UNIVERSITÀ DEGLI STUDI DI PADOVA

KUNGL TEKNISKA HÖGSKOLAN

Department of Information Engineering
Master Degree in Control Systems Engineering

Master's Thesis

Multi-Robot Coordination for Precision Agriculture Under Recurring Linear Temporal Logic

Developing an Efficient Synchronization Algorithm for Real-World Collaboration

Supervisor:
Professor Angelo Cenedese

KTH Supervisor:
Professor Dimos V. Dimarogonas

KTH Co-Supervisors:
Eleftherios Vlahakis
Victor Nan Fernandez-Ayala

Candidate:
Davide Peron

Academic Year 2023/2024

Graduation Date 03/09/2024

Multi-Robot Coordination for Precision Agriculture Under Recurring Linear Temporal Logic

Developing an Efficient Synchronization Algorithm for Real-World Collaboration

DAVIDE PERON

Date: August 27, 2024

Supervisor: Angelo Cenedese
Department of Information Engineering, UniPD
KTH Supervisors: Dimos V. Dimarogonas, Eleftherios Vlahakis, Victor Nan Fernandez-Ayala
Division of Decision and Control Systems, EECS, KTH

Abstract

The efficient coordination and collaboration of autonomous agents are critical for achieving complex tasks in various industrial and research applications. In this study, we focus on recurring tasks, typical in agronomic applications, where tasks need to be executed repeatedly over an indefinite horizon. We formally specify such tasks using Linear Temporal Logic (LTL), for planning and coordination of multi-agent systems. We develop a software package for multi-agent planning and coordination with a focus on precision agriculture, which can straightforwardly be extended to a wider range of multi-robot control applications. The main objective of the thesis is to develop a planning strategy that enables agents to execute tasks collaboratively, ensuring both reliability and reducing computational complexity compared to previous approaches.
To this end, we adapt a bottom-up approach to motion and task coordination that includes an offline plan synthesis and an online coordination scheme based on real-time exchange of request, reply, and confirmation messages. The offline synthesis generates an initial plan, while the online scheme dynamically adjusts it to accommodate multi-agent collaboration. Additionally, to address delays that may occur due to actions taking longer than anticipated, we propose a synchronization mechanism. This mechanism ensures that agents can initiate collaborative actions simultaneously, thus maintaining coordination despite the potential action delays introduced in the experimental scenario.
Furthermore, we design a Model Predictive Control (MPC) controller with collision avoidance capabilities to guarantee safe and efficient motion of agents within the workspace. Both the planning strategy and the MPC controller are implemented in Python and Robot Operating System 2 (ROS2) allowing deployment on a wide range of compatible robotic platforms.

To validate the effectiveness of our framework, we conduct extensive tests in various scenarios, including controlled simulations in ROS2 and practical field experiments involving a team of five robots, specifically 2 Turtlebots and 3 Hebi Rosies with manipulation capabilities. The results demonstrated a significant reduction in computational complexity compared to previous methods, and superior adaptability to various experimental setups.

Keywords

Multi-Rrobot Systems, Robot Collaboration, Linear Temporal Logic, ROS2 Implementation

Sommario

L'efficiente coordinamento e la collaborazione dei robot autonomi sono fondamentali per svolgere compiti complessi in diverse applicazioni industriali e di ricerca. In questo studio, ci concentriamo su compiti ricorrenti, tipici delle applicazioni agronomiche, dove questi devono essere eseguiti ripetutamente per un periodo indefinito. Specificheremo formalmente tali compiti attraverso Linear Temporal Logic (LTL) per la pianificazione e il coordinamento dei sistemi multi-agente. Sviluppiamo un pacchetto software per la pianificazione e il coordinamento multi-agente con un focus sull'agricoltura di precisione, che può essere esteso in modo semplice a un'ampia gamma di applicazioni di controllo multi-robot. L'obiettivo principale della tesi è sviluppare una strategia di pianificazione che consenta agli agenti di eseguire compiti in collaborazione, garantendo sia l'affidabilità sia la riduzione della complessità computazionale rispetto agli approcci precedenti.
A tal fine, adattiamo un approccio bottom-up per il coordinamento dei movimenti e dei compiti che include una sintesi offline del piano e uno schema di coordinamento online basato sullo scambio in tempo reale di messaggi di richiesta, risposta e conferma. La sintesi offline genera un piano iniziale, mentre lo schema online lo aggiorna dinamicamente per consentire la collaborazione multi-agente. Inoltre, per affrontare i ritardi che possono verificarsi a causa del fatto che le azioni richiedono più tempo del previsto, proponiamo un meccanismo di sincronizzazione. Questo meccanismo assicura che gli agenti possano iniziare azioni collaborative simultaneamente, mantenendo così il coordinamento nonostante i potenziali ritardi introdotti nello scenario sperimentale.
Inoltre, progettiamo un controllore Model Predictive Control (MPC) capace di evitare collisioni per garantire un movimento sicuro ed efficiente degli agenti all'interno dell'area di lavoro. Sia la strategia di pianificazione che il controllore MPC sono implementati in Python e Robot Operating System 2 (ROS2), consentendo l'impiego in un'ampia gamma di robot compatibili.
Per convalidare l'efficacia del nostro framework, conduciamo estesi test in vari scenari, incluse simulazioni controllate in ROS2 ed esperimenti pratici sul campo coinvolgendo un team di cinque robot, nello specifico 2 Turtlebot e 3 Hebi Rosie con capacità di manipolazione. I risultati hanno dimostrato una significativa riduzione della complessità computazionale rispetto ai metodi precedenti e una superiore adattabilità a diversi setup sperimentali.

Parole Chiave

Sistemi Multi-Robot, Collaborazione Robotica, Linear Temporal Logic, Implementazione in ROS2

Sammanfattning

Effektiv koordinering och samarbete av autonoma agenter är kritiskt för utförande av komplexa uppgifter i olika industriella och forskningsapplikationer. I denna studie fokuserar vi på återkommande uppgifter, typiska för agronomiska tillämpningar, där uppgifter behöver utföras upprepade gånger över en obestämd tidshorisont. Vi specificerar formellt sådana uppgifter med Linjär Temporal Logik (LTL) för planering och koordinering av multiagentsystem. Vi utvecklar ett mjukvarupaket för multiagentplanering och -koordinering med fokus på precisionsjordbruk, som enkelt kan utökas till ett brett spektrum av multi-robotregleringsapplikationer. Avhandlingens huvudmål är att utveckla en planeringsstrategi som möjliggör för agenter att utföra uppgifter i samarbete, vilket säkerställer både pålitlighet och minskad beräkningskomplexitet jämfört med tidigare tillvägagångssätt.
För detta ändamål anpassar oss vi ett bottom-up-tillvägagångssätt för rörelse- och uppgiftskoordinering som inkluderar en offline plansyntes och ett online koordineringsschema baserat på realtidsutbyte av begärnings-, svars- och bekräftelsemeddelanden. Offline syntesen genererar en initial plan medan onlineschemat dynamiskt justerar den för att möjliggöra multiagent samarbete. Dessutom, för att hantera potentiella fördröjningar som kan uppstå till följd av att handlingar tar längre tid än förväntat, föreslår vi en synkroniseringsmekanism. Denna mekanism säkerställer att agenter kan påbörja samarbetsåtgärder samtidigt, vilket upprätthåller koordinering trots potentiella fördröjningar som introduceras i det experimentella scenariot.
Dessutom designar vi en Model Predictive Control (MPC) regulator med kollisionsundvikande förmåga för att garantera säker och effektiv rörelse av agenter inom arbetsområdet. Både planeringsstrategin och MPC-regulatorn är implementerade i Python och Robot Operating System 2 (ROS2) vilket möjliggör distribution på ett brett utbud av kompatibla robotplattformar.
För att validera effektiviteten av vårt ramverk, utför vi omfattande tester i olika scenarion, inklusive kontrollerade simuleringar i ROS2 och praktiska fältexperiment bestående av ett team av fem robotar, specifikt 2 Turtlebots och 3 Hebi Rosies med manipulationsförmåga. Resultaten demonstrerar en signifikant reduktion av beräkningskomplexitet jämfört med tidigare metoder och överlägsen anpassningsförmåga till olika experimentupplägg.

Nyckelord

Multirobotsystem, Robot-samarbete, Linjär Temporal Logik, ROS2-implementering

Acknowledgments

First and foremost, I want to express my deepest gratitude to my supervisors, Lefteris and Victor. Their unwavering support, valuable insights, and constructive comments have been instrumental throughout the development of this work. Their guidance has not only improved the quality of this thesis but also enriched my academic journey.

A heartfelt thanks goes to my examiner here in Sweden, Professor Dimarogonas, whose expertise and thoughtful feedback have been invaluable. I am equally grateful to Professor Cenedese in Padova for providing me with this incredible opportunity. Their combined efforts have made this academic endeavor possible, and I truly appreciate their support.

A massive thanks to all my friends in Padova and the new ones I have made here at KTH. Your friendship and encouragement have made this experience unforgettable. A special thanks to Doğa for all our coffee breaks, they have been a great way to relax and have some fun.

To my family, thank you so much for your support during this new chapter of my life. Your encouragement has been a constant source of strength. I especially want to mention my Nonna Mima for always being there whenever I needed something. Her support has been incredibly comforting and gave me the ability to overcome every challenge I faced.

Thank you all for making this journey so enriching and memorable.

Stockholm, August 2024
Davide Peron

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS AND ABBREVIATIONS

AP	Atomic Proposition
CANOPIES	Collaborative Paradigm for Human Workers and Multi-Robot Teams in Precision Agriculture Systems
CBF	Control Barrier Function
FTS	Finite Transition System
KTH	KTH Royal Institute of Technology
LQR	Linear Quadratic Regulator
LTL	Linear Temporal Logic
MIP	Mixed Integer Programming
MoCap	Motion Capture
MPC	Model Predictive Control
NBA	Nondeterministic Büchi Automaton
PBA	Product Büchi Automaton
ROI	Region Of Interest
ROS2	Robot Operating System 2 [1]

sc-LTL	Syntactically Cosafe LTL
SML	Smart Mobility Lab
TS	Transition System
wFTS	weighted Finite Transition System

LIST OF SYMBOLS USED

The following symbols will be used throughout the thesis. This list defines only the basic symbols; in the thesis body, we will often use these symbols with superscripts to denote the specific agent to which they belong.

$\mathcal{A}_{\mathcal{P}}$	Generic PBA, see equation (2.7),	page 11
$\mathcal{B}$	Generic NBA, see equation (2.6),	page 10
2^{Ψ}	Power Set of Ψ,	page 7
α	An Extended class $\mathcal{K}$ function,	page 14
$\boldsymbol{u}$	Input of a dynamical system,	page 12
$\boldsymbol{x}$	State of a dynamical system,	page 12
$\Delta_d^{a_j}$	The performance metrics used to filter the agents for the MIP,	page 39
$\longrightarrow$	Transition relation in a TS, FTS or wFTS,	page 9
Confirm	The confirmation message, see equation (4.14),	page 37
Ready	The Ready message,	page 42
Reply	The Reply message, see equation (4.13),	page 34
Request	The Request message, see equation (4.12),	page 33
Start	The Start message,	page 43
$\mathcal{N}$	Set of Heterogeneous Agents,	page 26

$\mathcal{N}_F$	The filtered set of agents after Procedure 1,	page 39
$\mathcal{N}_H$	A feasible set of agents for the MIP,	page 38
$\mathcal{O}$	The set of obstacles,	page 44
$\mathcal{T}$	Generic TS, see equation (2.4),	page 9
$\mathcal{T}_\mathcal{G}$	The agent wFTS, see equation (4.6),	page 30
$\mathcal{T}_\mathcal{M}$	Motion wFTS,	page 28
$\mathcal{T}_w$	Generic wFTS, see equation (2.5),	page 9
Cond	Condition function of the action model $\mathscr{A}$,	page 29
Depd	Dependence function of the action model $\mathscr{A}$,	page 29
Dura	Duration function of the action model $\mathscr{A}$,	page 29
L	Labelling function in a TS, FTS or wFTS,	page 9
T	Weight function in a wFTS,	page 10
$\mathscr{A}$	The non-movement related action model, see equation (4.5),	page 29
$\overline{T}$	The time before a planned collaboration,	page 35
Π	Set of states in a TS, FTS or wFTS,	page 9
Π_0	Set of initial states in a TS, FTS or wFTS,	page 9
$\Pi_\mathcal{M}$	Set of ROIs,	page 27
π_d	The d-th generic ROI,	page 33
π_d	The d-th generic ROI,	page 33
Ψ	Set of Atomic Propositions,	page 6
ρ	The sequence of actions associated with τ, see equation (4.8),	page 31
Σ	Set of actions,	page 9
σ_0	The *None* actiona,	page 29
Σ_a	Set of *active* actions, see equation (4.3),	page 28

Σ_c	Set of *collaborative* actions, ... page 28
σ_d	Generic agent action, ... page 29
σ_d	The d-th generic action, ... page 33
σ_d	The d-th generic action, ... page 33
Σ_h	Set of *assisitng* actions, ... page 28
Σ_l	Set of *local* actions, ... page 28
τ	The path in a wFTS, see equation (4.7), ... page 31
τ_{init}	The initial plan of an agent, see equation (4.11), ... page 33
φ	Generic LTL formula, see equation (2.1), ... page 7
a_i, a_j	The i-th and j-th generic agents of $\mathcal{N}$, ... page 26
b_d	The boolean variable indicating the ability to execute σ_d, ... page 34
$c_d^{a_j}$	The optimization variable for the MIP, ... page 37
D	The Detour dictionary, ... page 35
H	The request horizon, ... page 33
$h(\boldsymbol{x})$	A Control Barrier Function, ... page 14
o_i	The generic i-th obstacle, see equation (4.21), ... page 44
R	The radius of obstacles agents or ROI, see equation (4.21), ... page 44
T_c	The time before a collaborative action is bound to happen, ... page 33
t_d	The time required before the agent is able to perform σ_d, ... page 35
T_{rem}	The time remaining before the end of an action, ... page 34

CHAPTER 1 INTRODUCTION

Over the last 50 years, autonomous robots have become integrated into various sectors of society, including the medical field [2], delivery services [3], warehouse management [4], and interplanetary exploration [5]. One significant area where autonomous robots are gaining importance is the agricultural industry. The labor-intensive tasks traditionally performed by farmers since ancient times can now be accomplished using autonomous robots. This transition is thoroughly reviewed in a recent study, which highlights various applications developed to automate this sector [6]. The Collaborative Paradigm for Human Workers and Multi-Robot Teams in Precision Agriculture Systems (CANOPIES) project [7], which this work is part of, specifically focuses on the automation of table-grape vineyard harvesting. CANOPIES's aim is to develop a novel collaborative human-robot paradigm in the field of precision agriculture for permanent crops where farmworkers can efficiently work together with teams of robots to perform agronomic interventions. To reach this goal, among other objectives, it is necessary to devise a high-level plan, *i.e.,* a set of instructions, that the robots have to follow to complete their assigned tasks. Within the project, this is done through the use of the Linear Temporal Logic (LTL) formalism, which, combined with the use of Nondeterministic Büchi Automaton (NBA), can lead to the creation of plans that satisfy specific tasks assigned to the robotic agents. The focus of this work is the development of an algorithm that will allow robotic agents to collaborate to complete complex tasks that would otherwise be unfeasible for a single agent. One of the main requirements is that it needs to integrate within the framework of CANOPIES, so the algorithm must be compatible with LTL specifications, including the ability to perform recurring tasks *i.e.,* tasks that need to be executed repeatedly over an indefinite horizon. Additionally, we aim to develop a setup that can be scaled, minimizing overall computational complexity as much as possible.

This work not only contributes to the field of robotics but also has significant implications for the future of agriculture, potentially transforming traditional farming practices and increasing productivity in a sustainable manner.

AGROFOOD MAKER
Maker Faire
INTELLIGENZA ARTIFICIALE E ROBOTICA PER AGRICOLTURA DI PRECISIONE NEL LAZIO

Figure 1.1: CANOPIES's stand at Maker Faire, Rome, 2023

1.1 Background

The field of multi-robot systems has seen substantial advancements, driven by the need for efficiency and automation in various industries. In particular, the agricultural sector has greatly benefited from these advancements, with autonomous robots being increasingly used to perform labor-intensive tasks that were traditionally done manually. These robots can now handle complex tasks, such as harvesting, planting, and monitoring crops, with greater precision and efficiency.

One of the key methods enabling these advancements is LTL, which provides a formal framework for specifying the behavior of robots over time. LTL allows for the definition of complex tasks. Using Transition System (TS) to model the states and actions of the robots is another popular paradigm used in this field, this allows the LTL to define tasks based on the states of the robots, thus giving the ability to specify high-level tasks.

Model Predictive Control (MPC) is another important paradigm that enhances the capabilities of multi-robot systems. MPC allows robots to make real-time decisions based on predictions of future states, ensuring that they can adapt to changes in their environment and avoid collisions. By integrating MPC with LTL-based planning, robots can be equipped with the ability to handle dynamic and unpredictable environments while still following their high-level plans.

In summary, the combination of LTL, TS, and MPC provides a powerful toolkit for developing advanced multi-robot systems capable of performing complex tasks. This thesis aims to leverage these methods to create an algorithm that enables efficient collaboration among robots, particularly in the context of the CANOPIES project. By developing such an algorithm, we aim to enhance the productivity and sustainability of agricultural practices, paving the way for more widespread adoption of autonomous robots in this critical industry.

1.2 Problem

This subsection serves as an overview of the main challenges that this work aims to tackle. First, we will provide a general definition of the problem we are trying to address, and then we will mention the engineering issues that arise from this problem.

1.2.1 Problem Definition

The main problem addressed in this thesis is the design of an algorithm that facilitates multi-agent coordination, where each agent must satisfy their locally assigned recurring LTL task. This involves ensuring that each robot can execute collaborative actions requiring more than one agent to be completed. The algorithm must dynamically adjust the plans of individual robots to accommodate collaborative actions without violating their LTL constraints. Moreover, it must handle real-time synchronization among robots to prevent conflicts and ensure seamless operation. The overarching objective is to create a system where robots can efficiently and reliably perform complex, interdependent tasks in a dynamic and not fully known environment.

A more detailed and rigorous mathematical formulation of the problem statement will be given in Chapter 4 once all the necessary background information has been covered.

1.2.2 Scientific and Engineering Issues

The key scientific and engineering issues involved in this work include:

- Minimizing computational complexity to enable implementation on commercially available hardware.
- Integrating the algorithm within the existing framework of the CANOPIES project to ensure compatibility with LTL specifications and recurring tasks.
- Ensuring the scalability and robustness of the algorithm to handle larger systems and real-world discrepancies.
- Validating the algorithm through both simulations and real-world experiments

1.3 Purpose

The purpose of this thesis is to develop and validate an advanced algorithm for effective multi-agent collaboration in dynamic environments. The algorithm aims to ensure safety and reliability by adapting to real-time changes and synchronizing robots to guarantee the execution of collaborative actions. Addressing the ethical implications of automation, this research promotes the development of systems that are both technologically advanced and socially responsible. The algorithm will be implemented and validated within the Robot Operating System 2 (ROS2) through simulations and real-world experiments, aiming to enhance the efficiency and reliability of multi-robot systems and contribute significantly to the fields of robotics and automation.

Achieving these goals will benefit several stakeholders. Industrial sectors that rely on automation will see improved efficiency and safety in their operations. Human operators will benefit from enhanced safety measures and the ability to work alongside advanced robotic systems, reducing their physical workload while maintaining job relevance. Researchers and developers in robotics will gain a validated algorithm that can be further refined and applied in various contexts. However, it is crucial to address anticipated ethical issues such as balancing automation with job retention for human workers and ensuring that robots do not lead to significant job displacement. Sustainability concerns focus on the efficient use of resources and energy in automated systems, while social issues revolve around enhancing human-robot collaboration without compromising human safety or ethical standards. This research aims to promote responsible and sustainable practices in the development and deployment of multi-agent systems.

1.4 Goals

This project aims to develop an algorithm that allows multi-robot collaboration while adhering to the specific LTL formulas assigned to each agent. The project objectives have been organized into the following subgoals:

1. Familiarize with the theoretical foundations of Linear Temporal Logic (LTL), TS, Nondeterministic Büchi Automaton (NBA), and Model Predictive Control (MPC).

2. Review existing literature on multi-agent collaboration protocols that incorporate LTL specifications.

3. Develop the multi-robot collaboration algorithm and provide formal proof of its correctness.

4. Familiarize with the planner developed at Smart Mobility Lab (SML) at KTH Royal Institute of Technology (KTH) and adapt it to Robot Operating System 2 [1] (ROS2) to ensure future compatibility.

5. Implement the algorithm within the ROS2 environment, integrating it with the existing software at SML. This step also includes the integration of the MPC controller with collision avoidance to manage the movement of the available robots at SML.

6. Conduct tests using ad-hoc simulations designed to represent typical scenarios to assess the scalability of the protocol. Additionally, conduct field experiments at SML to evaluate the algorithm's performance on actual hardware.

1.5 Research Methodology

The initial phase of the research will focus on a review of existing algorithms that can allow multi-agent collaboration under LTL specifications. Once a suitable baseline algorithm has been found, the next step will involve its modification and further development to align with the overarching objectives of this thesis. Additionally, any segment of the algorithm requiring proven and effective existing tools will incorporate these, taking into consideration their complexity and potential issues that could affect the overall performance of the final algorithm.

