth noticing that the harvesting procedure described above pertains to the case where the robot's end-effector is equipped with both scissors for cutting the peduncle and fingers for grasping the grape.

Figure 3.17 top) shows the time evolution of the actual (solid blue line) and desired (dashed red line) position of the tool involved in the harvesting procedure, whereas Figure 3.17 bottom) reports the time evolution of the normalized positions and velocities of the torso and right arm joints (solid lines) with respect to their normalized limit values (dashed red lines).

3.6 Conclusions

This chapter introduces a control architecture tailored to address the challenges of managing robotic platforms with complex kinematic structures in dynamic and unstructured environments, such as those encountered in agricultural domains. The architecture presented in this chapter combines the HQP framework with the CBFs methodology, thereby enabling

Figure 3.16: Top-left) Image of the robot with both harvesting tools in the home configuration. Top-right: image of the robot with the right harvesting tool in the pre-grasp position. Bottom-left) Image of the robot with the right harvesting tool in the pre-release position. Bottom-right) Image of the robot with the right harvesting tool in the release position.

systems such as bi-manual mobile robots to autonomously manage safety and internal configurations while performing operational tasks. Extensive experimental validation, conducted in both laboratory and real-field settings within the EU-funded CANOPIES project, corroborates the effectiveness and adaptability of the proposed solution.

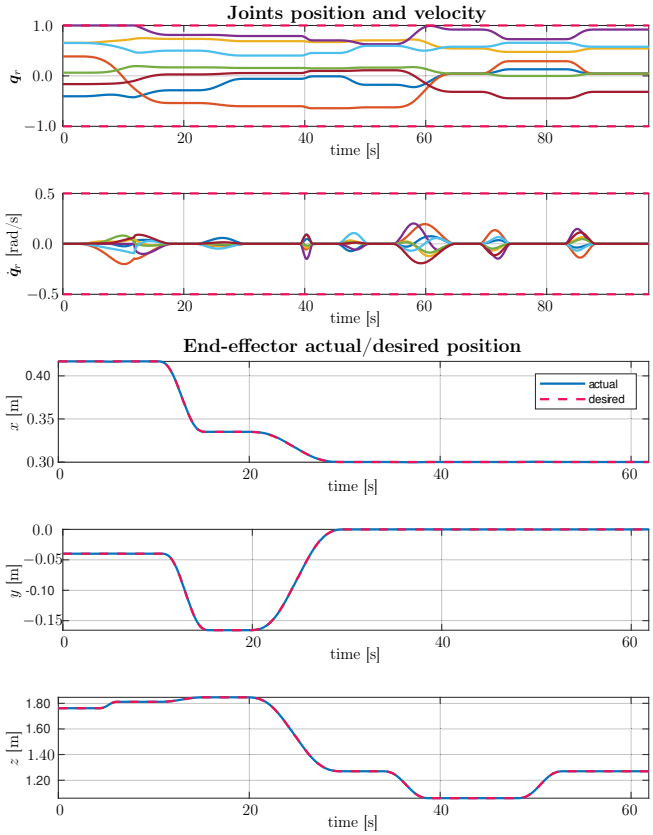


Figure 3.17: Top) Time evolution of normalized joint positions (top) and velocities (bottom) of the right arm (solid lines), shown with respect to their normalized limit values (dashed red lines). Bottom) x, y, and z coordinates over time of the actual (solid blue line) and desired (dashed red line) position of the right end-effector.

Chapter 4

Adaptive Shared Control for Human–Robot Collaboration within a Human-Safety-Oriented Framework

4.1 Introduction

As anticipated in Section 1.1, in recent years, an increasing number of sectors have had to deal with the challenge of addressing growing global demand for goods and services while simultaneously reducing resource consumption and mitigating environmental impact. As a result of this situation and recent technological advancements, robots are becoming increasingly prevalent in humans' daily lives, appearing in contexts ranging from factories [5] to operating rooms [6], from agricultural fields [7] to homes [9], where they are more and more often employed to perform tasks classified as dangerous and/or tedious for humans. However, as robots become more integrated into society, an important question arises: how can we share workspaces with robots in a manner that is both safe and low-invasive for everyone involved?

Regarding safety, current collaborative robots are inherently equipped with several safety features, including lightweight structures and emergency stop procedures. However, in some cases, such as unstructured and dynamic environments, these measures may not be sufficient to ensure that robots pose no risk to humans who enter or share their workspace.

This means that additional safety strategies must be devised to ensure a safe yet effective coexistence. In particular, it is essential to guarantee that, under any operational condition of either the human operator or the robots, the human operator's safety level can be assessed and that suitable interventions are implemented whenever this safety is at risk of being compromised. This aspect is already addressed by several industrial standards. Among these, of particular relevance is the technical specification ISO/TS 15066:2016 "Robots and robotic devices - Collaborative Robots", which defines the safety requirements for industrial robotic systems.
Although ensuring the safety of human operators who enter or share the robots' workspace is of utmost importance, it is equally important to recognize the necessity of enabling robots to effectively accomplish their assigned tasks while observing their constraints. This implies the need to find a balance between enabling the robot to perform its activities and implementing strategies that ensure human safety. In other words, the implemented safety strategies can not be excessively conservative, as this could lead to unnecessary interventions that may hinder the robot in fulfilling its assigned activities.
As previously outlined, robots are becoming increasingly prevalent in humans' daily lives. Nonetheless, for such systems, achieving complete autonomy is often infeasible or economically inefficient for the majority of real-world tasks. Therefore, in recent years, to address this situation, there has been a growing trend toward integrating human–robot collaboration strategies in various industries and domestic environments. This trend has been primarily driven by the recognition that humans and robots possess distinct yet complementary capabilities, which, when properly combined, can enhance task execution and optimize overall productivity. This collaboration can assume different forms, including *i*) the mere sharing of a workspace, where humans and robots operate on distinct tasks, and *ii*) the performance of joint tasks, wherein humans contribute with their cognitive abilities and deliberately exchange forces with robots. However, of the two cases mentioned above, the second is particularly noteworthy, as it underscores the authentic synergy that can be achieved through human–robot collaboration.
Building upon the considerations discussed above, this chapter pursues two main objectives: *i*) to provide a comprehensive analysis of the concepts of human safety and shared control, which represent distinct yet inherently interrelated components of human–robot interaction frameworks;

and *ii*) to propose effective approaches for addressing these aspects.

4.2 Human Safety Strategies

This section examines the issue of human safety. In particular, it provides an overview of existing methodologies aimed at ensuring safe coexistence and collaboration, identifying the principal challenges and limitations associated with each. Subsequently, it introduces the proposed strategy designed to achieve a balance between safety and operational efficiency, while overcoming most of the identified issues.

4.2.1 Existing Approaches and Their Limitations

A widely adopted approach to ensure safe coexistence relies on reactive evasive strategies: when a human approaches the robot, the robot withdraws to enhance safety. In this framework, several studies have been proposed. For example, [70] introduces a virtual repulsive force that moves the robot away from the human, with its magnitude proportional to a danger index derived from distance, velocity, and inertia parameters. [71], on the other hand, presents a neural network-based approach for predicting human behavior, which enables the robot to anticipate potential risks to humans and implement preventive measures before such risks become imminent. [72], instead, introduces a fast method for computing distances between the robot and moving obstacles with RGB-D sensors, which is then employed to generate repulsive forces. The problem of computing repulsive forces for collision-avoidance actions, especially when obstacles move faster than the robot, has also been investigated in [73], where the authors formulated a QP problem aimed at minimizing deviations from the reference trajectory. By contrast, [74] presents an Explicit Reference Governor framework that exploits both repulsive and attractive forces to generate trajectories driving the robot toward a desired pose, while complying with both input and state constraints imposed by the robot's nonlinear dynamics. Another approach to ensuring human safety relies on modulating the robot's velocity without deviating from the desired path, while ensuring the speed and separation monitoring (SSM) [75]. This approach has been extended in [76], where the robot links are modeled as beams and an optimization problem is formulated to determine the

velocity scaling factor and joint velocity commands, while ensuring both a minimum human–robot separation and adherence to the robot's physical constraints. This work has been recently extended in [77] to include an efficient real-time method for computing danger zones, namely the 3D volumes around the robot's links where safety constraints are violated. Conversely, [78] proposes an approach for generating optimal, smooth stop trajectories for collision avoidance, generated accounting for both the robot's dynamics and torque limitations. In contrast, [79] presents a proactive optimization framework that aims to compute the path minimizing the total execution time under a worst-case safety scenario, considering both human state and trajectory slowdowns for safety reasons. A further approach to human safety was introduced in [80], which integrates depth and thermal cameras with a fuzzy inference system to modulate the robot's velocity. Furthermore, [81] addressed the case of multiple mobile manipulators, proposing a trajectory-scaling strategy that incorporates both relative distance and velocity information of the robotic team with respect to human operators.

Despite their efficiency, both categories of the aforementioned approaches, namely evasive-based and trajectory-scaling-based, entail not only advantages but also significant disadvantages that should not be underestimated. In the case of evasive-based approaches, for example, the main benefit lies in their ability to mitigate collision risks through timely interventions whenever the separation distance drops below the prescribed threshold. Nonetheless, an overuse of such interventions may undermine both the accomplishment of assigned tasks and the overall operational effectiveness. In contrast, in the case of trajectory-scaling-based approaches, the main advantage lies in their ability to preserve the nominal path, thus enabling the robot to continue to perform its assigned tasks while maintaining high operational efficiency. Nevertheless, their primary limitation is that they do not minimize the risk of collisions. In particular, without proper calibration of the velocity modulation module, namely the component responsible for scaling the trajectory, these approaches may lead the robot to approach the human too closely, thus increasing the likelihood of collisions. Another significant drawback of velocity modulation approaches is their tendency to induce robot halts and stalls, particularly in situations where a person is stationary on the designated path.

On the basis of the aforementioned considerations and challenges, a novel

hybrid approach has been formulated to ensure the safety of humans who enter or share the robots' workspace. The approach combines velocity modulation and path modification, whose relative contribution is determined through the selection of suitable weights within a properly formulated optimization problem. In this way, the system can leverage the strengths of both categories of methods previously discussed. Specifically, through proper velocity modulation and path adaptation, the proposed approach enables the robot to simultaneously avoid collisions with humans and ensure continuity in the execution of the assigned operational activities. Furthermore, the approach has been encapsulated in a dedicated module, named *Safety Planner*, and integrated with the control architecture outlined in Chapter 3, thereby enhancing its capabilities by utilizing system redundancy to ensure a higher level of human safety.

4.2.2 Safety Field

Current collaborative robots are typically endowed with several safety features, including lightweight structures and emergency stop procedures. Nevertheless, these measures may not be sufficient to ensure that robots pose no risks to human workers who enter or share their workspace, particularly in complex, unstructured, and dynamic environments. Therefore, relying solely on the already implemented safety features may be insufficient to guarantee a safe yet efficient coexistence. This implies that in such situations, additional safety strategies must be designed and implemented. A simple yet effective approach is proposed in [81], where a safety field based on the human–robot relative position is defined. The proposed formulation considers both the overall robotic structure and an arbitrary number $n_{\hbar}$ of relevant human points, such as the endpoints of the links composing the human skeleton, as in [82]. To derive the overall safety field, a local index that quantifies human safety for a pair of points, specifically a single point P of the robot structure and a single point $P_{\hbar}$ of the human operator, must first be considered. Specifically, let $\boldsymbol{p} \in \mathbb{R}^3$ denote the position vector of the point P, $\boldsymbol{p}_{\hbar} \in \mathbb{R}^3$ the position vector of the point $P_{\hbar}$, and $d = \|\boldsymbol{p} - \boldsymbol{p}_{\hbar}\|$ the Euclidean distance between the P and $P_{\hbar}$; the local safety index can be derived as:

$$f(\boldsymbol{p}, \boldsymbol{p}_{\hbar}) = \chi(d), \tag{4.1}$$

where χ denotes a non-negative, continuous, monotonically increasing Lipschitz function. Based on the local index in Eq. (4.1), the overall safety field can be derived by integrating it along the robot links and then evaluating it for each human point. In more detail, as in [81], each robot link ℓ can be approximated as a line segment starting at $\boldsymbol{p}_\ell^0$ and ending at $\boldsymbol{p}_\ell^1$. Let $s \in [0,1]$ denote the curvilinear abscissa parameterizing the segment ℓ, with the corresponding point $\boldsymbol{p}_\ell^s \in \mathbb{R}^3$ on the segment defined as:

$$\boldsymbol{p}_\ell^s = \boldsymbol{p}_\ell^0 + (\boldsymbol{p}_\ell^1 - \boldsymbol{p}_\ell^0)s.$$

The link safety index can be derived as:

$$F_\ell\left(\boldsymbol{p}_\ell^0, \boldsymbol{p}_\ell^1, \boldsymbol{p}_\hbar\right) = \int_0^1 f(\boldsymbol{p}_\ell^s, \boldsymbol{p}_\hbar)\, ds. \tag{4.2}$$

At this point, considering all links of the robotic platform and the j-th human point $\boldsymbol{p}_{\hbar,j}$, the safety field that quantifies the interaction safety between the j-th human point and the complete robot structure can be obtained as:

$$F^j\left(\boldsymbol{q}, \boldsymbol{p}_{\hbar,j}\right) = \sum_{\ell=1}^{n} F_\ell\left(\boldsymbol{p}_\ell^0, \boldsymbol{p}_\ell^1, \boldsymbol{p}_{\hbar,j}\right). \tag{4.3}$$

Based on the index in Eq. (4.3), and by considering all the $n_\hbar$ human points, the overall safety field can be computed as:

$$F = \frac{1}{n_\hbar} \sum_{j=1}^{n_\hbar} F^j\left(\boldsymbol{q}, \boldsymbol{p}_{\hbar,j}\right). \tag{4.4}$$

The above formulation also enables accounting for multiple human operators by simply adding their respective points in Eq. (4.4). It is important to emphasize that, for the computation of the safety field, the robot must be able to detect and localize the human operator within the shared workspace. Several methods proposed in the literature can be used for this purpose, such as [83, 84].

4.2.3 Human-Safety-Oriented Control Framework

Consider a dual-arm robotic system performing a cooperative task encoded by a task function $\boldsymbol{\sigma}$, as defined in Eq. (3.31), for which a nominal desired trajectory $\boldsymbol{\sigma}_d(t)$ is provided by a Trajectory Generation module. Moreover, consider a human operator who shares the workspace with the robot, monitored through a Perception System, and whose safety level is assessed by the index $F(t)$ in Eq. (4.4). The objective is to design a Safety Planner capable of:

1. scaling down the nominal trajectory $\boldsymbol{\sigma}_d(t)$;
2. modifying the nominal path of $\boldsymbol{\sigma}_d(t)$;
3. exploiting the redundancy of the system to generate internal joint motions;

so as to generate a safe trajectory $\boldsymbol{\sigma}_s(t)$ as in Eq. (4.8), that satisfies the safety condition:

$$F(t) \geq \underline{F}(t), \tag{4.5}$$

where $\underline{F}(t)$ denotes a time-varying safety threshold, while accounting for the robot's kinematic constraints. Figure 4.2.3 shows the integration of the devised planner into the control framework detailed in Chapter 3.

It is worth pointing out that the proposed formulation requires the definition of a time-varying safety threshold $\underline{F}(t)$. As demonstrated in [81], it can be selected so as to ensure a minimum safety distance between the human operator and the robot. Alternatively, it may also be empirically calibrated by evaluating indicators such as the operator's stress level and perceived risk. In this regard, several human biological reactions may be exploited as valuable measures, as discussed in [85].

4.2.3.1 Trajectory Modification

Assume that the desired *nominal* trajectory $\boldsymbol{\sigma}_d(t)$ is parameterized with respect to a time parameter $c(t)$, i.e., $\boldsymbol{\sigma}_d(c(t))$, with $c : [t_0, t_f] \in \mathbb{R} \rightarrow [t_0, t_f] \in \mathbb{R}$, where t_0 and t_f denote the initial and final instants of the nominal trajectory, respectively. This formulation makes it possible to

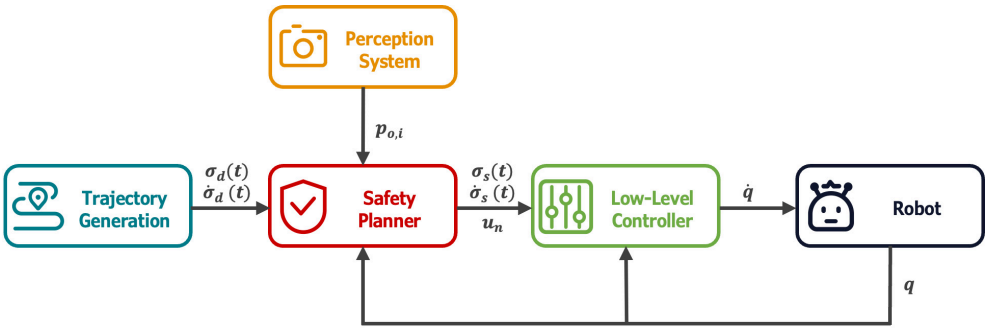


Figure 4.1: Overall control architecture. A Trajectory Generation module generates a nominal trajectory that is then modified by a Safety Planner in function of the Perception System outputs in order to obtain a reference trajectory. Finally, the latter is given in input to a controller that computes the control input to be sent to the robot.

modulate the temporal evolution of the trajectory by adjusting $c(t)$, without altering its geometric path. The time derivative of the scaled trajectory $\boldsymbol{\sigma}_d(c(t))$ can then be expressed as:

$$\boldsymbol{\nu}_d(c(t), \dot{c}(t)) = \boldsymbol{\eta}(c(t))\ \dot{c}(t), \tag{4.6}$$

with:

$$\boldsymbol{\eta}(c(t)) = \begin{bmatrix} \frac{\partial \boldsymbol{\sigma}_{a_p,d}}{\partial c} \\ 2\boldsymbol{E}(\boldsymbol{o}_a)\frac{\partial \boldsymbol{\sigma}_{a_o,d}}{\partial c} \\ \frac{\partial \boldsymbol{\sigma}_{r_p,d}}{\partial c} \\ 2\boldsymbol{E}(\boldsymbol{o}_r)\frac{\partial \boldsymbol{\sigma}_{r_o,d}}{\partial c} \end{bmatrix}, \quad \boldsymbol{E}(\boldsymbol{o}_y) = \begin{bmatrix} -o_{y,2} & -o_{y,1} & -o_{y,4} & o_{y,3} \\ -o_{y,3} & o_{y,4} & o_{y,1} & -o_{y,2} \\ -o_{y,4} & -o_{y,3} & o_{y,2} & o_{y,1} \end{bmatrix}.$$

Additionally, suppose that a time-varying term is introduced $\Delta\boldsymbol{\sigma} = \begin{bmatrix} \Delta\boldsymbol{\sigma}_{a_p}^T & \Delta\boldsymbol{\sigma}_{a_o}^T & \Delta\boldsymbol{\sigma}_{r_p}^T & \Delta\boldsymbol{\sigma}_{r_o}^T \end{bmatrix}^T \in \mathbb{R}^{14}$, which enables controlled deviations from the nominal path, thus enabling the generation of a safe trajectory $\boldsymbol{\sigma}_s(t)$ $(\boldsymbol{\nu}_s(t))$. In this formulation, $\Delta\boldsymbol{\sigma}_{a_p}^T$ and $\Delta\boldsymbol{\sigma}_{a_o}^T$ represent the variations applied to the absolute position and orientation variables,

respectively, while $\Delta\boldsymbol{\sigma}_{r_p}^T$ and $\Delta\boldsymbol{\sigma}_{r_o}^T$ represent the variations applied to the relative position and orientation variables, respectively.

By denoting with $\Delta\boldsymbol{\nu} \in \mathbb{R}^{12}$ the vector of the deviation velocities, which is defined as:

$$\Delta\boldsymbol{\nu} = \begin{bmatrix} \dot{\Delta}\boldsymbol{\sigma}_{a_p}^T & \Delta\boldsymbol{\omega}_a^T & \dot{\Delta}\boldsymbol{\sigma}_{r_p}^T & \Delta\boldsymbol{\omega}_r^T \end{bmatrix}^T, \tag{4.7}$$

the safe trajectory $\boldsymbol{\sigma}_s(t)$ ($\boldsymbol{\nu}_s(t)$) can be computed as follows:

$$\boldsymbol{\sigma}_s(t) = \begin{bmatrix} \boldsymbol{\sigma}_{a_p,s} \\ \boldsymbol{\sigma}_{a_o,s} \\ \boldsymbol{\sigma}_{r_p,s} \\ \boldsymbol{\sigma}_{r_o,s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_{a_p,d} + \Delta\boldsymbol{\sigma}_{a_p} \\ \boldsymbol{\sigma}_{a_o,d} * \Delta\boldsymbol{\sigma}_{r_o}^{-1} \\ \boldsymbol{\sigma}_{r_p,d} + \Delta\boldsymbol{\sigma}_{r_p} \\ \boldsymbol{\sigma}_{r_o,d} * \Delta\boldsymbol{\sigma}_{r_o}^{-1} \end{bmatrix}, \quad \boldsymbol{\nu}_s(t) = \begin{bmatrix} \dot{\boldsymbol{\sigma}}_{a_p,s} \\ \boldsymbol{\omega}_{a,s} \\ \dot{\boldsymbol{\sigma}}_{r_p,s} \\ \boldsymbol{\omega}_{r,s} \end{bmatrix} = \boldsymbol{\nu}_d(c(t), \dot{c}(t)) + \Delta\boldsymbol{\nu}, \tag{4.8}$$

which represents the actual trajectory to be tracked by the robot.

Let $\Delta\boldsymbol{\epsilon} \in \mathbb{R}^{12}$ denote the vector of position and orientation deviations, defined as

$$\Delta\boldsymbol{\epsilon} = \begin{bmatrix} \Delta\boldsymbol{\sigma}_{a_p}^T & \boldsymbol{\varrho}^T\Delta\boldsymbol{\sigma}_{a_o} & \Delta\boldsymbol{\sigma}_{r_p}^T & \boldsymbol{\varrho}^T\Delta\boldsymbol{\sigma}_{r_o} \end{bmatrix}^T, \tag{4.9}$$

where $\boldsymbol{\varrho}\Delta\boldsymbol{\sigma}_{a_o}$ and $\boldsymbol{\varrho}\Delta\boldsymbol{\sigma}_{r_o}$ represent the vector components of $\Delta\boldsymbol{\sigma}_{a_o}$ and $\Delta\boldsymbol{\sigma}_{r_o}$, respectively. The path modification is governed by the following law:

$$\Delta\boldsymbol{\nu} = -\boldsymbol{T}\Delta\boldsymbol{\epsilon} + \boldsymbol{\zeta}, \quad \Delta\boldsymbol{\epsilon}(t_0) = \mathbf{0}_{12}, \tag{4.10}$$

where: $\mathbf{0}_m$ denotes the m-dimensional zero vector; $\boldsymbol{T} = \text{diag}\{\tau_{a_p}\boldsymbol{I}_3,\ \tau_{a_o}\boldsymbol{I}_3,\ \tau_{r_p}\boldsymbol{I}_3,\ \tau_{r_o}\boldsymbol{I}_3\}$, with $\tau_{(\cdot)}$ being scalar positive gains interpretable as the stiffness coefficients of a virtual spring pushing the safe trajectory towards the nominal one; and $\boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\zeta}_{a_p}^T & \boldsymbol{\zeta}_{a_o}^T & \boldsymbol{\zeta}_{r_p}^T & \boldsymbol{\zeta}_{r_o}^T \end{bmatrix}^T \in \mathbb{R}^{12}$, where each $\boldsymbol{\zeta}_{(\cdot)} \in \mathbb{R}^3$ denotes an input vector to be designed. It is worth noticing that if all the input vectors $\boldsymbol{\zeta}_{(\cdot)}$ are zero-vectors, i.e., $\boldsymbol{\zeta}_{(\cdot)} = \mathbf{0}_3$, the safe trajectory $\boldsymbol{\sigma}_s$ asymptotically

converges to the nominal trajectory $\boldsymbol{\sigma}_d$. As for their design, the vectors are defined as

$$\boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\psi}_{a_p} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \boldsymbol{\psi}_{a_o} & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{\psi}_{r_p} & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{\psi}_{r_o} \end{bmatrix} \begin{bmatrix} u_{a_p} \\ u_{a_o} \\ u_{r_p} \\ u_{r_o} \end{bmatrix} = \boldsymbol{\Psi}(t)\boldsymbol{u}_\alpha, \tag{4.11}$$

where $\boldsymbol{\Psi} \in \mathbb{R}^{12\times 4}$ represents any matrix that enables deviation from the nominal trajectory $\boldsymbol{\sigma}_d$ along proper directions $\boldsymbol{\psi}_{(.)} \in \mathbb{R}^3$ and may be selected according to the specific operation to be performed, while the components of $\boldsymbol{u}_\alpha$ (i.e., u_{a_p}, u_{a_o}, u_{r_p}, and u_{r_o}) are scalar inputs that modulate the amount of the deviation. It is worth noticing that if $c(t) = t$ and $\Delta\boldsymbol{\epsilon} = \mathbf{0}_{12}$, it follows that $\boldsymbol{\sigma}_d(c(t)) = \boldsymbol{\sigma}_d(t)$. Furthermore, if $\dot{c} = 1$, then $\boldsymbol{\nu}_s(t) = \boldsymbol{\nu}_s(c(t)) = \boldsymbol{\nu}_d(c(t))$, which indicates that the system tracks the nominal reference velocity. The rationale for this design choice lies in the intention to decouple nominal trajectory scaling from path modification. Specifically, adjusting the scaling parameter $c(t)$ enables the online scaling of the nominal trajectory $\boldsymbol{\sigma}_d(t)$, whereas changes in $\Delta\boldsymbol{\sigma}$ result in modifications of the path.

4.2.3.2 Safety Planner

To design the Safety Planner mentioned in Section 4.2.3, the time derivative of the safety field F, defined in Eq. (4.4), must first be derived. Specifically, in the remainder of this section, it will be proven to be linear with respect to the derivative of the scaling parameter $\dot{c}(t)$, the path modification input $\boldsymbol{u}_\alpha$, and the input $\dot{\boldsymbol{q}}_n$ in Eq. (**??**). This property is exploited to implement the scaling and path modification procedure.

As for the time derivative $\dot{F}$, this can be obtained from the derivative of f in Eq. (4.1). Specifically, $\dot{f}$ can be computed as:

$$\dot{f} = \frac{\partial \chi\left(d^s_{\ell,\hbar}\right)}{\partial d^s_{\ell,\hbar}} \dot{d}^s_{\ell,\hbar}, \tag{4.12}$$

where $d^s_{\ell,\hbar} = \|\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar\|$, i.e., the Euclidean distance between P^s_ℓ, which denotes the point on the robot link ℓ corresponding to the curvilinear abscissa s, and $P_\hbar$, which represents a generic point of the human operator. Regarding $\dot{d}^s_{\ell,\hbar}$, this term represents the time derivative of $d^s_{\ell,\hbar}$ and can be derived as follows:

$$\dot{d}^s_{\ell,\hbar} = \frac{(\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar)^T(\dot{\boldsymbol{p}}^s_\ell - \dot{\boldsymbol{p}}_\hbar)}{d^s_{\ell,\hbar}}, \tag{4.13}$$

with $d^s_{\ell,\hbar} \neq 0$. At this point, consider the relationship between the linear velocity of the point $\boldsymbol{p}^s_\ell$ and the robot joint variables $\boldsymbol{q}$, which can be expressed as:

$$\dot{\boldsymbol{p}}^s_\ell = \boldsymbol{J}^s_\ell(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{4.14}$$

where $\boldsymbol{J}^s_\ell(\boldsymbol{q}) \in \mathbb{R}^{3\times n}$ denotes the position Jacobian matrix associated with the point $\boldsymbol{p}^s_\ell$, i.e., the first three rows of the Jacobian matrix that relates the joint velocities to the linear/angular velocity of $\boldsymbol{p}^s_\ell$. By considering Eqs.(2.9)-(2.10) and, for the sake of readability, omitting the explicit dependence of the Jacobian matrices on $\boldsymbol{q}$, Eq. (4.14) can be reformulated as:

$$\dot{\boldsymbol{p}}^s_\ell = \boldsymbol{J}^s_\ell \boldsymbol{J}^\dagger_\sigma(\boldsymbol{\nu}_s + \boldsymbol{K}_\sigma\tilde{\boldsymbol{\sigma}}) + \boldsymbol{J}^s_\ell \boldsymbol{N}_\sigma \dot{\boldsymbol{q}}_n, \tag{4.15}$$

Substituting Eqs. (4.6)-(4.11) into Eq. (4.14), yields:

$$\dot{\boldsymbol{p}}^s_\ell = \boldsymbol{\gamma}_1\dot{c} + \boldsymbol{\Gamma}_2\boldsymbol{u}_\alpha + \boldsymbol{\Gamma}_3\dot{\boldsymbol{q}}_n + \boldsymbol{\gamma}_4, \tag{4.16}$$

which is linear with respect to $\dot{c}$, $\boldsymbol{u}_\alpha$, and $\dot{\boldsymbol{q}}_n$. As for the coefficients

$$\begin{cases} \boldsymbol{\gamma}_1 = \boldsymbol{J}^s_\ell \boldsymbol{J}^\dagger_\sigma \boldsymbol{\eta}, \\ \boldsymbol{\Gamma}_2 = \boldsymbol{J}^s_\ell \boldsymbol{J}^\dagger_\sigma \boldsymbol{\Psi}, \\ \boldsymbol{\Gamma}_3 = \boldsymbol{J}^s_\ell \boldsymbol{N}_\sigma, \\ \boldsymbol{\gamma}_4 = \boldsymbol{J}^s_\ell \boldsymbol{J}^\dagger_\sigma[\boldsymbol{K}_\sigma\tilde{\boldsymbol{\sigma}} - \boldsymbol{T}\Delta\boldsymbol{\epsilon}], \end{cases} \tag{4.17}$$

with $\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_4 \in \mathbb{R}^3$, $\boldsymbol{\Gamma}_2 \in \mathbb{R}^{3\times 4}$, and $\boldsymbol{\Gamma}_3 \in \mathbb{R}^{3\times n}$.

At this point, leveraging Eqs. (4.12) and (4.15), the derivative of the safety index $\dot{f}$ can be rewritten as

$$\dot{f} = \lambda_1 \dot{c} + \boldsymbol{\lambda}_2^T \boldsymbol{u}_\alpha + \boldsymbol{\lambda}_3^T \dot{\boldsymbol{q}}_n + \lambda_4, \tag{4.18}$$

where $\lambda_1, \lambda_4 \in \mathbb{R}$, $\boldsymbol{\lambda}_2 \in \mathbb{R}^4$, and $\boldsymbol{\lambda}_3 \in \mathbb{R}^3$ can be obtained as

$$\begin{cases} \lambda_1 = \frac{\partial \chi}{\partial d^s_{\ell,\hbar}} \frac{(\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar)^T}{d^s_{\ell,\hbar}} \boldsymbol{\gamma}_1, \\ \boldsymbol{\lambda}_2^T = \frac{\partial \chi}{\partial d^s_{\ell,\hbar}} \frac{(\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar)^T}{d^s_{\ell,\hbar}} \boldsymbol{\Gamma}_2, \\ \boldsymbol{\lambda}_3^T = \frac{\partial \chi}{\partial d^s_{\ell,\hbar}} \frac{(\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar)^T}{d^s_{\ell,\hbar}} \boldsymbol{\Gamma}_3, \\ \lambda_4 = \frac{\partial \chi}{\partial d^s_{\ell,\hbar}} \frac{(\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar)^T}{d^s_{\ell,\hbar}} \boldsymbol{\gamma}_4 - \frac{\partial \chi}{\partial d^s_{\ell,\hbar}} \frac{(\boldsymbol{p}^s_\ell - \boldsymbol{p}_\hbar)^T}{d^s_{\ell,\hbar}} \dot{\boldsymbol{p}}_\hbar. \end{cases} \tag{4.19}$$

By leveraging Eqs. (4.2)-(4.4), the time derivative of the local safety index can be extended to the entire structure of the dual-arm robot and to all the relevant human points

$$\dot{F}(t) = \mu_1(t)\dot{c}(t) + \boldsymbol{\mu}_2^T(t)\boldsymbol{u}_\alpha + \boldsymbol{\mu}_3^T(t)\dot{\boldsymbol{q}}_n + \mu_4(t), \tag{4.20}$$

where $\mu_1, \mu_4 \in \mathbb{R}$, $\boldsymbol{\mu}_2 \in \mathbb{R}^4$, and $\boldsymbol{\mu}_3 \in \mathbb{R}^3$ can be obtained as

$$\begin{cases} \mu_1 = \frac{1}{n_\hbar} \sum_{j=1}^{n_\hbar} \sum_{\ell=1}^{n} \int_0^1 \lambda_1(\boldsymbol{p}^s_\ell, \boldsymbol{p}_{\hbar,j}, \boldsymbol{q}, c)\, ds, \\ \boldsymbol{\mu}_2 = \frac{1}{n_\hbar} \sum_{j=1}^{n_\hbar} \sum_{\ell=1}^{n} \int_0^1 \boldsymbol{\lambda}_2(\boldsymbol{p}^s_\ell, \boldsymbol{p}_{\hbar,j}, \boldsymbol{q}, c)\, ds, \\ \boldsymbol{\mu}_3 = \frac{1}{n_\hbar} \sum_{j=1}^{n_\hbar} \sum_{\ell=1}^{n} \int_0^1 \boldsymbol{\lambda}_3(\boldsymbol{p}^s_\ell, \boldsymbol{p}_{\hbar,j}, \boldsymbol{q}, c)\, ds, \\ \mu_4 = \frac{1}{n_\hbar} \sum_{j=1}^{n_\hbar} \sum_{\ell=1}^{n} \int_0^1 \lambda_4(\boldsymbol{p}^s_\ell, \boldsymbol{p}_{\hbar,j}, \boldsymbol{q}, c)\, ds, \end{cases} \tag{4.21}$$

that, as $\dot{f}$ in (4.18), is linear with respect to $\dot{c}$, $\boldsymbol{u}_\alpha$, and $\dot{\boldsymbol{q}}_n$.

By setting $\dot{c} = u_c$ and $\dot{\boldsymbol{q}}_n = \boldsymbol{u}_n$, the overall system dynamics can be formulated as

$$\begin{bmatrix} \dot{c} \\ \Delta\boldsymbol{\nu} \\ \dot{F} \end{bmatrix} = \begin{bmatrix} 0 \\ -\boldsymbol{T}\Delta\boldsymbol{\epsilon} \\ \mu_4 \end{bmatrix} + \begin{bmatrix} 1 & \mathbf{0}_4^T & \mathbf{0}_n^T \\ \mathbf{0}_{12} & \boldsymbol{\Psi} & \boldsymbol{O}_{12\times n} \\ \mu_1 & \boldsymbol{\mu}_2^T & \boldsymbol{\mu}_3^T \end{bmatrix} \begin{bmatrix} u_c \\ \boldsymbol{u}_\alpha \\ \boldsymbol{u}_n \end{bmatrix}. \tag{4.22}$$

Then, by introducing the state vector $\boldsymbol{\xi} = \begin{bmatrix} c & \Delta\boldsymbol{\sigma}^T & F \end{bmatrix}^T$, the overall system dynamics in Eq. (4.22) can be reformulated as

$$\dot{\boldsymbol{\xi}} = \boldsymbol{f}(t,\boldsymbol{\xi}) + \boldsymbol{g}(t,\boldsymbol{\xi}) \begin{bmatrix} u_c \\ \boldsymbol{u}_\alpha \\ \boldsymbol{u}_n \end{bmatrix} = \boldsymbol{f}(t,\boldsymbol{\xi}) + \boldsymbol{g}(t,\boldsymbol{\xi})\boldsymbol{u}_\xi, \tag{4.23}$$

which is structurally equivalent to Eq. (2.17).

Before proceeding, it is useful to briefly recall the role of each component of the overall input vector $\boldsymbol{u}_\xi \in \mathbb{R}^{n+5}$

- u_c scales down the nominal trajectory $\boldsymbol{\sigma}_d$;
- the components of $\boldsymbol{u}_\alpha = \begin{bmatrix} u_{a_p} & u_{a_o} & u_{r_p} & u_{r_o} \end{bmatrix}^T$ modulate the deviation from $\boldsymbol{\sigma}_{a_p,d}$, $\boldsymbol{\sigma}_{a_o,d}$, $\boldsymbol{\sigma}_{r_p,d}$, and $\boldsymbol{\sigma}_{r_o,d}$, respectively;
- $\boldsymbol{u}_n$ generates internal joint motions that increase the safety field value without affecting the operational task value $\boldsymbol{\sigma}$.

Building on this, the objective is to design a Planner (more precisely, a Safety Planner) capable of computing an input $\boldsymbol{u}_\xi$ that ensures the safety field value remains, and when necessary is driven, above the prescribed lower threshold $\underline{F}$, while satisfying additional constraints on u_c and $\boldsymbol{u}_\alpha$, which are detailed below and incorporated into a dedicated QP problem.

Before introducing the dedicated QP problem to be addressed by the Safety Planner, consider the following CBF

$$\underline{h}_f = F - \underline{F}, \tag{4.24}$$

Based on this CBF, the constraint on the decision variables enforcing Eq. (4.5) can be formulated as

$$\boldsymbol{J}_f \boldsymbol{u}_\xi \geq -k_f \underline{h}_f - \mu_4 \tag{4.25}$$

where $k_f > 0$ denotes a scalar gain, whereas $\boldsymbol{J}_f$ is a Jacobian matrix which, according to Eq. (4.20), is given by

$$\boldsymbol{J}_f = \begin{bmatrix} \mu_1 & \boldsymbol{\mu}_2^T & \boldsymbol{\mu}_3^T \end{bmatrix}. \tag{4.26}$$

In order to determine an $\boldsymbol{u}_\xi$ that guarantees the safety field value remains, and when required is enforced, above the prescribed lower threshold $\underline{F}$, while also fulfilling the additional constraints on u_c and $\boldsymbol{u}_\alpha$, the Safety Planner has to solve the following constrained optimization problem

$$\min_{\boldsymbol{u}_\xi} \quad \frac{1}{2}(\boldsymbol{u}_\xi - \boldsymbol{u}_{\xi,n})^T \boldsymbol{Q}_\xi (\boldsymbol{u}_\xi - \boldsymbol{u}_{\xi,n}) + \frac{1}{2} q_w w_f^2 \tag{4.27}$$

$$s.t. \quad 0 \leq u_c \leq 1, \tag{4.28}$$

$$\underline{\boldsymbol{u}}_\alpha \leq \boldsymbol{u}_\alpha \leq \bar{\boldsymbol{u}}_\alpha, \tag{4.29}$$

$$\boldsymbol{J}_f \boldsymbol{u}_\xi + w_f \geq -k_f \underline{h}_f - \mu_4. \tag{4.30}$$

Regarding the constraints

- Eq. (4.28) constrains the time scaling parameter $u_c = \dot{c}$ to lie within the interval $[0, 1]$. In particular, the inequality $\dot{c} \geq 0$ ensures that the time parameter c never decreases, thus preventing the trajectory from being traveled backwards, while the inequality $\dot{c} \leq 1$ guarantees that the trajectory is never traveled at a velocity higher than the nominal one.
- Eq. (4.29) imposes a lower ($\underline{\boldsymbol{u}}_\alpha$) and upper ($\bar{\boldsymbol{u}}_\alpha$) bound on the deviation velocity. Such constraints are important in practice, as they prevent abrupt departures from the nominal trajectory. Furthermore, it should be emphasized that, if the application necessitates

an upper bound on the maximum path deviation $\Delta\boldsymbol{\sigma}$, such a requirement can likewise be incorporated into the proposed framework through the use of the CBF approach.

- Eq. (4.30) addresses the human-robot safety according to the three statements outlined in the introduction of this Chapter. Specifically, it enables the definition of a virtual input vector that exploits the system's redundant degrees of freedom to enhance and maintain the Safety Field above the prescribed threshold. The introduction of the slack variable w_f is necessary to relax the constraint whenever the input $\boldsymbol{u}_\alpha$ reaches the limits imposed by the constraints in Eq. (4.29).

Concerning the objective function in Eq. (4.27), its purpose is to determine the input vector $\boldsymbol{u}_\xi = \begin{bmatrix} u_c & \boldsymbol{u}_\alpha^T & \boldsymbol{u}_n^T \end{bmatrix}$ that is closest to the desired input vector $\boldsymbol{u}_{\xi,d} = \begin{bmatrix} u_{c,d} & \boldsymbol{u}_{\alpha,d}^T & \boldsymbol{u}_{n,d}^T \end{bmatrix}$, where

$$\begin{cases} u_{c,d} = 1, \\ \boldsymbol{u}_{\alpha,d} = \mathbf{0}_4, \\ \boldsymbol{u}_{n,d} = k_n \frac{\partial F}{\partial \boldsymbol{q}}, \end{cases} \tag{4.31}$$

with $k_n > 0$. In more detail, when $u_c = u_{c,d}$, it follows that $\dot{c} = 1$, meaning that the cooperative task is performed at nominal speed. Likewise, $\boldsymbol{u}_\alpha = \boldsymbol{u}_{\alpha,d}$ yields $\Delta\boldsymbol{\epsilon} = \mathbf{0}_{12}$, which ensures that the nominal path is followed without deviations. Finally, when $\boldsymbol{u}_n$ is as close as possible to $\boldsymbol{u}_{n,d}$. i.e., proportional to the gradient of the safety field, enables the exploitation of the system's redundancy to enhance the Safety Field value.

Concerning $q_w \in \mathbb{R}$, it is a positive scalar weight related to the scalar slack variable w_f, whereas $\boldsymbol{Q}_\xi \in \mathbb{R}^{(n+5)\times(n+5)}$ is a positive-definite weight matrix that can be tuned to modulate the relative contribution of each of the three virtual inputs. For instance, assigning a higher value to the weight corresponding to u_c entails that trajectory deviation and internal motions are prioritized over scaling.

Once the QP problem in Eq. (4.27) has been solved, the Safety Planner exploits the corresponding solutions in Eq. (4.8) to generate the reference trajectory $\boldsymbol{\sigma}_s(t)$ ($\boldsymbol{\nu}_s(t)$).

Algorithm 1 outlines the algorithmic procedure of the proposed Safety Planner. Specifically, at first, the procedure begins with the computation

of the Safety Field value F according to Eq (4.4); then, the QP problem (4.27) is solved by taking as input the computed F together with the nominal trajectory $\boldsymbol{\sigma}_d(t)$ ($\boldsymbol{\nu}_d(t)$). The resulting solutions are then exploited to derive the reference trajectory $\boldsymbol{\sigma}_s(t)$ ($\boldsymbol{\nu}_s(t)$), according to the following steps:

- u_c is integrated over time to derive c;
- The values of c and $\boldsymbol{u}_\alpha$ are then substituted into Eq. (4.10) to compute the deviation velocity $\Delta\boldsymbol{\nu}$, which is subsequently integrated over time to obtain $\Delta\boldsymbol{\sigma}$;
- Finally, $\Delta\boldsymbol{\sigma}$, c, and u_c are used in Eq.(4.8) to compute the reference trajectory $\boldsymbol{\sigma}_s(\boldsymbol{\nu}_s)$. This trajectory, along with the vector $\boldsymbol{u}_n = \dot{\boldsymbol{q}}_n$ obtained from Eq.(4.27), is then supplied as input to the controller in use.

At this point, to also account for higher-priority tasks (e.g., joint position and velocity limits), the Safety Planner's output is integrated into the HQP framework through the High Control Layer introduced in Section 3.3.2. The trajectory tracking can be achieved through a two-step procedure. The first step consists in properly embedding $\boldsymbol{\nu}_s$ and $\boldsymbol{\sigma}_s$ within the expression of $\boldsymbol{b}_\sigma$ in Eq. (3.34), as illustrated below

$$\boldsymbol{b}_\sigma = \boldsymbol{\nu}_s + \boldsymbol{K}_\sigma \tilde{\boldsymbol{\sigma}}, \tag{4.32}$$

with

$$\tilde{\boldsymbol{\sigma}} = \begin{bmatrix} \boldsymbol{\sigma}_{a_p,s} - \boldsymbol{\sigma}_{a_p} \\ \boldsymbol{\varrho}_{\tilde{\sigma}_{a_o}} \\ \boldsymbol{\sigma}_{r_p,s} - \boldsymbol{\sigma}_{r_p} \\ \boldsymbol{\varrho}_{\tilde{\sigma}_{r_o}} \end{bmatrix}, \tag{4.33}$$

where $\boldsymbol{\varrho}_{\tilde{\sigma}_{a_o}}$ is the vector part of the quaternion $\boldsymbol{\sigma}_{a_o,s} * \boldsymbol{\sigma}_{a_o}^{-1}$, while $\boldsymbol{\varrho}_{\tilde{\sigma}_{r_o}}$ is the vector part of the quaternion $\boldsymbol{\sigma}_{r_o,s} * \boldsymbol{\sigma}_{r_o}^{-1}$. This error term enables the tracking of the safe trajectory $\boldsymbol{\sigma}_s$ ($\boldsymbol{\nu}_s$) generated by the Safety Planner. In contrast, the second step involves the proper integration of $\dot{\boldsymbol{q}}_n$, that is, the reference describing the desired internal motion. This goal can be

accomplished by introducing an additional layer into the HQP framework to manage null-space motions, according to the following approach.

$$\begin{aligned} \min_{\boldsymbol{w}_{(\cdot)},\dot{\boldsymbol{q}}} \quad & \frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{Q}_{(\cdot)}\dot{\boldsymbol{q}} + \boldsymbol{w}_{(\cdot)}^T\boldsymbol{Q}_{w_{(\cdot)}}\boldsymbol{w}_{(\cdot)} \\ \text{s.t.} \quad & \dot{\boldsymbol{q}} + \boldsymbol{w}_{(\cdot)} = \dot{\boldsymbol{q}}_n \\ & \cdots \\ & \underline{\boldsymbol{b}}_1 \leq \boldsymbol{J}_1(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{w}_1^\star \leq \bar{\boldsymbol{b}}_1. \end{aligned} \tag{4.34}$$

The introduction of this term allows to increase the safety field value without affecting the operational tasks. It should be emphasized that, owing to the possible activation of additional constraints related to such tasks, the null space associated with the higher-priority tasks may differ from that employed within the Safety Planner to compute $\dot{\boldsymbol{q}}_n$; thus slack variables must be employed at this stage.

