KTH Electrical Engineering

Hybrid Control of Multi-robot Systems under Complex Temporal Tasks

MENG GUO

Doctoral Thesis
Stockholm, Sweden 2016

KTH Royal Institute of Technology
School of Electrical Engineering
Department of Automatic Control
SE-100 44 Stockholm
SWEDEN

TRITA-EE 2015:79
ISSN 1653-5146
ISBN 978-91-7595-720-3

Akademisk avhandling som med tillstånd av Kungliga Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i elektro- och systemteknik fredag den 22 januari 2016, klockan 10:00, i sal F3, Kungliga Tekniska högskolan, Lindstedtsvägen 26, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Autonomous robots like household service robots, self-driving cars and drones are emerging as important parts of our daily lives in the near future. They need to comprehend and fulfill complex tasks specified by the users with minimal human intervention. Also they should be able to handle un-modeled changes and contingent events in the workspace. More importantly, they shall communicate and collaborate with each other in an efficient and correct manner. In this thesis, we address these issues by focusing on the distributed and hybrid control of multi-robot systems under complex individual tasks.

We start from the nominal case where a single dynamical robot is deployed in a static and fully-known workspace. Its local tasks are specified as Linear Temporal Logic (LTL) formulas containing the desired motion. We provide an automated framework as the nominal solution to construct the hybrid controller that drives the robot such that its resulting trajectory satisfies the given task. Then we expand the problem by considering a team of networked dynamical robots, where each robot has a locally-specified individual task also as LTL formulas. In particular, we analyze four different aspects as described below.

When the workspace is only partially known to each robot, the nominal solution might be inadequate. Thus we first propose an algorithm for initial plan synthesis to handle partially infeasible tasks that contain hard and soft constraints. We design an on-line scheme for each robot to verify and improve its local plan during run time, utilizing its sensory measurements and communications with other robots. It is ensured that the hard constraints for safety are always fulfilled while the soft constraints for performance are improved gradually.

Secondly, we introduce a new approach to construct a full model of both robot motion and actions. Based on this model, we can specify much broader robotic tasks and it is used to model inter-robot collaborative actions, which are essential for many multi-robot applications to improve system capability, efficiency and robustness. Accordingly, we devise a distributed strategy where the robots coordinate their motion and action plans to fulfill the desired collaboration by their local tasks.

Thirdly, continuous relative-motion constraints among the robots, such as collision avoidance and connectivity maintenance, are closely related to the stability, safety and integrity of multi-robot systems. We propose two different hybrid control approaches to guarantee the satisfaction of all local tasks and the relative-motion constraints at all time: the first one is based on potential fields and nonlinear control technique; the second uses Embedded Graph Grammars (EGGs) as the main tool.

At last, we take into account two common cooperative robotic tasks, namely service and formation tasks. These tasks are requested and exchanged among the robots during run time. The proposed hybrid control scheme ensures that the real-time plan execution incorporates not only local tasks of each robot but also the contingent service and formation tasks it receives.

Some of the theoretical results of the thesis have been implemented and demonstrated on various robotic platforms.

Sammanfattning

Denna avhandling fokuserar på distribuerad och hybridstyrning av multi-robotsystem för komplexa, lokala och tidsberoende uppgifter. Dessa uppgifter specificeras av logiska formler rörande robotens rörelser och andra ageranden. Avhandlingen behandlar ett tvärvetenskapligt område som integrerar reglering av nätverkade robotsystem och planering baserad på formella metoder. Ett ramverk för hybridstyrning av flera dynamiska robotar med lokalt specificerade uppgifter presenteras. Fyra huvudscenarier betraktas: (1) robot-planering med motstridiga arbetsuppgifter inom ett delvis okänt arbetsområde; (2) beroende uppgifter för en grupp heterogena och samverkande robotar; (3) relativa rörelsebegränsningar hos varje robot; samt (4) robotar med uppgifter som begärs och bekräftas under körning. Numeriska simuleringar och experiment visas för att validera de teoretiska resultaten.

To Wei.

Acknowledgements

I would like to firstly express my gratitude to my main advisor Prof. Dimos V. Dimarogonas for your persisting support, guidance and encouragement in both my life and work. You introduced me to the world of research with the master thesis project in 2011. I am extremely grateful that you give me the perfect blend of guidance and independence during this journey. I also would like to thank my co-advisor Prof. Karl H. Johansson for your great knowledge and enthusiasm. Your passion for research will continue influencing me later on.

The five years of PhD study at KTH have been the most memorable time in my life so far. Automatic Control Lab has been such a joyful place to live and work. I thank Martin A., Håkan, Martin J., Olle, Kuo-yun, Sebastian, Yuzhe, Niclas B., Niclas E., Burak, Arda, Hamid, Burak, Antonio A., Valerio, Davide, Demia, Patricio, Euhanna, Christian, José, André, Weiguo, Mohamed, Afrooz, Emma, Adam, Bart, Davide, Alessandra, Miguel, Kaveh, Assad, Xinlei, Kin, Rong, Giulio, Bart, Sindri, José Mairton Barros, Pedro L., Pedro P., Adam, Alexandros for being great colleagues. I thank Jana and Dimitris for helpful discussions and guidance. Special thanks to Antonio G., I could not ask for a better office neighbor and gym buddy! Thanks also go to the administrators: Hanna, Kristina, Anneli, Gerd, Silvia and Karin for always being helpful and creating a pleasant working atmosphere. I also thank Niclas B. for helping me on the Swedish abstract and Prof. Henrik Sandberg for reviewing the thesis.

I would like to thank Prof. Magnus Egerstedt for hosting me at the GRITS Lab at Georgia Tech during April - July, 2015. I have had a great and fruitful time there and get to know lots of awesome people. Huge thanks to collaborators in EU RECONFIG project: Alejandro and Michele for great learning experiences on ROS and computer vision and for being such fun companions during our demo trips; George, Panos, Babis for being great hosts at NTUA, Athens.

Life is much more than just work, especially for a PhD :) I thank Viktor and Camille for all the climbing sessions together; Joel and Michael for our enlightening studies; Yalin and Camille for the adventurous hiking trips...

Finally, I would like to thank my family for always believing in me, and my wife Wei for your genuine and everlasting love.

Meng Guo
Stockholm, January 2016

Contents

Chapter 1

Introduction

Autonomous robots such as cars, drones and domestic service robots appear in the popular media more and more frequent. In recent years, the technical development, manufacturing and installation of industrial and domestic robots have been boosted by the unprecedented evolution of digital processing. Robots and embedded computers have become more powerful in terms of speed and capacity, and at the same time more affordable [148]. An essential feature of autonomous robots is that they are expected to comprehend daily tasks specified by non-expert end-users, reason about them, figure out a plan and more importantly execute the plan to accomplish the tasks without or with minimal human intervention.

Wireless communication technology enables autonomous robots to be connected with each other and with internal or external sensors. The emerging Internet of Things [150] will allow robots to have more accurate and up-to-date information about their operation space. Even better, cloud-computing platforms provide them on-demand access to computing and storage resources for an enormous amount of real-time data. These robot-to-infrastructure and robot-to-robot communications should be tailored specifically for task planning and execution, in order to save bandwidth and improve efficiency.

Lastly, a group of coordinated autonomous robots can be more efficient and can achieve more complex tasks than a collection of independent single robots [9]. For instance, three coordinated robots can carry a heavy object together which can not be carried by three independent robots; passengers with close-by destinations can share the same autonomous car through carpooling. The effectiveness of a multi-robot system is closely related to the complexity and flexibility of the underlying coordination scheme. Thus the core problem for any multi-robot application is to design a suitable coordination scheme that can improve the overall performance while keeping the cost and complexity of coordination low.

All issues mentioned above bring the need for a new framework for the modeling, design and analysis of interconnected multi-robot systems under complex individual tasks. In the remainder of this chapter, we provide some motivating applications of this thesis, which is followed by the main contributions and the thesis outline.

Figure 1.1: Some examples of service robots (left to right, top to bottom): Gostai, TUG, Pepper, Baxter and Robot Bear (photo courtesy of Gostai robotics, Aethon, Aldebaran, Rethink robotics and Riken).

1.1 Motivating applications

In this section we provide some real-life applications that motivate the topics addressed in this thesis.

1.1.1 Service robots

Different from industrial robots that are mostly built for fast, reliable and accurate repetitions, domestic service robots on the contrary are designed to be safe, easy-to-use and good at interactions with humans. Since we are living in an aging society, more and more personal caring services are demanded in hospitals and households. Furthermore, surveillance tasks are tedious and potentially dangerous for humans. Compared with human nurses, housekeepers or security guards, autonomous robots are constantly available, reliable and capable of more than one type of tasks. Some well-known commercial products on the market are shown in Figure 1.1, including the Gostai robot [47] designed for telepresence and surveillance tasks for office-like environments; the service robot from TUG [3] which delivers goods such as medication, lab specimens, food and linens in the hospital around the clock; the service robot Pepper [6] which can accompany people, talk to them, and understand their emotions; the user-friendly ReThink Baxter robot [123] which collaborate with small-scale family-business owners in their assembly lines; and the robot Bear [125]

Figure 1.2: Left: a customized guide robot for online users to explore the museum at night; right: a service robot that can perform daily errands (photo courtesy of Wired Magazine and TU/e).

which serves as a nurse assistant and provides personal caring for the elderly. In addition, some customized service robots are designed for more specific purposes. As shown in Figure 1.2, the custom-made guide robot at Tate Britain museum allows any ordinary Internet surfer to explore the masterpieces inside the museum *after dark* by live-streaming the robot's vision with commentary [149]; the service robot Amigo [138] developed at TU/e can be configured to perform various domestic errands at home and hospital environments. It is estimated in [127] that more than 500,000 units of domestic service robots will be sold worldwide before the year 2020. Thus it is reasonable to foresee that autonomous service robots will play more and more important roles in our daily lives.

Now imagine that you have several service robots like those shown in Figure 1.1 in your house and that you want to give them the following commands:

- "*Gostai, take photos of the kids in both bedrooms and send them to me*";
- "*Baxter, go to the kitchen, make a cup of coffee and bring it back to me*";
- "*Pepper, vacuum the living room and turn on the dish washer in the kitchen*";
- "*Robot bear, move the table from the balcony to the study*".

The above task specifications are typical as daily household errands, from which we can easily draw the following characteristics: firstly, they are addressed to each individual robot separately as the human user normally has a clear idea about the distinctive motion and actuation capabilities of these robots; secondly, they are specified as the desired robot actions at certain locations, e.g., "take photos" at "bedrooms", "make coffee" at "kitchen", and "vacuum" the "living room", and these actions should happen in a desired temporal sequence, e.g., first "vacuum the living room" and then "turn on the dish washer"; thirdly, some of the desired actions can be done by the robot itself like "take photos" and "vacuum the room", while some may require collaborations with other robots like "move the table". Most

Figure 1.3: Left: the self-driving car by Google; right: the Amazon's "prime" air delivery service by autonomous drones (photo courtesy of Google and Amazon).

of the existing solutions are designed for rigid and simple task executions, thus incapable of handling complex tasks as listed above but much human intervention is needed. Thus we intend to design a systemic and automated framework that allows the human users to describe these aforementioned tasks in a concise and formal way; enables the robot to synthesize its local motion and action plan, and execute this plan autonomously to fulfill its task; and lastly allows the team of robots to coordinate and collaborate with each other to fulfill all local tasks.

1.1.2 Autonomous cars and drones

Autonomous cars, also known as self-driving cars, are capable of sensing the traffic environment and navigating to the desired destination without human guidance. The last decade has witnessed an increasing interest in autonomous-driving technologies from both academia and industry. Almost every major car manufacturer around the world is developing autonomous or driver assistance technology to upgrade its existing models, e.g., adaptive cruise control, automatic parking, collision avoidance and high-way platooning. The prototype of Google self-driving car as shown in Figure 1.3 has been tested on road for over 1 million miles [46]. Self-driving taxis will begin trails in Japan from next year [32]. It is estimated by IEEE [75] that up to 75% of all vehicles will be autonomous by 2040, which may lead to an extraordinarily efficient and intelligent transportation system. Particularly, this will allow everyone to get around easily and safely with just a push of a button, regardless of the ability to drive. Sometimes we may have complex driving tasks in mind, like:

- "*Eventually arrive home, but on the way pass by any supermarket of this store and the kid's school, always avoid highways*";
- "*Go to any branch of this bank and then that restaurant, park at a close-by parking area where there is free parking*".

These tasks are clearly much more complex than just "going from A to B", but still quite common as our daily driving routine. They all have a temporal characteristic for different locations to visit and there are properties of interest attached to these

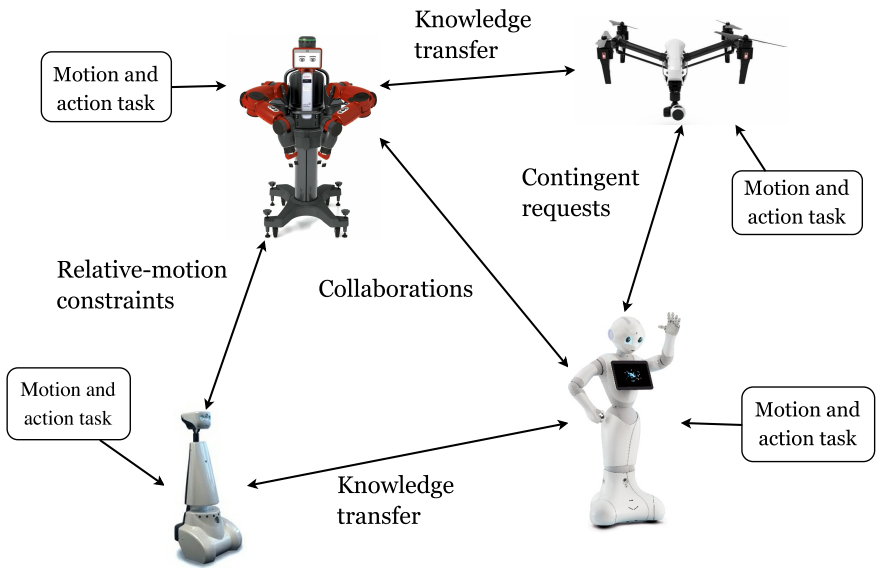


Figure 1.4: A scenario of several autonomous ground and aerial vehicles coexisting within the same workspace. Each of them is assigned a local motion and action task to accomplish. The links represent the constraints and collaborative relations between the robots, which are also the main issues to be addressed in the thesis.

locations, e.g., "any supermarket of this store", "any branch of this bank" and "parking area with free parking". This means that an extra level of logic reasoning and planning is required to find a higher-level plan as the sequence of locations to visit and then merge this plan with the point-to-point navigation technique.

Unmanned aerial vehicles (UAVs), also known as drones, have been used by Amazon as shown in Figure 1.3 to deliver small packages into customers' hands in 30 minutes or less [8], by German railway officials to find and track graffiti drawers [137], and by winemakers in California to monitor grape growth [33]. Even though most drones nowadays can fly stably without the need for human operators to control each motor, they still rely on manual inputs for the destinations and the associated routes. This might be trivial for simple tasks like flying from point A to point B. However, for many of the practical applications mentioned above, the desired task may be much more complex, such as:

- "*Always if requested, pick up the package from the dispatch station and deliver it to the specified destination; otherwise stay at the docking station*";
- "*Surveil locations A, B and C in any order and take videos there. In case of detecting a target, sound alarm and notify the control station*".

The temporal tasks above require the robots to be aware of various real-time events

in the workspace and react to them accordingly, e.g., "pick up the package when requested" and "sound the alarm when detecting a target". In other words, a static plan as a sequence of locations to visit is not adequate anymore, yielding that another challenge here is to monitor events of interest in the workspace and act upon them according to the task specification.

Last but not least, imagine that there are more than one autonomous car and drone being deployed within the same workspace. They potentially become an air-ground collaboration team, e.g., a car may request the drone to visit a certain location that is not reachable by itself but critical for its plan execution; or vice versa, a drone may request a car to inspect the detailed properties of a certain location, due to its limited resolution; and several drones can form a formation and patrol around a car for better security. How to achieve these complex inter-robot interaction and coordination in a systematic, formal and efficient way is another major issue we address in this thesis.

1.2 General problem formulation

Consider a team of dynamical and heterogeneous robots that are capable of moving and performing various actions within a common workspace, as shown in Figure 1.4. Each robot is locally or individually assigned a complex temporal task, which can be dependent or independent on other robots' individual tasks. The general problems we tackle in this thesis include the following aspects:

P1: a local motion and task planning scheme for each robot to fulfill its given task;

P2: an efficient information exchange protocol among the robots regarding real-time features of the workspace;

P3: a coordination strategy for a team with dependent local tasks where some sub-tasks require collaborations among the robots;

P4: a hybrid motion control scheme to incorporate additional relative-motion constraints between the robots;

P5: an on-line plan adaptation algorithm that integrates real-time updates of the workspace model and contingent task requests from other robots.

The overall goal of addressing the above aspects is to always guarantee the satisfaction of all local tasks, while at the same time respecting the various constraints. Particularly, Chapter 3 addresses **P1** under the assumption of static workspace. Chapter 4 solves **P1**, **P2** and **P5** for the multi-robot system under partially-known workspace. Dependent tasks are tackled in Chapter 5 which answers parts of **P1** and **P3**. Then Chapter 6 proposes two different hybrid control strategies that addresses **P1** and **P4**. At last, Chapter 7 solves **P5** partially by considering contingent service and formation tasks that are exchanged among the robots during run time.

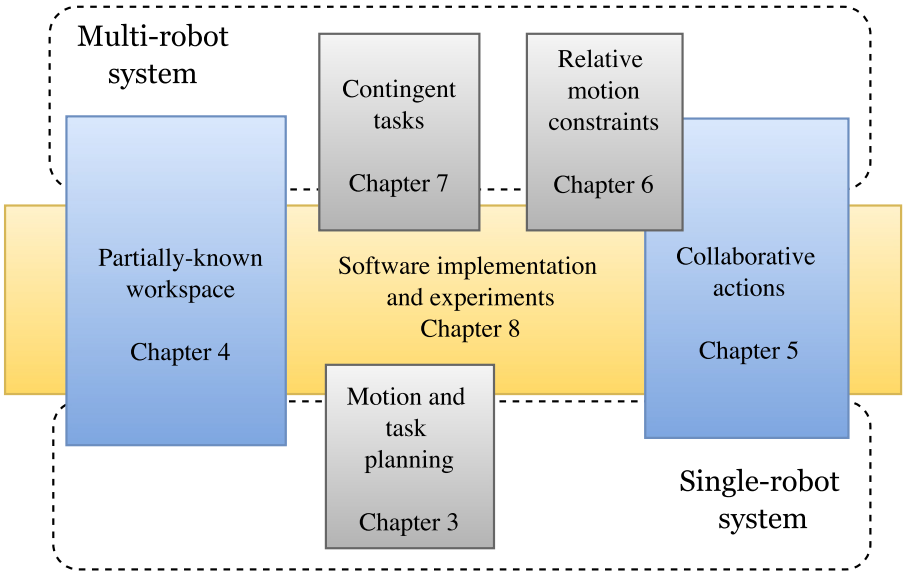


Figure 1.5: Illustrations of the contributions and outline of this thesis.

1.3 Contributions and outline

In this section, we summarize the main contributions and outline of this thesis including the publications each chapter is based upon, as shown in Figure 1.5.

Chapter 2 — Background

Theoretical backgrounds are presented for robot motion and task planning, formal-methods-based verification and synthesis, and multi-robot systems. After that, we review some related work to this thesis.

Chapter 3 — Robot motion and task planning

The nominal problem of robot motion and task planning under a complex local task given as Linear Temporal Logic (LTL) formulas is formulated. Given the robot dynamics and workspace structure, we propose an automated framework that firstly synthesizes the discrete motion plan and then constructs the hybrid controller that drives the robot to execute this plan. It is shown that the robot's resulting trajectory satisfies the given task.

Chapter 4 — Knowledge transfer in partially-known workspace

A team of dynamical robots that coexist within the same workspace is considered in this chapter, where each robot is assigned a local task as LTL formulas for its desired

motion. Moreover, the robots can exchange information through communication links. The workspace is however only partially known to each robot, yielding the nominal solution from the previous chapter inadequate. Thus we first propose a novel approach to handle partially infeasible tasks that contain hard and soft constraints. Then we design a distributed and on-line scheme for each robot to update its system model based on both its real-time measurements and inter-robot communications, verify and adapt its local plan, and modify its hybrid controller accordingly. It is ensured that the hard constraints concerning safety are always fulfilled while the satisfaction for the soft constraints is improved gradually for performance. The control scheme is distributed as only local interactions are assumed. The above results have been published in:

- **M. Guo** and D. V. Dimarogonas. Multi-agent Plan Reconfiguration under Local LTL Specifications. *International Journal of Robotics Research (IJRR)*, 34(2): 218-235, 2015.
- **M. Guo** and D. V. Dimarogonas. Distributed Plan Reconfiguration via Knowledge Transfer in Multi-agent Systems under Local LTL Specifications. *IEEE International Conference on Robotics and Automation (ICRA)*, Hongkong, China, 2014.
- **M. Guo** and D. V. Dimarogonas. Reconfiguration in Motion Planning of Single- and Multi-agent Systems under Infeasible Local LTL Specifications. *IEEE Conference on Decision and Control (CDC)*, Firenze, Italy, 2013.
- **M. Guo**, K. H. Johansson and D. V. Dimarogonas. Revising Motion Planning under Linear Temporal Logic Specifications in Partially Known Workspaces. *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

Chapter 5 — Dependent local tasks with collaborative actions

In order to specify much broader and more practical robotic tasks, a new algorithm to construct the complete model of both robot motion and actions (instead of only motion) is introduced. Based on this model, we specify a local temporal task as the desired robot motion and actions. Furthermore, inter-robot collaborations are essential for many multi-robot applications to improve system-wide capability, efficiency and robustness. Thus we start by constructing a dependency relation among the robots using collaborative and assisting actions. Then we propose a distributed and on-line coordination scheme where each robot sends requests to its neighbors regarding the collaboration it needs, confirms requests from others, and adapts its motion and action plan to fulfill the confirmed collaboration. All decisions are made locally by each robot based on local computation and communication among neighboring robots. This framework is scalable and resilient to robot failures as the dependency relation is formed and removed dynamically according to the

plan execution status and robot capabilities, instead of pre-assigned robot identities. The above results have been published in:

- **M. Guo** and D. V. Dimarogonas. Bottom-up Motion and Task Coordination for Loosely-coupled Multi-agent Systems with Dependent Local Tasks. *IEEE International Conference on Automation Science and Engineering (CASE)*. Gothenburg, Sweden, 2015.

- **M. Guo**, K. H. Johansson and D. V. Dimarogonas. Motion and Action Planning under LTL Specification using Navigation Functions and Action Description Language. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.

Chapter 6 — Inter-robot relative motion constraints

The same multi-robot system is treated in this chapter where each robot is assigned a local LTL task as its desired motion and actions, but under additional continuous relative-motion constrains, such as relative-distance constrains and connectivity maintenance of the underlying communication network. These constraints are closely related to the stability, safety and integrity of the overall multi-robot system, which are otherwise often only assumed, not guaranteed, in related literature. We propose here two different hybrid control approaches to tackle this problem: the first one is based on potential fields and nonlinear control technique to handle scenarios where explicit inter-robot communication is limited and costly; the second approach uses Embedded Graph Grammars (EGGs) as the main tool to specify local interaction rules and switching control modes for each robot. Both approaches are distributed and guarantee the satisfaction of all local tasks, while the relative-motion constraints are respected at all time. The above results have been reported in:

- **M. Guo**, J. Tumova and D. V. Dimarogonas. Communication-Free Multi-Agent Control under Local Temporal Tasks and Relative-Distance Constraints. *IEEE Transactions on Automatic Control (TAC)*, 2015. Conditionally accepted.

- **M. Guo**, M. Egerstedt and D. V. Dimarogonas. Hybrid Control of Multi-robot Systems Using Embedded Graph Grammars. *IEEE International Conference on Robotics and Automation(ICRA)*. Stockholm, Sweden, 2016. Submitted.

- **M. Guo**, J. Tumova and D. V. Dimarogonas. Hybrid Control of Multi-Agent Systems under Local Temporal Tasks and Relative-Distance Constraints. *IEEE Conference on Decision and Control (CDC)*. Osaka, Japan, 2015.

- **M. Guo**, J. Tumova and D. V. Dimarogonas. Cooperative Decentralized Multi-agent Control under Local LTL Tasks and Connectivity Constraints. *IEEE Conference on Decision and Control (CDC)*. Los Angeles, USA, 2014.

Chapter 7 — Contingent service and formation tasks

Two common cooperative robotic tasks are taken into account in this chapter, namely service and formation tasks. The service request is a short-term task provided by one robot to another, while the formation task is a relative deployment requirement among the robots with predefined transient responses imposed by an associated performance function. These tasks are requested and exchanged among the robots during run time, which are unknown a priori. An on-line communication protocol is designed first for each robot to handle these contingent requests and replies. Furthermore, we propose a novel plan adaptation scheme such that the updated plan incorporates not only the robot's original local task but also its newly received contingent tasks. Based on the updated plan, the underlying hybrid controller is then reconstructed such that both local and contingent tasks are satisfied. The above results have been reported in:

- **M. Guo**, C. P. Bechlioulis, K. Kyriakopoulos and D. V. Dimarogonas. Hybrid Control of Multi-agent Systems with Contingent Temporal Tasks and Prescribed Formation Constraints. *IEEE Transactions on Control of Network Systems (TCNS)*, 2015. Submitted.

Chapter 8 — Software implementation and experiments

The software design and implementation for the main algorithms proposed in thesis are described in this chapter. Then we demonstrate some experimental results over different robotic platforms to validate the theoretical contributions.

Chapter 9 — Conclusions and future work

The thesis is concluded with a summary and discussion of the results, where some future research directions are also presented.

Other publications

The following publications are not covered in this thesis, but contain material that motivates the work presented here:

- **M. Guo**, M. M. Zavlanos and D. V. Dimarogonas. Controlling the Relative Agent Motion in Multi-Agent Formation Stabilization. *IEEE Transactions on Automatic Control (TAC)*, 2013.
- **M. Guo** and D. V. Dimarogonas. Consensus with Quantized Relative State Measurements. *Automatica*, 49(8): 2531–2537, 2013.
- **M. Guo** and D. V. Dimarogonas. Nonlinear Consensus via Continuous, Sampled, and Aperiodic Updates. *International Journal of Control (IJC)*, 86(4): 567-578, 2013.

- **M. Guo**, D. V. Dimarogonas and K. H. Johansson. Distributed Real-time Fault Detection and Isolation For Cooperative Multi-agent Systems. *American Control Conference (ACC)*, Montreal, Canada, 2012.

- **M. Guo** and D. V. Dimarogonas. Quantized Cooperative Control Using Relative State Measurements. *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, Orlando, USA, 2011.

Contributions by the author

The contributions of the thesis are mainly the results of the author's own work, in collaboration with the respective coauthors. Experiments presented in Chapter 8 are conducted with partners in EU Reconfig project, Smart Mobility Lab at KTH and GRITS lab at Georgia Tech.

1.4 Notation and acronyms

Notations

$\top$	Boolean constant `True`
$\bot$	Boolean constant `False`
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^N$	N-dimensional vector space over $\mathbb{R}$
$\mathbb{R}^+$	Set of non-negative real numbers
$\mathbf{1}_N$	Vector of ones with length N
$\mathbb{Z}$	Set of integer numbers
$\mathbf{det}(A)$	Determinant of matrix A
$\mathbf{rank}(A)$	Rank of matrix A
$\mathbf{trace}(A)$	Trace of matrix A
$\mathbf{v}[i]$	The i-th element of vector $\mathbf{v} \in \mathbb{R}^N$
$\|x\|_1$	ℓ_1-norm of vector x: $\sum_{i=1}^{N} \|x[i]\|$
$\|x\|$	Euclidean norm of vector x: $\sqrt{x^T x}$
$\lambda_{\min}(A)$	Smallest eigenvalue of matrix A
$\lambda_{\max}(A)$	Maximum eigenvalue of matrix A
$\mathcal{B}(c, r)$	Circular area $\{x \in \mathbb{R}^2 \mid \|x - c\| \leq r\}$
$\otimes$	Kronecker product
$[x]$	Round function of $x \in \mathbb{R}$ and $[0.5] = 1$.
$\mathbf{e}^t$	Natural exponential function
$\ln(x)$	Natural logarithmic function

Acronyms

ADL	Action Description Language
BA	Büchi Automaton
CTL	Computation Tree Logic
EGG	Embedded Graph Grammar
FTS	Finite Transition System
LTL	Linear Temporal Logic
MDP	Markov Decision Process
MTL	Metric Temporal Logic
RTL	Real-time Temporal Logic
NBA	Non-deterministic Büchi Automaton
ROS	Robot Operating System
sc-LTL	Syntactically co-safe LTL
sc-RTL	Syntactically co-safe RTL
wFTS	Weighted Finite Transition System

Chapter 2

Background

This chapter includes a short introduction to the research background of the thesis, including the motion and task planning problem of mobile robots, the model-checking algorithm for verification and synthesis, and the control and coordination problem of multi-robot systems. Then we provide a review of some recent work related to the remaining chapters of the thesis.

2.1 Motion and task planning

To begin with, the term "robot" used in this thesis generally refers to the physical machine that can move around in its workspace, sense the workspace and perform various actions. Robot motion planning is the problem of finding the appropriate actuation signals as the control inputs that can drive the robot from an initial state to a goal state, e.g., move from one location to another, grasp an object of interest or operate a machine. It is also called planning in the continuous statespace [97] or a control problem [28]. Despite the fact that the objective of motion planning is easy to specify, it is not trivial to solve due to the existence of dynamic and kinematic constraints, external disturbances, and execution or measurement noises [95]. Model-based methods have attracted lots of attention where the robot's motion is abstracted and modeled by ordinary differential or difference equations. Given these models, analytic solutions with provable correctness can be found such as the navigation function for sphere workspace and obstacles [87], the potential-field-based control algorithm for workspace with triangular partitions [103], the sampling-based motion planning techniques like probabilistic roadmap (PRM) method [81] and rapidly-exploring random trees (RRT) [79, 96, 99].

On the other hand, robot task planning is the problem of finding a sequence of robot actions that the robot can follow to accomplish a complex task. For instance, suppose the robot is asked to make a cup of coffee: first it needs to figure out a plan as to find a cup, operate the coffee machine, and pour the coffee; then it needs to execute the plan by successfully accomplish each part of the plan in the desired sequence. As proposed in [43], each action can be described by (1) the precondition

that has to be fulfilled before this action can be performed; (2) the effect on the system state after performing this action. The system under consideration such as a robot is traditionally modeled as discrete state-transition systems [22]. By activating one action, the system state can be changed to one or several states. Consequently, the planning problem is transformed to finding the desired sequence of actions or essentially a path that leads the system from the initial state to the goal state. Different representations of the system states may result in various complexities when solving the planning problem [43]. Logic-based representation is currently one of the most popular formalisms used by many planning tools, like STRIPS [37] and PDDL [111]. The solving process is similar to human deliberation that chooses actions by anticipating their outcomes. Besides, learning has also emerged as a power tool [92] nowadays due to the abundance of measurement data and computing resources. The robot can figure out the solution through trial and error, collecting data from previous examples or watching human demonstrations. Another interesting solution is proposed in the form of behavior trees [109, 116]. Unlike finite-state machines, a behavior tree controls the flow of the robot's decision making by considering real-time measurements and the accumulated information about the current status of the robot.

The greatest distinction between robot motion and task planning is that the state space for motion planning is typically continuous and possibly unbounded, thus yielding it impossible to enumerate all the states explicitly as in the task planning. Furthermore, the set of allowed actions is also normally infinite due to the continuous input space, of which the notions of condition and effect are not well defined since a dynamical system may evolve differently under the same input.

2.2 Verification and synthesis based on formal methods

Formal methods are a particular kind of mathematical techniques that are used in software and hardware engineering for specifying and verifying system properties, as part of a more reliable, secure and robust system design [20]. One of the well-known verification techniques is model checking, also called property checking, which exhaustively and automatically checks whether a system model satisfies a given specification that contains the desired system properties [11, 21, 112]. The model can be an actual hardware or software system or its mathematical abstraction as a finite transition system. The specification normally involves desired system properties or security requirements such as absence of bad states, deadlocks or livelocks. The model-checking algorithm returns either success indicating that all possible system behaviors satisfy the property, or a counterexample as one possible behavior that fails the property. Temporal logic languages such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) provide a concise and formal way to specify both propositional and temporal requirements on the system behavior. A method for approximate model checking of stochastic hybrid systems with provable approximation guarantees is proposed in [1]. The stochastic hybrid system is first

approximated by a finite state Markov chain, which is then model checked for probabilistic invariance.

Model-checking algorithms have also attracted much interest for the purpose of synthesis rather than verification. In particular, many recent work integrates classic motion planning algorithms with model-checking techniques to treat complex motion tasks specified by temporal logics [14, 15, 36, 135] such as LTL and CTL mentioned above. Compared with the traditional objective of point-to-point navigation, other control tasks such as reachability, safety and liveness can be described directly by the above logic formulas. Stochastic dynamical systems are considered in [156] by first constructing a finite-state abstraction with a given precision, based on which a control strategy is then synthesized to satisfy LTL specifications. When applying model-checking algorithms for synthesis rather than verification, the desired outcome is distinctively different: the purpose of verification is to find any system behavior that *violates* the property, which can be then used as counter-example to guide the system modification to achieve the desired property; for synthesis purpose we are only interested in the system behavior that *satisfies* the property and mostly is optimized regarding certain cost functions.

2.3 Multi-robot systems

It is seldom that one autonomous robot is a stand-alone system, but it often coexists and interacts with other robots within the same workspace. A multi-robot team under proper coordination can achieve much more complex tasks than a collection of independent single robots, e.g., several robots can collaborate on one task that cannot be done by one robot alone; two robots can switch parts of their tasks to improve mutual efficiency. However, this also introduces some limitations: one robot's behavior is constrained by other robots, e.g., it needs to avoid collision with others while moving around; the common resources in the workspace are shared among all robots. Thus both motion coordination and task coordination are crucial for the performance and functionality of multi-robot systems. Any solution for the above aspects can lie along the spectrum between being centralized and fully decoupled. Centralized solutions typically treat the multi-robot system as a large single system by composing the configuration spaces of all individual robots, while in the fully-decoupled case plans or motion decisions are generated locally based on local communication and coordination with nearby robots during run time.

Regarding the motion coordination of multi-robot systems, many related work can be found on designing motion control strategies to avoid inter-robot collisions when each robot is moving and executing its own plan within the same workspace. Three different motion coordination schemes are proposed in [98] given different assumptions on the robots' initial paths, while at the same optimizing each robot's local performance function. The concept of "reciprocal velocity obstacle" is proposed in [145] to avoid inter-robot collisions while navigating a large number of robots within clustered environments, without the need for explicit communication. [24]

derives a closed-form feedback control law that ensures collision-free paths while navigating each robot to its goal point. In addition, there has been an increasing interest on designing distributed control strategies for system-level goals, like alignment with a team leader [31], relative formation [30] and target containment [77].

On the other hand, regarding the task coordination problem, multi-robot systems are often deployed for the purpose of distributed problem solving [29], where a global task is decomposed into smaller subtasks and then assigned to each robot. This formulation favors a tightly-coupled and top-down approach, where each robot receives commands from the central planner and executes them in a synchronized manner [88]. There is another type of problem formulation assuming that each robot is assigned a local task independently and there is no global task. As a result, each robot acts according to its plan in order to accomplish its own task, and meanwhile it interacts with other robots when necessary, in terms of information exchange and collaborations. This formalism favors loosely-coupled and bottom-up approaches, which resembles many practical systems where each robot has a clear job assignment.

2.4 Related work

In this section, we provide a review of some recent work related to the following chapters and compare the proposed solutions.

Model-checking-based control synthesis

The model-checking algorithms have been used for both control and plan synthesis of dynamic systems. As the first step, various strategies can be found in [7, 44] to construct finite symbolic abstraction of the dynamic system. An automated framework for controller synthesis is proposed in [136] for discrete-time linear systems where the control objective is specified by LTL formulas over the output trajectory, which is more complex than the traditional goal of system stabilization and output regulation. Under a similar setup, [10] adds a quadratic function associated with the cost of satisfying the LTL control objective, which is ought to be minimized. Discrete-time piecewise affine systems are considered in [84, 154, 155], where the control task is specified as LTL formulas over linear predicates of the statespace. Moreover, dynamical systems modeled as Markov Decision Processes (MDP) are considered in [27] where a dynamic control policy is generated automatically to satisfy the LTL task specification with maximal probability. A robustness index is defined and measured for the satisfiability of continuous-time signals under metric temporal logic (MTL) specifications in [35]. Then control objectives specified as MTL formulas are considered in [2] where a generic control strategy is proposed.

On the other hand, [36] firstly proposed the complete framework of automated controller synthesis for autonomous robots under LTL tasks. The robot's motion is abstracted by its dynamical transitions within a partition of the workspace, as a finite transition system. Then a high-level discrete plan as a sequence of regions to visit is synthesized by the model-checking algorithm, which is then implemented by

the low-level continuous controller. Graphical tools are developed in [134] for non-expert end-users to formulate the intended tasks freely, which are then translated to LTL formulas. Robot manipulation tasks given as LTL formulas are addressed in [71] with a multi-layered structure of real-time planning and actuation.

Partially-known workspace

A critical assumption of the model-checking-based formalism for control synthesis mentioned above is that the workspace needs to be perfectly known and correctly modeled by the finite transition system. Then the discrete plan is normally generated off-line and is executed by the robot no matter what has changed in the workspace. As also mentioned in [27], the robot does not react to its actual observations, yielding that this formalism lacks of reconfigurability and real-time adaptation.

Some existing work takes into account the case when a complete representation of the workspace is not available. In [27, 151], the robot's motion within the uncertain workspace is modeled as nondeterministic Markov decision processes (MDP), where the LTL task specification is satisfied by the proposed control strategy with maximal probability. Instead, a two-player General Reactivity GR(1) game between the robot and the environment is constructed in [91, 153] and a receding-horizon control and planning scheme is introduced. Then a winning strategy can be synthesized by exhaustively searching through all allowed combinations of the robot movements and the admissible workspaces.

Instead of aiming for an off-line plan that covers every situation, we propose in our work [59], which can be found in Chapter 4, to create firstly a preliminary plan based on the initially-available knowledge about the robot dynamics and the workspace model. Then while executing this plan, the robot gathers real-time observations about the workspace and feedback for the plan execution status, based on which the plan can be verified and revised to ensure its correctness and feasibility. Similar ideas appear in [104] by locally patching the invalid transitions, which however can not handle changes of the regions' properties. A hybrid motion planning algorithm is proposed in [108] that refines the workspace model iteratively during run time based on sensory updates, under task specifications given as co-safe LTL formulas.