Mathematical proofs will be used throughout the thesis to assess key properties and characteristics of the algorithm, all of these properties will then be used to ensure its correctness. Afterward, ad-hoc local simulations, implemented using ROS2 and Python, will be conducted to test the algorithm's performance under nominal scenarios. Lastly, field experiments will be performed on actual hardware at SML to assess the algorithm's real-world applicability and effectiveness, fulfilling the thesis's goal of addressing real-world challenges.

1.6 Delimitations

Due to time limitations, the algorithm must be integrated with the existing planner developed at SML, some compromises expressed in terms of assumptions will be formulated in the following. These compromises are expected to constrain specific behaviors that would otherwise be problematic with the currently available software. Moreover, during the experimental phase, we will focus only on the movement of the robots, omitting the execution of other actions. This is due to the prohibitive time requirements needed to implement all necessary tasks within the strict project timeline; however, this is not expected to compromise the evaluation of the algorithm's effectiveness. Finally, the development of the MPC controller will be limited to the robots selected for the experiments, although the framework and algorithm can be extended to any robotic agent compatible with ROS2.

1.7 Structure of the thesis

Chapter 2 presents relevant background information about LTL, TS, NBA, MPC, and Mutli-robot collaboration under LTL specification. Chapter 3 presents the methodology and methods used to solve the problem. Chapter 4 describes the setup we considered and the algorithm developed to solve the problem. Chapter 5 shows and studies the results of the simulations and experiments. Chapter 6 concludes the thesis and expands it with extra information about the future research directions that may be available for this work.

CHAPTER 2 BACKGROUND

This chapter provides basic background information on Linear Temporal Logic (LTL), and its integration with Nondeterministic Büchi Automaton (NBA) and Transition System (TS) to provide the agents with a plan that satisfies the assigned formula. Additionally, we introduce the Model Predictive Control (MPC) framework, used as the low-level controller for agent movement. We briefly address Control Barrier Functions (CBFs), implemented as constraints within the controller. Lastly, this chapter reviews related work on multi-agent collaboration under LTL specification.

2.1 Linear Temporal Logic

This section introduces Linear Temporal Logic (LTL), a logical formalism suitable for defining linear-time properties and hence capable of specifying complex system properties and tasks. LTL plays a fundamental role in formal verification methodologies, particularly in the context of computer systems and concurrent processes. Additionally, it can be used to describe complex planning objectives for autonomous robots. LTL enables the specification of temporal behaviors, allowing one to describe how properties evolve and persist throughout system executions. It provides a simple yet mathematically rigorous syntax for expressing characteristics regarding the relationships between state labels in executions.
Although the term ”temporal” implies a relationship with the system’s real-time behavior, this is only true in an abstract sense since LTL allows for the specification of event order. Furthermore, the linear sequence of time instants considered is discrete, meaning that, the present moment refers to the current state, and the next moment corresponds to the immediate successor state.

2.1.1 LTL Syntax

The first step to rigorously define LTL formulas is to identify and define the main components that constitute them. The basic components are Atomic Propositions (APs) which are Boolean variables that can either be true or false. We define the set of atomic propositions as Ψ, where

if $a \in \Psi$ then a is an atomic proposition.
Next, we define the boolean constant $\top \triangleq True$, and the boolean connectors: negation ($\neg$), conjunction ($\wedge$), and the two temporal operators, *next* ($\bigcirc$) and *until* (U).
With these key elements defined, we can now provide a formal definition of LTL formulas:

Definition 2.1 ([8])**.** LTL formulas defined over the set Ψ of atomic propositions can be obtained according to the following syntax:

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \tag{2.1}$$

where $a \in \Psi$ and φ,φ_1,φ_2 are LTL formulas.

Based on the syntax of Definition 2.1, we can define the following operators:

- *Disjunction* ($\vee$): $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$.
- *Implication* ($\rightarrow$): $\varphi_1 \rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$.
- *Eventually* ($\lozenge$): $\lozenge\varphi := \top\mathsf{U}\varphi$.
- *Always* ($\square$): $\square\varphi := \neg\lozenge\neg\varphi$.

Lastly, we can define the "*infinitely often* φ" behavior, which will be referred to as a recurring task or behavior, as:

$$\square\lozenge\varphi \tag{2.2}$$

This recurring task indicates that there always exists a future moment i such that for any moment j, where $i \geq j$, a φ-sate is visited.

2.1.2 LTL Semantics

LTL formulas represents path properties, which means a path can either satisfy an LTL formula or not. To determine if a path satisfies an LTL formula, first, we define the semantics of an LTL formula φ defined as a language $Words(\varphi)$ that includes all infinite words over the alphabet 2^Ψ that satisfy φ, where 2^Ψ is the power set of Ψ . This implies that each LTL formula is associated with a single linear-time property.

Definition 2.2 (**Semantics of LTL** [8])**.** Let φ be an LTL formula over Ψ. The linear-time property induced by φ is

$$Words(\varphi) = \{\delta \in (2^\Psi)^\omega \mid \delta \models \varphi\} \tag{2.3}$$

Where $(2^\Psi)^\omega$ denotes the set of words that arise from the infinite concatenation of words in 2^Ψ and the satisfaction relation $\models \subseteq (2^\Psi)^\omega \times$LTL is the smallest relations with the following properties:

$$
\begin{array}{ll}
\sigma \models \top & \\
\sigma \models a & \text{iff } a \in A_0 \ (\textit{i.e.,}\ A_0 \models a) \\
\sigma \models \varphi_1 \wedge \varphi_2 & \text{iff } \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi & \text{iff } \sigma \not\models \varphi \\
\sigma \models \bigcirc\varphi & \text{iff } \sigma[1\ldots] = A_1A_2A_3\ldots \models \varphi \\
\sigma \models \varphi_1 \mathsf{U} \varphi_2 & \text{iff } \exists j \geq 0.\ \sigma[j\ldots] \models \varphi_2 \text{ and } \sigma[i\ldots] \models \varphi_1,\ \forall 0 \leq i < j
\end{array}
$$

Where for $\sigma = A_0A_1A_2\ldots \in (2^{\Psi})^{\omega}$, $\sigma[j\ldots] = A_jA_{j+1}A_{j+2}\ldots$ is the suffix of σ starting in the $(j+1)$st symbol A_j. Extending those concepts to an interpretation over paths and states, the LTL formula φ holds in state s if all paths starting in s satisfy φ.

To illustrate the semantics of LTL, consider the set of atomic propositions $\Psi = \{p, q, r\}$. The power set 2^{Ψ} includes all possible subsets of Ψ, representing different combinations of propositions that can be true at any state.
Consider the LTL formula $\varphi = \Diamond(p \wedge \bigcirc q)$, which reads as "eventually, p holds and in the next state q holds". This formula defines a language $Words(\varphi)$ that includes all infinite words satisfying this property.
Let's examine the infinite words γ and τ to check if they satisfy φ:

- $\gamma = \{\{r\}, \{p, q\}, \{q\}, \{\}, \{p\}, \{q\}, \{\}, \ldots\}$: At position 1, we have $\{p, q\}$ so both p and q hold, and at position 2, we have $\{q\}$, so q holds. This means there is a state where p holds and in the next one q holds.

- $\tau = \{\{r\}, \{p\}, \{r\}, \{\}, \{q\}, \{p\}, \{\}, \ldots\}$: At position 1, we have $\{p\}$, so p holds, but at position 2, we have $\{r\}$, so q does not hold. Similarly, at position 5, we have $\{p\}$, so p holds, but at position 6, we have $\{\}$, so q does not hold. There is no state where p holds and in the next one q holds.

Since γ contains a state where p is followed by q, it satisfies $\Diamond(p \wedge \bigcirc q)$, so $\gamma \in Words(\varphi)$. Conversely, τ does not meet this condition, so $\tau \notin Words(\varphi)$.
For comprehensive information on LTL syntax and semantics refer to [8, Ch. 5.1]

2.2 Transition System

Transition Systems (TSs) are used in computer science as models to describe the behavior of systems. They can be represented as directed graphs where nodes represent states, and edges represent transitions between the states. A state describes some information about a system at a certain moment of its behavior while transitions specify how the system can evolve from one state to another. In this work, we will consider APs as states, while we will consider action labels as transitions, a more detailed explanation of the transition system used will be given in Section 4.1, for now we will limit to the definition the basic concepts.

Definition 2.3 ([8])**.** A Transition System (TS) is defined as the tuple:

$$\mathcal{T}\,(\Pi, \Pi_0, \Psi, \Sigma, \longrightarrow, \mathrm{L}) \tag{2.4}$$

where:

- Π is a set of states,
- $\Pi_0 \subseteq \Pi$ is a set of initial states,
- Ψ is a set of atomic propositions,
- Σ is a set of actions,
- $\longrightarrow \subseteq \Pi \times \Sigma \times \Pi$ is a transition relation,
- $\mathrm{L} : \Pi \to 2^{\Psi}$ is the labelling function.

A TS such that Π, Ψ, Σ are finite is called Finite Transition System (FTS).

To simplify notation, we use $\pi \xrightarrow{\sigma} \pi'$ instead of $(\pi,\ \sigma,\ \pi') \in \longrightarrow$. The operational behavior of a transition system can be described as follows: The system begins in an initial state $\pi_0 \in \Pi_0$ and evolves according to the transition relation $\longrightarrow$. Specifically, if the current state is π, a transition $\pi \xrightarrow{\sigma} \pi'$ is deterministically chosen, indicating that the action σ is performed, and the system transitions from state π to state π'. This process repeats in state π' and continues until a state with no outgoing transitions is encountered.

Remark 2.1. Π_0 might be empty; in such a case, the transition system exhibits no behavior as there is no initial state to select.

The labeling function L assigns a set $\mathrm{L}(\pi) \in 2^{\Psi}$ of APs to each state π. $\mathrm{L}(\pi)$ represents the specific APs $\psi \in \Psi$ that hold true in state π. If φ is a LTL formula, then π satisfies φ if the interpretation provided by $\mathrm{L}(\pi)$ makes φ true, denoted as: $\pi \models \varphi$ iff $\mathrm{L}(\pi) \models \varphi$.

Given this formal definition of a TS, we will subsequently consider a modified version known as a weighted Finite Transition System (wFTS), which is defined as follows. In the rest of this thesis, any reference to a Finite Transition System (FTS) or TS will implicitly refer to this finite and weighted version (wFTS).

Definition 2.4 ([8])**.** A weighted Finite Transition System (wFTS) is defined as the tuple:

$$\mathcal{T}_w = (\Pi, \Pi_0, \Psi, \Sigma, \longrightarrow, \mathrm{L}, \mathrm{T}) \tag{2.5}$$

Where:

- Π is a finite set of states,

- $\Pi_0 \subseteq \Pi$ is a finite set of initial states,
- Ψ is a finite set of atomic propositions,
- Σ is a finite set of actions,
- $\longrightarrow \subseteq \Pi \times \Sigma \times \Pi$ is a transition relation,
- $\mathrm{L} : \Pi \rightarrow 2^{\Psi}$ is the labelling function,
- $\mathrm{T} : \Pi \times \Sigma \times \Pi \rightarrow \mathbb{R}^{+}$ is the weight function that represents the cost of transitions in $\longrightarrow$

2.3 Nondeterministic Büchi Automaton

Definition 2.5 ([9])**.** A Nondeterministic Büchi Automaton (NBA) is defined as the tuple:

$$\mathcal{B} = \left(S, S_0, 2^{\Psi}, \delta, \mathcal{F}\right) \tag{2.6}$$

Where:

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- 2^{Ψ} is the input alphabet,
- $\delta : S \times 2^{\Psi} \rightarrow 2^{S}$ is the transition function,
- $\mathcal{F} \subseteq S$ is the set of accepting states.

An infinite run **s** of a NBA is an infinite sequence of states $\mathbf{s} = s_0 s_1 \ldots$ generated by an infinite sequence of input alphabets $\sigma = \sigma_0 \sigma_1 \ldots \in \left(2^{\Psi}\right)^{\omega}$, where $s_0 \in S_0$ and $s_{k+1} \in \delta\left(s_k, \sigma_k\right), \forall k \geq 0$. An infinite run **s** is accepted by $\mathcal{A}_{\varphi}$ if and only Inf$(q) \cap \mathcal{F} \neq \emptyset$, where Inf$(s)$ is the set of states that appear in **s** infinitely often.

Remark 2.2. We use a NBA because it allows multiple transitions for a given state and input, accepting an infinite word if at least one path visits an accepting state infinitely often. In contrast, a Deterministic Büchi Automaton has exactly one transition for each state and input pair. It accepts an infinite word if the single path it follows visits an accepting state infinitely often.

2.3.1 LTL Planning

In this subsection, the initial planning for the system will be explained. This is part of the original software developed at SML, and all the details can be found in [10].
The planner node takes as input an LTL task φ comprised of two formulas φ^{hard} and φ^{soft}, and a wFTS $\mathcal{T}_w = (\Pi, \Pi_0, \Psi, \Sigma, \longrightarrow, \mathrm{L}, \mathrm{T})$. These two LTL formulas define the high-level task: the hard task φ^{hard} specifies requirements that must be strictly satisfied, while the soft task φ^{soft} includes optional objectives that can be violated if they conflict with the hard task.

Remark 2.3. Even though this section considers both hard and soft tasks, it is important to note that in future chapters, when we refer to an LTL task, we will always consider φ^{hard} and set $\varphi^{soft} = \top$.

First, the hard and soft LTL formulas φ^{hard} and φ^{soft} are used to generate the respective NBAs $\mathcal{B}_{\varphi^{hard}}$ and $\mathcal{B}_{\varphi^{soft}}$ via the LTL2BA software [11]. Then, the planner generates a combined NBA $\mathcal{B}_{\varphi} := \mathcal{B}_{\varphi^{hard}} \times \mathcal{B}_{\varphi^{soft}} = \left(S, S_0, 2^{\Psi}, \delta, \mathcal{F}\right)$ using the safety ensured product automaton developed in [12]. Afterwards, building upon the work of [13], a Product Büchi Automaton (PBA) is defined as follows:

Definition 2.6 ([10])**.** A Product Büchi Automaton (PBA) is defined as the tuple:

$$\mathcal{A}_{\mathcal{P}} = \mathcal{B}_{\varphi} \otimes \mathcal{T}_w = (\mathcal{S}_{\mathcal{P}}, \mathcal{S}_{\mathcal{P},0}, \delta_{\mathcal{P}}, \mathcal{F}_{\mathcal{P}}, \mathcal{W}_{\mathcal{P}}) \tag{2.7}$$

Where:

- $\mathcal{S}_{\mathcal{P}} = S \times \Pi$ is a finite set of states,
- $\mathcal{S}_{\mathcal{P},0} = S_0 \times \Pi_0$ is the set of initial states,
- $\delta_{\mathcal{P}}$ is the transition relation defined as follows $(\langle s, \pi\rangle, \langle s', \pi'\rangle) \in \delta_{\mathcal{P}}$ iff $\exists \sigma \in \delta, s' \in \delta(s, \sigma)$ and $\exists \sigma_a \in \Sigma, (\pi, \sigma_a, \pi') \in \longrightarrow$,
- $\mathcal{F}_{\mathcal{P}} = (F \times \Pi)$ is the set of accepting states,
- $W_{\mathcal{P}} : \delta_{\mathcal{P}} \to \mathbb{R}^+, W_{\mathcal{P}}\left(\langle s, \pi\rangle, \langle s', \pi'\rangle\right) = W_P\left(\pi, \pi'\right)$ is the weight function.

From this PBA, we can determine an optimal run and map it back to the wFTS $\mathcal{T}_w$ using model-checking techniques [14]. Accepting runs exhibit a prefix-suffix structure given by: $r_{\mathcal{P}} = p_0, p_1 \cdots p_k \left(p_{k+1} \cdots p_n p_k\right)^{\omega}$, where $p_0 \in \mathcal{S}_{\mathcal{P},0}$ and $p_k \in \mathcal{F}_{\mathcal{P}}$. The output word comprises two segments: a finite prefix executed once, transitioning from the initial state p_0 to an accepting state p_k, and an infinitely repeating suffix looping from p_k back to itself [14]. Furthermore, the optimal accepting run is one that minimizes a cost function defined by transition weights. This optimal run also has a corresponding input word, *i.e.,* an action sequence for the agent to follow to meet its specifications. This plan, like the run, adheres to the prefix-suffix structure. For comprehensive implementation details, refer to [10].

2.4 Model Predictive Control

One of the most exciting results in the control theory field was the Linear Quadratic Regulator (LQR), developed in the 1960s, that offered an optimal solution for linear time-invariant multi-input multi-output systems. This solution featured several desirable properties: it was unique, asymptotically stable, and came with an easily calculable formula for the feedback gain. However, LQR has significant limitations in the presence of nonlinearities or when constraints are imposed, which are common in practical scenarios.

In the late 1970s and early 1980s, driven by the industrial need for more efficient and robust control strategies, algorithms that could predict future system behavior and optimize control actions began to emerge. This development laid the foundation for Model Predictive Control (MPC). MPC uses a model to predict future outputs and solves an optimization problem at each time step to determine the best control action for that specific instant.

Before delving into the rigorous mathematical formulation of MPC, it's helpful to understand its basic principles. MPC is centered around the concept of receding horizon control, which involves solving a finite horizon optimization problem at each time step. The goal is to minimize a cost function subject to system dynamics and constraints. The main steps of MPC are as follows:

1. **Model Development**: Create a mathematical model of the process, which can be nonlinear, to predict its behavior.

2. **Prediction**: Use the developed model to predict future outputs over a predefined prediction horizon based on the current state and a sequence of future inputs.

3. **Optimization**: Formulate an optimization problem using a cost function that typically includes terms for tracking performance and control effort, subject to constraints on inputs and outputs.

4. **Implementation**: After solving the optimization problem, apply only the first input to the process. Repeat the optimization step at the next time instant using updated measurements.

One of the primary advantages of MPC is its ability to handle nonlinear dynamics and constraints, adapting to various unplanned situations while providing high-performance control. Additionally, the receding horizon principle integrates feedback into the control action, ensuring continuous supervision of the system's current state.
However, implementing MPC poses several challenges. The optimization problem must be solved in real-time with the available hardware moreover stability and robustness are not inherently guaranteed. The optimization problem may also become infeasible at some future time step, meaning a plan that meets all constraints might not exist, thus persistent feasibility is not assured.

2.4.1 Mathematical Formulation

This subsection will be dedicated to the formal mathematical description of the optimization problem related to MPC. Consider the generic nonlinear system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ where $\boldsymbol{x} \in \mathcal{X}$ is the state and $\boldsymbol{u} \in \mathcal{U}$ is the input. Now we consider sampling time T we can obtain the Euler discretization of the system which will be defined at the generic instant in time k as $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + f(\boldsymbol{x}_k, \boldsymbol{u}_k) \cdot T$. Moreover, we define the prediction horizon N, and we will call the sequence of output over which we optimize as $U = \{\boldsymbol{u}_k, \boldsymbol{u}_{k+1}, \dots, \boldsymbol{u}_{k+N-1}\}$. Lastly, we define the stage cost function at time k as $q(\boldsymbol{x}_k, \boldsymbol{u}_k)$ and the terminal cost function as $p(\boldsymbol{x}_{k+N})$. we have now all the elements to give a general definition of the optimization problem according to [15]:

$$
\begin{aligned}
\min_{U} \quad & \sum_{i=0}^{N-1} [q(\boldsymbol{x}_{k+i}, \boldsymbol{u}_{k+i})] + p(\boldsymbol{x}_{k+N}) \\
\text{subject to:} \quad & \\
& \boldsymbol{x}_{k+i+1} = \boldsymbol{x}_{k+i} + f(\boldsymbol{x}_{k+i}, \boldsymbol{u}_{k+i}) \cdot T, \quad i = 0, \dots, N-1 \\
& \boldsymbol{x}_{k+i} \in \mathcal{X}, \quad i = 1, \dots, N \\
& \boldsymbol{u}_{k+i} \in \mathcal{U}, \quad i = 0, \dots, N-1 \\
& \boldsymbol{x}_k \text{ given} \\
& \text{other constraints}
\end{aligned}
\tag{2.8}
$$

It is important to note that different parameters can be used to tune the controller. Firstly, the sampling time T and the prediction horizon N are critical parameters that act as a tradeoff between the controller's performance and computational cost. Using a low value of T and a high value of N can improve the controller's accuracy, but this comes at the expense of increased computational cost. Therefore, the optimal values for these parameters are application-dependent and cannot be predetermined. Additionally, we have some degrees of freedom in tuning the controller by selecting the most appropriate cost function for the specific application.

2.4.2 Control Barrier Functions

Control Barrier Functions (CBFs) are used in control theory to ensure the safety of dynamic systems. They define a barrier in the state space that the system's trajectories should not cross, effectively acting as constraints. By incorporating CBFs into the control design *e.g.,* in MPC, one can ensure that the system avoids unsafe states while achieving desired behaviors. CBFs are used as constraints in optimization problems to generate control inputs that keep the system within safe operational limits.
In the following, we will provide theoretical insights required to understand how to use CBFs as constraints for the MPC optimization problem defined in Equation (2.8).