Algorithm 1 Safety Planner algorithm

Require: Nominal trajectory $\boldsymbol{\sigma}_d(\boldsymbol{\nu}_d)$, Joint position vector $\boldsymbol{q}$, Human point positions $\boldsymbol{p}_{\hbar,1}, \cdots, \boldsymbol{p}_{\hbar,n_\hbar}$.

Begin

$F = \texttt{SafetyField}\ (\boldsymbol{q},\ \boldsymbol{p}_{\hbar,1}, \cdots, \boldsymbol{p}_{\hbar,n_\hbar}) \rightarrow$ Eq. (4.4)

$$\begin{bmatrix} u_c \\ \boldsymbol{u}_\alpha \\ \boldsymbol{u}_n \end{bmatrix} = \texttt{SolveQP}\ (\bar{F}, \boldsymbol{\sigma}_d, \boldsymbol{\nu}_d) \rightarrow \text{Eq. (4.27)}$$

$c = \texttt{Integration}\ (u_c)$

$\Delta\boldsymbol{\nu} = \texttt{DeviationVelocity}\ (\boldsymbol{u}_\alpha, c, u_c) \rightarrow$ Eq. (4.10)
$\Delta\boldsymbol{\sigma} = \texttt{Integration}\ (\Delta\boldsymbol{\nu})$

$$\begin{bmatrix} \boldsymbol{\sigma}_s \\ \boldsymbol{\nu}_s \end{bmatrix} = \texttt{SafeTrajectory}$$

$$\begin{bmatrix} \boldsymbol{\sigma}_s \\ \boldsymbol{\nu}_s \end{bmatrix} = \texttt{SafeTrajectory}\ (\Delta\boldsymbol{\sigma}, \Delta\boldsymbol{\nu}, c) \rightarrow \text{Eq. (4.8)}$$

Return: Safe trajectory $\boldsymbol{\sigma}_s(\boldsymbol{\nu}_s)$, Null-space velocity vector $\boldsymbol{u}_n$

End

4.3 Shared-Control Strategies

This section focuses on the topic of shared control in the context of human-robot collaboration. It reviews the principal methodologies applied in situations where humans and robots intentionally exchange forces to accomplish a joint objective, highlighting their main constraints and shortcomings. Subsequently, it introduces a novel strategy designed to enable the operator to support robotic systems when operational deadlocks occur.

4.3.1 Existing Approaches and Their Limitations

As anticipated in Section 4.1, collaboration can assume different forms, including *i*) the mere sharing of a workspace, where humans and robots operate on distinct tasks, and *ii*) the performance of joint tasks, wherein humans contribute their cognitive abilities and deliberately exchange forces with robots. In the first scenario, the main objective of the robot control strategy is to guarantee the safety of humans by preventing any potential harm that may arise from collisions between the robot and the human [86]. This objective can be achieved through several approaches, including the implementation of evasive maneuvers to maintain a safe separation distance [87], or the adoption of dynamic trajectory scaling methods, as illustrated in [81]. However, as outlined in Section 4.1, of the two cases mentioned above, the second is particularly noteworthy, as it underscores the authentic synergy that can be achieved through human–robot collaboration. In this context, considerable attention, especially in recent years, has been directed toward *shared control* scenarios, which, in certain ways, are inspired by *human–human* interaction contexts, as robot autonomy is retained to a certain degree and equal roles may be assigned to both robotic and human counterparts. The shared control paradigm is commonly employed in teleoperation scenarios [88], where the human operator provides control inputs through a haptic interface. At the same time, the robot retains a certain level of autonomy, for example, by adopting collision-avoidance strategies based on its perceptual abilities. This chapter, however, is particularly focused on shared control within the framework of physical human-robot interaction. Over the past decades, several

approaches have been proposed in this domain [89]. For instance, [90] introduces a game-theory-based solution, in which both the human and the robot seek to optimize a common cost function. Within this formulation, the roles of the human and the robot evolve dynamically according to the force applied by the human operator. More precisely, the stronger the force exerted by the human, the greater the influence of the human motion on the system. In [91], the authors introduce a heuristic agreement index that regulates the robot's role according to the degree of alignment between human and robot forces. Building on this line of research, [92] presents a data-driven stochastic modeling approach, which handles uncertainties in human behavior through the formulation of a risk-sensitive optimization problem. More recently, [93] has achieved a comparable outcome by adjusting the robot's trajectory in accordance with the forces exerted by the human operator. An alternative solution to this problem is presented in [94]. More in detail, the authors propose a strategy in which control is determined using a metric derived from a Bayesian filter, whose value is dynamically adapted based on online sensor measurements while accounting for variations caused by noise. Moreover, the authors demonstrate the stability of the proposed shared control architecture, even under communication delays between the human operator and the robot. In [95], the authors addressed tasks involving interactions with the environment, introducing the notion of *corrective shared autonomy*, a novel approach that allows users to refine critical robot state variables, including position, force, and execution rate. In this study, the authors demonstrate both the feasibility and the benefits of the proposed method, such as lowering user effort and physical strain. In the framework of shared control and physical interaction, the work in [96] investigates variable admittance by introducing the concept of *power envelope regulation*, i.e., a mechanism for adapting admittance parameters in accordance with human intent while safeguarding interaction safety.

Building upon the aforementioned considerations, a practical approach has been developed to effectively address complex operations, such as vineyard harvesting tasks, by leveraging human-manipulator interaction. Specifically, the approach is designed to enhance robotic systems' capabilities within dynamic, human-shared environments. This goal is realized through the possibility for human operators to physically intervene when the system encounters operational stagnation. More in detail, this chapter envisions the case of a bi-manual mobile robot equipped with advanced

perception capabilities and F/T sensors (Figure 3.1), whose autonomy level is dynamically adjusted according to the surrounding environment, and with perception system outputs employed for bunch detection and cut-point estimation. Furthermore, the robot is endowed with the ability to request human intervention in cases of perception or goal-reaching failure. Such failures typically arise when grape bunches overlap or when peduncles are wrapped around vineyard canes, conditions that significantly hinder the identification of the cutting point. As for the robot's autonomy level, it is handled through a variable admittance controller, whose gains are adapted to permit either the human or the robot to assume leadership, depending on the perception system's performance. This controller is seamlessly integrated into the Hierarchical Quadratic Programming (HQP) control framework described in Section 2.5, which enables the robot to accomplish multiple tasks concurrently.

4.3.2 Adaptive Shared Control Framework

This section presents in detail the strategies developed to endow the robotic systems, controlled by the framework introduced in Chapter 3, with the capability to physically interact with both human operators and the surrounding environment, while guaranteeing the integrity of the robot itself and the safety of operators throughout the interaction.

4.3.2.1 Admittance

The human–robot interaction strategy described below is formulated as an operational task within the HQP framework introduced in Chapter 3. Its objective is to enable the controlled system to track a desired end-effector trajectory while maintaining compliant behavior with respect to external forces that may arise from contact with human operators or the environment. Assume that the desired trajectories for the end-effectors are generated by a dedicated framework component, referred to as the Trajectory Generation module, and defined as follows:

$$\begin{aligned}\boldsymbol{a}_d &= \begin{bmatrix}\ddot{\boldsymbol{p}}_{l,d}^T & \boldsymbol{\alpha}_{l,d}^T & \ddot{\boldsymbol{p}}_{r,d}^T & \boldsymbol{\alpha}_{r,d}^T\end{bmatrix}^T,\\ \boldsymbol{v}_d &= \begin{bmatrix}\dot{\boldsymbol{p}}_{l,d}^T & \boldsymbol{\omega}_{l,d}^T & \dot{\boldsymbol{p}}_{r,d}^T & \boldsymbol{\omega}_{r,d}^T\end{bmatrix}^T,\\ \boldsymbol{\rho}_d &= \begin{bmatrix}\boldsymbol{p}_{l,d}^T & \boldsymbol{o}_{l,d}^T & \boldsymbol{p}_{r,d}^T & \boldsymbol{o}_{r,d}^T\end{bmatrix}^T,\end{aligned}$$

where $\boldsymbol{p}_{(\cdot),d}$, $\dot{\boldsymbol{p}}_{(\cdot),d}$, and $\ddot{\boldsymbol{p}}_{(\cdot),d}$ denote the desired linear position, velocity and acceleration, respectively, while $\boldsymbol{o}_{(\cdot),d}$, $\boldsymbol{\omega}_{(\cdot),d}$, and $\boldsymbol{\alpha}_{(\cdot),d}$ denote the desired orientation quaternion, angular velocity and angular acceleration, respectively, for the *r*ight and *l*eft end-effectors.

Assuming that a wrench sensor is mounted at the wrist of both manipulators, the following vector can be defined:

$$\boldsymbol{h} = \begin{bmatrix}\boldsymbol{h}_l^T & \boldsymbol{h}_r^T\end{bmatrix}^T = \begin{bmatrix}\boldsymbol{f}_l^T & \boldsymbol{\mu}_l^T & \boldsymbol{f}_r^T & \boldsymbol{\mu}_r^T\end{bmatrix}^T, \tag{4.35}$$

which represents the vector of the external wrenches measured at the end-effectors, obtained by stacking the forces $\boldsymbol{f}_{(\cdot)}$ and moments $\boldsymbol{\mu}_{(\cdot)}$ associated with both end-effectors. The objective of this strategy is to lead the system to exhibit the following dynamic:

$$\boldsymbol{K}_m\,\tilde{\boldsymbol{a}} + \boldsymbol{K}_d\,\tilde{\boldsymbol{v}} + \boldsymbol{K}_p\,\tilde{\boldsymbol{\rho}} = \boldsymbol{h}, \tag{4.36}$$

where:

$$\tilde{\boldsymbol{a}} = \boldsymbol{a}_d - \boldsymbol{a}, \quad \tilde{\boldsymbol{v}} = \boldsymbol{v}_d - \boldsymbol{v}, \quad \tilde{\boldsymbol{\rho}} = \begin{bmatrix}\boldsymbol{p}_{l,d} - \boldsymbol{p}_l\\ \tilde{\boldsymbol{\varrho}}_l\\ \boldsymbol{p}_{r,d} - \boldsymbol{p}_r\\ \tilde{\boldsymbol{\varrho}}_r\end{bmatrix}, \tag{4.37}$$

represent the acceleration, velocity, and configuration errors, with $\tilde{\boldsymbol{\varrho}}_l$ and $\tilde{\boldsymbol{\varrho}}_r$ denoting the vector components of the quaternions $\boldsymbol{o}_{l,d} * \boldsymbol{o}_l^{-1}$ and $\boldsymbol{o}_{r,d} * \boldsymbol{o}_r^{-1}$, respectively. While,

$$\boldsymbol{K}_m = \begin{bmatrix} \boldsymbol{K}_{m,l} & \boldsymbol{0}_{6\times6} \\ \boldsymbol{0}_{6\times6} & \boldsymbol{K}_{m,r} \end{bmatrix} \in \mathbb{R}^{12\times12}, \tag{4.38}$$

$$\boldsymbol{K}_d = \begin{bmatrix} \boldsymbol{K}_{d,l} & \boldsymbol{0}_{6\times6} \\ \boldsymbol{0}_{6\times6} & \boldsymbol{K}_{d,r} \end{bmatrix} \in \mathbb{R}^{12\times12}, \tag{4.39}$$

$$\boldsymbol{K}_p = \begin{bmatrix} \boldsymbol{K}_{p,l} & \boldsymbol{0}_{6\times6} \\ \boldsymbol{0}_{6\times6} & \boldsymbol{K}_{p,r} \end{bmatrix} \in \mathbb{R}^{12\times12}, \tag{4.40}$$

represent the desired virtual mass, damping, and stiffness, respectively, which may be either constant or time-dependent.

Since the HQP framework in Eq. (2.23) does not allow direct incorporation of constraints involving acceleration, these must necessarily be reformulated in terms of velocity before being incorporated. On this basis, the end-effectors' acceleration can be approximated by numerically deriving their velocity:

$$\boldsymbol{a}(t) = \frac{\boldsymbol{v}(t) - \boldsymbol{v}(t - T_s)}{T_s}, \tag{4.41}$$

where $\boldsymbol{v}(t)$ denotes the current velocity, whereas $\boldsymbol{v}(t - T_s)$ represents the velocity at the previous time step, with T_s indicating the sampling period imposed by the digital implementation of the control law. In virtue of the above approximation, Eq. (4.36) can be rewritten as:

$$\boldsymbol{K}_m \boldsymbol{a}_d - \boldsymbol{K}_m \frac{\boldsymbol{v}}{T_s} + \boldsymbol{K}_m \frac{\boldsymbol{v}(t - T_s)}{T_s} + \boldsymbol{K}_d\, \tilde{\boldsymbol{v}} + \boldsymbol{K}_p\, \tilde{\boldsymbol{\rho}} = \boldsymbol{h}, \tag{4.42}$$

which, in turn, can be reformulated as:

$$\left(-\frac{\boldsymbol{K}_m}{T_s} - \boldsymbol{K}_d \right) \boldsymbol{v} = \boldsymbol{h} - \boldsymbol{\gamma}, \tag{4.43}$$

where:

$$\boldsymbol{\gamma} = \boldsymbol{K}_m\, \boldsymbol{a}_d - \frac{\boldsymbol{K}_m}{T_s}\, \boldsymbol{v}(t - T_s) - \boldsymbol{K}_d\, \boldsymbol{v}_d - \boldsymbol{K}_p\, \tilde{\boldsymbol{\varrho}}.$$

Substituting Eq. (3.7) into Eq. (4.3.2.1) yields:

$$\left(\frac{\boldsymbol{K}_m}{T_s} + \boldsymbol{K}_d\right) \boldsymbol{J}\ \dot{\boldsymbol{q}} = -\boldsymbol{\gamma} - \boldsymbol{h}. \tag{4.44}$$

By reformulating Eq. (4.44) as:

$$\boldsymbol{J}_{\text{adm}}\ \dot{\boldsymbol{q}} = \boldsymbol{b}_{\text{adm}}, \tag{4.45}$$

where $\boldsymbol{J}_{\text{adm}} = \left(\frac{\boldsymbol{K}_m}{T_s} + \boldsymbol{K}_d\right) \boldsymbol{J}(\boldsymbol{q})$ and $\boldsymbol{b}_{\text{adm}} = -\boldsymbol{\gamma} - \boldsymbol{h}$; it can be be incorporated as an equality constraint within the HQP control framework.

4.3.2.2 Hand-Guiding

An alternative human–robot interaction strategy is hand-guiding, which enables the human operator to manually bring the end-effectors to the desired positions. This functionality is particularly useful, for instance, to bring the cutting tool to the grape peduncle when it is obstructed by leaves. This control strategy can be directly obtained from the admittance one by imposing $\boldsymbol{K}_p = \boldsymbol{O}_{12\times12}$ and $\boldsymbol{a}_d = \boldsymbol{v}_d = \boldsymbol{0}$ in Eq. (4.36).

As a result of the above substitution, the constraint to be incorporated within the HQP control framework, to enforce the robot to exhibit the desired behavior, takes the following form:

$$\boldsymbol{J}_{\text{hg}}\ \dot{\boldsymbol{q}} = \boldsymbol{b}_{\text{hg}}, \tag{4.46}$$

where:

$$\boldsymbol{J}_{\text{hg}} = \left(\frac{\boldsymbol{K}_m}{T_s} + \boldsymbol{K}_d\right) \boldsymbol{J}(\boldsymbol{q}),$$

and

$$\boldsymbol{b}_{\text{hg}} = -\frac{\boldsymbol{K}_m}{T_s}\ \boldsymbol{v}(t - T_s) - \boldsymbol{h}.$$

4.4 Validation

This section reports the experimental results obtained from the validation campaigns carried out to evaluate the control strategies described in Sections 4.2.3 and 4.3.2. Both strategies have been extensively validated. Specifically, each was first tested in an indoor laboratory environment and then in a real-world scenario.

4.4.1 Human-Safety Validation

This section presents the experimental results obtained during the validation of the safety framework described within in Section 4.2.3.
The robot used for validation is shown in Figure 3.1 and described in Section 3.5. Table 3.1 reports the kinematic constraints applied during the experiments, expressed as upper and lower bounds on joint positions and velocities.
The validation was conducted in two scenarios: *i*) an indoor laboratory environment, where the robot executed a predefined periodic trajectory cooperatively and symmetrically with both end-effectors, and *ii*) an outdoor environment, consisting in a real vineyard, where the robot performed a harvesting operation with a single end-effector. In all the experiments, the robot operated in a shared workspace alongside a human operator, each carrying out distinct tasks. For instance, in precision agriculture settings, the human might either conduct inspections of harvested fruit quality or carry out parallel harvesting activities in the proximity of the robot as it harvests additional grape bunches. In both cases, safety represents a paramount requirement, especially given that in such scenarios robots' end-effectors are usually equipped with scissors. In all the experiments, the inputs modulating the amount of the deviation in terms of orientation are set to zero ($\boldsymbol{\psi}_{a_o} = \mathbf{0}_3$ and $\boldsymbol{\psi}_{r_o} = \mathbf{0}_3$ in Eq. (4.11)), meaning that the Safety Planner will not modify the orientation throughout the experiment. Conversely, the inputs modulating the amount of the deviation in terms of position are defined individually for each experiment and detailed in its description.
Regarding the human point position estimation process, it is managed by the Perception System module shown in Figure 4.2.3, which relies on OpenPose [97], a widely adopted framework for human skeleton keypoints

detection and tracking in the image plane. As in most cases, also in this case, these 2D keypoints, once obtained, are then projected into Cartesian space and converted into 3D coordinates. This process exploits depth information acquired by the sensor associated with the Perception System Module [98]. Then, for the safety field value computation, only three relevant keypoints, namely the chest of the human and the left and right wrists, have been considered. Furthermore, the function χ in Eq. (4.1) has been set as d. Figure 4.4.1 shows the robot alongside the human skeleton recognized by OpenPose within the RViz environment, with the relevant human skeleton keypoints highlighted in red.

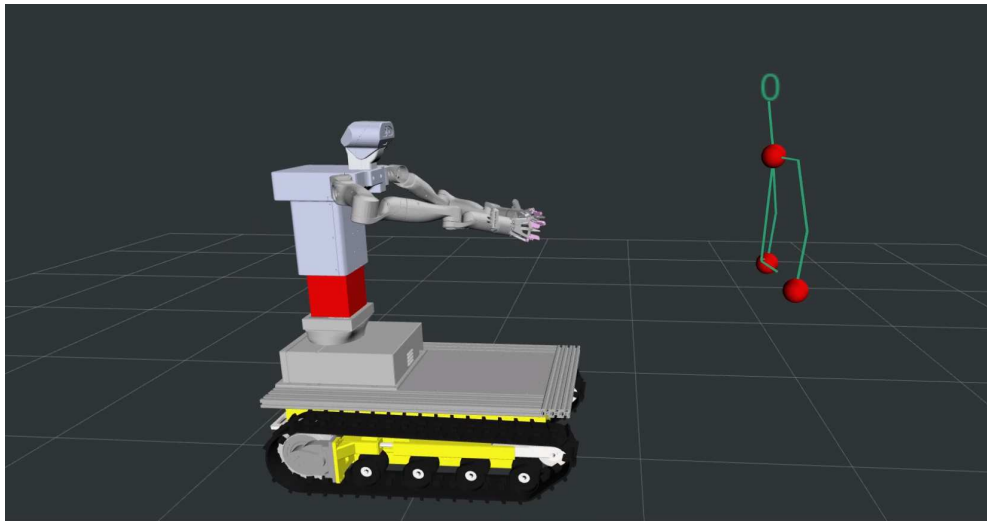


Figure 4.2: The robot alongside the human skeleton in the Rviz environment. The red spheres represent the human points taken into consideration for the safety field value computation.

Finally, the gains adopted in the HQP controller as well as the Safety Planner parameters employed throughout the experiments are summarized in Table 4.1. In this table, the variables $\underline{u}_{\alpha_{(\cdot)}}$ and $\bar{u}_{\alpha_{(\cdot)}}$ are used to denote generic elements of the corresponding vectors $\underline{\boldsymbol{u}}_{\alpha}$ and $\bar{\boldsymbol{u}}_{\alpha}$.

4.4.1.1 Laboratory Experiments

In the following, two indoor experiments are presented, where the robot's end-effectors perform a cooperative motion to track a nominal segment with the absolute frame. A video showing the two experiments is available

CONTROL		SAFETY PLANNER	
Parameter	Equation	Parameter	Equation
$\phi_i = 2$	Eq. (2.19)	$\tau_{(\cdot)} = 2$	Eq. (4.10)
$\boldsymbol{K}_\sigma = \text{diag}\{50\ \boldsymbol{I}_3,\ 20\ \boldsymbol{I}_3\}$	Eq. (4.32)	$k_f = 10$	Eq. (4.25)
		$k_n = 1$	Eq. (4.31)
		$\underline{u}_{\alpha_{(\cdot)}} = -0.4$	Eq. (4.29)
		$\bar{u}_{\alpha_{(\cdot)}} = 0.4$	Eq. (4.29)
		$\underline{F} = 0.75$	Eq. (4.24)

Table 4.1: Control gains and Safety Planner parameters.

at the link [1]. Figure 4.3 shows the laboratory conditions adopted for these experiments.

Figure 4.3: Two snapshots of the robot performing a cooperative motion in laboratory environment. On the left, the operator is far enough from the robot such as the safety field is above the considered threshold. On the right, the operator gets close to the robot making the robot modify the nominal trajectory.

In both the experiments, the motion is performed symmetrically by setting $\alpha = 0.5$ and $\beta = 1$ in Eq. (3.31). In this configuration, the task function $\boldsymbol{\sigma}_a$ represents the absolute pose, whereas $\boldsymbol{\sigma}_r$ represents the relative pose. The difference between the two experiments lies in the different weights assigned to the virtual inputs u_c and $\boldsymbol{u}_\alpha$. The purpose is to highlight the effects of trajectory scaling and path modification on the nominal trajectory separately. In both experiments, the orientation of the absolute frame is preserved at its initial value, the relative position between the end effectors is constrained to reach the following target value

$$\boldsymbol{\sigma}_{r_p,d} = \begin{bmatrix} 0 & -0.30 & 0 \end{bmatrix}^T, \tag{4.47}$$

[1] `https://youtu.be/iWa9UofAXWU`

and the relative orientation is maintained at its initial value. In addition, in both cases, a human operator enters the workspace and approaches the robot while it is engaged in the nominal trajectory tracking. This interaction leads to a reduction in the safety field value, thereby prompting the Safety Planner to deviate from the nominal path and scale down the nominal trajectory. In both experiments, the input $\boldsymbol{\zeta}_{r_p}$ is set to $\mathbf{0}_3$, meaning that the Safety Planner does not modify the relative position between the end-effectors. Conversely, the input related to the absolute position deviation is set to

$$\boldsymbol{\zeta}_{a_p} = \hat{\boldsymbol{n}}_o^a \, u_{\alpha_{a_p}} = \frac{\boldsymbol{p}_{\hbar} - \boldsymbol{\sigma}_{a_p}}{||\boldsymbol{p}_{\hbar} - \boldsymbol{\sigma}_{a_p}||} \, u_{\alpha_{a_p}}, \tag{4.48}$$

where $\hat{\boldsymbol{n}}_o^a$ represents the unit vector that connects the position vectors of the considered human point to that of the absolute reference frame.

In the first experiment, the components of the $\boldsymbol{Q}_\xi$ matrix in Eq. (4.27) are tuned such that the Safety Planner, in maintaining the safety field value above the minimum threshold, prioritizes trajectory deviations rather than trajectory scaling. This is achieved by decreasing the weight associated with the virtual input $u_{\alpha_{a_p}}$ compared to that assigned to u_c. Figure 4.4 shows the obtained results.

Specifically, Figure 4.4 top-left) reports the temporal evolution of the safety field value (solid blue line) compared to the imposed minimum threshold $\underline{F}$ (horizontal red-dashed line). For the first 25 seconds of the experiment, the human operator maintains a certain distance from the robot while it performs the requested movement twice; afterward, he extends his right arm toward the robot, causing the safety field value to decrease. As the robot attempts to reach the waypoints nearest to the human for the third time, the safety field value approaches the imposed threshold, thereby causing the activation of the constraints of the QP problem in Eq. (4.27). At that point, the Safety Planner generates a deviation from the nominal trajectory that guarantees the safety field value remains greater than or equal to the imposed minimum threshold. This is particularly evident in Figures 4.4 top-right) and 4.4 mid-left). In particular, the former shows the nominal (in blue) and safe (in red) trajectories, whereas the latter reports the computed virtual input $u_{\alpha_{a_p}}$ (bottom part) along with the corresponding $\Delta\boldsymbol{\sigma}$ (top part), which

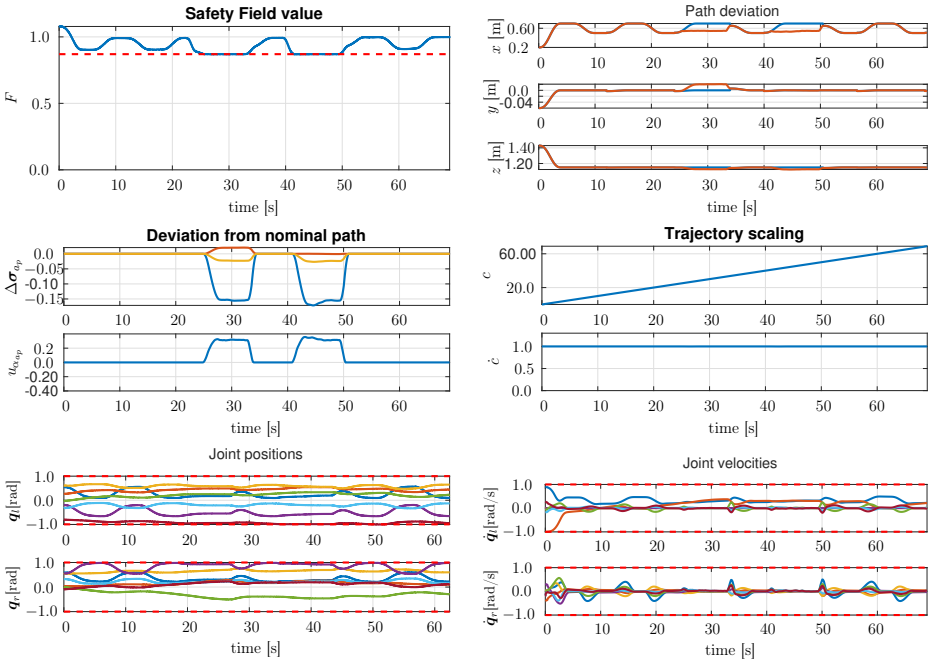


Figure 4.4: First experiment, u_c weighted more than u_α. Top-left) Safety field value (solid blue line) over time compared to the imposed minimum threshold (horizontal red-dashed line). Top-right) Nominal (blue line) and safe (red line) trajectories for the absolute frame. Mid-left) Deviation from the nominal path and virtual input u_α. Mid-right) Parameter c and virtual input u_c. Bottom-left) Normalized joint positions (solid lines) with the imposed minimum and maximum thresholds (red-dashed lines). Bottom-right) Normalized joint velocities (solid lines) with the imposed minimum and maximum thresholds (red-dashed lines).

provides a quantitative measure of the deviation from the nominal trajectory. Conversely, Figure 4.4 mid-right) shows the temporal evolution of the virtual input u_c (bottom part) along with the corresponding c value (top part). This figure confirms that the scaling input, in this case, is barely used by the Safety Planner to raise the safety field value, which is a direct consequence of the relative weight attributed to this virtual input. Finally, Figures 4.4 bottom-left) and Figure 4.4 bottom-right) show the temporal evolution of the normalized joint positions and velocities (solid lines) against their corresponding normalized minimum and maximum thresholds (red-dashed lines). These figures demonstrate

that, despite the circumstances, the HQP controller successfully fulfills the high-priority tasks. Approximately 10 seconds after the previous action, the human operator withdraws the arm, thereby allowing the robot to reach the desired waypoint. This sequence, in which the human extends the right arm toward the robot and subsequently retracts it after 10 seconds, is repeated multiple times throughout the experiment in order to demonstrate and validate the effectiveness of the Safety Planner. It should be emphasized that, throughout the experiment, in addition to the aforementioned terms, the Safety Planner also computes the virtual input $\dot{\boldsymbol{q}}_n$, whose purpose is to generate internal motions that increase the safety field value.

Conversely, in the second experiment, the $\boldsymbol{Q}_\xi$ matrix in Eq. (4.27) is tuned so that the Safety Planner prioritizes trajectory scaling rather than trajectory deviations. This is obtained by assigning a higher weight to the virtual input $u_{\alpha_{ep}}$ compared to that assigned to u_c. The absolute position waypoints and the human operator's actions remain identical to those of the previous experiment. The corresponding results are presented in Figure 4.5.

In this case as well, the safety field value never falls below the imposed minimum threshold, as evidenced in Figure 4.5 top-left), which depicts its temporal evolution (solid blue line) compared to the imposed minimum threshold $\underline{F}$ (horizontal red-dashed line). However, Figure 4.5 mid-right) reveals that, in this case, unlike what occurred during the previous experiment, the Safety Planner relies more on the virtual input u_c. In particular, the figure shows that, to guarantee that the safety field value remains greater than or equal to the imposed minimum threshold, the Safety Planner reduces $\dot{c}$ value until the nominal trajectory is completely halted, which occurs at $t = 27$ s and $t = 47$ s, when $\dot{c} = 0$. Figures 4.5 top-right) and 4.5 mid-left) corroborate the above observations. Specifically, they show that, in this case, the deviation from the nominal path is smaller than in the previous one. Meanwhile, Figures 4.5 bottom-left) and 4.5 bottom-right) show that, in this case as well, the joint kinematic constraints are consistently satisfied over the entire experiment duration.

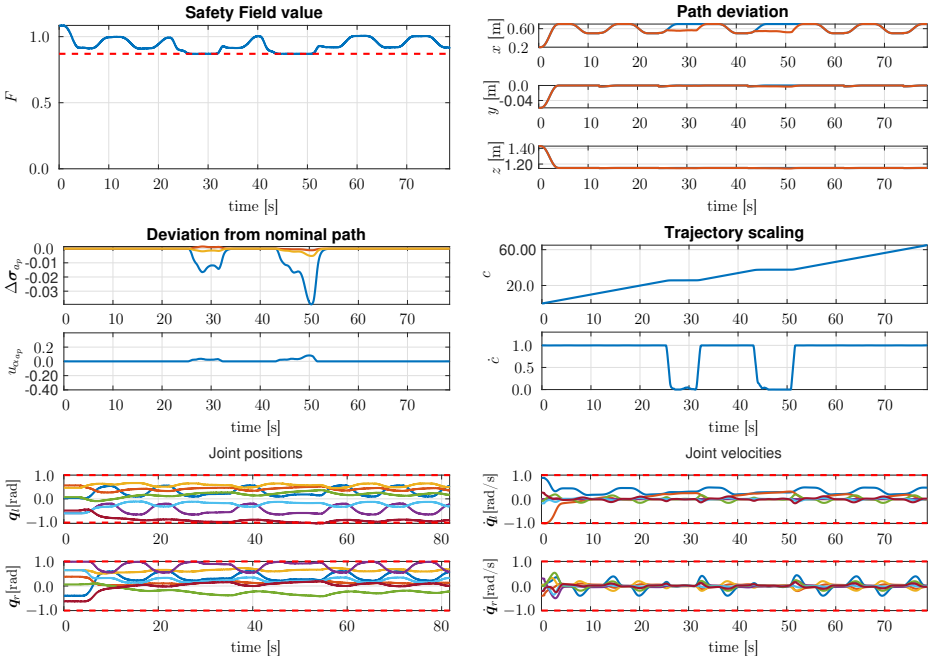


Figure 4.5: Second experiment, u_c weighted less than u_α. Top-left) Safety field value (solid blue line) over time compared to the imposed minimum threshold (horizontal red-dashed line). Top-right) Nominal (blue line) and safe (red line) trajectories for the absolute frame. Mid-left) Deviation from the nominal path and virtual input u_α. Mid-right) Parameter c and virtual input u_c. Bottom-left) Normalized joint positions (solid lines) with the imposed minimum and maximum thresholds (red-dashed lines). Bottom-right) Normalized joint velocities (solid lines) with the imposed minimum and maximum thresholds (red-dashed lines).

4.4.1.2 Outdoor Experiments

The following subsection presents the results of the outdoor validation, obtained from a harvesting experiment conducted during one of the CANOPIES project's experimental campaigns. Although harvesting operations were performed in both single- and dual-arm modes, as documented in the video [23], for the sake of brevity, only the results pertaining to single-arm harvesting are presented here. The cutting position of the

[2] https://youtu.be/w0b_ftCX7iQ
[3] https://youtu.be/8-HFKas2Qho

peduncle to be cut has been obtained in real-time using the perception software developed by one of the project partners [69], which identifies the bunches to be harvested and outputs the corresponding 3D coordinates of the cutting point. This task, similarly to the human skeleton detection and tracking, has been achieved using data collected by the RealSense D435i RGB-D camera embedded in the robot's head.

For this experiment, in order to control the pose of the two end-effectors independently (*uncooperative motion*) and select right end-effector to execute the harvesting task, the parameters in Eq. (3.31) were set as $\alpha = 1$ and $\beta = 0$. Under this configuration, the task function $\boldsymbol{\sigma}_{r}$ describes the right end-effector pose. The corresponding path deviation input $\boldsymbol{\zeta}_{r_p}$ has been set as

$$\boldsymbol{\zeta}_{r_p} = \hat{\boldsymbol{n}}_o^{r} \, u_{\alpha_{r_p}} = \frac{\boldsymbol{p}_{\hbar} - \boldsymbol{\sigma}_{r_p}}{||\boldsymbol{p}_{\hbar} - \boldsymbol{\sigma}_{r_p}||} \, u_{\alpha_{r_p}}, \tag{4.49}$$

where $\hat{\boldsymbol{n}}_o^{r}$ represents the unit vector that connects the position vectors of the considered human point to that of the right end-effector frame.
Regarding the nominal right end-effector position and orientation trajectories, these are produced by the Trajectory Generation module by interpolating a sequence of proper waypoints with trapezoidal velocity-profiled trajectories. The waypoints are the following:

- a pre-grasp configuration, located at a predefined distance from the peduncle to be cut;
- a grasp configuration, which enables the peduncle to be cut and the corresponding bunch to be secured;
- a pre-release configuration, located at a fixed distance from the collection box on the mobile base;
- a release configuration, located directly above the box.

In this experiment, a human operator shares the workspace with the robot while it performs a harvesting operation. The robot tracks the nominal trajectory and continues until the operator approaches, which triggers the activation of the constraint in Eq. (4.30). For this case, the weights related to the two inputs, $u_{\alpha_{r_p}}$ and u_c, were chosen to exploit both contributions. A video showing the experimental validation conducted

under representative conditions is available at the link [4]. In contrast, the experimental validation performed under realistic conditions is available at the link [5]. Figure 4.6 shows some snapshots of the the experimental validation conducted under representative conditions, while Figure 4.7 shows some snapshots of one of the experiments performed under realistic conditions.

Figure 4.6: Snapshots of human-robot safety experiments in representative conditions with robot performing a single-arm harvesting operation. Top Left: results of the grape perception for the planning of the nominal agronomic trajectories. Top Right: the robot starts the agronomic harvesting while the operator is far away from the robot. Bottom Left: the operator get close to the robot while it grasps the harvested grape. Bottom Right: the operator moves away from the robot that releases the grape in the box.

The outcomes of the experiment are presented in Figure 4.8.

Figure 4.8 top-left) shows the evolution of the safety field value (solid blue line) over time compared to the imposed minimum threshold $\underline{F}$ (horizontal red-dashed line). As can be observed, the human operator approaches the robot twice during the experiment: once around $t = 10$ s and again around $t = 50$ s. Specifically, at $t \approx 10$ s, the human approaches the robot for approximately 10 s, leading to a reduction in the safety field value that

[4] `https://youtu.be/Q9D5Dm-KbjE`
[5] `https://youtu.be/ntIx4JtrZ8k`

Figure 4.7: Snapshots of human-robot safety experiments in realistic conditions with robot performing a single-arm harvesting operation. Top figures shows the robot performing the grape harvesting without the human intervention. Bottom figures show the human approaching the harvested grape (and the robot) to taste it.

approaches the imposed minimum threshold. This event mainly results in a deviation from the nominal trajectory, as can be observed in Figures 4.8 top-right) and 4.8 mid-left). Approximately 10 seconds after the initial approach, the human operator steps back, allowing the robot to resume nominal trajectory tracking without modifications. At $t \approx 50$ s, the human approaches the robot again. This time, the virtual input $u_{\alpha_{r_p}}$ reaches its maximum value, and the virtual input u_c is employed to scale down the trajectory. About 20 seconds after the approach, the human operator steps back, allowing the robot to resume the nominal trajectory tracking without modifications.

It should be emphasized that, although the algorithm is designed to maximize the safety field value, it does not guarantee adherence to the minimum threshold. Indeed, during the second approach phase, both the virtual inputs reach their respective limits, causing the safety field value to drop below the imposed minimum threshold. However, even in this case, the human operator's safety is ensured, as $\dot{c} = 0$ implies that the Safety

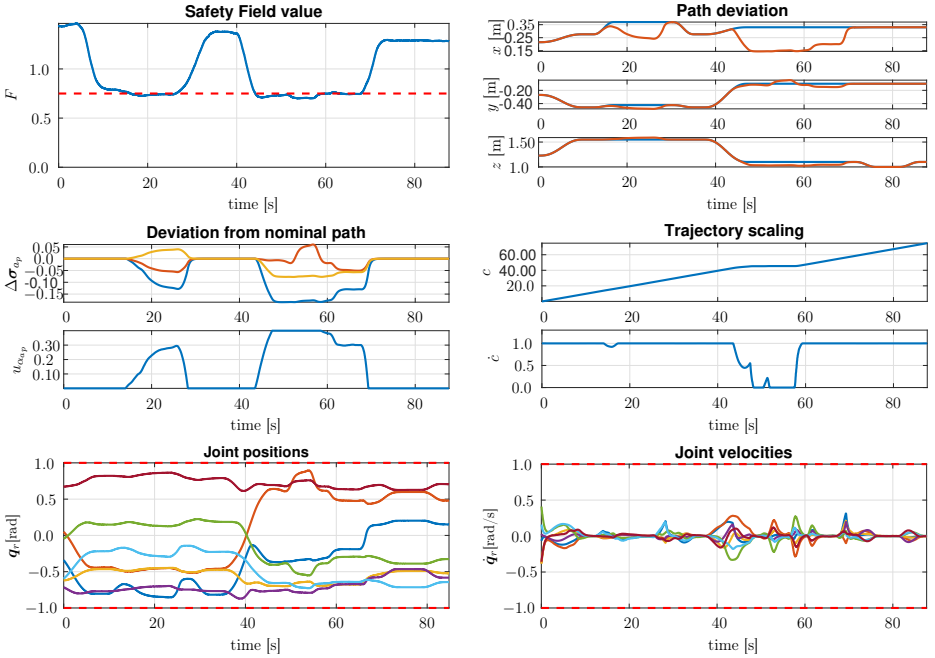


Figure 4.8: Field experiment. Top-left) Safety field value (solid blue line) over time compared to the imposed minimum threshold (horizontal red-dashed line). Top-right) Nominal (blue line) and safe (red line) trajectories for the absolute frame. Mid-left) Deviation from the nominal path and virtual input u_α. Mid-right) Parameter c and virtual input u_c. Bottom-left) Normalized joint positions (solid lines) with the imposed minimum and maximum thresholds (red-dashed lines). Bottom-right) Normalized joint velocities (solid lines) with the imposed minimum and maximum thresholds (red-dashed lines)

Planner suspends the tracking of the nominal trajectory. Figures 4.8 bottom-left) and 4.8 top-right) demonstrate that the joint kinematic limits are fully satisfied during the entire experiment.

4.4.1.3 Comparison with other Null-Space Based Approaches

To demonstrate its effectiveness, the safety algorithm was additionally benchmarked against two baseline strategies: an evasive motion baseline and an emergency stop baseline. In the following, the method presented in this chapter is referred to as A1, the evasive motion baseline as A2, and the emergency stop method as A3.

Algorithm A2 draws on the approach in [87], wherein the control input is modulated as a function of the *danger field*. Here, an analogous policy is applied using the safety field:

$$\dot{\boldsymbol{q}} = m\dot{\boldsymbol{q}}_\sigma + \left[\boldsymbol{I} - m\boldsymbol{J}^\dagger\boldsymbol{J}\right]\dot{\boldsymbol{q}}_0 \tag{4.50}$$

where

$$m = \begin{cases} 1 & \text{if} \quad F \geq (1+\epsilon)\underline{F} \\ 0 & \text{if} \quad F \leq (1-\epsilon)\underline{F} \\ \dfrac{1}{2} + \dfrac{1}{2}\sin\dfrac{\pi(F-\underline{F})}{2\epsilon\underline{F}} & \text{otherwise,} \end{cases} \tag{4.51}$$

$\dot{\boldsymbol{q}}_\sigma$ denotes the joint-velocity command for accomplishing the operational task, which can be computed using the closed-loop inverse kinematics (CLIK) algorithm as $\dot{\boldsymbol{q}}_\sigma = \boldsymbol{J}_\sigma^\dagger(\dot{\boldsymbol{\sigma}}_s + \boldsymbol{K}_\sigma\tilde{\boldsymbol{\sigma}})$, $\dot{\boldsymbol{q}}_0$ corresponds to the gradient of the safety field, and $\epsilon < 1$ is a design constant introduced to guarantee smoothness of the control command.

Algorithm A3 is a simple emergency-stop algorithm wherein the robot's desired trajectory is halted when the value of the safety field drops below a certain threshold $\underline{F}$. Within the framework described in Section 4.2.3.1, this behavior is achievable by adjusting the control input $\boldsymbol{u}_c$ according to the following rule:

$$\boldsymbol{u}_c = \begin{cases} 0 & \text{if} \quad F \leq \underline{F} \\ 1 & \text{if} \quad F > \underline{F} \end{cases} \tag{4.52}$$

while maintaining all the components of the virtual input $\boldsymbol{u}_\alpha$ at zero (i.e., $\boldsymbol{u}_\alpha = \boldsymbol{0}$). The comparison relies on the following metrics: (M1) Average safety field value, (M2) Average path error, defined as the mean deviation from the nominal path, and (M3) Robot idle time, defined as the amount of time during which the robot task is interrupted. All the algorithms have been evaluated under an identical simulation scenario, where the robot was required to periodically reach two absolute-position waypoints connected through a trapezoidal velocity profile, while preserving both its absolute orientation and its initial relative pose. Concurrently, a human

operator moved from the right side to the left side of the robot, causing the safety field value to approach the imposed minimum threshold, which was set to $\underline{F} = 0.9$. It should be emphasized that the mobile base DoFs were also considered in all three simulations. Figure 4.9 reports the obtained results.

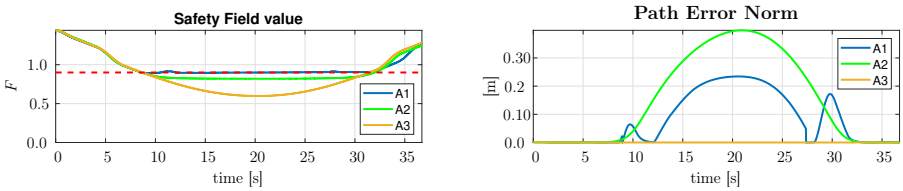


Figure 4.9: Comparison. A1 - approach described in this chapter; A2 - evasive motion; A3 - emergency stop.

Figure 4.9.Top) shows the temporal evolution of the safety field value for the three algorithms. As can be noticed, with Algorithm A3 the safety field reaches the lowest value compared to Algorithm A1 and A2; this is because the human operator gets closer to the robot after it is stopped, and the algorithm does not foresee any deviation from the nominal trajectory. Conversely, Algorithm A2 moves the end-effectors away from the human, leading to better results; however, the safety field still falls below the threshold, which is unavoidable since the method entails no formal constraint on the safety field value in the formulation. Finally, regarding Algorithm A1, it manages to maintain the safety field value above the minimum threshold for the whole experiment, this is due to the constraint on the safety field value included in the corresponding QP formulation. Figure 4.9.Bottom) shows the temporal evolution of the path error norm for the three algorithms. As can be seen, Algorithm A3 introduces no path error, since it exclusively modifies the reference velocity while preserving the nominal path. Concerning the other two approaches, A1 results in smaller deviations from the nominal trajectory than A2. This is motivated by the fact that, compared to A2, it has additional degrees of freedom to exploit, whereas the evasive approach can only modify the path. Table 4.2 presents the values of the evaluation metrics for the three algorithms across the simulations, highlighting that the method proposed in this chapter achieves the lowest robot idle time as well.

ALGORITHM	M1	M2	M3
A1	0.99	0.08	0
A2	0.95	0.0	0
A3	0.86	0.16	62

Table 4.2: Comparison metrics. A1 - approach described in this chapter; A2 - evasive motion; A3 - emergency stop. M1 - average safety field value; M2 - average path error; M3 - robot idle time (%).

4.4.2 Shared-Control Validation

The control strategies described in Section 4.3.2 were extensively validated under both controlled laboratory and real-world conditions. The results of this validation are presented below. Table 3.1 provides the robot kinematic constraints, specified in terms of maximum and minimum positions and velocities, while Table 4.3 reports the adopted values for each parameter and gain, respectively.