Infeasible task

The initial knowledge of the workspace might be partially incorrect, which may render the intended task infeasible. As also mentioned in [34, 83, 121], the nominal framework presented above simply reports a failure when the given task specification is not realizable. It is desired that users could get feedbacks about why the planning has failed and how to resolve this failure. [34] and [83] address this problem systematically by finding the relaxed specification automaton that is closest to the original one and can be fulfilled by the system. [121] introduces a way to analyze the environment and system components contained in the infeasible specification, and identify the possible cause. The main difference between our work [51] and the

above references is that we put emphasis on how to synthesize the motion plan that fulfills the infeasible task the most, instead of analyzing its infeasible parts.

The problem of synthesizing a least-violating control strategy under a set of safety rules are addressed in [140, 141]. However the level of satisfiability is measured differently from our approach in [51, 56], as discussed in Chapter 4. In particular, we not only measure how many states along the plan violate the safety specifications, but also how much each of those states violates these specifications. [94] proposes a similar solution to [51] but an additional weighting is enforced to the set of atomic propositions. Moreover, we introduce in [51, 56] the notion of soft and hard constraints as two distinctive parts of the task specification, where the hard constraints concern safety requirements and soft constraints are for system performances.

Motion and action planning

To specify robotic tasks of practical interest, it is often necessary to include various actions at different regions. Thus we propose in our work [58] a generic framework that combines the model-checking-based robot motion planning with action planning using action description languages (ADL), as described later in Chapter 5. Some relevant work can be found that integrates the model-checking-based motion planning with action planning. In [132], since the underlying actions can only be performed at fixed regions, the specification can be easily restated as regions to visit. Or independent propositions are created for each action in [91] since the actions can be activated and de-activated at any time. The above approaches are not be applicable if some actions can only be performed when the workspace, or the robot itself has to satisfy certain conditions, or choices have to be made among the allowed actions.

Multi-robot system with temporal tasks

The above motion and task planning framework has also been extended to multi-robot systems that consist of multiple dynamical robots. Many existing work can be found on decomposing a global task specification to bisimilar local ones in a top-down manner, which can then be executed by the robots in a synchronized [85] and partially-synchronized [26] manner. Uncertainties in traveling time are tackled in [142] where an optimal path planning algorithm is proposed for multi-robot systems under a global task specification. [79] formulates the multi-vehicle routing problem under a global LTL task as a mixed integer linear programming (MILP) problem and the derived local plans need to be executed in a synchronized fashion by each robot. There are several drawbacks of this top-down approach: the plans for the whole team are synthesized by the central unit, with a high computational complexity subjected to the combinatorial blowup; it is vulnerable to robot failures and contingencies during run time.

From an opposite viewpoint, we assume that the local task specifications are assigned locally to each robot and there is no specified global task. Namely we consider a team of cooperative robots with different, independently-assigned, even

conflicting individual tasks as considered in our work [51, 52, 56, 57], which can be found in Chapters 4-6. This loosely coupled bottom-up formulation is particularly useful for multi-robot systems where the robots have heterogeneous capabilities and distinctive task assignment. As explained in the remaining chapters, it allows more flexibility of the system since plans and decisions are made locally by each robot [52]; conflicts are resolved and collaborations are coordinated during run time [56, 57]. A similar formalism is adopted in [38] where a decentralized framework for collaborative feasibility verification is proposed, which however does not include the way to resolve potentially-conflicting tasks. A receding-horizon algorithm for distributed coordination is proposed in [139] for multi-robot systems under dependent LTL task specifications.

Relative-motion constraints

Relative-motion constraints often arise in multi-robot systems as the robots coexist within the same workspace, such as collision avoidance [25], network connectivity [76, 158], or relative-velocity constraints [62]. As also pointed out in [62, 157, 158], obeying these constraints is of great importance for the stability, safety and integrity of the overall multi-robot team, which however are often imposed by assumption rather than treated as extra control objectives. We address a version of this problem in our early work [61], where we propose a dynamic leader-follower coordination and control strategy such that the relative-distance constraints among neighboring robots are respected. Later in our work [67, 68] as included in Chapter 6, we improve this scheme by a fully decentralized and communication-free solution that is applicable to low-cost robotic systems equipped with only range and angle sensors, but without communication capabilities. Different from [38, 61] where a satisfying discrete plan is enough, the initial plan synthesis algorithm proposed in [67, 68] minimizes the bottle-neck cost of a satisfying plan. The proposed motion control scheme guarantees almost global convergence and the satisfaction of relative-distance constraints at all time, for an arbitrary number of leaders that are active under different local goals. Most related literature only allows for a single leader [61, 122] or multiple leaders with the same global goal [117, 157]. Additionally, three different local coordination policies are proposed in [67] to incorporate different types of local tasks. Different from [86], these policies are locally applicable to robots without explicit communication modules.

Control with Embedded Graph Grammars

As mentioned above, relative-motion constrains are crucial for the safety and stability of multi-robot systems in general. However, most of related work neglects this aspect, e.g, inter-robot collision is not handled formally in [56, 139] and connectivity of the communication network is taken for granted in most cases [57, 143]. We take advantage of Embedded Graph Grammars (EGGs) in our work [69] as described later in Chapter 6, to specify local interaction and communication rules among the robots,

in order to address simultaneously local LTL tasks and relative-motion constraints. EGGs have been introduced in [113, 114]. They provide a natural framework to encode the robot dynamics, local information exchange and switching control modes in a unified manner. Their successful applications to multi-robot systems can be found in, e.g., coverage control [114], self-reconfiguration of modular robots [120], and autonomous multi-robot deployment [131]. Note that only local interactions or communication are needed for the execution of EGGs, yielding it suitable for large-scale multi-robot systems.

Contingent service and formation tasks

Two common cooperative robotic tasks, namely service and formation tasks, are taken into account in our work [63], of which the details are given in Chapter 7. Service requests are of particular interest as they encapsulate the scenario where one robot needs another robot's assistance on a short-term task. The service requests can only be exchanged among the robots in real-time, which means that each robot has to react to the received service requests in a contingent way. Similar work on real-time reactivity for dynamical systems under temporal tasks can be found in [90], where sensory inputs are included in the General Reactivity GR(1) formulas to take into account possibly dynamical environments, as well as in [91], where similar techniques are applied for various single-robot and multi-robot applications. The local plans of each robot are synthesized off-line to handle all modeled changes in the environment, which is however not feasible in our formulation as the service and formation requests are exchanged in real-time under a non-predefined manner.

Another common cooperative robotic task arises when an robot requests from another one to form a relative configuration to accomplish a collaborative task. We denote such a requirement in [63] as a formation request. Similar ideas of imposing formation constraints for multi-robot systems appeared in [107], where however a global formation task for the whole team is embodied. In the same vein, although the formation control has been extensively studied for multi-robot systems, e.g., in [76, 117, 118], we enforce prescribed performance constraints on the transient response of the formation process in [63]. The prescribed performance control problem was originally studied for high-order MIMO nonlinear single-robot systems [13] and was recently extended to multi-robot systems with single integrator dynamics [80] and nonlinear dynamics [12]. However, the existing prescribed performance control technique is enhanced in our work [63] by: (i) considering transient performance specifications on the overall formation error, in contrast to the relative coordinates approach in [12], thus simplifying the control design; (ii) combining it with the high-level discrete planning such that various control modes are activated according to the real-time execution of the discrete plan.

Chapter 3

Robot motion and task planning

An automated motion and task planning framework for a mobile robot is introduced in this chapter as the nominal solution. In particular, a finite-state transition system is constructed to serve as the discrete abstraction of the robot motion within the workspace. The robot is assigned a complex task which is specified as temporal logic formulas over this transition system. Through the model-checking algorithm, we synthesize firstly a discrete motion plan that satisfies the given task and more importantly minimizes a total cost. Then based on this discrete plan, we construct a hybrid control strategy that drives the robot such that its final trajectory satisfies the task specification.

3.1 Abstraction of robot motion

The robot's motion within a certain workspace is abstracted as a finite-state transition system (FTS) [11]. This FTS is constructed by integrating two aspects: (i) the workspace model; (ii) the robot dynamics and its navigation controller.

3.1.1 The workspace model

The workspace we consider is a bounded n-dimensional space, denoted by $\mathcal{W}_0 \subset \mathbb{R}^n$, within which there exists W smaller regions of interest $\pi_i \subset \mathcal{W}_0$, $\forall i = 1, 2, \cdots, W$. Denote by $\Pi = \{\pi_1, \cdots, \pi_W\}$ the set of all smaller regions. We require that Π is a full partition of the workspace and any two regions $\pi_i, \pi_j \in \Pi$ do not overlap. Namely, $\bigcup_{i=1}^{W} \pi_i = \mathcal{W}_0$ and $\pi_i \cap \pi_j = \varnothing$, $\forall i, j = 1, 2, \ldots, W$ and $i \neq j$.

Atomic propositions are boolean variables that can be either `True` or `False`, which are denoted by $\top$ and $\bot$ for brevity. They are used to express known properties about the state of the robot. Specifically, in order to indicate the robot's position, we define the set of atomic propositions $AP_r = \{a_{r,i}\}$, $i = 1, \cdots, W$, where

$$a_{r,i} = \begin{cases} \top & \text{if the robot is in region } \pi_i, \\ \bot & \text{otherwise,} \end{cases} \tag{3.1}$$

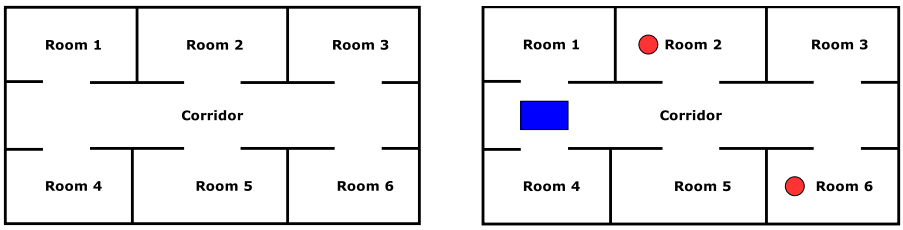


Figure 3.1: Left: the office-like partition of six rooms and one corridor. Right: after adding in AP_p, with two balls (in red) and one basket (in blue).

where $a_{r,i}$ can be evaluated by monitoring the measurements from a localization or positioning system. The requirement that $\bigcup_{i=1}^{n} \pi_i = \mathcal{W}_0$ is important to ensure that the robot's position is tracked at all time. Beside the geometric structure of the workspace, we also would like to express some generic properties within the workspace that are of interest to potential tasks. Denote by $AP_p = \{a_{p,1}, a_{p,2}, \cdots, a_{p,M}\}$ the set of atomic propositions for these properties. For simplicity, we set $AP = AP_r \cup AP_p$ as the set of all atomic propositions.

Definition 3.1. The labeling function $L : \Pi \to 2^{AP}$ maps a region $\pi_i \in \Pi$ to the set of atomic propositions satisfied by π_i, and $a_{r,i} \in L(\pi_i)$ by default, $\forall i = 1, \cdots, W$. ▲

Note that partial satisfaction of a proposition is not allowed. Namely, if only a part of region π_i satisfies $a \in AP$ and the other part does not, then π_i should be split into two regions: one satisfies a and the other does not. AP_p greatly improves the flexibility when expressing the intended tasks because one generic property can represent one type of regions without explicitly specify the location of these regions.

Example 3.1. An office-like workspace has six rooms and one corridor, which gives the partition in Figure 3.1. Two properties are "there is a basket in the region" and "there is a ball in the region". ▲

Additionally, since every region is a dense subset of the n-dimensional space, it is impossible to represent each region by the set of points contained in it. Thus it is crucial to represent and encode these regions efficiently. For instance, rectangles can be encoded by its center coordinate, height and width; sphere by its center and radius; triangular by the coordinates of its corners. There are also automated partition tools like Delaunay triangulation [100] and Voronoi diagram [97]. Note that this level of partition is preliminary and not robot-specific.

3.1.2 Robot dynamics and navigation controller

We assume the robot satisfies the following continuous dynamics:

$$\dot{x}(t) = f(x(t), u(t)), \tag{3.2}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ are the position and control signal; $f(\cdot)$ is Lipschitz continuous [82]. Thus the system is deterministic, i.e., given an initial state $x(0)$, a control input $u(t) : \mathbb{R} \to \mathbb{R}^m$ produces an unique state trajectory. We mainly take into account the single-integrator dynamics or the unicycle model. However the proposed framework can be potentially applied to other dynamics.

Given the preliminary partition from Section 3.1.1 and the agent dynamics by (3.2), we need to abstract the robot's ability to transit from one region to another, which is not necessarily the adjacency relation in the geometrical sense. Instead it is defined in the control-driven fashion.

Definition 3.2. There is a transition from π_i to π_j if an admissible navigation controller $\mathcal{U} : \mathbb{R}^n \times \Pi \times \Pi \to \mathbb{R}^m$:

$$u(t) = \mathcal{U}(x(t), \pi_i, \pi_j), \qquad t \in [t', t''] \tag{3.3}$$

exists that could drive the system (3.2) from **any** point in region π_i to a point in region π_j in finite time. At the same time the robot should stay within π_i or π_j. Namely, $x(t') \in \pi_i$, $x(t'') \in \pi_j$ and $x(t) \in \pi_i \cup \pi_j$, $\forall t \in [t', t'']$. ▲

The above definition is closely related to the implementation of a discrete motion plan described in Section 3.3.3. Two main navigation techniques are discussed in the thesis: (i) [103] proposes a potential-field-based feedback control algorithm. It navigates a differential-driven mobile robot from any point inside a region to an adjacent region through a desired facet. The partition is based on generalized Voronoi diagram and a smooth vector filed is constructed over each triangular region; (ii) [87] provides a provably correct point-to-point navigation algorithm, by constructing an exact navigation function for sphere workspace and obstacles. It has been successfully applied in both single [105] and multi-agent [24] navigation control under different geometric constraints, like single-integrator [106], double-integrator [23] vehicles. By following the negated gradient of the navigation function, a collision free path is guaranteed from almost any initial position in the free space to any goal position in the free space given that the workspace is valid.

It is rarely the case that a navigation technique is applicable to any type of partitions. For instance, the potential-filed-based method requires a triangular partition while the navigation-function-based approach needs all sphere structures. This means that the preliminary workspace partition might be modified in terms of number, size and shape of the regions. Example 3.2 shows some cases where the preliminary partition is modified after incorporating the robot dynamics and navigation techniques. This might lead to an over-approximation or under-approximation [129] of the actual workspace as some regions are shrunk or expanded during the process. In some cases, another navigation technique needs to be designed when it is infeasible to incorporate one navigation technique to a given workspace model.

Example 3.2. As shown in Figure 3.2, the office-like workspace from Figure 3.1 is further partitioned since the underlying navigation technique relies on triangular

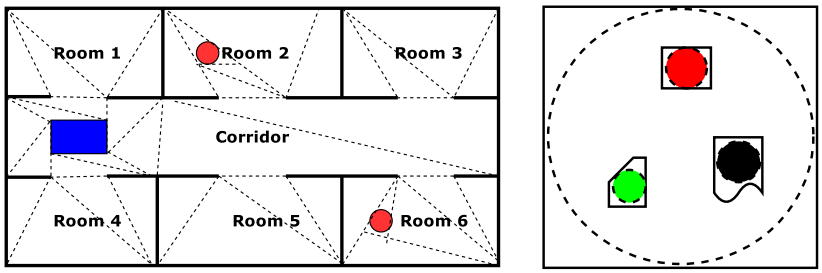


Figure 3.2: Results of workspace abstraction after incorporating the navigation techniques, as explained in Example 3.2.

partition. The irregular regions in the right image are approximated by circular areas due to its navigation-function-based construction. ▲

3.1.3 Control-driven and weighted FTS

The robot motion is abstracted as transitions among the regions $\Pi = \{\pi_1, \cdots, \pi_N\}$. With a slight abuse of notation, we denote by $\pi_i = \{\texttt{the robot is at region}\ \pi_i\}$, $i = 1, \cdots, N$, the states that reflect which region the robot is currently visiting. The transitions or state changes represent that the robot has moved from one region to another. Recall that the transition relation is not necessarily the adjacency relation in the geometrical sense, but by Definition 3.2. Formally the control-driven and weighted finite-state transition system (FTS) is defined below:

Definition 3.3. The weighted FTS is a tuple $\mathcal{T}_c = (\Pi, \longrightarrow_c, \Pi_0, AP, L_c, W_c)$, where $\Pi = \{\pi_i, i = 1, \cdots, W\}$ is the set of states; $\longrightarrow_c \subseteq \Pi \times \Pi$ is the transition relation by Definition 3.2. For simplicity, $\pi_i \longrightarrow_c \pi_j$ is equivalent to $(\pi_i, \pi_j) \in \longrightarrow_c$; $\Pi_0 \subseteq \Pi$ is the initial state, to indicate where the robot may start from; $AP = AP_r \cup AP_p$ is the set of atomic propositions; $L_c : \Pi \to 2^{AP}$ is a labeling function by Definition 3.1; $W_c : \Pi \times \Pi \to \mathbb{R}^+$ is the weight function as the cost of a transition in $\longrightarrow_c$. ▲

We assume that $\mathcal{T}_c$ does not have a terminal state. The successors of state π_i are defined as $\texttt{Post}(\pi_i) = \{\pi_j \in \Pi \,|\, \pi_j \longrightarrow_c \pi_j\}$. An infinite *path* of $\mathcal{T}_c$ is an infinite state sequence $\tau = \pi_1\pi_2\cdots$ such that $\pi_1 \in \Pi_0$ and $\pi_i \in \texttt{Post}(\pi_{i-1})$ for all $i > 0$. The *trace* of a path is the sequence of sets of atomic propositions that are true in the states along that path, i.e., $\texttt{trace}(\tau) = L_c(\pi_1)L_c(\pi_2)\cdots$. The trace of $\mathcal{T}_c$ is defined as $\texttt{Trace}(\mathcal{T}_c) = \cup_{\tau \in I} \texttt{trace}(\tau)$, where I is the set of all infinite paths in $\mathcal{T}_c$. An useful way to represent an infinite path is to use the ω-operator, to indicate the segment that has to be repeated infinitely many times [11].

The weighted FTS $\mathcal{T}_c$ is *fully-known* if it reflects the actual workspace model and robot dynamics; $\mathcal{T}_c$ is called *static* if $\mathcal{T}_c$ does not change with time.

3.2 Task specification as LTL formulas

A language is needed to specify a complex task, which on one hand should be expressive enough to specify various types of tasks, and on the other hand should be formal enough to avoid ambiguity and misinterpretation. Linear time Temporal Logic (LTL) provides a concise and formal way to specify both propositional and temporal constraints on the system behavior.

3.2.1 Syntax and semantics

Linear-time temporal logic (LTL) is defined using the following syntax:

$$\varphi ::= \top \mid a \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2, \tag{3.4}$$

where $a \in AP$ and $\wedge$ (*or*), $\neg$ (*not*), $\bigcirc$ (*next*), U (*until*). For brevity, we omit the derivations of other useful operators like $\Box$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*) and refer to Chapter 5 of [11]. An infinite *word* over the alphabet 2^{AP} is an infinite sequence $\sigma \in (2^{AP})^\omega$ that $\sigma = S_0\, S_1\, S_2 \cdots$, where $S_k \in 2^{AP}$ for all $k = 1, 2, \cdots$, where S_k is the set of atomic propositions that are true at time step k.

The semantics of LTL for an infinite word σ is given via a doubly-recursive definition of the relation $(\sigma, k) \vDash \varphi$, i.e., σ *satisfies* φ at time step k.

Definition 3.4. The semantics of LTL is defined as follows:

$$\begin{aligned}
(\sigma, k) \vDash a \quad &\leftrightarrow \quad a \in S_k \\
(\sigma, k) \vDash \neg\varphi \quad &\leftrightarrow \quad (\sigma, k) \nvDash \varphi \\
(\sigma, k) \vDash \bigcirc\varphi \quad &\leftrightarrow \quad (\sigma, k+1) \vDash \varphi \\
(\sigma, k) \vDash \varphi_1 \vee \varphi_2 \quad &\leftrightarrow \quad (\sigma, k) \vDash \varphi_1 \text{ or } (\sigma, k) \vDash \varphi_2 \\
(\sigma, k) \vDash \varphi_1 \,\mathsf{U}\, \varphi_2 \quad &\leftrightarrow \quad \exists k' \in [k, +\infty],\ (\sigma, k') \vDash \varphi_2 \text{ and} \\
&\qquad\quad \forall k'' \in (k, k'),\ (\sigma, k'') \vDash \varphi_1 .
\end{aligned}$$

▲

Any LTL formula φ is satisfied by σ at time step 0 if $(\sigma, 0) \vDash \varphi$ (for simplicity we denote by $\sigma \vDash \varphi$). The *words* of φ is defined as the set of words that satisfy φ at time step 0, i.e., $\texttt{Words}(\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \vDash \varphi\}$. Given an infinite path τ of $\mathcal{T}_c$ and an LTL formula φ over AP, $\texttt{trace}(\tau)$ is a word over the alphabet 2^{AP}. Thus we can verify if $\texttt{trace}(\tau)$ satisfies φ according to the semantics (3.4).

Definition 3.5. An infinite path τ **satisfies** φ, i.e., $\tau \vDash \varphi$ if its trace $\texttt{trace}(\tau) \vDash \varphi$. A satisfying path is also called a **plan** for φ. ▲

LTL formulas can be used to specify various robot control tasks, such as safety ($\Box\neg\varphi_1$, globally avoiding φ_1), ordering ($\Diamond(\varphi_1 \wedge \Diamond(\varphi_2 \wedge \Diamond\varphi_3))$, φ_1, φ_2, φ_3 hold in sequence), response ($\varphi_1 \Rightarrow \varphi_2$, if φ_1 holds, φ_2 will hold in future), repetitive surveillance ($\Box\Diamond\varphi$, φ holds infinitely often).

Another particular class of LTL we consider in this thesis is the syntactically co-safe LTL (sc-LTL) [93]. They only contain the $\bigcirc$, U and $\Diamond$ operators and are

written in positive normal form [35]. In contrast, the satisfaction of an sc-LTL formula can be achieved in a finite time, i.e., each word satisfying an sc-LTL formula φ consists of a *satisfying prefix* that can be followed by an arbitrary suffix.

3.2.2 Problem formulation

As discussed in the introduction, a single counter example would be enough for the purpose of verification, i.e., to verify that not all infinite paths of $\mathcal{T}_c$ satisfy φ. However for plan synthesis, since the derived plan needs to be implemented by autonomous robots, we need to find a plan that fulfills certain structure.

Prefix-suffix structure

As a plan is essentially an infinite sequence of states in $\mathcal{T}_c$, it is not convenient to encode, analyze or manipulate both in theory and software implementation. Thus we consider the plan with the *prefix-suffix* structure:

$$\tau = \langle \tau_{\mathrm{pre}}, \tau_{\mathrm{suf}} \rangle = \tau_{\mathrm{pre}} \left[\tau_{\mathrm{suf}}\right]^{\omega} \tag{3.5}$$

where the prefix τ_{pre} is transversed only once and the suffix τ_{suf} is repeated infinitely. A plan with this prefix-suffix structure has a finite representation as (3.5). This structure is also called *lasso-shapeed* in [130] with the stem τ_{pre} and the loop τ_{suf}.

Definition 3.6. φ is called **feasible** if there exists an infinite path τ of $\mathcal{T}_c$ that satisfies φ. ▲

With the above preliminaries in hand, the problem formulation for the nominal scenario could be stated as follows:

Problem 3.1. *Given the control-driven wFTS $\mathcal{T}_c$ and an LTL formula φ over AP, (i) find a plan with the prefix-suffix structure in* (3.5)*; (ii) construct the hybrid control strategy based on* (3.3) *to execute the derived plan.* ▲

3.3 Hybrid controller synthesis

In this section, we describe in detail how to synthesize the discrete plan that solves the first part of Problem 3.1.

Büchi automaton

Given an LTL formula φ over AP, there exists a Nondeterministic Büchi automaton (NBA) over 2^{AP} corresponding to φ, denoted by $\mathcal{A}_\varphi$.

Definition 3.7. The NBA $\mathcal{A}_\varphi$ is defined by a five-tuple:

$$\mathcal{A}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F}), \tag{3.6}$$

where Q is a finite set of states; $Q_0 \subseteq Q$ is a set of initial states; 2^{AP} is the alphabet; $\delta : Q \times 2^{AP} \to 2^Q$ is a transition relation; $\mathcal{F} \subseteq Q$ is a set of accepting states. ▲

An infinite run of the NBA is an infinite sequence of states that starts from an initial state and follows the transition relation. Namely, $r = q_0 q_1 q_2 \cdots$, where $q_0 \in Q_0$ and $q_{k+1} \in \delta(q_k, S)$ for some $S \in 2^{AP}$, $k = 0, 1, \cdots$. Moreover, r is called *accepting* if $\texttt{Inf}(r) \cap \mathcal{F} \neq \varnothing$, where $\texttt{Inf}(r)$ is the set of states that appear in r infinitely often. We denote the successors of $q_m \in Q$ by $\texttt{Post}(q_m) = \{q_n \mid \exists S \in 2^{AP},\ q_n \in \delta(q_m, S)\}$.

Definition 3.8. Given an infinite word $\sigma = S_0 S_1 S_2 \cdots$ over 2^{AP}, its **resulting run** in $\mathcal{A}_\varphi$ is denoted by $r_\sigma = q_0 q_1 q_2 \cdots$, which satisfies: (i) $q_0 \in Q_0$; (ii) $q_{i+1} \in \delta(q_i, S_i)$, $\forall i = 1, 2, \cdots, \infty$. Similar statement holds for a finite word $\bar{\sigma} = S_0 S_1 S_2 \cdots S_{N+1}$. ▲

Note that since $\mathcal{A}_\varphi$ is nondeterministic, there may exist *multiple* resulting runs of the same world. Denote by $\mathcal{L}_\omega(\mathcal{A}_\varphi)$ the accepted language of $\mathcal{A}_\varphi$, which is the set of infinite words that result in an accepting run of $\mathcal{A}_\varphi$, i.e., $\mathcal{L}_w(\mathcal{A}_\varphi) = \{\sigma \in (2^{AP})^\omega \mid r_\sigma \text{ is an accepting run}\}$.

Lemma 3.1. $\mathcal{L}_w(\mathcal{A}_\varphi) = \texttt{Words}(\varphi)$

Proof. See proof of Theorem 5.37 in [11] ■

The translation process from an LTL formula to its corresponding NBA can be done in time and space $2^{\mathcal{O}(|\varphi|)}$ [11]. However, there are fast translation algorithms [41], which generates NBA with few states and transitions. Furthermore it is tedious to list all input alphabets for each transition, particularly given a large set of AP. Thus it is important to represent them in an compact and efficient manner. The translation algorithm from [41] generates a boolean expression for each transition, which accepts all alphabets that enable this transition (as shown in Figure 3.3). Binary decision diagrams (BDD) are well-known for their efficiency to represent and evaluate boolean functions [4] . As a result, an NBA can be encoded symbolically and efficiently. More details can be found in Section 8.

Example 3.3. The NBA that corresponds to $\varphi = (\square \diamond a_1) \wedge (\square \diamond a_2) \wedge (\square \diamond a_3) \wedge (\square \neg a_4)$ is derived from [41] and shown in Figure 3.3. The transition from state q_1 to q_3 is given by $q_3 \in \delta(q_1, l)$ where the boolean expression $l = (a_2 \,\&\, \neg a_4)$ encodes four input alphabets $\{a_2\}$, $\{a_2, a_1\}$, $\{a_2, a_3\}$, $\{a_2, a_1, a_3\}$. ▲

3.3.1 Product Büchi automaton

The automaton-based model-checking algorithm can be found in [147] and Algorithm 11 of [11]. It is based on checking the emptiness of the product Büchi automaton. Since $\texttt{Words}(\varphi) = \mathcal{L}_w(\mathcal{A}_\varphi)$ and $\texttt{trace}(\tau) \in \texttt{Trace}(\mathcal{T}_c)$, the original problem is equivalent to finding the intersection $\texttt{Trace}(\mathcal{T}_c) \cap \mathcal{L}_w(\mathcal{A}_\varphi)$, which is actually the language of the product Büchi automaton $\mathcal{A}_p = \mathcal{T}_c \otimes \mathcal{A}_\varphi$, which accepts all runs that are valid for $\mathcal{T}_c$ and at the same time satisfy φ.

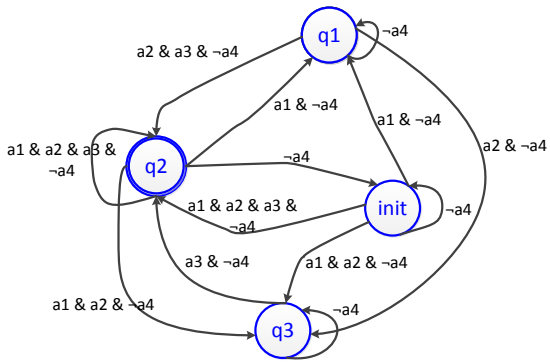


Figure 3.3: The NBA associated with $\varphi = (\square \diamond a_1) \wedge (\square \diamond a_2) \wedge (\square \diamond a_3) \wedge (\square \neg a_4)$ (constructed by program from [41]), as explained in Example 3.3.

It is important to mention that unlike the model-checking algorithm for verification, we do not negate the task specification before generating the associated NBA and the product automaton. This is because we are interested in the "good" behavior of the system that satisfies the specification, not the "bad" behavior that satisfies the negated specification for the purpose of verification.

Definition 3.9. The weighted product Büchi automaton is defined by $\mathcal{A}_p = \mathcal{T}_c \otimes \mathcal{A}_\varphi = (Q', \delta', Q'_0, \mathcal{F}', W_p)$, where $Q' = \Pi \times Q = \{\langle \pi, q \rangle \in Q' \,|\, \forall \pi \in \Pi, \ \ \forall q \in Q\}$; $\delta' : Q' \to 2^{Q'}$. $\langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$ iff $(\pi_i, \pi_j) \in \longrightarrow_c$ and $q_n \in \delta(q_m, L_c(\pi_i))$; $Q'_0 = \{\langle \pi, q \rangle \,|\, \pi \in \Pi_0, \ q_0 \in Q_0\}$, the set of initial states; $\mathcal{F}' = \{\langle \pi, q \rangle \,|\, \pi \in \Pi, \ q \in \mathcal{F}\}$, the set of accepting states; $W_p : Q' \times Q' \to \mathbb{R}^+$ is the weight function: $W_p(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) = W_c(\pi_i, \pi_j)$, where $\langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$. ▲

Since $\mathcal{A}_p$ remains a Büchi automaton [11], its infinite run and its accepting condition can be defined similarly as $\mathcal{A}_\varphi$. An infinite run R is called accepting if it intersects with the accepting set $\mathcal{F}'$ infinitely often. The successors of q'_s are given by $\texttt{Post}(q'_s) = \{q'_g \,|\, q'_g \in \delta'(q'_s)\}$. Given a state $q' = \langle \pi, q \rangle \in Q'$, its projection on Π is denoted by $q'|_\Pi = \pi$ and its projection on Q is denoted by $q'|_Q = q$. Given an infinite run $R = q'_0 q'_1 q'_2 \cdots$ of $\mathcal{A}_p$, its projection on Π is denoted by $R|_\Pi = q'_0|_\Pi \, q'_1|_\Pi \, q'_2|_\Pi \cdots$ and its projection on Q is denoted by $R|_Q = q'_0|_Q \, q'_1|_Q \, q'_2|_Q \cdots$.

Lemma 3.2. *If there exists an infinite path τ of $\mathcal{T}_c$ such that $\tau \vDash \varphi$, then **at least** an accepting run of $\mathcal{A}_p$ exists.*

Proof. See the proof of Theorem 4.63 from [11]. ■

Lemma 3.3. *Given an accepting run R of $\mathcal{A}_p$, then $R|_\Pi \vDash \varphi$.*

Proof. By definition, there exists an accepting state $q'_f \in \mathcal{F}'$ appearing in R infinitely often. Thus $q'_f|_Q \in \mathcal{F}$ appears in $R|_Q$ infinitely often, yielding that $R|_Q$ is an accepting

run. It can be easily shown that $R|_Q$ is one of the resulting runs of the trace of $R|_\Pi$. Thus $\texttt{trace}(R|_\Pi) \in \mathcal{L}_\omega(\varphi) = \texttt{Words}(\varphi)$, which implies $R|_\Pi \vDash \varphi$ by Definition 3.5. ■

Cost of an accepting run

We intended to find an infinite path τ of $\mathcal{T}_c$ with the prefix-suffix structure in (3.5), such that $\tau \vDash \varphi$. The following lemma is important for the correctness.

Lemma 3.4. *If there exists an infinite path τ with the prefix-suffix structure and $\tau \vDash \varphi$, then **at least** one accepting run of $\mathcal{A}_p$ exists with the prefix-suffix structure.*

Proof. Since $\tau \vDash \varphi$, then $\sigma = \texttt{trace}(\tau) \in \texttt{Words}(\varphi)$. Furthermore as $\mathcal{L}_w(\mathcal{A}_\varphi) = \texttt{Words}(\varphi)$ by Lemma 3.1, $\sigma \in \mathcal{L}_w(\mathcal{A}_\varphi)$, meaning that the resulting run r_σ in $\mathcal{A}_\varphi$ by Definition 3.8 is an accepting run of $\mathcal{A}_\varphi$.

Without loss of generality, let $\tau = \pi_0\pi_1\cdots\pi_i\cdots$ and $r_\sigma = q_0q_1\ldots q_j\cdots$. Now we prove that $R = \langle\pi_0, q_0\rangle\langle\pi_1, q_1\rangle\cdots\langle\pi_i, q_j\rangle\cdots$ is an accepting run of $\mathcal{A}_p$. First of all, $\langle\pi_0, q_1\rangle \in Q_0'$ as $\pi_0 \in \Pi_0$ and $q_0 \in Q_0$. Secondly, $\langle\pi_{i+1}, q_{j+1}\rangle \in \delta'(\langle\pi_i, q_j\rangle)$ as $(\pi_i, \pi_{i+1}) \in \longrightarrow_c$ and $q_{i+1} \in \delta(q_i, L_c(\pi_i))$. At last, since r_σ is an accepting run of $\mathcal{A}_\varphi$, there exists an accepting state $q_f \in \mathcal{F}$ that appears in r_σ infinitely often. Correspondingly, there exists at least one $\langle\pi, q_f\rangle$ that appears in R infinitely often and $\langle\pi, q_f\rangle \in \mathcal{F}'$, where π may stand for one or several states in Π. Since $\mathcal{F}'$ is finite, there must be one accepting state $q_f' \in \mathcal{F}'$ that appears in R infinitely often.

Then an accepting run with the prefix-suffix structure can be constructed by using the segment from $\langle\pi_0, q_0\rangle$ to q_f' as prefix and the segment starting from q_f' and back to q_f' as the suffix, which completes the proof. ■

Thus we could focus on the accepting runs of $\mathcal{A}_p$ with the *prefix-suffix* structure:

$$\begin{aligned} R = \langle R_{\text{pre}}, R_{\text{suf}}\rangle &= q_0'\,q_1'\cdots q_f'\,\left[\,q_f'\,q_{f+1}'\cdots\cdots q_n'\,\right]^\omega \\ &= \langle\pi_0, q_0\rangle\cdots\langle\pi_{f-1}, q_{f-1}\rangle\,\left[\,\langle\pi_f, q_f\rangle\langle\pi_{f+1}, q_{f+1}\rangle\cdots\cdots\langle\pi_n, q_n\rangle\,\right]^\omega, \end{aligned} \tag{3.7}$$

where $q_0' = \langle\pi_0, q_0\rangle \in Q_0'$ and $q_f' = \langle\pi_f, q_f\rangle \in \mathcal{F}'$. Note that there are no correspondences among the subscripts. The prefix part $R_{\text{pre}} = (q_0'\,q_1'\cdots q_f')$ from an initial state q_0' to one accepting state q_f' that is executed only once while the suffix part $R_{\text{suf}} = (q_f'\,q_{f+1}'\cdots\cdots q_n')$ from q_f' back to itself that is repeated infinitely. Given the finite representation as (3.7), there is a finite set of transitions appearing in R:

$$\texttt{Edge}(R) = \{(q_i', q_{i+1}'),\ i = 0, 1\cdots, (n-1)\} \cup \{(q_n', q_f')\}. \tag{3.8}$$

The structure also allows us to define the total cost of R:

$$\begin{aligned} \texttt{Cost}(R, \mathcal{A}_p) &= \sum_{i=0}^{f-1} W_p(q_i', q_{i+1}') + \gamma \sum_{i=f}^{n-1} W_p(q_i', q_{i+1}') \\ &= \sum_{i=0}^{f-1} W_c(\pi_i, \pi_{i+1}) + \gamma \sum_{i=f}^{n-1} W_c(\pi_i, \pi_{i+1}) \end{aligned} \tag{3.9}$$

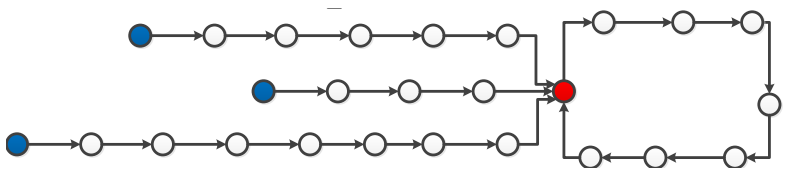


Figure 3.4: For every pair of $q'_0 \in Q'_0$ (in blue) and $q'_f \in \mathcal{F}'$ (in red), the shortest path from q'_0 to q'_f and the shortest cycle containing q'_f are computed.

where the first summation in (3.9) represents the accumulated weights of transitions along the prefix and the second the suffix; $\gamma \geq 0$ is the relative weighting on the cost of transient response (the prefix) and steady response (the suffix).

Problem 3.2. *Given the product automaton $\mathcal{A}_p$, find its accepting run with the prefix-suffix structure that* ***minimizes*** *the cost defined by* (3.9). ▲

We denote by R_{opt} the solution for the above problem, as the optimal accepting run. Its corresponding plan is given by $\tau_{\text{opt}} = R_{\text{opt}}|_{\Pi}$.

***Remark* 3.1.** τ_{opt} might not be the actual optimal plan with the prefix-suffix structure, whose cost is defined similarly as (3.9). As pointed out in [130], this optimality loss is due to the simplification during the translation process from LTL formulas to the corresponding NBA. However there are certain types of *tight* NBA such as [19] that preserves this quality. It means that the optimal plan can be found directly as the projection of the optimal run. But we have not found the software implementation for this translation algorithm yet. Nevertheless the trade-off between optimality and computational complexity remains as the tight NBA would certainly have far more states and edges as shown in the examples of [130]. ▲

3.3.2 Optimal run search

In this part we present firstly two methods to construct the product automaton and then the graph search algorithm to find the optimal accepting run.