Firstly it is necessary to define the concept of safety set $\mathscr{C}$, of which a geometrical interpretation is shown Figure 2.1. This represents a specific constraint of our design defined by a differentiable function $h(\boldsymbol{x})$ that has to always be non-negative in order to consider the problem to be safe. This set can be expressed as follows:

$$\mathscr{C} = \{\boldsymbol{x} \in \mathbb{R}^n \ : \ h(\boldsymbol{x}) \geq 0\} \tag{2.9}$$

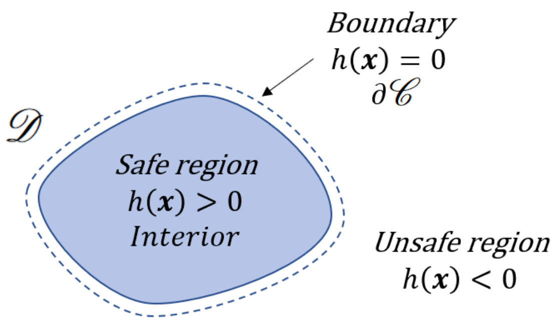

Figure 2.1: Geometric representation of the safe region $\mathscr{C}$

Definition 2.7 (Lipschitz). a function f such that

$$\mid f(x) - f(y) \mid \leq C \mid x - y \mid \tag{2.10}$$

for all x and y, where C is a constant independent of x and y, is called a Lipschitz function. For example, any function with a bounded first derivative must be Lipschitz.

Definition 2.8 (Extended class $\mathcal{K}$ function). A continuous function $\alpha \ : \ [0, a) \to [0, \infty)$ that is strictly increasing with $\alpha(0) = 0$

Definition 2.9 (CBF). Let set $\mathscr{C}$ be defined by Equation (2.9). $h(\boldsymbol{x})$ is a Control Barrier Function (CBF) for the system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}$ if there exists a locally Lipschitz Extended class $\mathcal{K}$ function α such that:

$$\sup_{\boldsymbol{u}\in\mathbb{R}^m} [L_f h(\boldsymbol{x}) + L_g h(\boldsymbol{x})\boldsymbol{u} + \alpha(h(\boldsymbol{x}))] \geq 0, \ \forall \boldsymbol{x} \in \mathscr{D} \subset \mathscr{C} \tag{2.11}$$

where $L_f h = \nabla h^T f(\boldsymbol{x}) \in \mathbb{R}$ and $L_g h = \nabla h^T g(\boldsymbol{x}) \in \mathbb{R}^{1\times m}$ are the Lie derivatives for the system and the operator $\nabla \ : \ \mathcal{C}^1(\mathbb{R}^n) \to \mathbb{R}^n$ is defined as the gradient $\dfrac{\partial}{\partial \boldsymbol{x}}$ of a scalar-valued differentiable function with respect to $\boldsymbol{x}$

Given a CBF $h(\boldsymbol{x})$ defined according to Definition 2.9, a valid constraint to be added to the MPC can be written as follows:

$$L_f h(\boldsymbol{x}) + L_g h(\boldsymbol{x})\boldsymbol{u} + \alpha(h(\boldsymbol{x})) \geq 0 \tag{2.12}$$

Without going into the details, it has been proven by [16] and [17], under mild assumptions that are always satisfied in this work, that if the system starts within the safety set $\mathscr{C}$, it will remain inside it. Conversely, if the initial condition is outside the set, the trajectory will asymptotically converge to the interior of the set.

2.5 Multi-agent Collaboration under LTL Specification

Multi-agent collaboration under LTL specifications is an active area of research that focuses on enabling multiple agents to work together to achieve complex tasks while adhering to temporal and logical constraints. This section explores various methodologies and algorithms developed to achieve such collaboration, ensuring that agents can efficiently and effectively meet specified goals. In this review, we will mostly focus on *bottom-up* approaches since we aim to achieve a decentralized structure, which is generally not possible with a *top-down* approach that typically requires a central monitoring unit for plan synthesis and execution.

The first significant contribution in this domain we will analyze has been proposed by Tumova *et al.,* in [18]. This work presents a *bottom-up* approach that incorporates strategies to mitigate the state-space explosion phenomenon, which is typical of this approach rendering it useless for real-world scenarios due to the high computational complexity. Each agent is assigned local tasks specified by LTL formulas, which define complex, high-level goals and can include requests for collaboration from other agents. Each agent is modeled as a FTS, and the workspace is divided into regions, represented by states in the FTS.
The *bottom-up* approach begins with a decomposition into finite horizons, meaning the planning problem is broken down into finite horizon planning problems that are solved iteratively. Additionally, an event-based synchronization mechanism is implemented, allowing efficient adaptation to varying durations of agents' discrete steps. These two solutions aim to address the state-space explosion problem.
During task execution and synchronization, each agent can request synchronization when needed for collaborative actions. This decomposes the problem of finding collective team behavior into several subproblems, ensuring that each agent's task is fulfilled from its perspective. This approach reduces computational complexity and avoids unnecessary synchronization, thereby ensuring efficient multi-agent planning and task execution.

The second contribution we will analyze has been proposed by Schuppe *et al.,* in [19]. This work also employs a *bottom-up* approach while addressing the state-space explosion phenomenon, with the main innovation being the use of game theory to solve the problem.
Each agent is assigned goals specified by LTL formulas, which may require cooperation among agents. Agents' behaviors are modeled as TSs, with states representing different conditions. The workspace is divided into regions, each represented by states in the TS. To synthesize the plan, each agent computes its strategy in isolation, making necessary assumptions about the states of other agents. These assumptions are resolved when composing individual strategies to ensure overall compliance. To achieve this, each agent's TS is combined with an NBA, incorporating assumptions about other agents. The problem is then reduced to safety games, which are easier to solve. These games ensure that the composed strategies meet the specifications. This algorithm effectively synthesizes multi-agent strategies by combining individual solutions and resolving inter-agent dependencies, ensuring scalability and robustness in dynamic environments.

Lastly, we will analyze the work of Guo *et al.,* proposed in [20]. The proposed system considers a loosely coupled framework, where agents are assigned local tasks specified by

Syntactically Cosafe LTL (sc-LTL) formulas [21], ensuring that tasks are finite. The authors assume that agents can navigate a workspace divided into regions, also obstacles are modeled as regions, and structured by an underlying grid. This motion can be abstracted using a wFTS. Communication between the robots is facilitated through a two-layer system: a static backbone network for group coordinators and a dynamic local network for direct neighbor communication. The action model developed includes local actions (independent), collaborative actions (requiring assistance), and assisting actions (helping others).
Initially, an offline process creates motion and action plans for each agent using model-checking algorithms to ensure LTL task satisfaction. If an agent identifies a collaborative action within a predefined horizon of its plan, it initiates a Request, Reply, and Confirmation cycle. The agent requests the necessary assisting actions from its neighbors, who respond based on their capabilities. Upon receiving all responses, a confirmation algorithm selects the best-suited agents for the required assisting actions, ensuring one agent per action. If a request is unmet, it is escalated to group coordinators, who seek assistance from other groups. Agents adjust their plans in real-time based on confirmations, incorporating confirmed collaborations, and if necessary, delaying actions and revising plans.

2.6 Summary

In this section, we have covered the key theoretical foundations relevant to this work. Additionally, in the previous section, we reviewed several related studies on multi-agent collaboration under LTL specifications. Notably, the final algorithm discussed appears promising for the overall goals of this thesis. Consequently, we will adopt a similar structure but modify certain key aspects to achieve our objectives. Specifically, we will:

1. Consider recurring LTL tasks, which are excluded from sc-LTL.
2. Aim to reduce complexity by minimizing the number of regions.
3. Ensure compatibility with existing software by utilizing the wFTS for planning instead of the PBA. Despite some necessary compromises, this approach will lower computational complexity, as detailed in Chapter 4.
4. Simplify the communication structure to a single layer. Although this simplification streamlines the overall structure, it could exponentially increase complexity, consequently, this problem will be addressed.

CHAPTER 3 METHODS

The purpose of this chapter is to provide an overview of the research methods used in this thesis. Section 3.1 describes the research process. Section 3.2 focuses on the data collection techniques, software, and hardware used for experimental design within this thesis. Section 3.3 explains the reliability and validity of the methods used, as well as an analysis of the data collected.

3.1 Research Process

The research process that will be followed in this can be summarized in four main steps:

Step 1 In parallel perform a literature study and design the algorithm.

Step 2 Implement the algorithm while integrating it with software previously developed.

Step 3 Test all the functionalities of the algorithm in a controlled/nominal ROS2 environment.

Step 4 Perform experiments on actual hardware at SML to validate the performance of the algorithm.

A diagram representing the research process is shown in Figure 3.1

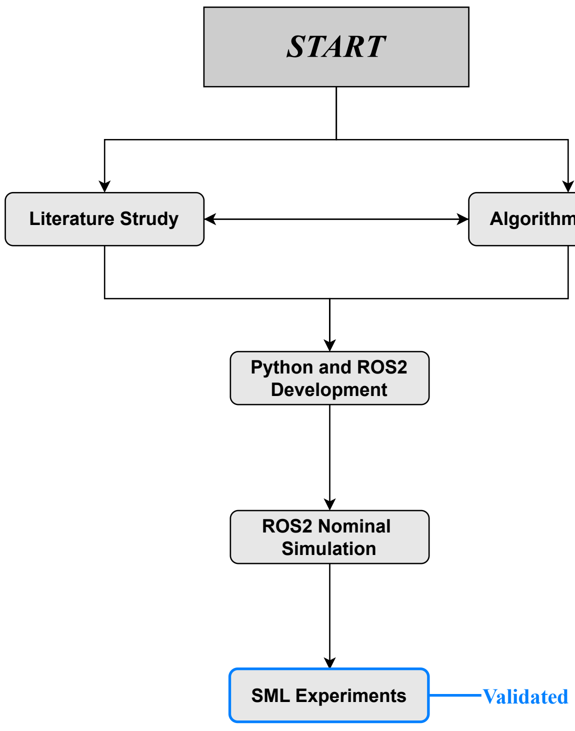

Figure 3.1: Diagram showing the research process

3.2 Data Collection and Experiment design

In this section, we will introduce the main frameworks and hardware components used during the experimental phase of our project. We will detail the main functionalities of each framework and hardware unit, and explain how they will be employed to collect data.

3.2.1 ROS2

ROS2 is the second iteration of the most used framework in robotics. It consists of a set of software libraries and tools that are developed by the ROS2 community for both research and industrial applications. As a middleware, ROS2 operates in conjunction with programming languages such as Python and C++, in this work we will mostly deal with Python. Furthermore, ROS2 provides a communication infrastructure among its core units, known as nodes. In this subsection, we will describe the major components of ROS2 used in this thesis.

Node

Nodes in ROS2 are individual software entities within a robotic system, each performing specific tasks or functions. These tasks typically correspond to different components of the system, such as sensors, actuators, controllers, or algorithms. Nodes communicate through a system of publishing and subscribing to topics, which enables them to exchange messages and coordinate activities efficiently. Lastly, it is important to note that for this communication to happen the nodes must run on the same local network.

In our project, each robotic agent is equipped with several specialized nodes. The first is an LTL planner node, which is responsible for creating the high-level plan for the agent, incorporating the algorithm developed in Chapter 4. Following this, there is a node dedicated to executing actions that the robots are designed to perform. The final node in the sequence is tasked with the actual execution of these actions, directly interfacing with the robot's hardware. It is important to note that while our work focused on the development and integration of the first two nodes, the execution node is provided by the robot's manufacturer.

Message

A message is the fundamental data structure used for the communication between nodes, it is designed to accommodate a wide range of built-in types. It supports standard primitive types such as integers, floats, Booleans and strings, as well as more complex structures like arrays of these primitive types. Additionally, messages in ROS2 are highly versatile and capable of adopting an arbitrarily nested form to meet each application's specific requirements.

Topic

A topic acts as a communication channel that enables nodes to exchange messages. Each topic is linked to a specific type of message, which dictates the structure and content of the data that can be both published and subscribed to on that topic. This structure enables asynchronous communication, allowing nodes to exchange information without direct dependencies on one another.

By publishing messages to topics, nodes make data available for other nodes subscribed to the same topics. This publish-subscribe messaging paradigm enhances modular and scalable communication within a distributed robotic system, promoting independent and asynchronous operation of nodes.

3.2.2 Qualisys Motion Capture

Qualisys Motion Capture (MoCap) [22] is a system designed for capturing and analyzing the movements of objects or subjects within 3D space. This system uses a combination of high-speed cameras and reflective markers, visible in Figure 3.5, to precisely track the position and orientation (pose) of objects with high accuracy.

The process begins with the placement of reflective markers on the objects or subjects of interest. These markers are designed to reflect infrared light emitted by the MoCap cameras, which allows to accurately determine their pose. Once the cameras detect these markers, the system employs advanced algorithms to triangulate their locations accurately.

Additionally, the developers have provided a ROS2 package that allows the transmission of pose data through specific topics and messages regarding each tracked object within the workspace. Furthermore, dedicated software, shown in Figure 3.2, allows for the assignment of markers to each object and for the visualization of the entire setup.

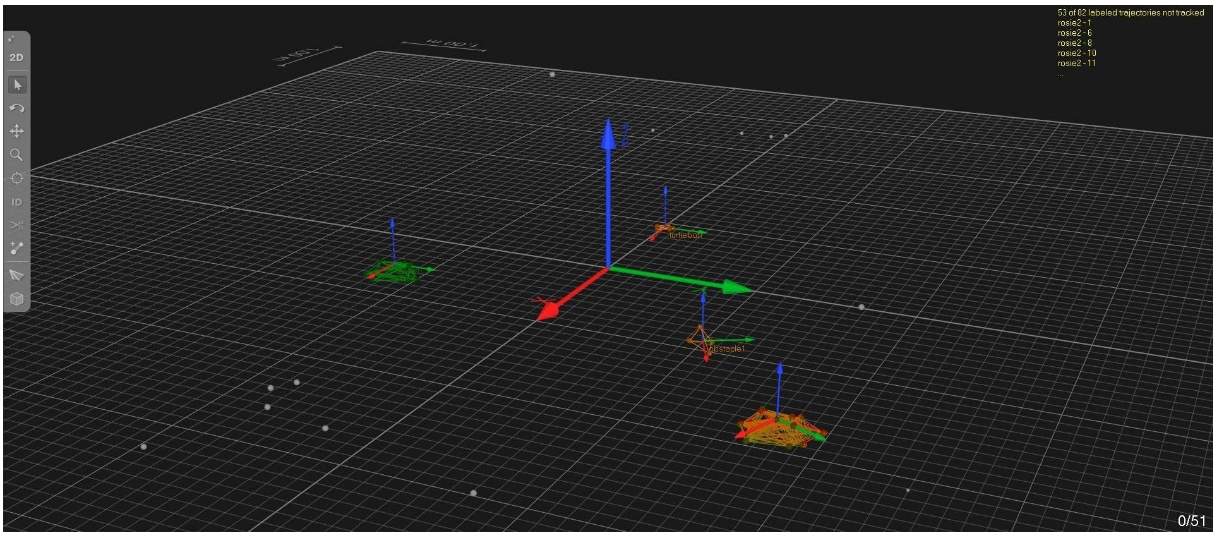

Figure 3.2: MoCap visual output

3.2.3 Hebi "Rosie" Mobile Omni-Directional Base

The Hebi 'Rosie' Mobile Omni-Directional Base [23], Rosie for short, is depicted in Figure 3.3. This robotic platform is equipped with omnidirectional wheels that enable it to maneuver effortlessly in any direction, making it particularly well-suited for navigating through tight spaces with high precision. Its compact design facilitates the seamless integration of additional components such as grippers, sensors, and various payloads. The Rosies at SML are equipped with a gripper allowing the robot to grasp, lift, and transport objects. Lastly, it is worth pointing out that the internal computer is running an Ubuntu distribution to ensure compatibility with ROS2.

In order to design the MPC controller for this platform, it is necessary first to define the robot's kinematic model. Additionally, to establish the constraints for the controller, we refer to the technical specifications outlined in Table 3.1. It is important to note that these specifications are based on those utilized during our experiments, rather than the manufacturer's original specifications.

Base radius	0.33 m
Maximum linear velocities	0.20 m/s
Minimum linear velocities	-0.20 m/s
Maximum angular velocity	2.00 rad/s
Minimum angular velocity	-2.00 rad/s

Table 3.1: Techincal specifications of Hebi Rosie

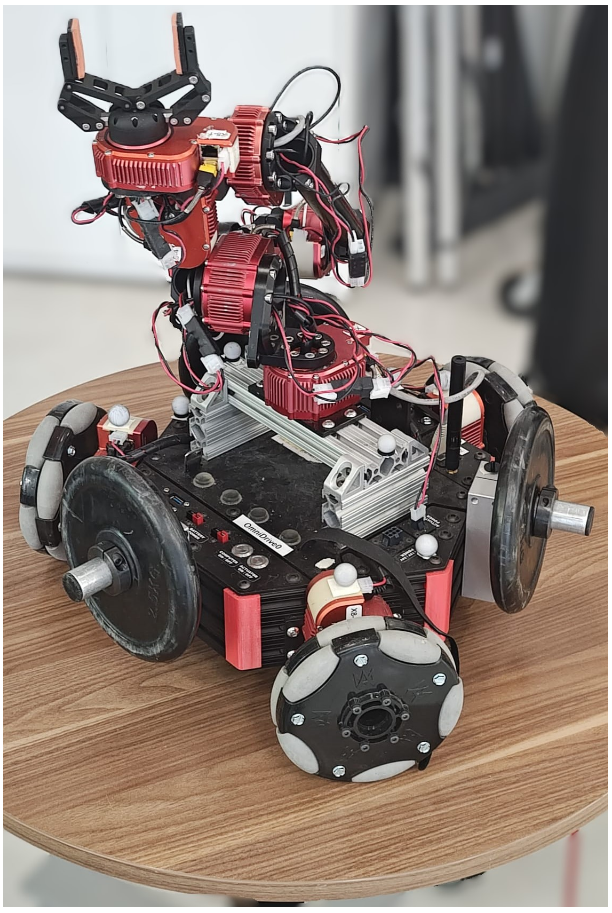

Figure 3.3: Hebi ”Rosie” Mobile Omni-Directional Base

Kinematic Model

Although an accurate model can be developed, a relatively simple model is often sufficient for practical purposes. Therefore, to design an effective MPC controller, a kinematic model of the Rosie is adequate.

A kinematic model, as widely discussed in the literature, describes the motion of mechanical points, bodies, and systems without considering the forces acting upon them or their physical properties. In our specific case, we need to consider a robot equipped with omnidirectional wheels that allow plane movement without altering its orientation.

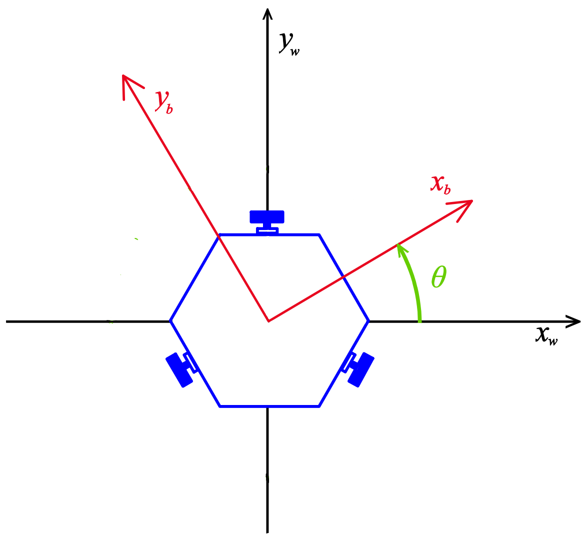

Figure 3.4: Omnidirectional robot kinematics model

Let $\mathcal{F}_w$ represent the world reference frame and $\mathcal{F}_b$ the body reference frame, with the origin located at the robot's center of mass. The state of the robot is defined as $\mathbf{x} = [x,\ y,\ \theta]^T$, representing the pose of $\mathcal{F}_b$ with respect to $\mathcal{F}_b$. The system inputs are $\mathbf{u} = [v_x,\ v_y,\ \omega]^T$, where v_x and v_y are the linear velocities along the x_b and y_b axis respectively, and ω is the angular velocity of $\mathcal{F}_b$ relative to $\mathcal{F}_w$.

As shown in Figure 3.4, it is straightforward to derive the kinematic equations of the unicycle model:

$$\begin{cases} \dot{x} = v_x \cdot \cos\theta - v_y \cdot \sin\theta \\ \dot{x} = v_x \cdot \sin\theta + v_y \cdot \cos\theta \\ \dot{\theta} = \omega \end{cases} \tag{3.1}$$

These equations describe how the linear and angular velocities influence the robot's pose over time.

3.2.4 Robotis TurtleBot3 Burger

The Robotis TurtleBot3 Burger [24], TurtleBot for short, is depicted in Figure 3.5. It comes equipped with differential drive wheels, along with a variety of onboard sensors, including laser distance sensors, and infrared sensors, for navigation and perception tasks. The robots available at SML are equipped with a Raspberry Pi 4 [25], which runs an Ubuntu distribution to ensure compatibility with ROS2.

In order to design the MPC controller for this platform, it is necessary first to define the robot's kinematic model. Additionally, to establish the constraints for the controller, we refer to the technical specifications outlined in Table 3.2. It is important to note that these specifications are based on those utilized during our experiments, rather than the manufacturer's original specifications.