PARAMETER	VALUE	EQ.
$\underline{\phi}_{\mathrm{jp},i}$,$\bar{\phi}_{\mathrm{jp},i}$	10	Eq. (3.12)
$\underline{\phi}_{\mathrm{vw},j}$	5	Eq. (3.19)
$\sigma_{\mathrm{vw},j,i}$	$0.3m$	Eq. (3.18)
$\underline{\phi}_{\mathrm{sc},j,i}$	10	Eq. (3.23)
$\sigma_{\mathrm{sc},j,head}$	$0.5m$	Eq. (3.22)
$\sigma_{\mathrm{sc},j,torso}$	$0.35m$	Eq. (3.22)
$\sigma_{\mathrm{sc},j,arm}$	$0.2m$	Eq. (3.22)
$\boldsymbol{K}_{m,l}$,$\boldsymbol{K}_{m,r}$	diag$\{20\boldsymbol{I}_3,\ 3\boldsymbol{I}_3\}$	Eq. (4.38)
$\boldsymbol{K}_{d,l}$,$\boldsymbol{K}_{d,r}$	diag$\{253\boldsymbol{I}_3,\ 27\boldsymbol{I}_3\}$	Eq. (4.39)
$\boldsymbol{K}_{p,l}$,$\boldsymbol{K}_{p,r}$	diag$\{800\boldsymbol{I}_3,\ 60\boldsymbol{I}_3\}$	Eq. (4.40)

Table 4.3: Value of the parameters used for the validation.

4.4.2.1 Laboratory Experiments

In the following sub-section presents the results of two experiments conducted in a controlled indoor environment to validate the strategies outlined in Section 4.3.2. All experiments took place in the LAI Robotics Laboratory of the University of Cassino and Southern Lazio.

The first experiment was designed to validate the admittance control

strategy outlined in Section 4.3.2.1 during simple human–robot interactions, specifically to assess the robot's ability to exhibit compliant behavior with respect to external forces. To this end, the desired trajectory of the left end-effector was defined as maintaining its position fixed at the initial value:

$$\boldsymbol{a}_{l,d} = \mathbf{0}, \quad \boldsymbol{v}_{l,d} = \mathbf{0}, \quad \boldsymbol{\rho}_{l,d} = \begin{bmatrix} \boldsymbol{p}_{l,i} \\ \boldsymbol{o}_{l,i} \end{bmatrix}, \tag{4.53}$$

where $\boldsymbol{p}_{l,i}$ and $\boldsymbol{o}_{l,i}$ represent the initial position and orientation of the left end-effector frame. The experiment also involved a human operator, who was instructed to manually displace the end-effector twice, releasing it after each displacement. The overall hierarchy taken into consideration for this experiment is shown in Figure 4.10.

Figure 4.10: The task hierarchy used to validate the operational task described in Section 4.3.2.1.

A video showing the indoor experimental validation of the admittance control strategy is available at the link [6]. Figure 4.11 shows the key moments of the laboratory experiment.

Figures 4.12 and 4.13 show the obtained results. In detail, Figure 4.12 shows the time evolution of the wrench measured by the wrench sensor mounted at the wrist of the left manipulator, while Figure 4.13 shows the

[6] `https://youtu.be/Fx1ZSfbAGBY`

Figure 4.11: Screenshot of the video showing the admittance control experiment. Top Left: the human operator starts physically interacting with the end-effector. Top Right: the human operator pulls down the end-effector. Bottom Left: the human operator stops interacting with the end-effector. Bottom Right: the end-effector returns to the desired position.

time evolution of the normalized joint positions and velocities (solid lines) with respect to the corresponding normalized minimum and maximum threshold values (dashed red lines).

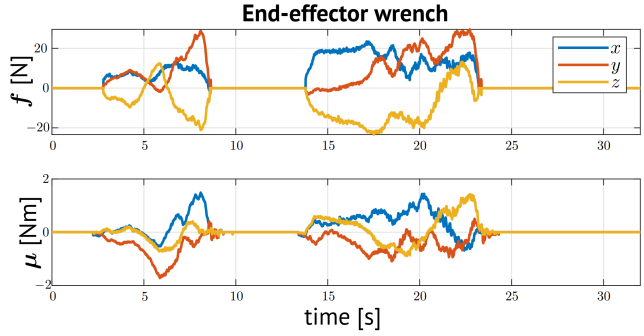


Figure 4.12: Interaction wrench of the end-effector over time during the experiment on admittance control.

The second experiment, in contrast, was aimed to validate the hand-guiding control strategy outlined in Section 4.3.2.2. The evaluation focused on assessing the strategy ability to allow manual adjustment of the

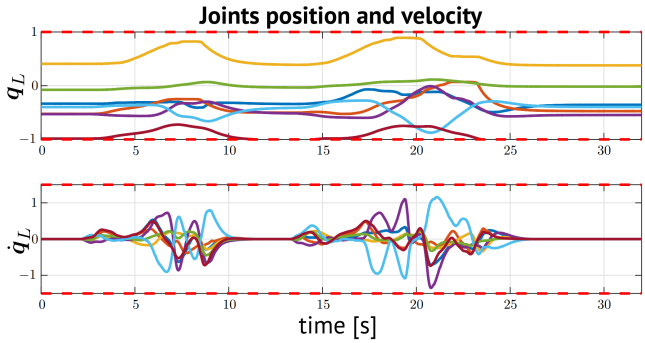


Figure 4.13: Top: normalized joints position over time during the experiment (solid lines) and their thresholds (red dashedlines); Bottom: joints velocity over time during the experiment (solid lines) and their thresholds (red dashed-lines)

arm's configuration. For this purpose, a human operator was instructed to physically interact with the left end-effector, while the robot was initialized in a predefined joint configuration with the hand-guiding strategy enabled exclusively on the left arm and the right arm maintained stationary. The overall hierarchy taken into consideration for this experiment is shown in Figure 4.14.

Figure 4.14: The task hierarchy used to validate the operational task described in Section 4.3.2.2.

A video showing the indoor experimental validation of the hand-guiding

control strategy is available at the link [7]. Figure 4.15 shows the key moments of the laboratory experiment.

Figure 4.15: Screenshot of the video showing the hand-guiding control experiment. Top Left: the human operator gets close to the robot. Top Right: the human operator starts physically interacting with the end-effector. Bottom Left: the human operator pulls up the end-effector. Bottom Right: the human operator stops interacting with the end-effector and it remains still.

Figures 4.16 and 4.17 show the obtained results. Specifically, Figure 4.16 shows the time evolution of the wrench measured by the wrench sensor mounted at the wrist of the left manipulator, while Figure 4.17 shows the time evolution of the normalized joint positions and velocities (solid lines) with respect to the corresponding normalized minimum and maximum threshold values (dashed red lines).

4.4.2.2 Real-World Experiments

This section reports the results of the validation campaign carried out under real-world conditions. The validation consisted of two harvesting experiments, both carried out during the CANOPIES project's experimental campaign.

Regarding the harvesting task, as previously described in Section 3.5.2, in order to harvest a grape bunch effectively, the robot must first detect

[7] `https://youtu.be/He0abs0u0fk`

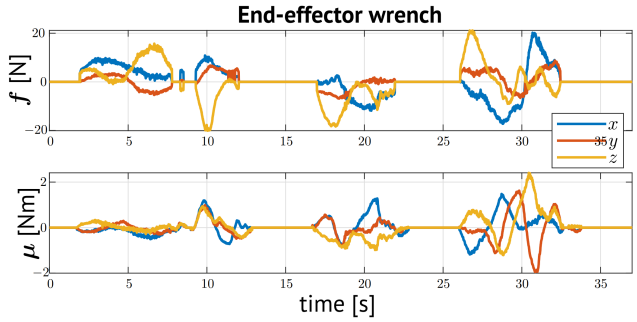


Figure 4.16: Interaction wrench of the end-effector over time during the experiment on hand-guiding control.

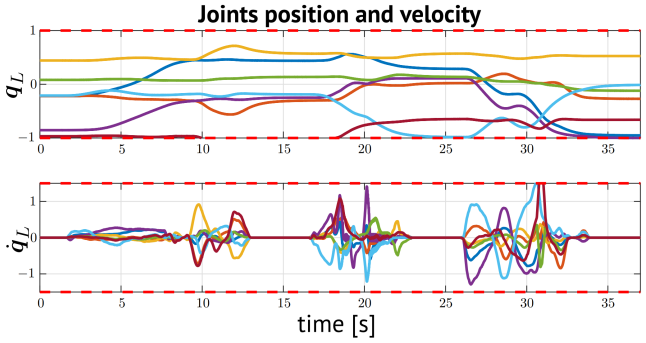


Figure 4.17: Top: normalized joints position over time during the experiment (solid lines) and their thresholds (red dashedlines); Bottom: joints velocity over time during the experiment (solid lines) and their thresholds (red dashed-lines)

the bunches within its surroundings and then accurately estimate the 3D position of the peduncle cut-point of each detected bunch. Once the collected data have been analyzed and the peduncle position has been estimated, the harvesting procedure is started. The Trajectory Generation module then computes the desired end-effector position and orientation trajectories, connecting a sequence of suitable waypoints through trapezoidal velocity profiles. This enables the robot to: *i*) move to a pre-grasp position, i.e., a location at a predefined distance from the peduncle; *ii*) proceed to the grasp position, i.e., move onto the peduncle ; *iii*) close the gripper to secure the bunch; *iv*) actuate the scissors to cut the peduncle; *v*) move to a pre-release position, i.e., at a predefined distance from the box where the grapes must be placed; and *vi*) deposit the bunch into the designated container. It is worth noting that the success of the procedure

strongly depends on the accuracy of the peduncle position estimation, which is particularly difficult to ensure owing to the variability in the distance between the cameras and the grape bunches, as well as occlusions that may be caused by leaves or adjacent bunches. To overcome these issues, the proposed approach involves performing the estimation from multiple viewpoints and exploiting the additional RGB-D sensors available in the system, such as the wrist-mounted ones. Specifically, the proposed approach first involves the processing of data acquired from the head-mounted RGB-D sensor to obtain an initial estimate of the bunches position. Then, if the peduncle of the selected bunch is occluded by leaves or adjacent grapes, or if the bunch is located too far from the head camera for an accurate estimation, one of the end-effectors is moved closer to the selected bunch to refine the peduncle position estimation using the wrist-mounted camera (see Figure 3.1). After this second run of the perception software, the harvesting procedure is started, provided that a reliable estimate of the cut-point of the selected bunch peduncle has been obtained.

However, it is worth noticing that, despite the strategy outlined above, the estimation procedure may nonetheless fail due to the inherent complexity of the task. In particular, occlusions caused by nearby bunches, canes, or leaves may continue to hinder the perception module from detecting the peduncle, even when multiple cameras and viewpoints are employed. In such situations, human intervention is required to complete the harvesting operation. To enhance the effectiveness of the overall harvesting procedure and come with the afore-mentioned challenges, a strategy has been designed to dynamically adjust the system's autonomy level according to the success of the estimation procedure. Specifically, two control modes have been defined: *autonomous* mode and *semi-autonomous* mode.

The *autonomous* mode relies on the control strategy outlined in Section 4.3.2.1 and is employed whenever the perception module is able to provide an accurate estimation of the 3D postion of the peduncle cut-point of the selected grape bunch. Under this mode, the robot autonomously performs all end-effector movements required to accomplish the entire harvesting procedure. Conversely, the *semi-autonomous* mode builds upon the control strategy described in Section 4.3.2.2 and is activated in case of failures of the perception system. Under this mode, partial control of the operation is assigned to a human operator, who is instructed to use

his/her perceptual and cognitive abilities to support the robot in completing the harvesting process. More in detail, whenever the perception module fails to provide an accurate estimation of the 3D position of the peduncle cut-point of the selected grape bunch, the control framework first activates the robot's text-to-speech module to request human intervention and then switches the operational mode from autonomous to semi-autonomous. As a result of this switch, the admittance parameters are adjusted to fit the new mode, in particular, to enable the human operator to grab the end-effector and manually position it at the cutting point. In parallel, the framework retains control over those tasks that are challenging for the operator to handle, including safety constraints and internal configuration management. This division of responsibilities permits the operator to focus solely on the harvesting procedure, while the system oversees all remaining control objectives. Once the operator has completed the assigned task, the robot switches back to autonomous mode and autonomously continues executing the remaining stages of the harvesting procedure.

Figure 4.18 shows a graphical representation of the Finite State Machine implementing the strategy devised to dynamically adjust the system's autonomy level based on the outcome of the estimation procedure.

4.4.2.2.1 Autonomous harvesting

In the following, the results of the experiment designed to validate the autonomous harvesting mode described in the first part of Section 4.4.2.2 are reported. The experiment was conducted in a scenario free of occlusions, where the perception system could reliably estimate the position of the peduncle cut-point of each detected grape bunch. A video showing the outdoor experimental validation discussed in this section is available at the link [8].

According to the FSM in Figure 4.18, the procedure begins with the robot in autonomous mode, executing the perception software to process the data collected by the RGB-D sensor placed in its head (Figure 3.15). After completing the analysis of the collected data, the perception system provides the results of the detection and localization processes to the control architecture, which then commands the robot to bring its right harvesting tool to the pre-grasp position (see Figure 4.19 Top-Right).

[8] `https://youtu.be/w0b_ftCX7iQ`

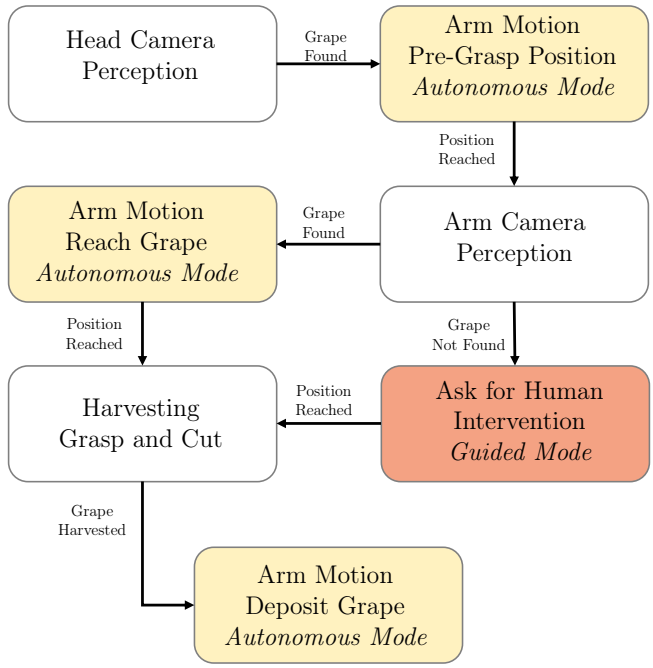


Figure 4.18: State machine describing the sequence of operations for human–robot collaboration in grape harvesting. The colors of the blocks encode the robot control mode: *autonomous* in yellow and *semi-autonomous* in orange. The blocks with a white background are related to robot components different from the manipulators (i.e., RGB-D cameras and end-effectors); thus, they are not affected by the specific control mode.

Then, the robot brings the tool to the grasping position and completes the harvesting procedure by closing the gripper to secure the bunch and actuating the scissors to cut the peduncle. After the grape has been harvested, the robot brings the grape first to the pre-release position (see Figure 4.19 Bottom-Left), and then to the release position (see Figure 4.19 Bottom-Right). Once the grape has been successfully released, the robot returns the tool to its initial configuration (see Figure 4.19 Top-Left).

Figure 4.20 shows the obtained results. Specifically, Figure 4.20.Top) shows the time evolution of the actual (solid blue line) and desired (dashed red line) position of the tool involved in the harvesting procedure, whereas Figure 4.20.Bottom) reports the time evolution of the normalized positions and velocities of the torso and right arm joints (solid lines) with respect to their normalized limit values (dashed red lines).

Figure 4.19: Top Left: image of the robot with both harvesting tools in the home configuration. Top Right: image of the robot with the right harvesting tool in the pre-grasp position. Bottom left: image of the robot with the right harvesting tool in the pre-release position. Bottom right: image of the robot with the right harvesting tool in the release position.

4.4.2.2.2 Semi-autonomous harvesting

In the following, the results of the experiment conducted to validate the semi-autonomous harvesting mode described in the latter part of Section 4.4.2.2 are presented. The experiment was conducted in a scenario where most bunches were occluded by leaves (see Figure 4.21 top-left), which hindered the perception system from accurately estimating the position of the peduncle cut-point.

According to the FSM in Figure 4.18, the procedure begins with the robot operating in autonomous mode, executing the perception software to process the data collected by the RGB-D sensor located in its head (see Figure 4.21 top-left). As shown in Figure 4.21 top-right), which provides an example of the perception software output in dense-vegetation scenarios, dense vegetation and resulting occlusions may not only reduce the number of detected bunches but also hinder the localization of their peduncles. In such cases, the control architecture commands the robot to bring its right end-effector to a pre-grasp position, i.e., a predefined

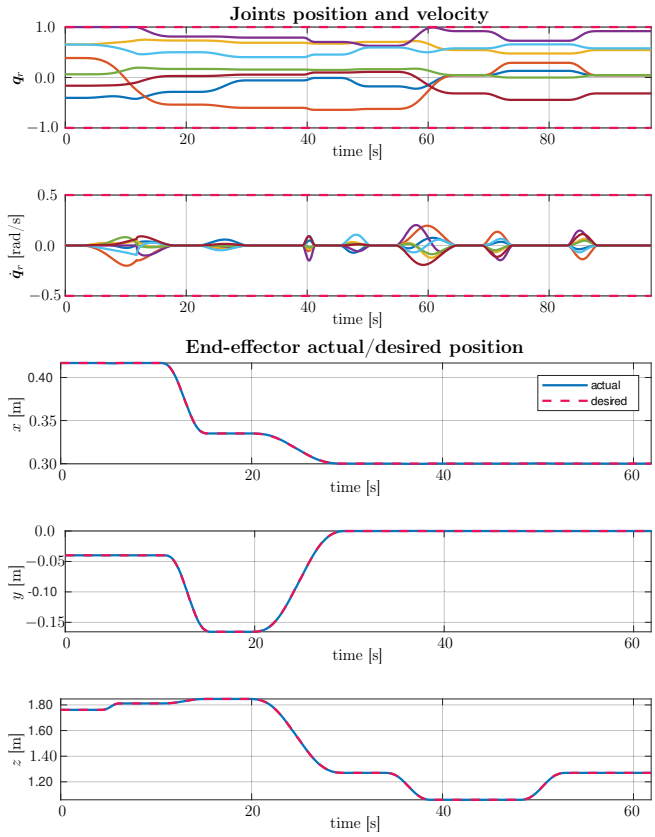


Figure 4.20: Top: time evolution of normalized joint positions (top) and velocities (bottom) of the right arm (solid lines), shown with respect to their normalized limit values (dashed red lines). Bottom: x, y, and z coordinates over time of the actual (solid blue line) and desired (dashed red line) position of the right end-effector.

distance from one of the detected bunches, and then reattempt the detection procedure with the wrist camera (Figure 4.21 bottom-left). However, in these scenarios, even after a second attempt performed from a closer viewpoint, the system may continue to fail to detect the peduncles of the identified bunches. When this occurs, the framework activates the robot's text-to-speech module to request assistance from a nearby human operator and then switches the operational task from admittance to hand-guiding, thereby changing the robot's mode of operation from autonomous to semi-autonomous. As a result of this switch, the admittance

Figure 4.21: Top Left: image of the grapes from the robot's head camera. Top Right: result of the grape recognition and peduncle localization process. Bottom left: images of grapes taken from the right wrist camera. Bottom Right: human operator assisting the robot during the harvesting procedure.

parameters are updated according to this mode, allowing the human operator to grab the end-effector and manually place it at the cutting point (Figure 4.21 bottom-right). As already mentioned, once the operator has completed the assigned task, the robot switches back to autonomous mode and autonomously continues executing the remaining stages of the harvesting procedure.

A video showing the outdoor experimental validation discussed in this section is available at the link [9].

Figure 4.22 shows the obtained results. Specifically, Figure 4.22.Top) shows the time evolution of the interaction wrench detected by the F/T sensor at the right end-effector during the harvesting experiment. As can be observed, between $t \approx 40$ s and $t \approx 60$ s, the human operator applies a force to guide the end-effector toward the cutting point. At the end of this phase, which occurs when when the end-effector is released and no contact is detected for more than 3 s, the robot's mode of operation is reverted

[9] `https://youtu.be/rRs9_ZnKZrE`

to autonomous and the admittance parameters are updated accordingly. From then on, the robot autonomously complete the remaining steps of the harvesting procedure. Specifically, it first closes the gripper to secure the bunch, then actuates the scissors to cut the peduncle (at $t \approx 70$ s), subsequently brings the grape first to the pre-release position, and finally releases it into the collecting box placed at the front part of its mobile base (at $t \approx 130$ s). It is worth noticing that, during this phase, the measured interaction wrench mainly consists of the gravitational force resulting from the grape weight (yellow line in the top plot of Figure 4.22). Figure 4.22.Middle) reports the time evolution of the normalized positions and velocities of the torso and right arm joints (solid lines) with respect to their normalized limit values (dashed red lines). Finally, 4.22.Bottom) shows the time evolution of the x, y, and z coordinates of the actual (solid blue line) and desired (dashed red line) positions of the right end-effector. The cyan-shaded region denotes the interval in which the system operates in *semi-autonomous* mode.

4.5 Conclusions

This chapter presented a comprehensive control framework designed to ensure safe and effective human–robot collaboration in real-world scenarios. The proposed architecture integrates the control scheme introduced in Chapter 3 with two complementary modules: a Safety Planner and a Shared Control Strategy. The Safety Planner guarantees human safety by quantifying it through a dedicated safety field and exploiting this measure to compute optimal trajectory adjustments, either by reducing velocity or deviating from a nominal path, via the solution of a Quadratic Programming (QP) problem. The resulting commands are incorporated into the HQP–CBF control framework, thereby allowing the robotic system to simultaneously execute multiple tasks while safeguarding human operators. In parallel, the Shared Control Strategy enables adaptive autonomy modulation in complex kinematic systems, such as dual-arm mobile robots performing harvesting operations. By dynamically adjusting the robot's admittance parameters according to task conditions estimated by a perception system, the framework allows smooth transitions between autonomous and collaborative behaviors. This mechanism is implemented through two distinct admittance behaviors, formulated as

operational tasks within the HQP framework and coordinated via a finite-state machine that governs their switching. The integrated approaches were extensively validated through experiments on a dual-arm robot in both laboratory and real-field environments. Results demonstrated the flexibility, robustness, and effectiveness of the proposed architecture in balancing human safety and task performance.

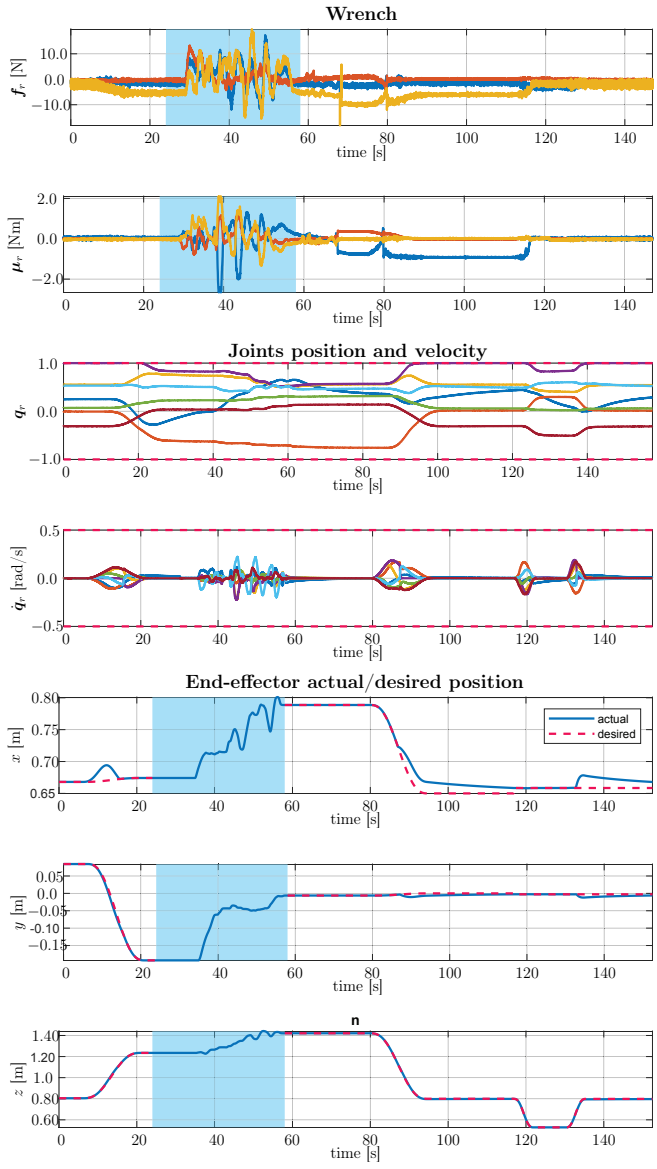


Figure 4.22: Top: time evolution of the interaction wrench detected by the F/T sensor at the right end-effector during the harvesting experiment. Middle: time evolution of normalized joint positions (top) and velocities (bottom) of the right arm (solid lines), shown with respect to their normalized limit values (dashed red lines). Bottom: x, y, and z coordinates over time of the actual (solid blue line) and desired (dashed red line) positions of the right end-effector. The cyan background highlights the phase in which the system is in *semi-autonomous* mode.

Chapter 5

A Human-in-the-Loop Scheduling and Task Allocation Framework for Multi-Human–Multi-Robot Collaboration

5.1 Introduction

Earlier chapters have provided an extensive analysis of the development of low-level control strategies for complex robotic systems, with particular emphasis on aspects concerning single robot–single human interactions, such as ensuring human safety, promoting effective coexistence, and managing physical interactions appropriately. However, the approaches discussed in the preceding chapters may prove insufficient to guarantee a safe and efficient human–robot coexistence, particularly in scenarios involving multiple agents that are required to operate in a coordinated fashion to achieve a shared objective. Therefore, to achieve coherent collective behaviors and enhance the overall system performance, it becomes essential to implement a higher-level control layer dedicated to multi-agent coordination.

As highlighted in Section 1.1, human–multi-robot teams have shown great potential in many settings [14, 15, 16, 99], such as logistics, industrial

manufacturing, healthcare, and precision agriculture. These collaborative systems leverage the unique cognitive strengths of humans and the physical ones of the robots to enhance productivity and operational efficiency [13]. However, significant challenges arise from this collaboration due to the complex social dynamics inherent in human-agent interactions [100, 101]. For instance, the presence of humans in the robot workspace adds complexity to planning and decision-making, as it must account for the dynamic and subjective nature of human behavior [102]. Within this framework, the chapter addresses the challenge of task allocation and scheduling in human–multi-robot teams, as illustrated in Figure 5.1. The strategies and methodologies outlined in this chapter are designed to achieve a two-fold objective: *i*) the minimization of quantitative performance indicators, including makespan (overall completion time), waiting time, and robot energy consumption; and *ii*) the prioritization of human comfort together with the accommodation of individual preferences, which may *evolve over time*. Optimizing these human-specific aspects is essential in human-robot collaboration to ensure socially acceptable and effective interactions [18, 19, 20]. This chapter presents a two-layer framework developed to accomplish the aforementioned objective, specifically tackling the challenge of task allocation and scheduling within human–multi-robot teams. In the first layer, a Mixed-Integer Linear Programming (MILP) optimization problem is formulated for offline task allocation and scheduling, assuming constant parameters of the agents, while explicitly accounting for the stochastic nature of human operation durations. In the second layer, dynamic adjustments to the plan are made to account for online human feedback and changes in the working agents' parameters. In particular, an interactive user interface allows human operators to provide input on high-level factors such as robot velocities when approaching them, distances between the operator and the service robots, and waiting times. Human feedback and measurements are used to adapt MILP parameters and weights online. Based on these updates, the framework determines whether reallocation is required or timing adjustments are sufficient. This feedback-driven approach ensures that task allocation and scheduling remain efficient over time, adapting to evolving conditions and individual human requirements. Moreover, it accounts for inherent human variability, such as differences in perceived time [103] and spatial distances [104]. Finally, to handle large-scale systems, an efficient resolution process is proposed that resorts to Constraint Programming (CP)

and a batch decomposition strategy, mitigating possible computational issues.

MILP has proven to be particularly effective for task allocation and scheduling in collaborative settings, as it enables the specification of *optimal* plans while accounting for multiple constraints, including resource capacities, precedence relations, and temporal dependencies. Such formulations have been widely employed to reduce the makespan in human–robot systems. In this regard, the work in [105] formulates a MILP problem for assigning and scheduling given tasks with multiple robots and a human operator by discretizing the time horizon, which makes the solution inefficient for long-horizon planning or fine discretization settings. Similarly, the study in [106] considers a discretization of the time horizon for scheduling the activities in a single-robot single-human setting. In doing so, ergonomics factors and human fatigue are considered in addition to the makespan. The limitation of the time-horizon discretization is addressed in the MILP formulation presented in [107], which adopts a continuous-time approach. This method introduces additional cost indices that account for workload distribution and quality measures, and it enables human supervision of robot operations when full robot autonomy is not feasible. It is worth noting that the limitation related to time-horizon discretization is also overcome in the work presented in this chapter, where continuous scheduling time instants are considered. MILP formulations have also been applied in the literature to solve balancing problems [108, 109]: in the former, the minimization of the total number of humans and robots is considered as a secondary objective beyond the assignment of tasks to stations, of humans and robots to stations and the scheduling of the tasks; in the latter, the minimization of the number of human workers in an assembly line with a fixed number of stations is addressed by explicitly accounting for the differing capabilities of human and robotic agents. However, in both works, no changes in online human parameters and/or preferences are considered.

The previous works, however, do not address the key challenge posed by the stochastic nature of human behavior. Differently, in [110], a multi-age concurrent Markov Decision Process framework is considered, and a MILP formulation is used to approximate action values for task assignments. However, the approach in [110] does not account for human preferences, particularly their dynamic nature, a further key challenge

in human-robot collaborative settings. The work presented within this chapter addresses this limitation by resorting to chance programming for human stochasticity and explicitly adapting the MILP parameters in response to continuous online human feedback. Alternatively to MILP, in the context of optimization-based approaches, the use of CP formulations has also been explored in [111], where a comparative analysis of these formulations is presented for task allocation and scheduling in human-robot teams, focusing on minimizing makespan in production and logistics tasks for aircraft assembly. Still, no consideration of online human changes is taken into account. As the main drawback of MILP formulations lies in their computational burden, given that these problems are NP-hard, existing approaches have explored different decomposition strategies or hybrid strategies. For instance, in [112], tasks are pre-divided into layers based on precedence constraints, and the optimization objective is to allocate and schedule them to minimize the cycle time of these layers. Similarly, [113] employs a layered task architecture in which task priorities are established according to their dependency hierarchy. However, sub-optimality generally arises from partitioning tasks into layers when handling complex scenarios. In contrast, the framework proposed in this chapter seeks to maintain initial optimality and applies online corrections to preserve it. A related approach is presented in [114], where a Hungarian algorithm is applied for the initial allocation of tasks to agents, followed by a MILP-based scheduling phase. A combination of different methodologies is also proposed in [115], where a hybrid approach combines behavior trees with a MILP formulation to achieve sub-optimal task allocation. This method incorporates general capability costs, accounting for factors such as the agents' kinodynamic properties, task durations, and human ergonomics.

To further reduce the computational burden (at the expense of optimality), additional works in the literature have explored the use of metaheuristic optimization techniques. For instance, genetic algorithms are exploited in [116] and [117], while particle swarm optimization with human supervision is considered in [118]. While these methods are flexible and can adapt to complex environments, they do not guarantee a global optimum. Moreover, human variability and preferences are not considered for online rescheduling and reallocation in the above works. A further common representation used in these collaborative contexts is

the AND/OR graph [119], a tool for modeling task dependencies, where AND nodes require all sub-tasks to be completed, and OR nodes indicate that completing any sub-task is sufficient to proceed. For instance, [120] employs this framework to design a scheduler that dynamically updates online to handle failures, ensuring robust task execution, while [121] utilizes an AND/OR graph to support dynamic role allocation in assembly tasks. At each step, an optimization algorithm identifies the next action and assigns it to the most suitable worker using real-time quantitative metrics. However, no consideration of online human preferences is taken into account in these works.

Finally, a few works in the literature also resort to a data-driven approach and apply Reinforcement Learning (RL) methods for task allocation and scheduling in collaborative settings. In this regard, the study in [122] introduces a Bayesian RL framework for adaptive assembly line balancing in scenarios with one robot and multiple humans by modeling worker task proficiencies using Hidden-parameter Markov Decision Processes. In contrast, the work in [123] presents a chessboard-based framework for human-robot collaborative task scheduling, modeled as a Markov game and optimized using deep Q-network-based multi-agent RL. While RL methods can potentially handle complex scenarios, they generally demand significant training data and can be computationally expensive.

Hence, most existing approaches *i*) neglect human-specific challenges such as behavior variability and evolving preferences; *ii*) are limited in the number of agents they can handle; *iii*) do not necessarily guarantee optimal solutions; *iv*) consider only a limited number of factors in the optimization process, such as makespan; and *v*) offer limited scalability. Moreover, current task allocation and scheduling approaches do not explicitly address the problem setting examined in this chapter. The proposed contribution advances the state of the art by introducing a MILP-based framework developed to achieve *optimal* task allocation and scheduling in human-robot collaborative teams, accommodating *arbitrary* numbers of human and robotic agents and modeling the human inherent stochasticity. Unlike existing methods, the proposed approach accounts for two critical aspects at execution time: *i)* human parameters variability, including differences in task execution times and locations, and *ii)* human preferences that may evolve over time. By addressing these factors, the framework ensures both adaptability and efficiency, enabling it to handle complex

collaborative scenarios with diverse and changing conditions. Additionally, an extension is proposed to address large-scale systems by relaxing the requirement for exact optimality in favor of improved computational traceability.

An example of problems that can be tackled using the above-mentioned framework is presented below.

Problem 1. *Consider a system with a set of working agents $\mathcal{A}$, including human operators in the set $\mathcal{H}$ and working robots in $\mathcal{W}$, performing operations $\mathcal{O}$ in the environment, and a set of service robots which perform services for each operation of the working agents. Assume that each human operator $a \in \mathcal{H}$ is equipped with a device to provide feedback on the perceived quality of service for each operation $\tau^o_{a,i}$. Consider that the durations of the working agents' operations δ^o_a are stochastic. At the same time, the service locations and the human feedback vary over time, requiring the system to adapt accordingly. The objective is to dynamically allocate and schedule the assistance tasks for service robots so that all operations are served while minimizing optimality metrics that account for various aspects of the assistance activities and accommodating time-varying human operators' preferences and parameter variations.*

The rest of the chapter is organized as follows. Section 5.2 introduces the scenario under consideration, detailing both the agents involved and the tasks performed by the team. Section 5.3 presents the proposed framework, outlining its architecture and offering a comprehensive mathematical formulation of the optimization problem designed to address the dynamic allocation and scheduling of assistance tasks. Section 5.4 specifies the quantities monitored in real time and explains how their variations dynamically affect the parameters of the MILP formulation presented in Section 5.3.1. Section 5.5 describes the strategies implemented for the online adjustment of the plan. Section 5.6 examines approaches tailored for large-scale instances, where the MILP formulation may become computationally prohibitive owing to the NP-hardness of the problem. Finally, Sections 5.7, 5.8, and 5.9 describe in detail the simulated and real-world experiments conducted to validate the effectiveness of the proposed framework.

5.2 Scenario

This section introduces the scenario under consideration, detailing both the agents involved and the tasks performed by the team. The notation introduced herein will be employed throughout the remainder of the chapter. Calligraphic capital letters denote sets, non-calligraphic capital letters denote binary variables, while lowercase letters are used otherwise. For a set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, the notation $i \in \mathcal{S}$ refers to the i th element of the set, and $|\mathcal{S}|$ to its cardinality. When the elements of the set are indexed by two indices, such as $\mathcal{S} = \{s_{1,1}, s_{1,2}, \ldots, s_{n,m}\}$, the notation $(i, j) \in \mathcal{S}$ indicates the element $s_{i,j}$. Table 5.1 summarizes the main notation and variables.

5.2.1 Environment

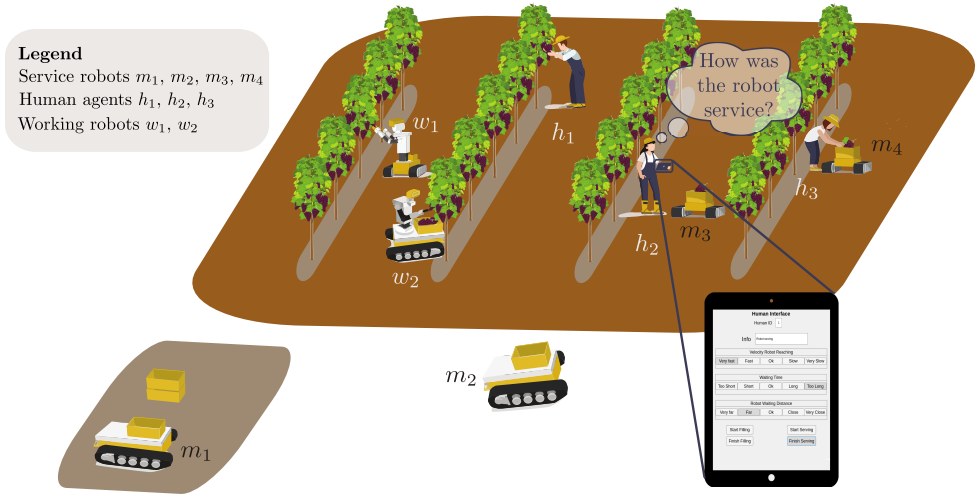


Figure 5.1: Illustration of the collaborative scenario with several working agents and service robots in a vineyard environment.

This chapter addresses a scenario characterized by a collection of heterogeneous working agents (Figure 5.1), $\mathcal{A}$, each engaged in a sequence of ordered operations, together with a set of heterogeneous mobile service robots, $\mathcal{M}$, that support them in their tasks. The set of working agents $\mathcal{A}$ includes both working robots, composing the set $\mathcal{W}$, and human operators, composing the set $\mathcal{H}$; thus, $\mathcal{A} = \mathcal{H} \cup \mathcal{W}$. Let n^m, n^a, n^h, and n^w

VARIABLE	MEANING
$n^m(n^w, n^h, n^a)$	Number of service robots (working robots, human operators, working agents)
$\mathcal{M}, \mathcal{W}, \mathcal{H}, \mathcal{A}, \mathcal{O}$	Sets of service robots, working robots, human operators, working agents and operations of $\mathcal{A}$
x_a^s $(x_{a,m}^w)$	Position for serving (waiting for) the working agent a
x_m^d	Base position of the service robot m
$l_{a,m}^d$ $(l_{a,m}^g)$	Length of the path from x_m^d to x_a^s $(x_{a,m}^w)$
$l_{a,m}^p$	Length of the path from $x_{a,m}^w$ to x_a^s
$v_{\min,m}^c$ $(v_{\max,m}^c)$	Minimum (maximum) speed of the robot m
μ_m	Physical factor of robot m
$\tau_{a,i}^o$	Operation i performed by the agent a
$\tau_{a,i}^g(\tau_{a,i}^w, \tau_{a,i}^p, \tau_{a,i}^s, \tau_{a,i}^d)$	Going (waiting, proximity, serving, depositing) task related to the operation (a, i)
$S_{a,i,m}$	Binary variable for the assignment of the assistance activity (a, i) to the robot m
$\underline{z}_{a,i}$ $(\overline{z}_{a,i})$	Start (end) time of the robotic assistance phase z for the activity (a, i), with $z \in \{g, w^s, p, s, d\}$
$\underline{o}_{a,i}$ $(\overline{o}_{a,i})$	Start (end) time of agent operation (a, i)
δ_a^o	Time for the agent a to perform an operation modeled as $\delta_a^o \sim \mathcal{N}(\varphi_a, \sigma_a)$
δ_m^s, δ_m^d	Time required by robot m to serve an agent, to perform any action at the deposit station
$w_{a,i}^o$	Waiting time for the operation (a, i)
$e_{a,i}^z$	Energy-like term for task z related to activity (a, i)
$v_{a,i,m}^z$	Velocity for task z related to activity (a, i) and performed by robot m
c_M (c_ϵ)	Arbitrary large (small) positive constant
$\alpha, \beta, \gamma, \kappa$	Cost function weights
ρ_a^s, ρ_a^e, $\eta_{a,i}^w$, $\eta_{a,i}^v$	Working severity index, efficiency index, satisfaction index for the waiting time and the velocity
$f_{a,i}^v$ $(f_{a,i}^w)$	Human feedback on the velocity (waiting time)
ι	Relative optimality index
b_s	Batch size
ω^z	Adaptation index related to z (with $z \in \{w, v\}$)

Table 5.1: Main notations introduced in the paper.

denote the numbers of service robots, working agents, human operators, and working robots, respectively. Consequently, $|\mathcal{M}| = n^m$, $|\mathcal{A}| = n^a$ with $n^a = n^h + n^w$, $|\mathcal{H}| = n^h$, and $|\mathcal{W}| = n^w$. In such a setting, distributing service activities among the robots and scheduling them effectively is essential to the overall performance of the system. This is particularly relevant when multiple working agents are involved, as it affects their idle

times and workload balance. The main objective, therefore, is to schedule the service robots' activities so that they provide timely, personalized, and adaptable support.
According to Figure 5.1, the environment comprises two main areas: a depot and a working area. Regarding the depot area, each service robot $m \in \mathcal{M}$ is assigned a specific location in it, referred to as the base position and indicated with x_m^d. The base position represents both the robot's starting location and the point to which it must return after completing a service activity, enabling it to perform subsequent service activities. For instance, the robot can release a load at the depot, ensuring it is ready to pick up additional loads afterward.
Regarding the working area, each working agent $a \in \mathcal{A}$ operates in it and requires services to be performed (at the end of each completed operation) at the designated service location x_a^s. The path length required to reach x_a^s from the depot position of the service robot m is denoted by $l_{a,m}^d$. Additionally, for each agent $a \in \mathcal{A}$, a desired distance $l_{a,m}^p$ is defined, representing the position at which the service robot m is required to wait, if necessary. The associated waiting location, $x_{a,m}^w$, lies at a distance $l_{a,m}^p$ from x_a^s. This constraint ensures that, when the service robot must remain near a working agent until the latter completes an operation, it does not stand too close. Otherwise, the robot's proximity may create discomfort for the human operator and hinder him/her from performing the task effectively and/or safely. Furthermore, the path length required to reach $x_{a,m}^w$ from the depot position of the service robot m is denoted as $l_{a,m}^g$. It is worth noting that working agents can generally change their working location over time. In such cases, the service and waiting locations and the path lengths are updated accordingly. In Figure 5.1, the depot is represented by the grey area, where yellow boxes are released and stored, while the working area is the part in brown comprising the vineyard and working agents.

5.2.2 Operations and Assistance Activities

For each working agent $a \in \mathcal{A}$, an ordered set of operations is defined and denoted by $\mathcal{O}_a = \{\tau_{a,1}^o, \ldots, \tau_{a,q_a}^o\}$, where $\tau_{a,i}^o$ denotes the i-th operation of agent a, and q_a represents the total number of operations performed by agent a. According to this structure, each operation $\tau_{a,i}^o$ must be carried out before $\tau_{a,i+1}^o$, for all $i \in \{1, \ldots, q_a - 1\}$. These sets are contained within

a global set, formally defined as $\mathcal{O} = \{\mathcal{O}_1 \cup \ldots \cup \mathcal{O}_m\}$. In the agricultural case study, which is the one considered in this chapter, an operation consists of filling a box with fruits within an agricultural context. In contexts similar to the one addressed in this chapter, the duration is usually regarded as a stochastic variable. More specifically, it is commonly modeled by means of a Gaussian distribution, $\delta_a^o \sim \mathcal{N}(\varphi_a, \sigma_a)$, where φ_a and σ_a represent the mean and standard deviation associated with agent a. This modeling paradigm is extensively adopted in the literature, as evidenced in [124, 125, 126, 127], and [128].
For each operation $\tau_{a,i}^o \in \mathcal{O}$, a corresponding assistance activity that a service robot m must carry out to enable agent a to perform the subsequent operation $\tau_{a,i+1}^o$ is defined. The assistance activity requires the service robot to reach an area close to the working agent, possibly wait for him/her/it to complete the operation, approach him/her/it and perform the service, and finally return to the depot. In the case study considered in this chapter, the service consists of replacing a box filled with harvested fruits with an empty one, thereby enabling the working agent to continue the harvesting mission. Accordingly, the assistance activity can be decomposed into five consecutive phases, hereafter referred to as tasks:

1. a *going* phase $\tau_{a,i}^g$, where the service robot, starting from its depot, navigates the field to go close to the working agent a in the position $x_{a,m}^w$. Start and final times of this phase are denoted as $\underline{g}_{a,i}$ and $\overline{g}_{a,i}$, respectively;

2. a *waiting* phase $\tau_{a,i}^w$, where the robot waits in $x_{a,m}^w$ for the end of i-th operation of the agent a to carry out the service. Start and final times of this phase are denoted as $\underline{w}_{a,i}^s$ and $\overline{w}_{a,i}^s$, respectively;

3. a *proximity* phase $\tau_{a,i}^p$, where the robot approaches the agent to reach the service location x_a^s. Start and final times of this phase are denoted as $\underline{p}_{a,i}$ and $\overline{p}_{a,i}$, respectively;

4. a *serving* phase $\tau_{a,i}^s$, where the robot performs the service for the working agent, with a minimum duration equal to δ_m^s. Start and final times of this phase are denoted as $\underline{s}_{a,i}$ and $\overline{s}_{a,i}$, respectively;

5. a *depositing* phase $\tau_{a,i}^d$, where the robot returns to the base position at the depot and performs any conclusive action with duration δ_m^d,

if needed (e.g., releasing a load at the depot). The start and final times of this phase are denoted as $\underline{d}_{a,i}$ and $\overline{d}_{a,i}$, respectively.

Figure 5.2 shows all the phases of the assistance activity carried out by the service robot m in support of the working agent a.