Full construction

Since only static and fully-known workspace is considered, both $\mathcal{T}_c$ and φ are time-invariant. Thus $\mathcal{A}_p$ can be constructed fully by Definition 3.9. In particular, given a FTS state $\pi_i \in \Pi$ and a NBA state $q_m \in Q$, the composed state $\langle \pi_i, q_m \rangle$ is added to Q' if it is not in Q' already. Then all transitions originated from $\langle \pi_i, q_m \rangle$ can be found by the definition of δ', namely $\forall \pi_j \in \texttt{Post}(\pi_i)$ and $\forall q_m \in \texttt{Post}(q_n)$, $\langle \pi_j, q_n \rangle \in \texttt{Post}(\langle \pi_i, q_m \rangle)$ if $q_n \in \delta(q_m, L(\pi_i))$.

After constructing $\mathcal{A}_p$ fully, Algorithm 3.1 takes as input arguments $\mathcal{A}_p$ and a set of starting states $S' \subseteq Q'$, which is set to Q'_0 by default, while it generates the optimal accepting run. Algorithm 3.1 utilizes Dijkstra's algorithm [97] for computing the shortest path from a single source node to a set of target nodes within a weighted

Algorithm 3.1 Search for optimal run, `OptRun`().

Input: $\mathcal{A}_p$, $S' = Q'_0$ by default
Output: R_{opt}
1. If Q'_0 or $\mathcal{F}'$ is empty, construct Q'_0 or $\mathcal{F}'$ first.
2. For each initial state $q'_0 \in S'$, call `DijksTargets`(G, q'_0, $\mathcal{F}'$).
3. For each accepting state $q'_f \in \mathcal{F}'$, call `DijksCycle`(G, q'_f).
4. Find the pair of $(q'_{0,\star}, q'_{f,\star})$ that minimizes the summed cost defined by (3.9).
5. Optimal accepting run R_{opt}, prefix: the shortest path from $q'_{0,\star}$ to $q'_{f,\star}$; suffix: the shortest cycle from $q'_{f,\star}$ and back to itself.

graph. In particular, function `DijksTargets`($\mathcal{A}_p$, q'_S, Q'_T) computes shortest paths in $\mathcal{A}_p$ from source state $q'_S \in Q$ to every target state belonging to the set $Q'_T \in Q'$. Function `DijksCycle`($\mathcal{A}_p$, q'_S) is used to compute shortest cycle from the source state q'_S back to itself. As shown in Figure 3.4, for each pair of initial and accepting states (q'_0, q'_f) where $q'_0 \in Q'_0$ and $q'_f \in \mathcal{F}'$, the shortest path from q'_0 to q'_f is obtained from line 1 of Algorithm 3.1 where `DijksTargets`$(\cdot)$ is called while the shortest cycle containing q'_f is obtained from line 2 of Algorithm 3.1 where `DijksCycle`$(\cdot)$ is called. At last, the pair $(q'_{0,\star}, q'_{f,\star})$ that minimizes the total cost defined by (3.9) is chosen. Then the optimal accepting run is determined by setting its prefix as the shortest path from $q'_{0,\star}$ to $q'_{f,\star}$ and its suffix as shortest cycle containing $q'_{f,\star}$.

The algorithm to compute $\mathcal{A}_p$ has the complexity proportional to the sum of the number of states and transitions in $\mathcal{A}_p$. The worst-case complexity of Algorithm 3.1 is given by $\mathcal{O}(|\delta'| \cdot \log |Q'_0| \cdot (|Q'_0| + |\mathcal{F}'|)))$ as essentially the Dijkstra algorithm [89, 97] has to be called over $\mathcal{A}_p$ by the number of times equal to $|Q'_0| + |\mathcal{F}'|$.

On-the-fly construction

Beside constructing $\mathcal{A}_p$ once for all, we would construct $\mathcal{A}_p$ on-the-fly along with the graph search Algorithm 3.1 and even the revision Algorithm 4.4 introduced later. In other words, the states and transition relations of $\mathcal{A}_p$ are built "on demand". When the search algorithm visits any state $q'_s \in Q'$, it constructs the adjacency relation of q'_s by iterating through all successors of q'_s and returns the corresponding transition $q'_g \in \delta'(q'_s)$ along with its weight. Note that each state $q'_s \in Q'$ is marked by the label "visited" or "unvisited", to indicate if the transitions originated from q'_s have to be constructed. In particular, if q'_s is marked "visited", it means that they have been constructed before and can be returned directly; if q'_s is marked "unvisited" or π_i belongs to $\widehat{\Pi}$, they need to be constructed by Definition 3.9 ; $\widehat{\Pi}$ will be defined in (4.19), which can be treated as $\varnothing$ for now. Detailed implementation of the on-the-fly construction can be found in [48].

The markers "visited" and "unvisited" improve the computational efficiency as the adjacency relation of states that have been visited before can be returned directly, meaning that the results from previous planning iterations can be reused later; $\widehat{\Pi}$ provides a highly efficient way to maintain and update $\mathcal{A}_p$ in case of updates

Size n^2	$t_{\text{full}}\,(s)$	$t_{\text{syn}}\,(s)$	$t_{\text{fly}}\,(s)$	Size n^2	$t_{\text{full}}\,(s)$	$t_{\text{syn}}\,(s)$	$t_{\text{fly}}\,(s)$
25	1.11	0.03	0.82	3025	64.77	2.98	74.53
225	8.85	0.23	6.88	4225	91.65	3.96	101.34
625	14.04	0.56	11.09	5625	118.24	5.44	144.78
1225	24.96	1.12	23.24	7225	150.77	7.00	178.99
2025	42.40	1.87	38.13	9025	187.01	8.95	218.17

Table 3.1: Numerical results for Example 3.4.

in $\mathcal{T}_c$, as later described in Algorithm 4.4 in Section 4.3. Then Algorithm 3.1 can be easily modified such that it takes the adjacency relation of $\mathcal{A}_p$ as an input, instead of the fully-constructed $\mathcal{A}_p$. Functions `DijksTargets`$(\cdot)$ and `DijksCycle`$(\cdot)$ still work in the same way since the Dijkstra algorithm only needs the adjacency relation in the breadth-first structure and the associated weight. Even though the worst-case computational complexity of Algorithm 3.1 remains the same for these two different methods to construct $\mathcal{A}_p$, the on-the-fly construction is essential for the partially-known and dynamic workspace that will be discussed in Section 4.3, where $\mathcal{T}_c$ has to be updated frequently.

Example 3.4. This example shows the general complexity of synthesizing a discrete motion and task plan using the proposed scheme. The FTS consists of $n \times n$ uniform grids as states and the cost from one region to an adjacent region is set randomly and uniformly within $[0, 1]$. The task is to deliver two objects to two different destinations separately and then return to the base station. The formula can be written similarly as (3.13). The location of the objects and destinations are chosen randomly. We keep track of: (i) t_{full}, the time needed to fully construct $\mathcal{A}_p$ by Definition 3.9; (ii) t_{syn}, the time taken to find the optimal accepting run by Algorithm 3.1 over the fully-constructed $\mathcal{A}_p$; (iii) t_{fly}, the time taken to construct $\mathcal{A}_p$ on-the-fly along the optimal search by Algorithm 3.1. They are measured by CPU time in seconds on a desktop computer (3.06 GHz Duo CPU and 8GB of RAM).

The associated NBA has 75 states and 877 edges. Judging from Table 3.1, given a fixed transition system and this particular task, these two different approaches consume almost the same amount of time. The construction of $\mathcal{A}_p$ takes almost 95% of the total time while the graph search algorithm is relatively fast. ▲

3.3.3 Control structure

Assume the optimal run R_{opt} from Algorithm 3.1 has the following format:

$$\begin{aligned} R_{\text{opt}} &= \langle R_{\text{pre}},\, R_{\text{suf}} \rangle \\ &= R_{\text{pre},1} R_{\text{pre},2} \cdots R_{\text{pre},N_{\text{pre}}} \left[R_{\text{suf},1} R_{\text{suf},2} \cdots R_{\text{suf},N_{\text{suf}}} \right]^{\omega}, \end{aligned} \tag{3.10}$$

where $R_{\text{pre}} = R_{\text{pre},1} \cdots R_{\text{pre},N_{\text{pre}}}$ is the prefix; $R_{\text{suf}} = R_{\text{suf},1} \cdots R_{\text{suf},N_{\text{suf}}}$ is the suffix.

Algorithm 3.2 Plan execution by hybrid control, `HybCon( )`

Input: R_{opt}, $x(t)$
Output: u, R_{past}, τ_{past}, q'_{cur}
$q'_{\text{cur}} = q'_{\text{next}} = R_{\text{pre},1}$, $\pi_{\text{next}} = q'_{\text{next}}|_{\Pi}$, $R_{\text{past}} = [\ ]$, $\tau_{\text{past}} = [\ ]$
while `True` **do**
  if $x(t) \in \pi_{\text{next}}$ confirmed **then**
    $q'_{\text{cur}} = q'_{\text{next}}$, $\tau_{\text{past}} = \tau_{\text{past}} + \pi_{\text{next}}$, $R_{\text{past}} = R_{\text{past}} + q'_{\text{next}}$
    $q'_{\text{next}} =$ `NextGoal`$(q'_{\text{cur}}, R_{\text{opt}})$, $\pi_{\text{cur}} = q'_{\text{cur}}|_{\Pi}$, $\pi_{\text{next}} = q'_{\text{next}}|_{\Pi}$
  $u(t) = \mathcal{U}(x(t), \pi_{\text{cur}}, \pi_{\text{next}})$

Thus the robot's status within the accepting run can be uniquely determined by the segment (prefix or suffix) and its index within that segment, which are denoted by seg and k. To generate the infinite plan using R_{pre} and R_{suf}, function `NextGoal`$(\cdot)$ takes the current product state $q'_{\text{cur}} = R_{seg,k}$ and R_{opt} as inputs and generates the next goal product state. Simply speaking, it firstly follows the prefix until the end of prefix. Then it switches to the suffix and follows it until the end. After that it restarts from the beginning of the suffix and repeats the same process. In this way, the suffix is executed infinitely many times.

Algorithm 3.2 executes the derived optimal run R_{opt} off-line. Initially the agent starts from $q'_{\text{pre},1}$; q'_{cur} and q'_{pre} are the current and next product state in R_{opt}; π_{cur} and π_{next} indicate the robot's current region and the next goal region. π_{next} is initialized as $\tau_{\text{pre},1}$; R_{past} is used to store the sequence of product states that has been reached in R_{opt}; τ_{past} is used to store the sequence of regions the robot has visited; Once a confirmation is acquired that π_{next} is reached, q'_{next} is added to R_{past} and correspondingly π_{next} is added to τ_{past}. Then q'_{next} is set to the next goal state given q'_{cur}. As a result, the controller $\mathcal{U}(x(t), \pi_{\text{cur}}, \pi_{\text{next}})$ is activated to drive the agent from π_{cur} to π_{next}. Algorithm 3.2 can be running for infinitely long time as the the suffix segment is repeated infinitely many times. Note that the condition $x(t) \in \pi_{\text{next}}$ holds when the robot belongs to its current goal region.

3.4 Case study

In this case study, we validate the proposed framework above by simulation results, while experiment results are presented in Chapter 8.

Workspace abstraction

The workspace is a testbed with size $2.4m \times 2.1m$ representing the office environment shown in Figure 3.1, consisting of three rooms on each side and one corridor in the middle. The corridor is partitioned into three smaller segments. Thus this workspace has nine regions in total "$r1, \cdots, r6, c1, c2, c3$", represented by propositions "r1,⋯, r6, c1, c2, c3". There are one red ball, one green ball and

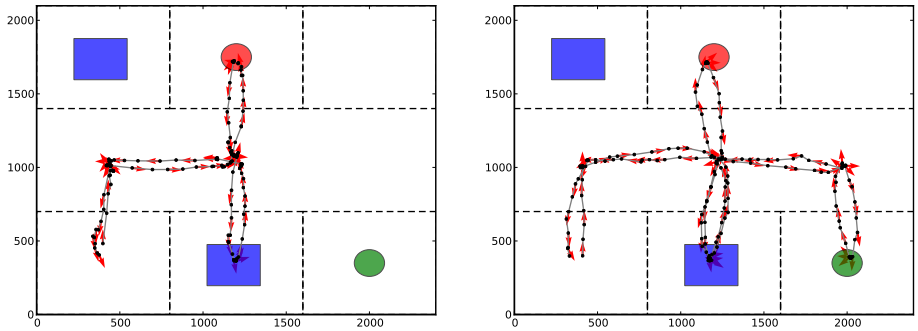


Figure 3.5: Left: the trajectory that fulfills φ_1. Right: the trajectory that fulfills φ_2.

two baskets in different rooms, represented by propositions "rball, gball, basket". The rectangular regions are encoded by the center point, width and height. The horizontally adjacent rooms are separated by vertical walls. The cost of moving from one region to another is estimated by the Euclidean distance between their centers. The region name and its labeling function are given by (from bottom to up, left to right): $(r1, \{\mathsf{r1}\})$, $(r2, \{\mathsf{r2}, \mathsf{basket}\})$, $(r3, \{\mathsf{r3}, \mathsf{gball}\})$, $(c1, \{\mathsf{c1}\})$, $(c2, \{\mathsf{c2}\})$, $(c3, \{\mathsf{c3}\})$, $(r4, \{\mathsf{r4}, \mathsf{basket}\})$, $(r5, \{\mathsf{r5}, \mathsf{rball}\})$, $(r6, \{\mathsf{r6}\})$. The robot we deployed is a NAO robot [5], which is an autonomous, programmable humanoid robot. Its state within the workspace is given by (x_r, y_r, θ_r), where x_r and y_r are the coordinate and $\theta_r \in [-\pi, \pi]$ is its orientation with respect to the x-axis. It has three basic control modules as "$\mathsf{move\_x}(\cdot)$", "$\mathsf{move\_y}(\cdot)$" and "$\mathsf{turn}(\cdot)$". Namely, it can move forward in its local x-axis by the given speed, move sideways in its local y-axis by the given speed and it can turn itself by the given angular speed. Since they are not free of actuation noises and disturbances, we design the following turn-and-forward feedback controller:

$$u = \begin{cases} \mathsf{move\_x}\big(\kappa_1 \cdot \|(x_g - x_r,\, y_g - y_r)\|\big), & \text{if } |\theta_{\text{dif}}| > \bar{\theta} \\ \begin{cases} \mathsf{turn}\big(\kappa_2 \cdot \theta_{\text{dif}}\big), & \text{if } \bar{\theta} < |\theta_{\text{dif}}| \le \pi \\ \mathsf{turn}\big(-\kappa_2 \cdot \mathtt{sign}(\theta_{\text{dif}}) \cdot (2\pi - \mathtt{abs}(\theta_{\text{dif}}))\big), & \text{if } |\theta_{\text{dif}}| > \pi \end{cases} \end{cases} \tag{3.11}$$

where (x_g, y_g) is the goal position; $\|x_g - x_r, y_g - y_r\|$ is the relative distance; $\theta_{\text{ref}} = \arctan(y_g - y_r, x_g - x_r)$ is the relative angle between the robot's position and the goal position; $\theta_{\text{dif}} = \theta_{\text{ref}} - \theta_r$ is the difference between robot's orientation and the desired orientation; $\bar{\theta} < \pi$ is a design parameter deciding when the robot should move forward; $\kappa_1, \kappa_2 > 0$ are design parameters as the proportional gain. Thus the robot would keep turning until it is facing the the goal position. The time interval to update u is also an important tunning parameter. It is verified by the simulation and experimental results that the proposed controller is effective. Then the generic controller $\mathcal{U}(x(t), \pi_i, \pi_j)$ in (3.3) can be obtained by setting the way point in (3.11)

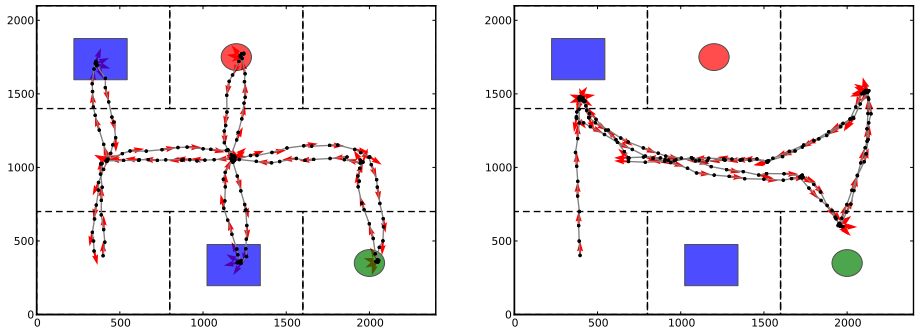


Figure 3.6: Left: the trajectory that fulfills φ_3. Right: the trajectory that fulfills φ_4.

as the center of the next goal region, which guarantees the robot's transition from one region to an adjacent one, except when there are walls in between. Based on this transition relation, we could construct $\mathcal{T}_c$ that consists of 9 regions and 16 edges. In the simulation, we add Gaussian noises to the actuation signal generated by (3.11).

Simulation results

We illustrate the effectiveness of the proposed framework by considering four different task specifications.

Case I: the robot has to pick up the red ball, drop it to the one of the baskets and then stay at room one. It is specified as the LTL formula: $\varphi_1 = \diamondsuit(\,\mathsf{rball} \wedge \diamondsuit\,\mathsf{basket}) \wedge \diamondsuit\,\square\,\mathsf{r1}$, which can be interpreted as "eventually pick up the red ball. Once it is done, move to one basket and drop it. At last come back to room one and stay there". It took $0.03s$ for Algorithm 3.1 to find the optimal plan: "$r1\,c1\,c2\,r5\,c2\,r2\,c2\,c1\,r1\,(r1)^{\omega}$", with the prefix cost 581 and suffix cost 1. Then the optimal plan is constructed and executed off-line by Algorithm 3.2 and the final trajectory is shown in Figure 3.5.

Case II: same delivery task as in Case I, but now it has to deliver two objects and is not allowed for the robot to carry two objects simultaneously. The task is specified by the LTL formula:

$$\begin{aligned}\varphi_2 = \diamondsuit\,(\,\mathsf{rball} \wedge \diamondsuit\,\mathsf{basket}) \wedge \diamondsuit(\,\mathsf{gball} \wedge \diamondsuit\,\mathsf{basket}) \wedge \diamondsuit\,\square\,\mathsf{r1} \\ \wedge\, \square(\,\mathsf{rball} \Rightarrow \bigcirc(\,\neg\mathsf{gball}\ \mathsf{U}\ \mathsf{basket})) \\ \wedge\, \square(\,\mathsf{gball} \Rightarrow \bigcirc(\,\neg\mathsf{rball}\ \mathsf{U}\ \mathsf{basket})),\end{aligned} \tag{3.12}$$

where we add in the constraints that another ball can be picked up only after the robot has dropped the ball in hand. It took $0.86s$ for Algorithm 3.1 to find the optimal plan: "$r1\,c1\,c2\,c3\,r3\,c3\,c2\,r2\,c2\,r5\,c2\,r2\,c2\,c1\ (r1)^{\omega}$", with the prefix cost 1021 and suffix cost 1. The final trajectory is shown in Figure 3.5. It can be seen that the robot picks up the green ball first, drop it in the basket in $r2$, picks up the green ball, and drop it in the basket in $r2$, which fulfills the imposed task.

Case III: same delivery task as in Case II, but now we require that the red ball has to be delivered to the basket in $r2$ while the green ball has to the basket in $r4$. Now the task can be specified by:

$$\begin{aligned}\varphi_3 = &\diamondsuit(\,\mathsf{rball} \wedge \diamondsuit(\,\mathsf{basket} \wedge \mathsf{r2})) \wedge \diamondsuit(\,\mathsf{gball} \wedge \diamondsuit(\,\mathsf{basket} \wedge \mathsf{r4})) \\ &\wedge \square(\,\mathsf{rball} \Rightarrow \bigcirc(\,\neg\mathsf{gball}\ \mathsf{U}\ \mathsf{basket})) \\ &\wedge \square(\,\mathsf{gball} \Rightarrow \bigcirc(\,\neg\mathsf{rball}\ \mathsf{U}\ \mathsf{basket})) \wedge \diamondsuit\square\ \mathsf{r1},\end{aligned} \tag{3.13}$$

where the location for each basket is specified. It took $1.39s$ for Algorithm 3.1 to find the optimal plan: "$r1\,c1\,c2\,r5\,c2\,r2\,c2\,c3\,r3\,c3\,c2\,c1\,r4\,c1\ (r1)^{\omega}$", with the prefix cost 1021 and suffix cost 1. The final trajectory is shown in Figure 3.6. It shows that the robot picks up the red ball first, drop it in the basket in $r2$, picks up the green ball, and drop it in the basket in $r4$.

Case IV: given a surveillance task, the robot needs to inspect rooms $r3$, $r4$ and $r6$ infinitely often. It can be written as the LTL formula: $\varphi_4 = (\square\diamondsuit\,\mathsf{r3}) \wedge (\square\diamondsuit\,\mathsf{r4}) \wedge (\square\diamondsuit\,\mathsf{r6})$. It took $0.01s$ for Algorithm 3.1 to find the optimal plan: "$r1\,c1\,r4\,c1\,c2\,c3\,r3\,c3\,(r6\,c3\,c2\,c1\,r4\,c1\,c2\,c3\,r3\,c3)^{\omega}$", with the prefix cost 1021 and suffix cost 1. The final trajectory is shown in Figure 3.6. It can be seen that the robot patrols these three rooms as required.

3.5 Summary

In this chapter, we presented the framework to synthesize the hybrid control strategy for a mobile robot to fulfill a high-level temporal task. We started from constructing the finite abstraction of the robot's motion within the workspace. Then we proposed an automated scheme to synthesize the discrete motion and task plan. At last, a hybrid control strategy was designed for the robot to execute this discrete plan such that its resulting trajectory satisfies the given task.

Chapter 4

Knowledge transfer in partially-known workspace

THE FRAMEWORK presented in Chapter 3 assumes that the workspace is fully-known in priori and remains static. As a result, the discrete motion plan is synthesized once off-line and executed by the hybrid controller regardless of the actual measurements of the robot during execution. However in many real-life applications, the workspace may only be partially known or even dynamic. In order to address this issue, we firstly propose a new synthesis algorithm for the discrete motion plan that maximally satisfies a partially infeasible task with soft and hard constraints. Moreover, while executing this plan, the robot may gather new observations about the workspace. Thus a real-time plan adaption algorithm is proposed to ensure that the hard constraints for safety are always fulfilled while the satisfaction for the soft constraints is improved gradually for performance. At last, we extend this scheme to multi-robot systems where the robots share their knowledge about the workspace through local interaction with each other. Numerical studies are provided in the end to validate this framework.

4.1 Infeasible tasks

We follow the same notation as in Chapter 3. The initial task φ assigned to the robot might be infeasible by Definition 3.6, e.g., either the task is actually infeasible or the task is feasible but the initial workspace model $\mathcal{T}_c$ is incomplete or incorrect. The nominal solution proposed in Chapter 3 would simply returns failure. An intriguing question to ask is as follows:

Problem 4.1. *Assume the task specification is* ***infeasible****, how should the specification be* ***relaxed*** *and more importantly how to synthesize the discrete plan that satisfies the original specification* ***as much as possible****?*

The work by [83] partially answers the above problem. It generates a relaxed specification automaton $\mathcal{A}'_\varphi$ which is close to $\mathcal{A}_\varphi$ and feasible over $\mathcal{T}_c$, see Section III-

C of [83]. Then the discrete plan can be synthesized by following the procedure as described in Section 3.3. However there are often more than one accepting run within $\mathcal{T}_c \otimes \mathcal{A}'_\varphi$ and they may fulfill the original specification to different extents. Instead we aim to firstly find the discrete plan that fulfills the task *the most* by certain criterion, based on which then the specification automaton is relaxed.

4.1.1 Relaxed product automaton

By Definition 3.6, φ is infeasible when the standard PBA $\mathcal{A}_p$ does not have an accepting run. Thus we need to relax the constraints imposed by $\mathcal{A}_\varphi$ to allow more transitions within $\mathcal{A}_p$, by the relaxed PBA below.

Definition 4.1. The relaxed PBA $\mathcal{A}_r = \mathcal{T}_c \times \mathcal{A}_\varphi = (Q', 2^{AP}, \delta', Q'_0, \mathcal{F}', W_r)$ is defined as: $Q' = \Pi \times Q = \{\langle \pi, q \rangle \,|\, \forall \pi \in \Pi, \forall q \in Q\}$; 2^{AP} is the alphabet: $AP = \{a_1, a_2, \cdots, a_K\}$; $\delta' : Q' \to 2^{Q'}$. $\langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$ if and only if $(\pi_i, \pi_j) \in \longrightarrow_c$ and $\exists\, l \in 2^{AP}$ such that $q_n \in \delta(q_m, l)$; $Q'_0 = \Pi_0 \times Q_0$ and $\mathcal{F}' = \Pi \times \mathcal{F}$ are the initial and accepting states; $W_r : Q' \times Q' \to \mathbb{R}^+$is the weight function to be defined. ▲

Two differences between $\mathcal{A}_r$ and $\mathcal{A}_p$ from Definition 3.9 are: (i) the constraint "$q_n \in \delta(q_m, L_c(\pi_i))$" when defining δ' is relaxed to "$\exists\, l \in 2^{AP}$ such that $q_n \in \delta(q_m, l)$" here; (ii) the weight function W_r is defined differently from W_p. We firstly introduce the evaluation function $\texttt{Eval} : 2^{AP} \to \{0, 1\}^K$:

$$\texttt{Eval}(l) = \nu \Longleftrightarrow [\, \nu_i \,] = \begin{cases} 1 & \text{if } a_i \in l, \\ 0 & \text{if } a_i \notin l, \end{cases} \tag{4.1}$$

where $i = 1, \cdots, K$; $l \in 2^{AP}$ and $\nu \in \{0, 1\}^K$. Namely, each subset of 2^{AP} is mapped to a K-dimensional Boolean vector. Then a distance function between two input alphabets $\rho : 2^{AP} \times 2^{AP} \to \mathbb{N}$ is defined as:

$$\rho(l, l') = \|\nu - \nu'\|_1 = \sum_{i=1}^{K} |\, \nu_i - \nu'_i \,|, \tag{4.2}$$

where $\nu = \texttt{Eval}(l)$, $\nu' = \texttt{Eval}(l')$ and $l, l' \in 2^{AP}$. $\|\cdot\|_1$ is the ℓ_1 norm. Then we could define the distance between an element $l \in 2^{AP}$ to a set $\chi \subseteq 2^{AP} (\chi \neq \varnothing)$ [16]:

$$\texttt{Dist}(l, \chi) = \begin{cases} 0 & \text{if} \quad l \in \chi, \\ \min_{l' \in \chi} \rho(l, l') & \text{otherwise.} \end{cases} \tag{4.3}$$

Note that $\texttt{Dist}(l, \chi)$ is not defined for $\chi = \varnothing$. An example of computing $\texttt{Dist}(\cdot)$ is given in Figure 4.1. Now we give the formal definition of W_r:

$$\begin{aligned} &W_r(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) \\ &= W_c(\pi_i, \pi_j) + \alpha \cdot \texttt{Dist}(L_c(\pi_i), \chi(q_m, q_n)), \end{aligned} \tag{4.4}$$

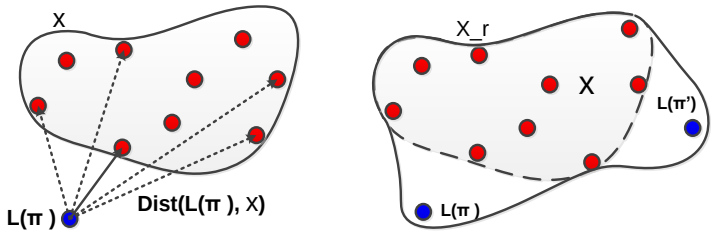


Figure 4.1: Left: the distance of $L_c(\pi)$ to a set of input alphabets χ (the solid line). Right: the input alphabets are revised by adding more elements.

where $\langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$; $\alpha \geq 0$ is a design parameter;

$$\chi(q_m, q_n) = \{l \in 2^{AP} | \, q_n \in \delta(q_m, l)\} \tag{4.5}$$

consists of all input alphabets that enable the transition from q_m to q_n in $\mathcal{A}_\varphi$. By Definition 4.1 there always exists $l \in 2^{AP}$ that $q_n \in \delta(q_m, l)$, thus $\chi(q_m, q_n) \neq \varnothing$ is ensured. $W_c(\pi_i, \pi_j)$ is the cost of the transition from π_i to π_j in $\mathcal{T}_c$; $\texttt{Dist}(L_c(\pi_i), \chi(q_m, q_n))$ measures how much the transition from π_i to π_j violates the constraints imposed by the transition from q_m to q_n. The design parameter α reflects the relative penalty on violating the original specification, and also the user's preference on a plan that has less cost or that fulfills the task more.

Example 4.1. Consider the NBA in Figure 3.3 and the transition from state q_1 to q_3, $\chi(q_1, q_3) = \{\{a_2\}, \{a_2, a_1\}, \{a_2, a_3\}, \{a_2, a_1, a_3\}\}$. Then $\texttt{Dist}(\{a_1\}, \chi(q_1, q_3)) = \rho(\{a_1\}, \{a_1, a_2\}) = 1$; $\texttt{Dist}(\{a_2\}, \chi(q_1, q_3)) = \rho(\{a_2\}, \{a_2\}) = 0$; $\texttt{Dist}(\{a_2, a_3\}, \chi(q_1, q_3)) = \rho(\{a_2, a_3\}, \{a_2, a_3\}) = 0$. ▲

Balanced accepting run

Since $\mathcal{A}_p$ does not have an accepting run, we instead search for an accepting run within $\mathcal{A}_r$. However the existence of an accepting run alone is not enough because: (i) they have different implementation costs; (ii) we would like to measure how much they violate the original specification. Thus we still consider the accepting runs with the *prefix-suffix* structure by (3.7):

$$\begin{aligned} R &= q'_0 q'_1 \cdots q'_{f-1} [\, q'_f q'_{f+1} \cdots\cdots q'_n \,]^\omega \\ &= \langle \pi_0, q_0 \rangle \cdots \langle \pi_{f-1}, q_{f-1} \rangle \, [\, \langle \pi_f, q_f \rangle \langle \pi_{f+1}, q_{f+1} \rangle \cdots\cdots \langle \pi_n, q_n \rangle \,]^\omega \,, \end{aligned} \tag{4.6}$$

where $q'_0 = \langle \pi_0, q_0 \rangle \in Q'_0$ and $q'_f = \langle \pi_f, q_f \rangle \in \mathcal{F}'$. Its total cost is calculated differently:

$$\texttt{Cost}(R, \mathcal{A}_r) = \texttt{cost}_\tau + \alpha \cdot \texttt{dist}_\varphi \,, \tag{4.7}$$

where the accumulated implementation cost of the motion plan $\tau = R|_\Pi$ is

$$\texttt{cost}_\tau = \Big(\sum_{i=0}^{f-1} + \gamma \sum_{i=f}^{n-1} \Big) W_c(\pi_i, \pi_{i+1}); \tag{4.8}$$

the accumulated distance of τ to $\mathcal{A}_\varphi$ is

$$\texttt{dist}_\varphi = \Big(\sum_{i=0}^{f-1} + \gamma \sum_{i=f}^{n-1} \Big) \texttt{Dist}(L_c(\pi_i), \chi(q_i, q_{i+1})); \tag{4.9}$$

the design parameter $\gamma \geq 0$ represents the relative weighting on the cost of transient response (the prefix) and steady response (the suffix).

Problem 4.2. *Find the accepting run of $\mathcal{A}_r$ that minimizes the cost by* (4.7). ▲

We call the solution to Problem 4.2 the *balanced* accepting run of $\mathcal{A}_r$, denoted by R_{bal}. The corresponding balanced plan is $\tau_{\text{bal}} = R_{\text{bal}}|_\Pi$.

***Remark* 4.1.** As mentioned in Section 3.3, each transition of the NBA is encoded by a boolean expression accepting all alphabets that enable this transition. This expression can be represented as a binary decision diagram (BBD) such that the distance function by (4.3) can be easily integrated. The idea is that for operator "$\vee$" it returns the *minimal* distance of its left and right branches, while for operator "$\wedge$" it returns the *summed* distance of them. Thus it is not necessary to enumerate all input alphabets to evaluate the distance. More details can be found in Section 8. ▲

4.1.2 Balanced plan synthesis

Given the values of α and γ, $\mathcal{A}_r$ can be either constructed fully by Definition 4.1 or on-the-fly as before. But the weight function is computed based on the distance function from (4.3). Consequently, given the value of α, γ, Algorithm 3.1 can be called with respect to the full or on-the-fly construction of $\mathcal{A}_r$, to derive the balanced accepting run R_{bal}. Then the corresponding balanced plan is $\tau_{\text{bal}} = R_{\text{bal}}|_\Pi$.

***Remark* 4.2.** Although $\mathcal{A}_r$ allows more transitions compared ro $\mathcal{A}_p$, any balanced plan is a valid path of $\mathcal{T}_c$, i.e., the transition relation of $\mathcal{T}_c$ is never relaxed when constructing $\mathcal{A}_r$. Thus τ_{bal} is always implementable. ▲

Furthermore, given R_{bal}, $\mathcal{T}_c$ and $\mathcal{A}_\varphi$, we can easliy computes the associated $\texttt{cost}_\tau$, $\texttt{dist}_\varphi$ and the relaxed specification automaton $\mathcal{A}'_\varphi$. While iterating through the transitions along R_{bal}, we can construct $\mathcal{A}'_\varphi$ by adding new transitions to $\mathcal{A}_\varphi$; compute $\texttt{cost}_\tau$ and $\texttt{dist}_\varphi$ as defined in (4.8) and (4.9). It can be verified that the obtained $\mathcal{A}'_\varphi$ is a valid relaxation [83] of $\mathcal{A}_\varphi$. Note each R_{bal} corresponds to a balanced plan τ_{bal} and a revised specification automaton $\mathcal{A}'_\varphi$.

Lemma 4.1. *If* $\texttt{dist}_\varphi = 0$*, then* $\tau_{\text{bal}} \vDash \varphi$.

Proof. Since $\texttt{Dist}(\cdot) \geq 0$ by (4.3), the accumulated distance $\texttt{dist}_\varphi = 0$ implies $q_n \in \delta(q_m, L_c(\pi_i))$ for all transitions $(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle)$ along R_{bal}. Since $\mathcal{A}_p$ and $\mathcal{A}_r$ have the same states with the same sets of initial and accepting states, R_{bal} is also an accepting run for $\mathcal{A}_p$ by Definition 3.9. Then its corresponding plan τ_{bal} satisfies φ by Lemma 3.3. ■

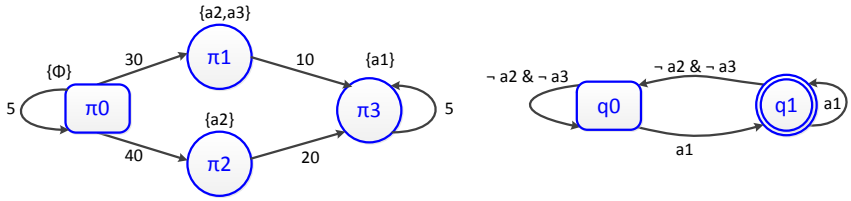


Figure 4.2: Left: the FTS $\mathcal{T}_c$ with labels and weights. Transitions are labeled by the costs. Right: the NBA $\mathcal{A}_\varphi$ associated with $\varphi = \Diamond \square\, a_1 \wedge \square\neg(a_2 \wedge a_3)$.

However it may not be trivial to determine the appropriate value of α for the desired balance between the cost and distance to the task. As an extension, Algorithm 3.1 could be called under different α to generate various balanced accepting runs, among which the unique ones are saved as the candidates. They can be compared regarding the associated $\texttt{cost}_\tau$ and $\texttt{dist}_\varphi$. The chosen τ_{opt} can be executed off-line by constructing the hybrid controller as proposed in Algorithm 3.2.

Theorem 4.2. *If φ is feasible over $\mathcal{T}_c$, the balanced plan τ_{bal} **satisfies** φ if $\alpha > \underline{\alpha}$, where $\underline{\alpha}$ is given by* (4.10).

Proof. If φ is feasible over $\mathcal{T}_c$, Algorithms 3.1 return the optimal accepting run R_{opt} with the total cost (under the same γ) by (3.9):

$$\texttt{Cost}(R_{\text{opt}}, \mathcal{A}_p) = \underline{\alpha}. \tag{4.10}$$

Clearly, R_{opt} is also a valid accepting run of $\mathcal{A}_r$ since $\mathcal{A}_p$ and $\mathcal{A}_r$ have the same states with the same sets of initial and accepting states. Moreover, under the same γ, $\texttt{Cost}(R_{\text{opt}}, \mathcal{A}_p) = \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_r)$. Assuming that τ_{bal} does not satisfy φ, then $\texttt{dist}_\varphi \geq 1$ by (4.9). As a result, the total cost of R_{bal} by (4.7) satisfies: $\texttt{Cost}(R_{\text{bal}}, \mathcal{A}_r) > \alpha \cdot \texttt{dist}_\varphi > \alpha$. Since $\alpha > \underline{\alpha} = \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_p) = \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_r)$, it implies $\texttt{Cost}(R_{\text{bal}}, \mathcal{A}_r) > \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_r)$. However by the definition of the balanced run, R_{bal} is the accepting run of $\mathcal{A}_r$ with the least total cost, i.e., $\texttt{Cost}(R_{\text{bal}}, \mathcal{A}_r) \leq \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_r)$, which leads to a contradictory. Thus the proposed method can be applied directly when φ is feasible over $\mathcal{T}_c$ but choosing $\alpha > \underline{\alpha}$. Algorithm 3.1 will automatically select the accepting run that satisfies φ, i.e., $\texttt{dist}_\varphi = 0$. ∎

Example 4.2. As shown in Figure 4.2, the robot has to go from region π_0 to π_3 and stay there, meanwhile avoid all regions satisfying properties a_2 or a_3. Three alternative plans are obtained by varying α ($\gamma = 5$), as shown in Figure 4.3: (i) when the penalty on violating φ is low, $\mathcal{A}_\varphi$ is revised by adding q_1 to $\delta(q_0, \varnothing)$, q_1 to $\delta(q_1, \varnothing)$ and the balanced plan is $[\pi_0]^\omega$ (black hexagram, $\texttt{cost}_\tau$ 30, $\texttt{dist}_\varphi$ 6); (ii) when the penalty is increased, $\mathcal{A}_\varphi$ is revised by adding q_1 to $\delta(q_0, \{a_2, a_3\})$, where the balanced plan is $\pi_0\,\pi_1\,[\pi_3]^\omega$ (blue square, $\texttt{cost}_\tau$ 65, $\texttt{dist}_\varphi$ 2); (iii) when the penalty is severe, $\mathcal{A}_\varphi$ is revised by adding q_1 to $\delta(q_0, \{a_2\})$, where the balanced plan

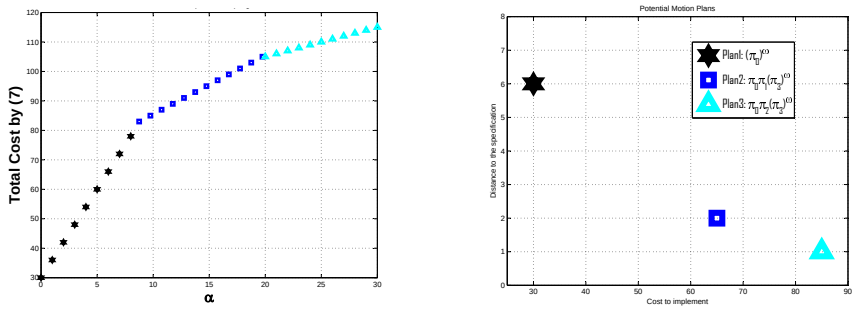


Figure 4.3: Left: the total cost of the balanced runs when $\gamma = 5$ under different α. Right: the unique balanced runs, located by their $\texttt{cost}_\tau$ (x-axis) and $\texttt{dist}_\varphi$ (y-axis).

is $\pi_0\,\pi_2\,[\pi_3]^\omega$ (cyan triangle, $\texttt{cost}_\tau$ 85, $\texttt{dist}_\varphi$ 1). Note that in (iii) the robot passes through π_2 which satisfies only a_2, instead of π_1 which satisfies both a_2 and a_3. ▲

4.2 Soft and hard specifications

The previous section presents how to synthesis a balanced motion and task plan that satisfies the potentially-infeasible task as much as possible. However, sometimes a specification contains two distinctive parts: one part for hard constraints that concerns safety or security and should not be violated for all time; another part for soft constraints and additional achievement that should be satisfied as much as possible. In this section, we propose a solution that meets these requirements.