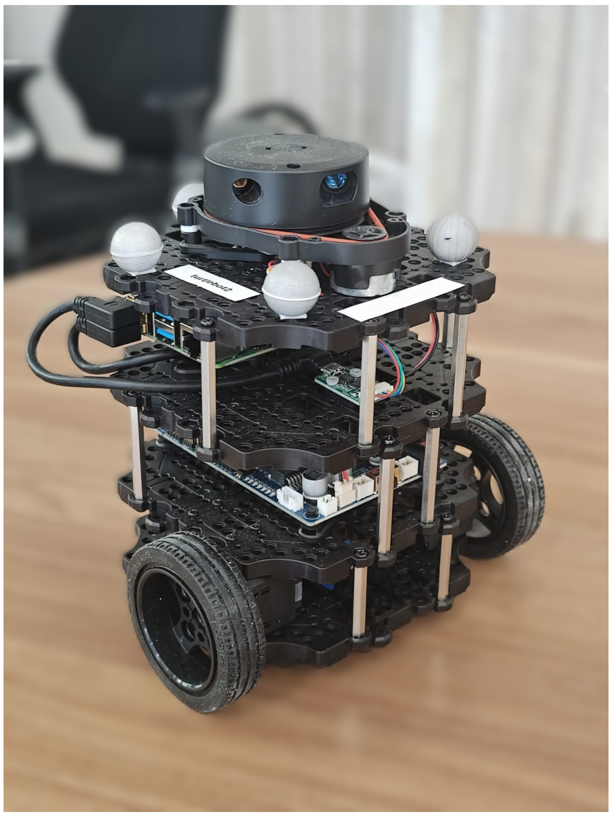

Figure 3.5: Robotis TurtleBot3 Burger

Robot radius	0.11 m
Maximum linear velocity	0.20 m/s
Minimum linear velocity	-0.20 m/s
Maximum angular velocity	1.30 rad/s
Minimum angular velocity	-1.30 rad/s

Table 3.2: Techincal specifications of Robotis TurtleBot3 Burger

Kinematics Model

As previously mentioned, the TurtleBot is a differential drive robot, meaning each wheel can be controlled independently. However, thanks to the ROS2 libraries developed by the manufacturer, it is not necessary to directly control each wheel. Instead, we only need to specify the desired linear and angular velocities. The low-level controller implemented within the ROS2 framework translates these velocities into the appropriate wheel speeds. This abstraction simplifies the definition of the kinematic model, allowing us to treat the TurtleBot as a unicycle for control purposes.

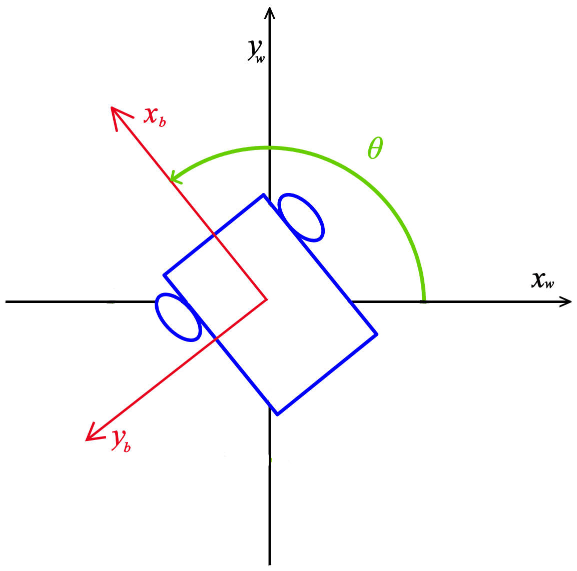

Figure 3.6: Unicycle kinematics model

Let $\mathcal{F}_w$ represent the world reference frame and $\mathcal{F}_b$ the body reference frame, with the origin located at the robot's center of mass. The state of the robot is defined as $\mathbf{x} = [x,\ y,\ \theta]^T$, representing the pose of $\mathcal{F}_b$ with respect to $\mathcal{F}_b$. The system inputs are $\mathbf{u} = [v,\ \omega]^T$, where v is the linear velocity along the sagittal axis (*i.e.,* the x axis), and ω is the angular velocity of $\mathcal{F}_b$ relative to $\mathcal{F}_w$.

As shown in Figure 3.6, it is straightforward to derive the kinematic equations of the unicycle model:

$$\begin{cases} \dot{x} = v \cos\theta \\ \dot{y} = v \sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{3.2}$$

These equations describe how the linear and angular velocities influence the robot's pose over time.

3.3 Assessing Reliability and Validity of Data Collected

The most important question to address now is the reliability of the steps undertaken in the research process to develop the required capabilities for the goals stated in Chapter 1. To evaluate this, we first need to examine the methods employed and the data collected.

3.3.1 Validity and Reliability of the Method

A primary concern with the proposed methods is their generalizability and adaptability to systems beyond the specific robots used for testing. However, while developing the algorithm, we considered a generic set of agents without restricting their capabilities, as detailed in Section 4.1. This ensures that the algorithm can adapt to any type of agent we choose. The only aspect where this work is specific is in the design of the MPC controller for the tested robots. Nevertheless, as discussed in Section 2.4, this is a generic framework applicable to any dynamical system. Furthermore, ROS2 can easily handle more complex systems like quadcopters. Thus, although we have primarily focused on ground robots, the methodology should effectively apply to any system compatible with ROS2.

Another concern is scalability and whether the methods used offer any guarantees of reducing complexity. It is important to note that mathematical proofs can provide theoretical assurances regarding complexity reduction, which will be addressed in this work.

3.3.2 Data Validity and Reliability

A primary concern with the data obtained from the experiments is its validity and reliability. However, since we are recording the same data used to compute the controller values from the MPC, we can determine that as long as the MPC functions correctly, the data should be reliable enough to draw meaningful conclusions.

Another major aspect to consider is whether the communication between the robots is reliable enough to ensure that all necessary data is sent to the correct agents. In this case, the reliable and stable architecture of ROS2 provides the necessary guarantees. Therefore, if the overall results provided by the algorithm, which will be recorded, are satisfactory, we can conclude that the communication was reliable enough to draw valid conclusions.

Lastly, it is necessary to mention how we will measure complexity in the experimental settings, which will primarily rely on measuring execution time. Unfortunately, execution time can be influenced by a multitude of factors, many of which are beyond our control. To ensure fair comparisons when considering complexity and to draw accurate conclusions, we will control all the variables within our ability and conduct multiple experiment runs to minimize the impact of uncontrollable factors.

CHAPTER 4 MULTI-AGENT COLLABORATION

This chapter presents the majority of the work developed for this thesis. First, we will explain the agents setup and problem statement, providing a clear understanding of the context and the specific challenges addressed. Following this, we delve into the core of this work, which is the multi-agent collaboration algorithm developed. This section will detail its design, highlight the innovative aspects, and discuss their pros and cons with respect to previous research. Lastly, we will briefly discuss the Model Predictive Control (MPC) controller developed for the field experiments, outlining the formulation and the parameters used. Through these sections, this chapter aims to offer a comprehensive overview of the work developed and the advancements made in the field of multi-agent collaboration under Linear Temporal Logic (LTL) specification.

4.1 Agents Setup

This section will focus on explaining how the multi-agent system is composed. Moreover, significant emphasis will be placed on modeling each agent's behavior through the use of weighted Finite Transition System (wFTS) and the specific syntax of LTL to specify each agent's task, which will be considered in the rest of the work.

4.1.1 Multi-agent System

We consider a set of heterogeneous agents $\mathcal{N} = \{a_i,\ i = 1, 2, ..., N \mid N \geq 1\}$. These agents can move in a partially known workspace and perform different primitive actions. These actions can be combined into more complex tasks that may require collaboration with other agents, given in the form of an LTL formula.

Moreover, we assume that we are working with loosely coupled multi-agent systems. In these systems, agent collaborations are sporadic, meaning they are required only occasionally relative to the total actions each agent performs for its local tasks. A more in-depth explanation of this assumption will be provided in Subsection 4.2.5.1 Lastly, we will consider that all the agents are connected to the same network, moreover, the number of agents connected is supposed to be limited and within the limitation given by the network implemented. Given these premises, any agent can exchange Robot Operating System 2 [1] (ROS2) messages directly with any other agent inside the workspace.

4.1.2 Motion Transition System

As seen in Section 2.5, most previous approaches rely on a fully partitioned workspace and then build a wFTS. A common approach, employed by Guo *et al.,* in [20], is to define a grid-like set of regions where the robot can be at a given moment. While this approach has advantages, it increases computational complexity since the number of regions where actions are completed is low compared to the total number of regions in the wFTS. Additionally, many regions represent obstacles, implying that the workspace must be well-known a priori by each agent.
In this work, we developed an alternative approach that reduces the number of regions by considering only a limited set, called Region Of Interests (ROIs). Obstacle avoidance and movement between regions are managed by the low-level controller, specifically the MPC controller developed in Section 4.3. This approach reduces the computational complexity of the final algorithm due to the fewer regions. However, it sacrifices the ability to locate the agent at any time within the wFTS.

We will now detail the specifics of the workspace setup developed for this thesis. Each agent a_i knows only the set of M^{a_i} ROIs, denoted by $\Pi_{\mathcal{M}}^{a_i} = \pi_1^{a_i}, \pi_2^{a_i}, ..., \pi_{M^{a_i}}^{a_i}$. These regions are known a priori by each agent, and each agent knows only the regions necessary for its actions. While this is sufficient for the collaboration algorithm to work, the low-level controller may require a more detailed knowledge of the workspace, as specified in Section 4.3. These requirements may vary based on the controller's implementation and type.

Remark 4.1. It is important to note that we do not assume $\bigcup_{j=1}^{M_{a_i}} \pi_j^{a_i} = \mathcal{W}$, where $\mathcal{W}$ is the workspace. By reducing the number of known regions, we simplify the wFTS used for agent motion. This reduction decreases the number of states within the wFTS, translating to lower computational complexity in both the initial planning and the plan adaptation to allow collaboration with other agents.

A significant drawback of this approach is the potential for an agent to be in a position not represented in the wFTS, within the workspace at time t. One possible solution is to introduce a specific state in the wFTS to represent this scenario. However, this would complicate determining a consistent travel time *i.e.,* weight, from this new state to the target ROI, as the initial ROI would also influence this time. Due to such limitation, this approach will be discarded and the agent will be considered in the region where the movement originated until the low-level controller successfully drives it to the target ROI.

As previously implied we can model this type of motion through a wFTS defined according to Definition 2.4

Definition 4.1. The wFTS that models the agent motion is defined as follows:

$$\mathcal{T}_{\mathcal{M}}^{a_i} \triangleq \left(\Pi_{\mathcal{M}}^{a_i}, \Pi_{\mathcal{M},0}^{a_i}, \Psi_{\mathcal{M}}^{a_i}, \Sigma_{\mathcal{M}}^{a_i}, \longrightarrow_{\mathcal{M}}^{a_i}, \mathrm{L}_{\mathcal{M}}^{a_i}, \mathrm{T}_{\mathcal{M}}^{a_i}\right) \tag{4.1}$$

where:

- $\Pi_{\mathcal{M}}^{a_i}$ is the set of ROIs defined above.
- $\Pi_{\mathcal{M},0}^{a_i} \in \Pi_{\mathcal{M}}^{a_i}$ is starting ROI of agent a_i,
- $\Psi_{\mathcal{M}}^{a_i}$ is the set of Atomic Propositions (APs) describing the properties of the workspace,
- $\Sigma_{\mathcal{M}}^{a_i}$ is the set of actions, namely the actions to move from an ROI to another one,
- $\longrightarrow_{\mathcal{M}}^{a_i} \subseteq \Pi_{\mathcal{M}}^{a_i} \times \Sigma_{\mathcal{M}}^{a_i} \times \Pi_{\mathcal{M}}^{a_i}$ is the transition relation, representing the valid transitions between the different ROIs,
- $\mathrm{L}_{\mathcal{M}}^{a_i} : \Pi^{a_i} \rightarrow 2^{\Psi_{\mathcal{M}}^{a_i}}$ is the labeling function, indicating the properties held by each ROI,
- $\mathrm{T}_{\mathcal{M}}^{a_i} : \longrightarrow_{\mathcal{M}}^{a_i} \rightarrow \mathbb{R}^+$ is the transition time function, representing the estimated time necessary for each transition; it is a design parameter,

4.1.3 Action Model

After defining a model that captures the movement and localization of the agent in the workspace, we need to establish a framework that abstracts the non-movement actions of the agent. This model must distinguish between different types of actions: those that require collaboration between agents and those that an agent can complete independently.

Definition 4.2. The set of all non-movement related actions that a generic agent a_i can perform is:

$$\Sigma_{\mathscr{A}}^{a_i} \triangleq \Sigma_l^{a_i} \cup \Sigma_c^{a_i} \cup \Sigma_h^{a_i} \tag{4.2}$$

where:

- $\Sigma_l^{a_i}$: the set of *local* actions that can be performed without collaboration from other agents. These actions can be initiated by agent a_i during the initial planning.
- $\Sigma_c^{a_i}$: the set of *collaborative* actions that require help from other agents. These actions can be initiated by agent a_i during the initial planning.
- $\Sigma_h^{a_i}$: the set of *assisting* actions that the agent can perform to help other agents. These actions cannot be initiated by agent a_i during the initial planning.

From this definition, we derive the set of *active* actions of agent a_i, which are the actions that the agent itself can initiate:

$$\Sigma_a^{a_i} \triangleq \Sigma_l^{a_i} \cup \Sigma_c^{a_i} \tag{4.3}$$

Additionally, we call $\Sigma_h^{\sim a_i}$ the set of *external* assisting actions that agent a_i depends on, which can be provided by other agents in $\mathcal{N}$.

Definition 4.3. The action $\sigma_0 = None$ represents that the agent is not performing any action. This action is assumed to be a *local* action that can be performed by all agents:

$$\sigma_0 \in \Sigma_l^{a_i} \quad \forall a_i \tag{4.4}$$

Definition 4.4. The non-movement related action model of agent a_i is defined as the tuple:

$$\mathscr{A}^{a_i} \triangleq (\Sigma_{\mathscr{A}}^{a_i}, \Psi_{\mathscr{A}}^{a_i}, \mathrm{L}_{\mathscr{A}}^{a_i}, \mathrm{Cond}^{a_i}, \mathrm{Dura}^{a_i}, \mathrm{Depd}^{a_i}) \tag{4.5}$$

where:

- $\Sigma_{\mathscr{A}}^{a_i}$ is the set of non-movement related actions, as defined in Definition 4.2.
- $\Psi_{\mathscr{A}}^{a_i}$ is the set of APs related to $\Sigma_{\mathscr{A}}^{a_i}$.
- $\mathrm{L}_{\mathscr{A}}^{a_i} : \Sigma_{\mathscr{A}}^{a_i} \to 2^{\Psi_{\mathscr{A}}^{a_i}}$ is the labeling function. We assume $\mathrm{L}_{\mathscr{A}}^{a_i}(\sigma_a) \subseteq \Psi_{\mathscr{A}}^{a_i}$, $\forall \sigma_a \in \Sigma_a^{a_i}$, while $\mathrm{L}_{\mathscr{A}}^{a_i}(\sigma_h) = \emptyset$, $\forall \sigma_h \in \Sigma_h^{a_i}$. This ensures that an agent does not directly initiate an assisting action in its initial plan.
- $\mathrm{Cond}^{a_i} : \Sigma_{\mathscr{A}}^{a_i} \times 2^{\Psi_{\mathcal{M}}^{a_i}} \to \top/\bot$ specifies the set of region properties that must be satisfied to execute an action.
- $\mathrm{Dura}^{a_i} : \Sigma_{\mathscr{A}}^{a_i} \to \mathbb{R}^+$ represents the estimated duration of each action, where $\mathrm{Dura}^{a_i}(\sigma_s) = T_s > 0$ serves as a design parameter.
- $\mathrm{Depd}^{a_i} : \Sigma_{\mathscr{A}}^{a_i} \to 2^{\Sigma_h^{\sim a_i}} \times 2^{\Pi^{\mathcal{N}}}$ denotes the dependence function, where $\Pi^{\mathcal{N}} = \bigcup\limits_{a_i \in \mathcal{N}} \Pi_{\mathcal{M}}^{a_i}$. This function represents the relationship between an action executed by agent a_i and the tuple composed of assisting actions performed by other agents and the possible regions where each action needs to be performed. We assume $\mathrm{Depd}^{a_i}(\sigma_s) = \emptyset, \forall \sigma_s \in \Sigma_l^{a_i} \cup \Sigma_h^{a_i}$, and $\mathrm{Depd}^{a_i}(\sigma_c) \subseteq \Sigma_h^{\sim a_i} \times 2^{\Pi^{\mathcal{N}}}, \forall \sigma_c \in \Sigma_c^{a_i}$. This implies that each collaborative action relies on a set of assisting actions from other agents, and distinct collaborative actions may depend on the same assisting actions.

Remark 4.2. The duration of the action *None*, $\mathrm{Dura}^{a_i}(\sigma_0) = T_0 > 0$, is critical for delaying collaboration with other agents if needed. This parameter must be accurately chosen for the specific application.

Remark 4.3. Each action in the dependence function is unique, meaning $\forall \sigma_d, \sigma_e \in \mathrm{Depd}^{a_i}(\sigma_m)$ such that $d \neq e$, then $\sigma_d \neq \sigma_e$. This ensures compatibility with the existing planner.

4.1.4 Agent Transition System

As we have seen in Section 2.3 to build the Product Büchi Automaton (PBA) and consequently initialize the planner, it is necessary to define a wFTS representing the agent. Now, we need to combine the motion wFTS defined in Definition 4.1 and the action model defined in Definition 4.4, and build an wFTS according to Definition 2.4.

Definition 4.5. Given the motion wFTS, $\mathcal{T}_{\mathcal{M}}^{a_i}$, and the action model, $\mathscr{A}^{a_i}$, related to agent a_i, the wFTS that models the complete behaviour of agent a_i is:

$$\mathcal{T}_{\mathcal{G}}^{a_i} \triangleq \left(\Pi_{\mathcal{G}}^{a_i}, \Pi_{\mathcal{G},0}^{a_i}, \Psi_{\mathcal{G}}^{a_i}, \Sigma_{\mathcal{G}}^{a_i}, \longrightarrow_{\mathcal{G}}^{a_i}, \mathrm{L}_{\mathcal{G}}^{a_i}, \mathrm{T}_{\mathcal{G}}^{a_i}\right) \tag{4.6}$$

Where:

- $\Pi_{\mathcal{G}}^{a_i} = \Pi_{\mathcal{M}}^{a_i} \times \Sigma_{\mathscr{A}}^{a_i}$ is the set states of agent a_i. Specifically, $\pi_{\mathcal{G},j}^{a_i} = \langle \pi_k^{a_i}, \sigma_n^{a_i} \rangle \in \Pi_{\mathcal{G}}^{a_i} \quad \forall \pi_k^{a_i} \in \Pi_{\mathcal{M}}^{a_i}$, $\forall \sigma_n^{a_i} \in \Sigma_{\mathscr{A}}^{a_i}$
- $\Pi_{\mathcal{G},0}^{a_i} = \langle \Pi_{\mathcal{M},0}^{a_i}, \textit{None} \rangle$ is the initial state.
- $\Psi_{\mathcal{G}}^{a_i} = \Psi_{\mathscr{A}}^{a_i} \cup \Psi_{\mathcal{M}}^{a_i}$ is the set of APs related to the complete agent model.
- $\Sigma_{\mathcal{G}}^{a_i} = \Sigma_{\mathcal{M}}^{a_i} \bigcup \Sigma_{\mathscr{A}}^{a_i}$. We will adopt the same subdivision into local, collaborative, and assisting actions as introduced in Definition 4.2, *i.e.,* $\Sigma_{\mathcal{G}}^{a_i} = \Sigma_l^{a_i} \cup \Sigma_c^{a_i} \cup \Sigma_h^{a_i}$. We consider $\Sigma_{\mathcal{M}}^{a_i} \subset \Sigma_l^{a_i}$, meaning the movement actions will be considered local actions.
- $\longrightarrow_{\mathcal{G}}^{a_i} \subseteq \Pi_{\mathcal{G}}^{a_i} \times \Sigma_{\mathcal{G}}^{a_i} \times \Pi_{\mathcal{G}}^{a_i}$ is the transition relation:

  $\left(\langle \pi_h^{a_i}, \sigma_m^{a_i} \rangle, \sigma_n^{a_i}, \langle \pi_j^{a_i}, \sigma_p^{a_i} \rangle\right) \in \longrightarrow_{\mathcal{G}}^{a_i}$ if:

  1. $\sigma_m^{a_i} = \sigma_p^{a_i} = \textit{None}$, then $(\pi_h^{a_i}, \sigma_n^{a_i}, \pi_j^{a_i}) \in \longrightarrow_{\mathcal{M}}^{a_i}$. The transition between ROIs is possible only if the agent is not performing any action.
  2. $\sigma_m^{a_i} = \textit{None}$ and $\sigma_p^{a_i} = \sigma_n^{a_i} \neq \textit{None}$ and $\pi_h^{a_i} = \pi_j^{a_i}$, then $\mathrm{Cond}^{a_i}(\sigma_n^{a_i}, L_{\mathcal{M}}^{a_i}(\pi_h)) = \top$. An action can be performed only if the required region properties are satisfied and the agent is not performing another action.
  3. $\sigma_m^{a_i} \in \Sigma_{\mathscr{A}}^{a_i}$ and $\sigma_n^{a_i} = \sigma_p^{a_i} = \textit{None}$ and $\pi_h^{a_i} = \pi_j^{a_i}$. After an action is performed (including *None*), the agent must transition to the *None* action state.
- $\mathrm{L}_{\mathcal{G}}^{a_i} : \Sigma_{\mathcal{G}}^{a_i} \rightarrow 2^{\Psi_{\mathcal{G}}^{a_i}}$ is the labeling function, defined as: $\mathrm{L}_{\mathcal{G}}^{a_i}\left(\langle \pi_h^{a_i}, \sigma_m^{a_i} \rangle\right) = \mathrm{L}_{\mathcal{M}}^{a_i}(\pi_h^{a_i}) \cup \mathrm{L}_{\mathscr{A}}^{a_i}(\sigma_m^{a_i})$
- $\mathrm{T}_{\mathcal{G}}^{a_i} : \longrightarrow_{\mathcal{G}}^{a_i} \rightarrow \mathbb{R}^+$ is the estimated duration of each transition.