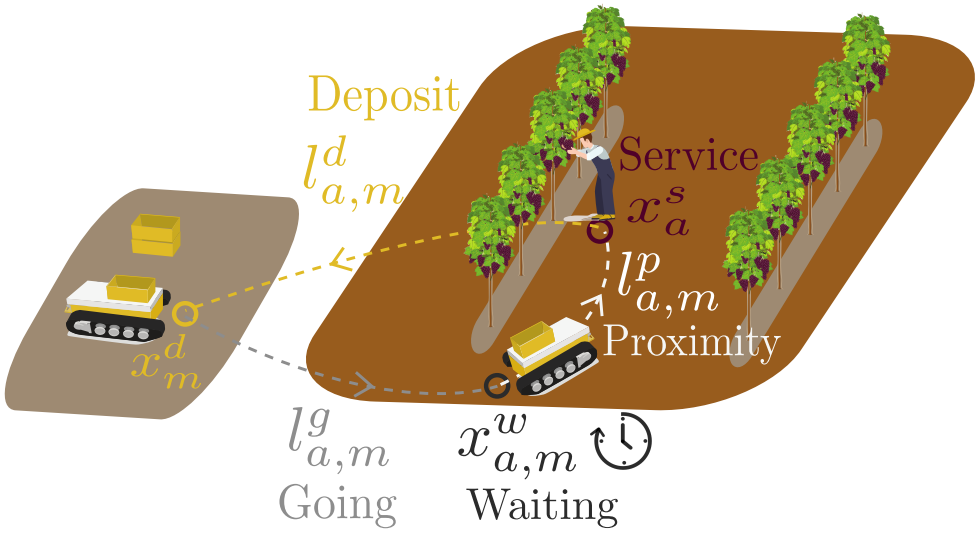


Figure 5.2: Illustration of the phases of an assistance activity.

5.2.3 Service robots and working agents

For each service robot m, both the minimum and maximum cruising velocities required to traverse the field are defined. Specifically, they are denoted as $v^c_{\min,m}$, with $v^c_{\min,m} > 0$, and $v^c_{\max,m}$, with $v^c_{\max,m} \geq v^c_{\min,m}$, respectively, and may differ among robots. The maximum velocity is constrained by the robot's physical capabilities, whereas the minimum velocity depends not only on these physical limitations, such as overcoming high friction on challenging terrain, but also on the necessity to reduce the robot's presence within the working area. Limiting the time spent by service robots in the working area is thus essential to avoid interference with the operations of both working agents and other service robots. Accordingly, the global minimum and maximum allowed velocities across all service robots can be formally expressed as $v^c_{\min} = \min_{m \in \mathcal{M}} v^c_{\min,m}$ and $v^c_{\max} = \max_{m \in \mathcal{M}} v^c_{\max,m}$, respectively. For the proximity phase involving human operators $a \in \mathcal{H}$, additional velocity constraints that depend on

the specific human operator are imposed. These bounds are denoted as $v^p_{\min,a,m}$ and $v^p_{\max,a,m}$, respectively. It follows that $v^p_{\min,a,m} \geq v^c_{\min,m}$, and $v^p_{\max,a,m} \leq v^c_{\max,m}$, $\forall a \in \mathcal{H}$. The rationale for implementing different velocities during the proximity phase is to give the human operator greater control over the robot's behavior, particularly its velocity, when operating nearby. Notably, this is crucial, as the robot's velocity directly impacts the human operator's sense of safety and comfort. In the case study discussed in this chapter, the proximity velocities are initialized as $v^p_{\min,a,m} = 0.8v^c_{\min}$ and $v^p_{\max,a,m} = 0.75v^c_{\max}$. Further details regarding the online update of these bounds are provided in Section 5.4. For each service robot m, a positive parameter μ_m, hereafter referred to as *physical factor*, is defined to account for the robot's intrinsic physical properties, including mass and size. The larger μ_m is, the more physically massive the robot. Moreover, it is assumed that a low-level controller is available, ensuring that robot m can reliably track prescribed plans. Regarding the human operators, for each operator $a \in \mathcal{H}$, the following indices are defined

1. an *Efficiency Index* $\rho^e_a \in [0, 1]$, with 1 denoting the highest efficiency level and 0 the lowest within the human counterpart of the team. This index can, for instance, be defined as

$$\rho^e_a = \frac{\varphi_a}{\min_{a \in \mathcal{H}} \varphi_a}.$$

   In the case study discussed in this chapter, this index represents the ratio between the box-filling time of human agent a and the minimum box-filling time observed among all humans in the set $\mathcal{H}$;

2. a *Working Condition Severity Index* $\rho^s_a \in [0, 1]$, where 1 corresponds to maximum discomfort. This index may, for example, account for ergonomic strain, exposure to direct sunlight, or variations in perceived temperature.

The indices, along with task duration and service location, are assumed to be measurable. Although their estimation is beyond the scope of this chapter, it can be addressed by leveraging existing methodologies, such as those reported in [129] and [130]. Furthermore, it is assumed that each human operator is equipped with a device capable of reporting feedback on the perceived quality of service, thus facilitating the adaptive

mechanisms outlined in Section ??.

5.3 Optimization Framework

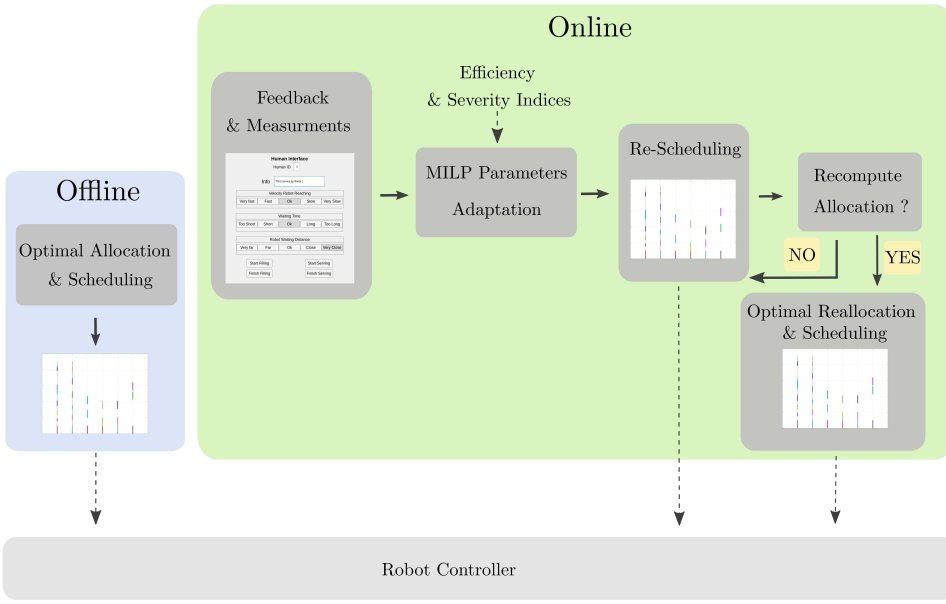


Figure 5.3: Schematic illustration of the framework.

In order to solve Problem 1 and to perform the dynamic allocation and scheduling of assistance tasks, a two-layer framework has been designed (Figure 5.3). In the first layer, a MILP optimization problem is formulated for offline allocation and scheduling, where the optimal solution is computed using the initial parameters of the working agents. To handle uncertainty in task durations, chance-constrained programming is adopted, while the remaining parameters are treated as constant. In the second layer, the plan is dynamically adjusted to accommodate human feedback and to respond to variations in the parameters of the working agents. To this end, the framework integrates a monitoring module that continuously tracks agent activities and gathers human feedback, which is then employed to update the parameters of the MILP formulation. The framework then determines whether task reallocation is necessary, considering both temporal adjustments and reassignment of tasks to service robots. If reallocation is not required, temporal adjustments are

performed through an updating module to ensure compliance with the imposed constraints. If reallocation is required, the MILP problem is resolved with the updated parameters, typically resulting in changes to both task timings and robot assignments, thus producing a plan that is better aligned with the evolving needs of the human operators. The second-layer iterative procedure is carried out until the completion of all operations performed by the working agents, thereby ensuring continuous adaptation to human feedback and delivering more tailored support to the operators. For large-scale scenarios, an efficient solution approach combining CP and batch decomposition is introduced in Section 5.6. This approach is based on iteratively computing allocation and scheduling while keeping the problem dimension fixed, thereby ensuring that the associated computational complexity remains fixed and tractable.

Before introducing the MILP-based formulation and presenting a formal solution to Problem 1, it is necessary to introduce the relevant decision variables. Specifically, let $S_{a,i,m}$ be a binary decision variable that equals 1 if service robot m executes the assistance activity associated with operation $\tau^o_{a,i}$, and 0 otherwise. Within this framework, the *task allocation* objective consists in determining the set of binary decision variables $S_{a,i,m}$, $\forall (a,i) \in \mathcal{O}$, $m \in \mathcal{M}$. The *task scheduling* objective, on the other hand, requires specifying the continuous decision variables related to the start and completion times of each task, namely $\underline{g}_{a,i}$, $\underline{w}^s_{a,i}$, $\underline{p}^s_{a,i}$, $\underline{s}_{a,i}$, $\underline{d}_{a,i}$, $\underline{o}_{a,i}$, $\overline{g}_{a,i}$, $\overline{w}^s_{a,i}$, $\overline{p}^s_{a,i}$, $\overline{s}_{a,i}$, $\overline{d}_{a,i}$, and $\overline{o}_{a,i}$.

5.3.1 MILP-based Formulation

This section presents the MILP optimization problem for task allocation and scheduling, which involves defining the binary variables $S_{a,i,m}$ and the continuous variables representing the start and completion times of each task and operation. Before presenting the formulation, the optimality metrics relevant to the context under investigation in this chapter are formally introduced. Specifically, these include:

- Waiting time of the working agents, whose reduction enables continuity of operations with minimal interruptions.
- Energy consumption of the service robots, whose minimization enhances efficiency and extends operational capacity.

- Overall makespan, whose reduction shortens the total completion time of all tasks in the system.

Let the scaled makespan be denoted as Δ and defined as

$$\Delta = \frac{\max_{(a,i)\in\mathcal{O}} \overline{d}_{a,i}}{|\mathcal{O}|(l^d_{\max} + l^g_{\max} + l^p_{\max})/v^c_{\min} + |\mathcal{O}|(\delta^s_{\max} + \delta^d_{\max})},$$

where this definition accounts for the fact that the last task is always a depositing one, and a scaling denominator is introduced to ensure the comparability of the factors. Specifically, $l^d_{\max}$,$l^g_{\max}$, and $l^p_{\max}$ represent the longest distances for the depositing, going, and proximity phases, respectively; $v^c_{\min}$ denotes the minimum cruising velocity within the team; and $\delta^s_{\max}$ and $\delta^d_{\max}$ indicate the maximum durations of the service and depositing phases, respectively.

Let the scaled waiting time associated with the operation $\tau_{a,i}$ be denoted as $w^o_{a,i}$ and defined as

$$w^o_{a,i} = \frac{\left(\underline{o}_{a,i+1} - \overline{o}_{a,i}\right)}{(l^g_{\max} + l^p_{\max})/v^c_{\min}} \quad \text{with } i > 1,$$

where the denominator corresponds to the maximum time required to reach the working agent a from any deposit station and acts as a scaling factor. Moreover, the average cruising velocities $v^g_{a,i,m}, v^p_{a,i,m}, v^d_{a,i,m}$ are introduced to formally characterize the motion of service robot m during the execution of the assistance tasks $\tau^g_{a,i}$, $\tau^p_{a,i}$ and $\tau^d_{a,i}$, respectively, and are defined as follows

$$\begin{aligned} v^g_{a,i,m} &= \frac{l^g_{a,m}}{\overline{g}_{a,i} - \underline{g}_{a,i}}, \qquad v^p_{a,i,m} = \frac{l^p_{a,m}}{\overline{p}_{a,i} - \underline{p}_{a,i}}, \\ v^d_{a,i,m} &= \frac{l^d_{a,m}}{\overline{d}_{a,i} - \sum_{m\in\mathcal{M}} S_{a,i,m}\delta^d_m - \underline{d}_{a,i}}. \end{aligned} \tag{5.1}$$

On the basis of these quantities, the energy consumption associated with each phase $z \in \{g, p, d\}$ of the assistance task $\tau^z_{a,i}$, which involves robotic motion, can be formally defined. To this purpose, the variable $\Delta^z_{a,i}$ is

introduced, denoting the duration of the motion in phase z of the assistance task $\tau^z_{a,i}$ (i.e., the denominators in Eq. (5.1)). Following the physical modeling of energy consumption for mobile robots proposed in [131] and [132], the power consumption $p^z_{a,i,m}$ for each robot m performing the task $\tau^z_{a,i}$ can be modeled as

$$p^z_{a,i,m} = \varrho_{1m} + \varrho_{2m} v^z_{a,i,m} + \varrho_{3m} v^{z\,2}_{a,i,m} \tag{5.2}$$

where ϱ_{1m} denotes the static component, associated with computing units, sensors, communication devices, and internal electrical losses, and is typically considered to exhibit negligible fluctuation. The velocity-dependent terms capture the effects of friction and inertia, characterized by the constants ϱ_{2m} and ϱ_{3m}, respectively, incorporating the physical factor μ_m. Assuming that each robot moves at a constant cruising velocity, and according to Eq. (5.1), the total energy spent to travel a distance $l^z_{a,m}$ can be derived as

$$\begin{aligned} e^z_{a,i,m} &= \int_0^{\Delta^z_{a,i}} p^z_{a,i,m} dt = p^z_{a,i,m} \Delta^z_{a,i} \\ &= \varrho_{1m} \Delta^z_{a,i} + \varrho_{2m} l^z_{a,m} + \varrho_{3m} v^z_{a,i,m} l^z_{a,m}. \end{aligned} \tag{5.3}$$

The expression is linear in both the task duration $\Delta^z_{a,i}$ and the robot velocity $v^z_{a,i}$. Moreover, $l^z_{a,m}$ remains constant once specified the task z, the robot m, and the agent a. Based on Eq. (5.3), the aggregated energy-like term associated with each assistance task z ($z \in \{g, p, d\}$) can be formally defined as

$$e^z_{a,i} = \frac{\sum_{m \in \mathcal{M}} S_{a,i,m} e^z_{a,i,m}}{\max\limits_{m \in \mathcal{M}} \left(\varrho_{1m} \dfrac{l^z_{\max}}{v^c_{\min}} + \varrho_{2m} l^z_{\max} + \varrho_{3m} v^c_{\max} l^z_{\max} \right)}, \tag{5.4}$$

where the denominator serves as a scaling term.

Based on the quantities introduced above, the optimization problem formulated in this chapter aims to minimize the following cost function

$$C = \sum_{\substack{(a,i)\in\mathcal{O},\\ a\in\mathcal{H}}} \alpha_{a,i}\, w^o_{a,i} + \beta \sum_{\substack{(a,i)\in\mathcal{O},\\ a\in\mathcal{W}}} w^o_{a,i} + \\ + \frac{\gamma}{3} \sum_{(a,i)\in\mathcal{O}} \left(e^g_{a,i} + \zeta_{a,i} e^p_{a,i} + e^d_{a,i}\right) + \kappa\Delta \tag{5.5}$$

where $\alpha_{a,i}$, $\zeta_{a,i}$, with $(a,i) \in \mathcal{O}$, β, γ and κ are positive weights. The cost function is thus formulated as a weighted sum comprising: *i*) the waiting times $w^o_{a,i}$, distinguished between human and robotic agents; *ii*) the energy terms related to the service robots $e^z_{a,i}$, with $z \in \{g,p,d\}$ and the proximity-phase energy terms scaled by the coefficients $\zeta_{a,i}$; and *iii*) the makespan Δ. The relative importance/contribution of each term is established through the assigned weight. In this regard, as outlined in Section 5.4.2, the weights $\alpha_{a,i}$ and $\zeta_{a,i}$ are updated adjusted over time on the basis of human operators' feedback, thereby ensuring a more effective alignment with their needs.

In order to ensure appropriate task allocation and scheduling, the following constraints are taken into account

1. *Assignment of the assistance tasks:*

$$\textstyle\sum_{m\in\mathcal{M}} S_{a,i,m} = 1, \quad \forall (a,i) \in \mathcal{O}. \tag{5.6}$$

   This equality guarantees that every operation (a,i) associated with a working agent is assigned to exactly one service robot.

2. *Duration of the going tasks:*

$$\overline{g}_{a,i} - \underline{g}_{a,i} \geq \sum_{m\in\mathcal{M}} S_{a,i,m} \frac{l^g_{a,m}}{v^c_{\max,m}}, \tag{5.7a}$$

$$\overline{g}_{a,i} - \underline{g}_{a,i} \leq \sum_{m\in\mathcal{M}} S_{a,i,m} \frac{l^g_{a,m}}{v^c_{\min,m}}, \tag{5.7b}$$

   $\forall (a,i) \in \mathcal{O}$. These inequalities ensure that the duration of each going task $\tau^g_{a,i}$ is consistent with the speed limitations of the robots. Specifically, the first inequality requires that the scheduled duration of the going task $\tau^g_{a,i}$, i.e., $\overline{g}_{a,i} - \underline{g}_{a,i}$, is no less than the minimum

travel time for the assigned robot m (i.e., with $S_{a,i,m} = 1$) to cover the distance $l^g_{a,m}$ at its maximum admissible speed $v^c_{\max,m}$, under the assumption of constant travel velocity. The second inequality, by contrast, constrains the scheduled duration not to exceed the maximum time required by the assigned robot m to cover the distance $l^g_{a,m}$ at its minimum speed $v^c_{\min,m}$. It is worth noting that robots with $S_{a,i,m} = 0$ do not influence the duration of the task.

3. *Duration of the proximity tasks:*

$$\overline{p}_{a,i} - \underline{p}_{a,i} \geq \sum_{m \in \mathcal{M}} S_{a,i,m} \frac{l^p_{a,m}}{v^p_{\max,a,m}}, \tag{5.8a}$$

$$\overline{p}_{a,i} - \underline{p}_{a,i} \leq \sum_{m \in \mathcal{M}} S_{a,i,m} \frac{l^p_{a,m}}{v^p_{\min,a,m}}, \tag{5.8b}$$

$\forall (a,i) \in \mathcal{O}$. Analogous to the constraints in Eq. (5.7), these inequalities enforce that the scheduled duration of each proximity task $\tau^p_{a,i}$ respects the velocity range of the assigned robot m (with $S_{a,i,m} = 1$), namely $v^p_{\max,a,m}$ and $v^p_{\min,a,m}$. As discussed in Section 5.2.3, the proximity velocity limits are always consistent with the robot's physical capabilities, namely $v^p_{\max,a,m} \leq v^c_{\max,m}$, $v^p_{\min,a,m} \geq v^c_{\min,m}$ and $v^p_{\max,a,m} > v^p_{\min,a,m}$, $\forall a, m$.

4. *Duration of the depositing tasks:*

$$\overline{d}_{a,i} - \underline{d}_{a,i} \geq \sum_{m \in \mathcal{M}} S_{a,i,m} \left(\frac{l^d_{a,m}}{v^c_{\max,m}} + \delta^d_m \right), \tag{5.9a}$$

$$\overline{d}_{a,i} - \underline{d}_{a,i} \leq \sum_{m \in \mathcal{M}} S_{a,i,m} \left(\frac{l^d_{a,m}}{v^c_{\min,m}} + \delta^d_m \right), \tag{5.9b}$$

$\forall (a,i) \in \mathcal{O}$. These constraints, like the previous ones, bound the duration of the depositing tasks $\tau^d_{a,i}$, accounting for both the velocity limitations of the assigned service robots and the distance to be traversed. The task duration further accounts for the final depositing action, denoted as δ^d_m. As detailed in Section 5.2.2, in the case study considered in this chapter, δ^d_m denotes the time required by service robot m to exchange a filled box with an empty one at the depot.

5. *Order of the assistance tasks:*

$$\underline{w}^s_{a,i} = \overline{g}_{a,i}, \qquad \underline{p}_{a,i} = \overline{w}^s_{a,i}, \tag{5.10a}$$

$$\underline{s}_{a,i} = \overline{p}_{a,i}, \qquad \underline{d}_{a,i} = \overline{s}_{a,i}, \tag{5.10b}$$

$\forall (a,i) \in \mathcal{O}$. These constraints are formulated to enforce the correct sequencing of the assistance phases, ensuring that the end time of one phase coincides with the start time of the subsequent phase. Specifically, for each operation $\tau^o_{a,i}$, the equations establish that the waiting phase begins immediately after the conclusion of the going phase (first equation), which is then followed by the proximity phase (second equation), the service phase (third equation), and, ultimately, the depositing phase (fourth equation).

6. *Timing of service tasks:*

$$\overline{s}_{a,i} = \underline{s}_{a,i} + \textstyle\sum_{m \in \mathcal{M}} S_{a,i,m} \delta^s_m, \tag{5.11a}$$

$$\underline{s}_{a,i} \geq \overline{o}_{a,i}, \tag{5.11b}$$

$\forall (a,i) \in \mathcal{O}$. These constraints regulate the timing of service tasks associated with each operation $\tau^o_{a,i}$. In particular, the first constraint defines the task duration as a function of the service time δ^s_m of the assigned agent m, whereas the second guarantees that the initiation of the task occurs exclusively after the completion of the operation.

7. *Sequence of operations of working agents:*

$$\underline{o}_{a,1} = t_0, \qquad \forall a \in \mathcal{A} \tag{5.12a}$$

$$\underline{o}_{a,i} \geq \overline{s}_{a,i-1}, \qquad \forall (a,i) \in \mathcal{O}, \tag{5.12b}$$

$$\overline{o}_{a,i} - \underline{o}_{a,i} \geq \sqrt{2}\sigma_a \mathrm{erf}^{-1}(2\,pr_{\min} - 1) + \varphi_a, \qquad \forall (a,i) \in \mathcal{O} \tag{5.12c}$$

where t_0 represents the time at which the optimal solution is determined, $\mathrm{erf}(\cdot)$ denotes the error function, and $pr_{\min} \in (0,1)$ represents the required minimum probability. These constraints formalize the temporal structure of the operations assigned to each working agent $a \in \mathcal{A}$. Specifically: *i*) constraint (5.12a) establishes that the start time of the initial operation aligns with the moment the optimal solution is computed, This is motivated by the fact that, since at the beginning the working agents are in the environment, it

is reasonable for them to start executing their operations immediately; *ii*) constraint (5.12b) enforces that each operation can start only after the previous service has completed; and *iii*) constraint (5.12c) models operation durations through a chance-constrained approach [133], thereby capturing the stochastic nature of execution times. In this regard, it can be shown that the constraint is equivalent to enforcing

$$\Pr(\overline{o}_{a,i} - \underline{o}_{a,i} \geq \delta_a^o) \geq pr_{\min}, \qquad \forall (a,i) \in \mathcal{O}, \tag{5.13}$$

meaning that, with probability no less than $pr_{\min}$, the duration allocated to each operation $(a,i) \in \mathcal{O}$ is sufficient for its completion. A detailed explanation is available in [133].

8. *Execution of one task at a time:*

$$\underline{z}_{b,k} - \overline{u}_{a,i} \geq -c_M(2 - S_{a,i,m} - S_{b,k,m}) - c_M(1 - Q^{uz}_{a,i,b,k,m}), \tag{5.14a}$$

$$\overline{z}_{b,k} - \underline{u}_{a,i} \geq -c_M(2 - S_{a,i,m} - S_{b,k,m}) - c_M\, Q^{uz}_{a,i,b,k,m}, \tag{5.14b}$$

$\forall (a,i),(b,k) \in \mathcal{O}$, $m \in \mathcal{M}$ with $(a,i) \neq (b,k)$, where c_M is an arbitrarily large constant, $Q^{uz}_{a,i,b,k,m} \in \{0,1\}$ is an auxiliary binary decision variable, and $u, z \in \{g, w^s, p, s, d\}$. This notation indicates that, for example, when $u = g$, the variable $\underline{u}_{a,i}$ coincides with $\underline{g}_{a,i}$. The inequalities (5.14a)-(5.14b) guarantee that each service robot m executes only one task at a time, meaning it cannot assist multiple working agents simultaneously. To explain this, consider the case in which a service robot m is assigned to support two operations, (a,i) and (b,k) (i.e., $S_{a,i,m} = S_{b,k,m} = 1$), with $z = p$ and $u = d$. Under these conditions, the constraints enforce that either the proximity task of (b,k) is executed after the completion of the depositing task of (a,i), i.e., $\underline{p}_{b,k} \geq \overline{p}_{a,i}$ with $Q^{pd}_{a,i,b,k,m} = 1$, or the reverse, i.e., $\underline{p}_{a,i} \geq \overline{p}_{b,k}$ with $Q^{pd}_{a,i,b,k,m} = 0$. Conversely, Eqs. (5.14a) and (5.14b) impose no restrictions on the relative start and end times of operations (a,i) and (b,k) when the service robot is not assigned to both tasks, that is, when either $S_{a,i,m} = 0$ or $S_{b,k,m} = 0$. Extending this reasoning to all assistance phases and operations, the constraints guarantee coherent task allocation and sequencing

for every service robot.

Combining the above cost function and constraints, the following MILP formulation is derived, whose solution determines the optimal allocation and scheduling variables

$$\begin{aligned} \min \quad & \text{Cost function (5.5)} && \text{(5.15a)} \\ \text{s.t.} \quad & \text{Constraints (5.6)-(5.14)} && \text{(5.15b)} \end{aligned}$$

It is worth noticing that, although the case study presented in this chapter focuses on an agricultural domain, the proposed formulation is not limited to this context. Rather, it exhibits versatility and applicability to a broad spectrum of service tasks in diverse environments. For example, it can be adapted to industrial domains, such as assembly processes, where service robots assist human operators in complex production workflows.

5.4 Online Monitoring and Adaptation Based on Human Feedback

This section specifies the quantities monitored in real time and explains how their variations dynamically affect the parameters of the MILP formulation presented in Section 5.3.1.

5.4.1 Online Monitoring

As stated in Problem 1, both the parameters related to the working agents and the human feedback may vary over time, thereby influencing multiple parameters of the MILP formulation. Consequently, the online monitoring module is responsible for

- measuring and updating the positions of the working agents and the actual durations of their operations. As outlined in Section 5.2.1, whenever a working agent moves within the environment, the corresponding service and waiting locations, together with the path lengths, are updated accordingly. Without loss of generality, it is

assumed that positions remain unchanged during the waiting, proximity, and service phases, as the service robot is already approaching or in the vicinity of the working agent in these stages;

- acquiring human feedback regarding the perceived quality of service associated with each operation.

Specifically, for the last point, each human operator $a \in \mathcal{H}$, provides the following feedback concerning the i th operation served by the robot m

- *Waiting time feedback*, denoted as $f^w_{a,i} \in [-1, 1]$. This metric captures the perceived appropriateness of the waiting time. A value of -1 denotes that the waiting period is insufficient from the human perspective, suggesting that the human feels pressured by the service robot. In contrast, a value of 1 indicates that the waiting period is excessively long, suggesting a perceived inefficiency in the robot's assistance.
- *Proximity velocity feedback*, denoted as $f^v_{a,i} \in [-1, 1]$. This metric reflects the human perception of the robot's approach velocity. A value of -1 denotes that the robot is perceived as approaching too quickly, causing a sense of unsafety and discomfort. In contrast, a value of 1 indicates that the approach is perceived as excessively slow, suggesting inefficiency in the robot's assistance.
- *Desired waiting distance*, denoted as $f^l_{a,i} \in [-D, D]$, with D denoting a positive constant. This metric specifies the preferred adjustment in the waiting distance between the service robot and the human. In combination with the velocity feedback, it serves to improve the human's perceived safety and sense of control during the interaction with the robot.

On the basis of the feedback collected over time regarding both waiting time and velocity, two *satisfaction* indices are defined for human a: $\eta^w_{a,i}$, referring to waiting time, and $\eta^v_{a,i}$, referring to proximity velocity, formally defined as

$$\eta^z_{a,i} = g(f^z_{a,i}, \eta^z_{a,i-1}), \quad \text{with } i > 1$$

where $z \in \{w, v\}$ and $g(\cdot)$ represents a function encoding the "satisfaction dynamics" and is such that $\eta^z_{a,i} \in [-1, 1]$. As in the previous case, $\eta^w_{a,i}$

assumes positive values when the waiting time is perceived as excessively long across consecutive assistance operations, and negative values when it is perceived as too short. Analogous considerations hold for $\eta^v_{a,i}$. For the case $i = 1$, the index $\eta^z_{a,1}$ is initialized to $f^z_{a,i}$, with $z \in w, v$. One possible implementation of the function $g(\cdot)$ is

$$\eta^z_{a,i} = \theta f^z_{a,i} + (1 - \theta)\eta^z_{a,i-1},$$

where $\theta \in [0, 1]$ is a parameter that regulates the relative importance of the current feedback $f^z_{a,i}$ with respect to the preceding satisfaction index $\eta^z_{a,i-1}$. This function describes how the satisfaction index changes over time, combining the effect of the latest feedback with a weighted memory of past experiences. When θ is close to 1, the current feedback $f^z_{a,i}$ the satisfaction index is primarily influenced by the current feedback. In contrast, when θ is close to 0, the previous index $\eta^z_{a,i-1}$ prevails, leading to slower variations in the overall satisfaction level.

5.4.2 Parameters Adaptation

Based on human feedback for operation (a, i), the following MILP parameters are updated

- the weights $\alpha_{a,k}$ and $\zeta_{a,k}$ associated with future operations k of the human a (see (5.5));
- the velocity limits $v_{\min,a,m}$ and $v_{\max,a,m}$ $\forall m \in \mathcal{M}$ for the proximity phases of subsequent human operations;
- the waiting positions $x^w_{a,m}$ $\forall m \in \mathcal{M}$, which, in turn, influence the parameters $l^p_{a,m}$.

For this purpose, two *weighted* satisfaction indices of the human agent a are defined

$$\nu^z_{a,i} = (\rho^e_a + \rho^s_a)\, \eta^z_{a,i},$$

with $z \in \{w, v\}$, which weight the baseline satisfaction measure $\eta^z_{a,i}$ by both the efficiency ρ^e_a and severity condition ρ^s_a. The rationale behind these weighted indices is that the more efficient humans are and/or the

more severe the conditions under which they operate, the more their preferences should be prioritized. In other words, the more efficient humans are, or the more severe their working conditions, the more critical their perception of the robot's performance becomes. Accordingly, their satisfaction should carry greater weight in assessing the robot's service. The adaptation procedure for the parameters mentioned above is presented below.

5.4.2.1 Weights Adaptation

The update of the weights is governed by a law that incorporates both the previous weight values and the weighted satisfaction indices.

Specifically, the weights $\alpha_{a,k}$ associated with the waiting times of human agent a for subsequent operations k, where $k > i$, are updated as follows

$$\alpha_{a,k} = \max\{\alpha_{a,i} + k_\alpha \eta^w_{a,i}, 0\}, \ \ \forall k : k > i, (a,k) \in \mathcal{O}, \tag{5.16}$$

where k_α is a positive constant. Eq. (5.16) encodes the idea that an increased magnitude of the weighted human satisfaction index related to the waiting time ($\eta^w_{a,i}$) should result in larger variations of the weight $\alpha_{a,k}$ associated with future waiting-time contributions in the cost function. It should be recalled that $\eta^w_{a,i}$ takes positive values when the waiting time is perceived as long, and negative values when it is perceived as short. Therefore, positive values of $\eta^w_{a,i}$ increase the corresponding weight, thereby assigning higher priority to minimizing waiting times in the MILP resolution. Conversely, negative values reduce the weights, reflecting a diminished importance of waiting-time minimization. The maximum function guarantees that the updated weight $\alpha_{a,k}$ is non-negative.

The gains $\zeta_{a,k}$ associated with the energies spent to assist the operations k, with $k > i$, of the human agent a, are updated as follows

$$\zeta_{a,k} = \max\{\zeta_{a,i} - k_\zeta \eta^v_{a,i}, 0\}, \quad \forall k : k > i, (a,k) \in \mathcal{O} \tag{5.17}$$

where k_ζ is a positive constant. The rationale behind the Eq. (5.17) is similar to that of Eq. (5.16). Specifically, $\eta^v_{a,i}$ assumes positive values when the robot is perceived as too slow and negative values when it is

perceived as too fast. Since the robot's energy consumption is directly proportional to its velocity (see Eq. (5.4)), positive values of $\eta^v_{a,i}$ should entail a reduction of future weights relative to the current ones. This adjustment lowers the priority assigned to energy minimization during proximity phases, thereby enabling faster robot motion. Conversely, negative values should entail an increase in future weights with respect to the current ones, thus reinforcing the priority given to energy minimization during proximity phases and promoting slower robot motion.

It is worth noting that the usage of the weighted satisfaction indices $\eta^w_{a,i}$ and $\eta^v_{a,i}$ in the weight adaptation process provides a systematic mechanism for prioritizing among human operators when required. This approach enables decisions to be dynamically adapted to different levels of urgency and significance, thereby ensuring that the system's behavior remains consistent with human needs.

5.4.2.2 Velocity Bounds Adaptation

As the cost function C is expressed as a weighted sum of several contributions, certain types of feedback may be suppressed in favor of others. Such suppression is undesirable in the case of the robot's velocity during the proximity phase, owing to its significant impact on the human perception of safety. To mitigate this and to guarantee adherence to human preferences, an algorithm for adapting the velocity bounds is introduced, as detailed in Algorithm 2.

The procedure begins by identifying the robot m^* that has most recently served human a, namely the one satisfying $S_{a,i,m^*} = 1$. Subsequently, it is verified whether the human prefers an increase or a decrease in the robot's velocity. If the human expresses no preference ($\eta^v_{a,i} = 0$), i.e., he/she is satisfied with the velocity, the velocity bounds remain unchanged. When the human expresses a preference for faster motion (i.e, $\eta^v_{a,i} > 0$), the procedure checks whether the assigned robot is already the fastest within the team. If this condition holds (line 2 of Algorithm 2), weight adaptation within the cost function alone is insufficient to satisfy the human preference, necessitating an update of the proximity velocity bounds. Accordingly, the bounds of the assigned robot m^* are modified as follows. The maximum proximity velocity bound, $v^p_{\max,m^*,a}$, is increased by a factor

$(1+\eta^v_{a,i})$, while ensuring it does not exceed the maximum cruising velocity $v^c_{\max,m^*}$ (line 3). The minimum proximity velocity bound $v^p_{\min,m^*,a}$, on the other hand, is also increased by $(1+\eta^v_{a,i})$, but is saturated at $(1-c_\epsilon)v^p_{\max,m,a}$ (line 4), with c_ϵ denoting a small positive constant. For the remaining robots in $\mathcal{M}$, the adaptation of the bounds is propagated by considering their physical factors with respect to m^*. More specifically, for each robot $m \in \mathcal{M}$, with $m \neq m^*$, a scaling factor r_m is introduced, defined as the ratio of the physical factors μ_{m^*}/μ_m and is capped at a maximum of 1 (line 6). This definition implies that $r_m < 1$ when robot m has a greater mass than m^*, and $r_m = 1$ otherwise. Using this scaling, the upper velocity bound of robot m is updated by a factor $(1 + r_m\eta^v_{a,i})$ (line 7). This mechanism modulates the bound adaptation according to physical properties, in particular, when robot m is heavier than m^* (i.e., the robot associated with the feedback $\eta^v_{a,i}$), the increment factor is attenuated by r_m, producing a more conservative update. The rationale is that heavier robots may be perceived by humans as more imposing and potentially less safe, and thus their velocity bounds should be increased more cautiously. Conversely, robots with mass equal to or less than that of m^*, the full adjustment factor $\eta^v_{a,i}$ is adopted. The same logic applies to the update of the lower bound (line 8). When the human indicates a preference for slower motion (i.e., $\eta^v_{a,i} < 0$), a similar approach as above is applied. The procedure first verifies whether the slowest feasible robot has been assigned (line 9). If so, both the maximum and minimum proximity velocity bounds of robot m^* are decreased, while ensuring compliance with the physical velocity constraints (lines 10–11). Subsequently, for each robot $m \in \mathcal{M}$, with $m \neq m^*$, an additional scaling factor $\hat{r}_m$ is then introduced, defined as the ratio μ_{m^*}/μ_m and constrained to a minimum of 1 (line 13). According to this formulation, $\hat{r}_m > 1$ when robot m has greater mass than m^*, whereas $\hat{r}_m = 1$ otherwise. The minimum velocity bound $v^p_{\min,m,a}$ is reduced by the factor $(1 + \hat{r}_m\eta^v_{a,i})$, while ensuring it remains no lower than the physical minimum limit $v^c_{\min,m}$ (line 14). It is worth noting that when $\eta^v_{a,i} < 0$, the term $\hat{r}_m\eta^v_{a,i}$ is negative, implying that the multiplicative factor is strictly less than 1. This design ensures that, for robots m more massive than m^*, the reduction in speed is proportionally larger, as their greater physical presence may negatively affect human safety perception, thus necessitating a more cautious velocity profile. The maximum bound $v^p_{\max,m,a}$ is updated analogously (line 15). In both cases, the scaling factors are intentionally formulated to enforce

conservative updates, thereby applying more restrained modifications to heavier robots.

In conclusion, the proposed algorithm adapts the velocity bounds dynamically, taking into account both the robots' physical characteristics and human feedback, with the objective of balancing operational efficiency and human comfort. Furthermore, the formulation guarantees compliance with the robots' physical limitations, namely $v^p_{\max,a,m} \leq v^c_{\max,m}$ and $v^p_{\min,a,m} \geq v^c_{\min,m}$ $\forall a, m$.

Algorithm 2 Velocity Bounds Adaptation

Require: Human a, operation i concluded by a, velocity satisfaction $\eta^v_{a,i}$, variables $S_{a,i,m}, \forall m$, threshold c_ϵ

$m^* \leftarrow$ assigned robot$(S_{a,i,m}, \forall m)$ ▷ with $S_{a,i,m^*} = 1$
if $\eta^v_{a,i} > 0$ **and** $v^p_{a,i,m^*} = \max_{m\in\mathcal{M}}(v^p_{\max,m,h})$ **then** ▷ Human asks for faster robot
  $v^p_{\max,m^*,a} \leftarrow \min\{(1+\eta^v_{a,i})v^p_{\max,m^*,a}, v^c_{\max,m^*}\}$
  $v^p_{\min,m^*,a} \leftarrow \min\{(1+\eta^v_{a,i})v^p_{\min,m^*,a}, (1-c_\epsilon)v^p_{\max,m^*,a}\}$
  for each $m \in \mathcal{M}$, $m \neq m^*$ **do**
    $r_m \leftarrow \min\{\mu_{m^*}/\mu_m, 1\}$
    $v^p_{\max,m,a} \leftarrow \min\{(1+r_m\eta^v_{a,i})v^p_{\max,m,a}, v^c_{\max,m}\}$
    $v^p_{\min,m,a} \leftarrow \min\{(1+r_m\eta^v_{a,i})v^p_{\min,m,a}, (1-c_\epsilon)v^p_{\max,m,a}\}$
else if $\eta^v_{a,i} < 0$ **and** $v^p_{a,i,m^*} = \min_{m\in\mathcal{M}}(v^p_{\max,m,h})$ **then** ▷ Human asks for slower robot
  $v^p_{\min,m^*,a} \leftarrow \max\{(1+\eta^v_{a,i})v^p_{\min,m^*,a}, v^c_{\min,m^*}\}$
  $v^p_{\max,m^*,a} \leftarrow \max\{(1+\eta^v_{a,i})v^p_{\max,m^*,a}, (1+c_\epsilon)v^p_{\min,m^*,a}\}$
  for each $m \in \mathcal{M}$, $m \neq m^*$ **do**
    $\hat{r}_m \leftarrow \max\{\mu_m/\mu_{m^*}, 1\}$
    $v^p_{\min,m,a} \leftarrow \max\{(1+\hat{r}_m\eta^v_{a,i})v^p_{\min,m,a}, v^c_{\min,m}\}$
    $v^p_{\max,m,a} \leftarrow \max\{(1+\hat{r}_m\eta^v_{a,i})v^p_{\max,m,a}, (1+c_\epsilon)v^p_{\min,m,a}\}$

5.4.2.3 Waiting Position Adaptation

Given the desired increase/decrease of the proximity distance $f^l_{a,i}$ provided by human a and the assigned robot m^*, i.e., such that $S_{a,i,m^*} = 1$, the waiting distance l^p_{a,m^*} is updated as

$$l^p_{a,m^*} = l^{p^-}_{a,m^*} + f^l_{a,i},$$

where $l^{p^-}_{a,m^*}$ represents the waiting distance before adaptation. For the propagation to the remaining robots, the physical factors are once more

taken into account. Specifically, for each service robot $m \neq m^*$, the waiting distance is updated as follows

$$l^p_{a,m} = \begin{cases} l^{p^-}_{a,m} + \hat{r}_m f^l_{a,i}, & \text{if } f^l_{a,i} \geq 0 \\ l^{p^-}_{a,m} + r_m f^l_{a,i}, & \text{if } f^l_{a,i} < 0 \end{cases} \tag{5.18}$$

where r_m and $\hat{r}_m$ denote the scaling factors defined in lines 6 and 13 of Algorithm 2, respectively. The rationale is as follows: whenever the distance requires an increase ($f^l_{a,i} \geq 0$), this indicates that the human perceives the situation as unsafe or obstructed; consequently, a larger increase is applied if robot m is more massive than robot m^*. In contrast, when the distance requires a reduction ($f^l_{a,i} < 0$), a smaller decrease is applied if robot m is more massive than robot m^*. Following these adjustments, the waiting locations $x^w_{a,m}$ are modified accordingly, $\forall m \in \mathcal{M}$.

5.5 Online Plan Update

Once the parameters are updated according to the procedures outlined in the previous section, the plan must be adapted as needed. This online adaptation is carried out in two steps:

1. *Re-scheduling*: a heuristic-based re-scheduling procedure is implemented, which exploits the updated parameters to refine the timing of tasks and operations, thereby preserving feasibility.
2. *Re-allocation*: when the re-scheduling procedure alone results in significant suboptimality due to substantial parameter changes, a comprehensive re-allocation, including re-scheduling, is performed to obtain an optimal plan according to the MILP formulation described in Section 5.3.1.

The adoption of this two-step procedure is motivated by the substantial computational complexity of the MILP formulation, which is NP-hard and therefore inappropriate for frequent real-time re-computation in dynamic contexts. Therefore, MILP-based solutions are employed only when strictly required. For large-scale systems, complexity is additionally reduced through the strategy presented in Section 5.6.

5.5.1 Online re-scheduling

The heuristics-based online updating algorithm modifies the scheduling of tasks and operations to preserve compliance with the imposed constraints. Let t_{upd} denote the time instant when the current plan was last updated, and t_{mod} the time instant when the new parameters are obtained. Moreover, let $\delta^o_{a,\min} = \sqrt{2}\sigma_a \text{erf}^{-1}(2\,pr_{\min} - 1) + \varphi_a, \forall a \in \mathcal{A}$ be the minimum duration assigned to the operations of agent a. Algorithm 3 outlines the updating procedure for the case in which the working agent a^* experiences parameter modifications.

For simplicity, t_{mod} is used whenever the time variable is omitted. The algorithm proceeds as follows

1. *Initialization*: at t_{mod}, the plan is initialized using the most recent plan (t_{upd}) (line 1).
2. *Updating Tasks*: for each operation (a^*, i) that is completed at t_{mod} or thereafter, the algorithm identifies the corresponding robot m^* (line 4). Operations close to completion are also considered, since late-stage modifications may still necessitate adjustments to related robot assistance tasks. The analysis then proceeds depending on whether a timing violation or a change in the agent's position has occurred.

   The first case arises when the start time of the service task is earlier than the completion of the associated operation (line 4), indicating that agent a^* required more time than initially scheduled. As a result, the service is initiated prematurely, i.e., prior to the operation's conclusion. In this case, the algorithm sets both the end time of the waiting task (which is extended) and the start time of the proximity task to $\overline{o}_{a^*,i}$ (line 5). The subsequent phases associated with operation (a^*i) are then updated accordingly (lines 6–7), and a procedure is also invoked to shift, i.e., delay, subsequent tasks if needed (line 8). It is important to note that the robot's velocity during the proximity phase is deliberately preserved, as any modification could adversely affect the human's perceived safety.

   When a position variation occurs, the algorithm first verifies whether the current timings are *compatible* under the assumption

of maximum velocity during the going phase (line 11). Here, *compatibility* denotes that the timings of the tasks linked to operation (a^*, i) can be modified without affecting the final proximity phase time $\overline{p}_{a^*,i}$. If incompatibility is detected, the going phase is executed at the maximum feasible velocity, the subsequent timings are adjusted accordingly (lines 12–14), and the procedure to delay subsequent tasks is invoked when required (line 15). Conversely, if compatibility holds, the timings are simply adapted, without shifting subsequent tasks (line 17).

3. *Final Adjustments*: once all operations of agent a^* have been rescheduled, the start and end times of the operations associated with every other working agent a are updated by applying the minimum duration $\delta^o_{a,\min}$ (lines 18–20). Subsequently, t_{upd} is set equal to t_{mod} to register the most recent modifications to the plan.

4. *Procedure for Shifting Consecutive Tasks*: for the current operation (a^*, i), the procedure verifies whether the subsequent operation of the agent is scheduled to start before the completion of the preceding service. If so, the start time is updated, and the corresponding end time is adjusted on the basis of the minimum duration $\delta^o_{a,\min}$ (lines 1–3 of the procedure *HandleShift*).

   Subsequently, the algorithm verifies whether the acceleration of the robot during the depositing phase is feasible while maintaining $\overline{d}_{a^*,i}$ unchanged, thereby avoiding modifications to subsequent tasks (line 4). If acceleration is infeasible, the current depositing end time is stored in the variable $\overline{d}^{prev}_{a^*,i}$ (line 5), and a new depositing end time is computed under the assumption of maximum robot velocity (line 5). The shifting quantity $\phi = \overline{d}_{a^*,i} - \overline{d}^{prev}_{a^*,i}$ is then calculated (line 7), and all consecutive assistance tasks are uniformly delayed by ϕ. More precisely, for each service robot r, the set $\mathcal{T}_r$ of ordered assisted operations is retrieved (line 9). For every operation (a_r, i_r) consecutive to (a^*, i), each related assistance task is shifted by ϕ, postponing both start and end times accordingly. Importantly, this shift preserves the original operation order and does not violate any constraints, as it simply postpones the scheduled times to accommodate the updated parameters.