Problem formulation

The robot's finite transition system is still denoted by $\mathcal{T}_c$ from Definition 3.3. Its task specification remains an LTL formula over AP, but with the following structure:

$$\varphi = \varphi^{\text{soft}} \wedge \varphi^{\text{hard}}, \tag{4.11}$$

where φ^{soft} and φ^{hard} are "soft" and "hard" sub-formulas; φ^{hard} could include safety constraints like collision avoidance: "avoid all obstacles" or power-supply guarantee: "visit the charging station infinitely often"; φ^{soft} could include performance requirements like "collect as many objects" but the location of some objects is not known. Introducing soft and hard specifications is due to the observation that the partially-known workspace considered in Section 4.3 might render parts of the specification infeasible initially and thus yield the needs for them to be relaxed, while the safety-critical parts should not be relaxed during the process.

Problem 4.3. *Given the task specification by* (4.11)*, how to synthesize the motion and task plan such that* φ^{hard} *is satisfied* ***fully****, while* φ^{soft} *is satisfied* ***the most****?* ▲

4.2.1 Safety-ensured product automaton

Now there are two levels of specifications by φ^{hard} and φ^{soft}, with respect to which the property of a plan can be stated more specifically.

Definition 4.2. An finite or infinite path $\tau = \pi_1\pi_2\cdots$ of $\mathcal{T}_c$ is called: (i) **valid** if $(\pi_i, \pi_{i+1}) \in \longrightarrow_c$, for $i = 1, 2, \cdots$; (ii) **safe** if $\tau \vDash \varphi^{\text{hard}}$; (iii) **satisfying** if $\tau \vDash \varphi$. ▲

Let $\mathcal{A}^{\text{hard}} = (Q_1, 2^{AP}, \delta_1, Q_{1,0}, \mathcal{F}_1)$ and $\mathcal{A}^{\text{soft}} = (Q_2, 2^{AP}, \delta_2, Q_{2,0}, \mathcal{F}_2)$ be the NBA associated with φ^{hard} and φ^{soft}, respectively. Detailed definition can be found in Section 3.3. The functions $\chi_1(\cdot)$ of $\mathcal{A}^{\text{hard}}$ and $\chi_2(\cdot)$ of $\mathcal{A}^{\text{soft}}$ are defined analogously as (4.5). Now we propose a way to construct the relaxed but safety-ensured intersection of $\mathcal{A}^{\text{hard}}$ and $\mathcal{A}^{\text{soft}}$.

Definition 4.3. The relaxed intersection of $\mathcal{A}^{\text{hard}}$ and $\mathcal{A}^{\text{soft}}$ is defined by:

$$\tilde{\mathcal{A}}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F}), \tag{4.12}$$

where $Q = Q_1 \times Q_2 \times \{1, 2\}$; $Q_0 = Q_{1,0} \times Q_{2,0} \times \{1\}$; $\mathcal{F} = \mathcal{F}_1 \times Q_2 \times \{1\}$; $\delta : Q \times 2^{AP} \to 2^Q$, with $\langle \check{q}_1, \check{q}_2, \check{t} \rangle \in \delta(\langle q_1, q_2, t \rangle, l)$ when three conditions hold: (1) $l \in \chi_1(q_1, \check{q}_1)$; (2) $\chi_2(q_2, \check{q}_2) \neq \varnothing$; (3) $q_t \notin \mathcal{F}_t$ and $\check{t} = t$, *or* $q_t \in \mathcal{F}_t$ and $\check{t} = \texttt{mod}(t, 2) + 1$, where $t \in \{1, 2\}$ and `mod` is the modulo operation. ▲

Algorithm 4.3 constructs $\tilde{\mathcal{A}}_\varphi$ by Definition 4.3. Note that $\tilde{\mathcal{A}}_\varphi$ remains a Büchi automaton. We relax the requirement that there should exist a common input alphabet that enables the transitions from q_i to $\check{q}_i$ for $i \in \{1, 2\}$, compared with the standard definition of Büchi automata intersection (see Chapter 4.3 of [11]). An accepting run r of $\tilde{\mathcal{A}}_\varphi$ intersects with the accepting set $\mathcal{F}$ infinitely often. The last component $t \in \{1, 2\}$ in Q ensures that r has to intersect with both $\mathcal{F}_1 \times Q_2 \times \{1\}$ and $Q_1 \times \mathcal{F}_2 \times \{2\}$. This fact is used in the proof of Theorem 4.3 below. Denote by $r|_{Q_1}$ and $r|_{Q_2}$ the projection of r onto the states of $\mathcal{A}^{\text{hard}}$ and $\mathcal{A}^{\text{soft}}$, respectively.

Theorem 4.3. *Given an accepting run r of $\tilde{\mathcal{A}}_\varphi$, $r|_{Q_1}$ is an accepting run of $\mathcal{A}^{\text{hard}}$. Moreover, $\mathcal{L}_\omega(\tilde{\mathcal{A}}_\varphi) \subseteq \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$.*

Proof. By definition, at least one of accepting states in $\mathcal{F}$ should appear in r infinitely often. The projection of $\mathcal{F}$ onto Q_1 is $\mathcal{F}_1$, therefore one of the accepting states in $\mathcal{F}_1$ is visited infinitely often by $r|_{Q_1}$. Secondly, since $l \in \chi_1(q_1, \check{q}_1)$ is ensured by Definition 4.3, all transitions along r are valid for $\mathcal{A}^{\text{hard}}$. As a result, $r|_{Q_1}$ is an accepting run of $\mathcal{A}^{\text{hard}}$. For the second part, given any infinite word $\sigma \in \mathcal{L}_\omega(\tilde{\mathcal{A}}_\varphi)$, σ results in an accepting run of $\tilde{\mathcal{A}}_\varphi$ by Definition 3.8, denoted by r_σ. It has been proved that $r_\sigma|_{Q_1}$ is also an accepting run of $\mathcal{A}^{\text{hard}}$, which implies that $\sigma \in \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$. Thus for any $\sigma \in \mathcal{L}_\omega(\tilde{\mathcal{A}}_\varphi)$, $\sigma \in \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$ holds, namely $\mathcal{L}_\omega(\tilde{\mathcal{A}}_\varphi) \subseteq \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$. ■

Since we need to guarantee that φ^{hard} is fulfilled fully and φ^{soft} is satisfied as much as possible, we rely on the relaxed product automaton proposed previously to handle both feasible and potentially infeasible specifications.

Algorithm 4.3 Relaxed intersection, `RelaxInt( )`

Input: $\mathcal{A}^{\text{soft}}$, $\mathcal{A}^{\text{hard}}$
Output: $\tilde{\mathcal{A}}_\varphi$
foreach $q_1 \in Q_1$, $q_2 \in Q_2$, $t \in \{1, 2\}$ **do**
  $q_m = \langle q_1, q_2, t\rangle \in Q$
  If $(q_1 \in Q_{1,0}, q_2 \in Q_{2,0}, t \in \{1\})$, then $q_m \in Q_0$
  If $(q_1 \in \mathcal{F}_1, t \in \{1\})$, then $q_m \in \mathcal{F}$
  foreach $\check{q}_1 \in \texttt{Post}(q_1)$, $\check{q}_2 \in \texttt{Post}(q_2)$, $\check{t} \in \{1, 2\}$ **do**
    $q_n = \langle \check{q}_1, \check{q}_2, \check{t}\rangle \in Q$
    if $\big(q_1 \notin \mathcal{F}_1, t \in \{1\}, \check{t} \in \{1\}\big)$ *or* $\big(q_2 \notin \mathcal{F}_2, t \in \{2\}, \check{t} \in \{2\}\big)$ *or* $\big(q_1 \in \mathcal{F}_1, t \in \{1\}, \check{t} \in \{2\}\big)$ *or* $\big(q_2 \in \mathcal{F}_2, t \in \{2\}, \check{t} \in \{1\}\big)$ **then**
      $q_n \in \delta(q_m, l), \ \forall l \in \chi_1(q_1, \check{q}_1)$
return $\tilde{\mathcal{A}}_\varphi$

Definition 4.4. The safety-ensured and relaxed product Büchi automaton $\tilde{\mathcal{A}}_r = \mathcal{T}_c \times \tilde{\mathcal{A}}_\varphi = (Q', \delta', Q'_0, \mathcal{F}', W_p)$ is defined as follows: $Q' = \Pi \times Q = \{\langle \pi, q\rangle \,|\, \forall \pi \in \Pi, \forall q \in Q\}$; $\delta' : Q' \to 2^{Q'}$. $\langle \pi_j, q_n\rangle \in \delta'(\langle \pi_i, q_m\rangle)$ if and only if $(\pi_i, \pi_j) \in \longrightarrow_c$ and $q_n \in \delta(q_m, L_c(\pi_i))$; $Q'_0 = \Pi_0 \times Q_0$ and $\mathcal{F}' = \Pi \times \mathcal{F}$ are the initial and accepting states; $W_r : Q' \times Q' \to \mathbb{R}^+$ is the weight function:

$$\begin{aligned} &W_r(\langle \pi_i, q_m\rangle, \langle \pi_j, q_n\rangle) \\ =&W_c(\pi_i, \pi_j) + \alpha \cdot \texttt{Dist}(L_c(\pi_i), \chi_2(q_2, \check{q}_2)) \end{aligned} \tag{4.13}$$

where $\langle \pi_j, q_n\rangle \in \delta'(\langle \pi_i, q_m\rangle)$; $q_m = \langle q_1, q_2, t\rangle$; $q_n = \langle \check{q}_1, \check{q}_2, \check{t}\rangle$; $\alpha \geq 0$ is a design parameter; function $\texttt{Dist}(\cdot)$ is defined in (4.3). ▲

The weight function consists of two parts: $W_c(\pi_i, \pi_j)$ measures the implementation cost of the transition from π_i to π_j; $\texttt{Dist}(L_c(\pi_i), \chi_2(q_2, \check{q}_2))$ measures how much this transition violates the constraints imposed by $\mathcal{A}^{\text{soft}}$; α reflects the relative penalty on violating the soft specification.

Theorem 4.4. *Assume R is an accepting run of $\tilde{\mathcal{A}}_r$. Its projection on Π, $\tau = R|_\Pi$, is both **valid** and **safe** for $\mathcal{T}_c$ and φ by Definition 4.2.*

Proof. Firstly, τ is valid since every transition in δ' corresponds to a valid transition within $\longrightarrow_c$. Then since R is an accepting run of $\tilde{\mathcal{A}}_r = \mathcal{T}_c \times \tilde{\mathcal{A}}_\varphi$, then $\texttt{trace}(\tau) \in \mathcal{L}_\omega(\tilde{\mathcal{A}}_\varphi)$, i.e., $\texttt{trace}(\tau) \in \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$ by Theorem 4.3. Since $\texttt{Words}(\varphi^{\text{hard}}) = \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$ by Lemma 3.1, $\texttt{trace}(\tau) \in \texttt{Words}(\varphi^{\text{hard}})$, i.e., $\tau \vDash \varphi^{\text{hard}}$, thus τ is safe. ■

Same as before, to measure the implementation cost of different accepting runs of $\tilde{\mathcal{A}}_r$ and how much they violate the soft specification, we consider the accepting

runs with the *prefix-suffix* structure by (4.6). However its total cost is defined by:

$$\texttt{Cost}(R, \tilde{\mathcal{A}}_r) = \sum_{i=0}^{f-1} W_r(q'_i, q'_{i+1}) + \gamma \sum_{i=f}^{n-1} W_r(q'_i, q'_{i+1}) = \texttt{cost}_\tau + \alpha \cdot \texttt{dist}_{\varphi^{\text{soft}}}, \quad (4.14)$$

where $\gamma \geq 0$; the implementation cost of $\tau = R|_\Pi$ is $\texttt{cost}_\tau = (\sum_{i=0}^{f-1} + \gamma \sum_{i=f}^{n-1}) W_c(\pi_i, \pi_{i+1})$; the accumulated distance of τ with respect to $\mathcal{A}_{\varphi^{\text{soft}}}$ is $\texttt{dist}_{\varphi^{\text{soft}}} = (\sum_{i=0}^{f-1} + \gamma \sum_{i=f}^{n-1}) \texttt{Dist}(L_c(\pi_i), \chi_2(q'_i|_{Q_2}, q'_{i+1}|_{Q_2}))$, where $q'_i|_{Q_2}$ and $q'_{i+1}|_{Q_2}$ are the projection of q'_i and q'_{i+1} onto Q_2.

Problem 4.4. *Find the accepting run of* $\tilde{\mathcal{A}}_r$ *that* ***minimizes*** *the cost by* (4.14). ▲

We call the solution to Problem 4.4 as the *safe* accepting run of $\tilde{\mathcal{A}}_r$, denoted by R_{safe}. The corresponding *safe* plan is $\tau_{\text{safe}} = R_{\text{safe}}|_\Pi$.

4.2.2 Safe plan synthesis

Given the values of α and γ, $\tilde{\mathcal{A}}_r$ can be either constructed fully by Definition 4.1 or on-the-fly construction. But now the distance d reflects whether the input alphabet violates the hard specification and the distance to the set of input alphabets for the soft specification. Then the safe accepting run R_{safe} can be obtained in the prefix-suffix format by calling Algorithm 3.1 with respect to $\tilde{\mathcal{A}}_r$. By Theorem 4.4, its corresponding plan τ_{safe} is always valid and safe no matter how α and γ are chosen.

Same as in Section 4.1, the value of α could be tuned by calling Algorithm 3.1 under different α to generate candidates of R_{safe}. The associated $\texttt{cost}_\tau$ and $\texttt{dist}_{\varphi^{\text{soft}}}$ can be computed similarly. After deciding the safe accepting run, its corresponding plan τ_{safe} can be implemented by the hybrid control strategy following Algorithm 3.2.

Lemma 4.5. *If* $\texttt{dist}_{\varphi^{\text{soft}}} = 0$, *then* $\tau_{\text{safe}} \vDash \varphi$.

Proof. By Lemma 4.1, if $\texttt{dist}_{\varphi^{\text{soft}}} = 0$, R_{safe} is an accepting run for the un-relaxed product $\mathcal{T}_c \times \mathcal{A}_\varphi$ by Definition 4.1, where $\mathcal{A}_\varphi$ is the un-relaxed intersection [11] of $\mathcal{A}_{\varphi^{\text{soft}}}$ and $\mathcal{A}_{\varphi^{\text{hard}}}$. Thus its corresponding plan τ_{safe} satisfies φ by Lemma 3.3. ■

4.3 On-line plan adaptation

It is rarely the case that the system model, i.e., the transition system, is consistent with the actual workspace and robot dynamics. It means that the motion and task plan synthesized off-line may not be executed as expected. As a result, a real-time on-line control framework is needed, where the planning and execution are interleaved as shown in Figure 4.4. If the actual workspace is different from the workspace model, it is crucial to put the planner on-line and make it dynamic, such that it can monitor the plan execution, update the system model based on the real-time observation, and validate or revise the current plan.

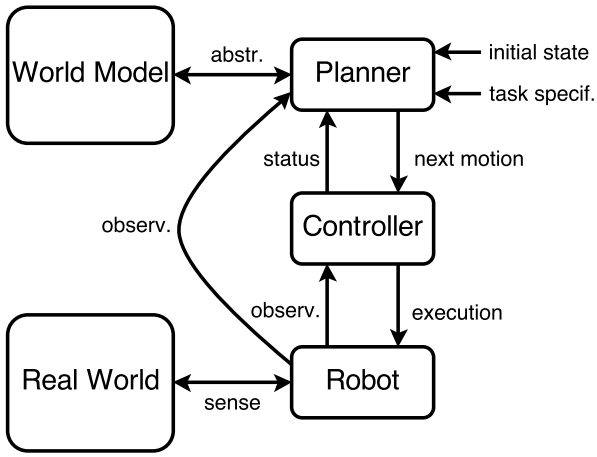


Figure 4.4: Diagram for the on-line planning and execution module.

Thus we discuss here how the FTS could be updated based on the robot's observations that come from both its sensing or communication functionality. We denoted by $\mathcal{T}_c^t$ the transition system at time $t \geq 0$, particularly

$$\mathcal{T}_c^t = (\Pi, \longrightarrow_c^t, \Pi_0, AP, L_c^t, W_c^t), \tag{4.15}$$

where the superscript indicates the time. Note that (i) the set of regions Π is static, meaning that no new regions are added or existing regions are removed; (ii) the set of initial states Π_0 and the set of known atomic propositions AP are also static as before. Furthermore, its task specification consists of hard and soft parts: $\varphi = \varphi^{\text{soft}} \wedge \varphi^{\text{hard}}$, as introduced in (4.11), which is invariant after the system starts. Denote by $\tilde{\mathcal{A}}_\varphi$ the relaxed intersection automaton from Algorithm 4.3.

Problem 4.5. *Assume the workspace is partially-known. The following problems are posed: (i) how to* ***model*** *the robot's sensing and communication functionalities; (ii) how to* ***update*** *the transition system accordingly; (iii) how to guarantee the motion and task plan is always* ***valid and safe****.* ▲

4.3.1 Initial plan synthesis

At $t = 0$, the initial motion and task plan can be obtained by the method proposed in Section 4.2, given the initial transition system $\mathcal{T}_c^0$ and φ. Denote by R^t, $\tau^t = R^t|_\Pi$, $\tilde{\mathcal{A}}_r^t$ the obtained safe accepting run, the safe plan and the relaxed product automaton at time $t \geq 0$. In this section, we assume that $\tilde{\mathcal{A}}_r^t$ is constructed on-the-fly and R^t is obtained by Algorithm 3.1 for $\tilde{\mathcal{A}}_r^t$. Note that the soft specification may be infeasible initially but it is guaranteed by Theorem 4.4 that τ^0 is safe and valid for $\mathcal{T}_c^0$.

4.3.2 Real-time knowledge update

The robot we consider has both the sensing ability to discover the workspace and the communication functionality to communicate with external sources. In this section, we discuss how these functionalities can be modeled.

Denote by $\mathbf{Sense}^t$ as the set of sensing information obtained at time $t \geq 0$. Note that this information might be gathered when a robot reaches a region or during the transition from one region to another. It has the following format:

$$\mathbf{Sense}^t = \{((\pi, S, S_\neg), E, E_\neg)\}, \tag{4.16}$$

where $\pi \in \Pi$ stands for a perceived region; $S \subseteq AP$ is the set of propositions satisfied by this region; $S_\neg \subseteq AP$ is the set of propositions not satisfied by this region; $(\pi_i, \pi_j, w) \in E$ if (π_i, π_j) needs to be added to $\longrightarrow_c^t$ with weight w or its weight is updated to w; $(\pi_i, \pi_j) \in E_\neg$ if (π_i, π_j) needs to be removed from $\longrightarrow_c^t$. $\mathbf{Sense}^t$ reflects the actual workspace at time t.

Example 4.3. The sensing info $((\pi_1, \{a_1, a_3\}, \{a_2\}), (\pi_1, \pi_2, 10), (\pi_1, \pi_3)) \in \mathbf{Sense}^t$ is received if it is observed that region π_1 satisfies proposition a_1 and a_3 but not a_2; $(\pi_1, \pi_2, 10) \in E$ if the transition from π_1 to π_2 is allowed with the cost 10; $(\pi_1, \pi_3) \in E_\neg$ if the transition from π_1 to π_3 is invalid. ▲

This sensing function can be modeled by assigning a sensing radius $h > 0$, such that all regions intersecting with the sphere $\{y \in \mathbb{R}^n \,|\, |y - x(t)| \leq h\}$ are visible, where $x(t) \in \mathbb{R}^n$ is the robot's position at time t. Different post-processing techniques might be used to abstract the essential information for (4.16) from raw sensing data.

Besides, communication with external sources is another important mean to retrieve information. This source can be another robot, a control base station, or even an on-line database. Whenever this robot communicates with the external source at time t, it sends the following request message:

$$\mathbf{Request}^t = \varphi|_{AP}\,, \tag{4.17}$$

which informs the external source the set of workspace properties this robot is interested in. The reply message it gets has the following format:

$$\mathbf{Reply}^t = \{(\pi', S', S'_\neg)\}, \tag{4.18}$$

where $S' \subseteq (\varphi|_{AP})$ and $S'_\neg \subseteq (\varphi|_{AP})$; S' and $S'_\neg$ can not both be empty; $\pi' \in \Pi$ is the region that satisfies S' but not $S'_\neg$. Note that S' and $S'_\neg$ only contain propositions that are relevant to the task φ. Depending on the actual communication protocol, the external source can decide how often it replies to the robot's request.

Example 4.4. The reply information $(\pi_1, \{a_1\}, \{a_2\}) \in \mathbf{Reply}^t$ is received if region π_1 satisfies proposition a_1 but not a_2. ▲

Transition system update

Thus at time t, the robot might obtain new knowledge from $\mathbf{Sense}^t$ and $\mathbf{Reply}^t$ as described before, based on which it needs to update its own system model. Denote by $\mathcal{T}_c^{t^-}$ and $\mathcal{T}_c^{t^+}$ as the transition system before and after the update at time t. Recall that $(\pi, S, S_\neg) \in \mathbf{Sense}^t$ or $\mathbf{Reply}^t$ indicates that the region $\pi \in \Pi$ satisfies S but not $S_\neg$. Then $L_c^{t^+}(\pi) = L_c^{t^-}(\pi) \cup S \setminus S_\neg$, where $L_c^{t^+}(\pi)$ and $L_c^{t^-}(\pi)$ are the labeling function of π before and after the update. Regarding $E, E_\neg \in \mathbf{Sense}^t$, new transitions are added or some existing transitions' weight is updated based on E while transitions in $E_\neg$ are removed. For brevity, $\widetilde{\Pi} \subseteq \Pi$ is used to store the set of regions within Π of which the labeling function is changed during the update; $\widehat{\Pi}$ stores the set of regions of which the adjacency relation needs to be reconstructed. Note that if both $\mathbf{Sense}^t$ and $\mathbf{Reply}^t$ are empty, $\mathcal{T}_c^{t^+}$ remains the same as $\mathcal{T}_c^{t^-}$.

4.3.3 Safety-ensured plan revision

Since $\mathcal{T}_c^t$ might be updated as described in previous part, the motion and task plan from the initial synthesis in Section 4.2 needs to be evaluated regarding their validity, safety and optimality.

Product automaton update

Denote by $\tilde{\mathcal{A}}_r^{t^-}$ and $\tilde{\mathcal{A}}_r^{t^+}$ as the relaxed product automaton corresponding to $\mathcal{T}_c^{t^-}$ and $\mathcal{T}_c^{t^+}$, respectively. To update $\tilde{\mathcal{A}}_r^t$, a brute-force approach would be to reconstruct the complete $\tilde{\mathcal{A}}_r^t$ from scratch by Definition 4.4 using $\tilde{\mathcal{A}}_\varphi$ and $\mathcal{T}_c^{t^+}$, or to re-evaluate all transitions within $\tilde{\mathcal{A}}_r^{t^-}$ that are relevant to the latest changes in $\mathcal{T}_c^{t^+}$ as proposed in [59]. However, both methods have the complexity proportional to the number of transitions within $\mathcal{A}_\varphi$ and more importantly most of the updated transitions of $\tilde{\mathcal{A}}_r^{t^+}$ might not be used by the plan revision Algorithm 4.4 later.

Thus we propose to incorporate the update information including $\widetilde{\Pi}$, E and $E_\neg$ into the adjacency relation function of $\tilde{\mathcal{A}}_r^t$. For any region $\pi_i \in \widetilde{\Pi}$ whose labels or adjacency relation has been changed, they are stored in a new set $\widehat{\Pi}$, namely

$$\widehat{\Pi} = \widetilde{\Pi} \cup \{\pi_i \,|\, (\pi_i, \pi_j) \in E, \text{ or } (\pi_i, \pi_j, w) \in E_\neg\}. \tag{4.19}$$

Thus all the transitions originated from q_s' whose projection onto Π belongs to $\widehat{\Pi}$, i.e., $q_s'|_\Pi \in \widehat{\Pi}$, have to be re-constructed. In this way, the update information is used only when q_s' is revisited by the revision Algorithm 4.4 and then $\delta'(q_s')$ is re-constructed.

Given the updated transition system $\mathcal{T}_c^{t^+}$ and the current plan τ^t, two natural questions arise: (i) is τ^t still valid or safe? (ii) if not, how can we modify τ^t such that it remains valid and safe for $\mathcal{T}_c^{t^+}$ and φ?

Verifying validity and safety

Recall that the current accepting run R^t has a finite set of transitions appearing in it, i.e., $\texttt{Edge}(R^t)$ by (3.8). By checking $\texttt{Edge}(R^t)$, we could validate if the current plan τ^t is still valid or safe.

Definition 4.5. Given the updated transition system $\mathcal{T}_c^{t^+}$, any transition $(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) \in \texttt{Edge}(R^t)$ is called: (i) **invalid** if $(\pi_i, \pi_j) \notin \longrightarrow_c^{t^+}$; (ii) **unsafe** if $L_c^{t^+}(\pi_i) \notin \chi_1(q_m|_{Q_1}, q_n|_{Q_1})$. ▲

Recall that Q_1 is the set of states of $\mathcal{A}^{\text{hard}}$. We use Ξ and $\aleph$ to store the invalid and unsafe transitions in R^t, respectively. They are obtained by iterating through each transition $(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle)$ within $\texttt{Edge}(R^t)$ and checks if (π_i, π_j) has been removed from $\mathcal{T}_c^{t^+}$ or if the changed label $L_c^{t^+}(\pi_i)$ would make this transition unsafe.

Theorem 4.6. *Assume R^t is an accepting run of $\tilde{\mathcal{A}}_r^{t^-}$; $\mathcal{T}_c^{t^-}$ is updated to $\mathcal{T}_c^{t^+}$; Ξ, $\aleph$ are the sets of invalid and unsafe transitions from above. Then (i) τ^t remains **valid** if and only if $\Xi = \varnothing$; (ii) τ^t remains **safe** if $\aleph = \varnothing$.*

Proof. Since R^t is an accepting run of $\tilde{\mathcal{A}}_r^{t^-}$, τ^t is both valid and safe for $\mathcal{T}_c^{t^-}$ by Theorem 4.4. If $\Xi = \varnothing$ and $\aleph = \varnothing$, $\texttt{Edge}(R^t)$ does not contain any invalid or unsafe transitions. Thus R^t remains an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$. "If" part of (i): by Theorem 4.4, τ^t is valid since R^t remains an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$. "Only if" part of (i): if $\Xi \neq \varnothing$, τ^t contains at least one invalid transition, thus not valid by Definition 4.2. "If" part of (ii): by Theorem 4.4, τ^t is safe since R^t remains an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$. ■

Plan revision

If Ξ or $\aleph$ are not empty, τ^t might be invalid or unsafe. A plan revision scheme is needed to guarantee its validity and safety: one straightforward approach could be to recall Algorithm 3.1 with respect to $\mathcal{T}_c^{t^+}$ and $\mathcal{A}_\varphi$, but using the robot's current region π_{cur} from Algorithm 3.2 as the initial region. Let the derived accepting run and corresponding plan be R_{new} and τ_{new}. In the following, we show that even though τ_{new} is valid and safe as proved in Theorem 4.4, it can not guarantee the actual safety if we take into account the robot's past trajectory. Given the robot's past trajectory τ_{past} and past run R_{past} from Algorithm 3.2, its complete trajectory is obtained by concatenating τ_{past} with τ_{new}, namely $\tau_{\text{comp}} = \tau_{\text{past}} + \tau_{\text{new}}$. Note that τ_{comp} remains the suffix-suffix format. A key observation is that the safety property of τ_{new} does not ensure the safety of the robot's complete trajectory starting from time 0. In fact, this is because when analyzing the corresponding runs of τ_{past} and τ_{new} in $\tilde{\mathcal{A}}_r^{t^+}$, the product state q'_{cur} (the last state of R_{past}) from Algorithm 3.2 may not be the same as the first product state in R_{new}. As a result, these two segments can not be concatenated into an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$.

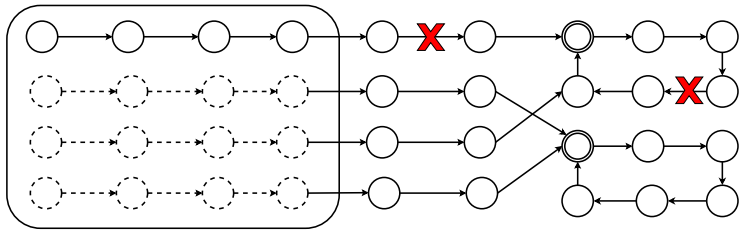


Figure 4.5: Locally revise the invalid transitions in R^t. The sequences of states in dashed line represent the corresponding runs given the robot's past trajectory.

Problem 4.6. *Assume* Ξ *or* $\aleph$ *are not empty. Given the robot's past trajectory* τ_{past}*, how to find the* ***new plan*** τ_{new} *such that its complete trajectory above is guaranteed to be* ***valid and safe****.* ▲

Before stating the solution, we need to define the set of corresponding product runs given the robot's past trajectory. As pointed out in Definition 3.8, the resulting run of an infinite or finite words in $\mathcal{A}_\varphi$ may not be *unique* because of the non-determinism of $\tilde{\mathcal{A}}_\varphi$. In other words, given the finite past trajectory τ_{past}, its trace may result in a set of runs in $\tilde{\mathcal{A}}_\varphi$, denoted by $r_{\tau_{\text{past}}}$:

$$r_{\tau_{\text{past}}} = \{r_{\bar{\sigma}} \text{ by Def. 3.8 for } \tilde{\mathcal{A}}_\varphi \,|\, \bar{\sigma} = \texttt{trace}(\bar{\tau}_{\text{past}})\}, \tag{4.20}$$

where $\bar{\tau}_{\text{past}} = \tau_{\text{past}}[1 : (|\tau_{\text{past}}| - 1)]$, i.e., the segment from the first state to the second last state of τ_{past}; $r_{\tau_{\text{past}}}$ is finite because τ_{past} is finite and the number of states in $\tilde{\mathcal{A}}_\varphi$ is finite. Consequently, the finite set of corresponding runs in $\tilde{\mathcal{A}}_r^{t^+}$ is given by

$$R_{\tau_{\text{past}}} = \{R \,|\, R|_\Pi = \tau_{\text{past}},\ R|_Q \in r_{\tau_{\text{past}}}\}, \tag{4.21}$$

where each finite run R is the synchronized product of τ_{past} and any run belonging to $r_{\tau_{\text{past}}}$ by (4.20). Since $R_{\tau_{\text{past}}}$ may contain multiple finite runs in $\tilde{\mathcal{A}}_r^{t^+}$, π_{cur} may correspond to multiple states in $\tilde{\mathcal{A}}_r^{t^+}$, which is denoted by

$$Q'_{\tau_{\text{past}}} = \{\texttt{last}(R) \,|\, R \in R_{\tau_{\text{past}}}\}, \tag{4.22}$$

where $\texttt{last}(R)$ is the last state of R within $R_{\tau_{\text{past}}}$. Clearly, $R_{\text{past}} \in R_{\tau_{\text{past}}}$ and $q'_{\text{cur}} \in Q'_{\tau_{\text{past}}}$; $Q'_{\tau_{\text{past}}}$ is derived as the reachable states in $\tilde{\mathcal{A}}_r^{t^+}$ given τ_{past}.

To solve Problem 4.6, we propose an algorithm that is similar to Algorithm 5 in [59]. Denote by R^{t^-}, R^{t^+}, τ^{t^-}, τ^{t^+} the accepting run and corresponding plan before and after the revision, respectively. Moreover, $R^{t^-}_{\text{pre}}$ and $R^{t^-}_{\text{suf}}$ are the prefix and suffix of R^{t^-}, while $R^{t^+}_{\text{pre}}$ and $R^{t^+}_{\text{suf}}$ are the prefix and suffix of R^{t^+}. Given the invalid or unsafe transition (q'_s, q'_{s+1}) in R^{t^-}, it belongs to either $R^{t^-}_{\text{pre}}$ or $R^{t^-}_{\text{suf}}$. In Algorithm 4.4, it firstly tries to locally finds a "bridging" segment that make up this transition by breadth-first search (as shown in Figure 4.5). $\texttt{tail}(q'_{s+1}, R^{t^-}_{\text{pre}})$ is the segment

Algorithm 4.4 Revise the current plan, `Revise( )`

Input: Ξ, $\aleph$, R^{t^-}, $\tilde{\mathcal{A}}_r^{t^+}$, $Q'_{\tau_{\text{past}}}$
Output: R^{t^+}
forall the $(q'_s, q'_{s+1}) \in (\Xi \cup \aleph)$ **do**
  if $(q'_s, q'_{s+1}) \in R^{t^-}_{\text{pre}}$ **then**
    $bridge$=`DijksTarget`$(\tilde{\mathcal{A}}_r^{t^+}, q'_{s-1},$ `tail`$(q'_s, R^{t^-}_{\text{pre}}))$
    if $bridge \neq \varnothing$ **then**
      $R^{t^-}_{\text{pre}} =$ `head`$(q'_{s-1}, R^{t^-}_{\text{pre}})$+$bridge$+`tail`(`last`$(bridge), R^{t^-}_{\text{pre}})$
    else
      $R^{t^+} =$ `OptRun`$(\tilde{\mathcal{A}}_r^{t^+}, Q'_{\tau_{\text{past}}})$
      return R^{t^+}
  if $(q'_s, q'_{s+1}) \in R^{t^-}_{\text{suf}}$ **then**
    same as the first case, but replace $R^{t^-}_{\text{pre}}$ by $R^{t^-}_{\text{suf}}$
return $R^{t^+} = \langle R^{t^-}_{\text{pre}}, R^{t^-}_{\text{suf}} \rangle$

of $R^{t^-}_{\text{pre}}$ after q'_s, not including q'_s; `head`$(q'_{s-1}, R^{t^-}_{\text{pre}})$ is the segment of $R^{t^-}_{\text{pre}}$ before q'_{s-1}, not including q'_{s-1}; `last`$(bridge)$ is the last state on the path $bridge$. Function `DijksTarget`$(\tilde{\mathcal{A}}_r^{t^+}, q'_S, Q'_T)$ is defined similarly as function `DijksTargets`$(\cdot)$ in Algorithm 3.1, which returns shortest path from the source state $q'_S \in Q'$ to *any* target state belonging to the set $Q'_T \subseteq Q'$. Thus it returns the shortest path once one of the targets is reached. At last, if function `DijksTarget`$(\cdot)$ returns an empty path $bridge$, it means that the accepting state `last`$(R^{t^-}_{\text{pre}})$ is not reachable from q'_{s-1}. Then `OptRun`$(\cdot)$ from Algorithm 3.1 is called to search for the balanced accepting run of $\tilde{\mathcal{A}}_r^{t^+}$, but using $Q'_{\tau_{\text{past}}}$ as the set of initial states (instead of the default Q'_0). Note that in Algorithm 4.4 above, R^{t^-} is revised iteratively and the condition $(q'_s, q'_{s+1}) \in R^{t^-}_{\text{pre}}$ or $R^{t^-}_{\text{suf}}$ is also checked iteratively.

Theorem 4.7. *The new plan τ_{new} can be obtained from Algorithm 4.4 such that τ_{comp} is **valid and safe**, if there exists one.*

Proof. If a new plan τ_{new} exists such that the complete path τ_{comp} is valid and safe, by Lemma 3.2 there exists an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$ whose projection onto Π is τ_{comp}. Given the invalid or unsafe transitions in R^{t^-} (in prefix or suffix), firstly Algorithm 4.4 tries to revise R^{t^-} by looking for the bridging segments. If no such segments exist, it means that the current accepting state is not reachable from q'_{s-1}. Instead it searches for the accepting run that starts from any of the product states in $Q'_{\tau_{\text{past}}}$ and consists of a cycle containing at least one of accepting states in $\tilde{\mathcal{A}}_r^{t^+}$. It means that an accepting run that starts from one of the initial states and *obeys* the

robot's past trajectory exists only if a bridging segment is found for the revision in Line 3 or a new accepting run starting from $Q'_{\tau_{\text{past}}}$ is found in Line 8. ■

The overall structure of the on-line planning scheme is given in Algorithm 4.6 later, where the accepting run R^t is updated by both the revision Algorithm 4.4 and the optimal search Algorithm 3.1. More details about the algorithms can be found in [48]. The next goal region for the hybrid controller also changes accordingly.

4.4 Collaborative knowledge transfer

Now assume that *a team* of autonomous robots coexist within the same workspace and each of them has an individual local task specified as LTL formulas. If the local task of each robot relies on only its local propositions, the team of robots are independent that their plans can be synthesized and executed independently from each other. However since the robots are usually located at different locations within the workspace, about which they have up-to-date knowledge. Those local knowledge, if shared among the team, might benefit each member such that they can make better and more informed plans. In the following, we propose a collaborative knowledge transfer scheme for a multi-robot system with independent local tasks.