  It is defined according to the cases presented in $\longrightarrow_{\mathcal{G}}^{a_i}$:

  1. $\mathrm{T}_{\mathcal{G}}^{a_i}\left(\langle \pi_h^{a_i}, \sigma_m^{a_i} \rangle, \sigma_n^{a_i}, \langle \pi_j^{a_i}, \sigma_p^{a_i} \rangle\right) = \mathrm{T}_{\mathcal{M}}^{a_i}\left(\pi_h^{a_i}, \sigma_n^{a_i}, \pi_j^{a_i}\right)$ *i.e.,* the estimated time to move between two ROIs.
  2. $\mathrm{T}_{\mathcal{G}}^{a_i}\left(\langle \pi_h^{a_i}, \sigma_m^{a_i} \rangle, \sigma_n^{a_i}, \langle \pi_j^{a_i}, \sigma_p^{a_i} \rangle\right) = \mathrm{Dura}^{a_i}(\sigma_n^{a_i})$ *i.e.,* the estimated time to complete $\sigma_n^{a_i}$.
  3. $\mathrm{T}_{\mathcal{G}}^{a_i}\left(\langle \pi_h^{a_i}, \sigma_m^{a_i} \rangle, \sigma_n^{a_i}, \langle \pi_j^{a_i}, \sigma_p^{a_i} \rangle\right) = T_0$ *i.e.,* the time required to complete *None*.

Remark 4.4. Due to the way we defined $\longrightarrow_{\mathcal{G}}^{a_i}$, all the actions are performed only if the agent is in the correct ROI.

Remark 4.5. To simplify notation from now on we will call $\mathrm{T}_{\mathcal{G}}^{a_i}(\sigma_n^{a_i}) = \mathrm{T}_{\mathcal{G}}^{a_i}\left(\pi_{\mathcal{G},m}^{a_i}, \sigma_n^{a_i}, \pi_{\mathcal{G},p}^{a_i}\right)$

Definition 4.6. The path of agent a_i is defined as:

$$\tau^{a_i} = \pi_{\mathcal{G},0}^{a_i}\pi_{\mathcal{G},1}^{a_i}\dots \tag{4.7}$$

where $\pi_{\mathcal{G},j}^{a_i} \in \Pi_{\mathcal{G}}^{a_i}$, $\pi_{\mathcal{G},0}^{a_i} = \Pi_{\mathcal{G},0}^{a_i}$, and $\left(\pi_{\mathcal{G},s}^{a_i}, \sigma_s^{a_i}, \pi_{\mathcal{G},s+1}^{a_i}\right) \in \longrightarrow_{\mathcal{G}}^{a_i}, \forall s = 0, 1, \dots$.

Definition 4.7. The sequence of actions that allow the transitions from the states of τ^{a_i} is defined as:

$$\rho^{a_i} = \sigma_0^{a_i}, \sigma_1^{a_i}, \dots \tag{4.8}$$

where $\sigma_s^{a_i} \in \Sigma_{\mathcal{G}}^{a_i}$, $\left(\pi_{\mathcal{G},s}^{a_i}, \sigma_s^{a_i}, \pi_{\mathcal{G},s+1}^{a_i}\right) \in \longrightarrow_{\mathcal{G}}^{a_i}, \forall s = 0, 1, \dots$ and $\pi_{\mathcal{G},s}^{a_i}, \pi_{\mathcal{G},s+1}^{a_i} \in \tau^{a_i}$

Definition 4.8. The trace of agent a_i is defined as:

$$\mathit{trace}\left(\tau^{a_i}\right) = L_{\mathcal{G}}^{a_i}\left(\pi_{\mathcal{G},0}^{a_i}\right) L_{\mathcal{G}}^{a_i}\left(\pi_{\mathcal{G},1}^{a_i}\right)\dots \tag{4.9}$$

4.1.5 LTL Specifications

In Section 2.1, the full LTL syntax and semantics were defined. This work focuses on a specific subset of LTL formulas tailored for generating plans for individual agents. These formulas are designed to accommodate finite-length detours, *i.e.,* a sequence of states between two adjacent states in a plan, without violating the original LTL formula associated with each agent. Our approach restricts LTL formulas to positive normal form, where negation ($\neg$) is only applied to atomic propositions. This constraint eliminates the widespread use of the $\Box$ operator. Additionally, we exclude the $\bigcirc$ operator to prevent overly interdependent actions; for the same reason, the U operator is solely employed to construct the $\Diamond$ operator.

Definition 4.9. The LTL grammar considered in this work is:

$$\varphi' ::= \top | a | \neg a | \varphi_1' \wedge \varphi_2' | \Diamond \varphi' | \tag{4.10}$$

where a is an AP.

Another important aspect of this work is the possibility of implementing recurring tasks, *i.e.,* tasks that repeat infinitely often. However, with the grammar defined by Equation (4.10), the generation of such tasks is not feasible. To enable this behavior, we need to incorporate the $\Box$ temporal operator. To prevent the emergence of problematic tasks that could hinder detour construction without violating the original LTL formula, we limit the structure of LTL tasks considered in this study.

Assumption 1. The LTL task assigned to each agent a_i is built as follows:

$$\varphi^{a_i} = \varphi_1' \wedge \Box\Diamond\varphi_2' \tag{4.11}$$

Where φ_1' and φ_2' are LTL formulas defined according to Definition 4.9.

Remark 4.6. The LTL tasks built with Equation (4.11) enable the integration of detours to any state within the wFTS between two consecutive states in the original plan, devised to satisfy the original task while ensuring it is not violated.

4.2 Collaboration Algorithm

This section presents the main work developed for the thesis: the collaboration algorithm. First, we provide a formal definition of the problem. Then, we offer a step-by-step explanation of the algorithm, highlighting the design choices and how lower computational complexity is achieved compared to previous approaches. Additionally, we demonstrate theoretical properties where possible to prove the correctness of the algorithm. The algorithm relies on a bottom-up motion and task coordination strategy that includes an offline initial plan synthesis and an online coordination scheme based on the real-time exchange of request, reply, and confirmation messages. The offline synthesis generates an initial plan, while the online scheme dynamically adjusts it to accommodate multi-agent collaboration.
Lastly, it is important to note that in this section only the pseudocode of the algorithms developed will be presented while for the full implementation and ROS2 integration we refer to Subsection 6.3.1

4.2.1 Problem Formulation

The local task of agent a_i, denoted by φ^{a_i}, is given as a LTL formula defined according to Equation (4.11) over the set of atomic propositions $\Psi_{\mathcal{G}}^{a_i}$ from Equation (4.6). Thus, φ^{a_i} can contain requirements on the agent's motion, local and collaborative actions. Given an infinite path τ^{a_i} of $\mathcal{T}_{\mathcal{G}}^{a_i}$, then τ^{a_i} fulfills φ^{a_i} if *trace* $(\tau^{a_i}) \models \varphi^{a_i}$ where the satisfaction relation is defined in Section 2.1.
Moreover, since we aim to apply the algorithm to a real-world scenario, we assume that all $T_{\mathcal{G}}^{a_i}$ are nominal values for each action's duration. These values serve as a lower bound on the effective time an action may take to be completed, acknowledging that some non-negligible delays may arise.

Remark 4.7. The existence of non-negligible delays implies that each action, $\sigma_d^{a_i}$ may take longer than the associated $T_{\mathcal{G}}^{a_i}(\sigma_d^{a_i})$, but it will still be completed within a finite amount of time.

Problem 1. Given $\mathcal{T}_{\mathcal{G}}^{a_i}$ and the locally assigned task φ^{a_i}, design a distributed control and coordination scheme such that φ^{a_i} is fulfilled for all $a_i \in \mathcal{N}$. This algorithm must also adapt to the delays induced by the experimental scenario and ensure that all actions involved in a collaboration start simultaneously for all agents involved.

4.2.2 Initial Planning

The first phase of the algorithm is the offline synthesis of an initial plan for each agent that satisfies φ^{a_i}. As explained in Subsection 2.3.1, this process is completed using the planner previously developed at KTH Royal Institute of Technology (KTH). Given the wFTS of each agent, $\mathcal{T}_{\mathcal{G}}^{a_i}$ and the locally assigned task φ^{a_i}, a PBA is built. Using model checking techniques this planner outputs the optimal initial plan for the agent, $\tau_{init}^{a_i}$ (Definition 4.6), *i.e.,* an accepting run for the PBA, namely a sequence of states that satisfy φ^{a_i}. Lastly, the sequence of actions that allow the transitions from the states of $\tau_{init}^{a_i}$ is called $\rho_{init}^{a_i}$ (Definition 4.7).

Remark 4.8. The use of this planner guarantees that *trace* $(\tau_{init}^{a_i}) \models \varphi^{a_i}$ with the trace defined according to Definition 4.8.

4.2.3 Request

As anticipated, the second phase of the algorithm relies on the online adaptation of the initial plans. This involves the exchange of request, reply, and confirmation messages between agents to determine which agents are best suited to assist with collaborative actions. Starting from this subsection, we will detail all the steps of the second phase of the algorithm. The first step we will analyze is the request for collaboration sent by an agent.

Definition 4.10. Given a collaborative action of agent a_i *i.e.,* $\sigma_c^{a_i} \in \Sigma_c^{a_i}$, the request message sent by agent a_i to all the other agents has the following structure:

$$\textbf{Request}^{a_i} = \{(\sigma_d, \pi_d, T_c^{a_i}) \forall \sigma_d \in \text{Depd}^{a_i}(\sigma_c^{a_i})\} \tag{4.12}$$

where:

- σ_d are the assistive actions required to complete $\sigma_c^{a_i}$,
- π_d are the regions where the respective σ_d is required to take place,
- $T_c^{a_i}$ is the amount of time before $\sigma_c^{a_i}$ is supposed to start according to the transitions time in a_i's plan.

To describe the algorithm that outputs **Request**^{a_i}, we assume a generic instant in time where the current state of agent a_i is $\pi_{\mathcal{G},l}^{a_i}$, which is the l-th element of a_i's plan τ^{a_i} (possibly different from $\tau_{init}^{a_i}$ due to modification from the collaboration algorithm), *i.e.,* $\pi_{\mathcal{G},l}^{a_i} = \tau^{a_i}[l]$. Moreover, the agent is currently executing action $\sigma_l^{a_i}$ *i.e.,* $\sigma_l^{a_i} = \rho^{a_i}[l]$, where ρ^{a_i} is the sequence of actions corresponding to τ^{a_i}.

This algorithm takes as input the index representing the current state and action (l), the sequence of actions associated with the agent's plan (ρ^{a_i}), the horizon length (H^{a_i}), which is the maximum time ahead a_i checks the planned actions to see if any collaborative one is present, and the time remaining before the end of the current action ($T_{rem} = \max(T_{\mathcal{G}}^{a_i}(\sigma_l^{a_i}) - \Delta t, 0)$, where Δt is the time elapsed from the start of the action to the current time instant).

The main idea behind the algorithm is to look at the future actions in the plan and check if any collaborative action is present within the given horizon. If so, we create the request; otherwise, we return an empty set and no message is sent to the other agents. The pseudocode of the algorithm is shown in Algorithm 4.1.

Algorithm 4.1: Check in Horizon and Request

```
Input  : $l$, $\rho^{a_i}$ $H^{a_i}$, $T_{rem}$
Output : Request$^{a_i}$
1  $s = 1$
2  $T_c^{a_i} = T_{rem}$
3  while $T_c^{a_i} < H^{a_i}$ do
4      $\sigma_c^{a_i} = \rho^{a_i}[l + s]$
5      if $\sigma_c^{a_i} \in \Sigma_c^{a_i}$ then
6          forall $\sigma_d \in$ Depd$^{a_i}(\sigma_c^{a_i})$ do
7              $\pi_d$ =Choose_ROI()
8              add $(\sigma_d, \pi_d, T_c^{a_i})$ to Request$^{a_i}$
9          return Request$^{a_i}$
10     $T_c^{a_i} = T_c^{a_i} + T_{\mathcal{G}}^{a_i}(\rho^{a_i}[l + s])$
11     $s = s + 1$
12 return $\emptyset$
```

The only element of Algorithm 4.1 that has not been explained is the function Choose_ROI. This is because it is a generic function that varies based on the specific experimental setup; it may take multiple inputs but outputs a specific ROI, which is where we want the specific assistive action to be completed. A more detailed definition of this function will be provided when we describe the experimental setup in Chapter 5.

4.2.4 Reply

Suppose now that agent a_i has a collaborative action in its plan. In this case, it will send **Request**^{a_i} to all agents $a_j \in \mathcal{N}$. All these agents must reply to a_i's request with a reply message. From now on, in this subsection, we will consider the perspective of a generic agent a_j.

Definition 4.11. Given the request for collaboration from agent a_i, **Request**^{a_i}, the reply message sent by agent a_j to a_i has the following structure:

$$\textbf{Reply}^{a_j} = \{(\sigma_d, \pi_d, b_d^{a_j}, t_d^{a_j}) \forall (\sigma_d, \pi_d, T_c^{a_i}) \in \textbf{Request}^{a_i}\} \tag{4.13}$$

where:

- $b_d^{a_j}$ is a boolean variable indicating the feasibility for agent a_j of offering action σ_d at region π_d

- $t_d^{a_j}$ is the time when a_j can provide the assisting action.

We assume that the request arrives at a generic instant in time when the current state of agent a_j is $\pi^{a_j}_{\mathcal{G},m}$, which is the m-th element of the current plan τ^{a_j} *i.e.,* $\pi^{a_j}_{\mathcal{G},m} = \tau^{a_j}[m]$. Moreover, the agent is currently executing action $\sigma^{a_j}_m$ *i.e.,* $\sigma^{a_j}_m = \rho^{a_j}[m]$. It is important to note that, in this case, when referring to τ^{a_j} and ρ^{a_j}, we refer to the plan and associated sequence of actions of agent a_j that the collaboration algorithm has possibly modified.

The algorithm that produces **Reply**^{a_j} takes as inputs the request received from a_i (**Request**^{a_i}), the index representing the current state and action (m), the current plan of a_j (τ^{a_j}), the wFTS associated to a_j ($\mathcal{T}^{a_j}_{\mathcal{G}}$), then $\overline{T}^{a_j}$, which is a variable indicating if a_j is currently collaborating with other agents. Specifically, $\overline{T}^{a_j} = 0$ if the agent is available to help, and $\overline{T}^{a_j} > 0$ if the agent is already collaborating, indicating when this collaboration is supposed to start, lastly it needs the time remaining before the end of the current action ($T_{rem} = \max(T^{a_i}_{\mathcal{G}}(\sigma^{a_j}_m) - \Delta t, 0)$, where Δt is the time elapsed from the start of the action to the current time instant). The algorithm, which is shown in Algorithm 4.2, not only outputs **Reply**^{a_j} but also D^{a_j} which is a dictionary containing path detours. For example, $D^{a_j}(\sigma_d)$ is the detour starting from $\tau^{a_j}[m+1]$ and ending in $\tau^{a_j}[m+2]$ that includes a visit to $\langle \pi_d, \sigma_d \rangle$ if this state is feasible for a_j.

Algorithm 4.2: Reply of agent a_j to a request from agent a_i

```
Input  : Request^{a_i}, $m$, $\tau^{a_j}$, $\mathcal{T}^{a_j}_{\mathcal{G}}$, $\overline{T}^{a_j}$, $T_{rem}$
Output : Reply^{a_j}, $D^{a_j}$
1  forall $(\sigma_d, \pi_d, T^{a_i}_c) \in$ Request^{a_i} do
2      if $\overline{T}^{a_j} = 0$ and $\langle \pi_d, \sigma_d \rangle \in \Pi^{a_j}_{\mathcal{G}}$ then
3          $\pi^{a_j}_{\mathcal{G},init} = \tau^{a_j}[m+1]$
4          $\pi^{a_j}_{\mathcal{G},targ} = \langle \sigma_d, \pi_d \rangle$
5          $\pi^{a_j}_{\mathcal{G},fin} = \tau^{a_j}[m+2]$
6          $D^{a_j}(\sigma_d), t^{a_j}_d$=GetDetour($\pi^{a_j}_{\mathcal{G},init}, \pi^{a_j}_{\mathcal{G},fin}, \pi^{a_j}_{\mathcal{G},targ}, \mathcal{T}^{a_j}_{\mathcal{G}}$)
7          add $(\sigma_d, \pi_d, \top, T_{rem} + t^{a_j}_d)$ to Reply^{a_j}
8      else
9          add $(\sigma_d, \pi_d, \perp, K)$ to Reply^{a_j}
10 return $D^{a_j}$, Reply^{a_j}
```

This simple algorithm checks if a_j can help with any of the actions in the request. Using the function `GetDetour`, which will be further examined below, it populates D^{a_j} with the feasible detours. It is important to note that if a certain state is not feasible for a_j or the agent is already occupied, then $t^{a_j}_d = K$ with K being a constant such that $K \gg T^{a_i}_c$.

Remark 4.9. $t^{a_j}_d$ and $T^{a_i}_c$ are nominal times given by the transition systems; they might not be the actual time the actions are completed due to non-negligible delays in the experimental setup (*e.g.,* movement or other actions in the plan may take more time than the nominal one).

Remark 4.10. Thanks to our definition of T_{rem}, we can be certain that $T_{rem} \geq 0$. If the action takes longer than the nominal time, our definition ensures that we avoid negative times, which would be nonsensical.

Remark 4.11. We suppose that agent a_i also replies to its own request. The reply message it will send has the following structure:

$$\textbf{Reply}^{a_i} = \{(\sigma_d, \pi_d, \perp, K) \forall (\sigma_d, \pi_d, T_c^{a_i}) \in \textbf{Request}^{a_i}\}$$

GetDetour

We will now examine `GetDetour`, which is used to generate the detour D. The algorithm description for this function will consider generic states $\pi_{\mathcal{G}}$ of a generic wFTS $\mathcal{T}_{\mathcal{G}}$. The inputs will be: $\pi_{\mathcal{G},init}$, the initial state of the detour; $\pi_{\mathcal{G},fin}$, the final state of the detour; $\pi_{\mathcal{G},targ}$, the target state we want the detour to visit; and the wFTS $\mathcal{T}_{\mathcal{G}}$. The outputs of the algorithm are the detour D and T_D, *i.e.,* the time required before starting the action that will allow the transition to $\pi_{\mathcal{G},targ}$.

Algorithm 4.3: Build a detour from the given states

```
Input  : $\pi_{\mathcal{G},init}$, $\pi_{\mathcal{G},fin}$, $\pi_{\mathcal{G},targ}$, $\mathcal{T}_{\mathcal{G}}$
Output : $T_D$, $D$
1 $(D_1, C_1)$=Dijkstra($\mathcal{T}_{\mathcal{G}}$, $\pi_{\mathcal{G},init}$, $\pi_{\mathcal{G},targ}$)
2 $(D_2, C_2)$=Dijkstra($\mathcal{T}_{\mathcal{G}}$, $\pi_{\mathcal{G},targ}$, $\pi_{\mathcal{G},fin}$)
3 $D$=$D_1$+$D_2[1:end]$                    // we avoid duplicating $\pi_{\mathcal{G},targ}$
4 $T_D$=$\sum_{i=0}^{len(C_1)-2} C_1[i]$
5 return $D$, $T_D$
```

This algorithm uses a modified version of the `Dijkstra` algorithm, which not only returns the shortest path between two given states in a wFTS (in our case $D1$ and $D2$) but also returns a list containing the costs of each action required to go from the initial state, the second argument of the `Dijkstra` function, to the target state, the third argument of the `Dijkstra` function (in our case $C1$ and $C2$).

The idea behind the algorithm is to call the `Dijkstra` function twice and then merge the two paths obtained. Lastly, we calculate T_D by considering that C_1 also includes the action required to reach $\pi_{\mathcal{G},targ}$.

It is important to note that in our use case, the value of T_D represents the minimum time required for a_j to start the assistive action requested without considering the time to finish the current action it is performing.

Remark 4.12. It is important to remark that the most computationally expensive operations in this algorithm are the two calls to `Dijkstra`, which generally have quadratic cost in the number of vertices of the graph in our case, the number of states. It is also important to underline that most of the previous approaches used the PBA, which has a higher number of vertices compared to the much simpler wFTS. The PBA approach was used to ensure that the task would still satisfy the LTL specification. However, in our case, this satisfaction is guaranteed by Assumption 1.

As a consequence, we can reduce the overall complexity of the algorithm. More detailed results will be presented in Chapter 5.

4.2.5 Confirm

At this point, we suppose that after a_i sent $\textbf{Request}^{a_i}$, it received $\textbf{Reply}^{a_j} \quad \forall a_j \in \mathcal{N}$. After collecting all the replies, it must choose the agents who are best suited to help complete the request. To communicate with the selected agents, a_i will need to send a confirmation message, providing the necessary information on whether to adapt their plans to assist and specifying which action they need to help with. If an agent is not required for any assisting action, the message will indicate that it can continue without adapting its plan.