Algorithm 3 Re-scheduling Algorithm

Require: agent a^* with updated parameters, current time t_{mod}, previous plan at time t_{upd}

plan(t_{mod}) ←plan(t_{upd})
for each $i \in \{1, \dots, q_{a^*}\}$ **and** $\overline{o}_{a^*,i} \geq t_{mod}$ **do**
  m^* ←assigned robot$\left(S_{a,i,m}, \forall m\right)$
  if $\underline{s}_{a^*,i} < \overline{o}_{a^*,i}$ **then** ▷ Agent slower than planned
    $\overline{w}^s_{a^*,i}$, $\underline{p}_{a^*,i} \leftarrow \overline{o}_{a^*,i}$
    $\overline{p}_{a^*,i} \leftarrow \underline{p}_{a^*,i} + l^p_{a^*,m^*}/v^p_{a^*,i,m^*}$
    $\underline{s}_{a^*,i} \leftarrow \overline{p}_{a^*,i}$, $\overline{s}_{a^*,i} \leftarrow \underline{s}_{a^*,i} + \delta^s_{m^*}$, $\underline{d}_{a^*,i} \leftarrow \overline{s}_{a^*,i}$
    call HandleShift(a^*, i)
  else if $x^s_a(t_{mod}) \neq x^s_a(t_{upd})$ **then** ▷ Agent changes position
    if $\overline{p}_{a^*,i} \leq \underline{g}_{a^*,i} + \frac{l^g_{a^*,m^*}}{v^c_{\max,m^*}} + \frac{l^p_{a^*,m^*}}{v^p_{a^*,i,m^*}}$ **then** ▷ cannot speed up going phase
      $\overline{g}_{a^*,i} \leftarrow \underline{g}_{a^*,i} + l^g_{a^*,m^*}/v_{\max,m^*}$
      $\underline{w}^s_{a^*,i}, \overline{w}^s_{a^*,i}, \underline{p}_{a^*,i} \leftarrow \overline{g}_{a^*,i}$
      $\overline{p}_{a^*,i} \leftarrow \underline{p}_{a^*,i} + l^p_{a^*,m^*}/v^p_{a^*,i,m^*}$
      $\underline{s}_{a^*,i} \leftarrow \overline{p}_{a^*,i}$, $\overline{s}_{a^*,i} \leftarrow \underline{s}_{a^*,i} + \delta^s_{m^*}$, $\underline{d}_{a^*,i} \leftarrow \overline{s}_{a^*,i}$
      call HandleShift(a^*, i)
    else ▷ speed up going phase
      $\overline{g}_{a^*,i}, \underline{w}^s_{a^*,i}, \overline{w}^s_{a^*,i}, \underline{p}_{a^*,i}$ ← update times
for each $(a, i) \in \mathcal{O}$ **with** $i > 1$ **do**
  if $\underline{o}_{a,i} < \overline{s}_{a,i-1}$ **then**
    $\underline{o}_{a,i} \leftarrow \overline{s}_{a,i-1}$, $\overline{o}_{a,i} \leftarrow \underline{o}_{a,i} + \delta^o_{a,\min}$
$t_{upd} \leftarrow t_{mod}$

Procedure HandleShift (a^*, i)
if $i < q_{a^*}$ **and** $\underline{o}_{a^*,i+1} < \overline{s}_{a^*,i}$ **then**
  $\underline{o}_{a^*,i+1} \leftarrow \overline{s}_{a^*,i}$
  $\overline{o}_{a^*,i+1} \leftarrow \underline{o}_{a^*,i+1} + \delta^o_{a^*,\min}$
if $\overline{d}_{a^*,i} - \underline{d}_{a^*,i} \leq l^d_{a^*,m^*}/v^c_{\max,m^*} + \delta^d_{m^*}$ **then** ▷ cannot speed up depositing phase
  $\overline{d}^{prev}_{a^*,i} \leftarrow \overline{d}_{a^*,i}$
  $\overline{d}_{a^*,i} \leftarrow \underline{d}_{a^*,i} + l^d_{a^*,m^*}/v^c_{\max,m^*} + \delta^d_{m^*}$
  $\phi \leftarrow \overline{d}_{a^*,i} - \overline{d}^{prev}_{a^*,i}$ ▷ shifting quantity
  for each $r \in \mathcal{M}$ **do**
    $\mathcal{T}_r \leftarrow$ ordered assistance operations robot (r, S)
    for each $(a_r, i_r) \in \mathcal{T}_r$ consecutive to (a^*, i) **do**
      plan ← postpone assistance tasks (a_r, i_r, ϕ)
else ▷ speed up deposit phase
  $\overline{d}_{a^*,i} \leftarrow$ update time

5.5.2 Online re-allocation

Once the updated schedule has been obtained using Algorithm 3, the resulting plan is compared against the preceding optimal solution. If the difference in cost between the two schedules exceeds a predefined threshold, the MILP solution is recomputed. In this context, t_{opt} represents the time at which the most recent optimal solution was obtained, while

$C^+(t)$ denotes the cost associated with the unstarted tasks based on the parameters available at time t. The relative optimality index

$$\iota \triangleq |C^+(t_{opt}) - C^+(t_{upd})|/C^+(t_{opt}) \tag{5.19}$$

is introduced to capture the change in the cost of unstarted tasks when comparing the parameters at t_{opt} with those at t_{upd}. This index provides a quantitative measure of the deviation of the updated schedule from the previously optimal plan. Whenever ι exceeds the positive threshold ι_t, the MILP problem in (5.15) is recomputed, using the updated parameters and weights, to re-allocate and re-schedule tasks. This mechanism ensures that re-optimization is triggered only when the current plan is substantially degraded compared to the most recent optimal plan. During this process, additional constraints ensure that tasks already in execution remain assigned to their respective service robots, thereby maintaining operational consistency and preventing unnecessary disruptions. Upon completion, t_{opt} is updated to t_{ch}.

5.6 Extension to Large-Scale Settings

For large-scale instances, the MILP formulation may become computationally prohibitive owing to the NP-hardness of the problem. To address this limitation, two complementary strategies are introduced

1. Reformulating the problem within a Constraint Programming (CP) framework [134].
2. Applying a batch decomposition approach to partition operations and service tasks.

5.6.1 Constraint Programming Reformulation

As anticipated in Section 2.7, Constraint Programming (CP), and in particular Constraint Optimization Problems (COPs), is particularly well suited to model scheduling scenarios characterized by temporal and logical constraints [39, 134]. CP accommodates diverse variable domains, such as integers, Booleans, and intervals, which are particularly suitable for temporal modeling, and supports a broad spectrum of constraint types,

including arithmetic, logical, conditional, and global constraints over sets of variables (such as allDifferent(·) and noOverlap(·)). These features facilitate the representation of complex scheduling rules. For a detailed treatment of CP, see [134].

In order to fully exploit the strengths of the CP paradigm, the constraints and variables defined in Section 5.3.1 must be reformulated to exploit key CP features, such as interval variables and global constraints, while maintaining the original cost function. Therefore, a set of *non-optional* interval variables $\tau^z_{a,i}, (a,i) \in \mathcal{O}, z \in \{o,g,w,p,s,d\}$ is introduced to represent the operations executed by each agent $a \in \mathcal{A}$ and the phases of the corresponding assistance activities. Furthermore, *optional* interval variables $\psi^z_{a,i,m}, z \in g,p,s,d$ are defined for each service robot $m \in \mathcal{M}$, where their inclusion in the solution denotes the assignment of robot m to the assistance activity associated with operation $(a,i) \in \mathcal{O}$. Concerning the constraints, the following subsection presents the reformulation of some key constraints selected from Section 5.3.1, since the others are either immediate or rely on similar patterns. For instance, consider constraints (5.14a)-(5.14b), which ensure that each service robot m performs at most one task simultaneously. Within the CP framework, these constraints can be reformulated by means of the noOverlap(·) global constraint, namely,

$$\text{noOverlap}([\psi^z_{a,i,m}]^{z\in\{g,p,s,d\}}_{a\in\mathcal{A},i\in\{1,\dots,q_a\}}),\ m \in \mathcal{M}$$

As for constraint (5.6), which guarantees that each operation (a,i) of a working agent is served by one service robot $m \in \mathcal{M}$, it can be expressed in the CP formulation using the alternative(·) global constraint, i.e.,

$$\text{alternative}(\tau^g_{a,i}, [\psi^g_{a,i,m}]_{m\in\mathcal{M}}),\ a \in \mathcal{A},\ i \in \{1,\dots,q_a\}$$

Finally, concerning constraints (5.7)-(5.9), which define the duration of each going, proximity, and depot phase, they can be expressed by combining the presenceOf(·) basic constraint with the ifThen(·) global constraints as follows

$$\text{ifThen}\left(\text{presenceOf}(\psi^z_{a,i,m}), \text{lengthOf}(\tau^z_{a,i}) = \frac{l^z_{a,m}}{v^z_{a,i,m}} + \delta^z_m\right)$$

where presenceOf$(\psi^z_{a,i,m}) \in 0, 1$ indicates whether the specified optional interval variable is present in the solution, i.e., whether the assistance activity associated with the operation (a, i) is assigned to the service robot m or not.

5.6.2 Batch Decomposition Strategy

In order to address large-scale systems more effectively, a batch decomposition strategy is introduced. This method divides operations and their associated service tasks into fixed-size batches, which are then solved in an iterative manner. Such an approach, widely adopted in the literature to enhance scalability in complex optimization settings [135, 136, 137, 138], achieves a balanced compromise between computational feasibility and solution quality. The strategy relies on first ordering operations according to a specified criterion (e.g., distance or heuristic measures) and then progressively adding new tasks to the planning process after each serving phase is completed. The overall procedure is summarized in Algorithm 4.

Given the batch size b_s^{in} and the operation set $\mathcal{O}$, operations are first ordered into a list $L_{\mathcal{O}}$ (line 1). The current time variable t_{curr} is initialized to zero (line 2), and the first b_s^{in} operations are extracted from $L_{\mathcal{O}}$ (line 3) to define the working set $\mathcal{O}^*$ and to construct the initial plan plan_{curr} (line 4). The algorithm then iteratively proceeds until $L\mathcal{O}$ is empty (line 5). At each iteration, the earliest service completion time strictly greater than t_{curr} within $\mathcal{O}^*$ is identified and assigned to t_{curr}. A set $\mathcal{P}_{fixed}$, representing the operations and tasks that must remain fixed in the subsequent optimization step, is initialized as empty, and the batch size b_s is reset to the initial input value b_s^{in} (lines 7–8). Subsequently, each operation in $\mathcal{O}^*$ is inspected: if the corresponding service task has been completed by $tcurr$, the operation together with all its assistance phases is fixed (line 11). If the service is ongoing but the operation has started, only the operation is fixed, and b_s is decremented by one (lines 13–14). Furthermore, if the going phase of the assistance is already in progress, the remaining assistance tasks are fixed as well (line 19). After completing the processing of all operations in $\mathcal{O}^*$, the algorithm inserts the first b_s operations from $L_{\mathcal{O}}$ into $\mathcal{O}^*$ and removes them from the list (line 20). Subsequently, the optimization algorithm is applied to the updated set $\mathcal{O}^*$, while all activities in $\mathcal{P}_{fixed}$ remain unchanged (line 21). This procedure facilitates

progressive planning, guarantees consistency with previously scheduled tasks, and maintains computational tractability.

Algorithm 4 Batch Decomposition Strategy

Require: b_s^{in}, $\mathcal{O}$
$L_{\mathcal{O}} \leftarrow \text{sort}(\mathcal{O})$
$t_{\text{curr}} \leftarrow 0$
$\mathcal{O}^* \leftarrow \text{pop}(L_{\mathcal{O}}, 1 : b_s^{in})$
$\text{plan}_{\text{curr}} \leftarrow$ optimization algorithm$(\mathcal{O}^*)$
while $L_{\mathcal{O}} \neq \emptyset$ **do**
    $t_{\text{curr}} \leftarrow$ next service end time$(\{\overline{s}_{a,i}\}_{(a,i)\in\mathcal{O}^*}, t_{\text{curr}})$
    $\mathcal{P}_{\text{fixed}} \leftarrow \emptyset$
    $b_s \leftarrow b_s^{in}$
    for all $(a, i) \in \mathcal{O}^*$ **do**
        if $\overline{s}_{a,i} \leq t_{\text{curr}}$ **then**
            $\mathcal{P}_{\text{fixed}} \leftarrow \text{add}(\tau_{a,i}^o, \tau_{a,i}^g, \tau_{a,i}^w, \tau_{a,i}^p, \tau_{a,i}^d)$
        else if $\overline{s}_{a,i} > t_{\text{curr}}$ **and** $\underline{o}_{a,i} < t_{\text{curr}}$ **then**
            $\mathcal{P}_{\text{fixed}} \leftarrow \text{add}(\tau_{a,i}^o)$
            $b_s \leftarrow b_s - 1$
        if $\underline{g}_{a,i} < t_{\text{curr}}$ **then**
            if $\tau_{a,i}^o \notin \mathcal{P}_{\text{fixed}}$ **then**
                $\mathcal{P}_{\text{fixed}} \leftarrow \text{add}(\tau_{a,i}^o)$
                $b_s \leftarrow b_s - 1$
            $\mathcal{P}_{\text{fixed}} \leftarrow \text{add}(\tau_{a,i}^g, \tau_{a,i}^w, \tau_{a,i}^p, \tau_{a,i}^d)$
    $\mathcal{O}^* \leftarrow \text{add}(\text{pop}(L_{\mathcal{O}}, 1 : b_s))$
    $\text{plan}_{\text{curr}} \leftarrow$ optimization algorithm$(\mathcal{O}^*, \mathcal{P}_{\text{fixed}})$
return $\text{plan}_{\text{curr}}$

5.7 Simulation Validation

The proposed framework was validated within a simulated agricultural case study involving a table-grape vineyard with human and robotic agents at work. As illustrated in Figure 5.1, human operators, together with robots equipped with manipulators, are tasked with harvesting grape bunches and placing them in boxes. Meanwhile, the service robots assist them by replacing the filled boxes with empty ones and transporting them to a depot. To provide feedback, humans interact through a customized graphical user interface (GUI).

5.7.1 ROS Architecture and Human Interface

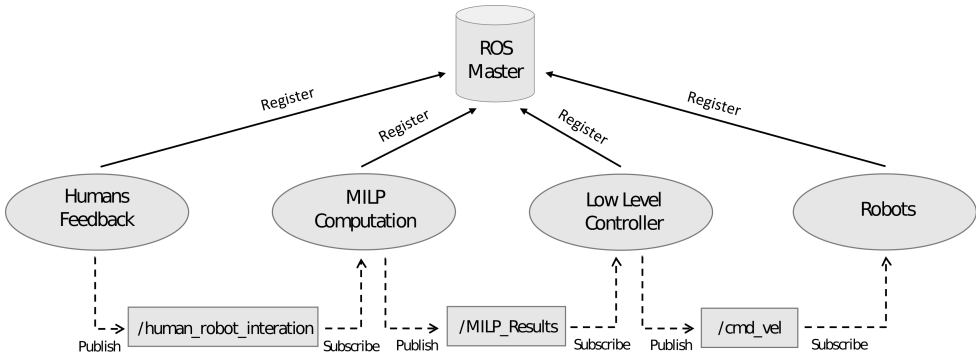


Figure 5.4: Schematic illustration of the ROS architecture.

The proposed architecture was designed and implemented on Ubuntu 20.04 under the ROS (Robot Operating System) Noetic Ninjemys framework. As illustrated in Figure 5.4, four main nodes were developed: *i)* a human feedback module, realized using Matlab, which handles the interaction with the human via a GUI, *ii)* a MILP computation node, realized again using Matlab, which solves the optimization problem using the 10th version of the Gurobi solver [139] as well as handles the online plan update, *iii)* a low-level controller, realized in C++, which provides the velocity commands to the robots based on the high-level plans, and *iv)* the hardware interface nodes which move the robots according to the velocity commands. In case of large-scale systems, the strategy in Section 5.6 is adopted, replacing the MILP computation node with a CP-based one, developed in Python and using the IBM ILOG CPLEX CP solver (version 22.1.1.0). The communication among the nodes was handled through ROS by introducing two custom ROS messages to share the plan and the human feedback.

Regarding the human feedback module, the graphical user interface (GUI) shown in Figure 5.5 was developed. The interface enables the monitoring of workers' parameters and the collection of their feedback after each service, thereby allowing the framework to adapt dynamically to individual needs and preferences. More in detail, each human a is provided with a GUI accessible via a tablet or laptop. The interface includes an informational area where real-time updates and notifications addressed to

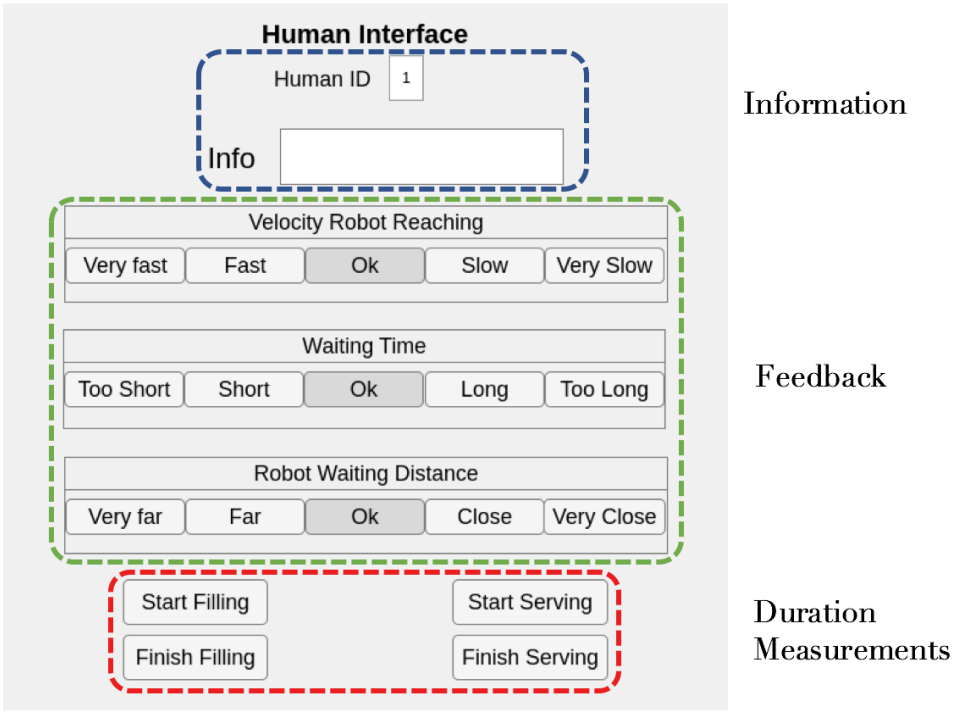


Figure 5.5: GUI for human feedback and measurement.

humans are displayed. For instance, it can display messages about an approaching robot, the expected end time of the operation, or other relevant notifications. Then, a feedback section is available that enables users to provide feedback on the *waiting time*, $f^w_{a,i}$, the *proximity velocity*, $f^v_{a,i}$, and the desired waiting distance, $f^l_{a,i}$. Each feedback type is presented as a set of five selectable options. In the case of waiting time, the available choices are *Too Short*, *Short*, *OK*, *Long*, and *Too Long*. These options are mapped to numerical values within the range $[-1, 1]$, where selecting *Too Short* corresponds to $f^w_{a,i} = -1$ and *Too Long* corresponds to $f^w_{a,i} = 1$. Similarly, for the velocity the options are *Very fast* (mapped into $f^v_{a,i} = -1$), *Fast*, *OK*, *Slow*, and *Very Slow* (mapped into $f^v_{a,i} = 1$), while for the desired waiting distance the options are *Very far* (mapped into $f^l_{a,i} = -1$), *Far*, *OK*, *Close*, and *Very close* (mapped into $f^l_{a,i} = 1$). This intuitive design ensures that users can easily express their preferences in real-time.

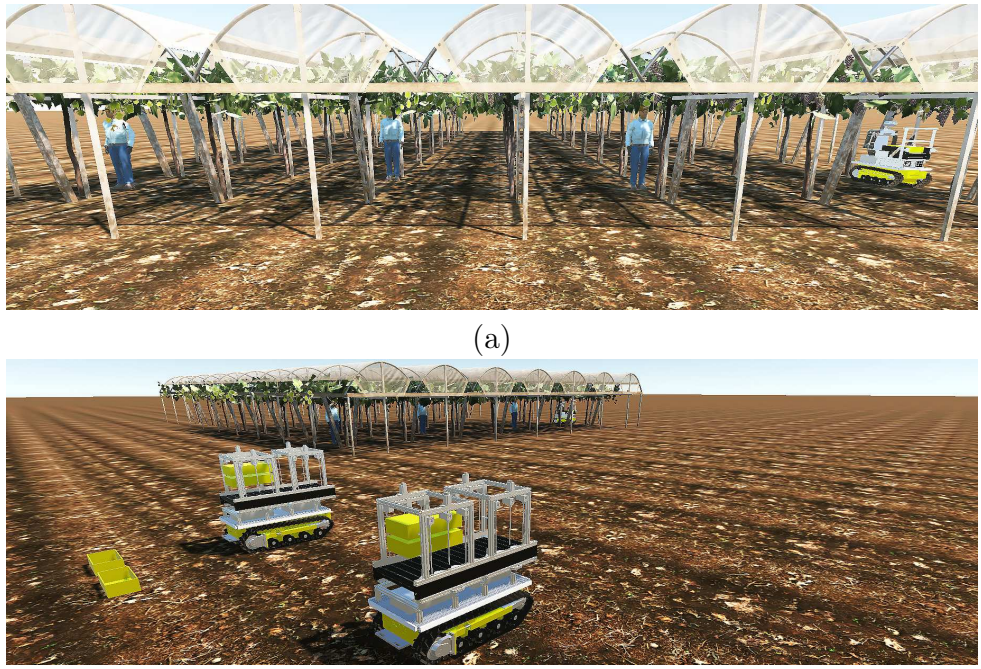
Finally, the GUI also features a measurement section that allows users

to manually record key time metrics, specifically the *Box Filling Time* and the *Serving Time*. This is achieved by clicking the respective *Start* and *Finish* buttons. While this manual approach is straightforward and effective for the study presented within this chapter, it is worth noting that alternative methods, such as automated measurements using camera-based data [140], could also be employed. However, integrating such automated approaches falls beyond this chapter's scope, which is focused on exploring how to use these measurements to adapt the plan properly.

The low-level controller ensures the tracking of desired trajectories and avoids collisions among serving robots for unicycle-like kinematics. In particular, trajectories are generated as cubic splines through the APIs provided by the Alglib library, and their tracking is achieved through an input/output feedback linearization control law [141, Chapter 11]. Concerning obstacle avoidance, this is achieved through a standard control strategy based on artificial potentials [142] that envisages the addition of a repulsive velocity for each robot i when the distance to another robot in the team is below a positive threshold d_{th}, which it was set equal to 3 m.

5.7.2 Simulation Setup

The simulations were obtained by running all the components of the proposed architecture on the same machine, a commercial laptop equipped with an NVIDIA GeForce RTX 3070 Ti GPU, an Intel Core i7H CPU, and 32GB RAM. They were carried out in a Unity-based virtual reality environment developed for the CANOPIES project, which can realistically reproduce a vineyard scenario. Figures 5.6.a) and 5.6.b) provide an overview of the simulation environment and the scenario adopted for the simulation experiments. As depicted in Figure 5.6.a), simulations took place in a table-grape vineyard made up of seven rows, each 3 m large and 15 m long, and involved four working agents, specifically three humans and one working robot, which were arranged as shown in the figure. The working robot comprised an Alitrak DCT-350P as the lower part and a dual-arm system as the upper part. Moreover, as shown in Figure 5.6.b), the two service robots considered were Alitrak DCT-350P mobile bases (without upper-body). To demonstrate the potential of using heterogeneous robots, the simulations were conducted with different values of the



(a)

(b)

Figure 5.6: Simulative agricultural scenario with the working agents (3 humans and 1 robot) and the service robots.

physical μ factor for each service robot. To demonstrate the potential of using heterogeneous robots, the simulations were conducted with different values of the physical μ factor for each service robot. In particular, the parameters were set to $\mu_1 = 0.75$ and $\mu_2 = 1$. Regarding the velocities, the general velocity limits and the proximity phase velocity limits were set as follows: $v^c_{\min,m} = 0.05$ m/s, $v^c_{\max,m} = 0.2$ m/s, $v^p_{\min,a,m} = 0.04$ m/s, and $v^p_{\max,a,m} = 0.15$ m/s, $\forall m \in \mathcal{M}$ and $\forall a \in \mathcal{A}$. The robots deposit base were located for m_1 at $x^d_1 = [-10.4, 7.0]$ and for m_2 at $x^d_2 = [-14.4, 7.0]$. The severity factor ρ^s_a of each of the three human workers in the simulation was initialized differently. In particular, they were set as follows: $\rho^s_1 = 0.5$, $\rho^s_2 = 0.75$, and $\rho^s_3 = 1$. The humans' filling times were initialized to 250 s, $\forall a \in \mathcal{H}$ and modeled as a Gaussian distribution $\mathcal{N}(\varphi_a, \sigma_a)$, with $\varphi_1 = 204$, $\varphi_2 = 275$, and $\varphi_3 = 213$ and $\sigma_a = 0.1\varphi_a$ $\forall a \in \mathcal{H}$. The desired minimum probability was set to $pr_{min} = 0.9$. The agents service position were as follows, $x^s_1 = [1, -1.375]$, $x^s_2 = [2.3, -4.125]$, $x^s_3 = [1, -9.625]$, $x^s_4 = [2.3, -15.125]$. The waiting distances were initialized as $l^p_{a,m} = 1$ m,

$\forall m \in \mathcal{M}$ and $\forall a \in \mathcal{H}$.

5.7.3 Simulation Results

A high-resolution video showing the simulation outlined in this section is available in the accompanying video and at the link[1].

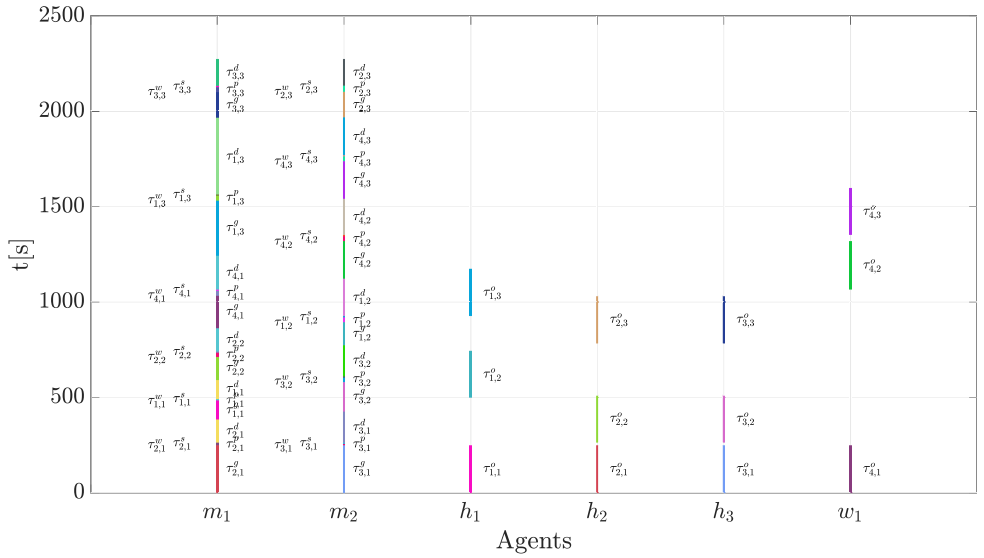


Figure 5.7: Simulation case study. First offline allocation with initial parameters.

ID	Waiting Distance $l^p_{a,m}$	Waiting Feedback $f^w_{a,1}$	Velocity Feedback $f^v_{a,1}$
h_1	$l^p_{1,1} = 0.5$ m (*Far*)	$f^w_{1,1} = 0.5$ (*Long*)	$f^v_{1,1} = 0.5$ (*Slow*)
h_2	$l^p_{1,1} = 1$ m (*OK*)	$f^w_{2,1} = -0.5$ (*Short*)	$f^v_{2,1} = 0$ (*OK*)
h_3	$l^p_{1,2} = 1$ m (*OK*)	$f^w_{3,1} = -0.5$ (*Short*)	$f^v_{3,1} = 0.5$ (*Slow*)

Table 5.2: Summary of the feedback provided by the three humans regarding the first operation service.

Figures 5.7-5.8 and Tables 5.2-5.3 contain the results obtained for the simulation. In particular, Figure 5.7 shows the first allocation plan, i.e.,

[1] `https://youtu.be/Bxg8cMFbyNo` - Notably, the video does not visually show box filling or exchanging operations, as the simulator did not support these. However, their timing was simulated for replanning purposes, so this omission does not affect the validity of the results, which focus on high-level coordination and are independent of platform-specific execution details.

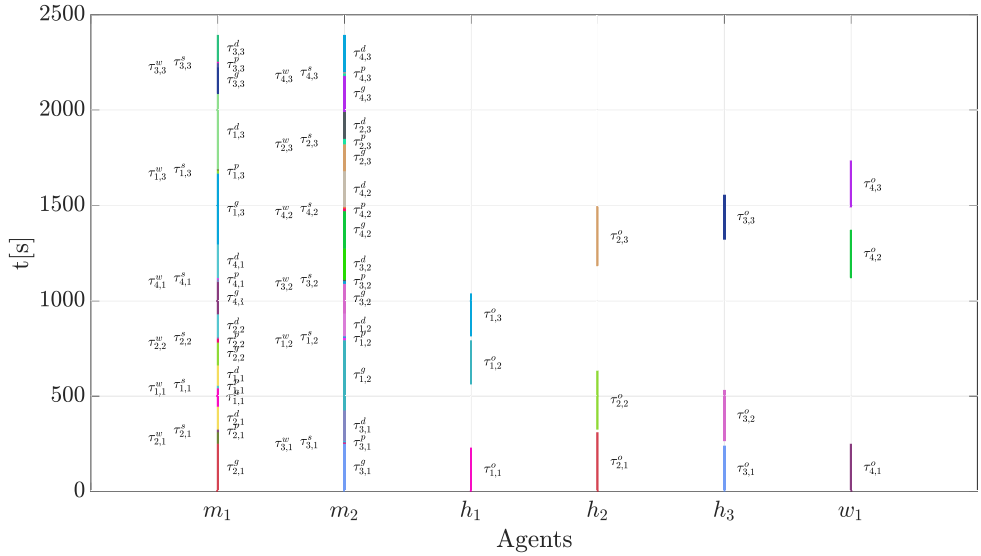


Figure 5.8: Simulation case study. Online reallocation with measured parameters and feedback.

ID	Op. End Time	$\alpha_{a,1}$	$\zeta_{a,1}$
h_1	$\overline{o}_{1,1} = 230$ s	$\alpha_{1,1} = 1.73$	$\zeta_{1,1} = 1.69$
h_2	$\overline{o}_{2,1} = 310$ s	$\alpha_{2,1} = 0.93$	$\zeta_{2,1} = 1.19$
h_3	$\overline{o}_{3,1} = 240$ s	$\alpha_{3,1} = 0.92$	$\zeta_{3,1} = 1.78$

Table 5.3: End times of the operations and values of the weights after each human has given the first feedback.

the allocation plan formulated by the framework using the configuration described at the beginning of this section. From left to right, the schedules for the service robots $m \in \mathcal{M}$, the human operators $h \in \mathcal{H}$, and the working robot $w \in \mathcal{W}$ are shown. In particular, the length of the lines accounts for the duration of the tasks. For the service robots, thicker lines represent the going phase, and the color coincides with the one used for the respective operation of the working agent. The subsequent phases are represented with thinner lines of different colors to be able to distinguish their different durations. As an example, the first task assigned to the service robot m_1 is the going one for assisting h_2, and both $\tau^g_{2,1}$ and $\tau^o_{2,1}$ are colored in red. Figure 5.8, instead, displays the re-allocation plan generated after the 3 humans gave their feedback at the end of the service as collected in Table 5.2 and through the interface

described in Section 5.7.1.

By following the initial plan, the human h_1 finished the first filling operation at time $\overline{o}_{1,1} = 230$ s and the respective service, which was performed by the robot m_1, ended at $\tau^s_{1,1} = 495.3$ s. At this point, the human used the GUI to provide the following feedback: *Long* for the waiting time ($f^w_{1,1} = 0.5$), *Slow* for the velocity of the robot during the proximity phase ($f^v_{1,1} = 0.5$) and the waiting distance was considered *Far*. This led to adapting the weights according to the procedure described in Section 5.4.2. The resulting weights are reported in Table 5.3, which shows that for the human 1 after the first service, it holds $\alpha_{1,k} = 1.73$ and $\zeta_{1,k} = 1.69$, with $k > 1$. As a consequence, re-allocation was triggered and, as shown by comparing Figures 5.7 and 5.8, the waiting was reduced for the next task from $w^o_{1,1} = 245.3$ s to $w^o_{1,2} = 20.5$ s to accommodate the feedback given for the waiting time. Additionally, the waiting distance was reduced to $l^p_{2,2} = 0.5$ m, and the duration of the proximity phase was maintained at 13.5 s, which, combined with a shorter distance to travel, led to a higher velocity. For h_2, the first filling was completed at $\overline{o}_{2,1} = 310$ s and the respective service was concluded at $\tau^s_{2,1} = 323.3$ s by the robot m_1. The human feedback reported *Short* for the waiting time ($f^w_{2,1} = -0.5$), *OK* for velocity during the proximity phase ($f^v_{2,1} = 0$), and *OK* for waiting distance. The weights values were $\alpha_{2,k} = 0.93$ and $\zeta_{2,k} = 1.19$, with $k > 1$. Consistent with the feedback provided by h_2 for a longer waiting time, the waiting time for the next task was increased from $w^o_{2,1} = 13.7$ s to $w^o_{2,2} = 269.5$ s. Moreover, the waiting distance during the proximity phase remained $l^p_{2,2} = 1$ m, while the proximity phase duration was maintained at 21 s. Concerning h_3, he/she completed the first filling at $\overline{o}_{3,1} = 240$ s, and the associated service was accomplished at $\tau^s_{3,1} = 263.7$ s by the robot m_2. In this case, the sent feedback reported *Short* for the waiting time ($f^w_{3,1} = -0.5$), *Slow* for velocity during the proximity phase ($f^v_{3,1} = 0$), and *OK* for waiting distance. After this service, the weights were $\alpha_{3,k} = 0.92$ and $\zeta_{3,k} = 1.78$, with $k > 1$. The reallocation led to an increased waiting time for the next task from $w^o_{3,1} = 23.7$ s to $w^o_{3,2} = 786$ s, addressing the feedback for longer waiting time. The robot's waiting distance during the proximity phase remained $l^p_{3,2} = 1$ m, and the proximity phase duration was reduced to 14 s for a faster approach. The remaining two operations of the four working agents until the final time $t = 2394.5$ s are shown in the video. As for the effectiveness of the simplified version of Algorithm

3, it was thoroughly validated in the study presented in [143].

5.7.4 Human Preference Validation

In order to demonstrate the capability of the proposed framework in addressing human needs, an evaluation metric, referred to as the *adaptation index* and denoted by ω^z (with $z = w$ for the waiting time $w^o_{a,i}$ and $z = v$ for the proximity velocity $v^p_{a,i,m}$), is introduced. This index is evaluated after the reallocation process and quantifies the quality of the allocation, thereby capturing the degree to which the system adapts to and incorporates human feedback. Higher scores indicate improved alignment with human preferences. The update mechanism governing this index is formalized in Algorithm 5. The algorithm takes as input the human feedback $f^z_{a,i}$, which indicates a preference for increasing or decreasing the parameter z, as well as the previous adaptation index denoted as ω^z_{prev} and initialized as 1. In the algorithm, $v^p_{a,i}$ represents the velocity parameter associated with the robot performing the assistance task (a, i) during the proximity phase. Positive feedback ($f^z_{a,i} > 0$) corresponds to a request for shorter waiting times ($z = w$) or higher velocities ($z = v$), while negative feedback ($f^z_{a,i} < 0$) indicates a preference for longer waiting times or lower velocities. In the following, the case $z = w$ is examined; nonetheless, analogous considerations apply to the case $z = v$ as well. If the feedback $f^z_{a,i} < 0$ (with $z = w$), the algorithm checks whether the current parameter value $z_{a,i}$ is greater than or less than its previous value $z_{a,i-1}$. An increase in $z_{a,i}$ (line 2) indicates that the framework has correctly adjusted to the expressed human preference, leading to an increment of ω^z proportionally to the feedback value (line 3). Conversely, a decrease in $z_{a,i}$ results in a corresponding reduction of ω^z, which denotes that the adaptation diverges from the provided human feedback (line 5). In contrast, when $f^z_{a,i} > 0$ (line 6), a decrease in $z_{a,i}$ is expected. When the parameter change is consistent with this expectation, ω^z is incremented to acknowledge successful adaptation (line 10); otherwise, it is decremented to indicate misalignment with the feedback (line 8).

Algorithm 5 Adaptation index computation

Require: feedback variable z with $z \in \{w, v\}$, human feedback $f^z_{a,i}$, previous adaptation index ω^z_{prev}

if $f^z_{a,i} < 0$ **and** $z = w$) **or** ($f^z_{a,i} > 0$ **and** $z = v$) **then**
▷ Human asks for longer waiting time ($z = w$) or faster robot ($z = v$)
  if $z_{a,i} > z_{a,i-1}$ **then**
    $\omega^z \leftarrow \omega^z_{prev} + \left| \frac{1}{f^z_{a,i}} \frac{z^{a,i} - z_{a,i-1}}{z_{a,i}} \right|$
  else if $z_{a,i} < z_{a,i-1}$ **then**
    $\omega^z \leftarrow \omega^z_{prev} - \left| \frac{1}{f^z_{a,i}} \frac{z_{a,i} - z_{a,i-1}}{z_{a,i}} \right|$
if ($f^z_{a,i} > 0$ **and** $z = w$) **or** ($f^z_{a,i} < 0$ **and** $z = v$) **then**
▷ Human asks for shorter waiting time ($z = w$) or slower robot ($z = v$)
  if $z_{a,i} > z_{a,i-1}$ **then**
    $\omega^z \leftarrow \omega^z_{prev} - \left| \frac{1}{f^z_{a,i}} \frac{z_{a,i} - z_{a,i-1}}{z_{a,i}} \right|$
  else if $z_{a,i} < z_{a,i-1}$ **then**
    $\omega^z \leftarrow \omega^z_{prev} + \left| \frac{1}{f^z_{a,i}} \frac{z_{a,i} - z_{a,i-1}}{z_{a,i}} \right|$

This chapter also presents a comparison between the proposed framework, which reallocates activities in response to human feedback, and a baseline version that neither incorporates feedback nor executes reallocations. For this purpose, 30 independent simulation runs were conducted, progressively increasing the number of human agents from 2 to 6, while fixing the number of boxes to be filled ($q_a, \forall a = 3$) and the number of service robots ($n^m = 2$). Human positions and feedback were randomly assigned: positions were uniformly sampled by varying the x and y coordinates in the interval $[3, 10]$, and feedback values were randomly chosen from the five alternatives illustrated in Figure 5.5.

Figure 5.9 shows the average and standard deviation values for ω^w obtained over the 30 runs (Similar results were obtained for ω^v). Specifically, the blue line describes the evolution of the index ω^w related to the proposed framework across the considered scenarios, while the orange line describes the evolution of the index ω^w associated with the framework that does not integrate human feedback. In the latter case, it can be observed that the index ω^w does not change significantly and remains nearly constant at a value of 1, indicating that the system fails to accommodate human needs. In contrast, with the proposed framework, the index ω^w exhibits an initial value of approximately 3.5 for $n^h = 2$ and gradually decreases to 2.3 as n^h reaches 6, thereby reflecting a more responsive

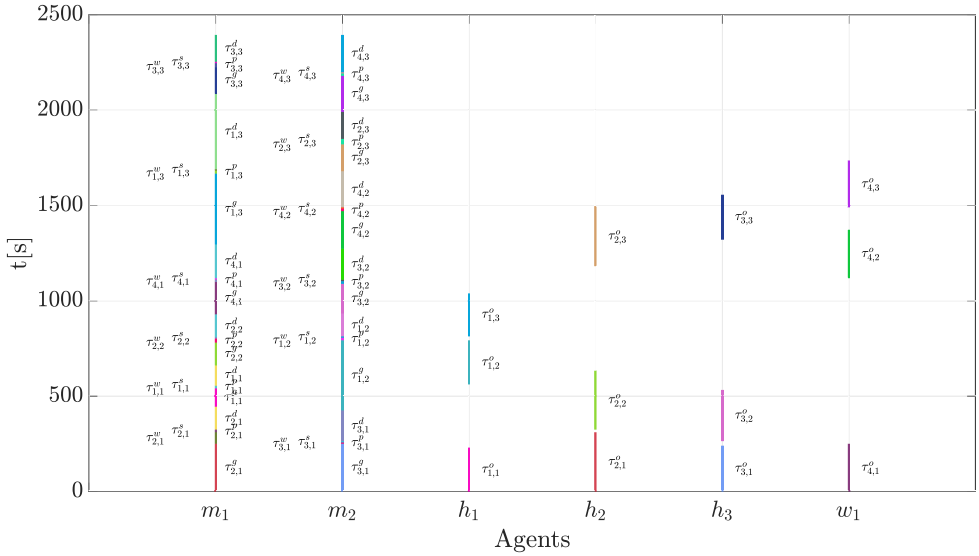


Figure 5.9: Comparison of the adaptation index with and without considering human feedback.

adaptation to human requests/feedback. The obtained results highlight two main points: *i*) the incorporation of human feedback has enabled the framework to effectively accommodate human needs, yielding a consistently higher adaptation index, and *ii*) as the number of human agents increases, the framework is subject to increasing difficulty in meeting all demands owing to the fixed and limited availability of service robots.

5.7.5 Human Stochasticity

In order to validate the proposed chance-constrained programming formulation for addressing uncertainty in human agents, a series of simulations was carried out with $n^a = 4$ working agents, each responsible for processing $q_a = 3$ boxes and assisted by $n^m = 2$ service robots. For each agent a, the processing time was modeled with a mean of $\varphi_a = 250 + 10a$ and a standard deviation of $\sigma_a = 0.1\varphi_a$. Six confidence levels, $pr_{\min} \in \{0.6, 0.7, 0.8, 0.9, 0.95, 0.99\}$, were tested. For each level, 50 duration samples per agent were generated according to the corresponding Gaussian distribution. Each sample was evaluated against the scheduled duration specified in the plan. Two performance indicators

were then computed: *i)* the feasibility rate (ranging from 0 to 1), i.e., how often the allocated duration was adequate to ensure task completion, and *ii)* the mean deviation between actual and planned durations for the feasible cases. Table 5.4 summarizes the results and shows that increasing the confidence level leads to a rise in both the feasibility of the allocated duration and the deviation from the actual task duration. This is because, as $pr_{\min}$ increases, the strategy becomes more conservative in assigning task durations, ensuring a higher probability of successful operation completion at the expense of longer planned durations.

$pr_{\min}$	Feasibility	Deviation [s]
0.6	0.60	27.924 ± 20.252
0.7	0.69	33.986 ± 22.087
0.8	0.79	39.871 ± 24.817
0.9	0.91	48.305 ± 28.620
0.95	0.96	57.399 ± 28.688
0.99	0.99	77.815 ± 31.530

Table 5.4: Impact of confidence level on duration feasibility and deviation from actual values.

5.7.6 Scalability Analysis

This section first evaluates the efficiency of the CP formulation relative to the MILP formulation, and then validates the batch decomposition strategy for large-scale systems. With regard to the first point, a comparative analysis was conducted by assigning single operations to each working agent, while task durations for the service robots were generated randomly. Both the MILP and CP formulations used identical duration vectors and the IBM ILOG CPLEX solver to guarantee methodological fairness. The number of working agents (n^a) and service robots (n^w) was systematically varied, and the two formulations were evaluated in terms of objective function values and computation times. Table 5.5 summarizes the comparative analysis. The results show that: *i)* both formulations consistently produce the same objective value, confirming that the CP-based approach preserves optimality; and *ii)* the CP formulation consistently requires less computation time than the MILP formulation. For example, as n^a increases from 1 to 10, MILP time grows sharply (from 0.001 s to 150.7 s), while CP remains efficient (from 0.001 s to 0.08 s).

Conversely, increasing n^w has minimal impact on computation time, since additional resources simplify the assignment.

# Agents		Computation time		Cost	
n^a	n^m	CP	MILP	CP	MILP
1	2	0.001	0.001	273	273
2	2	0.01	0.001	273	273
3	2	0.05	0.12	441	441
4	2	0.03	0.17	444	444
5	2	0.03	0.40	650	650
6	2	0.04	0.70	691	691
7	2	0.05	1.03	844	844
8	2	0.05	1.13	919	919
9	2	0.07	68.51	1071	1071
10	2	0.08	150.71	1154	1154
4	3	0.03	0.09	411	411
4	4	0.01	0.01	217	217
4	5	0.001	0.01	217	217
4	6	0.01	0.001	217	217
4	7	0.001	0.01	217	217
4	8	0.01	0.01	217	217
4	9	0.01	0.01	217	217
4	10	0.02	0.02	217	217

Table 5.5: Comparison between MILP and CP formulations.

As for the batch decomposition strategy, it was compared with the global approach, which allocates and schedules all activities simultaneously on relatively small instances solvable in a single run. For larger-scale systems, the computational performance of the batching strategy was further examined to evaluate its scalability. All experiments were carried out using the CP-based implementation, with two service robots and a single operation per working agent. In the decomposition strategy, the batch size was fixed at $b_s = 4$, and operations were ordered based on their average distance to the deposit stations of the service robots. Table 5.6 reports the results as the number of working agents increases, which constitutes the most computationally demanding scenario, as previously discussed.