Formally, we consider a team of autonomous robots with unique identities $i \in \mathcal{N} = \{1, 2, \cdots, N\}$, which coexist within the common but partially-known workspace $\mathcal{W}$. Robot i's possible motion within $\mathcal{W}$ is abstracted as a weighted FTS:

$$\mathcal{T}_i^t = (\Pi_i, \longrightarrow_i^t, \Pi_{i,0}, AP_i, L_i^t, W_i^t), \tag{4.23}$$

which is defined similarly as (4.15). The superscript $t \geq 0$ indicates that the workspace is partially known and might be updated after the system starts. Each robot $i \in \mathcal{N}$ has a locally-assigned task specification $\varphi_i = \varphi_i^{\text{soft}} \wedge \varphi_i^{\text{hard}}$, where φ_i^{soft} and φ_i^{hard} are the "soft" and "hard" constraints as before from (4.11).

Initial synthesis

At $t = 0$, for each robot $i \in \mathcal{N}$, the initial motion and task plan can be obtained as the safe plan by following the framework proposed in Section 4.2, regarding the initial transition system $\mathcal{T}_i^0$ and φ_i. We assume here the on-the-fly construction is used. Note that the soft specification may not be feasible initially and is relaxed by the balanced accepting run. Denote by $\tilde{\mathcal{A}}_{r,i}^t$ the relaxed product automaton of robot i at time t; R_i^t, τ_i^t as the balanced accepting run and the associated plan.

4.4.1 Knowledge transfer protocol

In Section 4.3, we describe how a single robot could update its transition system through its sensing and communication ability to inquire and retrieve knowledge from external sources. Belonging to the same multi-robot system, the other robots could be the external source. In other words, they could share and transfer their

Algorithm 4.5 Transfer Knowledge to other robots, `TranKnow( )`

Input: $\mathbf{Sense}_i^t$, $\mathbf{Request}_{j,i}^{t_0}$
Output: $\mathbf{Reply}_{i,j}^t$
forall the $(\pi, S, S_\neg) \in \mathbf{Sense}_i^t$ **do**
  forall the $\mathbf{Request}_{j,i}^{t_0}$ received at $t_0 < t$ **do**
    $\mathbf{Request}_{j,i}^t = (j, \varphi_j|_{AP_j}, i)$
    if $j \in \mathcal{N}_i^t$ **then**
      $S' = S \cap (\varphi_j|_{AP_j})$, $S'_\neg = S_\neg \cap (\varphi_j|_{AP_j})$
      if $S' \neq \varnothing$ or $S'_\neg \neq \varnothing$ **then**
        add $(\pi, S', S'_\neg)$ to $\mathbf{Reply}_{i,j}^t$
return $\mathbf{Reply}_{i,j}^t$

knowledge about the workspace collectively in real-time. Now we explain how to design the knowledge transfer protocol and more importantly how to combine it with the real-time planning scheme from Section 4.3.

The communication network represents how the information flows among the robots. Each robot i has a set of neighboring robots, denoted by $\mathcal{N}_i \subseteq \mathcal{N}$. Robot i can send messages directly to any robot belonging to $\mathcal{N}_i$. We take into account two different ways to model the communication network: (1) global communication with a fixed topology; (2) limited communication with a dynamic topology. In the first case, $\mathcal{N}_i$ is pre-defined and fixed after the system starts. In the second case, each robot has a communication range, denoted by $C_i \geq 0$. Robot i can only send messages to robot j if their relative distance is less than C_i, i.e., $|x_j(t) - x_i(t)| \leq C_i$ where $x_j(t), x_i(t) \in \mathbb{R}^n$ are the positions of robots j and i at time t. Then $\mathcal{N}_i^t = \{j \in \mathcal{N} \,|\, |x_j(t) - x_i(t)| \leq C_i\}$ is the time-varying neighboring set of robot i at time t.

Robot i is interested in all the propositions appearing in φ_i, namely $\varphi_i|_{AP_i}$. We propose a *subscriber-publisher* mechanism to reduce the communication load. Whenever robot j communicates with robot $i \in \mathcal{N}_j^t$ for the *first* time at time t, it follows the *subscribing* procedure: robot j sends a request message to robot i:

$$\mathbf{Request}_{j,i}^t = (j, \varphi_j|_{AP_j}, i), \tag{4.24}$$

which informs robot i the set of propositions robot j is interested. Each robot has a subscriber list, containing the request messages it has received. Note that each robot also keeps track of the robots which it has subscribed to, such that it sends a request to any of its neighboring robots only once.

The sensing update of robot i is denoted by $\mathbf{Sense}_i^t$, the structure of which is given in (4.16). Then the *publishing* phase of each robot follows an event-driven approach: whenever $(\pi, S, S_\neg) \in \mathbf{Sense}_i^t$ is obtained, it checks its subscriber list whether the content might be of interest to any of the subscribers regarding some propositions. If it is of interest to robot j regarding some propositions in in $\varphi_j|_{AP_j}$,

then robot i checks if $j \in \mathcal{N}_i^t$. If so, it publishes a reply message to robot j that

$$\mathbf{Reply}_{i,j}^t = \{(\pi, S', S'_\neg)\}, \tag{4.25}$$

where $S' = S \cap (\varphi_j|_{AP_j})$ and $S'_\neg = S_\neg \cap (\varphi_j|_{AP_j})$; Note that since S' and $S'_\neg$ only contain propositions that are relevant to robot's task φ, every reply message should contain useful knowledge. The above procedure is summarized in Algorithm 4.5. Note that through this communication mechanism, any request only needs to be sent once and every reply message contains useful knowledge. This subscriber-publisher scheme can be easily implemented by multicast or unicast wireless protocols.

4.4.2 On-line plan verification and adaptation

Upon receiving $\mathbf{Sense}_i^t$ and $\mathbf{Reply}_{j,i}^t$ from $j \in \mathcal{N}_i^t$, robot i could update its transition system $\mathcal{T}_i$ accordingly. Regarding its current motion and task plan τ_i^t, the validity and safety of τ_i^t needs to be verified as before. Moreover, in case τ_i^t is falsified, i.e., either invalid or unsafe, Algorithm 4.4 is called to revise τ_i^t such that it becomes valid and safe, where $\widehat{\Pi}_i^t$, $\aleph_i^t$, Ξ_i^t stand for the set of regions in $\mathcal{T}_i^t$ with modified labeling functions, the set of unsafe transitions and invalid transitions in R_i^t.

Algorithm 4.4 provides a way to locally revise the invalid or unsafe plan, which guarantees validity and safety by Theorem 4.4. However it does not maintain the cost optimality of the safe accepting run compared with Algorithm 3.1. The general problem of computing and maintaining the shortest path in a graph where the edges are inserted or deleted and edge weights are increased or decreased is referred to as the *dynamic shortest path problem* (DSPP) in [17] and [115]. As pointed out in both papers, it is inefficient to re-compute the shortest path "from scratch" using the well-known static solution like Dijkstra algorithm each time the graph changes.

Thus we propose an event-based criterion [72] to ensure the optimality check. Denote by Υ_i^t the accumulated number of number of changes in $\mathcal{T}_i^t$ at time t; $\Upsilon_i^{t^+} = \Upsilon_i^{t^-} + |\widehat{\Pi}_i^t|$ and $\Upsilon_i^0 = 0$. Denote by T_i the last time instant when Algorithm 3.1 is called. Let the thresholds $N_i^{\text{call}}, T_i^{\text{call}} \geq 0$ be chosen freely by robot $i \in \mathcal{N}$. Then whenever one of the conditions holds: (1) $\Upsilon_i^t \geq N_i^{\text{call}}$; (2) $t - T_i \geq T_i^{\text{call}}$, Algorithm 3.1 is called with respect the latest $\mathcal{T}_i^t$ to derive the safe plan, but using $Q'_{\tau_{\text{past}}}$ as the initial states. Afterwards, $\Upsilon_i^{t^+}$ is reset to zero and T_i to the current time.

4.4.3 Overall structure

The overall architecture is summarized in Algorithm 4.6, where the detail description of each function can be found in [49]. When the system starts, each robot synthesizes its own initial motion and task plan. It sends requests to neighboring robots. Then it checks if it receives any reply, sensing or request messages, based on which it replies to its subscribers, and updates its transition system. At last, it decides whether the revising algorithm or the optimal synthesis algorithm should be called by the triggering condition. It is worth mentioning that the next goal region $\pi_{i,\text{next}}$ changes

Algorithm 4.6 Cooperative on-line planning for each robot $i \in \mathcal{N}$

Input: $\mathcal{T}_i^0$, $\tilde{\mathcal{A}}_{\varphi_i}$, $\tilde{\mathcal{A}}_{r,i}^0$, $x_i(t)$
Output: R_i^t, τ_i^t, Υ_i^t, T_i^t
$R_i^0 = \texttt{OptRun}(\tilde{\mathcal{A}}_{r,i}^0)$
$q'_{i,\text{cur}} = q'_{i,\text{next}} = R^0_{i,[\text{pre},1]}$, $\pi_{i,\text{next}} = q'_{i,\text{next}}|_{\Pi_i}$, $R_{i,\text{past}} = [\ ]$, $\tau_{i,\text{past}} = [\ ]$
while `True` **do**
  send $\mathbf{Request}_{i,g}^t$
  check $\mathbf{Reply}_{h,i}^t$, $\mathbf{Request}_{j,i}^{t_0}$ and $\mathbf{Sense}_i^t$
  $\mathbf{Reply}_{i,j}^t = \texttt{TranKnow}(\mathbf{Sense}_i^t, \mathbf{Request}_{j,i}^{t_0})$
  send $\mathbf{Reply}_{i,j}^t$
  $(\mathcal{T}_i^{t^+}, \widetilde{\Pi}_i^t, \widehat{\Pi}_i^t) = \texttt{UpdaT}(\mathcal{T}_i^t, \mathbf{Sense}_i^t, \mathbf{Reply}_{h,i}^t)$
  $\tilde{\mathcal{A}}_{r,i}^{t^+} = \texttt{AdjProd}(\mathcal{T}_i^{t^+}, \widehat{\Pi}_i^t, \tilde{\mathcal{A}}_{\varphi_i})$
  $(\aleph_i^t,\ \Xi_i^t) = \texttt{ValidRun}(\tilde{\mathcal{A}}_{r,i}^{t^+}, R_i^t, \widetilde{\Pi}_i^t, E_\neg)$
  $\Upsilon_i^t = \Upsilon_i^t + |\widehat{\Pi}_i^t|$
  $Q'_{i,\tau_{\text{past}}} = \texttt{CorProd}(\tilde{\mathcal{A}}_{r,i}^{t^+}, \tau_{i,\text{past}})$
  if $\Upsilon_i^t \geq N_i^{\text{call}}$ *or* $t - T_i^t \geq T_i^{\text{call}}$ **then**
    $R_i^{t^+} = \texttt{OptRun}(\tilde{\mathcal{A}}_{r,i}^{t^+}, Q'_{i,\tau_{\text{past}}})$
    $\Upsilon_i^t = 0$, $T_i^t = t$, $q'_{i,\text{next}} = R^{t^+}_{i,[\text{pre},2]}$
  else if $\aleph_i^t \neq \varnothing$ *or* $\Xi_i^t \neq \varnothing$ **then**
    $R_i^{t^+} = \texttt{Revise}(\tilde{\mathcal{A}}_{r,i}^{t^+}, R_i^t, \aleph_i^t, \Xi_i^t, Q'_{i,\tau_{\text{past}}})$
    $q'_{i,\text{next}} = R^{t^+}_{i,[seg,k]}$
  if $x(t) \in q'_{i,\text{next}}|_{\Pi_i}$ confirmed **then**
    $q'_{i,\text{cur}} = q'_{i,\text{next}}$
    $\tau_{i,\text{past}} = \tau_{i,\text{past}} + \pi_{i,\text{next}}$, $R_{i,\text{past}} = R_{i,\text{past}} + q'_{i,\text{next}}$
    $q'_{i,\text{next}} = R^{t^+}_{i,[seg,k]} = \texttt{NextGoal}(q'_{i,\text{cur}}, R_i^{t^+})$
  $\pi_{i,\text{cur}} = q'_{i,\text{cur}}|_{\Pi_i}$, $\pi_{i,\text{next}} = q'_{i,\text{next}}|_{\Pi_i}$
  $u_i = \mathcal{U}(x_i(t), \pi_{i,\text{cur}}, \pi_{i,\text{next}})$

automatically whenever: (i) the accepting run R_i^t is updated by either the revision Algorithm 4.4 and the optimal search Algorithm 3.1; (ii) the current goal region is confirmed to be reached.

Theorem 4.8. *For each robot i at any time $t \geq 0$, its plan τ_i^t derived by Algorithm 4.6 is always **valid** and **safe** for $\mathcal{T}_i^t$ and φ_i. Moreover, for any $t' \geq 0$, there exists time $t \in [t', t' + T_i^{call}]$ such that τ_i^t corresponds to the **safe** accepting run.*

Proof. The first part follows from Theorems 4.6 and 4.7. Moreover τ_i^t is the safe plan for $\tilde{\mathcal{A}}_{r,i}^{t^+}$ given $Q'_{i,\tau_{\text{past}}}$ whenever Algorithm 3.1 is called. Due to the triggering

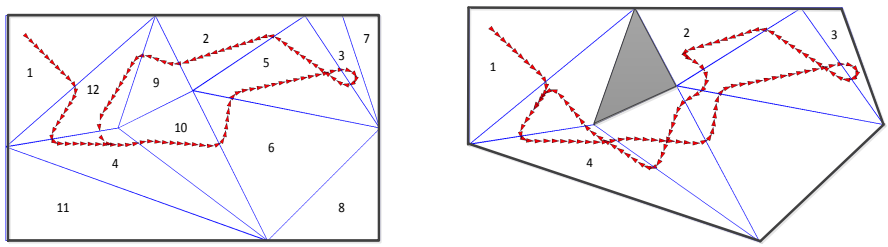


Figure 4.6: Left: Initial workspace and the the preliminary motion and task plan. Right: actual workspace and the final trajectory.

condition, it is called at least once within any time period of length T_i^{call}. ■

4.5 Case study

In this part, we present two different case studies for planning under partially-known workspaces: one for controlling a single unicycle robot under the task of surveilance; another for a team of autonomous robots under complex local tasks.

Single-robot system

Consider a unicycle robot that satisfies: $\dot{x_0} = v\cos\theta$, $\dot{y_0} = v\sin\theta$, $\dot{\theta} = w$, where $\mathbf{p}_0 = (x_0, y_0)^T \in \mathbb{R}^2$ is the center of mass, $\theta \in [0, 2\pi]$ is the orientation, and $v, w \in \mathbb{R}$ are the transition and rotation velocities.

Workspace model

The workspace we consider is shown in Figure 4.6, which consists of 12 polygonal regions. The continuous controller that drives the robot from an region to any geometrically adjacent region is based on [103], which is built by constructing vector fields over each cell for each face. The controller design is omitted here for brevity. There are three regions of interest and one regions is occupied by obstacles. The surveillance task is given by "visit region 2, 3, 4 infinitely often and avoid all possible obstacles". The LTL formula is given by "$\varphi = (\square\diamond a_1) \wedge (\square\diamond a_2) \wedge (\square\diamond a_3) \wedge (\square\neg a_4)$" and its associated NBA is shown in Figure 3.3.

Simulation results

The preliminary workspace is initialized as obstacles free and the associated $\mathcal{T}_c^0$ is constructed by Definition 3.3. The actual workspace is shown in Figure 4.6, where region 9 is occupied by obstacles and there are walls between some regions. The robot is capable of perceiving obstacles within a region and walls between adjacent regions. A preliminary motion plan is generated by Algorithms 3.1 and 3.2 (arrowed red line

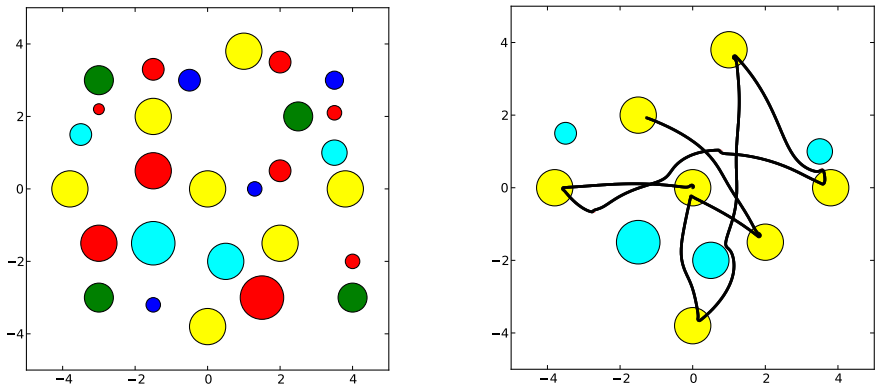


Figure 4.7: Left: the actual workspace as described in Section 4.5. Right: the final motion plan for aerial vehicles, which corresponds to the final optimal plan in Figure 4.9. It satisfies both $\varphi^{\text{hard}}_{\text{Ar\_i}}$ and $\varphi^{\text{soft}}_{\text{Ar\_i}}$, since all base stations (in yellow) are surveiled and non-fly areas (in cyan) are avoided.

in Figure 4.6), but it is not valid for the actual workspace as it intersects with the obstacle region 9. The robot moves according to the motion plan and reaches region 12, where it obtains the following information: $E_{\neg} = \{(\pi_4, \pi_{11}), (\pi_{11}, \pi_4)\}$ and $(\pi_9, \{a_4\}, \varnothing)$, namely region 9 satisfies a_4. The updated motion plan is illustrated by the arrowed red lines in Figure 4.6. Then the robot follows this updated motion plan. At region 6 and 3, it obtains the information: $E_{\neg} = \{(\pi_6, \pi_8), (\pi_8, \pi_6)\}$ and $E_{\neg} = \{(\pi_7, \pi_3), (\pi_3, \pi_7)\}$, respectively. In both cases 32 transitions are removed. But the motion and task plan remains valid because its corresponding accepting run remains valid. The final trajectory is shown in Figure 4.6.

Multi-robot system

In this case study, we consider a team of 15 autonomous robots: five of them are aerial vehicles that surveil over base stations; the rest are ground vehicles that collect food and water resources to supply the base stations.

Workspace model and robot description

As shown Figure 4.7, the workspace we consider is a $10 \times 10m^2$ square which is approximated by $\mathcal{B}_5([0, 0])$, where $[0, 0]$ is the origin and 5 is the radius. There are 7 base stations with size $1 \times 1\text{m}^2$ (in yellow, denoted by "`b1`",⋯,"`b7`"). Besides, there are numerous no-fly zone (in cyan, denoted by "`nfly`") and sphere obstacles (in red, denoted by "`obs`") at various locations with different sizes. Food (in green, denoted by "`food`") and water (in blue, denoted by "`water`") resources of various size are scattered in the free space.

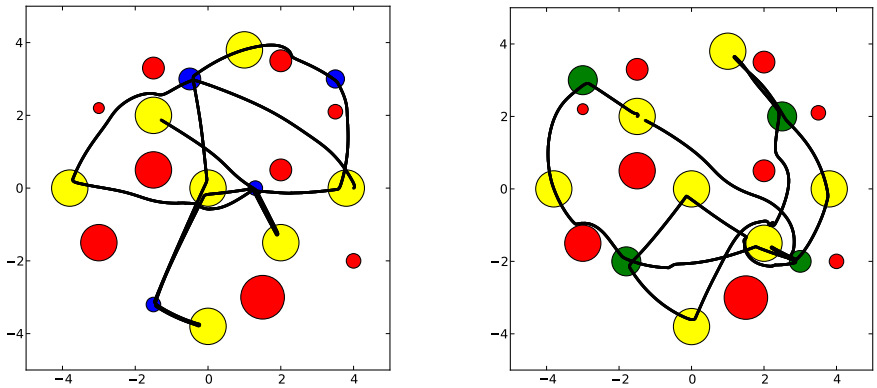


Figure 4.8: The final motion plan for robots Gw_i (left) and for robots Gf_i (right). It is shown that water (in blue) or food (in green) are fetched before any base station (in yellow) and all base stations are supplied infinitely often, while obstacle (in red) avoidance is always ensured.

Five aerial vehicles (denoted by Ar_1,⋯, Ar_5) start randomly from one of the base stations. For aerial vehicles, "b1",⋯, "b7", "nfly", "water", "food" are their known propositions, which does not include "obs". It means that aerial vehicles cannot detect obstacles on the ground. Initially, they know the location of some base stations and some no-fly zones, but not the water and food resources. They have an average speed $0.1m/s$ and a sensing radius of $2m$ in the x-y coordinate. They have the hard specification "repetitively visit at least one of the base stations, while avoiding all no-fly zones", and soft specification "visit all base stations infinitely often". In LTL formulas, the specifications are given as $\varphi^{\text{hard}}_{\text{Ar\_i}} = (\square\neg\,\texttt{nfly}) \wedge (\square\diamond(\varphi_{\texttt{one}}))$, and $\varphi^{\text{soft}}_{\text{Ar\_i}} = (\square(\diamond\texttt{b1}\wedge\diamond\texttt{b2}\wedge\cdots\wedge\diamond\texttt{b7}))$, where $\varphi_{\texttt{one}} \triangleq \texttt{b1}\vee\texttt{b2}\vee\cdots\vee\texttt{b7}$ and $i = 1,\cdots,5$. The NBA associated with $\varphi^{\text{hard}}_{\text{Ar\_i}}$ has 2 states and 4 edges by [41], while the one with $\varphi^{\text{soft}}_{\text{Ar\_i}}$ has 8 states and 43 edges.

The other ten robots are ground vehicles: five of them (denoted by Gf_1,⋯, Gf_5) collect food and the rest (denoted by Gw_1, ⋯, Gw_5) for water, to supply the base stations. For ground vehicles, "b1", ⋯, "b7", "obs", "water", "food" are known propositions, which does not include "nfly". Thus ground vehicles cannot recognize "nfly" zones. Initially, they start randomly from one base station and only knows the location of one water (or food) resource, but not the obstacles and other base stations. They have an average speed $0.05m/s$ and a sensing radius of 1m in the x-y coordinate. For ground vehicles, the hard specification is "avoid all obstacles and repetitively collect water (or food) resources to at least one base station" and the soft specification is "supply all base stations infinitely often". In LTL formulas, the hard and soft tasks for Gw_i are given by $\varphi^{\text{hard}}_{\text{Gw\_i}} = (\square\diamond\neg\texttt{obs}) \wedge \varphi_{\texttt{order}}$, $\varphi^{\text{soft}}_{\text{Gw\_i}} = (\square(\diamond\texttt{b1}\wedge\diamond\texttt{b2}\wedge\cdots\wedge\diamond\texttt{b7}))$, where $\varphi_{\texttt{order}} \triangleq (\square\diamond\ \texttt{water}) \wedge (\square(\texttt{water} \Rightarrow \bigcirc(\neg\texttt{water}\,\mathsf{U}\,(\varphi_{\texttt{one}}))) \wedge (\square((\varphi_{\texttt{one}}) \Rightarrow \bigcirc(\neg(\varphi_{\texttt{one}})\,\mathsf{U}\,\texttt{water}))$, which says that water

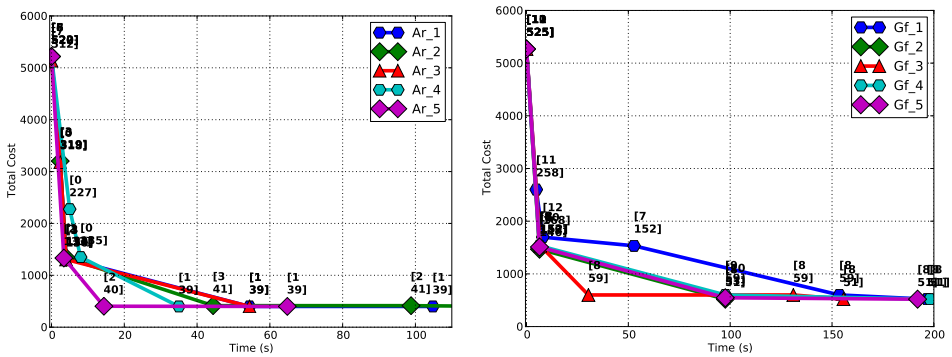


Figure 4.9: The evolution of the plan total cost of robots Ar_i (left) and Gf_i (right). For Ar_1, the initial plan has a total cost 5220 ($[5, 521]$), when it only knows two base stations. Its final plan has a total cost 391 ($[1, 39]$), as shown in Figure 4.7. For Gf_1, the initial plan has a total cost 5268 ($[12, 526]$), when it only knows the location of one base station and one food resource. Its final plan has the total cost 610 ($[3, 60]$), as shown in Figure 4.8.

must be fetched and supplied to at least one base station infinitely often. $\varphi^{\text{soft}}_{\text{Gw\_i}}$ is the same as for Ar_i, requiring that all base stations be supplied infinitely often. The NBA associated with $\varphi^{\text{hard}}_{\text{Gw\_i}}$ have 10 states and 30 edges by [41], while the one for $\varphi^{\text{soft}}_{\text{Gw\_i}}$ has 8 states and 43 edges. The soft and hard specifications $\varphi^{\text{hard}}_{\text{Gf\_i}}$ and $\varphi^{\text{soft}}_{\text{Gf\_i}}$ for Gf_i can be defined in a similar way by replacing proposition "`water`" with "`food`".

Clearly, for each robot the soft specification is impossible to fulfil initially as they have no complete knowledge about the location of all base stations. However, since "`b1`", "`b2`",⋯, "`b7`" belong to all vehicles, meaning that any relevant information can be shared within the group. Moreover, since the propositions "`water`" and "`food`" also belong to aerial vehicles, they could help the ground vehicles to discover relevant knowledge within a shorter time. Regarding the communication network, a dynamic topology where each robot has a communication radius (set to $5m$ for all robots).

Simulation results

Initially each robot has very limited knowledge, considering that they only know the location of one base station and none of the obstacle regions. We choose α to be 1000 and γ to be 10 as the task specifications focus on repetitive tasks. We set N_i^{call} and T_i^{call} to be $(3, 60s)$ for aerial vehicles and $(3, 100s)$ for ground vehicles. The system was simulated for $600s$ and it took $200s$ for the workspace to be fully discovered by all robots. Figures 4.7 and 4.8 show the trajectories corresponding to the suffix part of the optimal run, for the three groups Ar_i, Gf_i, Gw_i. It can be seen that both soft and hard specifications are fulfilled by all robots.

Figure 4.9 and 4.10 shows how the optimal plans of Ar_i, Gw_i and Gf_i evolve

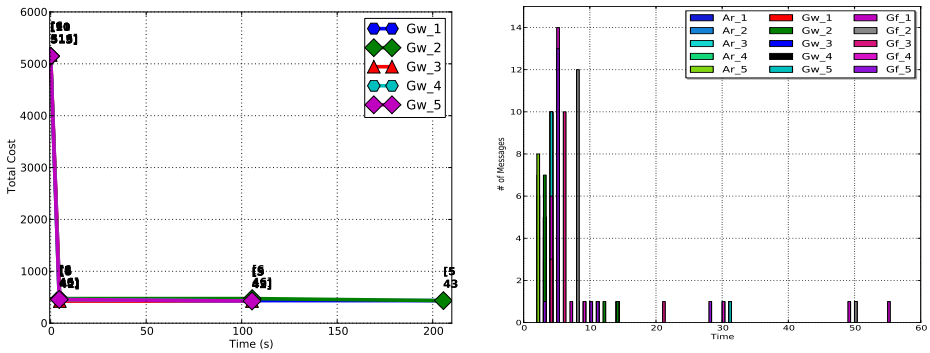


Figure 4.10: Left: the evolution of the plan total cost of robots Gw_i. For robot Gw_1, the initial plan has a total cost around 5147 ($[11, 513]$), when it only knows the location of one base station and one water resource. Its final plan has a total cost 456 ($[6, 45]$), as shown in Figure 4.8. Right: the number of messages (including both request and reply messages) received by each robot under the dynamic communication topology.

with time. In particular, under the proposed scheme, the total cost of the optimal plan for each robot decreases gradually until its workspace model is complete. It can be seen that the planner incorporates the knowledge update into the optimal plan, such that the soft task specification is satisfied more. Figure 4.10 illustrates the number of messages received by each robot under the dynamic communication topology. Due to the subscriber-publisher scheme, the number of messages being exchanged among the robots decreases over time.

4.6 Summary

In this chapter, we discussed about the motion and task planning problem for both single and multiple robot systems within a partially-known workspace. In particular, we first presented the method to synthesize the balanced plan for the potentially infeasible task with hard and soft constraints. Then we proposed an on-line scheme for plan adaptation and revision to guarantee the safety and feasibility of the current plan at all time. At last, we extended this framework to multi-robot systems with independent individual tasks by introducing the knowledge transfer protocol.

Chapter 5

Dependent local tasks with collaborative actions

Robots are deployed to move around within the workspace and more importantly to perform various actions to interact with it. Thus in this chapter, we first introduce a new method to construct the model of both robot motion and actions. Based on this model, we can specify a local temporal task as the desired robot motion and actions. Then a planning scheme for both robot motion and actions is proposed as an extension to the nominal solution presented in Chapter 3. Furthermore, inter-robot collaborations are essential for many multi-robot applications to improve system capability, efficiency and robustness. Thus we start from defining a dependency relation among the robots using collaborative and assisting actions. Then a distributed and on-line coordination scheme is proposed where each robot sends requests to its neighbors for collaborative tasks, confirms others' requests and adapts its local plan to fulfill the confirmed collaboration. All decisions are made locally by each robot based on local computation and communication between neighboring robots. This scheme is scalable and resilient to robot failures as the dependency relation is formed and removed dynamically according to the plan execution status and robot capabilities, instead of pre-assigned robot identities. Numerical simulations are provided in the end to validate the proposed scheme.

5.1 Motion and action planning

The task specifications considered so far only involve a sequence of regions to visit. However, for practical problems it is often necessary to perform various actions at different regions. In other words, the purpose of "going somewhere" is to "do something". The resulting plan is clearly a combination of transitions among different places and performing sequential actions. It would be inadequate to carry out the motion planning and action planning independently since the motion plan and action plan are closely related, i.e., "where to go" is motivated by "what to do there" and "what to do now" depends on "where it has been". Another observation

is that some actions can only be performed when certain conditions are fulfilled and as a result certain state variables might be changed. Moreover, we propose to separate the domain-specific knowledge [146] such as the workspace model and the robot's mobility in the workspace, from the domain-independent knowledge such as the action map based on the actions the robot is capable of. One advantage is the increased modularity that our framework is adaptable whenever the workspace is modified or the task specification is changed.

5.1.1 Complete robot model

The complete robot model is derived by composing the abstraction of robot mobility and the model of robot actions.

Abstraction of robot motion

Firstly, in order to distinguish the FTSs for mobility and action, we replace the FTS $\mathcal{T}_c$ in Definition 3.3 with $\mathcal{M}$ as the abstraction of robot's *mobility*:

$$\mathcal{M} = (\Pi_{\mathcal{M}}, \sigma_{\mathcal{M}}, \longrightarrow_{\mathcal{M}}, \Pi_{\mathcal{M},0}, \Psi_{\mathcal{M}}, L_{\mathcal{M}}, W_{\mathcal{M}}), \tag{5.1}$$

of which the notations are similar to $\mathcal{T}_c$; $\sigma_{\mathcal{M}}$ stands for the control strategy (3.3) symbolically; $\Psi_{\mathcal{M}} = \Psi_r \cup \Psi_p$ where $\Psi_r = \{\Psi_{r,i}\}$, $i = 1, \cdots, N$, is the set of propositions as region names; $\Psi_p = \{\Psi_{p,1}, \cdots, \Psi_{p,I}\}$ is the finite set of propositions as interested properties, some of which are relevant to robot's actions discussed later. For instance, "this room has product A" is a property relevant to the action "pickup A".

Model of robot actions

Action description language [42] provides an intuitive and powerful way for describing the preconditions and effects of different actions. Classic planning formalisms, like STRIPS [37, 119] provide an intuitive way to describe high-level actions the robot is capable of. Given a set of system states and actions, each action is described by specifying its precondition and effect on the states. Assume that the robot is capable of performing K different actions $\{\sigma_{\mathcal{B},1}, \cdots, \sigma_{\mathcal{B},K}\}$, implementable by the corresponding low-level controllers $\{\mathcal{K}_k\}$, $k = 1, \cdots, K$. Denote by $\Sigma_{\mathcal{B}} = \{\sigma_{\mathcal{B},0}, \cdots, \sigma_{\mathcal{B},K}\}$, where $\sigma_{\mathcal{B},0} \triangleq \texttt{None}$ indicates that none of these K actions is performed. Moreover, we introduce another two sets of propositions: (i) $\Psi_s = \{\Psi_{s,j}\}$, represents the internal states of the robot, $j = 1, 2 \cdots, J$, e.g., "the robot has product A"; (ii) $\Psi_b = \{\Psi_{b,k}\}$, where $\Psi_{b,k} = \top$ if and only if action k is performed, $k = 0, 1, \cdots, K$. We assume that any two actions cannot be concurrent, i.e., at most one element of Ψ_b can be true. The subscripts of Ψ_s and Ψ_b stand for the "state" and "behavior". Then each action in $\Sigma_{\mathcal{B}}$ is described by its precondition and effect functions:

The **precondition function**:

$$\texttt{Cond} : \Sigma_{\mathcal{B}} \times 2^{\Psi_p} \times 2^{\Psi_s} \longrightarrow \mathsf{True/False}, \tag{5.2}$$

takes one action in $\Sigma_{\mathcal{B}}$, subsets of Ψ_p and Ψ_s as inputs and returns a boolean value. Namely in order to perform that action, the conditions on the workspace properties and the robot's internal states have to be fulfilled. For instance, the action "pickup A" can only be performed if "the room has product A". While some actions like "take pictures" might be performed without such constraints and the condition is $\top$. Note the condition for $\sigma_{\mathcal{B},0}$ is $\top$ by definition.

The **effect function**:

$$\texttt{Eff} : \Sigma_{\mathcal{B}} \times (2^{\Psi_s} \times \Psi_b) \longrightarrow (2^{\Psi_s} \times \Psi_b), \tag{5.3}$$

represents the effect of the actions. As a result of performing action $\sigma_{\mathcal{B},k}$, the robot's internal states Ψ_s might be changed and Ψ_b is changed to indicate which action is performed. More specifically, (i) $\texttt{Eff}(\sigma_{\mathcal{B},0}, w_s, \Psi_{b,k}) = (w_s, \Psi_{b,0})$, where $w_s \subseteq 2^{\Psi_s}$ and $\forall \Psi_{b,k} \in \Psi_b$. Performing $\sigma_{\mathcal{B},0}$ does not change the robot's internal state and all elements in Ψ_b except $\Psi_{b,0}$ are set to $\bot$; (ii) $\texttt{Eff}(\sigma_{\mathcal{B},k}, w_s, \Psi_{b,l}) = (w'_s, \Psi_{b,k})$, where $w_s, w'_s \subseteq 2^{\Psi_s}$ and $\Psi_{b,l}, \Psi_{b,k} \in \Psi_b$, is the effect function of $\sigma_{\mathcal{B},k}$ for $k \neq 0$. For example, once the action "pickup A" is performed, the propositions "the robot has A" and "'pickup A' is performed" become true. Note that the effect functions can not modify the properties of the workspace.

Given Ψ_p, Ψ_s, Ψ_b and $\Sigma_{\mathcal{B}}$, $\texttt{Cond}$, $\texttt{Eff}$, the *action map* is defined as a tuple

$$\mathcal{B} = (\Pi_{\mathcal{B}}, \Sigma_{\mathcal{B}}, \Psi_p, \hookrightarrow_{\mathcal{B}}, \Pi_{\mathcal{B},0}, \Psi_{\mathcal{B}}, L_{\mathcal{B}}, W_{\mathcal{B}}), \tag{5.4}$$

where (i) $\Pi_{\mathcal{B}} \subseteq 2^{\Psi_s} \times \Psi_b$ is set of all assignments of Ψ_s and Ψ_b; (ii) Ψ_p serves as the input propositions, and 2^{Ψ_p} is the finite set of possible input assignments; (iii) the conditional transition relation $\hookrightarrow_{\mathcal{B}}$ is defined by $\pi_{\mathcal{B}} \times \alpha_{\mathcal{B}} \times 2^{\Psi_p} \times \pi'_{\mathcal{B}} \subseteq \hookrightarrow_{\mathcal{B}}$ if the following conditions hold: (1) $\alpha_{\mathcal{B}} \in \Sigma_{\mathcal{B}}$, $\pi_{\mathcal{B}}, \pi'_{\mathcal{B}} \in \Pi_{\mathcal{B}}$; (2) $\texttt{Cond}(\alpha_{\mathcal{B}}, 2^{\Psi_p}, \pi_{\mathcal{B}}) = \top$; (3) $\pi'_{\mathcal{B}} \in \texttt{Eff}(\alpha_{\mathcal{B}}, \pi_{\mathcal{B}})$; (iv) $\Pi_{\mathcal{B},0} \subseteq 2^{\Psi_s} \times \Psi_{b,0}$ is the initial state; (v) $\Psi_{\mathcal{B}} = \Psi_s \cup \Psi_b$ is the set of atomic propositions; (vi) $L_{\mathcal{B}}(\pi_{\mathcal{B}}) = \{\pi_{\mathcal{B}}\}$, i.e., the labeling function is the state itself; (vii) $W_{\mathcal{B}} : \hookrightarrow_{\mathcal{B}} \to \mathbb{R}^+$ is the weight associated with each transition and $W_{\mathcal{B}}(\pi_{\mathcal{B}}, \alpha_{\mathcal{B}}, 2^{\Psi_p}, \pi'_{\mathcal{B}})$ is estimated by the cost of action $\alpha_{\mathcal{B}}$.

***Remark* 5.1.** The set of states $\Pi_{\mathcal{B}}$ is defined as $2^{\Psi_s} \times \Psi_b$ instead of $2^{\Psi_s} \times 2^{\Psi_b}$ because only one element in Ψ_b can be true. ▲

Ψ_p can be viewed as external inputs [11] to the action map, i.e., within different regions the transition relations might be different due to their different properties. Moreover, $\mathcal{B}$ is nondeterministic in the sense that at each state $\pi_{\mathcal{B}}$ any action whose associated condition function is evaluated to be true, can be performed. It is worth mentioning that the action map is constructed independently of the structure of the workspace where the robot will be deployed. Furthermore, given an instance of the workspace property Ψ_p, the action map $\mathcal{B}$ is equivalent to a wFTS as all conditional transition relations can be verified or falsified.

Complete functionality model

As mentioned earlier, the mobility abstraction $\mathcal{M}$ from (5.1) and the action map $\mathcal{B}$ from (5.4) are adequate for the controller synthesis within certain problem domain.