Definition 4.12. Given $\textbf{Request}^{a_i}$, the confirmation message sent by agent a_i to the specific agent a_j, based on $\textbf{Reply}^{a_j} \quad \forall a_j \in \mathcal{N}$, has the following structure:

$$\textbf{Confirm}^{a_i}_{a_j} = \{(\sigma_d, \pi_d, c^{a_j}_d, T^{a_j}_d)\ \forall(\sigma_d, \pi_d, T^{a_i}_c) \in \textbf{Request}^{a_i}\} \tag{4.14}$$

where:

- $c^{a_j}_d$ is a Boolean variable indicating whether agent a_j is confirmed to provide σ_d at ROI π_d,
- $T^{a_j}_d$ indicates the estimated time when the collaboration should start. If the agent has not been selected, we will set $T^{a_j}_d < 0$ since this value will not be used.

The problem that needs to be solved revolves around the decision of the boolean variables $\{c^{a_j}_d,\ a_j \in \mathcal{N}\}$. These variables need to satisfy two constraints:

1. Each agent in $\mathcal{N}$ can be confirmed for at most one $(\sigma_d, \pi_d) \in \textbf{Request}^{a_i}$
2. Exactly one agent in $\mathcal{N}$ can be the confirmed collaborator for each action $(\sigma_d, \pi_d) \in \textbf{Request}^{a_i}$.

Lastly, we need to define a criterion that allows us to choose the best agents, which in our case are the ones who can provide the assisting action σ_d in a time $t^{a_j}_d$ that is as close as possible to $T^{a_i}_c$.

As previously defined, we know that $|\mathcal{N}| = N$ and we will denote $|\textbf{Request}^{a_i}| = M$. Without loss of generality, denote the set of assisting actions in $\textbf{Request}^{a_i}$ as $\{\sigma_d \mid d = 1, \ldots, M\}$. The problem of finding $\{c^{a_j}_d\}$ can be readily formulated as the following Mixed

Integer Programming (MIP) problem:

$$\min_{\{c_d^{a_j}, a_j \in \mathcal{N}\}_{d=1}^{M}} \quad \sum_{d=1}^{M} \sum_{j=1}^{N} c_d^{a_j} |t_d^{a_j} - T_c^{a_i}| \tag{4.15a}$$

$$\text{s.t.} \quad \sum_{d=1}^{M} b_d^{a_j} c_d^{a_j} \leq 1 \ \ \forall a_j \in \mathcal{N} \tag{4.15b}$$

$$\sum_{j=1}^{N} b_d^{a_j} c_d^{a_j} = 1 \ \ \forall d \in \{1, ..., M\} \tag{4.15c}$$

If the MIP has been solved we can calculate $T_d^{a_j}$ as follows:

- If $c_d^{a_j} = \top$ for any agent a_j and action σ_d then $T_d^{a_j} = t_d^{a_j}$.
- Otherwise $T_d^{a_j} = -1$

Remark 4.13. A necessary condition for the feasibility of Equation (4.15) $N > M$ since we need to be able to assign one agent to each action, moreover we will need to take into account that a_i will not be able to be assigned to any of the actions since its reply will have $b_d^{a_i} = \perp \forall \sigma_d$.

Remark 4.14. To guarantee the feasibility of Equation (4.15) there must exist a set $\mathcal{N}_H \subseteq \mathcal{N}$ such that:

- $|\mathcal{N}_H| = M$;
- Each agent must be able to help in at least one action;
- There exists a combination of these agents such that we can assign to each of them exactly one action and no action can be assigned to multiple agents

Filtering of the Replies

The general solution of the MIP has exponential complexity with respect to the number of optimization variables. Given that one of our objectives is to develop a scalable algorithm, the total number of agents involved can be high. Consequently, solving the MIP might take an extended amount of time, rendering it unsuitable for the experimental setup. To address this, we propose filtering the agents involved in the MIP while maintaining feasibility and optimality.

Procedure 1 (*Filtering Procedure*)**.** The filtering procedure can be divided into two steps. The first step involves removing agents that cannot provide any help, and the second step involves selecting only a subset of the best-performing agents for each action:

1. We define as $\mathcal{N}_U \subseteq \mathcal{N}$ as the set of agents who can assist in at least one action *i.e.,* $a_j \in \mathcal{N}_U \implies \exists(\sigma_d, \pi_d, b_d^{a_j}, t_d^{a_j}) \in \textbf{Reply}_{a_i}^{a_j}$ such that $b_d^{a_j} = \top$. After this first filtering, feasibility is preserved because $a_j \notin \mathcal{N}_U \implies a_j \notin \mathcal{N}_H$.

2. If $|\mathcal{N}_U| < M$, we will skip the second filtering phase and the calculation of the MIP, as it would be infeasible according to Remark 4.13. Otherwise, we proceed by defining the metrics used to filter the agents:

$$\Delta_d^{a_j} = |t_d^{a_j} - T_m| \tag{4.16}$$

In the second phase, for each $\sigma_d \in \textbf{Request}^{a_i}$, we sort the agents in ascending order of $\Delta_d^{a_j}$ and select the first M agents for each action, adding them to $\mathcal{N}_F$ which will be the set containing the filtered agents.

The output of the procedure will be the set of filtered agents $\mathcal{N}_F$ and its cardinality $N_F = |\mathcal{N}_F|$.

Once the filtering is complete, we solve the MIP on $\mathcal{N}_F$ instead of on $\mathcal{N}$. We summarize this procedure in Theorem 4.1

Remark 4.15. The number of agents involved in the MIP solution will be $M \leq |\mathcal{N}_F| \leq M^2$. If $N \gg M$, the complexity reduction achieved by the filtering procedure is substantial.

Theorem 4.1. *Consider the set of agents $\mathcal{N}$ and the M actions in* $\textbf{Request}^{a_i}$. *Let $\mathcal{N}_F$ be the set of filtered agents given by Procedure 1. Construct the MIP:*

$$\min_{\{c_d^{a_j}, a_j \in \mathcal{N}_F\}_{d=1}^{M}} \quad \sum_{d=1}^{M} \sum_{j=1}^{N_F} c_d^{a_j} |t_d^{a_j} - T_c^{a_i}| \tag{4.17a}$$

$$\text{s.t.} \quad \sum_{d=1}^{M} b_d^{a_j} c_d^{a_j} \leq 1 \;\; \forall a_j \in \mathcal{N}_f \tag{4.17b}$$

$$\sum_{j=1}^{N_F} b_d^{a_j} c_d^{a_j} = 1 \;\; \forall d \in \{1, ..., M\} \tag{4.17c}$$

Then the following are true:

1. *If the MIP of Equation* (4.15) *is feasible then the MIP of Equation* (4.17) *will be feasible.*

2. *The optimal solution of MIP of Equation* (4.15) *is the optimal solution of the MIP of Equation* (4.17).

Proof. We divide the proof into two parts. We first show that feasibility is kept after Procedure 1. Then, the second part will involve proving that the optimal solutions are the same for both versions of the MIP.

1. **Feasibility**: To prove that $\mathcal{N}_F$ includes a feasible solution if $\mathcal{N}$ includes one as well, we need to show that there exists a subset $\mathcal{N}_H \subseteq \mathcal{N}_F$ with the properties defined in Remark 4.14. Thanks to the first filtering phase, $\mathcal{N}_U$ contains only agents who can assist in at least one action, which implies that every agent in $\mathcal{N}_F$ can help with at least one action.
To construct $\mathcal{N}_H$ from the agents in $\mathcal{N}_F$, we consider the ordering given by $\Delta_d^{a_j}$ as defined in Equation (4.16), as we did in Procedure 1 for each action. We form sets $\mathcal{N}_d,\ d \in$

$1, \ldots, M$ such that $a_j \in \mathcal{N}_d$ if and only if $b_d^{a_j} = \top$ *i.e.,* agents who can assist with that action. We denote the size of these sets as $|\mathcal{N}_d| = M_d$, and without loss of generality, we assume $M_1 \leq M_2 \leq \ldots \leq M_M$. Since each agent can help with at least one action, we have $1 \leq M_d \leq M$. We then order the agents in ascending order of $\Delta_d^{a_j}$ for all $\mathcal{N}_d$. Starting with $\mathcal{N}_1$, we select the first agent, a_p, assign it to σ_1, and add it to $\mathcal{N}_H$. Next, we move to $\mathcal{N}_2$, find the first agent different from a_p, call it a_q, assign it to σ_2, and add it to $\mathcal{N}_H$. The existence of a_q is guaranteed since $M_2 = 1$ then $a_q \neq a_p$. Given that the original problem was feasible, a valid combination of agents must exist. If $M_2 \geq 2$, then selecting $a_q \neq a_p$ is always feasible. We continue this process until we reach $\mathcal{N}_M$, where we will select a_r, assign it to σ_M, and add it to $\mathcal{N}_H$. This construction ensures that $\mathcal{N}_H \subseteq \mathcal{N}_F$ with the properties defined in Remark 4.14, thus maintaining feasibility after filtering.

2. **Optimality**: To prove optimality, we first rewrite the objective function Equation (4.15a) as follows:

$$\sum_{j=1}^{N} c_1^{a_j} \cdot \Delta_1^{a_j} + \ldots + \sum_{j=1}^{N} c_M^{a_j} \cdot \Delta_M^{a_j} \tag{4.18}$$

Each term in this function corresponds to an action. For each term, we define the set $\mathcal{N}_d$ containing the M agents with the smallest $\Delta_d^{a_j}$ as defined in Equation (4.16). We then define the set of filtered agents as $\mathcal{N}_F = \bigcup_{d=1}^{M} \mathcal{N}_d$. We rewrite the objective function Equation (4.17a) as follows:

$$\sum_{a_j \in \mathcal{N}_1} c_1^{a_j} \cdot \Delta_1^{a_j} + \ldots + \sum_{a_j \in \mathcal{N}_M} c_M^{a_j} \cdot \Delta_M^{a_j} \tag{4.19}$$

Let $\{c_d^{a_j}\}_{\mathcal{N}}^*$ represent the optimal solution of the MIP in Equation (4.15), and $\{c_d^{a_j}\}_{\mathcal{N}_F}^*$ represent the optimal solution of the MIPin Equation (4.17). Suppose the two solutions differ. Then there must exist an agent $a_j \in \mathcal{N} \setminus \mathcal{N}_F$ such that $c_d^{a_j} = 1$ in $\{c_d^{a_j}\}_{\mathcal{N}}^*$, resulting in a smaller value for the objective function compared to any agent $a_i \in \mathcal{N}_F$ where $c_d^{a_i} = 1$ in $\{c_d^{a_j}\}_{\mathcal{N}_F}^*$. This implies:

$$\sum_{j=1}^{N} c_d^{a_j} \cdot \Delta_d^{a_j} < \sum_{a_j \in \mathcal{N}_d} c_d^{a_j} \cdot \Delta_d^{a_j} \tag{4.20}$$

Given that Equation (4.15c) requires $c_d^{a_j} = 1$ for only one a_j, this inequality in Equation (4.20) can be simplified to $\Delta_d^{a_j} < \Delta_d^{a_i}$. If this is the case, when ordering agents to build $\mathcal{N}_d$, a_j will appear before a_i. Since a_i was among the M best agents, a_j must also be one of the M best agents, implying $a_j \in \mathcal{N}_d$ and therefore $a_j \in \mathcal{N}_F$. This contradicts the initial hypothesis.

□

After attempting to solve the MIP the last step to complete this part of the algorithm is to build $\textbf{Confirm}_{a_j}^{a_i} \forall\, a_j \in \mathcal{N}$, consider the two cases that are as follows:

1. If $a_j \in \mathcal{N}_F$ we can divide this case into two further subcases:
   (a) If Equation (4.17) has a solution, both $\{c_d^{a_j}\}$ and $T_d^{a_j}$ exist. If $c_d^{a_j}$ is $\top$ add $(\sigma_d, \pi_d, \top, T_d^{a_j})$ to $\mathbf{Confirm}_{a_j}^{a_i}$; otherwise, add $(\sigma_d, \pi_d, \perp, -1)$ to $\mathbf{Confirm}_{a_j}^{a_i}$
   (b) If Equation (4.17) has no solutions, add $(\sigma_d, \pi_d, \perp, -1)$ to $\mathbf{Confirm}_{a_j}^{a_i}$
2. If $a_j \notin \mathcal{N}_F$, add $(\sigma_d, \pi_d, \perp, -1)$ to $\mathbf{Confirm}_{a_j}^{a_i}$ independently of the feasibility of Equation (4.17)

In case 1a the agents providing help will need to add the detour to their plan, while in the other cases, they will not, and continue with their original plan.

4.2.5.1 Loosely Coupled System

As previously mentioned, our focus is on implementing a distributed coordination strategy specifically designed for loosely coupled multi-agent systems. In these systems, agent collaborations are sporadic, meaning they are required only occasionally relative to the total actions each agent performs for its local tasks. When an agent formulates and solves the coordination problem using Equation (4.17), it should consistently find a solution within a bounded timeframe, provided some other agents are available to offer the necessary collaborative assistance.

Assumption 2. *(Loosely Coupled System)* There exists a finite time $\mathbf{T} > 0$ such that for each agent $a_i \in \mathcal{N}$ and any collaborative action $\sigma_c^{a_i}$ requested by agent a_i initially at time $t_c > 0$, problem in Equation (4.17) for $\sigma_c^{a_i}$ will have a solution within time $t_c + \mathbf{T}$.

It is important to note that this assumption does not require Equation (4.17) to always have a solution. Instead, it allows for the possibility of delaying the collaborative action, as we will discuss in Subsection 4.2.6, if no immediate solution is found. The coordination procedure will be repeated, and as long as there exists a finite time bound within which a solution can be found, the collaboration will eventually be accomplished. This assumption is reasonable for the tasks described by Assumption 1 formulas. The large number of agents available can provide the necessary collaboration, and the scarcity of collaborative actions relative to the total number of actions allows each agent to advance in its plan while also providing the needed assistance.

Remark 4.16. Assumption 2 is necessary to exclude some tightly coupled multiagent systems

4.2.6 Plan Adaptation

Once agent a_i has sent the confirmation messages to all other agents $a_j \in \mathcal{N}$, we need to distinguish between two possible behaviors for the agents. If a solution to the MIP has been found, all the selected assisting agents will need to adapt their plans. If no solution has been found, a_i will need to delay the collaboration. This subsection will focus on these scenarios.

Delay Collaboration

If Equation (4.17) has no solutions, then $\sigma_c^{a_i}$ cannot be fulfilled according to the current replies. Agent a_i will need to delay $\sigma_c^{a_i}$. This can be done by adding a detour of duration **T**, utilizing the self-loops given by $\sigma_0 = \mathit{None}$. However, since **T** is not known a priori, the idea is to delay $\sigma_c^{a_i}$ by an amount of time equal to T_{delay}, a design parameter, and then check again if Equation (4.17) has a solution through a new request. If a solution is found, we proceed with the original plan; otherwise, we delay the collaborative action again. Thanks to Assumption 2, there will eventually be a solution for Equation (4.17), and we will not need to delay the action any longer.

Once the assisting agents receive a negative collaboration message, they proceed without altering their plans.

Adapt to Collaborate

If Equation (4.17) has a solution, each assisting agent a_j will check if they have been selected, *i.e.,* if there exists $c_d^{a_j} = \top$ for any σ_d. If selected, the agent will update τ^{a_j} to incorporate $D^{a_j}(\sigma_d)$ and then set $\overline{T}^{a_j} = T_d^{a_j}$. If not selected, the agent will proceed without modifying τ^{a_j}. Regarding the requesting agent a_i, it will simply set $\overline{T}^{a_i} = T_c^{a_i}$.

Remark 4.17. For the assisting agents a_j involved in a collaboration, $\overline{T}^{a_j} = 0$ at the end of the detour to ensure the agent progresses in its plan. For the requesting agent a_i, $\overline{T}^{a_i} = 0$ after $\sigma_c^{a_i}$ has been completed.

4.2.7 Time Synchronization

Suppose that Equation (4.17) has a solution and a collaboration between the requesting agent a_i and the assisting agents a_j has been established. Let $\mathcal{S} \subseteq \mathcal{N}$ be the set composed only of the agents involved in this collaboration. As it stands, all the agents will have their respective actions planned at time $\overline{T}^a \forall a \in \mathcal{S}$. The main problem in synchronizing all the agents and allowing the collaboration to start at the same time is that $\overline{T}^a$ may differ among the agents involved. To solve this problem and synchronize all the agents, we have two possible solutions:

1. Add $\sigma_0 = \mathit{None}$ before the collaborative/assistive action so that all the actions involved will be executed at time $\max_a\{\overline{T}^a\}$. This solution will guarantee synchronization of the agents under nominal conditions, *i.e.,* when the time required to perform an action in the plan is as given by the wFTS. However, as explained in Remark 4.9, this is not the case in experimental scenarios where non-negligible delays in the time required to complete an action may occur. Therefore, this solution will be discarded.

2. Given the requesting agent a_i and the assisting agents $a_j \in \mathcal{S} \backslash a_i$, this solution leverages the communication capabilities between agents provided by ROS2 to effectively synchronize the agents in time. Moreover, it is resilient to delays introduced by actions in the experimental scenario. The synchronization strategy proceeds as follows: Once a_j is ready to execute the assigned σ_d, it will send a $\mathbf{Ready}^{a_j}$ message to a_i. When a_i is ready

to execute $\sigma_c^{a_i}$ and has received **Ready**^{a_j} messages from all $a_j \in \mathcal{S} \setminus a_i$, it will send a **Start** message to all a_j, indicating that all agents may now start their actions. This guarantees that they all start at the same moment in time.

The following result is valid for the second scenario:

Lemma 4.1. *There exist a finite time $T_s \geq 0$ such that if we consider t as the time when* **Confirm**$_{a_j}^{a_i}$ *has been sent; then at time $t + T_s$ the message* **Start** *will be sent and the agents will begin their collaboration.*

Proof. Consider a generic agent $a \in \mathcal{S} \setminus a_i$. At time t, it will be in state $\pi_{\mathcal{G},init}^a$. Let $\pi_{\mathcal{G},fin}^a = \langle \pi_d, \sigma_d \rangle$ be the state reached after completing the assistive action σ_d. This implies that there exists a finite sequence of actions $\rho_d \in \rho^a$ that will bring the agent from $\pi_{\mathcal{G},init}^a$ to $\pi_{\mathcal{G},fin}^a$, with ρ^a being the infinite sequence of actions associated with a's plan (τ^a). Note that the last action of ρ_d will be σ_d. For each $\sigma_i \in \rho_d$, there exists $T_{\sigma_i} = T_{\mathcal{G}}^a(\sigma_i) < \infty$, which is the estimated duration of σ_i given by the wFTS of a. Suppose each action is subject to a delay d_{σ_i} such that $0 \leq d_{\sigma_i} < \infty$. This means the effective time required to complete σ_i would be $T_{\sigma_i} + d_{\sigma_i}$.

We can now calculate the total time before starting the execution of σ_d, denoted as $T_S^a = \sum_{i=0}^{|\rho_d - 2|} (T_{\sigma_i} + d_{\sigma_i}) < \infty$. After T_S^a, agent a will start σ_d and at time $t + T_S^a$, it will send **Ready**^{a}. The same result can be reached for a_i.

Consider $T_S = \max_a\{T_S^a\} < \infty$ with $a \in \mathcal{S}$. This implies that at time $t + T_S$, all agents will have sent **Ready**^{a}. Moreover, a_i will be ready to perform $\sigma_c^{a_i}$ and consequently send **Start**, effectively synchronizing all the agents. □

4.2.8 Proof of Correctness

In this section we have described the core of the work *i.e.,* the collaboration and synchronization algorithm developed to solve Problem 1 the only thing that remains to do is to prove its correctness and the ability to solve the problem, this is what this section is for.

Theorem 4.2. *The collaboration and synchronization algorithm presented in Section 4.2 is able to solve Problem 1.*

Proof. We can divide this proof into two parts: first, we will show how each agent a_i satisfies its locally assigned task φ^{a_i}; second, we will prove how all the actions involved in a collaboration will start at the same time.

1. **Local Task Satisfaction**: Consider the initial plan of a_i, denoted as $\tau_{init}^{a_i}$. As mentioned in Remark 4.8, the planner developed at KTH guarantees the satisfaction of φ^{a_i}. The initial plan is then modified through the insertion of detours to accommodate both assisting actions and to delay collaborative actions. Thanks to Assumption 1, we are guaranteed that the insertion of detours will not void the satisfaction of φ^{a_i}. Moreover, Assumption 2, ensures that all collaborative actions will be executed, allowing all agents with these types of actions in their plans to advance. In conclusion, the algorithm guarantees the satisfaction of each locally assigned task.

2. **Synchronization**: Consider t as the time when the agents received a confirmation message then there will exist a finite time $T_S > 0$ such that at time $t + T_S$ all the actions involved in the collaboration related to the confirmation message will simultaneously start, by Lemma 4.1.

□

4.3 Model Predictive Controller

In Section 2.4, we defined the generic formulation for an MPC controller using a generic cost function without specifying the constraints. In this section, we will first explain the setup and constraints used for collision avoidance. Next, we will provide the specific formulation used in this work. Finally, we will specify the values of the parameters used to tune the controllers for both the Turtlebots and the Rosies.