For the global method, only the computation time and the non-normalized cost function were reported. In contrast, the batch method included the mean and standard deviation of per-batch solution times as well as the relative error of the total objective value with respect to the global optimum. The results show that the batch strategy achieves consistently low relative error (always below 0.04) while significantly reducing computation time. For instance, at $n^a = 10$, the global approach takes about

	Global		Batch	
n^a	Time [s]	Cost	Time [s]	Relative Cost Error
5	0.350	829.700	0.330 ± 0.255	0.0004
6	1.220	940.450	0.207 ± 0.032	0.0434
7	2.730	1088.333	0.323 ± 0.283	0.0008
8	4.770	1213.558	0.405 ± 0.233	0.0012
9	7.950	1376.055	0.317 ± 0.535	0.0017
10	41.750	1480.800	0.746 ± 0.984	0.0099

Table 5.6: Comparison between the global solution and the batch decomposition strategy.

41.8 s, whereas the batch method requires less than 0.75s per batch. Table 5.7 reports times for larger systems ($n^a = 15$ to 30), where computing the global solution is impractical. Even in these cases, the batch method remains efficient, with average computation times under 8 s, confirming its scalability.

n^a	Computation time [s]
15	0.640 ± 0.745
20	5.015 ± 6.771
25	4.596 ± 7.0416
30	7.757 ± 10.142

Table 5.7: Results with large-scale systems.

5.8 Real-world Experiments

This section presents the real-world laboratory validation of the proposed framework using the same agricultural scenario as in the simulation case study. The validation has been conducted within the laboratories of Roma Tre University as part of a collaboration.

The experiment involved two human workers performing harvesting tasks and two service mobile robots. The ROS architecture, human interface, and low-level controller used in the real-world setup were identical to those employed in the simulation. Unlike the simulations, the components were distributed across multiple machines in this case. Specifically, each low-level controller was run on the machine mounted on the corresponding service robot, a NUC with an Intel Core i7 CPU, and 16GB RAM. As shown in Figure 5.10, two TurtleBot2 platforms were used as service robots, whose localization was obtained through an OptiTrack

system capable of estimating position and orientation with an accuracy of about 10^{-4} m.

The waiting distances were set to $l^p_{1,m} = l^p_{2,m} = 1$ m, while the humans' filling times were modeled as a Gaussian distribution $\mathcal{N}(\varphi_a, \sigma_a)$, with $\varphi_1 = \varphi_2 = 97$, $\sigma_a = 0.1\varphi_a \ \forall a \in \mathcal{H}$, and the minimum probability was set as $pr_{min} = 0.9$. Since homogeneous robots are employed, the parameters μ_1 and μ_2 were both set to 1. Furthermore, the general velocity limits and the proximity phase velocity limits of both robots were set as follows: $v^c_{\min,m} = 0.05$ m/s, $v^c_{\max,m} = 0.2$ m/s, $v^p_{\min,a,m} = 0.04$ m/s; and $v^p_{\max,a,m} = 0.15$ m/s, $\forall m \in \mathcal{M}$ and $\forall a \in \mathcal{H}$. Regarding the severity factors, it was assumed that both humans involved in the experiment experienced the same level of discomfort. Accordingly, the values were set to $\rho^s_1 = \rho^s_2 = 0.5$. Both humans perform three box-filling operations throughout the entire experiment. Finally, the cost function weights were defined as follows: $\alpha = \beta = \gamma = \zeta = \kappa = 1$. The robots deposit bases were located for m_1 at $x^d_1 = [2.08, 1.48]$ and for m_2 at $x^d_2 = [-1.33, -2.14]$. The humans' service positions were as follows, h_1 at $x^s_1 = [-0.5, -1.2]$ and h_2 at $x^s_2 = [0.5, 0.63]$.

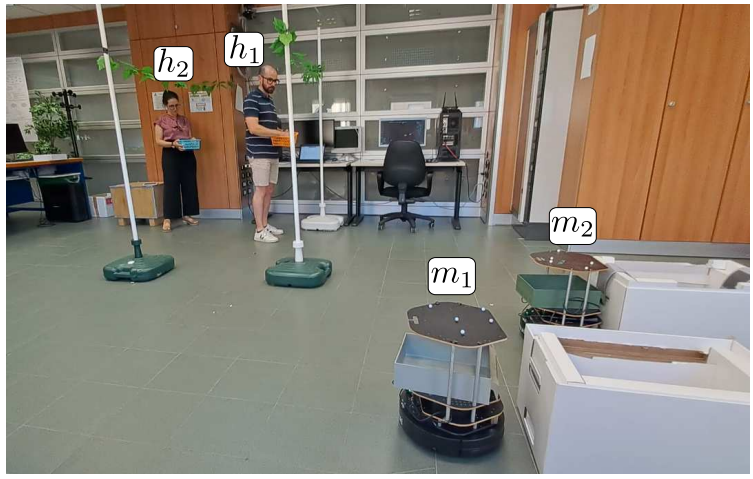
Figure 5.10 shows the key moments of the laboratory experiment, while Figure 5.11 shows the first allocation plan, i.e., the allocation plan formulated by the framework using the configuration described at the beginning of this section, and Figure 5.12 displays the plan resulting from the reallocation phase after both humans gave their feedback.

More in detail, Figure 5.10.a) shows the robots at the deposit station at the beginning of the experiment. The second picture, i.e., Figure 5.10.b), depicts Robot 1 at the service position related to Human 1. During this phase, the exchange ends at $t = 71$ s. Then, the human provides feedback about the robot's service. The feedback on the robot's arrival was *Very Fast* ($f^v_{2,1} = -1$) with a $\zeta_{1,k} = 1$, but the waiting time was considered *Too Long* ($f^w_{1,k} = 1$) leading to $\alpha_{1,k} = 1.97$ with $k > 1$. Additionally, Human 1 requested that the robot wait *Closer*, providing the feedback *Far*. Therefore, the distance was updated to $l^p_{1,m} = 0.5$ m and the proximity phase duration, i.e., $\overline{p}_{1,2} - \underline{p}_{1,2}$ was 25 s leading to a slower robot. The filling phase for the second task began at $t = 153$ s to accommodate the waiting time feedback. Meanwhile, Robot 2 is shown in the waiting

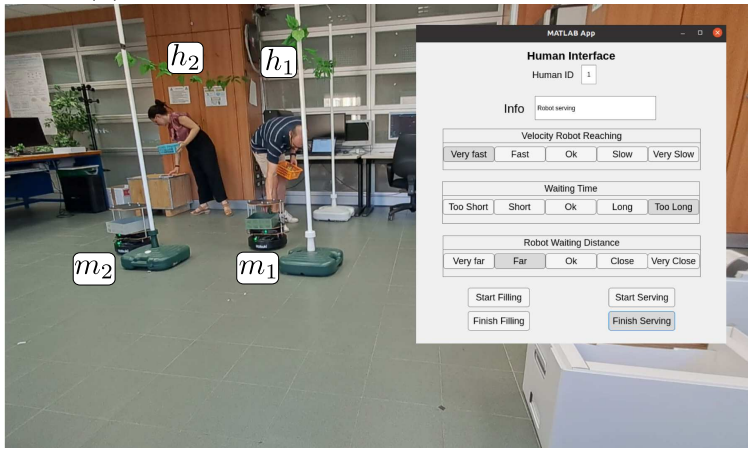
position, standing for Human 2 to finish filling. The final picture, Figure 5.10.c), shows Robot 1 in the going phase as it returns to the depot, while Robot 2 is completing its service at 162 s. Human 2 is also providing feedback, stating that the robot was *Very Slow* ($f^v_{2,1} = 1$) during the proximity phase, which led to a $\zeta_{2,k} = 1.94$ with $k > 1$. The waiting time was perceived as *Too Short* ($f^w_{2,1} = -1$) leading to $\alpha_{2,k} = 0.84$ with $k > 1$, and the waiting distance was *Close*, indicating a preference for the robot to wait slightly farther away. Based on these preferences, the allocation and scheduling were recomputed to adapt to this feedback, leading to a proximity phase duration updated to 14.24 s for a faster robot and a waiting distance $l^p_{2,m} = 1.5$ m. The velocity bounds were also modified as follows according to Algorithm 2: $v^p_{\min,2,1} = 0.036$ and $v^p_{\max,2,1} = 0.135$. Regarding the waiting time, the filling phase for the second task began at $t = 170$ s to increase the waiting time for the Human 2.

5.9 Conclusion

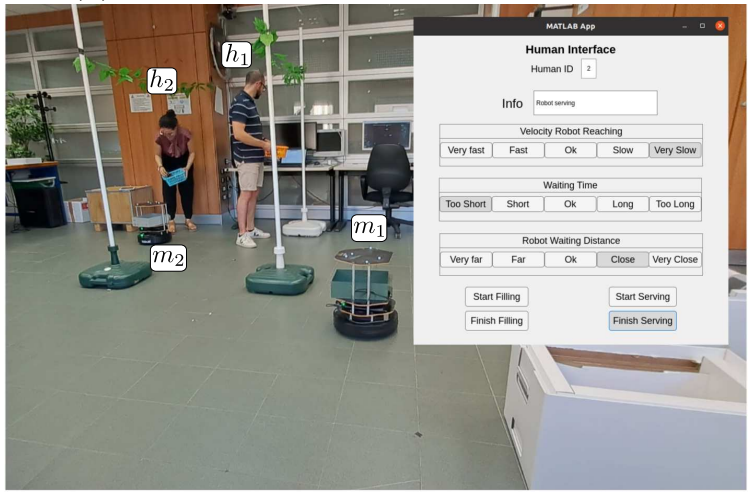
This chapter presents a dynamic framework for task allocation and scheduling, designed to enhance human-robot collaboration within the domain of precision agriculture. By employing a MILP formulation, the framework optimally assigns tasks to service robots while reducing waiting times, makespan, and energy consumption. The principal contributions of this study reside in the explicit integration of human stochasticity and real-time feedback, which allow the system to adapt to evolving operator requirements and to scale effectively to larger and more complex environments. Validation was conducted through both simulations and field trials in agricultural contexts, demonstrating the framework's effectiveness and robustness in environments characterized by variability in human behavior and external conditions. Future research will focus on integrating vision-based systems for behavioral interpretation and action prediction, thereby enhancing the framework's ability to better accommodate human needs and preferences.



(a) The two robots are at the deposit station.



(b) Robot 1 is serving while Robot 2 is waiting.



(c) Robot 1 is going back to the depot while Robot 2 is serving.

Figure 5.10: Key moments of the laboratory experiment.

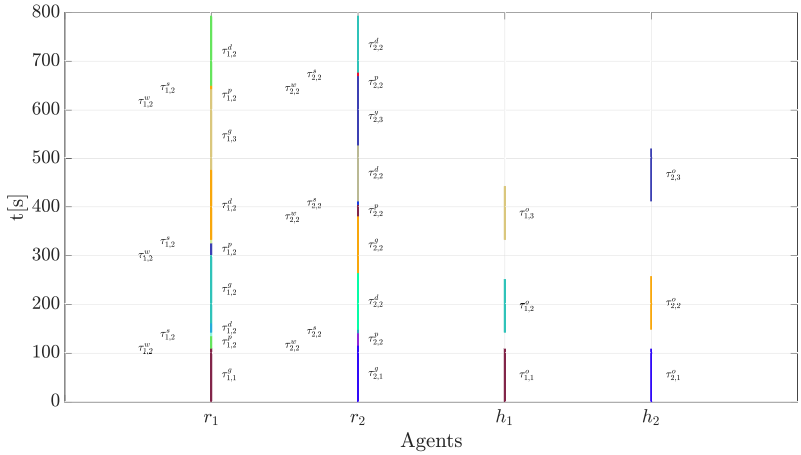


Figure 5.11: Laboratory experiment. First allocation using initial parameters for the human timings.

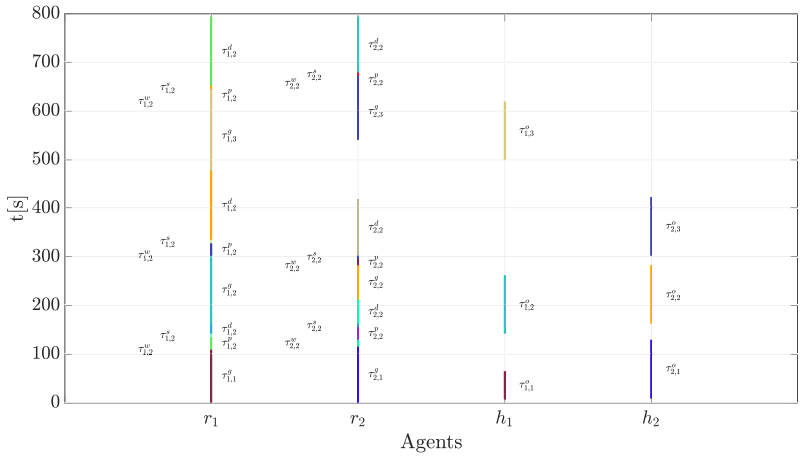


Figure 5.12: Laboratory experiment. Online schedule adjustment based on updated human parameters after the first operation.

Chapter 6

A Hybrid LLM-CP Framework for Reliable Task Planning in Heterogeneous Multi-Agent Systems

6.1 Introduction

As extensively discussed in Chapter 5, the introduction of a higher-level control layer dedicated to multi-agent coordination is essential to address many of the challenges associated with the deployment of such systems, especially those integrating multi-human–multi-robot. More in detail, the previous chapter emphasized that two of the most critical challenges emerging from the deployment of multi-agent systems (MASs) are: *i*) the inherent unpredictability of human behavior, often shaped by factors such as stress, fatigue, environmental context, and inter-agent trust; and *ii*) the heterogeneous nature of these systems, whose components, in addition to exhibiting complementary capabilities, also differ in perception, decision-making, and task execution. To address these challenges and fully benefit from the advantages resulting from their deployment, an optimal allocation and scheduling framework was introduced, capable of dynamically reallocating and rescheduling tasks across agents in such a way as to: *i*) optimize multiple performance metrics, including the completion time and energy consumption of robotic agents; *ii*) meet all specified constraints; and *iii*) reflect human preferences. However, fundamental

challenges remain, including the translation of high-level, often ambiguous, natural language instructions from human operators into a series of efficient and executable plans for these diverse agents [144, 145]. Up to recent years, this problem has been addressed through a variety of formal planning methods, which are capable of providing provably optimal or near-optimal solutions, and a comprehensive survey of cooperative multi-agent planning can be found in [146]. These approaches, including classical planners based on Planning Domain Definition Language (PDDL), Constraint Programming (CP), and Mixed-Integer Linear Programming (MILP), offer strong formal guarantees regarding plan correctness, feasibility, and resource allocation [147, 148, 149]. However, they require that the problem domain, agent capabilities, and task dependencies are formally encoded according to a rigid, structured formalism. For example, a PDDL-based framework, as seen in the work of [150], requires the manual formulation of domain and problem files, along with the definition of action costs, which are then fed into a planner to generate an optimal plan. Similarly, other multi-stage approaches, like the one proposed by [151], rely on a multi-stage procedure to facilitate the objective allocation of activities based on agent capabilities. This reliance on explicit, manually created domain models and formal knowledge remains a significant bottleneck to the adoption of these formalisms in complex real-world scenarios. This manual formalization process not only entails considerable effort and deep expertise in logic and planning languages but also imposes several limitations on flexibility and scalability. [152, 145].

In recent years, the rapid advancement of Large Language Models (LLMs) has opened a new paradigm for robot task planning, paving the way for a more flexible and intuitive approach. As emphasized in Section 2.8, LLMs possess a remarkable capacity for natural language understanding and high-level reasoning, allowing them to directly interpret human instructions and generate task plans with high flexibility [144, 153]. They can decompose complex missions into hierarchical trees [153], generate plans, allocate tasks, and even facilitate multi-agent coalition formation [144]. These LLM-driven frameworks, such as the one in [154], which translates natural language into action sequences for an embodied agent, provide a powerful and intuitive interface that allows non-expert users to utilize them within MAS scenarios. Other works, such as those by [155], [156], [157], and [158], have explored similar concepts, demonstrating the

potential of LLMs to enhance the capabilities of MASs. However, a major challenge persists: plans generated by LLMs alone often lack formal correctness and feasibility, presenting logical inconsistencies that can lead to unsafe or inefficient robot behavior [159, 160]. For instance, an LLM might generate a plan that is physically impossible due to spatial constraints or agent capabilities that are not properly considered, making their reasoning unreliable [161, 162].

The absence of formal guarantees and the potential for unreliable, or even "hallucinated", outputs generated by LLMs highlights the inherent limitations of a purely data-driven approach, especially when employed for real-world applications. For example, an LLM might generate a plan that violates critical safety constraints, propose a sequence of actions that is not physically executable, or allocate a task to an agent that lacks the necessary skills or resources. As highlighted by [159], ensuring that task plans and allocation outputs from LLMs are adequately examined and refined is crucial for maintaining system integrity and safety. These models lack a formal grounding mechanism, i.e., a robust mechanism for reasoning over complex spatiotemporal constraints, which may result in generating plans that are inefficient or physically unfeasible, or could lead to system malfunctions and/or resource waste. Despite their promise, a purely LLM-based approach is still not able to tackle complex Operations Research problems [163, 164].

To address the inherent limitations of both traditional formal methods and LLM-only techniques, this chapter introduces a two-layer hybrid architecture that combines the advantages of both approaches. In detail, the outlined framework is designed to synergistically combine the high-level semantic reasoning and flexibility of LLMs with the formal guarantees and optimization capabilities of CP. The first layer, powered by an LLM, is responsible for interpreting a natural language mission description and semantically decomposing it into a structured, ordered sequence of tasks. The second layer takes this structured plan and formulates it as a CP problem, enabling the efficient allocation of tasks and scheduling of actions across a heterogeneous team of agents while respecting temporal, resource, and agent-specific constraints. By exploiting the CP solver for the complex, combinatorial optimization and correctness checks, the framework ensures that the final executable plan is not only logically sound and feasible but also optimized according to the given metrics. The framework has been validated against two baseline approaches in a

multi-agent scenario [165, 166], demonstrating that the presented hybrid approach significantly improves plan reliability, feasibility, and optimality while leveraging the natural language flexibility and generalization capabilities of LLMs.

An example of problems that can be tackled using the above-mentioned framework is presented below.

Problem 2. *Consider a heterogeneous MAS comprising human, $\mathcal{H}$, and robotic, $\mathcal{R}$, agents endowed with diverse skills. Consider a set of objects $\mathcal{O}$ available in the shared workspace, where each object is characterized by distinctive properties. A high-level instruction, formulated in natural language, provides a description of the scenario, the team, and the overall mission to be accomplished. The framework goal is twofold:* i) *to formally establish the mission context, i.e., derive the set of tasks, $\mathcal{T}$, required for mission completion and identify the respective set of physical (e.g., properties of objects and/or tools to be employed) and temporal (e.g., possible precedence) constraints, and* ii) *define the optimal allocation and schedule of the agents' activities to accomplish the mission while minimizing a cost function accounting for the energy of robotic agents, the workload of human operators, and the overall makespan, while fulfilling all execution constraints.*

The rest of the chapter is organized as follows. Section 6.2 outlines the reference scenario, providing details about the agents and the tasks carried out by the team. Section 6.3 introduces the proposed framework, detailing its architecture and formalizing the optimization problem formulated to manage the allocation and scheduling of tasks. Finally, Sections 6.4, 6.5, and 6.6 report and discuss the simulated and real-world experiments conducted to validate the effectiveness of the proposed framework.

6.2 Scenario

This section details the problem under consideration, as well as the corresponding scenario. The notation (re-)introduced in this and subsequent chapters is summarized in Table 6.1.

As emphasized in Section 6.1 and later in Problem 2, a scenario comprising a set of heterogeneous agents, denoted by $\mathcal{A}$, cooperating within a

VARIABLE	MEANING
$k \in \mathcal{K}$	k-th element of the generic set $\mathcal{K}$.
$\lvert\mathcal{K}\rvert$	Cardinality of a generic set $\mathcal{K}$.
$\mathcal{A}$, $\mathcal{R}$, $\mathcal{H}$, $\mathcal{O}$, $\mathcal{T}$	Sets of agents, robot operators, human operators, objects, and tasks.
$\mathcal{S}_a$	Set of primitive actions that the agent a is able to perform.
$\boldsymbol{p}_a$ ($\boldsymbol{p}_o$)	Base location of agent a (object o).
m_o, $\mathcal{F}_o$	Mass and feature set of object o.
$\mathcal{M}_\tau$, $\mathcal{F}_\tau$, s_τ	Motion path instructions, Feature set of the required object, and Primitive action for task τ.
$d_{a,\tau,o}$	Distance that agent a has to travel while performing task τ with object o.
$d_{\max}$	Maximum travel distance.
$\delta_{a,\tau,o}$	Time needed by agent a to cover the distance $d_{a,\tau,o}$.
$v_{r,\min}$ ($v_{r,\max}$)	Minimum (Maximum) velocities of robot r.
$v_h(s,o)$ ($v_{h,\max}$)	Actual operating speed (Maximum velocity) of human h.
$v_{\min}$	Overall minimum velocity.
Δ_τ	Interval decision variable for task τ.
$\overline{\Delta}_{a,\tau}$	Optional interval decision variable related to assigning task τ to agent a.
$X_{\tau,o}$	Binary decision variable related to assigning object o to task τ.
μ	Normalized Makespan.
$\omega_{h,\tau,o}$	Workload of the human h while performing task τ with object o.
ω_h	Overall normalized workload of the human h.
$\omega_{h,\max}$	Maximum workload of the human h.
$\rho_{r,\tau}$	Power consumption of the robot r for the execution of task τ.
e_r	Overall normalized energy of robot r.
β, γ, ν	Cost function weights.
b_s	Batch size.
ϵ_m (ϵ_M)	Minimum (Maximum) relative error.

Table 6.1: Main notations (re-)introduced in the chapter.

shared environment to fulfill a common mission is considered. In the general case, this set may include both human operators, forming the set $\mathcal{H}$, and robotic agents, forming the set $\mathcal{R}$; thus, it holds $\mathcal{A} = \mathcal{H} \cup \mathcal{R}$. Moreover, the scenario comprises a set of objects, denoted as $\mathcal{O}$. Each object $o \in \mathcal{O}$ is located at a certain place at planning time, has a certain mass, and is characterized by distinctive features (e.g., shape, color, material). In detail, the object's position vector is denoted by $\boldsymbol{p}_o \in \mathbb{R}^3$, its mass by m_o, and its feature set by $\mathcal{F}_o$.

Regarding the agents, each agent $a \in \mathcal{A}$ is associated with: *i)* a base location, denoted as $\boldsymbol{p}_a \in \mathbb{R}^3$, and *ii)* a set of skills, denoted as $\mathcal{S}_a$, representing all *primitive actions* that the agent is able to perform. A primitive action may or may not involve an object in the environment. A few examples of primitive actions are pick-and-place, opening/closing, inspection, cleaning, approaching, mapping, and pouring operations. While the position of each agent within the environment may vary over time, it is assumed that every action begins with the agent located at its base position. For each skill $s \in \mathcal{S}_a$ of the agent a, the time to perform it is assumed to consist of two components: a constant term, denoted as $\delta_{a,s}$, e.g., for opening/closing the gripper, and an additional variable term, taking into account the distance that the agent has to traverse, e.g., to reach or manipulate an object, and the kinematic and dynamic constraints of the agent. For each robotic agent $r \in \mathcal{R}$, in addition to the previous parameters, its minimum and maximum velocity intensities are also considered, which are simply referred to as minimum and maximum velocities, respectively, in the following. These are denoted as $v_{r,\min}$ and $v_{r,\max}$, respectively, and it holds $v_{r,\max} \geq v_{r,\min} > 0$. The minimum velocity is due to several factors, such as, for example, effects of friction [167] and quantization of sensors' information [168], which are particularly relevant for motor servos at low speed, while the maximum velocity is related to the physical capabilities of the robot. Similarly, for each human operator $h \in \mathcal{H}$, his/her maximum operating speed $v_{h,\max}$ is considered. Then, the actual operating speed of the human h generally depends on different factors, like the primitive action involved $s \in \mathcal{S}_h$ (e.g., a welding operation may require slow and precise motion) and the type of possible object involved $o \in \mathcal{O}$ (e.g., handling a heavy object might lead to a slow motion). Hence, this actual operating speed is denoted as a function of the skill and object as follows $v_h(s, o)$, for which it holds that $v_h(\cdot) \leq v_{h,\max}$. Clearly, if no object is involved in the task performed by the human, the speed only depends on the skill, in addition to the specific agent. Based on the above velocity quantities, the overall minimum velocity of the agents can be defined as $v_{\min} = \min_{r \in \mathcal{R}, h \in \mathcal{H}, s \in \mathcal{S}_h, o \in \mathcal{O}} \{v_{r,\min}, v_h(s, o)\}$.

As stated in Problem 2, in this scenario, the agents cooperate to complete a common mission. A mission can be viewed as a set of tasks, denoted as $\mathcal{T}$, where each task corresponds to the execution of a primitive action, using certain path motion directives and possibly involving an object.

Specifically, for each task $\tau \in \mathcal{T}$, the corresponding primitive action s_τ is defined, along with the set of motion path instructions $\mathcal{M}_\tau$, collecting for instance the relevant poses to traverse and/or the type of path to realize (e.g., circular, linear), and the feature set of the involved object $\mathcal{F}_\tau$, which is empty if no object is involved. Furthermore, to assess if an agent $a \in \mathcal{A}$ is able to perform a task, a function $c_a(s_\tau, \mathcal{M}_\tau, \mathcal{F}_\tau) \in \{0, 1\}$ is introduced, which evaluates the capability of the agent to carry out the task τ. This returns 1 if the agent is capable, and 0 otherwise. Additionally, in order to accomplish a given mission, it is often necessary to enforce temporal constraints among the tasks. Following the layering approach in [169], the concept of ordered layers is introduced, denoted as $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_{n_L}$, with n_L denoting the number of total layers. Each layer $\mathcal{L}_i$ consists of a set of mutually independent tasks, i.e., they can be executed in parallel, and all tasks in $\mathcal{L}_i$ must be completed before any task in $\mathcal{L}_{i+1}$ can start. Based on these layers, a layer assignment function $l(\tau)$ is defined that associates each task τ in the set $\mathcal{T}$ with a respective priority level. Given two tasks $\tau, \rho \in \mathcal{T}$, the notation $l(\tau) \succ l(\rho)$ is used to denote that ρ is assigned a lower priority layer than τ.

6.3 Optimization Framework

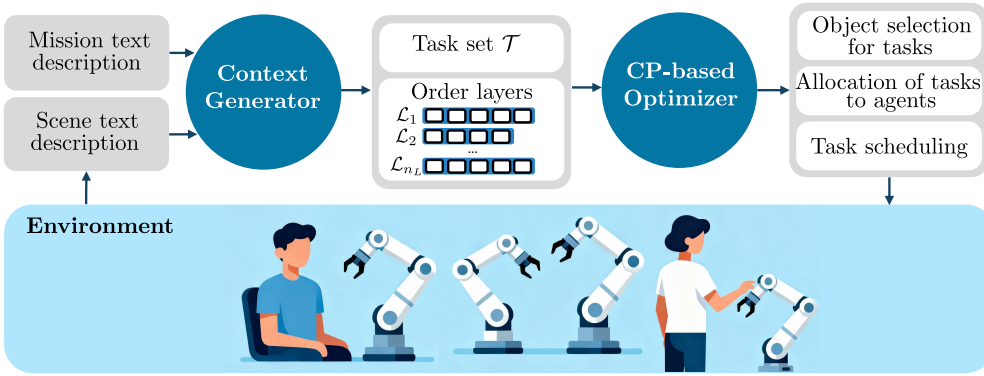


Figure 6.1: Overview of the proposed architecture

To solve Problem 2, a two-layer framework has been designed (Figure 6.1), composed of: *i*) a Context Generator, an LLM-based module designed to decompose the high-level mission into tasks and arrange them into a hierarchical structure; and *ii*) a CP-based Optimizer, an optimization module

dedicated to modeling and solving the constrained optimization problem to define the optimal allocation and scheduling of the tasks while fulfilling all execution constraints. This architecture enables the user to express high-level missions for the heterogeneous multi-agent system in natural language, while formally preserving optimality and feasibility properties of the task allocation and scheduling solution. This architecture allows the user to express high-level missions for the heterogeneous MAS in natural language, while formally preserving optimality and feasibility properties of the task allocation and scheduling solution. As extensively demonstrated in Section 6.4, this hybrid approach, which integrates the general knowledge of LLMs with the guarantees provided by optimization, consistently outperforms purely LLM-based solutions that do not include any explicit optimization phase. In the following, a detailed description of each module of the aforementioned framework is provided.

6.3.1 Context Generator

As previously mentioned, the first layer is responsible for decomposing the overall mission, provided by the user in natural language, into a sequence of tasks and arranging them into a hierarchical structure. This overall process is referred to as *context generation*. The procedures in this process are carried out by leveraging the world knowledge of pre-trained LLM models. The description of the mission, along with the scenario description and the set of directives to be observed, is reported in the prompt files, i.e., the user and system prompts, supplied as input to the LLM model. More in detail, the user prompt is structured as a natural language description of the mission to be accomplished, while the system prompt is designed as a JSON-like natural language description, including details of the scenario and the directives. The system prompt is organized into three sections: *i)* scene, *ii)* actions, and *iii)* directives. The scene section provides a taxonomy of the key elements in the environment, including a formal description of the spatial locations where objects can be processed or actions performed, as well as a structured representation of the available objects and their distinctive properties (e.g., shape, color, instances, position). The actions section lists the primitive actions that can be performed within the scene. The directives section contains rules guiding the LLM in generating the plan. Specifically, these can provide specific constraints that must be fulfilled to successfully complete

the mission (e.g., objects cannot overlap) and specify the required output structure and format (i.e., JSON). Figure 6.2 contains an illustrative example of both the user and system prompts, with each section highlighted in a different color. The generation of prompt files lies beyond the scope of the activity presented within this chapter; thus, they are assumed to be already available. Their design, however, follows reasonings similar to those discussed in [170, 160].

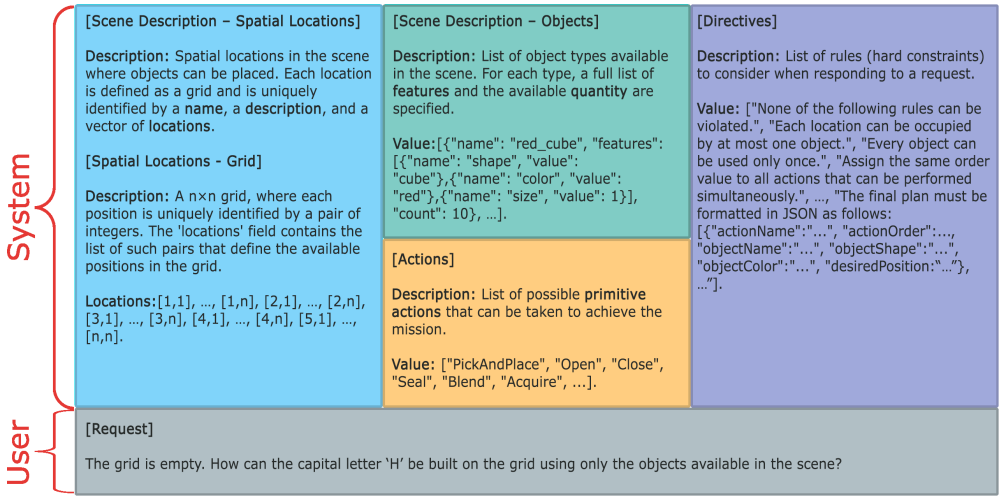


Figure 6.2: A schematic representation of the system and user prompts, with each section highlighted in a different color.

Once the prompt files have been generated, the LLM generates the context for the CP-optimization module. More in detail, the output of the LLM-based context generator module is: *i*) the decomposition of the mission described in the user prompt into a set of tasks ($\mathcal{T}$), where for each task τ the corresponding primitive action (s_τ) and motion instructions ($\mathcal{M}_\tau$) are provided; *ii*) the definition, for each task τ, whether it involves or not an object and, if so, specifying the set of features $\mathcal{F}_\tau$ it must possess; and *iii*) the organization of the tasks into a hierarchical structure, specifying their execution order and identifying those that may be performed in parallel. The outcome of this procedure is employed to derive the layer assignment function $l(\tau)$, $\forall \tau \in \mathcal{T}$.

6.3.2 CP-Based Optimizer

As stated above, the second layer of the architecture in Figure 6.1 is tasked with modeling and solving the constrained optimization problem that aims to define the optimal allocation and scheduling of the tasks given the agents and objects in the environment. In particular, this layer aims to define, for each task, the start and end times, the specific object to be used among those in the scene (if required), and the agent assigned to its execution. In doing so, an objective function, formulated as a weighted sum of relevant cost indices (e.g., makespan, workload, energy consumption), is optimized while ensuring compliance with predefined constraints (e.g., agents' capabilities) and constraints inferred by the LLM during the context generation process (e.g., execution order). A formal definition of decision variables is now introduced: *i*) an interval variable Δ_τ, $\forall \tau \in \mathcal{T}$, representing the time interval during which the task τ takes place; *ii*) a binary variable $X_{\tau,o}$, $\forall \tau \in \mathcal{T}, o \in \mathcal{O}$, encoding whether the object o is assigned to (or equivalently involved in) the task τ ($X_{\tau,o} = 1$) or not ($X_{\tau,o} = 0$); *iii*) an optional interval variable $\overline{\Delta}_{a,\tau}$, $\forall \tau \in \mathcal{T}, a \in \mathcal{A}$, specifying whether the agent a is responsible for the execution of the task τ; and *iv*) a continuous variable $v_{r,\tau}$, $\forall \tau \in \mathcal{T}, r \in \mathcal{R}$, representing the velocity of the robotic agent r for the task τ, which takes values in the range $[v_{r,\min}, v_{r,\max}]$ if r is assigned to its execution (presenceOf$(\overline{\Delta}_{r,\tau}) = \mathit{true}$), and equals 0 otherwise (presenceOf$(\overline{\Delta}_{r,\tau}) = \mathit{false}$). Regarding the optional variables, as mentioned in Section 2.7, these are typically employed to represent tasks that may or may not be executed. Instead, in our work, they are used to model alternative execution modes. For each task τ, a non-optional interval variable is introduced, representing its execution interval, along with a set of optional interval variables corresponding to the potential executing agents ($\overline{\Delta}_{a,\tau}$, $\forall a$). Among these, only the variable corresponding to the agent selected for execution will be included in the solution and will have the same values as the main interval variable of the task, while the remaining optional variables will be absent from the solution. Based on the context generator outputs, this layer encodes the constraints that the scheduling and allocation processes must satisfy and defines the cost function to be minimized. Finally, by leveraging the provided CP solver (e.g., IBM ILOG CPLEX[1]), it solves the formulated constraint optimization problem. Once the problem has been solved and

[1] `https://www.ibm.com/it-it/products/ilog-cplex-optimization-studio`

the corresponding decision variables have been determined, the execution of the generated plan can begin. During this phase, the robots' trajectories are progressively computed, while humans are gradually notified of the tasks assigned to them. Due to the intrinsic unpredictability of human behavior and the variability of real-world conditions, deviations from the nominal conditions considered in the planning phase may occur during the execution of the generated plan. For example, one or more agents may become unavailable or unable to complete specific operations. In such cases, one possible corrective measure for preserving the initial level of optimality as much as possible is the adoption of online re-allocation and re-scheduling strategies, such as those introduced in Chapter 5. It is worth pointing out that, in some cases, the formulated problem may turn out to be infeasible. This may occur when the plan generated by the LLM includes, as a result of hallucination phenomena, actions not specified in the system prompt or objects not available in the scene. In such cases, the entire procedure is restarted from the context generation process until a feasible plan is found.

6.3.2.1 CP-based formulation

The CP-Based Optimizer module addresses the allocation and scheduling procedure, i.e., determines the values of the binary variables $X_{\tau,o}$, the interval variables Δ_τ and (the optional ones) $\overline{\Delta}_{a,\tau}$, and the floating-point variables $v_{r,\tau}$. To determine the optimal allocation of objects to tasks, the assignment of tasks to agents, and the scheduling of task start and end times, the following optimality indices are considered: *i*) the makespan, which denotes the overall execution time, i.e., the total time required to complete all tasks in the plan and fulfill the mission specified in the user prompt; *ii*) the overall workload, defined as the total "effort" of the human operators; and *iii*) the cumulative energy consumption, which is defined as the total energy expended by all robotic agents while performing their assigned tasks.

For clarity of presentation and to avoid repeatedly distinguishing two similar cases, it is in the following assumed that each task τ involves an object, i.e., $\mathcal{F}_\tau \neq \emptyset$. The case without an object can be derived analogously. Furthermore, a distance function $d_{a,\tau,o}$ is introduced to denote the distance traveled by an agent $a \in \mathcal{A}$ in performing the task τ involving

object o. The maximum possible distance traveled is denoted as $d_{\max}$. The normalized makespan, indicated as μ, is computed as

$$\mu = \frac{\max\limits_{\tau \in \mathcal{T}} \text{endOf}(\Delta_\tau)}{|\mathcal{T}|\,(\delta_{\max} + d_{\max}/v_{\min})}, \tag{6.1}$$

with $\delta_{\max} = \max\limits_{a \in \mathcal{A}, s \in \mathcal{S}_a} \delta_{a,s}$ and $\text{endOf}(\cdot)$ providing the final time of the corresponding time interval variable. With regard to human workload, let $\omega_{h,\tau,o} \geq 0$ be the workload sustained by human agent h while performing task τ with object o. The overall normalized workload experienced by the human agent h over the plan execution is defined as follows

$$\omega_h = \frac{1}{\omega_{h,\max}} \sum_{\tau \in \mathcal{T},\, o \in \mathcal{O}} [\text{presenceOf}(\overline{\Delta}_{h,\tau}) \wedge X_{\tau,o}]\, \omega_{h,\tau,o}, \tag{6.2}$$

where $\omega_{h,\max}$ is a normalization factor representing the maximum workload that human h could face when performing a task. In the numerator, for each task and object, if the human h is responsible for performing the task τ ($\text{presenceOf}(\overline{\Delta}_{h,\tau}) = \textit{true}$), involving the object o ($X_{\tau,o} = 1$), then the respective workload $\omega_{h,\tau,o}$ is included in the human workload.
To formalize the robots' energy consumption, the physical model governing the energy dynamics of each robot is first presented. Let $\rho_{r,\tau}$ be the power consumption of the robotic agent r resulting from the execution of task τ. According to [131, 132], it can be modeled as

$$\rho_{r,\tau} = \varrho_{1,r} + \varrho_{2,r} v_{r,\tau} + \varrho_{3,r} v_{r,\tau}^2, \tag{6.3}$$

where $\varrho_{1,r}$ is a constant term representing the power consumption due to internal devices (e.g., computing units and sensors) and internal electric losses, while $\varrho_{2,r}$ and $\varrho_{3,r}$ are constants modeling the effects of friction and inertia. Assuming that each robotic agent moves at a constant velocity, the overall energy expenditure of robot r over the distance $d_{r,\tau,o}$, where o denotes the object assigned to task τ (i.e., such that $X_{\tau,o} = 1$), can be computed as

$$e_{r,\tau,o} = \int_0^{\delta_{r,\tau,o}} \rho_{r,\tau}\, dt = \rho_{r,\tau}\delta_{r,\tau,o} = \\ = \varrho_{1,r}\delta_{r,\tau,o} + \varrho_{2,r}d_{r,\tau,o} + \varrho_{3,r}v_{r,\tau}d_{r,\tau,o}, \tag{6.4}$$

where $\delta_{r,\tau,o}$ denotes the time needed by robot r to cover the distance $d_{r,\tau,o}$ and is defined as follows

$$\delta_{r,\tau,o} = \frac{d_{r,\tau,o}}{v_{r,\tau}} X_{\tau,o}. \tag{6.5}$$

The resulting expression is linear in both the motion duration $\delta_{r,\tau,o}$ and the robot velocity $v_{r,\tau}$. Building on (6.4), the overall normalized energy expenditure of robot r can be formulated as follows

$$e_r = \frac{\sum_{\tau\in\mathcal{T},o\in\mathcal{O}} [\text{presenceOf}(\overline{\Delta}_{r,\tau}) \wedge X_{\tau,o}]\; e_{r,\tau,o}}{\varrho_{1,r}\frac{d_{\max}}{v_{r,\min}} + \varrho_{2,r}d_{\max} + \varrho_{3,r}v_{r,\max}d_{\max}}. \tag{6.6}$$

In the numerator, for each task and object, if the robot r is responsible for performing the task τ ($\text{presenceOf}(\overline{\Delta}_{r,\tau}) = \mathit{true}$), involving the object o ($X_{\tau,o} = 1$), then the respective energy $e_{r,\tau,o}$, quantified using Eq. (6.4), is included in the cumulative energy consumption of robot r. The denominator serves as a normalization factor and represents the maximum energy consumption that robot r could experience performing a task under worst-case conditions, that is, when a task is executed at the lowest admissible traveling velocity ($v_{r,\min}$) and the maximum distance is traversed ($d_{\max}$).

According to Problem 2 and based on the above-defined quantities, the aim is to minimize a cost function J structured as a weighted sum of terms comprising the workload $\omega_{(\cdot)}$ of each human agent, the energy consumption $e_{(\cdot)}$ of each robotic agent, and the overall makespan μ

$$J = \beta \sum_{h\in\mathcal{H}} \omega_h + \gamma \sum_{r\in\mathcal{R}} e_r + \eta\mu, \tag{6.7}$$

where β, γ, and η are non-negative weighting factors. The proposed structure aims to achieve a balance among minimizing human effort, energy

consumption, and completion time, according to their respective weights. To ensure a consistent allocation and scheduling of all tasks, the following constraints are defined.

C1) Task execution order:

$$\text{ifThen}(l(\tau) \succ l(\rho), \text{endBeforeStart}(\Delta_\tau, \Delta_\rho)), \quad \forall \tau, \rho \in \mathcal{T}, \tau \neq \rho. \tag{6.8}$$

Equation (6.8) imposes that tasks must be executed in the order provided by the context generator module. Specifically, for any two distinct tasks $\tau, \rho \in \mathcal{T}$, if ρ is assigned a lower priority level than τ ($l(\tau) \succ l(\rho)$), then the execution of ρ can start only once τ has been completed (encoded by the function $\text{endBeforeStart}(\Delta_\tau, \Delta_\rho)$).

C2) Object allocation to tasks:

$$\text{ifThen}\left(\mathcal{F}_\tau \neq \emptyset, \text{sumOf}([X_{\tau,o}]_{o \in \mathcal{O}'_\tau}) = 1\right) \quad \forall \tau \in \mathcal{T}, \tag{6.9}$$

where $\mathcal{O}'_\tau = \{o \in \mathcal{O} \mid \mathcal{F}_\tau \subseteq \mathcal{F}_o\}$ represents the subset of $\mathcal{O}$ comprising only those objects that exhibit all the features required for task τ, while $[X_{\tau,o}]_{o \in \mathcal{O}'_\tau}$ denotes the vector of decision variables associated with potential object assignments for the task τ, i.e., it includes only the binary variables $X_{\tau,o}$ corresponding to objects endowed with all the features demanded by the task τ. Equation (6.9) enforces that, exactly one object ($\text{sumOf}(\cdot) = 1$) is assigned to each task τ and is selected from those meeting the condition $\mathcal{F}_\tau \subseteq \mathcal{F}_o$.

C3) Object allocation limitation:

$$\text{sumOf}([X_{\tau,o}]_{\tau \in \mathcal{T}'_o}) \leq 1, \quad \forall o \in \mathcal{O}, \tag{6.10}$$

where $\mathcal{T}'_o = \{\tau \in \mathcal{T} \mid \mathcal{F}_\tau \subseteq \mathcal{F}_o\}$ denotes the subset of tasks requiring the features provided by the object o ($\mathcal{F}_\tau \subseteq \mathcal{F}_o$), while $[X_{\tau,o}]_{\tau \in \mathcal{T}'_o}$ represents the vector of decision variables modeling the possible object–task assignments. Equation (6.10) guarantees that each object $o \in \mathcal{O}$ is assigned to at most one task. This constraint is implemented by enforcing, for each object $o \in \mathcal{O}$, that the sum of the binary decision variables modeling its assignment to a task ($[X_{\tau,o}]_{\tau \in \mathcal{T}'_o}$) is less than or equal to one.

Remark 1. *Note that in constraint C3, it is assumed that each object*

can be associated with at most one task. However, if a single object can be reused across multiple tasks, the proposed formulation can be easily extended. In this case, the object–task assignments should be modeled as optional interval variables. Moreover, a noOverlap *constraint (similar to C5 introduced in the following) must then be applied to prevent the simultaneous use of the same object by different tasks. For clarity of presentation, binary assignments are considered.*

C4) Agent assignment to tasks:

$$\text{alternative}(\Delta_\tau, [\overline{\Delta}_{a,\tau}]_{a\in\mathcal{A}'_\tau}), \quad \forall\tau \in \mathcal{T}, \tag{6.11}$$

where $\mathcal{A}'_\tau = \{a \in \mathcal{A} \mid c_a(s_\tau, \mathcal{M}_\tau, \mathcal{F}_\tau) = 1\}$ denotes the subset of agents capable of carrying out the task τ, while $[\overline{\Delta}_{a,\tau}]_{a\in\mathcal{A}'_\tau}$ represents the vector of decision variables modeling the possible agent–task assignments. Specifically, it includes only the optional interval variables $\overline{\Delta}_{a,\tau}$ corresponding to agents able to perform the task $\tau \in \mathcal{T}$ under consideration. Equation (6.11) guarantees that each task τ is assigned to exactly one agent, selected from those able to perform it ($c_a(s_\tau, \mathcal{M}_\tau, \mathcal{F}_\tau) = 1$). This constraint is implemented using the *alternative* global constraint (alternative$(\cdot)$), which relies on a filtering algorithm that requires as input: *i*) the non-optional interval variable modeling the temporal window of the task under consideration (Δ_τ); and *ii*) the set of optional interval variables representing the possible executing agents ($[\overline{\Delta}_{a,\tau}]_{a\in\mathcal{A}'_\tau}$). Based on this input, and in compliance with the other constraints defined within the problem model, the algorithm selects an interval from the set of optional intervals to be included in the solution. The agent associated with the selected interval will be responsible for the execution of the specified task.