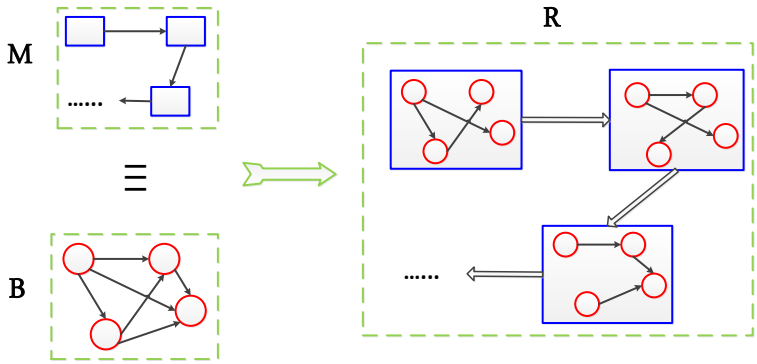


Figure 5.1: The action map $\mathcal{B}$ is composed with each region of $\mathcal{M}$, giving a complete description $\mathcal{R}$ of robot's functionalities.

However, in order to consider richer and more complex tasks involving both robot motion and actions, we need a *complete model* of robot's functionalities that combines these two parts. We propose the following way to compose $\mathcal{M}$ and $\mathcal{B}$ as a FTS:

$$\mathcal{R} = (\Pi_{\mathcal{R}}, \Sigma_{\mathcal{R}}, \longrightarrow_{\mathcal{R}}, \Pi_{\mathcal{R},0}, \Psi_{\mathcal{R}}, L_{\mathcal{R}}, W_{\mathcal{R}}), \tag{5.5}$$

where (i) $\Pi_{\mathcal{R}} = \Pi_{\mathcal{M}} \times \Pi_{\mathcal{B}}$ is the set of states; (ii) $\Sigma_{\mathcal{R}} = \sigma_{\mathcal{M}} \cup \Sigma_{\mathcal{B}}$ is the set of actions; (iii) $\longrightarrow_{\mathcal{R}} \subseteq \Pi_{\mathcal{R}} \times \Sigma_{\mathcal{R}} \times \Pi_{\mathcal{R}}$ is the transition relation, defined by the following rules: (1) $\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle \xrightarrow{\sigma_{\mathcal{M}}}_{\mathcal{R}} \langle \pi'_{\mathcal{M}}, \pi'_{\mathcal{B}} \rangle$ if $\pi_{\mathcal{M}} \xrightarrow{\sigma_{\mathcal{M}}}_{\mathcal{M}} \pi'_{\mathcal{M}}$ and $\pi_{\mathcal{B}} \xrightarrow{\sigma_{\mathcal{B},0}}_{\mathcal{B}} \pi'_{\mathcal{B}}$; (2) $\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle \xrightarrow{\alpha_{\mathcal{B}}}_{\mathcal{R}} \langle \pi_{\mathcal{M}}, \pi'_{\mathcal{B}} \rangle$ if $\pi_{\mathcal{B}} \times \alpha_{\mathcal{B}} \times L_{\mathcal{M}}(\pi_{\mathcal{M}}) \times \pi'_{\mathcal{B}} \subset \hookrightarrow_{\mathcal{B}}$, where $\alpha_{\mathcal{B}} \in \Sigma_{\mathcal{B}}$. (iv) $\Pi_{\mathcal{R},0} = \Pi_{\mathcal{M},0} \times \Pi_{\mathcal{B},0}$ contains the robot's initial region and initial internal state; (v) $\Psi_{\mathcal{R}} = \Psi_{\mathcal{M}} \cup \Psi_{\mathcal{B}}$ is the complete set of atomic propositions including Ψ_r, Ψ_p, Ψ_s and Ψ_b; (vi) $L_{\mathcal{R}} : \Pi_{\mathcal{R}} \to 2^{\Psi_{\mathcal{R}}}$ is the labeling function, $L_{\mathcal{R}}(\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle) = L_{\mathcal{M}}(\pi_{\mathcal{M}}) \cup L_{\mathcal{B}}(\pi_{\mathcal{B}})$; (vii) $W_{\mathcal{R}} : \longrightarrow_{\mathcal{R}} \to \mathbb{R}^+$, is the weight function on each transition, defined as: (1) $W_{\mathcal{R}}(\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle, \sigma_{\mathcal{M}}, \langle \pi'_{\mathcal{M}}, \pi'_{\mathcal{B}} \rangle) = W_{\mathcal{M}}(\pi_{\mathcal{M}}, \sigma_{\mathcal{M}}, \pi'_{\mathcal{M}})$; (2) $W_{\mathcal{R}}(\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle, \alpha_{\mathcal{R}}, \langle \pi_{\mathcal{M}}, \pi'_{\mathcal{B}} \rangle) = W_{\mathcal{B}}(\pi_{\mathcal{B}}, \alpha_{\mathcal{R}}, \pi'_{\mathcal{B}})$, if $\alpha_{\mathcal{R}} \in \Sigma_{\mathcal{B}}$.

***Remark* 5.2.** Note that $\hookrightarrow_{\mathcal{B}}$, $\sigma_{\mathcal{B},0}$ is released automatically whenever the controller (3.3) is activated. Because whenever the robot moves to a new region, it indicates that no actions within $\sigma_{\mathcal{B},k}$ are performed as we assume non-concurrent actions. ▲

Figure 5.1 illustrates the idea behind the process of parallel composition defined above. Blue squares represent the states of $\mathcal{M}$ and red cycles encode the states of $\mathcal{B}$. Loosely speaking, when composing them into $\mathcal{R}$, N copies of $\mathcal{B}$ are first made, corresponding to the N regions within the workspace. At the same time, the conditional transition relations in these copies are verified or falsified by verifying the conditions on the properties of each region.

5.1.2 Local task for motion and actions

The composed system $\mathcal{R}$ is a wFTS over the set of atomic propositions $\Psi_{\mathcal{R}}$. Recall that $\Psi_{\mathcal{R}} = \Psi_r \cup \Psi_p \cup \Psi_s \cup \Psi_b$. Among them, Ψ_r, Ψ_p are commonly seen in related work [15, 85, 132], but Ψ_s, Ψ_b allow us to express richer requirements on the robot's internal states and actions directly, as for example where these actions are desired and the preferred sequence. For instance, the task "eventually always drop A at region 1" can be expressed as $\varphi = \square \diamond (\Psi_{r,1} \wedge \Psi_{b,2})$, where $\Psi_{r,1}$="{the robot is in region 1"} and $\Psi_{b,2}$={"drop A" is performed}. Notice that we do not even need to specify where the robot should perform "pickup A".

5.1.3 Hybrid control strategy

We now state the problem we consider in this section:

Problem 5.1. *Given the full robot functionality $\mathcal{R}$ and the task φ, find its motion and action plan that* ***minimizes*** *the cost defined by* (3.9) *and construct the hybrid control strategy that executes this plan.* ▲

Since $\mathcal{R}$ is a standard finite transition system like $\mathcal{T}_c$ from Definition 3.3, the framework proposed in Chapter 3 can be applied directly to find the infinite optimal motion and action sequence that has a finite representation and minimizes the cost by 3.9. Namely, denote by R_{opt} the optimal accepting run of the product $\mathcal{R} \times \mathcal{A}_\varphi$ from Algorithm 3.1, which can be projected onto $\mathcal{R}$ as the *motion and action* plan $\tau_{\text{opt}} = R_{\text{opt}}|_{\Pi_{\mathcal{R}}}$. For each pair of sequential states $(\pi_{\mathcal{R},s}, \pi_{\mathcal{R},s+1})$ in τ_{opt} there exists an action $\alpha_{\mathcal{R}} \in \Sigma_{\mathcal{R}}$ such that $(\pi_{\mathcal{R},s}, \alpha_{\mathcal{R}}, \pi_{\mathcal{R},s+1}) \in \longrightarrow_{\mathcal{R}}$ from (5.5). Thus the underlying low-level control strategy can be synthesized by sequentially implementing the continuous controller associated with the actions along $\tau_{\mathcal{R}}$, in a similar way as Algorithm 3.2. In particular, if $\alpha_{\mathcal{R}} = \sigma_{\mathcal{B},k}$, the controller $\{\mathcal{K}_k\}$ that implements the action $\sigma_{\mathcal{B},k}$ is activated. If $\alpha_{\mathcal{R}} = \sigma_{\mathcal{M}}$, the navigation controller (3.3) is applied to drive the robot from the starting region to one point in the goal region.

The overall framework has four steps: (i) construct the mobility abstraction $\mathcal{M}$ and action map $\mathcal{B}$; (ii) construct the full functionality $\mathcal{R}$ by composing $\mathcal{M}$ and $\mathcal{B}$; (iii) synthesis optimal the motion and action plan τ_{opt}; (iv) execute the plan by the hybrid control strategy.

It is worth mentioning that $\mathcal{M}$ and $\mathcal{B}$ are constructed only once for the robot within a certain workspace and φ can express any task specification in terms of required motions and actions. Steps (ii)-(iv) are performed in an automated way. Whenever a new task specification is given, the complete functionalities model $\mathcal{R}$ remains unchanged and steps (iii)-(iv) are repeated to synthesize the corresponding plan. Whenever the workspace is modified, only $\mathcal{M}$ needs to be re-constructed but the action map $\mathcal{B}$ remains the same and can be reused.

5.2 Multi-robot systems with dependent local tasks

We have discussed in the previous section about how to construct a motion and action plan for a single robot when its individual local task can be fulfilled by itself. But what if one robot needs other robots' collaboration to accomplish one action in its plan, namely whether one robot can accomplish its local task depends on other robots' collaboration. In this section, we address this issue by considering a multi-robot system of N robots with identities $i \in \mathcal{N} = \{1, 2, \cdots, N\}$, as described in Section 4.4, with heterogeneous capabilities and dependent local tasks.

5.2.1 Collaborative actions

In this part, we construct the complete robot model for both motion and actions, where the abstraction for robot motion is similar to Section 5.1.1 but the model of robot actions is different, due to the existence of collaborative and assisting actions.

Same as before, the workspace consists of M partitions as the regions of interest, denoted by $\Pi = \{\pi_1, \pi_2, \cdots, \pi_M\}$. We assume that these symbols are assigned a priori and the workspace is fully-known by all robots. Robot i's motion within the workspace is modeled as a wFTS:

$$\mathcal{M}_i \triangleq (\Pi, \longrightarrow^i_{\mathcal{M}}, \Pi^i_{\mathcal{M},0}, \Psi^i_{\mathcal{M}}, L^i_{\mathcal{M}}, T^i_{\mathcal{M}}), \tag{5.6}$$

where $\longrightarrow^i_{\mathcal{M}} \subseteq \Pi \times \Pi$ is the transition relation; $\Pi^i_{\mathcal{M},0} \in \Pi$ is the initial region robot k starts from; $L^i_{\mathcal{M}} : \Pi \to 2^{\Psi^i_{\mathcal{M}}}$ is the labeling function, indicating the properties held by each region; $T^i_{\mathcal{M}} : \longrightarrow^i_{\mathcal{M}} \to \mathbb{R}^+$ estimates the time each transition takes. A path of $\mathcal{M}_i$ is a sequence of regions $\pi_0\pi_1\cdots\pi_P$, where $(\pi_n, \pi_{n+1}) \in \longrightarrow^i_{\mathcal{M}}$, $\forall n = 0, 1, \cdots, P-1$. Note that $\mathcal{M}_i$ might be different between the robots due to heterogeneity. Moreover, each robot i has a set of neighboring robots, denoted by $\mathcal{N}_i \subseteq \mathcal{N}$ and robot i can exchange messages directly with any robot $j \in \mathcal{N}_i$.

Action model

Besides the motion ability, robot i is capable of performing a set of actions denoted by $\Sigma_i \triangleq \Sigma^i_l \cup \Sigma^i_c \cup \Sigma^i_h$, where

- Σ^i_l is a set of *local* actions, which can be done by robot i itself;
- Σ^i_c is a set of *collaborative* actions, which can be done by robot i but requires collaborations from other robots;
- Σ^i_h is a set of *assisting* actions, which robot i offers to other robots to accomplish their collaborative actions.

In other words, Σ^i_l and Σ^i_c contain active actions that can be *initiated* by robot i, denoted by $\Sigma^i_a = \Sigma^i_l \cup \Sigma^i_c$, while Σ^i_h contains assisting actions only to *assist* other robots. By default, $\sigma_0 = \texttt{None} \in \Sigma^i_l$ means that none of the actions is performed.

Moreover, denote by $\Sigma_h^{\sim i}$ the set of *external* assisting actions robot i depends on, which can only be provided by some of its neighbors in $\mathcal{N}_i$, i.e., $\Sigma_h^{\sim i} \subseteq \cup_{i\in\mathcal{N}_i}\Sigma_h^i$. Table 5.1 shows the action sets of the robots that will be simulated in Section 5.3. The action model of robot i is modeled by a six-tuple:

$$\mathcal{B}_i \triangleq (\Sigma_i, \Psi_\Sigma^i, L_\Sigma^i, \texttt{Cond}_i, \texttt{Dura}_i, \texttt{Depd}_i), \tag{5.7}$$

where Σ_i is the set of actions defined earlier; Ψ_Σ^i is a set of atomic propositions related to the robot's *active* actions; $L_\Sigma^i : \Sigma_i \to 2^{\Psi_\Sigma^i}$ is the labeling function. $L_\Sigma^i(\sigma_h) = \varnothing$, $\forall\sigma_h \in \Sigma_h^i$ and $L_\Sigma^i(\sigma_a) \subseteq \Psi_\Sigma^i$, $\forall\sigma_a \in \Sigma_a^i$; $\texttt{Cond}_i : \Sigma_i \times 2^{\Psi_\mathcal{M}^i} \to \top/\bot$ indicates the set of region properties that have to be fulfilled in order to perform an action; $\texttt{Dura}_i : \Sigma_i \to \mathbb{R}^+$ is the time duration of each action. $\texttt{Dura}_i(\sigma_0) = T_0 > 0$ is a design parameter; $\texttt{Depd}_i : \Sigma_i \to 2^{\Sigma_h^{\sim i}}$ is the *dependence* function. $\texttt{Depd}_i(\sigma_s) = \varnothing$, $\forall\sigma_s \in \Sigma_l^i \cup \Sigma_h^i$ and $\texttt{Depd}_i(\sigma_c) \subseteq \Sigma_h^{\sim i}$, $\forall\sigma_c \in \Sigma_c^i$. Namely each collaborative action depends on a set of assisting actions from its neighbors. This is useful for defining complex collaborations involving multiple robots.

***Remark* 5.3.** Compared with defining dependency directly on robot identities, our action model allows more flexibility since their identities need not be known a priori and new or existing robots can be added or removed. Moreover, different from (5.4), the action model by (5.7) can model both local and collaborative actions. ▲

Definition 5.1. A local or assisting action $\sigma_s \in \Sigma_l^i \cup \Sigma_h^i$ is **done** at region $\pi_v \in \Pi$ if two conditions hold: (i) $\texttt{Cond}_i(\sigma_s, L_\mathcal{M}^i(\pi_v)) = \top$; (ii) σ_s is activated for period $\texttt{Dura}_i(\sigma_s)$. For a collaborative action $\sigma_c \in \Sigma_c^i$, another condition is needed: (iii) all assisting actions in $\texttt{Depd}_i(\sigma_c)$ are done by other robots at the same region π_v. ■

Complete robot model

A complete robot model of robot i, denoted by $\mathcal{R}_i$, refers to the finite transition system that models both its motion and actions.

Definition 5.2. Given $\mathcal{M}_i$ and $\mathcal{B}_i$, robot i's complete model can be constructed as follows: $\mathcal{R}_i = (\Pi_\mathcal{R}^i, \longrightarrow_\mathcal{R}^i, \Pi_{\mathcal{R},0}^i, \Psi_\mathcal{R}^i, L_\mathcal{R}^i, T_\mathcal{R}^i)$, where $\Pi_\mathcal{R}^i = \Pi \times \Sigma_i$. $\pi_{\mathcal{R},s} = \langle\pi_t, \sigma_n\rangle \in \Pi_\mathcal{R}^i$, $\forall\pi_t \in \Pi$, $\forall\sigma_n \in \Sigma_i$; $\longrightarrow_\mathcal{R}^i \subseteq \Pi_\mathcal{R}^i \times \Pi_\mathcal{R}^i$. $\big(\langle\pi_s, \sigma_m\rangle, \langle\pi_t, \sigma_n\rangle\big) \in \longrightarrow_\mathcal{R}^i$ if (i) $\sigma_n = \sigma_m = \sigma_0$, $\pi_s \longrightarrow_\mathcal{M}^i \pi_t$; *or* (ii) $\sigma_m = \sigma_0$, $\sigma_n \neq \sigma_0$ and $\pi_s = \pi_t$, $\texttt{Cond}_i(\sigma_n, L_\mathcal{M}^i(\pi_s)) = \top$; *or* (iii) $\sigma_m \in \Sigma_i$, $\sigma_n = \sigma_0$ and $\pi_s = \pi_t$; $\Pi_{\mathcal{R},0}^i = \Pi_{\mathcal{M},0}^i \times \sigma_0$ is the initial state; $\Psi_\mathcal{R}^i = \Psi_\mathcal{M}^i \cup \Psi_\Sigma^i$; $L_\mathcal{R}^i : \Pi_\mathcal{R}^i \to 2^{\Psi^i}$. $L_\mathcal{R}^i(\langle\pi_s, \sigma_m\rangle) = L_\mathcal{M}^i(\pi_s) \cup L_\Sigma^i(\sigma_m)$; $T_\mathcal{R}^i : \longrightarrow_\mathcal{R}^i \to \mathbb{R}^+$. For case (i) above, $T_\mathcal{R}^i(\langle\pi_s, \sigma_m\rangle, \langle\pi_t, \sigma_n\rangle) = T_\mathcal{M}^i(\pi_s, \pi_t)$; for case (ii), $T_\mathcal{R}^i(\langle\pi_s, \sigma_m\rangle, \langle\pi_t, \sigma_n\rangle) = \texttt{Dura}_i(\sigma_n)$; for case (iii), $T_\mathcal{R}^i(\langle\pi_s, \sigma_m\rangle, \langle\pi_t, \sigma_n\rangle) = T_0$. ■

Note that when defining $\longrightarrow_\mathcal{R}^i$ above, the condition of performing an action is verified over the properties of each region. Thus $\mathcal{R}_i$ is a standard FTS by [11]. Its finite path is denoted by $\tau_i = \pi_{\mathcal{R},0}\pi_{\mathcal{R},1}\cdots\pi_{\mathcal{R},P}$, where $\pi_{\mathcal{R},s} \in \Pi_\mathcal{R}^i$, $\pi_{\mathcal{R},0} \in \Pi_{\mathcal{R},0}^i$ and $(\pi_{\mathcal{R},s}, \pi_{\mathcal{R},s+1}) \in \longrightarrow_\mathcal{R}^i$, $\forall s = 0, \cdots, P-1$, of which the trace is given by $\texttt{trace}(\tau_i) = L_\mathcal{R}^i(\pi_{\mathcal{R},0})L_\mathcal{R}^i(\pi_{\mathcal{R},1}) \cdots L_\mathcal{R}^i(\pi_{\mathcal{R},P})$.

Robot	Σ_l^i	Σ_c^i	Σ_h^i	$\Sigma_h^{\sim i}$
$\mathfrak{R}_1$	l_A, u_A	l_B, u_B	$\varnothing$	h_B
$\mathfrak{R}_2$	s	$\varnothing$	h_B, h_{C_1}, h_F	$\varnothing$
$\mathfrak{R}_3$	$\varnothing$	o_M	h_{C_2}, h_F	h_M
$\mathfrak{R}_4$	s	a_C	h_M, h_F	h_{C_1}, h_{C_2}
$\mathfrak{R}_5$	m_D	$\varnothing$	h_B, h_{C_1}, h_{C_2}	$\varnothing$
$\mathfrak{R}_6$	o_E	c_F	h_B, h_M	h_F

Table 5.1: Action sets (except σ_0) described in Section 5.3.

5.2.2 Problem formulation

The local task of robot i, denoted by φ_i, is given as an sc-safe LTL formula over the set of atomic propositions $\Psi_{\mathcal{R}}^i$ from Definition 5.2. Thus φ_i can contain requirements on robot's motion, local and collaborative actions. The syntax and semantics of sc-LTL can be found in Section 3.2. As also mentioned there, an sc-safe LTL formula can be fulfilled by a finite prefix. In particular, given a finite path τ_i of $\mathcal{R}_i$, then τ_i fulfils φ_i if $\texttt{trace}(\tau_i) \vDash \varphi_i$ where the satisfaction relation is defined there. One special case is that when $\varphi_i \triangleq \top$, robot i does not have a local task and serves as an assisting robot. In summary, we address the following problem in this section:

Problem 5.2. *Given $\mathcal{R}_i$ and its locally-assigned task φ_i that depends on other robots' collaboration, design a distributed control and coordination scheme such that φ_i is fulfilled for all $i \in \mathcal{N}$.* ▲

5.2.3 Distributed task coordination

This section contains the main contribution as the bottom-up motion and task coordination scheme for multi-robot systems with dependent local tasks. It includes the off-line initial plan synthesis, the on-line request and reply messages exchange protocol, the real-time plan adaptation algorithm and the failure recovery mechanism.

Initial plan synthesis

Given the complete robot model and its local task, we can first synthesize an *initial* motion and action plan for each robot, which happens off-line and serves as a starting point for the real-time coordination and adaptation scheme in Section 5.2.3. We intend to find a finite path of $\mathcal{R}_i$ whose trace satisfies the co-safe formula φ_i as described in Section 5.2.2. Let $\mathcal{A}_{\varphi_i}$ be the NBA associated with φ_i from Section 3.2, i.e., $\mathcal{A}_{\varphi_i} = (Q_i, 2^{\Psi_{\mathcal{G}}^i}, \delta_i, Q_0^i, \mathcal{F}_i)$. The product automaton $\mathcal{A}_{p,i}$ is defined as follows:

$$\mathcal{A}_{p,i} = \mathcal{R}_i \otimes \mathcal{A}_{\varphi_i} = (Q_{p,i}, \delta_{p,i}, Q_{p,i,0}, \mathcal{F}_{p,i}, W_{p,i}), \tag{5.8}$$

where $Q_{p,i} = \Pi_{\mathcal{R}}^i \times Q_i$; $(\langle \pi_{\mathcal{R},s}, q_m \rangle, \langle \pi_{\mathcal{R},k}, q_n \rangle) \in \delta_{p,i}$ if it holds that $\pi_{\mathcal{R},s} \longrightarrow_{\mathcal{R}}^i \pi_{\mathcal{R},k}$ and $(q_m, L_{\mathcal{R}}^i(\pi_{\mathcal{R},s}), q_n) \in \delta_i$; $Q_{p,i,0} = \Pi_{\mathcal{R},0}^i \times Q_0^i$ is the set of initial states; $\mathcal{F}_{p,i} = \Pi_{\mathcal{R}}^i \times$

$\mathcal{F}^i$ is the set of accepting states; $W_{p,i} : \delta_{p,i} \times Q_{p,i} \to \mathbb{R}^+$. $W_{p,i}(\langle \pi_{\mathcal{R},s}, q_m \rangle, \langle \pi_{\mathcal{R},k}, q_n \rangle) = T^i_{\mathcal{R}}(\pi_{\mathcal{R},s}, \pi_{\mathcal{R},k})$, where $\langle \pi_{\mathcal{R},k}, q_n \rangle \in \delta_{p,i}(\langle \pi_{\mathcal{R},s}, q_m \rangle)$.

There exists a finite path of $\mathcal{R}_i$ satisfying φ_i if and only if $\mathcal{A}_{p,i}$ has a finite path from an initial state to an accepting state. Then this path could be projected back to $\mathcal{R}_i$ as a finite path, the trace of which should satisfy φ_i automatically [11]. Let $R^i_p = q^i_{p,0} q^i_{p,1} \cdots q^i_{p,P}$ be a finite path of $\mathcal{A}_{p,i}$, where $q^i_{p,0} \in Q_{p,i,0}$, $q^i_{p,P} \in \mathcal{F}_{p,i}$, $q^i_{p,l} \in Q_{p,i}$ and $(q^i_{p,l}, q^i_{p,l+1}) \in \delta_{p,i}$, $\forall l = 0, 1, \cdots, P-1$. The *cost* of R^i_p is defined by $\texttt{Cost}(R^i_p, \mathcal{A}_{p,i}) = \sum_{l=0}^{P-1} W_{p,i}(q^i_{p,l}, q^i_{p,l+1})$, which is the summed weights along R^i_p. The l_{th} element is given by $R^i_p[l] = q^i_{p,l}$ and the segment from the l_{th} to the k_{th} element is $R^i_p[l:k] = q^i_{p,l} q^i_{p,l+1} \cdots q^i_{p,k}$, where $l \le k \le P$.

Problem 5.3. *Find a finite path R^i_p of $\mathcal{A}_{p,i}$ with the above structure that minimizes its total cost.* ▲

Denote by $R^i_{p,\text{init}}$ the solution to the above problem. Algorithm 1 in [58] solves the above problem, which is also restated in Algorithm 3.1. It utilizes Dijkstra's algorithm for computing the shortest path from any initial state in $Q_{p,i,0}$ to every reachable accepting state in $\mathcal{F}^i_p$ and checks if there is cycle back to $q_{p,P}$. The worst-case complexity [89, 97] is $\mathcal{O}(|\delta_{p,i}| \cdot \log |Q_{p,i}| \cdot |Q_{p,i,0}|)$. By projecting $R^i_{p,\text{init}}$ onto $\Pi^i_{\mathcal{R}}$, it gives the initial motion and action plan $\tau^i_{\mathcal{R},\text{init}} = R^i_{p,\text{init}}|_{\Pi^i_{\mathcal{R}}}$ that fulfils φ_i.

***Remark* 5.4.** The initial plans are synthesized locally by each robot instead of by a central unit [18] or within a cluster [51]. ▲

The plan $\tau^i_{\mathcal{R},\text{init}}$ can be executed by activating the motion or actions in sequence. However since $\tau^i_{\mathcal{R},\text{init}}$ may contain several collaborative actions from Σ^i_c to satisfy φ_i, the successful execution of $\tau^i_{\mathcal{R},\text{init}}$ depends on other robots' collaboration, which however is not guaranteed since $\tau^i_{\mathcal{R},\text{init}}$ is synthesized off-line and locally. We resolve this problem by a real-time coordination and adaptation scheme in the following.

On-line collaborative task coordination

There is no guarantee that the initial plan $\tau^i_{\mathcal{R},\text{init}}$ can be executed successfully if it contains collaborative actions. In this part, we propose a distributed and on-line coordination scheme which involves four major parts: (i) a request and reply exchange protocol driven by collaborative actions in a finite horizon; (ii) an optimization and confirmation mechanism, by solving a mixed integer program given the replies; (iii) a real-time plan adaptation algorithm given the confirmation; (iv) an robot failure detection and recovery scheme along with the plan execution.

Planned motion and actions in horizon

Denote by $\pi^i_{\mathcal{R},t} \in \Pi^i_{\mathcal{R}}$ the state of robot i at time t. After the system starts, assume $\pi^i_{\mathcal{R},t}$ is the l_{th} element in $\tau^i_{\mathcal{R},\text{init}}$, namely, $\pi^i_{\mathcal{R},t} = \tau^i_{\mathcal{R},\text{init}}[l]$. Each robot $i \in \mathcal{N}$ is given a bounded planning horizon $0 < H_i < \infty$, which is the time ahead robot i checks

Algorithm 5.7 Plan in horizon and request, `Request()`

Input: $\tau^i_{\mathcal{R},\text{init}}$, $\pi^i_{\mathcal{R},t}$, H_i
Output: $\tau^i_{\mathcal{R},H}$, $\mathbf{Request}^i$
$\tau^i_{\mathcal{R},\text{init}}[l] = \pi^i_{\mathcal{R},t}$, $s = 0$, $T_m = 0$, $\mathbf{Request}^i = \varnothing$
while *($T < H_i$) and ($l + s \le |\tau^i_{\mathcal{R},init}|$)* **do**
  $s = s + 1$, $T_m = T_m + T^i_{\mathcal{R}}\left(\tau^i_{\mathcal{R},\text{init}}[l], \tau^i_{\mathcal{R},\text{init}}[l+s]\right)$
  $\langle \pi_v, \sigma_m \rangle = \tau^i_{\mathcal{R},\text{init}}[l+s]$
  if $\mathbf{Request}^i = \varnothing$ *and* $\sigma_m \in \Sigma^i_c$ **then**
    forall the $\sigma_d \in \mathtt{Depd}_i(\sigma_m)$ **do**
      add (σ_d, π_v, T_m) to $\mathbf{Request}^i$
$k = l + s$, $\tau^i_{\mathcal{R},H} = \tau^i_{\mathcal{R},\text{init}}[l:k]$
return $\tau^i_{\mathcal{R},H}$, $\mathbf{Request}^i$

its plan. Similar approach can be found in [153] for a single dynamic system. Then the sequence of states robot i is expected to reach within the time H_i, denoted by $\tau^i_{\mathcal{R},H}$, is the segment $\tau^i_{\mathcal{R},H} = \tau^i_{\mathcal{R},\text{init}}[l:k]$, where the index $k \ge l$ is the solution to this optimization problem: min k, subject to $\sum_{s=l}^{k} T^i_{\mathcal{R}}\left(\tau^i_{\mathcal{R},\text{init}}[s], \tau^i_{\mathcal{R},\text{init}}[s+1]\right) \ge H_i$. It can be solved by iterating through the sequence of $\tau^i_{\mathcal{R},\text{init}}$ and computing the accumulated cost, which is then compared with H_i. If it does not have a solution, it means H_i is larger than the total cost of the rest of the plan $\tau^i_{\mathcal{R},\text{init}}[l:]$, then $k = |\tau^i_{\mathcal{R},\text{init}}|$, see Algorithm 5.7. This time horizon avoids coordinating on collaborative actions that will be done within a long time from now.

Request to neighbors

Given $\tau^i_{\mathcal{R},H}$ as the motion and actions in horizon, robot i needs to check whether it needs others' collaboration within $\tau^i_{\mathcal{R},H}$. This is done by verifying whether a collaborative action needs to be performed to reach the states in $\tau^i_{\mathcal{R},H}$. More specifically, for the *first* state $\langle \pi_v, \sigma_m \rangle \in \tau^i_{\mathcal{R},H}$ satisfying $\sigma_m \in \Sigma^i_c$, robot i needs to *broadcast* a request to all robots within its communication network $\mathcal{N}_i$ regarding this action. This request message has the following format:

$$\mathbf{Request}^i = \{(\sigma_d, \pi_v, T_m), \forall \sigma_d \in \mathtt{Depd}_i(\sigma_m)\}, \tag{5.9}$$

where $\mathtt{Depd}_i(\sigma_m)$ is the set of external assisting actions that σ_m depends on by (5.7); $\pi_v \in \Pi$ is the region where σ_m will be performed; $T_m \ge 0$ is the time when σ_m will be performed from now. Assume that $\langle \pi_v, \sigma_m \rangle$ is the k_{th} element of $\tau^i_{\mathcal{R},H}$. Then $T_m = \sum_{s=l}^{k} T^i_{\mathcal{R}}\left(\tau^i_{\mathcal{R},H}[s], \tau^i_{\mathcal{R},H}[s+1]\right)$, see Algorithm 5.7. Each element $(\sigma_d, \pi_v, T_m) \in \mathbf{Request}^i$ contains the message that "robot i is requesting the assisting action σ_d at region π_v in the time T_m from now". The request message from robot i to each robot

Algorithm 5.8 Reply to request by robot j, `Reply()`

Input: $\mathbf{Request}_j^i$, $\widehat{R}_{p,-}^j$, $\mathcal{A}_{p,j}$, $\overline{T}_j$
Output: $\mathbf{Reply}_i^j$, $\widehat{P}$
forall the $(\sigma_d, \pi_v, T_m) \in \mathbf{Request}_j^i$ **do**
  if $\overline{T}_j$ *is* 0 **then**
    $\left(\widehat{R}_{p,+}^j, b_d^j, t_d^j\right) = \texttt{EvalReq}(\widehat{R}_{p,-}^j, (\pi_v, \sigma_d, T_m), \mathcal{A}_{p,j})$
    if b_d^j *is* $\top$ **then**
      $\widehat{P}(\sigma_d) = \widehat{R}_{p,+}^j$, add (σ_d, b_d^j, t_d^j) to $\mathbf{Reply}_i^j$
  add $(\sigma_d, \bot, 0)$ to $\mathbf{Reply}_i^j$
Return $\mathbf{Reply}_i^j$, $\widehat{P}$

$j \in \mathcal{N}_i$, denoted by $\mathbf{Request}_j^i$, is the same as $\mathbf{Request}^i$, i.e., $\mathbf{Request}_j^i = \mathbf{Request}^i$, $\forall j \in \mathcal{N}_i$.

***Remark* 5.5.** The request message is sent only for the *first* collaborative action in $\tau_{\mathcal{R},H}^i$ within the time horizon H_i (see Algorithm 5.7), as the outcome of this request would greatly affect the second collaborative action in $\tau_{\mathcal{R},H}^i$. ▲

Request evaluation and reply

Upon receiving the request, robot $j \in \mathcal{N}_i$ needs to evaluate this request in terms of *feasibility* and *cost*, in order to reply to robot i. Specifically, the reply message from robot j to robot i has the following format:

$$\mathbf{Reply}_i^j = \{(\sigma_d, b_d^j, t_d^j), \forall (\sigma_d, \pi_v, T_m) \in \mathbf{Request}_j^i\}, \tag{5.10}$$

where σ_d is the requested assisting action by robot i; b_d^j is a boolean variable indicating the feasibility of robot j offering action σ_d at region π_v; $t_d^j \geq 0$ is time when that can happen. We describe how to determine b_d^j and t_d^j below.

Denote by $\overline{T}_j \geq 0$ the finishing time of the current collaboration robot i is engaged in. It is initialized as 0 and updated in Section 5.2.3. As shown in Algorithm 5.8, (I) if $\overline{T}_j > 0$, it means robot j is engaged in a collaboration. Then $\mathbf{Reply}_i^j = \{(\sigma_d, \bot, 0), \forall (\sigma_d, \pi_v, T_m) \in \mathbf{Request}_j^i\}$, meaning that robot j would *reject* any request before its current collaboration is finished. (II) If $\overline{T}_j = 0$, it means robot j is available to offer assisting actions. Then for each request $(\sigma_d, \pi_v, T_m) \in \mathbf{Request}_j^i$, robot i needs to evaluate it in terms of feasibility and cost to determine b_d^j and t_d^j. Clearly, robot j needs to potentially revise its current plan to incorporate the request, i.e., to offer the assisting action σ_d at region π_v by time T_m. Denote by $\tau_{\mathcal{R},t^-}^j$ the plan of robot j before the *potential* revision, of which the corresponding accepting run is R_{p,t^-}^j. Assume that robot j's current state $q_{p,t}^j$ is the l_{th} element of R_{p,t^-}^j and the accepting state $q_{p,f}^j$ is the last and f_{th} element. Then the segment from $q_{p,t}^j$ to

Algorithm 5.9 Evaluate the request, `EvalReq`$(\cdot)$

Input: $\widehat{R}^j_{p,-}$, (π_v, σ_d, T_m), $q^j_{p,t}$, $\mathcal{A}_{p,j}$
Output: $\widehat{R}^j_{p,+}$, b^j_d, t^j_d
$q^j_{p,t} = \widehat{R}^j_{p,-}[1]$, $q^j_{p,f} = \widehat{R}^j_{p,-}[\text{-}1]$
Compute S_d, S_c
$\bar{c} = \texttt{Cost}(\widehat{R}^j_{p,-}, \mathcal{A}_{p,j})$
$(P_1, C_1) = \texttt{DijksTA}\big(\mathcal{A}_{p,j}, q^j_{p,t}, S_d, S_c\big)$
$(P_2, C_2) = \texttt{DijksTA}\big(\texttt{Reverse}(\mathcal{A}_{p,j}), q^j_{p,f}, S_d, \varnothing\big)$
forall the $q^j_{p,r} \in S_d$ **do**
  if $P1(q^j_{p,r})$ *and* $P2(q_{p,r})$ *exist* **then**
    $C_3(q^j_{p,r}) = |C_1(q^j_{p,r}) - T_m| + \alpha_j\big(C_1(q^j_{p,r}) + C_2(q^j_{p,r}) - \bar{c}\big)$
Find the $q^{j,\star}_{p,r} \in S_d$ that minimizes $C_3(q^j_{p,r})$
if $q^{j,\star}_{p,r} \neq \varnothing$ **then**
  $P = P_1(q^{j,\star}_{p,r}) + \texttt{Reverse}\big(P_2(q^{j,\star}_{p,r})\big)$
  Return $\widehat{R}^j_{p,+} = P$, $b^j_d = \top$, $t^j_d = C_1(q^{j,\star}_{p,r})$
Return $\widehat{R}^j_{p,+} = \varnothing$, $b^j_d = \bot$, $t^j_d = 0$

$q^j_{p,f}$ is given by $\widehat{R}^j_{p,-} = R^j_{p,t^-}[l\!:\!f]$. We intend to find another segment $\widehat{R}^j_{p,+}$ within $\mathcal{A}_{p,j}$ from $q^j_{p,t}$ to $q^j_{p,f}$, such that by following $\widehat{R}^j_{p,+}$: (i) robot j should *reach* state $\langle \pi_v, \sigma_d \rangle$; (ii) the time to reach $\langle \pi_v, \sigma_d \rangle$ should be *close* to T_m; (iii) the additional cost of $\widehat{R}^j_{p,+}$ compared to $\widehat{R}^j_{p,-}$ should be *small*. We enforce those conditions below.

Firstly, the set of product states in $\mathcal{A}_{p,j}$ corresponding to $\langle \pi_v, \sigma_d \rangle$ is given by $S_d = \{q_p \in Q^j_p \,|\, q_p|_{\Pi^j_{\mathcal{R}}} = \langle \pi_v, \sigma_d \rangle\}$. Consider $\widehat{R}^j_{p,+}$ with the following structure:

$$\widehat{R}^j_{p,+} = q^j_{p,t} \cdots q^j_{p,r} \cdots q^j_{p,f}, \tag{5.11}$$

where $q^j_{p,r} \in S_d$, meaning that it passes through at least one state within S_d. Thus the corresponding plan would contain $\langle \pi_v, \sigma_d \rangle$, which fulfils the condition (i) above. Regarding conditions (ii) and (iii), we define the *balanced* cost of $\widehat{R}^j_{p,+}$:

$$\begin{aligned} & \texttt{BalCost}(\widehat{R}^j_{p,+}, T_m, \mathcal{A}_{p,j}) \\ = \; & |\sum_{s=1}^{r-t} W^j_p\big(\widehat{R}^j_{p,+}[s], \widehat{R}^j_{p,+}[s+1]\big) - T_m| \\ & + \alpha_j \left(\texttt{Cost}(\widehat{R}^j_{p,+}, \mathcal{A}_{p,j}) - \texttt{Cost}(\widehat{R}^j_{p,-}, \mathcal{A}_{p,j})\right), \end{aligned} \tag{5.12}$$

where the first part stands for the time gap between the requested time T_m by robot i and the actual time based on $\widehat{R}^j_{p,+}$ (for condition (ii)); the second term is the additional cost of $\widehat{R}^j_{p,+}$, compared to $\widehat{R}^j_{p,-}$ (for condition (iii)); $\alpha_j > 0$ is a design parameter as the relative weighting.