4.3.1 Collision Avoidance

Unlike the algorithm presented in the previous section, where the agent was not required to know the position of any obstacles present in the workspace, the MPC controller needs to know their location to avoid them.

Definition 4.13. Let $\mathcal{O}$ be the set of obstacles in the workspace. We define a generic obstacle $o_i \in \mathcal{O}$ as a circular region in the workspace, represented by the tuple:

$$o_i = (x^{o_i}, y^{o_i}, R^{o_i}) \tag{4.21}$$

where:

- x^{o_i}, y^{o_i} are the planar coordinates of the center of the obstacle,
- R^{o_i} is the radius associated with the circular region representing o_i.

Moreover, we can divide the obstacles into two groups:

- Static Obstacles: These obstacles do not move, so their coordinates remain constant during the execution.
- Dynamic Obstacles: These obstacles change their position during execution and can represent other agents or people in the workspace. Their positions are updated at each iteration of the controller by the Motion Capture (MoCap).

The definition of agents as dynamic obstacles implies that each agent a_i will also be modeled as a circular region:

$$a_i = (x^{a_i}, y^{a_i}, R^{a_i}) \tag{4.22}$$

where the coordinates are provided by the MoCap, and the radius is a design parameter defined in Table 3.1 for the Rosies and in Table 3.2 for the Turtlebots.

Remark 4.18. It is fundamental to note that this implementation of the MPC controller considers only ground robots, as these are the robots available for the experiments. Consequently, we consider only the planar coordinates.

From now on, we will consider a_i as the agent using the controller and $o_i \in \mathcal{O}$ as the obstacle. Before providing the mathematical definition of the constraint, we need to define what will be considered a collision. For this purpose, a collision is defined as an event where the circular region representing o_i intersects with the region representing a_i. This can be mathematically expressed by the following equation:

$$h(\boldsymbol{x}^{a_i}) = h(x^{a_i}, y^{a_i}, \theta^{a_i}) = \sqrt{(x^{a_i} - x^{o_i})^2 + (y^{a_i} - y^{o_i})^2} - (R^{a_i} + R^{o_i}) \tag{4.23}$$

$h(\boldsymbol{x}^{a_i})$ defined in Equation (4.23) represents a valid Control Barrier Function (CBF), and its safe region is defined by $h(\boldsymbol{x}^{a_i}) \geq 0$, indicating that no collision occurs. The final step necessary to define the constraint that will be added to the MPC is the definition of the extended class $\mathcal{K}$ function $\alpha(x)$. For this implementation, we will consider:

$$\alpha(x) = kx \tag{4.24}$$

with the constant $k = 1.5$.

Definition 4.14. The constraint added to agent a_i to avoid the collision with obstacle o_i is:

$$L_f h(\boldsymbol{x}^{a_i}) + L_g h(\boldsymbol{x}^{a_i})\boldsymbol{u}^{a_i} + \alpha(h(\boldsymbol{x}^{a_i})) \geq 0 \tag{4.25}$$

where,

- $h(\boldsymbol{x}^{a_i})$ as defined in Equation (4.23)
- $\alpha(x) = 1.5x$

Remark 4.19. It is fundamental to note that we do not perform all the calculations of the Lie derivatives ourselves, as this will be handled by the optimization software used, namely CasADi [26].

4.3.2 MPC Formulation

Now that the collision avoidance constraints have been defined, we can move on to the detailed definition of the optimization problem used in the MPC controller. We will start by defining the objective of this controller and the corresponding objective function used in the optimization problem, along with the relative values for the tuning parameters. Afterwards, we will present the complete formulation of the problem by defining the necessary quantities considered in the experimental setup, which will be further analyzed in the following chapter.

The main objective of this controller is to reach a desired ROI, so the first step will be defining them; we will consider circular ROI so a similar definition as the one given for the agents and obstacles applies namely, we can model the j-th ROI of agent a_i ($\pi_j^{a_i} \in \Pi_{\mathcal{M}}^{a_i}$) as the tuple:

The main objective of this controller is to reach a desired ROI. We will consider circular ROIs, so a similar definition to the one given for agents and obstacles applies. Specifically, we can model the j-th ROI of agent a_i ($\pi_j^{a_i} \in \Pi_{\mathcal{M}}^{a_i}$) as the tuple:

$$\pi_j^{a_i} = (x^{\pi_j}, y^{\pi_j}, R^{\pi_j}) \tag{4.26}$$

The objective can be translated into a Cartesian regulation problem, where we want to drive the state of the agent to a desired position in the workspace. Note that we do not give any weight to the orientation of the agent. The problem is to design a cost function suitable for this task.

Definition 4.15. The cost function used to tackle the regulation problem is:

$$\sum_{i=0}^{N-1} (\boldsymbol{x}_{k+i}^{a_i} - \boldsymbol{x}^{\pi_j^{a_i}})^T Q (\boldsymbol{x}_{k+i}^{a_i} - \boldsymbol{x}^{\pi_j^{a_i}}) + {\boldsymbol{u}_{k+i}^{a_i}}^T R \boldsymbol{u}_{k+i}^{a_i} \tag{4.27}$$

where:

- $\boldsymbol{x}_{k+i}^{a_i}$ is the pose of agent a_i at instant $k+i$,
- $\boldsymbol{x}^{\pi_j^{a_i}} = [x^{\pi_j}, y^{\pi_j}, 0]$ is the pose associated to the target ROI,
- $\boldsymbol{u}_{k+i}^{a_i}$ is the input of agent a_i at instant $k+i$,
- Q and R are square matrices used for tuning the controller. The values for each application will be provided at the end of this section.

Definition 4.16. The MPC optimization problem tailored to the Cartesian regulation problem and collision avoidance can be defined as:

$$\begin{aligned}
\min_{U} \quad & \sum_{i=0}^{N-1} (\boldsymbol{x}_{k+i}^{a_i} - \boldsymbol{x}^{\pi_j^{a_i}})^T Q (\boldsymbol{x}_{k+i}^{a_i} - \boldsymbol{x}^{\pi_j^{a_i}}) + {\boldsymbol{u}_{k+i}^{a_i}}^T R \boldsymbol{u}_{k+i}^{a_i} \\
\text{subject to:} \quad & \boldsymbol{x}_{k+i+1} = \boldsymbol{x}_{k+i} + f(\boldsymbol{x}_{k+i}, \boldsymbol{u}_{k+i}) \cdot T, \quad i = 0, \dots, N-1 \\
& \boldsymbol{x}_{k+i}^{a_i} \in \mathcal{X}, \quad i = 1, \dots, N \\
& \boldsymbol{u}_{k+i}^{a_i} \in \mathcal{U}, \quad i = 0, \dots, N-1 \\
& \boldsymbol{x}_k^{a_i} \text{ given} \\
& L_f h(\boldsymbol{x}_{k+i}^{a_i}) + L_g h(\boldsymbol{x}_{k+i}^{a_i}) \boldsymbol{u}_{k+i}^{a_i} + \alpha(h(\boldsymbol{x}_{k+i}^{a_i})) \geq 0 \\
& \qquad\qquad \forall o_i \in \mathcal{O}, \quad i = 1, \dots, N
\end{aligned} \tag{4.28}$$

where:

- $f(\boldsymbol{x}, \boldsymbol{u})$ represent the kinematic model of the robot, namely Equation (3.1) for the Rosies and Equation (3.2) for the Turtlebots.
- $\mathcal{X}$ represents a rectangular workspace limited by the coordinates given in Table 5.1. We further restrict this workspace by adding R^{a_i} to the minimum values and subtracting it from the maximum values to ensure that the whole agent is inside the workspace and not just its center of mass.
- $\mathcal{U}$ represent the input set specified in Table 3.1 for the Rosies and Table 3.2 for the Turtlebots.
- $h(\boldsymbol{x}^{a_i})$ is the CBF defined in Equation (4.23).

The parameters used for tuning the controller are as follows:

- Prediction horizon $N = 30$ steps.
- Sampling time $T = 0.1$s
- State weight matrix $Q = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ for the Rosies and $Q = \begin{bmatrix} 25 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ for the Turtlebots.
- Input weight matrix $R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ for the Rosie and $R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ for the Turtlebots.

Remark 4.20. The main idea behind the matrix Q is to give equal weight to both the x and y coordinates while completely disregarding the orientation, as the objective is to solve a Cartesian regulation problem.

Remark 4.21. It is worth noting that the controller developed will not drive a_i exactly to the center of $\pi_j^{a_i}$. Instead, the controller will be stopped once $\boldsymbol{x}^{a_i}$ is inside a circular region with the same center as $\pi_j^{a_i}$ and a radius equal to $0.7R^{\pi_j}$. This strategy not only ensures faster transition times between regions but also allows multiple agents to be in the same region simultaneously if the region is large enough.

CHAPTER 5 RESULTS AND ANALYSIS

This chapter provides an overview of the results obtained from testing the algorithm. In Section 5.1, we will analyze the setup used to conduct the tests in depth. Then, in Section 5.2, we will examine the results obtained, both in terms of complexity and by comparing nominal and real-world simulations. Finally, in Section 5.3, we will analyze the reliability and validity of the data obtained.

5.1 Experimental Setup

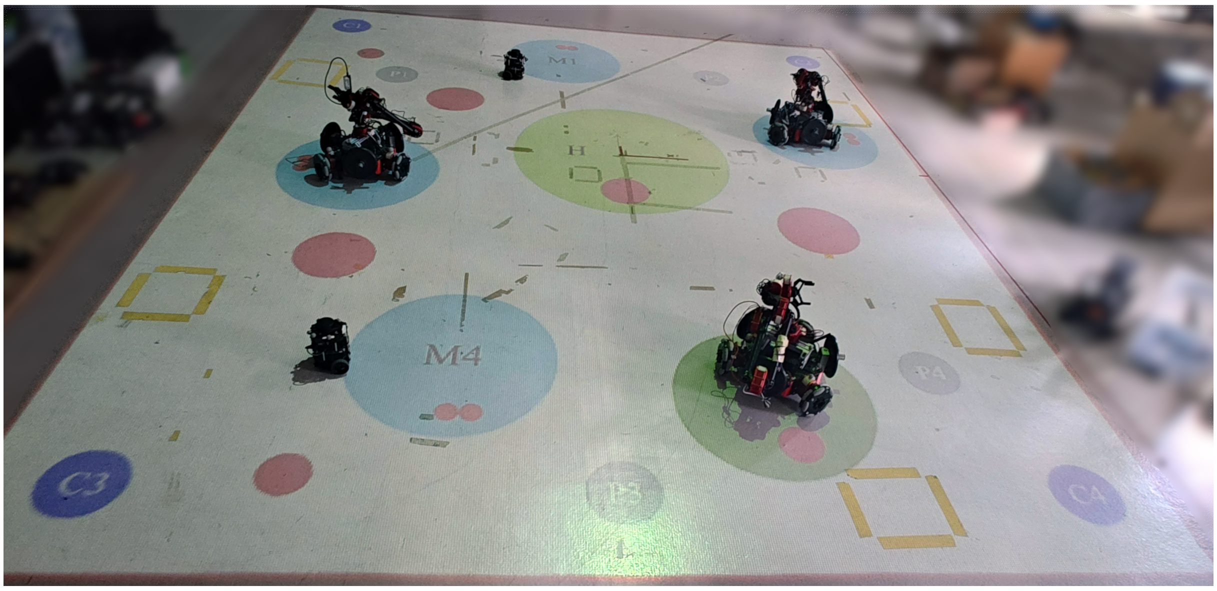

Figure 5.1: The experimental workspace as used during testing at Smart Mobility Lab (SML)

To evaluate the effectiveness of the proposed algorithm, we conducted simulations with a group of 5 robots composed of:

- 3 Rosies called rosie$_0$, rosie$_1$, and rosie$_2$.
- 2 Turtlebots called turtlebot$_1$, and turtlebot$_2$.

The robots move within the arena at SML,*i.e.*, the area of the laboratory where reliable and valid measurements can be obtained from the Motion Capture (MoCap). This arena contains various obstacles and Region Of Interests (ROIs) that the robots need to navigate and interact with. An abstraction of the workspace is shown in Figure 5.2. It is important to note that the locations of the ROIs in Figure 5.2 are indicative. The workspace has been projected onto the arena floor, and some keystone distortion issues were encountered, as it is possible to see in Figure 5.1. A detailed definition of the ROIs and their locations in the workspace can be found in Table 5.2. The boundary of the workspace is defined in Table 5.1.

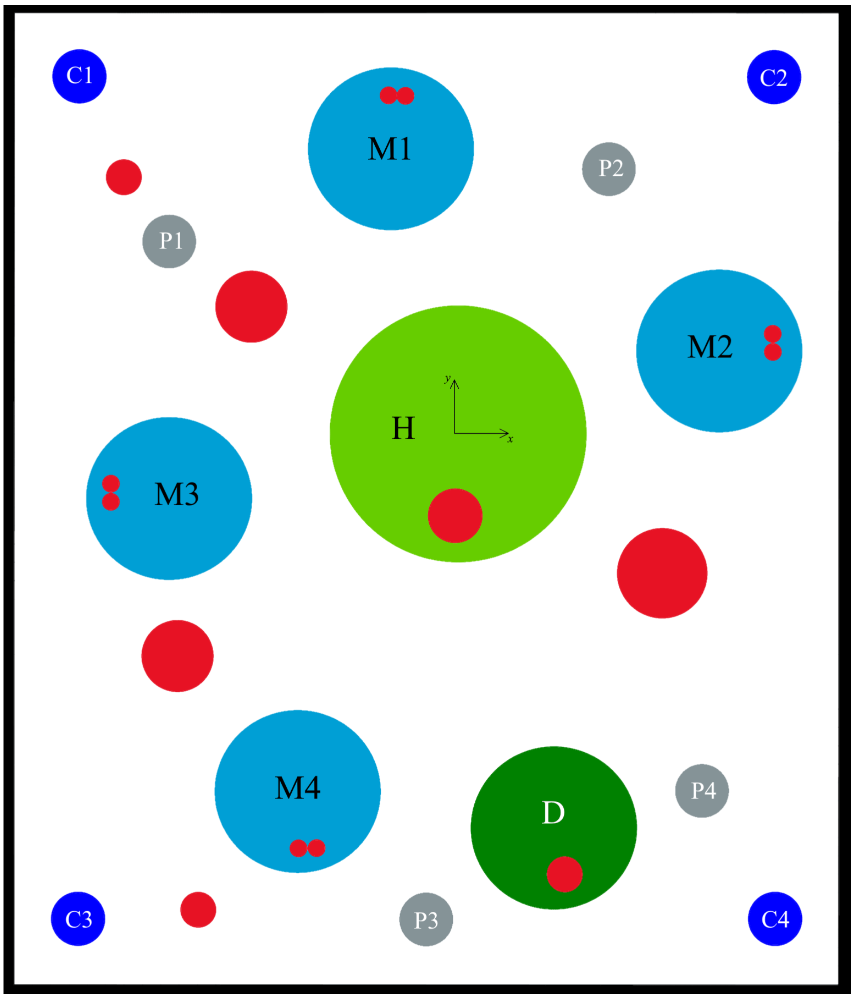

Figure 5.2: Abstraction of the experimental workspace where the ROIs are depicted with different colors, while the obstacles are depicted in red.

Maximum x Coordinate	1.88m
Minimum x Coordinate	-2.29m
Maximum y Coordinate	2.30m
Minimum y Coordinate	-2.87m

Table 5.1: Boundary of the Workspace

5.1.1 Motion wFTS

In this subsection, we will describe the weighted Finite Transition Systems (wFTSs) associated with each group of robots. The nominal transition time is calculated based on the distance between the centers of the ROIs, without considering any obstacle. The agents are allowed to move from any known ROI to any other known ROI. We will list all the regions known by each group of agents, and in Table 5.2, all the ROIs will be defined according to Equation (4.26) based on the measurements taken in the experimental setup.

- ROIs for the Rosies:
  - *Manipulation* regions ($M1 - 4$): these regions are depicted in light blue in Figure 5.2. They represent areas where the agents perform manipulation tasks.
  - *Harvesting* region (H): this region is depicted in light green in Fig. 5.2. It represents the area where the robot harvests grapes and places them into the basket until it is full.
  - *Delivery* region (D): this region is depicted in dark green in Fig. 5.2. It represents the area where the full bucket is delivered and an empty bucket is provided to the agent.
- ROIs for the Turtulebots:
  - *Connection* regions ($C1 - 4$): these regions are depicted in dark blue in Fig. 5.2. They represent areas where the robots perform connection checks.
  - *Patrol* regions ($P1 - 4$): these regions are depicted in grey in Figure 5.2. They represent areas where the robots take snapshots of the surroundings to check for possible dangers.

ROI	x **Coordinate** [m]	y **Coordinate** [m]	**Radius** [m]
$M1$	-0.31	1.53	0.43
$M2$	1.38	0.43	0.43
$M3$	-1.50	0.34	0.43
$M4$	-0.86	1.88	0.43
H	0.00	0.00	0.70
D	0.44	-2.07	0.42
$C1$	-1.96	1.95	0.14
$C2$	1.71	1.93	0.14
$C3$	-1.98	-2.53	0.14
$C4$	1.56	-2.55	0.14
$P1$	-1.49	1.03	0.14
$P2$	0.82	1.42	0.14
$P3$	-0.22	-2.55	0.14
$P4$	1.2	-1.88	0.14

Table 5.2: Definition of ROIs based on the experimental setup

5.1.2 Action Model

In this subsection, we will describe the set of non-movement related actions that each of the two groups of robots can perform, according to Definition 4.4. The action model for Rosies is detailed in Table 5.3, while the action model for Turtlebots is detailed in Table 5.4. The

Action	**Type**	**Cond**	**Dura** [s]
load	collaborative	H	8
help_load	assistive	H	8
manipulate	local	$M1$, $M2$, $M3$, $M4$	7
delvier	local	D	6
harvest	local	H	15
None	local		1

Table 5.3: Action model for the Rosies

Action	**Type**	**Cond**	**Dura** [s]
check_connection	collaborative	$C1$, $C3$	7
help_check_connection	assistive	$C2$, $C4$	5
patrol	local	$P1$, $P2$, $P3$, $P4$	6
None	local		1

Table 5.4: Action model for Turtlebots

final element necessary to fully define the action model is the dependence function, which can be described as follows:

- Depd$($*load*$) = ($*help_load*$, H)$.
- Depd$($*check_connection*$) = ($*help_check_connection*$, \{C2, C4\})$.

5.1.3 Task Specification

Lastly, this subsection will revolve around the Linear Temporal Logic (LTL) task definition for the different agents, moreover, we in Table 5.5 will define the starting ROI. This last step will allow us to fully define the setup used in the simulations.

The LTL tasks assigned to the agents are as follows:

$$\varphi^{\text{rosie}_0} = \Box\Diamond(manipulate \wedge M1 \wedge \Diamond(manipulate \wedge M2)) \quad (5.1)$$

The recurring task assigned to rosie$_0$ involves the manipulation of objects in ROI $M1$, then moving to $M2$ and manipulating objects there.

$$\varphi^{\text{rosie}_1} = \Box\Diamond(manipulate \wedge M3 \wedge \Diamond(manipulate \wedge M4)) \quad (5.2)$$

The recurring task assigned to rosie$_1$ involves the manipulation of objects in ROI $M3$, then moving to $M4$ and manipulating objects there.

$$\varphi^{\text{rosie}_2} = \square\lozenge(harvest \wedge \lozenge(load \wedge \lozenge(deliver))) \tag{5.3}$$

The recurring task assigned to rosie$_2$ involves harvesting grapes, which can only be done in H, then completing the collaboration required to load the full basket, and finally moving to D to deliver it.

$$\begin{aligned}\varphi^{\text{turtlebot}_1} = {} & \lozenge(patrol \wedge \lozenge(check\_connection \wedge C1))\wedge \\ & \square\lozenge(patrol \wedge P1 \wedge \lozenge(patrol \wedge P2))\end{aligned} \tag{5.4}$$

The task assigned to turtlebot$_1$ involves an initial phase where it must patrol one of the designated ROI. Subsequently, it must check the connection in $C1$. Finally, its recurring task begins, which entails periodically patrolling $P1$ and then $P2$.

$$\begin{aligned}\varphi^{\text{turtlebot}_2} = {} & \square\lozenge(patrol \wedge P3 \wedge \lozenge(patrol \wedge P4\wedge \\ & \lozenge(check\_connection \wedge C3)))\end{aligned} \tag{5.5}$$

The recurring task assigned to turtlebot$_2$ involves first patrolling $P3$ and then $P4$, and finally moving to $C3$ to check the connection.

Agent	**Starting ROI**
rosie$_0$	$M1$
rosie$_1$	$M3$
rosie$_2$	D
turtlebot$_1$	$P1$
turtlebot$_2$	$P3$

Table 5.5: Starting ROIs for the agents involved in the experimental setup

The only remaining aspect to completely describe the tasks is to define `Choose_ROI`, the function responsible for selecting the region in which an assisting action should take place. Its behavior for the specific application can be summarized as follows:

- *help_load* is available only at H, so this will be the ROI selected.
- *help_check_connection* has multiple options, but we want the agents performing this assisting action to be located in the opposite corner to the agent requesting the collaboration. Specifically:
  - If *check_connection* will be executed in $C1$, then `Choose_ROI` will select $C4$ as the ROI for *help_check_connection*.
  - If *check_connection* will be executed in $C3$, then `Choose_ROI` will select $C2$ as the ROI for *help_check_connection*.