C5) No concurrent executions:

$$\text{noOverlap}([\overline{\Delta}_{a,\tau}]_{\tau\in\mathcal{T}'_a}), \quad \forall a \in \mathcal{A}, \tag{6.12}$$

where $\mathcal{T}'_a = \{\tau \in \mathcal{T} \mid c_a(s_\tau, \mathcal{M}_\tau, \mathcal{F}_\tau) = 1\}$ is the subset of tasks that agent a is able to perform, while $[\overline{\Delta}_{a,\tau}]_{a\in\mathcal{A}'_\tau}$ represents the vector of decision variables modeling the possible agent–task assignments. Specifically, it includes only the optional interval variables $\overline{\Delta}_{a,\tau}$ corresponding to tasks that the agent a under consideration is able to perform. Equation (6.12) enforces that each agent $a \in \mathcal{A}$ can not execute more than one task at

a time. This constraint is implemented using the *noOverlap* global constraint, which relies on a filtering algorithm that requires as input the set of optional interval variables modeling the possible task assignments for agent a under consideration. Based on this input, and in compliance with the other constraints defined within the problem model, the algorithm refines the start and end times of the interval variables included in the solution to ensure non-overlapping execution.

C6) Robot velocities:

$$\text{ifThen}(\,\text{presenceOf}(\overline{\Delta}_{r,\tau}), v_{r,\tau} \geq v_{r,\min}), \quad \forall r \in \mathcal{R} \cap \mathcal{A}_{\tau}^{'},\ \tau \in \mathcal{T}_{a}^{'}, \tag{6.13a}$$

$$\text{ifThen}(\,\text{presenceOf}(\overline{\Delta}_{r,\tau}), v_{r,\tau} \leq v_{r,\max}), \quad \forall r \in \mathcal{R} \cap \mathcal{A}_{\tau}^{'},\ \tau \in \mathcal{T}_{a}^{'}, \tag{6.13b}$$

$$\text{ifThen}(\,\text{presenceOf}(\overline{\Delta}_{r,\tau}) = \mathit{false}, v_{r,\tau} = 0), \quad \forall r \in \mathcal{R} \cap \mathcal{A}_{\tau}^{'},\ \tau \in \mathcal{T}_{a}^{'}. \tag{6.13c}$$

Equations (6.13a)-(6.13b) constrain $v_{r,\tau}$ to be greater than zero and to take values within the interval $[v_{r,\min}, v_{r,\max}]$ when robot r is responsible for executing task τ (presenceOf$(\overline{\Delta}_{r,\tau}) = \mathit{true}$). Conversely, Equation (6.13c) constrains $v_{r,\tau}$ to be equal to 0 when robot r is not responsible for executing it (presenceOf$(\overline{\Delta}_{r,\tau}) = \mathit{false}$).

C7) Minimum duration:

$$\text{ifThen}\left(\text{presenceOf}(\overline{\Delta}_{r,\tau}) \wedge X_{\tau,o}, \text{lengthOf}(\Delta_{\tau}) \geq \delta_{r,s_{\tau}} + \frac{d_{r,\tau,o}}{v_{r,\tau}}\right), \quad \forall \tau \in \mathcal{T},\ r \in \mathcal{R} \cap \mathcal{A}_{\tau}^{'},\ o \in \mathcal{O}_{\tau}^{'}, \tag{6.14a}$$

$$\text{ifThen}\left(\text{presenceOf}(\overline{\Delta}_{h,\tau}) \wedge X_{\tau,o}, \text{lengthOf}(\Delta_{\tau}) \geq \delta_{h,s_{\tau}} + \frac{d_{h,\tau,o}}{v_{h}(s_{\tau}, o)}\right), \quad \forall \tau \in \mathcal{T},\ h \in \mathcal{H} \cap \mathcal{A}_{\tau}^{'},\ o \in \mathcal{O}_{\tau}^{'}. \tag{6.14b}$$

Equations (6.14a) and (6.14b) specify the minimum duration of each task in the plan. Specifically, Equation (6.14a) establishes that if o is the object involved in the execution of task τ ($X_{\tau,o} = 1$) and r is the robotic agent responsible for executing it (presenceOf$(\overline{\Delta}_{r,\tau}) = \mathit{true}$), then the task

duration must not be less than the sum of the times needed by robot r: i) the constant term associated with the primitive action (δ_{r,s_τ}); and ii) the variable term modeling the time to complete all displacements at velocity $v_{r,\tau}$ ($d_{r,\tau,o}/v_{r,\tau}$), where $v_{r,\tau}$ denotes the velocity assigned to robot r for task τ. Equation (6.14b) enforces an analogous constraint, but addressing the scenario in which task τ is assigned to a human operator rather than a robotic agent.

6.3.3 Batch decomposition strategy

In the case of large-scale systems involving several agents, tasks, and objects, the CP-based resolution might become computationally intensive, potentially compromising the practical usability of the method. To this end, a batch decomposition strategy, extending the one in Chapter 5, is proposed to partition tasks into fixed-size batches and solve them iteratively. The batch decomposition is a commonly used approach to improve scalability in complex optimization [135, 136, 137, 138], providing a practical balance between computational efficiency and solution quality. The idea behind the proposed strategy is to order the tasks within each layer according to a selected criterion, such as distance or heuristic evaluation, and progressively incorporate them into the planning procedure.

Algorithm 6 summarizes the main steps of the decomposition strategy. Given the batch size b_s and the sets $\mathcal{T}$, $\mathcal{A}$, and $\mathcal{O}$, the first step involves sorting the tasks within each layer $\mathcal{L}_i$, with $i \in \{1, \ldots, n_L\}$, according to a specified criterion (e.g., the base duration of the corresponding primitive action) and collecting them into an ordered vector $V_\mathcal{T}$. After that, the first b_s tasks are extrated from $V_\mathcal{T}$ to generate the vector $V_{\mathcal{T}^*}$ (line 2) and initialize three sets, $\mathcal{D}^{\mathcal{T}}_{\text{fix}}$, $\mathcal{D}^{\mathcal{A}}_{\text{fix}}$, and $\mathcal{D}^{\mathcal{O}}_{\text{fix}}$, as empty (line 3). These sets will contain the decision variables whose values have been determined and are consequently constrained to remain fixed throughout the following optimization stages. At this point, the CP-based optimization algorithm is executed considering the tasks in vector $V_{\mathcal{T}^*}$, and the available agents ($\mathcal{A}$) and objects ($\mathcal{O}$) (line 4). Subsequently, an iterative procedure is executed until the vector $V_\mathcal{T}$ is empty (line 5). Briefly, in each iteration, the algorithm examines every task τ^* within the vector $V_{\mathcal{T}^*}$ (line 6), fixes the corresponding decision variables to prevent any modification in later optimizations, then selects other b_s tasks for allocation and scheduling, and

executes the optimization algorithm on them. The fixing process is realized by adding the interval variable Δ_{τ^*} to $\mathcal{D}^{\mathcal{T}}_{\text{fix}}$ (line 7), then adding the optional interval variable $\overline{\Delta}_{a,\tau^*}$ of the assigned agent a to $\mathcal{D}^{\mathcal{A}}_{\text{fix}}$ (lines 8-11), and finally, if the task τ^* involves the use of an object, adding the binary variable $X_{\tau^*,o}$ of the assigned object o to $\mathcal{D}^{\mathcal{O}}_{\text{fix}}$ (lines 12-15). The assigned object is also removed by the set of available objects $\mathcal{O}$ (line 16) to prevent it from being used multiple times. After these steps, the algorithm transfers the first b_s tasks from $V_{\mathcal{T}}$ to $V_{\mathcal{T}^*}$ (line 18). The optimization algorithm is then re-executed by considering the vector $V_{\mathcal{T}^*}$, the agent and object sets $\mathcal{A}$ and $\mathcal{O}$, respectively, and the sets comprising all decision variables already computed and thus fixed in value, $\mathcal{D}^{\mathcal{T}}_{\text{fix}}$, $\mathcal{D}^{\mathcal{A}}_{\text{fix}}$, and $\mathcal{D}^{\mathcal{O}}_{\text{fix}}$ (line 19). This mechanism enables progressive planning while preserving consistency with previously scheduled tasks and ensuring computational tractability.

Algorithm 6 Batch Decomposition Strategy

Require: b_s, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{O}$
1: $V_{\mathcal{T}} \leftarrow \text{sort}(\mathcal{T}, [\mathcal{L}_i]_{i\in\{1,\dots,n_L\}})$
2: $V_{\mathcal{T}^*} \leftarrow \text{pop}(V_{\mathcal{T}}, 1 : b_s)$
3: $\mathcal{D}^{\mathcal{T}}_{\text{fix}}$, $\mathcal{D}^{\mathcal{A}}_{\text{fix}}$, $\mathcal{D}^{\mathcal{O}}_{\text{fix}} \leftarrow \emptyset$
4: Δ_{τ^*}, $\overline{\Delta}_{a,\tau^*}$, $X_{\tau^*,o}$ $(\forall \tau^* \in L_{\mathcal{T}^*}, a \in \mathcal{A}, o \in \mathcal{O}) \leftarrow$ optimization algorithm$(V_{\mathcal{T}^*}, \mathcal{A}, \mathcal{O})$
5: **while** $V_{\mathcal{T}} \neq \emptyset$ **do**
6: **for all** $\tau^* \in V_{\mathcal{T}^*}$ **do**
7: $\mathcal{D}^{\mathcal{T}}_{\text{fix}} \leftarrow \Delta_{\tau^*}$
8: **for all** $a \in \mathcal{A}$ **do**
9: **if** presenceOf$(\overline{\Delta}_{a,\tau^*}) = true$ **then**
10: $\mathcal{D}^{\mathcal{A}}_{\text{fix}} \leftarrow \overline{\Delta}_{a,\tau^*}$
11: **break**
12: **if** $\mathcal{F}_{\tau^*} \neq \emptyset$ **then**
13: **for all** $o \in \mathcal{O}$ **do**
14: **if** $X_{\tau^*,o} = 1$ **then**
15: $\mathcal{D}^{\mathcal{O}}_{\text{fix}} \leftarrow X_{\tau^*,o}$
16: $\mathcal{O} \leftarrow$ remove(o)
17: **break**
18: $V_{\mathcal{T}^*} \leftarrow \text{pop}(L_{\mathcal{T}}, 1 : b_s)$
19: Δ_{τ^*}, $\overline{\Delta}_{a,\tau^*}$, $X_{\tau^*,o}$ $(\forall \tau^* \in L_{\mathcal{T}^*}, a \in \mathcal{A}, o \in \mathcal{O}) \leftarrow$ optimization algorithm$(V_{\mathcal{T}^*}, \mathcal{A}, \mathcal{O}, \mathcal{D}^{\mathcal{T}}_{\text{fix}}, \mathcal{D}^{\mathcal{A}}_{\text{fix}}, \mathcal{D}^{\mathcal{O}}_{\text{fix}})$
20: **return** Δ_{τ}, $\overline{\Delta}_{a,\tau}$, $X_{\tau,o}$ $(\forall \tau \in \mathcal{T}, a \in \mathcal{A}, o \in \mathcal{O})$

6.4 Simulation validation

This section presents the simulation results, which validate the proposed architecture. As illustrative case studies, object sorting and letter assembly missions involving a heterogeneous team of agents were considered for validation purposes.

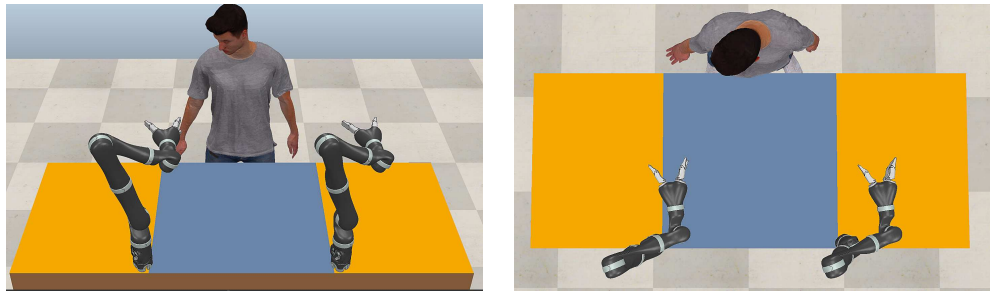


Figure 6.3: Simulation scenario overview.

6.4.1 Setup description

The considered simulation scenario, depicted in Figure 6.3, comprises two fixed-base collaborative manipulators and a human agent operating on a workbench. The latter is divided into two lateral regions and a central region, highlighted in orange and blue, respectively. The lateral regions represent the picking areas, containing the objects that can be assigned to tasks. The central region, on the other hand, serves as the working area, and its content is strictly determined by the specific case study under consideration. For example, it may contain an n-by-n grid, as in the letter assembly case study (see Figure 6.4-left), or a set of boxes, as in the object sorting case study (see Figure 6.5-left). Without loss of generality, each agent was assumed to be capable of reaching any location on the workbench. The presence of 64 colored blocks on the workbench was assumed: 40 cubes (C) and 24 parallelepipeds (P). Specifically, in each picking area, it was assumed that there were 5 cubes and 3 parallelepipeds for each of the following colors: red (R), green (G), yellow (Y), and blue (B). These objects formed the set $\mathcal{O}$ and followed the specifications provided in Section 6.2. Specifically, each object $o \in \mathcal{O}$: i) was assigned to a specific picking area and located at a random position $\boldsymbol{p}_o \in \mathbb{R}^3$ within

Color	Shape	Mass [Kg]	Size
R	C	0.12	1
	P	0.36	3
Y	C	0.09	1
	P	0.27	3
B	C	0.06	1
	P	0.18	3
G	C	0.03	1
	P	0.09	3

Table 6.2: Specifics of the objects.

that area; *ii*) had a mass m_o, chosen according to its shape and color (see Table 6.2); and *iii*) was characterized by a set of distinctive features $\mathcal{F}_o$, including shape, color, and size, where the size denoted the number of grid cells occupied by the object on an n-by-n grid whose cells were each of dimensions $0.1\,m \times 0.1\,m$.

Regarding the agents, each robotic agent $r \in \mathcal{R}$: *i*) occupied a position $\boldsymbol{p}_r \in \mathbb{R}^3$ close to the working area, as shown in Figure 6.3; *ii*) was able to perform the pick-and-place primitive action, i.e., $\mathcal{S}_r = \{\text{PaP}\}$; and *iii*) was characterized by a minimum and a maximum velocity set to $v_{r,\min} = 0.1\ m/s$ and $v_{r,\max} = 0.25\ m/s$, respectively. Conversely, each human agent $h \in \mathcal{H}$: *i*) was located at a position $\boldsymbol{p}_h \in \mathbb{R}^3$ close to the working area, as shown in Figure 6.3; *ii*) was able to perform the pick-and-place, box opening, and box closing primitive actions, i.e., $\mathcal{S}_h = \{\text{PaP}, \text{opening}, \text{closing}\}$; and *iii*) was characterized by a maximum operating speed $v_{h,\max}$ set to $0.3\ m/s$. Concerning the actual operating speed of the human operator h, performing the primitive action s with the object o, it was computed as follows

$$v_h(s, o) = \kappa_s \frac{v_{h,\max}}{1 + \alpha(m_o - m_{\min})}, \tag{6.15}$$

where $\kappa_s \in [0, 1]$ is a scaling factor depending on the primitive action, $\alpha \geq 0$ is a gain, and $m_{\min}$ represents the minimum mass among all the objects, i.e., $m_{\min} = \min_{o \in \mathcal{O}} m_o$. According to this formulation, the operating speed: *i*) decreases as the mass of the object involved in the task increases; and *ii*) depends not only on the mass of the object, but also on the specific primitive action performed by the agent.

As for the workload sustained by human agent h when executing task τ with object o involved, it was computed as follows

$$\omega_{h,\tau,o} = \text{lengthOf}(\Delta_\tau) m_o = \left(\delta_{h,s_\tau} + \frac{d_{h,\tau,o}}{v_h(s_\tau, o)} \right) m_o, \tag{6.16}$$

meaning that the workload increases with both the object mass and the duration of the motion. It is worth pointing out that the modeling of human operating speed and workload was beyond the scope of the activity presented in this chapter, and that the adopted models were intended

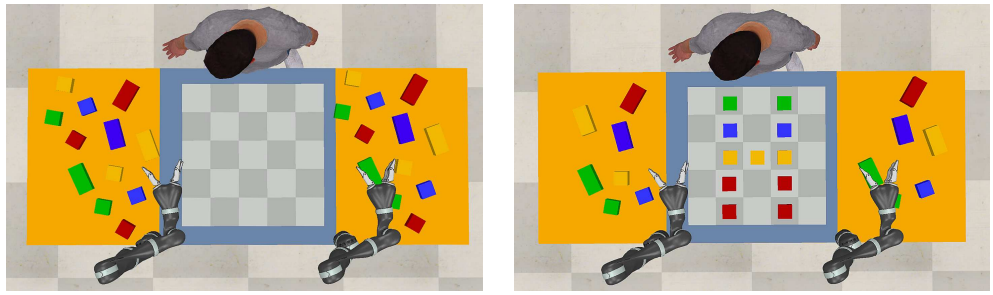


Figure 6.4: Overview of the simulation scenario at the beginning (left) and the end (right) of a letter assembly scenario, in which the objective is to arrange the objects in the scene to form a letter on an n-by-n grid. As an example, the capital letter "H" is shown on the right.

solely to demonstrate the generality and flexibility of the proposed approach.

6.4.2 Missions' description

As previously stated, the validation was carried out through the analysis of two case studies: *i*) letter assembly and *ii*) object sorting.

In the letter assembly case study, illustrated in Figure 6.4, the objective was to position objects in a 5-by-5 grid placed at the center of the working area to reproduce the structure of the desired letter as accurately as possible while using as few objects as possible. The assembly was required to proceed from the 5-th row toward the 1-st, with the 5-th row situated next to the human and the 1-st row located next to the robots. Without loss of generality, the capital letters "D" and "H" were considered as reference structures.

In the object sorting case study, on the other hand, which is depicted in Figure 6.5, the objective was to transfer objects into four boxes located within the working area. Each box was initially empty and had to be filled with one object of each color and at least one parallelepiped.

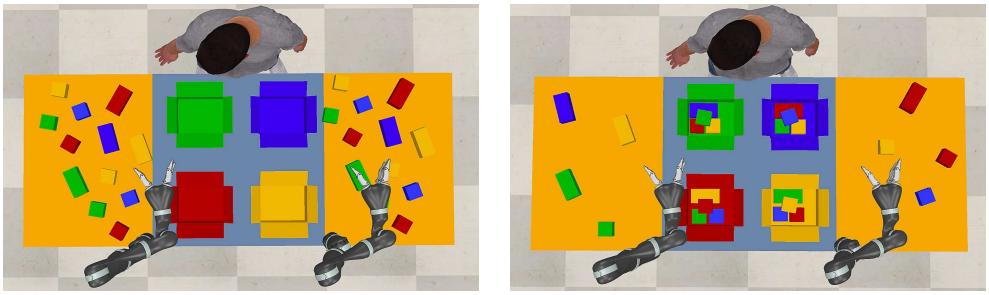


Figure 6.5: Overview of the simulation scenario at the beginning (left) and the end (right) of an object-sorting scenario, in which the goal is to organize some objects into four sealed boxes of distinct colors (red, yellow, blue, and green).

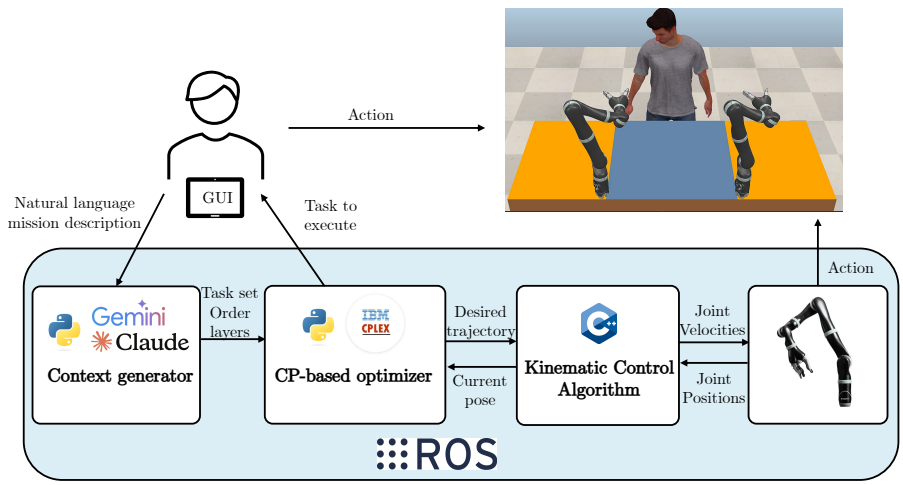


Figure 6.6: Overview of the ROS-based implementation.

6.4.3 Implementation details

The proposed framework relies on the ROS (Robot Operating System) Noetic Ninjemys middleware. As depicted in Figure 6.6, the architecture is composed of a GUI module and three main nodes: *i*) a context generator node, implemented in Python and capable of interfacing with either a local or a remote LLM, *ii*) a CP-based optimizer node, implemented in Python and responsible for formulating and solving the CP optimization problem described in Section 6.3.2.1, and *iii*) a robot control node, implemented in C++ and responsible for generating the velocity commands for

the manipulators. In more detail, the context generator node is responsible for sending the textual prompts to the selected LLM and handling its responses. The CP-based optimizer node is responsible for implementing the CP optimization problem described in Section 6.3.2.1, using the context information provided by the previous node, and determining the optimal task allocation and scheduling. To this aim, it resorts to version 22.1.1 of IBM ILOG CPLEX, a state-of-the-art optimization solver widely recognized for its efficiency in handling constrained optimization problems [171, 172]. Finally, the robot control node is responsible for generating the joint velocity commands for manipulators, whereas the GUI module is responsible for displaying the planned tasks to human operators.

Concerning the CP optimization problem, the weighting coefficients of the cost function in Eq. (6.7) were set to $\beta = 4$, $\gamma = 1$, and $\eta = 10$, whereas the batch size b_s was set to 4.

The described architecture was entirely developed on a commercial laptop equipped with an Intel Core $i7 - 12700H$ CPU (up to 4.90 GHz), 32 GB DDR5 RAM, and a 2 TB SSD.

6.4.4 Baselines and evaluation metrics

Each case study was evaluated using three approaches: the proposed one and two baselines.

- In the proposed approach, referred to as LLM_{CG}+$\text{CP}_{\text{A,S}}$, the LLM-based module acts exclusively as a context generator (CG), identifying the set of tasks ($\mathcal{T}$) required to complete the assigned mission and their hierarchical relationships ($\mathcal{L}_i$, $i \in \{1, ..., n_L\}$). The CP-based module, on the other hand, handles: i) the allocation (A) procedures related to the identified tasks, which involves defining the value of the binary variables $X_{\tau,o}$ and the optional interval variables $\overline{\Delta}_{a,\tau}$; and ii) the scheduling (S) procedures, which, in contrast, involve defining the value of the interval variables Δ_τ and the continuous variables $v_{r,\tau}$.
- In the first baseline, referred to as $\text{LLM}_{\text{CG,A}}$+CP_{S}, the LLM-based module is responsible not only for context generation but also for allocation procedures. Specifically, in addition to identifying the set

of tasks ($\mathcal{T}$) required to complete the assigned mission and their hierarchical relationships ($\mathcal{L}_i,\ i \in \{1, ..., n_L\}$), it also defines the value of the binary variables $X_{\tau,o}$ and the optional interval variables $\overline{\Delta}_{a,\tau}$. As for the CP-based module, in this case, it is responsible only for the scheduling procedures, namely defining the value of the interval variables Δ_τ and the continuous variables $v_{r,\tau}$.

- In the second baseline, referred to as $\text{LLM}_{\text{CG,A,S}}$, an approach similar to that proposed in [173] is adopted. Specifically, in this case, the LLM-based module handles the entire pipeline for producing the final plan, with the CP-based module excluded from all stages. In more detail, in this case, the LLM-based module performs the following operations: *i*) generates the context, identifying the set of tasks ($\mathcal{T}$) required to complete the assigned mission and their hierarchical relationships ($\mathcal{L}_i,\ i \in \{1, ..., n_L\}$); *ii*) handles the allocation procedures, which involve defining the value of the binary variables $X_{\tau,o}$ and the optional interval variables $\overline{\Delta}_{a,\tau}$; and *iii*) manages the scheduling procedures, which, on the other hand, involve defining the value of the interval variables Δ_τ and the continuous variable $v_{r,\tau}$.

As stated in Section 6.3.1, when decomposing the mission described in the user prompt into a set of tasks ($\mathcal{T}$), for each task τ, the LLM also provides a set of motion instructions ($\mathcal{M}_\tau$), in addition to the corresponding primitive action (s_τ). In the considered setting, this set $\mathcal{M}_\tau$ comprises only the relevant positions. Specifically, for the proposed approach (LLM_{CG}+$\text{CP}_{\text{A,S}}$), it contains the final position of the operation, while for the two baselines it contains two positions, representing the final position of the operation and the initial position of the object to use, respectively.

For each method, two variants were considered, differing in the LLM model used as the backbone. The models employed are Gemini 2.5 Pro and Claude 4.5 Sonnet.

Concerning the prompts for the LLM-based module, they were structured as shown in Figure 6.2 for the proposed method, while some changes were introduced for the two baselines. In both cases, these changes were intended to provide additional information to the LLM-based module to correctly carry out the allocation and scheduling procedures. In detail, in

the system prompt, the positions of each object, grid cell, and box in the scene were introduced, and the "action" section was replaced with a more detailed "agents" section. The latter specified a complete description of each agent in the scene, including its ID, spatial position, list of primitive actions it could perform, and velocity limits. Moreover, additional directives were added. Each directive corresponded to the natural language translation of a constraint from the CP formulation presented in Section 6.3.2.1. The number and type of directives introduced for each approach depended strictly on the procedures that the LLM-based module had to handle. In detail, for baseline $\text{LLM}_{\text{CG,A}}$+CP_{S}, the natural language translation of all constraints related to the allocation procedures (e.g., Eq. (6.12)) was introduced, whereas for baseline $\text{LLM}_{\text{CG,A,S}}$, the natural language translation of all constraints related to the scheduling procedures (e.g., Eq. (6.8)) was also introduced. A possible natural language translation of the constraint in Eq. (6.12) is "An agent cannot carry out multiple tasks simultaneously", while for the constraint in Eq. (6.8) it is "The start and end times of each task must be determined according to its position within the task hierarchy. Specifically, any task at level i cannot commence before the completion of all tasks at lower levels". Finally, as for the template specifying the desired structure for the final plan, it was extended in both cases. In particular, it was introduced: *i*) an "objectID" field, in which the LLM was required to specify the ID of the object associated with the task; *ii*) an "agentID" field, in which the LLM was required to report the ID of the agent responsible for task execution; and *iii*) "startTime" and "endTime" fields, in which the LLM was required to specify the start and end times of the task. The latter two fields were included only for baseline $\text{LLM}_{\text{CG,A,S}}$. Regarding the user prompt, for both baselines, it was extended to include, in addition to the mission description, details about the additional procedures to be handled by the LLM and the cost function to be optimized. Specifically, for the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} case, the following instruction was introduced: "For every action, assign an available object ("objectID") that matches the required shape and colour, and an agent ("agentID") such that human workload, robot energy use, and total makespan are all minimized". Whereas, for the baseline $\text{LLM}_{\text{CG,A,S}}$, the following instruction was introduced: "While preserving the original plan, allocate a start and end time to each action according to the directives, minimizing the makespan, human workload, and robotic energy consumption". Some illustrative examples of prompts

are available at the link.[2].

Due to the stochastic variability inherent in the LLMs' outputs [174, 175], multiple tests were conducted to assess the performance of each method. Specifically, for each case study, 3 sets of objects were considered, differing in their initial positions on the workbench. For each set, 10 tests were executed for each approach, based on which statistics were computed for the performance metrics detailed below.

To evaluate the performance of each method, different metrics were introduced. Let $J^b_{i,k}$ and $\overline{J}_{i,k}$ denote the values of the objective function in Eq. (6.7) obtained from the generic baseline under consideration and the LLM_{CG}+$\text{CP}_{\text{A,S}}$ method, respectively, at the end of the k-th trial on the i-th object set. The corresponding relative error $\epsilon^b_{i,k}$ was defined as

$$\epsilon_{i,k} = \frac{J^b_{i,k} - \bar{J}_{i,k}}{\bar{J}_{i,k}}, \tag{6.17}$$

which is positive when the proposed method outperforms the baseline in terms of optimality and negative otherwise. Note that the higher the error, the better the performance of the LLM_{CG}+$\text{CP}_{\text{A,S}}$ method compared to the baseline. Based on this quantity, the minimum and maximum relative errors (denoted as ϵ_m and ϵ_M, respectively) were computed over the 10 trials for each baseline and object set. Moreover, a *feasibility* metric was introduced to evaluate whether a plan satisfies all the directives listed in the system prompt and meets all the constraints defined in the constraint programming formulation (see Section 6.3.2.1). Based on this, the feasibility percentage ($f\%$), i.e., the percentage of feasible plans on the 10 trials, was evaluated for each object set and method. For each object set and method, the mean values of the following parameters were also derived from the outcomes of the 10 trials: *i*) the objective function (J); *ii*) the overall makespan (μ); *iii*) the cumulative energy consumption of robotic agents (denoted as e); *iv*) the overall workload of human agents (denoted as ω); and v) the cumulative computation time of the LLM-based and the CP-based modules.

[2] `https://git.new/ZR6l5jr`

6.4.5 Performance evaluation

This section first presents an illustrative example of the proposed method (LLM_{CG}+$\text{CP}_{\text{A,S}}$) for assembling the capital letter "H". Then, the results of a general performance comparison between the proposed approach and the two baselines, for both the assembly and sorting missions, are presented.

Regarding the example, it began with the context generation by the LLM-based module, which, as previously stated, is responsible for identifying the tasks necessary to accomplish the mission and their hierarchical relationships. The textual prompts used during this stage are the same as those shown in Figure 6.2. A graphical representation of the generated output is shown in Figure 6.7.

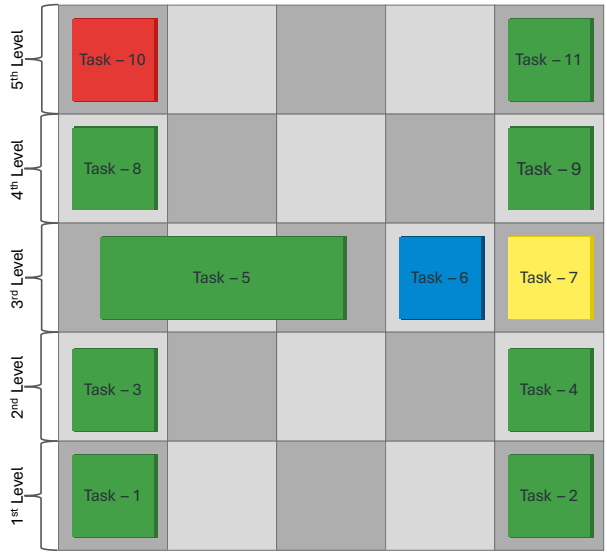


Figure 6.7: Simulative validation. Graphical representation of the output generated by the LLM-based module for the example assembly case of the capital letter "H".

The LLM decomposed the mission into 11 tasks and, to fulfill the row-ordering directive specified in Section 6.4.2, organized them into a 5-level hierarchy. Each of the identified tasks involved handling an object and executing a pick-and-place action. The involved object set comprised 10 cubes and 1 parallelepiped. The context generation phase was followed by the optimization phase, during which the CP-based module formulated

and solved a constrained optimization problem to allocate and schedule tasks. A graphical representation of the resulting output is provided in Figure 6.8.

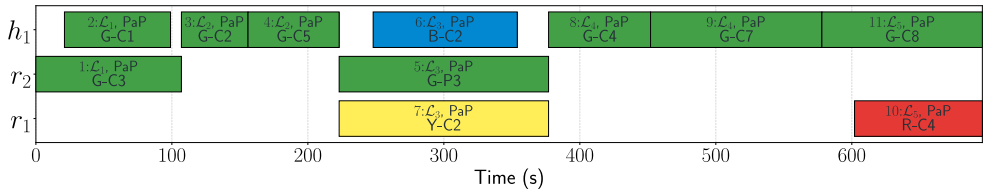


Figure 6.8: Simulative validation. Example of an optimal plan for assembling a capital letter "H", generated from the output of the LLM-based module shown in Figure 6.7.

The figure contains a set of colored bars, each associated with an agent and distinguished by a color. Each bar represents a specific task and contains details of the corresponding task, including the task ID, the corresponding hierarchical level, the primitive action to be performed by the agent, and, when necessary, the specific object to be used. For example, consider the first bar associated with the human agent h_1, which represents the first task assigned to him/her. The task τ associated with this bar: *i*) corresponds to the second element of the set $\mathcal{T}$; *ii*) is located at the first level ($\mathcal{L}_1$) of the task hierarchy; and *iii*) involves executing a PaP action with the green cube labeled "green-cube1". As can be observed, the human agent was primarily responsible for the PaP operations involving the lightest objects in the scene, whereas the robots were responsible for those involving the heaviest ones. This choice was the result of several factors, such as: *i*) the relative distance of the objects from the agents; *ii*) the relative distance of the objects from their designated final locations; *iii*) the mass of the objects; and *iv*) the values assigned to the weighting coefficients of the cost function components. Regarding the weights, as stated at the end of Section 6.4.3, the value assigned to the workload weighting coefficient ($\beta = 4$) is higher than that assigned to the energy consumption weighting coefficient ($\gamma = 1$), thus encouraging the lightest objects to be assigned to the human agent rather than to the robots.

The performance results obtained from the different methods over all missions are now presented and analyzed. These values are presented in Tables 6.3–6.8. More in detail, Tables 6.3–6.5 report the results obtained

using Gemini 2.5 Pro, whereas Tables 6.6–6.8 report the outcomes obtained with Claude 4.5 Sonnet. In all tables, each macro row is related to a set of objects, whose "ID" is specified in the first column, and presents, for each approach, the evaluation metrics resulting from the outcomes of the 10 trials conducted with that set.

First, an analysis of Table 6.3 is provided, which collects the outcomes obtained with Gemini 2.5 Pro for the assembly of the capital letter "H". The table overall confirm the effectiveness of the proposed approach and its superiority over the two baselines for the considered mission. For example, consider the column associated with the feasibility rate ($f\%$) of generated plans. It can be observed that both the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} and the proposed approach (LLM_{CG}+$\text{CP}_{\text{A,S}}$) achieved a feasibility rate of 100% across all object sets. Conversely, the baseline $\text{LLM}_{\text{CG,A,S}}$, which relies exclusively on the LLM-based module, achieved lower feasibility rate values across all object sets, ranging from 60% to 40%. This indicates that the baseline $\text{LLM}_{\text{CG,A,S}}$ is not reliable in terms of feasibility, highlighting the need for a hybrid planning approach. Regarding the computation time ($\text{CP} + \text{LLM (s)}$), it can be observed that the proposed approach required less time than the other two to produce a solution. Specifically, the proposed approach required $\approx 24s$ in total, the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} required $\approx 36s$, whereas the baseline $\text{LLM}_{\text{CG,A,S}}$ required $\approx 46s$). Concerning the minimum and maximum relative errors (ϵ_m, ϵ_M), it can be noticed that, for both baseline approaches, the minimum and maximum relative errors took positive values across all the object sets. Specifically, the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} exhibited a minimum (maximum) relative error ranging from 0.23 to 0.44 (0.95 to 1.70), while the baseline $\text{LLM}_{\text{CG,A,S}}$ exhibited a minimum (maximum) relative error ranging from 0.32 to 0.49 (0.91 to 2.26). Two observations can be derived from this: *i*) both baselines underperform compared to the proposed approach across all object sets; and *ii*) the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} outperforms the baseline $\text{LLM}_{\text{CG,A,S}}$. This is motivated by the fact that the first baseline resorts to an optimization procedure for scheduling, enabling it to achieve more optimal results than the LLM-only baseline and further confirming the effectiveness of hybrid approaches. The previous observations are further confirmed by the results in the column associated with the average cost function (J), where it can be noticed that the proposed approach yielded plans with lower cost function values compared to those produced by the

two baselines. This behavior can be primarily attributed to the fact that both baselines yielded, on average, plans with greater makespan and energy consumption values compared to those produced by the proposed method. For example, in the case of the first set of objects, the average normalized makespan (energy consumption) was 0.11 (0.50), with the proposed approach, 0.15 (2.28) with the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S}, and 0.22 (1.83) with the baseline $\text{LLM}_{\text{CG,A,S}}$. This shows that the plans generated by the two baseline approaches have a higher cumulative duration and involve a greater energy consumption than those generated by the proposed approach. This behavior, which holds for all object sets, can be primarily attributed to a suboptimal management of task allocation and velocity generation processes by the two baselines. As for the workload (ω), it can be observed that the two baseline approaches yielded solutions with similar normalized workload values, which are lower than those associated with solutions produced by the proposed approach. This behavior suggests that the LLM tends to prioritize optimizing the workload of human agents over other parameters.

The observations made throughout the analysis of Table 6.3 are further supported by Table 6.4, which reports the evaluation metrics obtained from simulations focused on the assembly of a capital letter "D". Similar observations can also be derived from Table 6.5, which provides the results of the object sorting mission. Specifically, it can be observed that a larger performance gap exists between the proposed approach and the two baselines for this case study, with the proposed approach outperforming them. This trend is particularly evident for baseline $\text{LLM}_{\text{CG,A,S}}$. Indeed, analyzing the feasibility rate ($f\%$) column, it can be noticed that, even in this case, the baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} and the proposed approach continued to perform equivalently in terms of feasibility, generating feasible plans in all cases, whereas the baseline $\text{LLM}_{\text{CG,A,S}}$ performed worse than the previous cases. In particular, the average feasibility rate of its plans decreased significantly, reaching a range from 50% to 10%. This suggests that a pure LLM approach may struggle to optimally handle all the phases of the process described in Section 6.3, particularly in the case of more complex missions. As for baseline $\text{LLM}_{\text{CG,A}}$+CP_{S}, it can be observed that its performance remained quite stable across the considered case studies. This suggests that, unlike baseline $\text{LLM}_{\text{CG,A,S}}$, the performance of baseline $\text{LLM}_{\text{CG,A}}$+CP_{S} does not appear to degrade significantly as the mission

complexity increases. However, as can be noticed, it continued to be outperformed by the proposed approach.

Regarding the results obtained with Claude 4.5 Sonnet on the three case studies, summarized in Tables 6.7–6.8, observations analogous to those made on solutions generated by Gemini 2.5 Pro can be derived. However, comparing the metrics obtained from solutions generated by Gemini 2.5 Pro with those from solutions produced by Claude 4.5 Sonnet, it can be observed that, in the letter assembly case study, Claude outperformed Gemini on average, producing more feasible solutions and lower relative error values. Conversely, in the object sorting case study, Gemini performed better than Claude. This is particularly evident in the baseline $\text{LLM}_{\text{CG,A,S}}$ case, where no feasible solutions were generated across any of the considered object sets.

ID	Case	$f(\%)$	CP + LLM (s)	ϵ_m	ϵ_M	J	μ	ω	e
1	$\text{LLM}_{\text{CG}}+\text{CP}_{\text{A,S}}$	100	$11.97 + 11.37$		-	2.80	0.11	0.28	0.60
	$\text{LLM}_{\text{CG,A}}+\text{CP}_{\text{S}}$	100	$1.52 + 33.07$	0.27	0.95	4.14	0.15	0.08	2.28
	$\text{LLM}_{\text{CG,A,S}}$	60	$0.00 + 46.05$	0.32	0.91	4.37	0.22	0.08	1.83
2	$\text{LLM}_{\text{CG}}+\text{CP}_{\text{A,S}}$	100	$11.74 + 12.50$		-	2.80	0.10	0.26	0.73
	$\text{LLM}_{\text{CG,A}}+\text{CP}_{\text{S}}$	100	$2.97 + 33.29$	0.23	1.51	5.72	0.20	0.14	3.18
	$\text{LLM}_{\text{CG,A,S}}$	60	$0.00 + 49.52$	0.45	2.26	6.06	0.28	0.14	2.67
3	$\text{LLM}_{\text{CG}}+\text{CP}_{\text{A,S}}$	100	$11.46 + 12.15$		-	2.59	0.10	0.24	0.65
	$\text{LLM}_{\text{CG,A}}+\text{CP}_{\text{S}}$	100	$2.16 + 34.19$	0.44	1.70	4.96	0.18	0.12	2.64
	$\text{LLM}_{\text{CG,A,S}}$	40	$0.00 + 43.52$	0.49	1.49	5.06	0.28	0.12	1.78

Table 6.3: Comparison of the proposed method against the two baselines ($\text{LLM}_{\text{CG,A}}+\text{CP}_{\text{S}}$,$\text{LLM}_{\text{CG,A,S}}$) in the simulations for assembling the capital letter "H", employing Gemini 2.5 Pro.

ID	Case	$f(\%)$	CP + LLM (s)	ϵ_m	ϵ_M	J	μ	ω	e
1	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$11.15 + 13.64$		-	2.24	0.10	0.18	0.54
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$2.02 + 33.21$	0.14	1.88	3.96	0.15	0.23	1.56
	$\text{LLM}_{\text{CG,A,S}}$	40	$0.00 + 43.52$	0.24	2.09	4.28	0.25	0.23	0.86
2	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$11.00 + 12.02$		-	2.18	0.10	0.16	0.61
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$2.01 + 29.45$	0.11	1.19	3.35	0.14	0.09	1.57
	$\text{LLM}_{\text{CG,A,S}}$	70	$0.00 + 42.44$	0.22	1.45	3.69	0.22	0.09	1.09
3	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$10.62 + 14.88$		-	2.20	0.09	0.21	0.43
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$1.98 + 35.17$	0.12	1.40	3.58	0.15	0.13	1.52
	$\text{LLM}_{\text{CG,A,S}}$	50	$0.00 + 44.22$	0.23	1.65	3.87	0.23	0.13	1.05

Table 6.4: Comparison of the proposed method against the two baselines ($\text{LLM}_{\text{CG,A}}$+CP_{S},$\text{LLM}_{\text{CG,A,S}}$) in the simulations for assembling the capital letter "D", employing Gemini 2.5 Pro.

ID	Case	$f(\%)$	CP + LLM (s)	ϵ_m	ϵ_M	J	μ	ω	e
1	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$24.32 + 17.22$		-	3.14	0.08	0.23	1.41
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$15.85 + 42.63$	0.23	0.97	4.53	0.17	0.04	2.70
	$\text{LLM}_{\text{CG,A,S}}$	20	$0.00 + 60.05$	0.20	1.04	5.11	0.14	0.04	3.59
2	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$24.63 + 14.16$		-	3.02	0.08	0.21	1.34
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$16.32 + 42.28$	0.19	0.55	3.86	0.15	0.03	2.24
	$\text{LLM}_{\text{CG,A,S}}$	10	$0.00 + 77.20$	0.20	0.20	3.63	0.18	0.03	1.73
3	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$23.42 + 15.37$		-	3.09	0.08	0.24	1.29
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$17.62 + 38.17$	0.25	1.00	4.37	0.11	0.38	1.78
	$\text{LLM}_{\text{CG,A,S}}$	50	$0.00 + 68.50$	0.09	1.17	4.51	0.10	0.38	1.98

Table 6.5: Comparison of the proposed method against the two baselines ($\text{LLM}_{\text{CG,A}}$+CP_{S},$\text{LLM}_{\text{CG,A,S}}$) in the simulations for the object-sorting, employing Gemini 2.5 Pro.

ID	Case	$f(\%)$	CP + LLM (s)	ϵ_m	ϵ_M	J	μ	ω	e
1	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$12.76 + 16.66$	-		2.80	0.11	0.29	0.58
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$1.55 + 24.55$	0.36	0.90	4.63	0.18	0.10	2.44
	$\text{LLM}_{\text{CG,A,S}}$	80	$0.00 + 50.49$	0.84	1.31	5.79	0.13	0.10	4.12
2	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$12.14 + 15.53$	-		2.80	0.11	0.28	0.62
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$2.32 + 23.77$	0.26	0.93	4.78	0.17	0.18	2.30
	$\text{LLM}_{\text{CG,A,S}}$	90	$0.00 + 50.72$	0.44	1.39	5.82	0.12	0.18	3.84
3	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$11.72 + 17.08$	-		2.61	0.10	0.24	0.66
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$2.16 + 25.20$	0.25	1.06	4.32	0.17	0.09	2.30
	$\text{LLM}_{\text{CG,A,S}}$	80	$0.00 + 48.58$	0.61	1.61	5.37	0.12	0.09	3.86

Table 6.6: Comparison of the proposed method against the two baselines ($\text{LLM}_{\text{CG,A}}$+CP_{S},$\text{LLM}_{\text{CG,A,S}}$) in the simulations for assembling the capital letter "H", employing Claude 4.5 Sonnet.

ID	Case	$f(\%)$	CP + LLM (s)	ϵ_m	ϵ_M	J	μ	ω	e
1	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$10.96 + 13.08$	-		2.14	0.09	0.17	0.59
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$0.92 + 31.81$	0.11	0.25	2.51	0.11	0.06	1.18
	$\text{LLM}_{\text{CG,A,S}}$	100	$0.00 + 39.93$	0.18	0.41	2.76	0.10	0.06	1.54
2	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$11.72 + 15.77$	-		2.15	0.09	0.15	0.68
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$1.10 + 32.10$	0.05	0.44	2.65	0.11	0.09	1.19
	$\text{LLM}_{\text{CG,A,S}}$	90	$0.00 + 38.91$	0.14	0.70	2.91	0.09	0.09	1.64
3	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$12.13 + 15.14$	-		2.09	0.10	0.15	0.55
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$0.97 + 27.09$	0.00	1.01	2.57	0.11	0.09	1.16
	$\text{LLM}_{\text{CG,A,S}}$	80	$0.00 + 38.19$	0.16	0.65	2.79	0.09	0.09	1.53

Table 6.7: Comparison of the proposed method against the two baselines ($\text{LLM}_{\text{CG,A}}$+CP_{S},$\text{LLM}_{\text{CG,A,S}}$) in the simulations for assembling the capital letter "D", employing Claude 4.5 Sonnet.