Problem 5.4. *Given* $\widehat{R}^j_{p,-}$, S_d *and* $\mathcal{A}_{p,j}$, *find the path segement* $\widehat{R}^j_{p,+}$ *that minimizes* (5.12). ▲

Algorithm 5.9 solves the above problem by the *bidirectional* Dijkstra algorithm. It utilizes the function `DijksTA`$(\cdot)$ that computes shortest paths in a weighted graph from the single source state to every state in the set of target states, while at the same time *avoiding* a set of states. It is a simple extension of the classic Dijkstra shortest path algorithm [97]. `DijksTA`$(\mathcal{A}_{p,j}, q^j_{p,t}, S_d, S_c)$ determines the shortest path (saved in P_1) from $q^j_{p,t}$ to every state in S_d while avoiding any state belonging to S_c and the associated costs (saved in C_1), where S_c is the set of all product states associated with a collaborative *or* an assisting action: $S_c = \{q_p \in Q_{p,j} \,|\, q_p|_{\Pi^j_{\mathcal{R}}} = \langle \pi^j_{\mathcal{R}}, \sigma_n \rangle, \sigma_n \in \Sigma^j_c \cup \Sigma^j_h\}$. Then `DijksTA`$(\texttt{Reverse}(\mathcal{A}_{p,j}), q^j_{p,f}, S_d, \varnothing)$ is called to determined the shortest path (saved in P_2) from $q^j_{p,f}$ to every state in S_d within the reversed $\mathcal{A}_{p,j}$ and the associated distances (saved in C_2); `Reverse`$(\mathcal{A}_{p,j})$ is the directed graph obtained by inverting the direction of all edges in $G(\mathcal{A}_{p,j})$ while keeping the weights unchanged, where $G(\mathcal{A}_{p,j})$ is the directed graph associated with $\mathcal{A}_{p,j}$ [11]. For each state $q^j_{p,r} \in S_d$, the balanced cost of the corresponding $\widehat{R}^j_{p,+}$ by (5.12) is computed. The one that yields the minimal cost is denoted by $q^{j,\star}_{p,r}$. At last, $\widehat{R}^j_{p,+}$ is formed by concatenating the shortest path from $q^j_{p,t}$ to $q^{j,\star}_{p,r}$ and the reversed shortest path from $q^j_{p,f}$ to $q^{j,\star}_{p,r}$. Last but not least, if $q^{j,\star}_{p,r}$ returns empty, it means that robot i could not offer the requested collaboration thus $b^j_d = \bot$ and $t^j_d = 0$. The complexity [89, 97] of function `DijksTA`$(\cdot)$ over $\mathcal{A}_{p,j}$ is $\mathcal{O}(|\delta_{p,j}| \cdot \log |Q_{p,j}|)$; reversing $\mathcal{A}_{p,j}$ has the complexity linear to $\mathcal{O}(|\delta_{p,j}|)$, where $|Q_{p,j}|$, $|\delta_{p,j}|$ are the number of states and transitions in $\mathcal{A}_{p,j}$.

***Remark* 5.6.** Note that $\widehat{R}^j_{p,+}$ is the *potentially-revised* run, i.e., robot j does not change its current plan but saves $\widehat{R}^j_{p,+}$ in $\widehat{P}$ (see Algorithm 5.8) and waits for the confirmation from robot i, which will be discussed in Section 5.2.3. ▲

It is worth mentioning that in case robot i receives requests from multiple robots, it needs to reply to one robot first and wait for the confirmation before it replies to the next robot. Table 5.2 shows the request and reply messages regarding two collaborative actions in Section 5.3.

Lemma 5.1. *If* $b^j_d = \top$ *from Algorithm 5.9, action* σ_d *can be done at the time* t^j_d *by robot* j *following* $\widehat{R}^j_{p,+}$.

Proof. Since the first segment of $\widehat{R}^j_{p,+}$ from $q^j_{p,t}$ to $q^{j,\star}_{p,r}$ is derived by `DijksTA`$(\cdot)$ of Algorithm 5.9, it does not contain any collaborative or assisting actions except σ_d. Thus it can be accomplished by robot j itself with only motions and local actions, of which the time is t^j_d. ■

Request	$\mathfrak{R}_1, (h_B, r_4, 11)$	$\mathfrak{R}_4, (h_{C_1}, r_5, 14)$	$\mathfrak{R}_4, (h_{C_2}, r_5, 14)$
$\mathfrak{R}_1$	$--$	$(h_{C_1}, \perp, 0)$	$(h_{C_2}, \perp, 0)$
$\mathfrak{R}_2$	$(h_B, \top, 13.1)$	$(h_{C_1}, \top, 14.6)$	$(h_{C_2}, \perp, 0)$
$\mathfrak{R}_3$	$(h_B, \perp, 0)$	$(h_{C_1}, \perp, 0)$	$(h_{C_2}, \top, 16.2)$
$\mathfrak{R}_4$	$(h_B, \perp, 0)$	$--$	$--$
$\mathfrak{R}_5$	$(h_B, \top, 15.7)$	$(h_{C_1}, \top, 15.4)$	$(h_{C_2}, \top, 15.4)$
$\mathfrak{R}_6$	$(h_B, \top, 18.2)$	$(h_{C_1}, \perp, 0)$	$(h_{C_2}, \perp, 0)$

Table 5.2: Request and reply messages exchanged for collaborative actions o_M and a_C in Section 5.3, regarding assisting actions h_B and h_{C_1}, h_{C_2}.

Confirmation

Based on the replies from $j \in \mathcal{N}_i$, robot i needs to acknowledge them by sending back confirmation messages:

$$\mathbf{Confirm}_j^i = \{(\sigma_d,\, c_d^j,\, f_m), \forall \sigma_d \in \texttt{Depd}_i(\sigma_m)\}, \tag{5.13}$$

where σ_d is the requested assisting action; c_d^j is a boolean variable, indicating whether robot j is confirmed to provide σ_d; f_m is the time to finish action σ_m. The choices of $\{c_d^j, j \in \mathcal{N}_i\}$ should satisfy two constraints: (i) *exactly* one robot in $\mathcal{N}_i$ can be the confirmed collaborator for each action $\sigma_d \in \texttt{Depd}_i(\sigma_m)$; (ii) each robot in $\mathcal{N}_i$ can be confirmed for *at most* one action in $\texttt{Depd}_i(\sigma_m)$. Meanwhile, the finishing time f_m should be as early as possible.

Let $|\mathcal{N}_i| = N_1$ and $|\texttt{Depd}_i(\sigma_m)| = N_2$. Without loss of generality, denote by $\mathcal{N}_i = \{1, \cdots, N_1\}$ and $\texttt{Depd}_i(\sigma_m) = \{\sigma_1, \cdots, \sigma_{N_2}\}$. The problem of finding $\{c_d^j\}$ and f_m can be readily formulated as an integer programming problem [152]:

$$\begin{aligned}
&\textbf{min} \quad f_m \\
&\textbf{s.t.} \quad f_m = \mathbf{max}_d\{c_d^j \cdot t_d^j,\, T_m\} \\
&\qquad \sum_{d=1}^{N_2} b_d^j \cdot c_d^j \le 1, \quad \forall j \in \{1, \cdots, N_1\}, \\
&\qquad \sum_{j=1}^{N_1} b_d^j \cdot c_d^j = 1, \quad \forall d \in \{1, \cdots, N_2\},
\end{aligned} \tag{5.14}$$

where $(\sigma_d,\, b_d^j,\, t_d^j) \in \mathbf{Reply}_i^j$ from (5.10). Any stand-alone integer programming solver can be used to obtain $\{c_d^j\}$ and f_m once (5.14) is formulated, e.g., "`Gurobi`" and "`CVXOPT`". Then $\forall j \in \mathcal{N}_i$ and $\forall \sigma_d \in \texttt{Depd}_i(\sigma_m)$, consider two cases: (I) if (5.14) has a solution, both $\{c_d^j\}$ and f_m exist. If c_d^j is $\top$, add $(\sigma_d, \top, f_m)$ to $\mathbf{Confirm}_j^i$; otherwise, add $(\sigma_d, \perp, 0)$ to $\mathbf{Confirm}_j^i$; (II) if (5.14) has no solutions, add $(\sigma_d, \perp, 0)$ to $\mathbf{Confirm}_j^i$. It means that σ_m can not be fulfilled according to the current replies. Then how robot i needs to delay σ_m and revise its plan will be given in Section 5.2.3.

Algorithm 5.10 Delay collaboration, `DelayCol()`

Input: $\widehat{R}^i_{p,-}$, $q^i_{p,t}$, $\langle \pi_v, \sigma_d \rangle$, $\mathcal{A}_{p,i}$, $\widehat{\Sigma}^i_c$
Output: $\widehat{R}^i_{p,+}$
Compute S_d given $\langle \pi_v, \sigma_d \rangle$, S_c
$q^i_{p,t} = \widehat{R}^i_{p,-}[1]$, $q^i_{p,f} = \widehat{R}^i_{p,-}[\text{-}1]$
$(P_1, C_1) = \texttt{DijksTA}\big(\mathcal{A}_{p,i}, q^i_{p,t}, S_d, S_c\big)$
$(P_2, C_2) = \texttt{DijksTA}\big(\texttt{Reverse}(\mathcal{A}_{p,i}), q^i_{p,f}, S_d, \varnothing\big)$
forall the $q^i_{p,d} \in S_d$ **do**
  if $C_1(q^i_{p,d}) > T_m + D_i$ *and* $P_2(q^i_{p,d})$ *exists* **then**
    $\widehat{R}^k_{p,+} = P_1(q^i_{p,d}) + \texttt{Reverse}\big(P_2(q^i_{p,d})\big)$
    Return $\widehat{R}^i_{p,+}$

***Remark* 5.7.** The optimization problem (5.14) is solved locally by robot i regarding the requested collaborative task σ_m, with $|\mathcal{N}_i| \cdot |\texttt{Depd}_i(\sigma_m)|$ Boolean variables. ▲

Plan adaptation

After sending out the confirmation messages, robot i checks the following: (I) if (5.14) has a solution, it means that σ_m can be fulfilled and $R^i_{p,t}$ remains unchanged. $\overline{T}_i$ is set to f_m to indicate that robot i is engaged in the collaboration until the time f_m; (II) otherwise, it means that according to the current replies σ_m can not be done as planned in $R^i_{p,t}$. Thus robot i needs to revise its plan by delaying this collaborative action σ_m. Algorithm 5.10 revises $R^i_{p,-}$ and delays σ_m by time D_i, where $D_i > 0$ is a design parameter. Function $\texttt{DijksTA}(\cdot)$ from Algorithm 5.9 is used to find a path from $q^i_{p,t}$ to one state in S_d whose cost is larger than $T_m + D_i$ and at the same time reachable to the accepting state $q^i_{p,f}$. Such a path can always be found as the action σ_0 that takes time T_0 can be repeated as many times as needed.

On the other hand, upon receiving $\mathbf{Confirm}^i_j$, each robot $j \in \mathcal{N}_i$ checks the following: (I) if $b^j_d = \top$, it means robot j is confirmed to offer the assisting action σ_d. As a result, it modifies its plan based on the potential set of plans $\widehat{P}$ from Algorithm 5.9. In particular, the plan segment $\widehat{R}^j_p$ is set to $\widehat{R}^j_{p,+}$ and $\overline{T}_j$ is set to f_m; (II) if robot $b^j_d = \bot, \forall \sigma_d \in \texttt{Depd}_i(\sigma_m)$, it means robot j is not confirmed as a collaborator. Then R^j_p remains unchanged and $\overline{T}_j$ is set to 0. Afterwards, robot i and all confirmed collaborators in $\mathcal{N}_i$ would execute its plan by following the motion and action in sequence. They would reject any further request as described in Section 5.2.3 until the collaboration for action σ_m is done.

Theorem 5.2. *If* (5.14) *has a solution, the time of accomplishing action* σ_m *is* f_m.

Proof. Since each assisting action $\sigma_d \in \texttt{Depd}_i(\sigma_m)$ has been assigned exactly to one robot $j \in \mathcal{N}_i$, then σ_d can be accomplished by robot j at time t^j_d by Lemma 5.1.

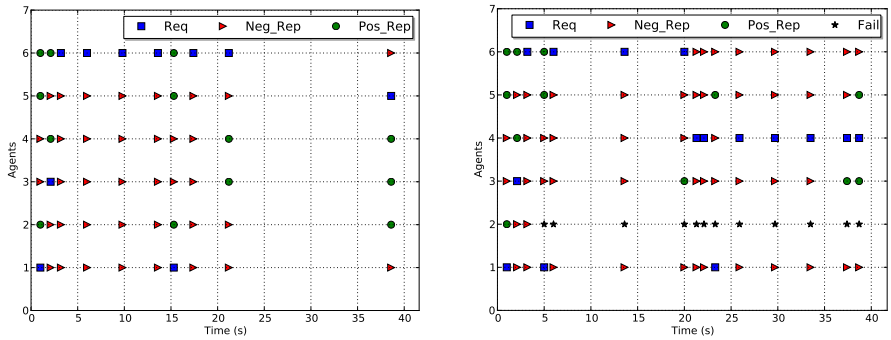


Figure 5.2: Messages exchanged for both scenarios of Section 5.3. Blue square stands for request messages by (5.9) while red triangles and green dots stand for reply messages by (5.10) with b_d^j being $\bot$ and $\top$. Black stars indicate the robot failure.

Thus σ_m can be accomplished by robot i at the time f_m, which is the latest time for all actions by (5.14). ■

Since each robot has a finite plan as a finite sequence of motion and actions, when one robot finishes executing its plan, it would become an assisting robot by setting its task $\varphi_i \triangleq \top$. Then it would stay at one region unless it is confirmed to collaborate with others.

5.2.4 Failure detection and recovery

Due to the unavoidable characteristics and constraints of physical robots, any robot may fail (stop being functional) at anytime. Detection of this type of failure is particularly important for collaborations involving several robots.

We propose an inquiry and acknowledgment mechanism as follows: robot i that has confirmed on its collaborative action σ_m would continuously monitor the status of its confirmed collaborators for each assisting action $\sigma_d \in \texttt{Depd}_i(\sigma_m)$ until σ_m is done. If, for instance, robot $h \in \mathcal{N}_i$ who is confirmed to offer action σ_d fails to acknowledge robot i's inquiry for a limited time, robot i may assume that robot h has failed. As a result, robot i needs to re-assign the assisting action σ_d to another robot in $\mathcal{N}_i$, as follows: (i) send a new request $\{(\pi_v, \sigma_d, T_m)\}$ to all neighbors within $\mathcal{N}_i$ and wait for the reply; (ii) based on the replies $\{(\sigma_d, b_d^j, t_d^j)\}, \forall j \in \mathcal{N}_i\}$, choose the new collaborator $\hat{h} \in \mathcal{N}_i$ for action σ_d as follows: $\hat{h} = \text{argmin}_{j \in \mathcal{N}_i}\{|f_m - t_d^j \cdot b_d^j|\}$, where f_m is the previously confirmed time from (5.14). Namely, it chooses robot $\hat{h}$ that can offer action σ_d at the time closest to f_m; (iii) send the confirmation $\{(\sigma_d, \top, f_m)\}$ to robot $\hat{h}$; (iv) robot $\hat{h}$ adapts its plan and be engaged in the collaboration on σ_m as described in Section 5.2.3.

Loosely-coupled system

As mentioned earlier, we aim to apply this distributed coordination scheme to loosely-coupled multi-robot systems, where collaborations among the robots are (i) local in the sense that only neighboring robots are needed; (ii) sparse in the sense that they are needed infrequently compared with the total number of activities of all robots required by their local tasks, which imposes the following assumption:

Assumption 5.1. *There exists a finite time* $\mathbf{T} > 0$ *such that for each robot* $i \in \mathcal{N}$ *and any collaborative action* σ_m *requested by robot* i *initially at time* $t_m > 0$*, problem* (5.14) *for* σ_m *will have a solution within time* $t_m + \mathbf{T}$. ▲

Namely, the above assumption indicates that any collaborative action required by each robot $i \in \mathcal{N}$ in order to satisfy the local task φ_i should always eventually be provided in finite time by robot i and its neighbors.

5.2.5 Overall structure

During the real-time execution, each robot executes its plan and checks first if any request is received. If so, it replies to them by Algorithm 5.8, waits for the confirmation and adjusts its plan accordingly. Otherwise, it sends out requests by Algorithm 5.7, waits for reply, sends confirmation back by (5.14) and at last adapts its plan by Algorithm 5.10. The correctness of the proposed scheme is guaranteed by Theorem 5.3 below:

Theorem 5.3. *Under Assumption 5.1, the proposed coordination scheme solves Problem 5.2. Namely, all local tasks* φ_i *can be accomplished in finite time,* $\forall i \in \mathcal{N}$.

Proof. Starting from the initial plan $\tau^i_{\mathcal{R},\text{init}}$ for robot $i \in \mathcal{N}$, motion and local actions in $\tau^i_{\mathcal{R},\text{init}}$ can be accomplished locally by an robot itself. If $\tau^i_{\mathcal{R},\text{init}}$ remains unchanged for robot i, we only need to show that collaborative actions in $\tau^i_{\mathcal{R},\text{init}}$ can be accomplished. The fact that $\tau^i_{\mathcal{R},\text{init}}$ remains unchanged indicates that robot i's requests for each collaborative action σ_m are fulfilled, i.e., (5.14) for σ_m has a solution. By Theorem 5.2, action σ_m can be accomplished in finite time f_m. Since $\tau^i_{\mathcal{R},\text{init}}$ is a finite sequence, φ_i can be satisfied in finite time as every motion and action inside can be done in finite time. On the other hand, if $\tau^i_{\mathcal{R},\text{init}}$ has to be adapted in real-time, the reasons are: (i) robot i is confirmed to assist its neighbor j on one collaboration; (ii) robot i has made a request for a collaborative action σ_m and it is delayed by Algorithm 5.10 as (5.14) has no solution. For case (i), Algorithm 5.9 guarantees that after the assistance its updated run $\hat{R}^i_{p,+}$ still satisfies φ_i in finite time. For case (ii), by Assumption 5.1, (5.14) will have a feasible solution within at most time $\mathbf{T}$, meaning that σ_m will be done within finite time. ■

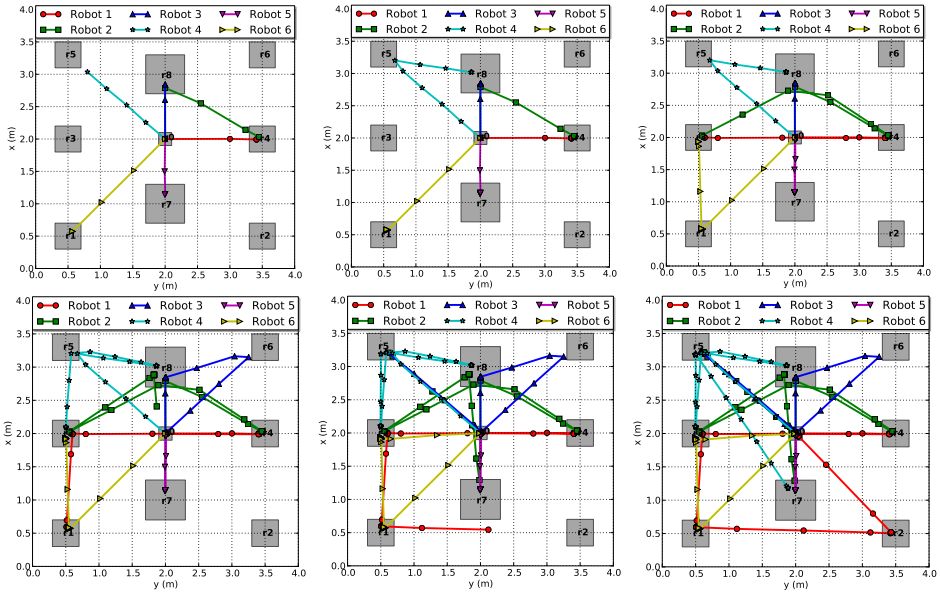


Figure 5.3: Snapshots of collaborations on l_B, o_M, u_B, c_F and a_C in Section 5.3.

5.3 Case study

In the case study, we present a simulated example of six autonomous robots with heterogeneous capacities. The proposed algorithms are implemented in Python 2.7.

System description

The workspace of size $4m \times 4m$ is given in Figure 5.3, within which there are nine rectangular regions of interest $r_0, r_1, r_2, \cdots, r_8$ (in gray). The regions are labelled by the objects of interest, like A, B, C, M, E, F. Denote by the six robots $\mathfrak{R}_1, \mathfrak{R}_2, \cdots, \mathfrak{R}_6$. They all satisfy the single-integrator dynamics, i.e., $\dot{x} = u$, where $x, u \in \mathbb{R}^2$ are the 2-D position and velocity. The robots have velocities between $0.6m/s$ and $1m/s$. Besides, the action sets of each robot are shown in Table 5.1. Robot $\mathfrak{R}_1$ can load and unload (l_A, u_A) a light object A; load and unload (l_B, u_B) a heavy object B with others' assisting action h_B. Robot $\mathfrak{R}_2$ can take pictures (s); help load and unload (h_B) B; help assemble (h_{C1}) C; help charge (h_F) F. Robot $\mathfrak{R}_3$ can operate (o_M) machine M with assisting action h_M; help assemble (h_{C2}) C; help charge (h_F) F. Robot $\mathfrak{R}_4$ can assemble (a_C) C with assisting actions h_{C1}, h_{C2}; help operate (h_M) M; help charge (h_F) F. Robot $\mathfrak{R}_5$ can maintain D, help assemble (h_{C1}, h_{C2}) C, and help load and unload (h_B) B. Robot $\mathfrak{R}_6$ can operate object (o_E) E, charge object (c_F) F with assisting action h_F, help operate (h_M) M and help load and unload (h_B) B. We assume all actions have duration $10s$ and the time horizon H_i is set to $20s$ uniformly. Any two robots can exchange messages directly. "`Gurobi`" for

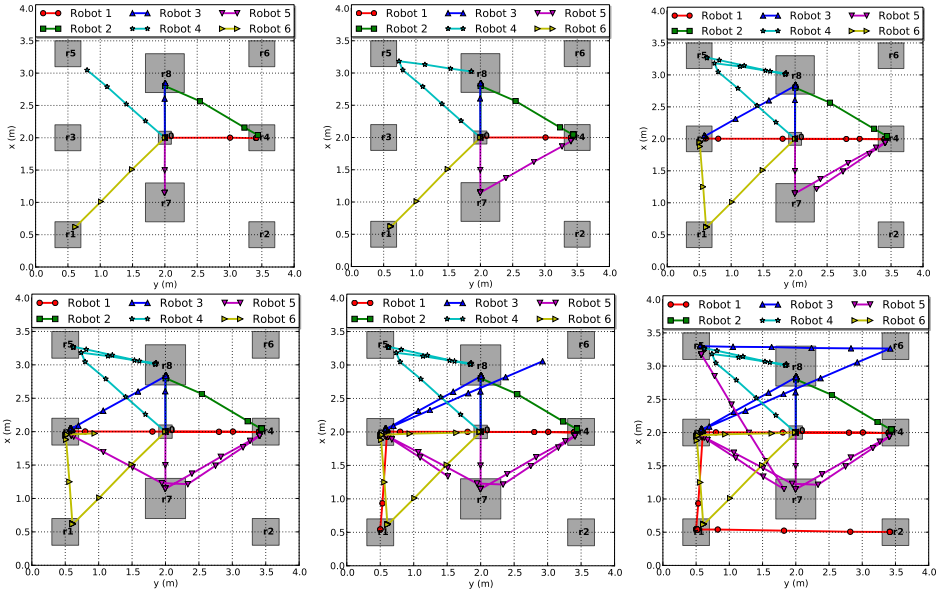


Figure 5.4: Snapshots of collaborations on l_B, o_M, u_B, c_F and a_C for Section 5.3. Note that $\mathfrak{R}_2$ failed at $t = 5s$, instead $\mathfrak{R}_5$ is re-assigned to assist $\mathfrak{R}_1$ on l_B, u_B.

Python is used as the mix integer programming solver.

Each robot is assigned a local task as follows: robot $\mathfrak{R}_1$ has to deliver A to r_2 and B to r_3. Then $\varphi_1 = \Diamond\big(l_A \wedge \Diamond(r_2 \wedge \bigcirc u_A)\big) \wedge \Diamond\big(l_B \wedge \Diamond(r_3 \wedge \bigcirc u_B)\big)$; Robot $\mathfrak{R}_2$ has to surveil regions r_7, r_8 and take pictures there. $\varphi_2 = \Diamond(r_7 \wedge \bigcirc s) \wedge \Diamond(r_8 \wedge \bigcirc s)$; Robot $\mathfrak{R}_3$ has to operate M and visit r_6. $\varphi_3 = \Diamond(o_M \wedge \Diamond r_6)$; Robot $\mathfrak{R}_4$ has to take a photo in r_7, assemble C and visit r_6. $\varphi_4 = \Diamond(r_7 \wedge \bigcirc s) \wedge \Diamond(a_C)$; Robot $\mathfrak{R}_5$ needs to maintain D and back to region r_0. $\varphi_5 = \Diamond(m_D \wedge \Diamond r_0)$; Robot $\mathfrak{R}_6$ needs to operate E first and then charge F. $\varphi_6 = \bigcirc(o_E \wedge \Diamond c_F)$.

Simulation results

All robots start form the center $(2m, 2m)$. The synthesized initial plan of each robot is as follows: $r_0 r_4\, l_B\, r_3\, u_B\, r_1\, l_A\, r_2\, u_A$ for $\mathfrak{R}_1$; $r_0\, r_8\, s\, r_7\, s$ for $\mathfrak{R}_2$; $r_0\, r_8\, o_M\, r_6$ for $\mathfrak{R}_3$; $r_0\, r_5\, a_C\, r_7\, s$ for $\mathfrak{R}_4$; $r_0\, r_7\, m_D$ for $\mathfrak{R}_5$; $r_0\, r_1\, o_E\, r_3 c_F$ for $\mathfrak{R}_6$. We simulate first one nominal scenario and another with $\mathfrak{R}_2$'s failure since $t = 5s$. The messages exchanged during both scenarios are shown in Figure 5.2.

Nominal scenario. The system is simulated for $70.3s$ before all robots accomplish their tasks, during which five collaborations among the robots are accomplished. As shown in Figure 5.3, robot $\mathfrak{R}_1$ firstly sends the request on action h_B and the reply messages are shown in Table 5.2. $\mathfrak{R}_2$ is confirmed as the collaborator and changes its plan to $r_0\, r_4\, h_B\, r_8\, s\, r_7\, s$. $\mathfrak{R}_1$ finishes action l_B at $t = 14.3s$ At the same time, $\mathfrak{R}_4$ is chosen to help $\mathfrak{R}_3$ on action o_M, which is done at $t = 21.1s$. Afterwards, $\mathfrak{R}_1$

finishes u_B at $t = 28.4s$ with $\mathfrak{R}_2$ offering h_B. Robot $\mathfrak{R}_6$'s request for action c_F keeps getting delayed until $t = 21.1s$, as $\mathfrak{R}_2$, $\mathfrak{R}_3$, $\mathfrak{R}_4$ are engaged in other collaborations. Afterwards robot $\mathfrak{R}_4$ is confirmed to offer action h_F and c_F is done at $t = 38.0s$. After that, robot $\mathfrak{R}_4$ sends the request for a_C regarding h_{C_1} and h_{C_2}. The reply messages are shown in Table 5.2. After solving (5.14), $\mathfrak{R}_2$ and $\mathfrak{R}_3$ offer actions h_{C_1} and h_{C_2}, which are done at $t = 54.1s$. The simulation video is online [54].

Failure recovery. To illustrate the effectiveness of our approach for handling robot failures, we stop simulating robot $\mathfrak{R}_2$ since $t = 5s$ whereas the other robots remain the same. As shown in Figure 5.4, initially robot $\mathfrak{R}_2$ is confirmed to offer h_B to $\mathfrak{R}_1$. However since $\mathfrak{R}_2$ fails at $t = 5s$, $\mathfrak{R}_1$ detects that by the inquiry and acknowledgment mechanism described in Section 5.2.4 and resends a request regarding h_B, for which $\mathfrak{R}_5$ is confirmed as the new collaborator. Then l_B is done at $t = 22.3s$. Afterwards, again with the help of $\mathfrak{R}_5$, $\mathfrak{R}_1$ finishes u_B at $t = 37.8s$. $\mathfrak{R}_3$ finishes o_M with the help of $\mathfrak{R}_4$ at $t = 29.7s$. Before this time, robot $\mathfrak{R}_6$ has to delay its action c_F as both $\mathfrak{R}_3$ and $\mathfrak{R}_4$ are engaged in o_M and $\mathfrak{R}_2$ has failed. Then $\mathfrak{R}_3$ offers h_F to $\mathfrak{R}_6$ at $t = 36s$. After that, $\mathfrak{R}_4$ finishes a_C with the help of $\mathfrak{R}_3$ and $\mathfrak{R}_5$ at $t = 68.9s$. At last, each robot fulfills its local task by $t = 76.5s$. The simulation video is online [55].

5.4 Summary

In this chapter, we first presented an automated framework for a single robot to fulfill its local task given as the desired motion and actions. Then we introduced a novel method to define the dependency relations within multi-robot systems using collaborative and assisting actions. Based on this dependency, a bottom-up motion and task coordination scheme was proposed where the collaborative sub-tasks are coordinated and executed successfully during run time.

Chapter 6

Inter-robot relative motion constraints

RELATIVE-MOTION CONSTRAINTS, such as relative-distance constraints, collision avoidance and connectivity maintenance of the communication network, are closely related to the stability, safety and integrity of the overall multi-robot system. Thus this chapter is focused on the actual robot dynamics and the relative-motion constrains among the robots when moving within the same workspace. Particularly, we provide two different control strategies to handle these inter-robot constrains along with the local LTL tasks of each robot. The first approach integrates the potential-field-based controller design with the discrete plan coordination; the second uses Embedded Graph Grammars (EGGs) as the main tool. Both approaches are distributed and can guarantee the satisfaction of all local tasks while obeying the relative-distance constraints at all time. Numerical simulations are provided in the end to validate both control approaches.

6.1 Potential-field-based hybrid control

In this section, we firstly formulate the problem where each robot with a single-integrator dynamics is required to fulfill a local LTL task and simultaneously subject to the relative-motion constraints. Then we propose a hybrid control strategy that relies on the potential-field-based continuous controller design and a distributed task coordination scheme.

6.1.1 Problem formulation

Consider again a team of N autonomous robots with identities $i \in \mathcal{N} = \{1, 2, \cdots, N\}$. Each robot $i \in \mathcal{N}$ satisfies the single-integrator dynamics:

$$\dot{x}_i(t) \triangleq u_i(t), \qquad i \in \mathcal{N}, \tag{6.1}$$

where $x_i(t)$, $u_i(t) \in \mathbb{R}^2$ are the respective state and the control input of robot i at time $t > 0$. Let $x_i(0)$ be the given initial state. The robots are modeled as point masses without volume, i.e., inter-robot collisions are not considered.

Moreover, each robot has a sensing radius $r > 0$, which is assumed to be identical for all robots. Namely, each robot can only observe another robot' state if their relative distance is less than r. Thus, given $\{x_i(0), i \in \mathcal{N}\}$, we define the embedded graph $G_0(t) \triangleq (\mathcal{N}, E_0(t))$, where $(i, j) \in E_0(t)$ if $\|x_i(t) - x_j(t)\| < r$. We assume that $G_0(0)$ is connected initially and one of the control objectives is to ensure that $G_0(t)$ remains connected for all time $t \geq 0$.

Task specifications over services

Within the 2D workspace, each robot $i \in \mathcal{N}$ has a set of $M_i \geq 1$ regions of interest, denoted by $\Pi_i \triangleq \{\pi_{i1}, \cdots, \pi_{iM_i}\}$. These regions can be of different shapes, such as spheres, triangles, or polygons. For simplicity of presentation, $\pi_{i\ell} \in \Pi_i$ is here represented by a circular area around a point of interest: $\pi_{i\ell} = \mathcal{B}(c_{i\ell}, r_{i\ell}) = \{y \in \mathbb{R}^2 \mid \|y - c_{i\ell}\| \leq r_{i\ell}\}$, where $c_{i\ell} \in \mathbb{R}^2$ is the center; $r_{i\ell} \geq r_{\min}$ is the radius and $r_{\min} > 0$ is the minimal radius for all regions. Other shapes than spheres would require an under-approximation of these shapes as spheres first, in order to apply the proposed solution. We assume the following:

Assumption 6.1. *(I)* $\|c_{i\ell}\| < c_{\max}$, $\forall i \in \mathcal{N}$ *and* $\forall \pi_{i\ell} \in \Pi_i$, *where* $c_{\max} > 0$ *is a given constant. (II)* $\|c_{i\ell_i} - c_{j\ell_j}\| > 2\, r_{\min}$, $\forall i, j \in \mathcal{N}$, $\forall \pi_{i\ell_i} \in \Pi_i$ *and* $\forall \pi_{j\ell_j} \in \Pi_j$. ▲

Moreover, there is a set of atomic propositions known to robot i, denoted by AP_i. Each region of interest is associated with a subset of AP_i through the labeling function $L_i : \Pi_i \to 2^{AP_i}$. Without loss of generality, we assume that $AP_i \cap AP_j = \varnothing$, for all $i, j \in \mathcal{N}$ such that $i \neq j$. We view the atomic propositions $L_i(\pi_{i\ell})$ as a set of *services* that robot i can provide when being present in region $\pi_{i\ell} \in \Pi_i$. Hence, upon the visit to $\pi_{i\ell}$, the robot i chooses among $L_i(\pi_{i\ell})$ the subset of atomic propositions to be evaluated as true, i.e., the subset of services it *provides* among the available ones. These services are abstractions of action primitives that can be executed in different regions, such as manipulation tasks or data gathering. Some services within AP_i may depend on the other robots' collaborations, meaning that they can be provided only if the other robots are around.

We denote by $\mathbf{x}_i(T)$ the *trajectory* of robot i during the time interval $[0, T)$, where $T > 0$ and T can be infinity. The trajectory $\mathbf{x}_i(T)$ is associated with a unique finite or infinite sequence, called a *path*, $\mathbf{p}_i(T) \triangleq \pi_{i1}\pi_{i2}\ldots$ of regions in Π_i that robot i crosses, and with a finite or infinite sequence of time instants $t'_{i0}t_{i1}t'_{i1}t_{i2}t'_{i2}\cdots$ when i enters or leaves the respective regions. Formally, for all $k \geq 1$: $0 = t'_{i0} \leq t_{ik} \leq t'_{ik} < t_{ik+1} < T$, $x_i(t) \in \pi_{ik}$, for $\pi_{ik} \in \Pi_i$, $\forall t \in [t_{ik}, t'_{ik}]$, and $x_i(t) \notin \pi_{i\ell}$, $\forall \pi_{i\ell} \in \Pi_i$ and $\forall t \in (t'_{ik-1}, t_{ik})$. However, robot i may choose to provide services only at some regions along the path $\mathbf{p}_i$. Denote by $\overline{\mathbf{p}}_i(T) = \pi_{i\ell_1}\pi_{i\ell_2}\ldots$ the *effective path* as a subsequence of $\mathbf{p}_i$ such that $\ell_k < \ell_{k+1}$, $\forall k \geq 1$ and $\pi_{i\ell_k} \in \mathbf{p}_i(T)$, $\forall \pi_{i\ell_k} \in \overline{\mathbf{p}}_i(T)$. The *word* produced by robot i is given by the provided services along the sequence of regions in $\overline{\mathbf{p}}_i$. In particular, at region for $\pi_{i\ell_k} \in \overline{\mathbf{p}}_i(T)$, robot i chooses to provide a set of services w_{ℓ_k}, where $w_{\ell_k} \neq \varnothing$ and $w_{\ell_k} \subseteq L_i(\pi_{i\ell_k})$ is a subset of services available at region $\pi_{i\ell_k}$. Namely the produced word $\mathtt{trace}_i(T) = w_{\ell_1}w_{\ell_2}\cdots$ complies

with $\overline{\mathbf{p}}_i(T)$ if $\varnothing \subset w_{\ell_k} \subseteq L_i(\pi_{i\ell_k})$, $\forall \pi_{i\ell_k} \in \overline{\mathbf{p}}_i(T)$. Thus an robot's behavior is fully determined by its trajectory, its effective path and the word it produces.

The local task of each robot $i \in \mathcal{N}$ is specified as a general LTL or an sc-LTL formula φ_i over AP_i and captures requirements on the services to be provided by robot i. In this work, we do not focus on how the service providing is executed by an robot; we only aim at controlling an robot's motion to reach the regions where these services are available. Given the trajectory $\mathbf{x}_i(T)$ of robot i, the satisfaction of its task formula φ_i is defined as follows:

Definition 6.1. Robot i's trajectory $\mathbf{x}_i(T)$ satisfies φ_i if there exists an effective path $\overline{\mathbf{p}}_i(T)$ and a compliant word $\mathtt{word}_i(T)$ such that $\mathtt{word}_i(T) \vDash \varphi_i$. ▲

Cost of an effective path

Since we are interested in the quantitative cost of satisfying a local task, we propose the following way to measure the cost of an effective path. The motion of robot i within the workspace is estimated through a weighted transition system [11]: $\mathcal{T}_i \triangleq (\Pi_i', \longrightarrow_i, L_i, AP_i, \pi_{i,0}', W_i)$, where $\Pi_i' \triangleq \Pi_i \cup \{\pi_{i0}\}$. $\pi_{i0} \triangleq x_i(0)$ represents the robot's initial position symbolically; $\longrightarrow_i \triangleq \Pi_i' \times \Pi_i'$ is the transition relation, which is the full Cartesian product; L_i and AP_i are the labelling function and the set of propositions for services defined earlier; $\pi_{i,0}' \triangleq \pi_{i0}$ is the initial state; $W_i : \longrightarrow_i \to \mathbb{R}^+$ approximates the cost of each transition, $W_i(\pi_{i\ell_1}, \pi_{i\ell_2}) \triangleq \|c_{i\ell_1} - c_{i\ell_2}\| - r_{i\ell_1} - r_{i\ell_2}$, $\forall(\pi_{i\ell_1}, \pi_{i\ell_2}) \in \longrightarrow_i$, where $\pi_{i\ell_1}, \pi_{i\ell_2} \in \Pi_i$; and $W_i(\pi_{i0}, \pi_{i\ell}) \triangleq \|x_i(0) - c_{i\ell}\|$ to evaluate the distance from the robot's initial position to any region $\pi_{i\ell} \in \Pi_i$.