5.2 Results Analysis

In this section, we will analyze the major results obtained from the developed algorithm. First, we will present the gains in terms of complexity. Then, we will analyze and compare the results between Robot Operating System 2 [1] (ROS2) simulations and the experimental results obtained at SML.

5.2.1 Complexity Reduction

When evaluating the overall complexity of the algorithm, the two most computationally expensive operations are the calls to the `Dijkstra` algorithm and the solution of the Mixed Integer Programming (MIP). While for the first case, we will provide numerical results based on the experimental setup, for the second we will refer to the considerations expressed in 4.15 since, due to time constraints, we could not produce numerical results. Anyway, it is notable to mention that in the scenario considered solving the MIP would require approximately $10 - 20$ms, so it is compatible with the real-time requirement to be able to run the algorithm in the field experiments.

The computational complexity of the `Dijkstra` algorithm is $O(V^2)$, where V indicates the number of vertices in the graph, in our case, the number of states of $\mathcal{T}_{\mathcal{G}}$. This number is reduced compared to previous approaches by using the ROI representation for $\mathcal{T}_{\mathcal{M}}$ instead of a more typical grid structure.
Consider the structure of the workspace, represented by a rectangle of width 4.17m and height 5.17m (Table 5.1), the radius of the Rosies 0.33m (Table 3.1), and the radius of the Turtlebots 0.10m (Table 3.2). To fully partition the workspace with a grid structure where each robot can fit inside, we would need a 6×7 grid for the Rosies and a 20×25 grid for the Turtlebots. The reduction in the number of states for $\mathcal{T}_{\mathcal{M}}$ is highlighted in Table 5.6. Another significant factor

Agent	**States in $\mathcal{T}_{\mathcal{M}}$**	**States with Grid**	**Reduction**
Rosie	6	42	85.7%
Turtlebot	8	500	98.4%
Average Reduction			92.1%

Table 5.6: Computational gains derived by the use of the $\mathcal{T}_{\mathcal{M}}$ instead of a grid structure

reducing the complexity when applying the `Dijkstra` algorithm is the use of the wFTS $\mathcal{T}_{\mathcal{G}}$ instead of the Product Büchi Automaton (PBA) $\mathcal{A}_{\mathcal{P}}$, which was used in previous approaches to guarantee the satisfaction of locally assigned tasks. The full extent of the reduction in the number of states for each agent and assigned task is shown in Table 5.7. It is fundamental to remark, as shown in Table 5.7, that the more complex the task assigned to an agent the greater the overall reduction we obtain. From the data in Table 5.7 and Table 5.7, it is clear that our efforts to reduce computational complexity have been successful, achieving the desired complexity reduction. Another important remark

Agent	States in $\mathcal{T}_{\mathcal{G}}$	States in $\mathcal{A}_{\mathcal{P}}$	Reduction
rosie$_0$	14	56	75.0%
rosie$_1$	14	56	75.0%
rosie$_2$	14	126	88.9%
turtlebot$_1$	16	176	90.9%
turtlebot$_2$	16	144	88.9%
Average Reduction			83.7%

Table 5.7: Computational gains derived by the use of the wFTS instead of the PBA

5.2.2 ROS2 Simulations

After analyzing the computational complexity reduction, we now turn our attention to the simulations carried out under nominal conditions with ROS2. An extensive simulation is shown in Figure 5.3. In this figure, we see all the non-movement related actions executed by the agents (with $None$ not shown to avoid cluttering the plot). Different colors are used to differentiate the types of actions: blue for local actions, green for collaborative actions, and red for assistive actions. Additionally, a star indicates the time when a collaboration actually started. A detailed

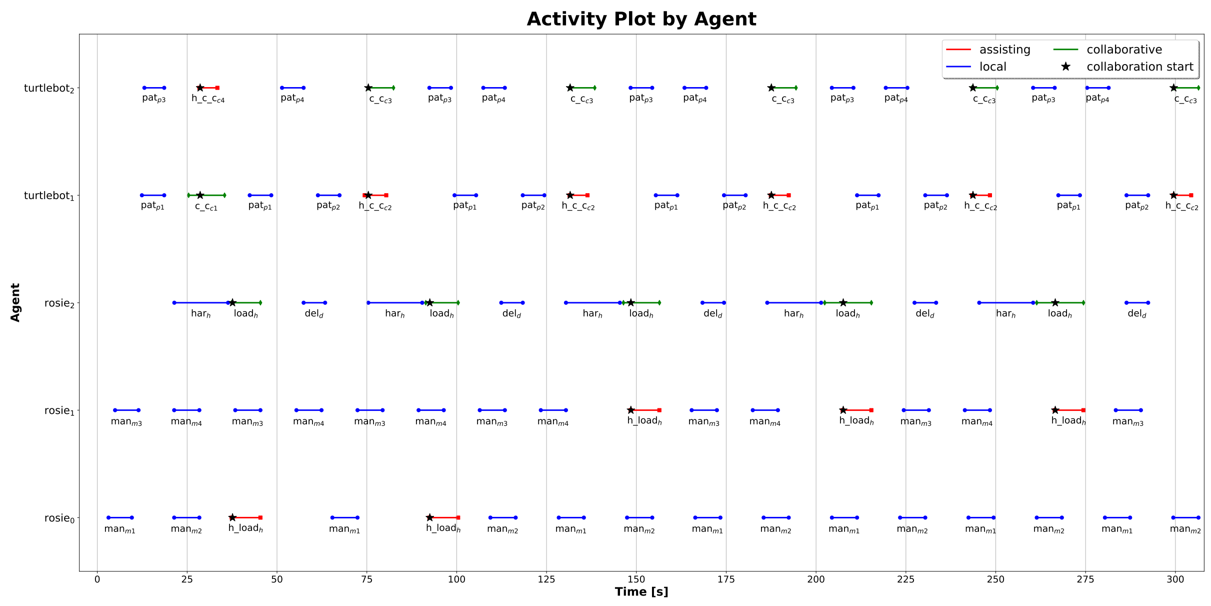

Figure 5.3: Results of the simulation performed in ROS2 under nominal conditions. Only non-movement related actions are shown. The subscripts under the action names indicate the regions where these actions were completed.

analysis of the sequence of actions completed by each agent reveals that, as expected from the theoretical results, the algorithm guarantees the satisfaction of the LTL task assigned to each agent, while also allowing for the insertion of assistive actions.

For example, consider the two Turtlebots. turtlebot$_1$ initially patrols $P1$ since it is the least expensive ROI (initial ROI) where it can perform its task. Later, it requests help for a connection check in $C1$, prompting turtlebot$_2$ to adjust its initial plan and reach $C4$ to assist turtlebot$_1$. An important observation from this collaboration is that turtlebot$_1$ waits for turtlebot$_2$ to be ready before starting the action, demonstrating that the synchronization mechanism is effectively working as intended. After this collaboration, turtlebot$_1$ alternates between patrolling ROIs $P1$ and $P2$, satisfying its assigned task while also accommodating requests to help turtlebot$_2$. A similar analysis can be conducted to verify the satisfaction of tasks assigned to each agent. Turning our attention to the three Rosies, we observe the MIP in action. Sometimes rosie$_0$ is tasked with helping rosie$_2$, while other times rosie$_1$ is selected to allow the agents to progress with their local tasks. This dynamic selection ensures that the workload is balanced and tasks are completed efficiently.

Through this simulation, we have demonstrated that the developed algorithm guarantees task satisfaction and the necessary adaptability to start all actions involved in a collaboration simultaneously.

5.2.3 Field Experiments

Since the nominal simulation presented in the previous subsection yielded positive results, we now move to the analysis of the results obtained from the tests conducted at SML, where we see the Model Predictive Control (MPC) controller in action. The simulation is shown in Figure 5.4. This figure depicts a plot with the same characteristics as the one described for the ROS2 simulations. Analyzing the plot, we can reach the same conclusion regarding the satisfaction of

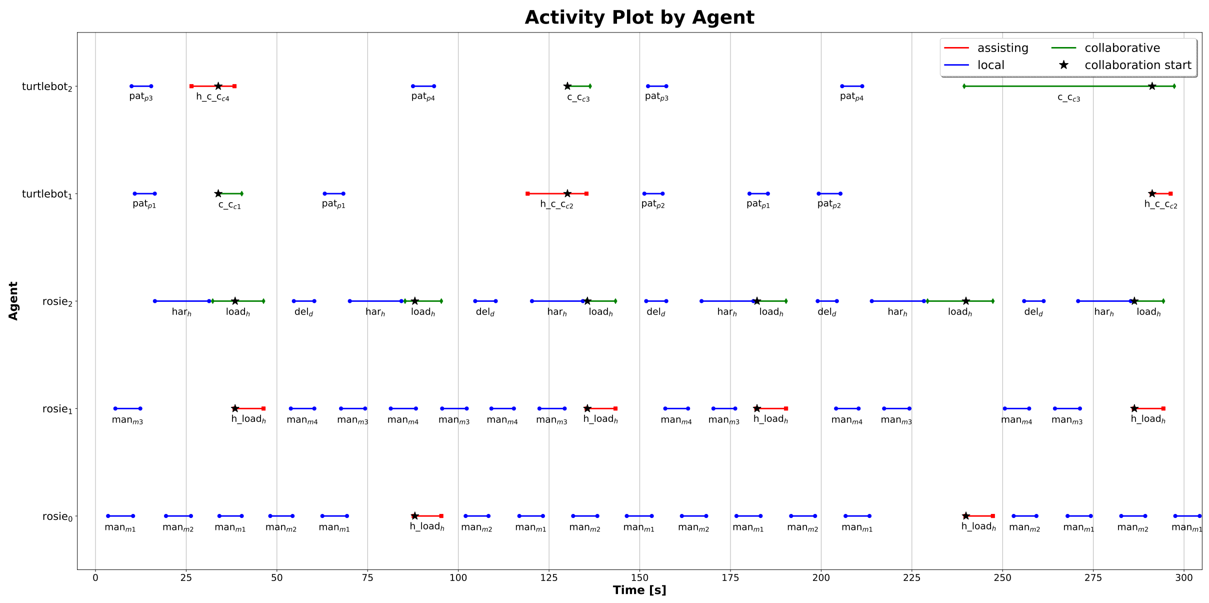

Figure 5.4: Results of the experiments performed at SML. Only non-movement related actions are shown. The subscripts under the action names indicate the regions where these actions were completed.

the tasks. Moreover, observing the behavior of the Rosies, we notice that the time between two actions is slightly longer than in the nominal simulation, this is mainly due to the presence of obstacles which was not considered in the transition time used for the nominal simulations. This indicates that the MPC controller is efficient and successfully avoids obstacles, as will be shown in the video linked at the end of this section.

The main issue arises when considering the Turtlebots. There are significant differences in the times required to move within the workspace compared to the nominal times. This behavior can be traced back to two main reasons: firstly, an imperfect tuning of the MPC controller; and secondly, the software provided by the manufacturer, which proved to be challenging to work with and prone to frequent crashes, such as the one observed Figure 5.4 for turtlebot$_1$ between 225 and 275 seconds. We are currently working on mitigating these problems through extensive tuning of the controller and possible adaptation of the manufacturer's code.

The video showing the experiment conducted at SML is available at `https://youtu.be/PnZjAZy23zI`.

5.3 Validity and Reliability Analysis

The last consideration is the reliability of the methods and data, as discussed in Chapter 2. Given the results of both the ROS2 simulations and the experimental tests, which confirmed the satisfaction of the LTL task assigned to the agents, along with the expected complexity reduction, we can conclude that the methods used to develop the algorithm proved to be reliable.

Regarding the data, although some issues arose while using the Turtlebots, the MPC controller developed proved to be reliable for the Rosies. Moreover, the communication to synchronize the agents worked as expected, confirming that the data obtained is reliable enough to guarantee applicability to real-life systems.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The conclusion chapter serves as a synthesis of the entire thesis, providing a comprehensive summary of the findings, discussing their implications, and suggesting directions for future research. It ties together the theoretical and empirical results discussed throughout the thesis, highlighting the contributions to the field.

6.1 Conclusions

Reflecting on the research objectives presented in Chapter 1, this study aimed to develop and test an algorithm for effective multi-agent coordination in dynamic environments. The following conclusions can be drawn:

1. **Achievement of Objectives**: The algorithm successfully allows multi-agent coordination, ensuring that each agent satisfies its locally assigned recurring LTL task. The developed algorithm dynamically adjusted the plans of individual robots to accommodate collaborative actions without violating their Linear Temporal Logic (LTL) constraints, thereby meeting the primary objectives of the research.

2. **Algorithm Performance**: Through simulations and real-world experiments, the algorithm demonstrated reliability and efficiency in coordinating multiple agents. The integration of Model Predictive Control (MPC) and LTL provided robust solutions to real-time synchronization and collision avoidance.

3. **Implications for Robotics**: The research contributes significantly to the field of robotics by presenting an algorithm that enhances the productivity and sustainability of agricultural practices. The application within the Collaborative Paradigm for Human Workers and Multi-Robot Teams in Precision Agriculture Systems (CANOPIES) project, focusing on automating table-grape vineyard harvesting, showcased the practical benefits and potential scalability of the algorithm.

4. **Theoretical Contributions**: This study advances the understanding of multi-agent systems by integrating LTL and MPC frameworks, offering a novel approach to solving complex collaborative tasks in dynamic environments. The formal proof of the algorithm's correctness underlines its theoretical robustness.

6.2 Limitations

Despite the successes, several limitations were encountered, primarily due to the time constraints imposed on the project:

- **Computational Complexity**: While the algorithm was designed to minimize computational complexity, real-time implementation on commercially available hardware posed challenges. Further optimization is required to enhance performance.
- **Scalability Issues**: The algorithm's scalability to larger systems and more complex environments needs additional validation. Initial results are promising, but extensive testing in diverse scenarios is necessary.
- **Limited Actions and Compatibility**: The current implementation focuses primarily on the movement of robots, with limited consideration for other actions. Moreover, the implementation of the MPC controller was limited to specific robots (Rosies and Turtlebots), although the framework can be extended to other robotic agents.

6.3 Future Work

Building on the findings and addressing the limitations identified, the following future research directions are proposed:

- **Extensive Field Testing**: Conducting extensive field tests in various agricultural settings to validate the scalability and robustness of the algorithm in diverse real-world environments.
- **Improved Control Efficiency**: Enhance the control algorithms for Turtlebots and other robots with similar issues to reduce delays and improve overall performance.
- **Extension to Additional Actions**: Incorporate a wider range of actions beyond movement, enabling robots to perform more complex collaborative tasks.
- **Human-Robot Collaboration**: Investigating the potential for human-robot collaboration, focusing on developing intuitive interfaces and protocols for seamless interaction between human operators and robotic systems.

6.3.1 Codes and scripts

The codes developed for Robot Operating System 2 [1] (ROS2) packages, including the initial planner update to ROS2, the entire implementation of the algorithm, and the MPC controller presented in Chapter 4, as well as the package used for the experiments, will eventually be published on the author's personal GitHub at `https://github.com/DavidePeron19`. For now, the code will remain private as a publication is being prepared based on this thesis work.

6.4 Reflections

Reflecting on the research journey, several insights can be highlighted:

- **Research Process**: The iterative process of developing, testing, and refining the algorithm provided valuable lessons on the importance of flexibility and adaptability in research.
- **Collaborative Efforts**: The success of this research underscores the importance of collaboration among researchers and end-users to address complex challenges and achieve practical solutions.
- **Impact on the Field**: This research has the potential to significantly impact the field of robotics, offering new methodologies for multi-agent coordination and contributing to the advancement of autonomous systems in agriculture.

In conclusion, this thesis has made substantial contributions to the field of multi-agent systems, presenting a robust and scalable algorithm for dynamic environments. The insights gained and the proposed future work provides a solid foundation for continued research and development in this area of robotics.

REFERENCES

[1] “Ros2 - robot operating system 2,” available at: https://www.ros.org/. [Pages xi, 4, 27, 53, and 59.]

[2] Y. Rivero-Moreno, M. Rodriguez, P. Losada-Muñoz, S. Redden, S. Lopez-Lezama, A. Vidal-Gallardo, D. Machado-Paled, J. C. Guilarte, S. Teran-Quintero, Y. Rivero *et al.*, “Autonomous robotic surgery: Has the future arrived?” *Cureus*, vol. 16, no. 1, 2024. [Page 1.]

[3] R. Bogue, “The role of robots in logistics,” *Industrial Robot: the international journal of robotics research and application*, 2024. [Page 1.]

[4] A. Dhaliwal, “The rise of automation and robotics in warehouse management,” in *Transforming Management Using Artificial Intelligence Techniques*. CRC Press, 2020, pp. 63–72. [Page 1.]

[5] V. Verma, M. W. Maimone, D. M. Gaines, R. Francis, T. A. Estlin, S. R. Kuhn, G. R. Rabideau, S. A. Chien, M. M. McHenry, E. J. Graser *et al.*, “Autonomous robotics is driving perseverance rover’s progress on mars,” *Science Robotics*, vol. 8, no. 80, p. eadi3099, 2023. [Page 1.]

[6] J. J. Roldán, J. del Cerro, D. Garzón-Ramos, P. Garcia-Aunon, M. Garzón, J. De León, and A. Barrientos, “Robots in agriculture: State of art and practical experiences,” *Service robots*, vol. 12, no. 2, pp. 67–90, 2018. [Page 1.]

[7] CANOPIES project, “A Collaborative Paradigm for Human Workers and Multi-Robot Teams in Precision Agriculture Systems,” in *https://www.project-canopies.eu/*, european Commission under the H2020 Framework Programme. [Online]. Available: https://www.project-canopies.eu/. [Page 1.]

[8] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 01 2008, vol. 26202649. ISBN 978-0-262-02649-9 [Pages 7, 8, and 9.]

[9] J. R. Büchi, *On a Decision Method in Restricted Second Order Arithmetic*. New York, NY: Springer New York, 1990, pp. 425–435. [Online]. Available: https://doi.org/10.1007/978-1-4613-8928-6_23 [Page 10.]

[10] R. Baran, X. Tan, P. Varnai, P. Yu, S. Ahlberg, M. Guo, W. S. Cortez, and D. V. Dimarogonas, “A ros package for human-in-the-loop planning and control under linear temporal logic tasks,” in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021. doi: 10.1109/CASE49439.2021.9551648 pp. 2182–2187. [Pages 10 and 11.]

[11] P. Gastin and D. Oddoux, “Fast ltl to büchi automata translation,” in *Proceedings of the 13th International Conference on Computer Aided Verification*, ser. CAV ’01. Berlin, Heidelberg: Springer-Verlag, 2001. ISBN 3540423451 p. 53–65. [Page 11.]

[12] M. Guo and D. V. Dimarogonas, “Multi-agent plan reconfiguration under local ltl specifications,” *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015. doi: 10.1177/0278364914546174. [Online]. Available: https://doi.org/10.1177/0278364914546174 [Page 11.]

[13] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, “Optimal path planning for surveillance with temporal-logic constraints,” *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011. doi: 10.1177/0278364911417911. [Online]. Available: https://doi.org/10.1177/0278364911417911 [Page 11.]

[14] C. Belta, B. Yordanov, and E. Gol, *Formal Methods for Discrete-Time Dynamical Systems*. Springer Cham, 01 2017, vol. 89. ISBN 978-3-319-50762-0 [Page 11.]

[15] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN 9780975937730 [Page 13.]

[16] Y. Emam, P. Glotfelter, and M. Egerstedt, “Robust barrier functions for a fully autonomous, remotely accessible swarm-robotics testbed,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE Press, 2019. doi: 10.1109/CDC40024.2019.9029886 p. 3984–3990. [Online]. Available: https://doi.org/10.1109/CDC40024.2019.9029886 [Page 14.]

[17] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs for safety critical systems,” *IEEE Transactions on Automatic Control*, vol. 62, no. 8, p. 3861–3876, Aug. 2017. doi: 10.1109/tac.2016.2638961. [Online]. Available: http://dx.doi.org/10.1109/TAC.2016.2638961 [Page 14.]

[18] J. Tumova and D. V. Dimarogonas, “Multi-agent planning under local ltl specifications and event-based synchronization,” 2016, available at https://people.kth.se/~dimos/autom2016_Tumova_final.pdf. [Page 15.]

[19] G. F. Schuppe and J. Tumova, “Multi-agent strategy synthesis for ltl specifications through assumption composition,” in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020. doi: 10.1109/CASE48305.2020.9216991 pp. 533–540, available at https://kth.diva-portal.org/smash/get/diva2:1614597/FULLTEXT01.pdf. [Page 15.]

[20] M. Guo and D. V. Dimarogonas, “Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks,” *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 797–808, 2017. doi: 10.1109/TASE.2016.2628389 Available at https://people.kth.se/~dimos/pdfs/tase17.pdf. [Pages 15 and 27.]

[21] O. Kupferman and M. Y. Vardi, “Model checking of safety properties,” in *Formal Methods Syst. Design*, vol. 19, no. 3, Nov. 2001, pp. 291–314. [Page 16.]

[22] “Qualisys motion capture,” available at: https://www.qualisys.com/. [Page 19.]

[23] “Hebi ”rosie” mobile omni-directional base with arm and gripper,” available at: https://www.hebirobotics.com/robotic-mobile-platforms. [Page 20.]

[24] “Robotis turtlebot3 burger rpi4,” available at: https://en.robotis.com/shop_en/item.php?it_id=901-0118-201. [Page 22.]

[25] “Raspberry pi 4,” available at: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Page 22.]

[26] “Casadi,” available at: https://web.casadi.org/. [Page 45.]