ID	Case	$f(\%)$	CP + LLM (s)	ϵ_m	ϵ_M	J	μ	ω	e
1	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$46.89 + 18.84$	-		3.05	0.09	0.19	1.38
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$53.05 + 27.15$	0.21	1.96	6.09	0.14	0.65	2.13
	$\text{LLM}_{\text{CG,A,S}}$	0	-	-		-	-	-	-
2	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$44.86 + 17.82$	-		3.29	0.10	0.21	1.48
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$51.72 + 28.05$	0.16	0.41	4.15	0.13	0.19	2.07
	$\text{LLM}_{\text{CG,A,S}}$	0	-	-		-	-	-	-
3	LLM_{CG}+$\text{CP}_{\text{A,S}}$	100	$36.05 + 16.26$	-		3.21	0.10	0.22	1.35
	$\text{LLM}_{\text{CG,A}}$+CP_{S}	100	$52.50 + 27.54$	0.35	0.84	4.98	0.16	0.25	2.42
	$\text{LLM}_{\text{CG,A,S}}$	0	-	-		-	-	-	-

Table 6.8: Comparison of the proposed method against the two baselines ($\text{LLM}_{\text{CG,A}}$+CP_{S},$\text{LLM}_{\text{CG,A,S}}$) in the simulations for the object-sorting, employing Claude 4.5 Sonnet.

6.5 Laboratory experiment

This section presents the real-world validation of the proposed approach. The validation was carried out in a laboratory environment through an assembly mission with the capital letter "L" as the reference structure. A high-resolution video of the experiment is available at the link[3].

The experiment involved 2 agents, one human operator and one 7-DoF Kinova Jaco[2] manipulator, as depicted in Figure 6.9. Human

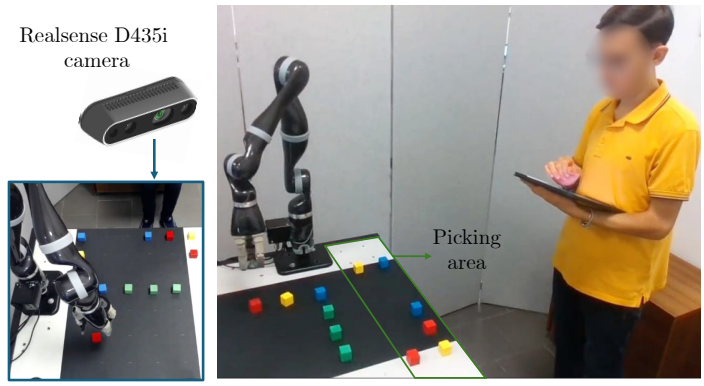


Figure 6.9: Real-world validation setup.

and robotic agents were characterized by the same set of skills as in the simulation case, while the minimum and maximum velocities were defined as $v_{h,\max} = 0.08\ m/s$ for humans, and as $v_{r,\min} = 0.05\ m/s$ and $v_{r,\max} = 0.06\ m/s$ for robots. The considered scenario comprised one picking area and a set of 12 objects, with three cubic-shaped objects for each of the following colors: red (R), yellow (Y), blue (B), and green (G). Mass and size of the objects were as in Table 6.2. The description of these objects, specifically their positions and features, was obtained using a dedicated perception module composed of a sensor camera and a detection-recognition model. Specifically, the images of the scene were acquired through an Intel RealSense D435i camera, whereas the identification and characterization of the objects were performed through a YOLO [**?**] network trained to detect and recognize colored building blocks. The same ROS-based architecture used in the simulation case study was employed in the real-world setting, with the addition of two nodes: one

[3] `https://youtu.be/56z4UdbLYe0`

handling the camera and the other handling the YOLO identification network. Additionally, Gemini 2.5 Pro was used for the LLM-based module, and the weighting coefficients of the cost function (6.7) were chosen as in the simulation case, i.e., $\beta = 4$, $\gamma = 1$, and $\eta = 10$.

The experiment for assembling the capital letter "L" began with the context generation using the LLM model. The textual prompts used in this step were similar to those shown in Figure 6.2, with some adjustments to reflect the scenario and mission considered in the experiment. Specifically, in the system prompt, the content of the "Scene" section was modified to provide a detailed description of the real-world environment, and no directive about the row filling order was included. The user prompt remained almost the same, with the only difference being the request for assembling a capital letter "L" instead of a capital letter "H". A graphical representation of the output of the LLM-based module is shown in Figure 6.10. The LLM decomposed the mission into 6 tasks and organized them at

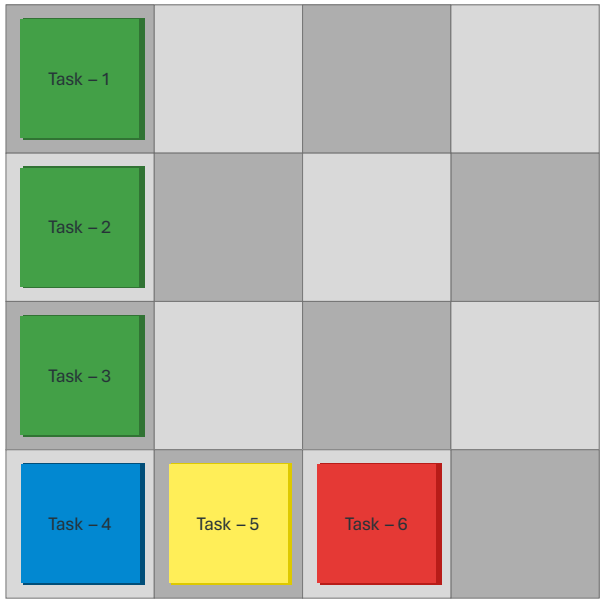


Figure 6.10: Graphical representation of the output generated by the LLM-based module during the real-world experiment for the assembly of the capital letter "L".

the same level of the task hierarchy, meaning that all could be executed in parallel. Each of the identified tasks involved handling a colored cube

and executing a pick-and-place action to compose the desired letter. This information was then exploited by the CP-based module to allocate and schedule the identified tasks while minimizing the cost function (6.7). A graphical representation of the resulting plan is provided in Figure 6.11. The figure shows the tasks assigned to each agent (with the human at the top and the robot at the bottom) and their timing. For each task, the ID, the corresponding hierarchical level, the primitive action to be performed by the agent, and the allocated object are shown. It can be observed that, as in the assembly case of the capital letter "H" presented in Section 6.4.5, the human agent was, in this case as well, primarily responsible for the PaP operations involving the lightest objects in the scene. This can primarily be attributed to the fact that the value assigned to the workload weighting coefficient ($\beta = 4$) is higher than that assigned to the energy consumption weighting coefficient ($\gamma = 1$). As discussed for the simulation case study, this configuration encourages the assignment of the operation involving the lightest objects in the scene to the human and those involving the heaviest ones to the robots. Figure 6.12

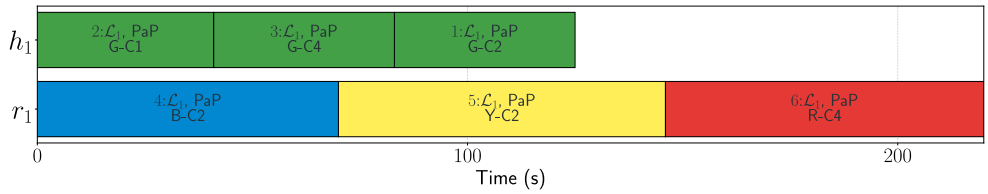


Figure 6.11: Optimal plan resulting from the real-world laboratory experiment for the assembly of the capital letter "L".

shows key moments of the experiment. Specifically, Figure 6.12 top-left) provides an overview of the scenario at $t = 0s$, with all objects located at their starting positions. Figure 6.12 top-right) shows the execution at $t \approx 90s$, where the human operator and the robot were performing the PaP of the "green-cube2" and the "yellow-cube2", respectively. Figure 6.12 bottom-left) highlights the phase of the plan corresponding to $t \approx 130s$, in which the human operator was awaiting the next task and the robot was completing the PaP of the "yellow-cube2". Finally, Figure 6.12 bottom-right) provides an overview of the final configuration of the scenario, which reflects the desired letter assembly.

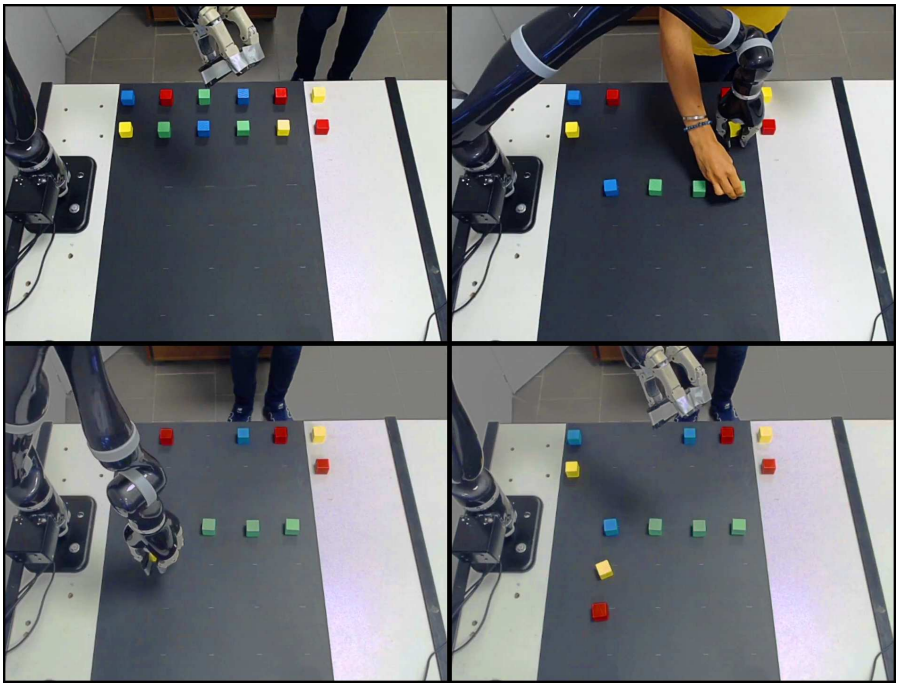


Figure 6.12: Screenshots from the real-world experiment.

In conclusion, the experiment validated the effectiveness of the method in realizing collaborative allocation and scheduling, given a natural language description of the desired mission.

6.6 Conclusions

This chapter presented an architecture designed to optimize task allocation and scheduling in scenarios involving heterogeneous multi-agent teams. In detail, by leveraging the world-knowledge capabilities of LLMs, a Context Generator transforms the natural language mission description into an ordered and structured sequence of tasks. Subsequently, a CP-based Optimizer formulates and solves a constraint optimization problem to achieve an optimal task allocation and scheduling, consistent with temporal, resource, and agent-specific constraints. The effectiveness of the proposed architecture was first assessed in simulation against two baseline approaches, across missions of different complexity and LLM models, and

then in a laboratory environment with human-robot teams. The experimental results demonstrated the superiority of the proposed framework, which produced more optimal solutions in terms of makespan, energy consumption, and human workload, while requiring less computation time. Furthermore, the findings highlighted the critical limitations of a purely LLM-driven approach, which exhibited a high rate of plan infeasibility, thereby confirming that a formal optimization layer is essential for ensuring safety and reliability in real-world applications.

Chapter 7

Conclusions and Future Works

The work presented in this thesis introduced a series of control and decision-making strategies integrated into a comprehensive two-layer architecture, consisting of a control layer and a task coordination layer. At the control layer, the proposed architecture combines Hierarchical Quadratic Programming (HQP) and Control Barrier Functions (CBFs) to enable robots to autonomously manage internal configurations while ensuring operational safety during task execution. To promote safe yet effective human–robot collaborations, the architecture also integrates a Safety Planner and a Shared Control Strategy. The Safety Planner guarantees human safety by quantifying it through a dedicated safety field and using this measure to compute optimal trajectory adjustments—either by reducing velocity or deviating from a nominal path—through the solution of a Quadratic Programming (QP) problem. Conversely, the Shared Control Strategy provides smooth modulation of the system's autonomy level by dynamically adjusting the robot's admittance parameters.

At the task coordination layer, the architecture incorporates a dynamic framework for task allocation and scheduling, employing a Mixed-Integer Linear Programming (MILP) formulation to optimize resource usage and performance under human variability and real-world uncertainties. The integration of stochastic human models and real-time feedback enables the framework to adapt the plan in response to evolving operator conditions and external factors, effectively balancing objectives such as makespan, waiting time, and energy consumption.

In addition to this framework, a second one is introduced at the same

layer, designed to manage large-scale problems and bridge the gap between ambiguous human instructions and the generation of feasible, optimal, and reliable plans. This framework combines Large Language Models (LLMs) with Constraint Programming (CP). Specifically, it leverages the natural language understanding and reasoning capabilities of LLMs to process ambiguous human inputs, while relying on CP to ensure the generation of optimal and constraint-compliant plans.

The proposed architecture has been extensively validated across simulation, laboratory, and real-world environments. Real-world experiments were conducted within the context of the H2020 EU-funded project CANOPIES. Future research will focus on: *i*) further integrating LLM-based reasoning into the architecture, particularly at the control layer, to enhance the system's autonomy and decision-making capabilities; *ii*) incorporating LLM-based or EMG-based methodologies to enable online adaptation of admittance parameters according to both human and robot states; and *iii*) integrating vision-based systems at the task allocation layer for behavioral interpretation and action prediction, thereby improving responsiveness to human needs and preferences.

Bibliography

[1] N. Ghodsian, K. Benfriha, A. Olabi, V. Gopinath, and A. Arnou, "Mobile manipulators in industry 4.0: A review of developments for industrial applications," *Sensors*, vol. 23, no. 19, p. 8026, 2023.

[2] M. Garabini, D. Caporale, V. Tincani, A. Palleschi, C. Gabellieri, M. Gugliotta, A. Settimi, M. G. Catalano, G. Grioli, and L. Pallottino, "Wrapp-up: A dual-arm robot for intralogistics," *IEEE Robotics & Automation Magazine*, vol. 28, no. 3, pp. 50–66, 2021.

[3] P. Štibinger, G. Broughton, F. Majer, Z. Rozsypálek, A. Wang, K. Jindal, A. Zhou, D. Thakur, G. Loianno, T. Krajník, and M. Saska, "Mobile manipulator for autonomous localization, grasping and precise placement of construction material in a semi-structured environment," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2595–2602, 2021.

[4] A. Ollero, M. Tognon, A. Suarez, D. Lee, and A. Franchi, "Past, present, and future of aerial robotic manipulators," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 626–645, 2022.

[5] A. Hentout, M. Aouache, A. Maoudj, and I. Akli, "Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017," *Advanced Robotics*, vol. 33, no. 15-16, pp. 764–799, 2019.

[6] A. E. Abdelaal, P. Mathur, and S. E. Salcudean, "Robotics in vivo: A perspective on human–robot interaction in surgical robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 221–242, 2020.

[7] J. P. Vasconez, G. A. Kantor, and F. A. A. Cheein, "Human–robot interaction in agriculture: A survey and current challenges," *Biosystems engineering*, vol. 179, pp. 35–48, 2019.

[8] J. Palmieri, P. Di Lillo, M. Lippi, S. Chiaverini, and A. Marino, "A control architecture for safe trajectory generation in human-robot collaborative settings," *IEEE Transactions on Automation Science and Engineering*, 2024.

[9] K. S. Jones and E. A. Schmidlin, "Human-robot interaction: toward usable personal service robots," *Reviews of Human Factors and Ergonomics*, vol. 7, no. 1, pp. 100–148, 2011.

[10] P. Di Lillo, E. Simetti, F. Wanderlingh, G. Casalino, and G. Antonelli, "Underwater intervention with remote supervision via satellite communication: Developed control architecture and experimental results within the dexrov project," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 1, pp. 108–123, 2021.

[11] G. Wang, W. Wang, P. Ding, Y. Liu, H. Wang, Z. Fan, H. Bai, Z. Hongbiao, and Z. Du, "Development of a search and rescue robot system for the underground building environment," *Journal of Field Robotics*, vol. 40, no. 3, pp. 655–683, 2023.

[12] M. Selvaggio, M. Cognetti, S. Nikolaidis, S. Ivaldi, and B. Siciliano, "Autonomy in physical human-robot interaction: A brief survey," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7989–7996, 2021.

[13] S. Ramadurai and H. Jeong, "Effect of human involvement on work performance and fluency in human-robot collaboration for recycling," 2022, pp. 1007–1011.

[14] S. Kumar, C. Savur, and F. Sahin, "Survey of human–robot collaboration in industrial settings: Awareness, intelligence, and compliance," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 51, no. 1, pp. 280–297, 2021.

[15] F. A. Cheein, D. Herrera, J. Gimenez, R. Carelli, M. Torres-Torriti, J. R. Rosell-Polo, A. Escolà, and J. Arnó, "Human-robot interaction in precision agriculture: Sharing the workspace with service units," in *IEEE Int. Conf. Ind. Technol.*, 2015, pp. 289–295.

[16] M. Lippi and A. Marino, "Human multi-robot physical interaction: a distributed framework," *J. Intell. Robot. Syst.*, vol. 101, no. 2, p. 35, 2021.

[17] S. E. Hashemi-Petroodi, S. Thevenin, S. Kovalev, and A. Dolgui, "Operations management issues in design and control of hybrid human-robot collaborative manufacturing systems: a survey," *Annual Reviews in Control*, vol. 49, pp. 264–276, 2020.

[18] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon, "A human aware mobile robot motion planner," vol. 23, no. 5, pp. 874–883, 2007.

[19] J. Truc, P.-T. Singamaneni, D. Sidobre, S. Ivaldi, and R. Alami, "Khaos: a kinematic human aware optimization-based system for reactive planning of flying-coworker," 2022, pp. 4764–4770.

[20] G. Silano, A. Afifi, M. Saska, and A. Franchi, "A signal temporal logic planner for ergonomic human–robot collaboration," in *Int. Conf. Unmanned Aircraft Syst.*, 2023, pp. 328–335.

[21] C. L. Bethel, K. Salomon, R. R. Murphy, and J. L. Burke, "Survey of psychophysiology measurements applied to human-robot interaction," 2007, pp. 732–737.

[22] D. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on Man-Machine Systems*, vol. 10, no. 2, pp. 47–52, 1969.

[23] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[24] P. Di Lillo, F. Arrichiello, G. Antonelli, and S. Chiaverini, "Safety-related tasks within the set-based task-priority inverse kinematics framework," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6130–6135.

[25] B. Siciliano and J.-J. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, 1991, pp. 1211–1216 vol.2.

[26] E. Simetti and G. Casalino, "A novel practical technique to integrate inequality control objectives and task transitions in priority based control," *Journal of Intelligent & Robotic Systems*, vol. 84, 12 2016.

[27] S. Moe, G. Antonelli, A. Teel, K. Pettersen, and 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*, vol. 3, 04 2016.

[28] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431.

[29] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[30] J. Slotine and B. Siciliano, "A general framework for managing multiple tasks in highly redundant robotic systems," in *proceeding of 5th International Conference on Advanced Robotics*, vol. 2, 1991, pp. 1211–1216.

[31] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987.

[32] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial Intelligence*, vol. 1, no. 3, pp. 193–204, 1970. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000437027090007X

[33] C. Floudas and X. Lin, "Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications," *Annals of Operations Research*, vol. 139, pp. 131–162, 10 2005.

[34] J. Cai, K. Nguyen, N. Shrestha, A. Good, R. Tu, X. Yu, S. Zhe, and T. Serra, *Getting Away with More Network Pruning: From Sparsity to Geometry and Linear Regions*, 05 2023, pp. 200–218.

[35] N. Ejaz and S. Choudhury, "A comprehensive survey of linear, integer, and mixed-integer programming approaches for optimizing resource allocation in 5g and beyond networks," 2025. [Online]. Available: https://arxiv.org/abs/2502.15585

[36] H. Ling, Z. Wang, and J. Wang, "Learning to stop cut generation for efficient mixed-integer linear programming," 2024. [Online]. Available: https://arxiv.org/abs/2401.17527

[37] K. E. C. Booth, T. T. Tran, G. Nejat, and J. C. Beck, "Mixed-integer and constraint programming techniques for mobile robot task planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 500–507, 2016.

[38] F. Rossi, P. van Beek, and T. Walsh, "Chapter 1 - introduction," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, F. Rossi, P. van Beek, and T. Walsh, Eds. Elsevier, 2006, vol. 2, pp. 3–12. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574652606800052

[39] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-based scheduling: applying constraint programming to scheduling problems*. Springer Science & Business Media, 2001, vol. 39.

[40] Y. Bukchin and T. Raviv, "Constraint programming for solving various assembly line balancing problems," *Omega*, vol. 78, pp. 57–68, 2018.

[41] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. USA: Elsevier Science Inc., 2006.

[42] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2307.06435

[43] A. Neumann, E. Kirsten, M. B. Zafar, and J. Singh, "Position is power: System prompts as a mechanism of bias in large language models (llms)," in *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '25. ACM, Jun. 2025, p. 573–598. [Online]. Available: http://dx.doi.org/10.1145/3715275.3732038

[44] K. Chang, S. Xu, C. Wang, Y. Luo, X. Liu, T. Xiao, and J. Zhu, "Efficient prompting methods for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2404.01077

[45] A. Mukhtar Abubakar, D. Gupta, and S. Parida, "A reinforcement learning approach for intelligent conversational chatbot for enhancing mental health therapy," *Procedia Computer Science*, vol. 235, pp. 916–925, 01 2024.

[46] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can, not as i say: Grounding language in robotic affordances," 2022. [Online]. Available: https://arxiv.org/abs/2204.01691

[47] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2025. [Online]. Available: https://arxiv.org/abs/2402.06196

[48] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati, "Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change," 2023. [Online]. Available: https://arxiv.org/abs/2206.10498

[49] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, vol. 43, no. 2, p. 1–55, Jan. 2025. [Online]. Available: http://dx.doi.org/10.1145/3703155

[50] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[51] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[52] G. Antonelli, "Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems," *IEEE Transactions on Robotics*, vol. 25, no. 5, pp. 985–994, 2009.

[53] F. Flacco, A. De Luca, and O. Khatib, "Control of redundant robots under hard joint constraints: Saturation in the null space," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 637–654, 2015.

[54] E. Simetti and G. Casalino, "A novel practical technique to integrate inequality control objectives and task transitions in priority based control," *Journal of Intelligent & Robotic Systems*, vol. 84, pp. 877–902, 2016.

[55] P. Di Lillo, F. Pierri, G. Antonelli, F. Caccavale, and A. Ollero, "A framework for set-based kinematic control of multi-robot systems," *Control Engineering Practice*, vol. 106, p. 104669, 2021.

[56] Y. Lee, J. Ahn, J. Lee, and J. Park, "Computationally efficient hqp-based whole-body control exploiting the operational-space formulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 5197–5202.

[57] M.-J. Kim, D. Lim, G. Park, and J. Park, "Humanoid balance control using centroidal angular momentum based on hierarchical quadratic programming," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 6753–6760.

[58] D. Koung, O. Kermorgant, I. Fantoni, and L. Belouaer, "Cooperative multi-robot object transportation system based on hierarchical quadratic programming," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6466–6472, 2021.

[59] F. Ferraguti, C. T. Landi, A. Singletary, H.-C. Lin, A. Ames, C. Secchi, and M. Bonfè, "Safety and efficiency in robotics: The control barrier functions approach," *IEEE Robotics & Automation Magazine*, vol. 29, no. 3, pp. 139–151, 2022.

[60] R. Gehlhar and A. D. Ames, "Separable control lyapunov functions with application to prostheses," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 559–564, 2021.

[61] E. A. Basso and K. Y. Pettersen, "Task-priority control of redundant robotic systems using control lyapunov and control barrier function based quadratic programs," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9037–9044, 2020.

[62] Q. Nguyen and K. Sreenath, "Robust safety-critical control for dynamic robotics," *IEEE Transactions on Automatic Control*, vol. 67, no. 3, pp. 1073–1088, 2022.

[63] P. Chiacchio, S. Chiaverini, and B. Siciliano, "Direct and inverse kinematics for coordinated motion tasks of a two-manipulator system," *Journal of Dynamic Systems, Measurement, and Control*, vol. 118, no. 4, pp. 691–697, 12 1996. [Online]. Available: https://doi.org/10.1115/1.2802344

[64] F. Caccavale and M. Uchiyama, *Cooperative Manipulation*. Cham: Springer International Publishing, 2016, pp. 989–1006. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_39

[65] H. A. Park and C. G. Lee, "Extended cooperative task space for manipulation tasks of humanoid robots," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6088–6093.

[66] F. Basile, F. Caccavale, P. Chiacchio, J. Coppola, and A. Marino, "A decentralized kinematic control architecture for collaborative and cooperative multi-arm systems," *Mechatronics*, vol. 23, no. 8, pp. 1100–1112, 2013.

[67] J. Jiang, Y. Wang, Y. Jiang, H. Xie, H. Tan, and H. Zhang, "A robust visual servoing controller for anthropomorphic manipulators with field-of-view constraints and swivel-angle motion: Overcoming system uncertainty and improving control performance," *IEEE Robotics & Automation Magazine*, vol. 29, no. 4, pp. 104–114, 2022.

[68] G. Coll-Ribes, I. J. Torres-Rodríguez, A. Grau, E. Guerra, and A. Sanfeliu, "Accurate detection and depth estimation of table grapes and peduncles for robot harvesting, combining monocular

depth estimation and cnn methods," *Computers and Electronics in Agriculture*, vol. 215, December 2023.

[69] T. A. Ciarfuglia, I. M. Motoi, L. Saraceni, M. Fawakherji, A. Sanfeliu, and D. Nardi, "Weakly and semi-supervised detection, segmentation and tracking of table grapes with limited and noisy data," *Computers and Electronics in Agriculture*, vol. 205, p. 107624, 2023.

[70] D. Kulić and E. A. Croft, "Real-time safety for human–robot interaction," *Rob. Auton. Syst.*, vol. 54, no. 1, pp. 1 – 12, 2006.

[71] N. Najmaei and M. R. Kermani, "Prediction-based reactive control strategy for human-robot interactions," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 3434–3439.

[72] F. Flacco, T. Kröger, A. D. Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *IEEE Int. Conf. Robot. Autom.*, 2012, pp. 338–345.

[73] H. Liu, D. Qu, F. Xu, Z. Du, K. Jia, J. Song, and M. Liu, "Real-time and efficient collision avoidance planning approach for safe human-robot interaction," *Journal of Intelligent & Robotic Systems*, vol. 105, no. 4, p. 93, 2022.

[74] K. Merckaert, B. Convens, C.-j. Wu, A. Roncone, M. M. Nicotra, and B. Vanderborght, "Real-time motion control of robotic manipulators for safe human–robot coexistence," *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102223, 2022.

[75] J. A. Marvel, "Performance metrics of speed and separation monitoring in shared workspaces," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 2, pp. 405–414, 2013.

[76] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 882–893, 2016.

[77] B. Lacevic, A. M. Zanchettin, and P. Rocco, "Safe human-robot collaboration via collision checking and explicit representation of danger zones," *IEEE Transactions on Automation Science and Engineering*, 2022.

[78] L. Scalera, R. Vidoni, and A. Giusti, "Optimal scaling of dynamic safety zones for collaborative robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 3822–3828.

[79] M. Faroni, M. Beschi, and N. Pedrocchi, "Safety-aware time-optimal motion planning with uncertain human state estimation," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12 219–12 226, 2022.

[80] M. Costanzo, G. De Maria, G. Lettera, and C. Natale, "A multimodal approach to human safety in collaborative robotic workcells," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 1202–1216, 2022.

[81] M. Lippi and A. Marino, "Human multi-robot safe interaction: A trajectory scaling approach based on safety assessment," *IEEE Trans. Control Syst. Technol.*, pp. 1–16, 2020.

[82] M. Ye, Q. Zhang, L. Wang, J. Zhu, R. Yang, and J. Gall, *A Survey on Human Motion Analysis from Depth Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 149–187.

[83] L. A. Schwarz, A. Mkhitaryan, D. Mateus, and N. Navab, "Human skeleton tracking from depth data using geodesic distances and optical flow," *Image and Vision Computing*, vol. 30, no. 3, pp. 217–226, 2012.

[84] O. H. Jafari, D. Mitzel, and B. Leibe, "Real-time rgb-d based people detection and tracking for mobile robots and head-worn cameras," in *IEEE Int. Conf. Robot. Autom.*, 2014, pp. 5636–5643.

[85] T. Arai, R. Kato, and M. Fujita, "Assessment of operator stress induced by robot collaboration in assembly," *CIRP annals*, vol. 59, no. 1, pp. 5–8, 2010.

[86] P. Di Lillo, D. Di Vito, and G. Antonelli, "Merging global and local planners: real-time replanning algorithm of redundant robots within a task-priority framework," *IEEE Transactions on Automation Science and Engineering*, 2022.

[87] B. Lacevic, P. Rocco, and A. M. Zanchettin, "Safety assessment and control of robotic manipulators using danger field," *IEEE Trans. Robot.*, vol. 29, no. 5, pp. 1257–1270, 2013.

[88] B. Xi, S. Wang, X. Ye, Y. Cai, T. Lu, and R. Wang, "A robotic shared control teleoperation method based on learning from demonstrations," *International Journal of Advanced Robotic Systems*, vol. 16, no. 4, p. 1729881419857428, 2019.

[89] D. Losey, C. McDonald, E. Battaglia, and M. O'Malley, "A review of intent detection, arbitration, and communication aspects of shared control for physical human–robot interaction," *Applied Mechanics Reviews*, vol. 70, 01 2018.

[90] Y. Li, K. P. Tee, W. L. Chan, R. Yan, Y. Chua, and D. K. Limbu, "Continuous role adaptation for human–robot shared control," *IEEE Trans. Robot.*, vol. 31, no. 3, pp. 672–681, June 2015.

[91] A. Mörtl, M. Lawitzky, A. Kucukyilmaz, M. Sezgin, C. Basdogan, and S. Hirche, "The role of roles: Physical cooperation between humans and robots," *Int. J. Robot. Res.*, vol. 31, no. 13, pp. 1656–1674, 2012.

[92] J. R. Medina, T. Lorenz, and S. Hirche, "Synthesizing anticipatory haptic assistance considering human behavior uncertainty," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 180–190, Feb 2015.

[93] D. P. Losey and M. K. O'Malley, "Trajectory deformations from physical human–robot interaction," *IEEE Trans. Robot.*, vol. 34, no. 1, pp. 126–138, Feb 2018.

[94] R. Balachandran, H. Mishra, M. Cappelli, B. Weber, C. Secchi, C. Ott, and A. Albu-Schaeffer, "Adaptive authority allocation in shared control of robots using bayesian filters," in *2020 IEEE International Conf. on Robotics and Automation (ICRA)*, 2020, pp. 11 298–11 304.

[95] M. Hagenow, E. Senft, R. Radwin, M. Gleicher, B. Mutlu, and M. Zinn, "Corrective shared autonomy for addressing task variability," *IEEE Rob. and Autom. Letters*, vol. 6, no. 2, pp. 3720–3727, 2021.

[96] J. Chen and P. I. Ro, "Human intention-oriented variable admittance control with power envelope regulation in physical human-robot interaction," *Mechatronics*, vol. 84, p. 102802, 2022.

[97] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[98] R. P. Joshi, A. Choi, X. Z. Tan, M. K. Van den Broek, R. Luo, and B. Choi, "ROS OpenPose," https://github.com/ravijo/ros_openpose, 2019.

[99] R. Maderna, M. Poggiali, A. M. Zanchettin, and P. Rocco, "An online scheduling algorithm for human-robot collaborative kitting," 2020, pp. 11 430–11 435.

[100] R. Iftikhar, Y.-T. Chiu, M. S. Khan, and C. Caudwell, "Human–agent team dynamics: A review and future research opportunities," *IEEE Trans. Eng. Manag.*, vol. 71, pp. 10 139–10 154, 2024.

[101] H. Chen, S. Alghowinem, C. Breazeal, and H. W. Park, "Integrating flow theory and adaptive robot roles: A conceptual model of dynamic robot role adaptation for the enhanced flow experience in long-term multi-person human-robot interactions," 2024, pp. 116–126.

[102] A. Gottardi, M. Terreran, C. Frommel, M. Schoenheits, N. Castaman, S. Ghidoni, and E. Menegatti, "Dynamic human-aware task planner for human-robot collaboration in industrial scenario," in *Eur. Conf. Mob. Robots*, 2023, pp. 1–8.

[103] J. Kaduk, M. Cavdan, K. Drewing, and H. Hamann, "From one to many: How active robot swarm sizes influence human cognitive processes," 2024, pp. 1207–1212.

[104] B. Leichtmann and V. Nitsch, "How much distance do humans keep toward robots? literature review, meta-analysis, and theoretical considerations on personal space in human-robot interaction," *J. Environ. Psychol.*, vol. 68, p. 101386, 2020.

[105] S. Zhang, Y. Chen, J. Zhang, and Y. Jia, "Real-time adaptive assembly scheduling in human-multi-robot collaboration according to human capability," 2020, pp. 3860–3866.

[106] M. Pearce, B. Mutlu, J. Shah, and R. Radwin, "Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1772–1784, 2018.

[107] M. Lippi, P. Di Lillo, and A. Marino, "A task allocation framework for human multi-robot collaborative settings," 2023, pp. 7614–7620.

[108] A. Nourmohammadi, M. Fathi, and A. H. Ng, "Balancing and scheduling assembly lines with human-robot collaboration tasks," *Comp. & Oper. Res.*, vol. 140, p. 105674, 2022.

[109] I. Dimény, T. Koltai, C. Sepe, T. Murino, V. Gallina, and T. Komenda, "Milp model to decrease the number of workers in assembly lines with human-robot collaboration," *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 169–174, 2021.

[110] N. Dhanaraj, S. V. Narayan, S. Nikolaidis, and S. K. Gupta, "Contingency-aware task assignment and scheduling for human-robot teams," 2023, pp. 5765–5771.

[111] D. Guo, "Fast scheduling of human-robot teams collaboration on synchronised production-logistics tasks in aircraft assembly," *Robot. Comput.-Integr. Manuf.*, vol. 85, p. 102620, 2024.

[112] A. Pupa and C. Secchi, "A safety-aware architecture for task scheduling and execution for human-robot collaboration," 2021, pp. 1895–1902.

[113] S. Alirezazadeh and L. A. Alexandre, "Dynamic task scheduling for human-robot collaboration," vol. 7, no. 4, pp. 8699–8704, 2022.

[114] A. Monguzzi, M. Badawi, A. M. Zanchettin, and P. Rocco, "A mixed capability-based and optimization methodology for human-robot task allocation and scheduling," 2022, pp. 1271–1276.

[115] F. Fusaro, E. Lamon, E. De Momi, and A. Ajoudani, "An integrated dynamic method for allocating roles and planning tasks for mixed human-robot teams," 2021, pp. 534–539.

[116] T. Bänziger, A. Kunz, and K. Wegener, "Optimizing human–robot task allocation using a simulation tool based on standardized work descriptions," vol. 31, pp. 1635–1648, 2020.

[117] C. Ferreira, G. Figueira, and P. Amorim, "Scheduling human-robot teams in collaborative working cells," vol. 235, p. 108094, 2021.

[118] N. D. Tehrani, A. Krzywosz, I. Cherepinsky, and S. Carlson, "Multi-objective task allocation for multi-agent systems using hierarchical cost function," 2022, pp. 12 045–12 050.

[119] L. H. De Mello and A. C. Sanderson, "And/or graph representation of assembly plans," *IEEE Trans. Robot. Autom.*, vol. 6, no. 2, pp. 188–199, 1990.

[120] A. Pupa, W. Van Dijk, C. Brekelmans, and C. Secchi, "A resilient and effective task scheduling approach for industrial human-robot collaboration," *Sensors*, vol. 22, no. 13, p. 4901, 2022.

[121] E. Merlo, E. Lamon, F. Fusaro, M. Lorenzini, A. Carfi, F. Mastrogiovanni, and A. Ajoudani, "Dynamic human-robot role allocation based on human ergonomics risk prediction and robot actions adaptation," 2022, pp. 2825–2831.

[122] H.-R. Lee, S. Park, and J. Lee, "Bayesian reinforcement learning for adaptive balancing in an assembly line with human-robot collaboration," *IEEE Access*, 2024.

[123] T. Yu, J. Huang, and Q. Chang, "Optimizing task scheduling in human-robot collaboration with deep multi-agent reinforcement learning," vol. 60, pp. 487–499, 2021.

[124] R. Liu, M. Natarajan, and M. C. Gombolay, "Coordinating human-robot teams with dynamic and stochastic task proficiencies," vol. 11, no. 1, pp. 1–42, 2021.

[125] T. Wu, Z. Zhang, L. Guo, H. Song, X. Xie, and S. Ren, "A hybrid evolutionary algorithm for the stochastic human–robot collaborative disassembly line balancing problem considering carbon emission optimization," *Eng. Appl. Artif. Intell.*, vol. 135, p. 108703, 2024.

[126] Y. Abu Farha and J. Gall, "Uncertainty-aware anticipation of activities," in *IEEE/CVF Int. Conf. Computer Vision*, 2019.

[127] S. Mugisha, V. K. Guda, C. Chevallereau, D. Chablat, and M. Zoppi, "Motion prediction with gaussian processes for safe human-robot interaction in virtual environments," *IEEE Access*, 2024.

[128] K. Lockwood, G. Strenge, Y. Bicer, T. Imbiriba, M. P. Furmanek, T. Padir, D. Erdogmus, E. Tunik, and M. Yarossi, "Gaussian process-based prediction of human trajectories to promote seamless human-robot handovers," 2023, pp. 2259–2266.

[129] L. Tsao, L. Li, and L. Ma, "Human work and status evaluation based on wearable sensors in human factors and ergonomics: A review," *IEEE Trans. Human-Mach. Syst.*, vol. 49, no. 1, pp. 72–84, 2018.

[130] A. Ajoudani, A. M. Zanchettin, S. Ivaldi, A. Albu-Schäffer, K. Kosuge, and O. Khatib, "Progress and prospects of the human–robot collaboration," *Autonomous Robots*, vol. 42, pp. 957–975, 2018.

[131] P. Tokekar, N. Karnad, and V. Isler, "Energy-optimal velocity profiles for car-like robots," 2011, pp. 1457–1462.

[132] R. Parasuraman, K. Kershaw, P. Pagala, and M. Ferre, "Model based on-line energy prediction system for semi-autonomous mobile robots," in *Int. Conf. Intelligent Systems, Modelling and Simulation*, 2014, pp. 411–416.

[133] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," vol. 27, no. 6, pp. 1080–1094, 2011.

[134] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[135] G. M. Kopanos, C. A. Méndez, and L. Puigjaner, "Mip-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry," *European J. Oper. Res.*, vol. 207, no. 2, pp. 644–655, 2010.

[136] M. C. Ferris, C. T. Maravelias, and A. Sundaramoorthy, "Simultaneous batching and scheduling using dynamic decomposition on a grid," *INFORMS Journal on Computing*, vol. 21, no. 3, pp. 398–410, 2009.

[137] I. Harjunkoski, C. T. Maravelias, P. Bongers, P. M. Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand, and J. Wassick, "Scope for industrial applications of production scheduling models and solution methods," *Comp. & Chemical. Eng.*, vol. 62, pp. 161–193, 2014.

[138] G. P. Georgiadis, A. P. Elekidis, and M. C. Georgiadis, "Optimization-based scheduling for the process industries: from theory to real-life industrial applications," *Processes*, vol. 7, no. 7, p. 438, 2019.

[139] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[140] Y. Kong and Y. Fu, "Human action recognition and prediction: A survey," *Int. J. Comp. Vision*, vol. 130, no. 5, pp. 1366–1401, 2022.

[141] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, 2010.

[142] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Auton. robots*, vol. 13, pp. 207–222, 2002.

[143] M. Lippi, J. Gallou, J. Palmieri, A. Gasparri, and A. Marino, "Human-multi-robot task allocation in agricultural settings: a mixed integer linear programming approach," 2023, pp. 1056–1062.

[144] S. S. Kannan, V. L. N. Venkatesh, and B.-C. Min, "SMART-LLM: Smart multi-agent robot task planning using large language models," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*. IEEE, 2024.

[145] X. Zhang, H. Qin, F. Wang, Y. Dong, and J. Li, "LaMMA-P: Generalizable Multi-Agent Long-Horizon Task Allocation and Planning with LM-Driven PDDL Planner," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[146] A. Torreno, E. Onaindia, A. Komenda, and M. Stolba, "Cooperative multi-agent planning: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, pp. 1–32, 2017.

[147] M. Peng, Z. Chen, J. Yang, J. Huang, Z. Shi, Q. Liu, X. Li, and L. Gao, "Automatic MILP Model Construction for Multi-Robot Task Allocation and Scheduling Based on Large Language Models," *arXiv preprint arXiv:2503.13813*, 2025.

[148] K. Obata, T. Aoki, T. Horii, T. Taniguchi, and T. Nagai, "LiP-LLM: Integrating Linear Programming and dependency graph with Large Language Models for multi-robot task planning," *arXiv preprint arXiv:2410.21040*, 2024.

[149] S. Bezrucav, Y. Liu, and B. Corves, "Optimization-based or ai task planning for scenarios with cooperating mobile manipulators?" in *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics - ICINCO*, INSTICC. SciTePress, 2021, pp. 115–122.

[150] Q. Chen and Y.-J. Pan, "An optimal task planning and agent-aware allocation algorithm in collaborative tasks combining with pddl and popf," 2024.

[151] F. Ranz, V. Hummel, and W. Sihn, "Capability-based task allocation in human-robot collaboration," *Procedia Manufacturing*, vol. 9, pp. 182–189, 2017, 7th Conference on Learning Factories, CLF 2017.

[152] M. Deng, B. Fu, L. Li, and X. Wang, "Integrating LLMs and Digital Twins for Adaptive Multi-Robot Task Allocation in Construction," *arXiv preprint arXiv:2506.18178*, 2025.

[153] P. Gupta, D. Isele, B. Dariush, E. Sachdeva, P.-H. Huang, S. Bae, and K. Lee, "Generalized Mission Planning for Heterogeneous Multi-Robot Teams via LLM-constructed Hierarchical Trees," *arXiv preprint arXiv:2501.16539*, 2025, v1.

[154] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," 2023.

[155] S. Izquierdo-Badiola, G. Canal, C. Rizzo, and G. Alenyà, "Plancollabnl: Leveraging large language models for adaptive plan generation in human-robot collaboration," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 17 344–17 350.

[156] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 286–299.

[157] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov, "Can language models solve graph problems in natural language?" *Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS)*, vol. 37, pp. 30 840–30 861, 2024.

[158] Y. Talebirad and A. Nadiri, "Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents," 2023.

[159] I. Obi, V. L. Venkatesh, W. Wang, R. Wang, D. Suh, T. I. Amosa, W. Jo, and B.-C. Min, "SafePlan: Leveraging Formal Logic and Chain-of-Thought Reasoning for Enhanced Safety in LLM-based Robotic Task Planning," *arXiv preprint arXiv:2503.06892*, 2025.

[160] Z. Zhou, K. Yao, J. Song, Z. Shu, and L. Ma, "ISR-LLM: Iterative Self-Refined Large Language Model for Long-Horizon Sequential Task Planning," *arXiv preprint arXiv:2308.13724*, 2023, v1.

[161] J. Huang, R. Wang, J. Li, and M. Yang, "A survey on zero-shot planners with large language models," in *International Conference on Machine Learning*, 2022.

[162] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[163] Z. Xiao, J. Yu, H. Zheng, and M. Deng, "Chain-of-experts: When llms meet complex operations research problems," in *The twelfth international conference on learning representations*, 2023.

[164] A. AhmadiTeshnizi, W. Gao, and M. Udell, "Optimus: Scalable optimization modeling with (mi) lp solvers and large language models," *arXiv preprint arXiv:2402.10172*, 2024.

[165] K. Shirai, C. C. Beltran-Hernandez, M. Hamaya, A. Hashimoto, S. Tanaka, K. Kawaharazuka, K. Tanaka, Y. Ushiku, and S. Mori, "Vision-language interpreter for robot task planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 2051–2058.

[166] N. Gupta and D. S. Nau, "On the complexity of blocks-world planning," in *Artificial Intelligence*, vol. 56, no. 2, 1992, pp. 223–254.

[167] B. S. R. Armstrong, "Control of machines with non-linear, low-velocity friction: A dimensional analysis," in *Experimental Robotics I*, V. Hayward and O. Khatib, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 180–195.

[168] M. Nakamura, S. Goto, and N. Kyura, *4 Quantization Error of a Mechatronic Servo System*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 79–96.

[169] A. Pupa and C. Secchi, "A safety-aware architecture for task scheduling and execution for human-robot collaboration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1895–1902.

[170] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *International Conference on Neural Information Processing Systems (NIPS)*, 2022, pp. 24 824–24 837.

[171] B. Zhang, C. Lu, L. lei Meng, Y. yan Han, H. Sang, and X. chu Jiang, "Reconfigurable distributed flowshop group scheduling with a nested variable neighborhood descent algorithm," in *Expert Systems with Applications*, vol. 217, 2023, p. 119548.

[172] F. Cruz-Mencia, J. Cerquides, A. Espinosa, J. C. Moure, and J. A. Rodriguez-Aguilar, "Parallelisation and Application of AD 3 as a Method for Solving Large Scale Combinatorial Auctions," in *Lecture Notes in Computer Science*, vol. LNCS-9037, 2015, pp. 153–168.

[173] S. A. Prieto and B. Garcia de Soto, "Large Language Models for Robot Task Allocation," in *3rd Future of Construction Workshop at the International Conference on Robotics and Automation (ICRA)*, 2024, pp. 17–20.

[174] Y. Lee, K. Son, T. S. Kim, J. Kim, J. J. Y. Chung, E. Adar, and J. Kim, "One vs. many: Comprehending accurate information from multiple erroneous and inconsistent ai generations," in *ACM Conference on Fairness, Accountability, and Transparency*, 2024, pp. 2518—2531.

[175] Q. V. Liao and J. Wortman Vaughan, "Ai Transparency in the Age of LLMs: A Human-Centered Research Roadmap," in *Harvard Data Science Review, Special Issue 5*, 2024.