Consider that one of robot i's effective path is given by $\overline{\mathbf{p}}_i(T) = \pi_{i\ell_1}\pi_{i\ell_2}\ldots$. Then this effective path is assigned with a cost. In this work, the cost is defined as the maximal distance traveled between two consecutive regions along the path. Formally, denote by ϑ the set of consecutive regions in $\overline{\mathbf{p}}_i(T)$, i.e., $\vartheta = \{(\pi_{i0}', \pi_{\ell_1}), (\pi_{\ell_k}, \pi_{\ell_{k+1}}), \forall k = 1, 2, \ldots\}$. Its cost is defined as:

$$\boldsymbol{cost}(\overline{\mathbf{p}}_i(T)) \triangleq \max_{(\pi_s, \pi_g)\in\vartheta} \{W_i(\pi_s, \pi_g)\}, \tag{6.2}$$

which is still valid when $T = \infty$. Namely, we want to minimize the maximal distance travelled between two consecutive regions in the effective path. This is due to the consideration that for services-related specifications it is of great interest to ensure the frequency at which a service is provided. The standard cost definition as the cumulative cost can be used instead by incorporating the synthesis algorithms proposed in [56, 78]. Formally the problem we consider is stated below:

Problem 6.1. *Given a team of N robots defined above and their task specifications as in Section 6.1.1, design a distributed control law u_i, the associated effective path $\overline{\mathbf{p}}_i(T)$ and its corresponding word $\mathtt{word}_i(T)$, $\forall i \in \mathcal{N}$, such that for $T = \infty$: (1) $\mathtt{word}_i(T)$ satisfies φ_i; and (2) $\overline{\mathbf{p}}_i(T)$ has minimal cost by (6.2); and (3) $\|x_i(t) - x_j(t)\| < r$, $\forall(i, j) \in E_0(0)$, $\forall t \in [0, T)$.* ▲

Note that the task specifications φ_i, $\forall i \in \mathcal{N}$ need not be the same type among the robots, i.e., φ_i can be either a syntactically co-safe or a general LTL formula. More details can be found in Section 6.1.4.

The proposed solution consists of **four** layers: (i) an offline synthesis scheme for the initial discrete plan of each robot, i.e., the effective path of progressive goal regions and the sequence of services to be provided; (ii) a distributed continuous control scheme that guarantees that one of the robots reaches its progressive goal region in finite time while the relative-distance constraints are fulfilled at all time; (iii) a hybrid control layer that coordinates the discrete plan execution and the continuous control law switching, to ensure the satisfaction of each robot's local task; (iv) a real-time plan adaptation algorithm to improve each agent's discrete plan, given its updated position and its plan execution status.

6.1.2 Initial optimal plan synthesis

We aim to find an effective path for each robot $i \in \mathcal{N}$ such that (i) there exists a compliant word satisfying φ_i and (ii) the effective path is optimal for the cost function from (6.2). Let $\mathcal{A}_{\varphi_i}$ be the NBA associated with φ_i from Section 3.2, i.e., $\mathcal{A}_{\varphi_i} = (Q_i, 2^{AP_i}, \delta_i, Q_{i,0}, \mathcal{F}_i)$. Firstly, we build a product automaton $\mathcal{A}_{p,i} = \mathcal{T}_i \otimes \mathcal{A}_{\varphi_i} = (Q_{p,i}, 2^{AP_i}, \delta_{p,i}, Q_{p,i,0}, \mathcal{F}_{p,i}, W_{p,i})$ as described in Section 3.3.1, with a slight change in $\delta_{p,i}$ reflecting the form of the words compliant with the effective paths: $((s,q),\sigma,(s',q')) \in \delta_{p,i}$ if $(s,s') \in \longrightarrow$ and $q' \in \delta(q,\sigma)$, where $\varnothing \subset \sigma \subseteq L(s)$. An accepting run ϱ_i of $\mathcal{A}_{p,i}$ over an input word w_i projects onto a run τ_i of $\mathcal{T}_i$, such that w complies with the effective path $\overline{\mathbf{p}}_i(T)$ and satisfies φ_i, while for any word w_i that satisfies φ_i and is compliant with the path $\overline{\mathbf{p}}_i(T)$, there exists an accepting run of $\mathcal{A}_{p,i}$ that projects onto a run τ_i of $\mathcal{T}_i$ that is compliant with w.

In Algorithm 6.11, we modify the Dijkstra's algorithm (see, e.g., [97]) to find the finite paths from a state $v \in Q_{p,i}$ to all the other states minimizing the bottleneck weight, where the bottleneck weight of a path is defined as the maximal weight of the individual edges on the path. The output of the $\texttt{MinBot}(v)$ algorithm is the distance function $\texttt{D}_v : Q_{p,i} \to \mathbb{R}^+$, where $\texttt{D}_v(u)$ gives the minimal bottleneck weight from v to the state $u \in Q_{p,i}$, and the predecessor function $\texttt{P}_v : Q_{p,i} \to Q_{p,i}$, where $\texttt{P}_v(u)$ gives the predecessor of $u \in Q_{p,i}$ on the minimal-bottleneck path. Then we can synthesize the optimal effective path of $\mathcal{A}_{p,i}$ and the associated word as follows:

(I) For all $v_0 \in Q_{p,i,0}$, compute $(\texttt{D}_{v_0}, \texttt{P}_{v_0}) = \texttt{MinBot}(v_0)$; (II) For all $v_f \in F_{p,i}$, compute $(\texttt{D}_{v_f}, \texttt{P}_{v_f}) = \texttt{MinBot}(v_f)$; (III) Find the pair (v_0, v_f), where $v_0 \in Q_{p,i,0}$ and $v_f \in F_{p,i}$ that minimizes the term $\max\left\{\texttt{D}_{v_0}(v_f), \max_{(v,v_f)\in\delta_{p,i}}\{\texttt{D}_{v_f}(v), W_{p,i}(v,v_f)\}\right\}$, where $\texttt{D}_{v_0}(v_f)$ is the minimal bottleneck from v_0 to v_f; and the second term is the minimal bottleneck from v_f back to itself. Note that v is any predecessor of v_f given by $\delta_{p,i}$. Denote the optimal pair as $(v_0^\star, v_f^\star)$; (IV) The computed accepting run of $\mathcal{A}_{p,i}$ is in the prefix-suffix form $\varrho_i = \varrho_{i,\text{pre}}\,(\varrho_{i,\text{suf}})^\omega$, where $\varrho_{i,\text{pre}}$ is the minimal-bottleneck path from $v_0^\star$ to $v_f^\star$, computed based on $\texttt{P}_{v_0^\star}$; $\varrho_{i,\text{suf}}$ is the minimal-bottleneck cycle from $v_f^\star$ back to itself, computed similarly based on $\texttt{P}_{v_f^\star}$.

Algorithm 6.11 Minimum bottleneck path, $\texttt{MinBot}(v)$

Input: Product automaton $\mathcal{A}_{p,i}$ and $v \in Q_{p,i}$
Output: $\mathtt{D}_v, \mathtt{P}_v$
$S \coloneqq Q_{p,i}$; $\mathtt{D}_v(v) \coloneqq 0$
forall the $u \in Q_{p,i}$ **do**
  if $u \neq v$ **then**
    $\mathtt{D}_v(u) \coloneqq \infty$; $\mathtt{P}_v(u) \coloneqq None$
while $S \neq \varnothing$ **do**
  $u \coloneqq \operatorname{argmin}_{u \in S}\{\mathtt{D}_v(u)\}$
  remove u from S
  forall the $u' \in \delta_{p,i}(u)$ **do**
    $b \coloneqq \max\{\mathtt{D}_v(u), W_{p,i}(u,u')\}$
    if $b \leq \mathtt{D}_v(u')$ **then**
      $\mathtt{D}_v(u') \coloneqq b$; $\mathtt{P}_v(u') \coloneqq u$
return $\mathtt{D}_v, \mathtt{P}_v$

The accepting run ϱ_i of $\mathcal{A}_{p,i}$ is naturally projected onto $\mathcal{T}_i$ as follows, which results into the initial plan: $\tau_i(0) = \tau_{i,\mathrm{pre}}(\tau_{i,\mathrm{suf}})^\omega$, where $\tau_{i,\mathrm{pre}} = (\pi_{i1}, w_{i1})\cdots(\pi_{ik_i}, w_{ik_i})$ is the plan prefix, and $\tau_{i,\mathrm{suf}} = (\pi_{ik_i+1}, w_{ik_i+1})\cdots(\pi_{iK_i}, w_{iK_i})$ is the periodical plan suffix; (π_{ik}, w_{ik}) is called the *progressive goal region*; $\pi_{ik} \in \Pi_i$ and $\varnothing \subset w_{ik} \subseteq L_i(\pi_{ik})$, $\forall k = 1, \cdots, K_i$. Thus the word corresponding to $\tau_i(0)$ is given by its projection onto AP_i, namely $\texttt{word}_i(T) = \tau_i(0)|_{AP_i} = w_{i1}\cdots w_{ik_i}(w_{ik_i+1}\cdots w_{iK_i})^\omega$; then the effective path $\overline{\mathbf{p}}_i$ is given as the projection of $\tau_i(0)$ onto Π_i, namely $\overline{\mathbf{p}}_i(T) = \tau_i(0)|_{\Pi_i} = \pi_{i1}\cdots\pi_{ik_i}(\pi_{ik_i+1}\cdots\pi_{iK_i})^\omega$. We denote by $\overline{\mathbf{p}}_{i,\mathrm{pre}}(T) = \pi_{i1}\cdots\pi_{ik_i}$ the prefix of the effective path and by $\overline{\mathbf{p}}_{i,\mathrm{suf}}(T) = \pi_{ik_i+1}\cdots\pi_{iK_i}$ the suffix. For general LTL formulas φ_i, the plan τ_i indicates the desired effective path $\overline{\mathbf{p}}_i(T)$ and the infinite sequence of services $\texttt{word}_i(T)$; for sc-LTL formulas φ_i, $\tau_{i,\mathrm{pre}}$ indicates the desired effective path $\overline{\mathbf{p}}_i(T)$ and the finite sequence of services $\texttt{word}_i(T)$. Moreover, any trajectory that satisfies the prefix with an arbitrary extension would satisfy φ_i. The computational complexity [89, 97] of the above algorithm is $\mathcal{O}(|\delta_{p,i}| \cdot \log|Q_{p,i}| \cdot |Q_{p,i,0}| \cdot |\mathcal{F}_{p,i}|)$ in the worst case, where $|Q_{p,i}|$, $|\delta_{p,i}|$ are the number of states and transitions in $\mathcal{A}_{p,i}$.

6.1.3 Continuous controller design

As stated previously, each robot synthesizes its initial plan as a sequence of goal regions to reach and a set of services to provide there. However, these goal regions of different robots can be at different locations and potentially far away. Furthermore, the relative-distance constraints require the neighboring robots to stay close. The state-of-the-art motion control technique for multi-robot systems under relative-distance constraints can only handle a single leader [122] or multiple robots with a global objective, e.g., formation [117, 157]. Here we propose a distributed motion

control scheme that allows an *arbitrary* number of active robots with individual goals and guarantees almost global convergence to one active robot's goal, while ensuring the relative-distance constraints at all time.

Before stating the control scheme, let us first introduce the notion of connectivity graph, which allows us to handle the relative-distance constraints. Recall that each robot has a limited sensing radius $r > 0$ as mentioned in Section 6.1.1. Let $\kappa \in (0, r)$ be a given constant. Then we define the connectivity graph $G(t)$ as follows:

Definition 6.2. Let $G(t) \triangleq (\mathcal{N}, E(t))$ denote the undirected time-varying connectivity graph at time $t \geq 0$, where $E(t) \subseteq \mathcal{N} \times \mathcal{N}$ is the set of edges. (I) $G(0) = G_0(0)$; (II) At time $t > 0$, $(i, j) \in E(t)$ iff one of the following conditions hold: (i) $\|x_i(t) - x_j(t)\| \leq r - \kappa$; or (ii) $r - \kappa < \|x_i(t) - x_j(t)\| \leq r$ and $(i, j) \in E(t^-)$, where $t^- < t$ and $|t - t^-| \to 0$. ▲

Note that the condition (II) guarantees that a new edge will only be added when the distance between two previously-unconnected robots decreases below $r - \kappa$. Namely, there is a hysteresis effect when adding new edges to the connectivity graph. Consequently, each robot $i \in \mathcal{N}$ has a time-varying set of neighbors $\mathcal{N}_i(t) = \{j \in \mathcal{N} \,|\, (i, j) \in E(t)\}$. Let the progressive goal region of robot $i \in \mathcal{N}$ at time t be given by $\pi_{ig} = \mathcal{B}(c_{ig}, r_{ig}) \in \Pi_i$. We propose the following two different control modes:
(1) the ***active*** mode:

$$\mathbf{C}_{act}: \quad u_i(t) \triangleq -d_i\, p_i - \sum_{j \in \mathcal{N}_i(t)} h_{ij}\, x_{ij}, \tag{6.3}$$

(2) the ***passive*** mode:

$$\mathbf{C}_{pas}: \quad u_i(t) \triangleq - \sum_{j \in \mathcal{N}_i(t)} h_{ij}\, x_{ij}, \tag{6.4}$$

where $x_{ij} \triangleq x_i - x_j$; $p_i \triangleq x_i - c_{ig}$; and the coefficients are

$$d_i \triangleq \frac{\varepsilon^3}{(\|p_i\|^2 + \varepsilon)^2} + \frac{\varepsilon^2}{2\,(\|p_i\|^2 + \varepsilon)};\; h_{ij} \triangleq \frac{r^2}{(r^2 - \|x_{ij}\|^2)^2}, \tag{6.5}$$

where $\varepsilon > 0$ is a *key* design parameter to be appropriately tuned. We show in detail how to choose ε in the sequel. Both controllers in (6.3) and (6.4) are nonlinear and rely on only locally-available states, particularly, $x_i(t)$ and $x_j(t), j \in \mathcal{N}_i(t)$.

Assume that $G(T_s)$ is connected at time $T_s > 0$. Moreover, assume that there are $1 \leq N_a \leq N$ robots that are in the *active* mode obeying (6.3) with its goal region as $\pi_{ig} = \mathcal{B}(c_{ig}, r_{ig}) \in \Pi_i$; and the rest $N_p = N - N_a$ robots that are in the *passive* mode obeying (6.4). For simplicity, denote by the group of active and passive robots $\mathcal{N}_a, \mathcal{N}_p \subseteq \mathcal{N}$, respectively. In the rest of this section, we show that under *arbitrary* number of active robots, by following the control laws (6.3) and (6.4), exactly *one* active robot can reach its goal region within finite time $T_f \in (T_s, +\infty)$, while the relative distance $\|x_i(t) - x_j(t)\| < r$, $\forall (i, j) \in E(T_s)$ and $\forall t \in [T_s, T_f]$.

Relative-distance maintenance

In this part, we show that the relative-distance constraints are always satisfied under the control laws (6.3) and (6.4). We consider the potential-field function below:

$$V(t) \triangleq \frac{1}{2} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}_i(t)} \phi_c(x_{ij}) + b_i \sum_{i\in\mathcal{N}} \phi_g(x_i) \tag{6.6}$$

where $\phi_c(\cdot)$ is an attractive potential to robot i's neighbors:

$$\phi_c(x_{ij}) \triangleq \frac{1}{2} \frac{\|x_{ij}\|^2}{r^2 - \|x_{ij}\|^2}, \qquad \|x_{ij}\| \in [0,\, r-\delta); \tag{6.7}$$

while $\phi_g(\cdot)$ is an attractive force to robot i's goal, defined by:

$$\phi_g(x_i) \triangleq \frac{\varepsilon^2}{2} \frac{\|p_i\|^2}{\|p_i\|^2 + \varepsilon} + \frac{\varepsilon^2}{4} \ln(\|p_i\|^2 + \varepsilon), \tag{6.8}$$

where function $\ln(\cdot)$ is the natural logarithm; $b_i \in \mathbb{B}$ indicates the robot i's control mode. Namely, $b_i = 1$, $\forall i \in \mathcal{N}_a$ and $b_i = 0$, $\forall i \in \mathcal{N}_p$. Clearly $V(t)$ is lower-bounded by $N_a \ln(\varepsilon)\varepsilon^2/4$. Moreover, it can be verified that

$$\nabla_{x_i} V = \frac{\partial V}{\partial x_i} = b_i\, d_i\, p_i + \sum_{j\in\mathcal{N}_i(t)} h_{ij}\, x_{ij} = -u_i. \tag{6.9}$$

Theorem 6.1. *$G(t)$ remains connected and no existing edges within $E(T_s)$ will be lost, namely $E(T_s) \subseteq E(t)$, $\forall t \geq T_s$.*

Proof. Assume that the network $G(t)$ remains *invariant* during the time period $[t_1, t_2) \subseteq [T_s, \infty)$, i.e., no edges are added or removed. Thus the neighboring sets $\{\mathcal{N}_i,\, i \in \mathcal{N}\}$ also remain invariant and $V(t)$ is differentiable for $t \in [t_1, t_2)$. Then the time derivative of $V(t)$ is given by

$$\dot{V}(t) = \sum_{i\in\mathcal{N}} (\nabla_{x_i} V)^T u_i = -\sum_{i\in\mathcal{N}} \Big\| b_i\, d_i\, p_i + \sum_{j\in\mathcal{N}_i(t)} h_{ij}\, x_{ij} \Big\|^2 \leq 0, \tag{6.10}$$

meaning that $V(t)$ is non-increasing, $\forall t \in [t_1, t_2)$. Thus $V(t) \leq V(T_s) < +\infty$ for $t \in [t_1, t_2)$. On the other hand, assume a *new* edge (p, q) is added to $G(t)$ at $t = t_2$, where $p, q \in \mathcal{N}$. By Definition 6.2, $\|x_{pq}(t_2)\| \leq r - \kappa$ and $\phi_c(x_{pq}(t_2)) = \frac{(r-\kappa)^2}{\kappa(2r-\kappa)} < +\infty$ since $0 < \kappa < r$. Denote by $\widehat{E} \subset \mathcal{N}\times\mathcal{N}$ the set of newly-added edges at $t = t_2$. Let $V(t_2^+)$ and $V(t_2^-)$ be the value of $V(t)$ before and after adding the set of new edges to $G(t)$ at $t = t_2$. We get $V(t_2^+) = V(t_2^-) + \sum_{(p,q)\in\widehat{E}} \phi_c(x_{pq}(t_2)) \leq V(t_2^-) + |\widehat{E}|\, \frac{(r-\kappa)^2}{\kappa(2r-\kappa)} < +\infty$, where we use the fact that $|\widehat{E}|$ is bounded as $\widehat{E} \subset \mathcal{N} \times \mathcal{N}$. Thus $V(t) < +\infty$ also holds when new edges are added at time t_2. Similar analysis can be found in [76]. As a result, $V(t) < +\infty$ for $t \in [T_s, \infty)$. Note that $V(t)$ is non-increasing when $G(t)$

remains unchanged and increases when new edges are added. By Definition 6.2, one existing edge $(i, j) \in E(t)$ will be lost only if $x_{ij}(t) = r$. It implies that $\phi_c(x_{ij}) \to +\infty$ and $V(t) \to +\infty$ by (6.6). By contradiction, we can conclude that new edges might be added into $E(t)$ but no existing edges within $E(t)$ will be lost, namely $E(T_s) \subseteq E(t)$, $\forall t \geq T_s$. If $G(T_s)$ is connected, then $G(t)$ remains connected for all $t \geq T_s$. ■

Convergence analysis

In this part, we analyze in detail the convergence properties of the closed-loop system, i.e., the multi-robot system under the control laws (6.3) and (6.4) for *any* number of active and passive robots. We have shown that the potential function $V(t)$ is lower-bounded and non-increasing when $G(t)$ remains invariant by Theorem 6.1 above. Since no existing edges can be lost and the number of robots is finite, we first show that the graph $G(t)$ becomes complete and thus invariant afterwards when the system converges to the set of critical points defined in the sequel. By LaSalle's invariance principle [82] we only need to find out the largest invariant set within the set $\{x_i, \forall i \in \mathcal{N} \,|\, \dot{V}(t) = 0\}$. By enforcing $\dot{V}(t) = 0$, it implies:

$$b_i\, d_i\, p_i + \sum_{j \in \mathcal{N}_i(t)} h_{ij}\, x_{ij} = 0, \quad \forall i \in \mathcal{N}. \tag{6.11}$$

Then we can construct one $N \times N$ diagonal matrix $\mathbf{D}$ that $\mathbf{D}(i,i) = b_i\, d_i$, $\forall i \in \mathcal{N}$ and $\mathbf{D}(i,j) = 0$, $i \neq j$ and $i, j \in \mathcal{N}$. and another $N \times N$ matrix $\mathbf{H}$ that $\mathbf{H}(i,i) = \sum_{j \in \mathcal{N}_i} h_{ij}$, $\forall i \in \mathcal{N}$ and $\mathbf{H}(i,j) = -h_{ij}$, $i \neq j$ and $\forall (i, j) \in E(t)$ while $\mathbf{H}(i,j) = 0$, $\forall (i, j) \notin E(t)$. Note that $h_{ij} > 0$ as $\|x_{ij}\| \in [0, r)$ by (6.9), $\forall (i, j) \in E(t)$. As a result, $\mathbf{H}$ is the Laplacian matrix of the graph $G(t) = (\mathcal{N}, E(t), h)$, where $h(i, j) = h_{ij}$, $\forall (i, j) \in E(t)$. Then (6.11) is equivalent to:

$$\mathbf{H} \otimes \mathbf{I}_2 \cdot \mathbf{x} + \mathbf{D} \otimes \mathbf{I}_2 \cdot (\mathbf{x} - \mathbf{c}) = 0, \tag{6.12}$$

where $\otimes$ is the Kronecker product [73]; $\mathbf{x}$ is the stack vector for x_i, $i \in \mathcal{N}$ and $\mathbf{x}[i] = x_i$; $\mathbf{I}_2$ is the 2×2 identity matrix; $\mathbf{c}$ is the stack vector for c_{ig} and $\mathbf{c}[i] = c_{ig}$ if $i \in \mathcal{N}_a$ and $\mathbf{c}[i] = \mathbf{0}_2$ if $i \in \mathcal{N}_p$, where $\mathbf{0}_2$ is a 2×1 zero vector. Let $\mathcal{C}$ be the set of critical points of $V(t)$ that satisfy (6.12), i.e., $\mathcal{C} \triangleq \{x \in \mathbb{R}^{2N} \,|\, \mathbf{H} \otimes \mathbf{I}_2 \cdot \mathbf{x} + \mathbf{D} \otimes \mathbf{I}_2 \cdot (\mathbf{x} - \mathbf{c}) = 0\}$. Now we show that at the critical points within $\mathcal{C}$ the relative distances between any two robots can be made arbitrarily small by reducing ε and as a result the underlying network becomes a complete graph.

Lemma 6.2. *For all critical points $\mathbf{x}_c \in \mathcal{C}$, (I) $\|x_{ij}\|$ can be made arbitrarily small by reducing ε, $\forall (i, j) \in E(t)$; (II) there exists $\varepsilon_0 > 0$ that if $\varepsilon < \varepsilon_0$, then the connectivity graph $G(t)$ is complete.*

Proof. (I) For a critical point $\mathbf{x}_c \in \mathcal{C}$, $\sum_{(i,j) \in E(t)} h_{ij} \|x_{ij}\|^2 = \mathbf{x}_c^T \cdot (\mathbf{H} \otimes \mathbf{I}_2) \cdot \mathbf{x}_c$. holds.

Combining it with (6.12), we get

$$\begin{aligned}\sum_{(i,j)\in E(t)} h_{ij}\|x_{ij}\|^2 &= -\mathbf{x}_c^T\cdot(\mathbf{D}\otimes\mathbf{I}_2)\cdot(\mathbf{x}_c-\mathbf{c})\\ &= -(\mathbf{x}_c-\mathbf{c})^T\cdot(\mathbf{D}\otimes\mathbf{I}_2)\cdot(\mathbf{x}_c-\mathbf{c}) - \mathbf{c}^T\cdot(\mathbf{D}\otimes\mathbf{I}_2)\cdot(\mathbf{x}_c-\mathbf{c})\\ &= -\sum_{i\in\mathcal{N}} b_i\, d_i\left(\|p_i\|^2 + c_{i\ell}^T p_i\right) \le \sum_{i\in\mathcal{N}} b_i\,\|c_{i\ell}\|\, d_i\|p_i\|.\end{aligned}$$

Since it can be verified that $d_i\,\|p_i\| < \varepsilon\sqrt{\varepsilon}$ for $\|p_i\| \ge 0$ and $\|c_{i\ell}\| < c_{\max}$ is given in Assumption 6.1, we get $\sum_{(i,j)\in E(t)} h_{ij}\|x_{ij}\|^2 < N_a\, c_{\max}\,\varepsilon\sqrt{\varepsilon} \le N\, c_{\max}\,\varepsilon\sqrt{\varepsilon}$, where we use the fact that $b_i = 0$, for $i \in \mathcal{N}_p$ and $\mathcal{N}_a \le N$. Thus $\forall(i,j)\in E(t)$, it holds that $h_{ij}\|x_{ij}\|^2 < N\, c_{\max}\,\varepsilon\sqrt{\varepsilon} \triangleq \varsigma$. It can be verified that $h_{ij}\|x_{ij}\|^2$ is monotonically increasing as a function of $\|x_{ij}\|$. This implies that $\forall(i,j)\in E(t)$, $\|x_{ij}\|^2 \le r^2\,\varsigma$, or equivalently $\|x_{ij}\|^2 \le \varepsilon\sqrt{\varepsilon}\,\xi$, where $\xi \triangleq r^2\, N\, c_{\max}$. Thus $\|x_{ij}\|$ can be made arbitrarily small by reducing ε. (II) Moreover, let ε_0 satisfy the condition

$$(N-1)\sqrt{\varepsilon_0\sqrt{\varepsilon_0}\,\xi} < r-\delta. \tag{6.13}$$

If $\varepsilon < \varepsilon_0$, then for *any* pair $(p,q)\in\mathcal{N}\times\mathcal{N}$, $\|x_{pq}\|$ satisfies $\|x_{pq}\| = |x_p - x_1 + x_1 - x_2 + \cdots - x_q| \le (N-1)\sqrt{\varepsilon\sqrt{\varepsilon}\,\xi} < r-\delta$, because there exists a path in $G(t)$ of maximal length N from any node $p\in\mathcal{N}$ to another node q as $G(t)$ remains connected for $t > T_s$ by Theorem 6.1; and $\|x_{ij}\| \le \varepsilon\sqrt{\varepsilon}\,\xi$ from above, $\forall(i,j)\in E(t)$. By Definition 6.2 this implies $(p,q)\in E(t)$. Thus $G(t)$ is a complete graph when $x(t)\in\mathcal{C}$. ■

Namely, at the critical points, the graph $G(t)$ is complete and thus remains invariant afterwards. Before stating the convergence property, we need to define the following set for each active robot $i\in\mathcal{N}_a$:

$$\mathcal{S}_i \triangleq \{\mathbf{x}\in\mathbb{R}^{2N} \mid \|\mathbf{x} - \mathbf{1}_N\otimes c_{ig}\| \le r_S(\varepsilon)\}, \tag{6.14}$$

where $r_S(\varepsilon) \triangleq \sqrt{3N\,\varepsilon} + \sqrt{(N-1)\varepsilon\sqrt{\varepsilon}\,\xi}$. Loosely speaking, $\mathcal{S}_i$ represents the neighbourhood around the goal region center of an active robot $i\in\mathcal{N}_a$. Furthermore, let $\mathcal{S} \triangleq \cup_{i\in\mathcal{N}_a}\mathcal{S}_i$ and $\mathcal{S}^{\neg} \triangleq \mathbb{R}^{2N}\setminus\mathcal{S}$. In the following, we analyze the properties of the critical points of $V(t)$ within the regions $\mathcal{S}$ and $\mathcal{S}^{\neg}$. More specifically: by Lemma 6.3 there are no local minima but saddle points within $\mathcal{S}^{\neg}$; by Lemma 6.4 these saddle points are non-degenerate; by Lemmas 6.5-6.7 all critical points within $\mathcal{S}$ are local minima. To explore these properties, we compute the second partial derivatives of $V(t)$ with respect to x_i, which are given by

$$\frac{\partial^2 V}{\partial x_i\partial x_i} = b_i\, d_i\otimes\mathbf{I}_2 + b_i\, d_i'\, p_i\cdot p_i^T + \sum_{j\in\mathcal{N}_i(t)}\left(h_{ij}\otimes\mathbf{I}_2 + h_{ij}'\, x_{ij}\cdot x_{ij}^T\right), \tag{6.15}$$

$$\frac{\partial^2 V}{\partial x_i\partial x_j} = -h_{ij}\otimes\mathbf{I}_2 - h_{ij}'\, x_{ij}\cdot x_{ij}^T, \qquad \forall j\neq i, \tag{6.16}$$

where

$$d_i' = \frac{-4\,\varepsilon^3}{(\|p_i\|^2+\varepsilon)^3} + \frac{-\varepsilon^2}{(\|p_i\|^2+\varepsilon)^2},\ \text{and}\ h_{ij}' = \frac{4\,r^2}{(r^2-\|x_{ij}\|^2)^3}.$$

Lemma 6.3. *There are no local minima of V within $\mathcal{S}^{\neg}$.*

Proof. We prove this by showing that if a critical point $\mathbf{x}_c \in \mathcal{S}^{\neg}$ there always exists a direction $\mathbf{z} \in \mathbb{R}^{2N}$ at $\mathbf{x}_c$ such that the quadratic form $\mathbf{z}^T \nabla^2 V \mathbf{z}$ is negative semi-definite. Given a critical point $\mathbf{x}_c \in \mathcal{C}$ and $\mathbf{x}_c \in \mathcal{S}^{\neg}$, then by definition $\|\mathbf{x} - \mathbf{1}_N \otimes c_{i\ell}\| > r_S(\varepsilon)$, $\forall i \in \mathcal{N}_a$. Besides, for any $i \in \mathcal{N}_a$, we can bound $\|\mathbf{x} - \mathbf{1}_N \otimes c_{i\ell}\|$ as follows:

$$\begin{aligned}
&\|\mathbf{x} - \mathbf{1}_N \otimes c_{i\ell}\| = \|\mathbf{x} - \mathbf{1}_N \otimes x_i + \mathbf{1}_N \otimes (x_i - c_{ig})\| \\
&\leq \sqrt{\sum_{j \in \mathcal{N}} \|x_{ij}\|^2} + \sqrt{N}\,\|p_i\| \leq \sqrt{(N-1)\varepsilon\sqrt{\varepsilon}\,\xi} + \sqrt{N}\,\|p_i\|,
\end{aligned}$$

where we use the fact that $\|x_{ij}\|^2 \leq \varepsilon\sqrt{\varepsilon}\,\xi$ at $\mathbf{x}_c$, $\forall (i,j) \in E(t)$ by Lemma 6.2. By comparing it with $r_S(\varepsilon)$ which is the lower bound, we get $\|p_i\| \geq \sqrt{3\,\varepsilon}$, $\forall i \in \mathcal{N}_a$. Choose $\mathbf{z} \triangleq \mathbf{1}_N \otimes z$, where $z \in \mathbb{R}^2$ and $\|z\| \triangleq 1$. Then $\mathbf{z}^T \nabla^2 V \mathbf{z}$ is evaluated by using (6.15)-(6.16): $\mathbf{z}^T \nabla^2 V \mathbf{z} = \sum_{i \in \mathcal{N}} b_i\, d_i\, z^T z + b_i\, d_i'\, z^T p_i\, p_i^T z \triangleq z^T M z$, where $M \triangleq \sum_{i \in \mathcal{N}_a} (d_i \otimes \mathbf{I}_2 + d_i'\, p_i\, p_i^T)$ is a 2×2 Hermitian matrix. The trace of M is computed as

$$\mathbf{trace}(M) = \sum_{i \in \mathcal{N}_a} 2\,d_i + d_i'\,\|p_i\|^2 = \varepsilon^3 \sum_{i \in \mathcal{N}_a} \frac{3\varepsilon - \|p_i\|^2}{(\|p_i\|^2 + \varepsilon)^3} < 0,$$

as we have shown that $\|p_i\| \geq \sqrt{3\varepsilon}$ above, $\forall i \in \mathcal{N}_a$ if $\mathbf{x}_c \in \mathcal{S}^{\neg}$. On the other hand, denote by $p_i = [p_{i,x},\, p_{i,y}]$ the coordinates of p_i. The determinant of M is given by

$$\begin{aligned}
\mathbf{det}(M) &= -\Big(\sum_{i \in \mathcal{N}_a} d_i'\, p_{i,x}\, p_{i,y}\Big)^2 + \Big(\sum_{i \in \mathcal{N}_a} d_i + d_i'\, p_{i,x}^2\Big)\Big(\sum_{i \in \mathcal{N}_a} d_i + d_i'\, p_{i,y}^2\Big) \\
&\geq \frac{1}{2} \sum_{i,\, j \in \mathcal{N}_a} \Big[(d_i + d_i' \|p_i\|^2)(d_j + d_j' \|p_j\|^2)\Big] > 0,
\end{aligned}$$

since $d_i' \|p_i\|^2 < -d_i$ for $\|p_i\| > \sqrt{3\varepsilon}$, $\forall i \in \mathcal{N}_a$; and $(p_{i,x} p_{i,y} - p_{j,x} p_{j,y})^2 \leq \|p_i\|^2 \|p_j\|^2$ by Cauchy-Schwarz inequality [73]. Denote by λ_1 and λ_2 the eigenvalues of M, where $\lambda_1, \lambda_2 \in \mathbb{R}$ as M is Hermitian. Since $\mathbf{trace}(M) < 0$ and $\mathbf{det}(M) > 0$, then M is negative definite and both eigenvalues are negative [73], i.e., $\lambda_1, \lambda_2 < 0$. Thus for any vector $z \in \mathbb{R}^2$, $z^T M z < 0$. Namely, for any vector $\mathbf{z} = \mathbf{1}_N \otimes z$ where $z \in \mathbb{R}^2$, $\mathbf{z}^T \nabla^2 V \mathbf{z} < 0$. To conclude, for any $\mathbf{x}_c \in \mathcal{C}$, if $\mathbf{x}_c \in \mathcal{S}^{\neg}$ then $\mathbf{x}_c$ is not a local minimum. ∎

Lemma 6.4. *There exists $\varepsilon_1 > 0$ such that if $\varepsilon < \varepsilon_1$, all critical points of V in $\mathcal{S}^{\neg}$ are non-degenerate saddle points.*

Proof. To show that V is Morse we use Lemma 3.8 from [126], which states that the non-singularity of a linear operator follows from the fact that its associated quadratic form is sign definite on complementary subspaces.

Let $\mathcal{Q} = \{v \in \mathbb{R}^{2N} \,|\, v = \mathbf{1}_N \otimes z,\ z \in \mathbb{R}^2\}$. In Lemma 6.3, we have shown that for any vector $v \in \mathcal{Q}$, $v^T \nabla^2 V v < 0$. Let $\mathcal{P} = \{v \in \mathbb{R}^{2N} \,|\, v = \mathbf{e}_N \otimes z,\ \mathbf{e}_N \perp \mathbf{1}_N,\ \mathbf{e}_N \in \mathbb{R}^N,\ z \in \mathbb{R}^2\}$. Firstly, it can be easily verified that $\mathcal{P}$ is the orthogonal complement of $\mathcal{Q}$. In the

following, we show that $\nabla^2 V$ is positive definite in $\mathcal{P}$. Let $\mathbf{z} \in \mathcal{P}$, i.e., $\mathbf{z} \triangleq \mathbf{e}_N \otimes z \triangleq [z_1^T, z_2^T, \cdots, z_n^T]^T$, where $z \in \mathbb{R}^2$, $\mathbf{e}_N \in \mathbb{R}^N$, $\mathbf{e}_N^T \perp \mathbf{1}_N$, $z_i \in \mathbb{R}^2$, $\forall i \in \mathcal{N}$. The quadratic form $\mathbf{z}^T \nabla^2 V \mathbf{z}$ at $\mathbf{x}_c$ can be computed explicitly using (6.15)-(6.16):

$$\begin{aligned}
\mathbf{z}^T \nabla^2 V \mathbf{z} &= \sum_{i \in \mathcal{N}_a} \left(d_i \|z_i\|^2 + d_i' |p_i^T z_i|^2 \right) \\
&+ \sum_{(i,j) \in E(t)} \left(h_{ij} \|z_i - z_j\|^2 + 2 h_{ij}' |(x_i - x_j)^T (z_i - z_j)|^2 \right) \\
&\geq \sum_{i \in \mathcal{N}_a} \left(d_i \|z_i\|^2 + d_i' |p_i^T z_i|^2 \right) + \sum_{(i,j) \in E(t)} h_{ij} \|z_i - z_j\|^2 \\
&\geq \sum_{i \in \mathcal{N}_a} \left(d_i + d_i' \|p_i\|^2 \right) \|z_i\|^2 + \mathbf{z}^T (\mathbf{H} \otimes \mathbf{I}_2) \mathbf{z},
\end{aligned}$$

where we use the fact that $h_{ij}' > 0$, $d_i' < 0$ and $|p_i^T z_i| \leq \|p_i\| \|z_i\|$. It can be verified that $d_i + d_i' \|p_i\|^2 > -0.1\varepsilon$ for $\|p_i\| \geq \sqrt{3\varepsilon}$, $\forall i \in \mathcal{N}_a$. Moreover, the second term can be lower-bounded by $\mathbf{z}^T (\mathbf{H} \otimes \mathbf{I}_2) \mathbf{z} = (\mathbf{e}_N \otimes z)^T \cdot (\mathbf{H} \otimes \mathbf{I}_2) \cdot (\mathbf{e}_N \otimes z) = (\mathbf{e}_N^T \cdot \mathbf{H} \cdot \mathbf{e}_N) \|z\|^2 \geq \lambda_2(\mathbf{H}) \|\mathbf{z}\|^2$, where we apply the Courant-Fischer Theorem [73]: $\min_{\mathbf{e}_N \perp \mathbf{1}_N} \{\mathbf{e}_N^T \cdot \mathbf{H} \cdot \mathbf{e}_N\} = \lambda_2(\mathbf{H}) \|\mathbf{e}_N\|^2 > 0$, since $\mathbf{H}$ is the Laplacian matrix defined in (6.12), which is positive semidefinite with $\lambda_1(\mathbf{H}) = 0$, of which the corresponding eigenvector is $\mathbf{1}_N$; and the second smallest eigenvalue $\lambda_2(\mathbf{H}) > 0$. In addition, since $h_{ij} > 1/r^2$ and $G(t)$ is a complete graph at $\mathbf{x}_c$ by Lemma 6.2, it holds that $\lambda_2(\mathbf{H}) > N/r^2$ by [45]. This implies that $\mathbf{z}^T \nabla^2 V \mathbf{z} \geq \sum_{i \in \mathcal{N}_a} \left(\frac{N}{r^2} + d_i + d_i' \|p_i\|^2 \right) \|z_i\|^2 \geq \sum_{i \in \mathcal{N}_a} \left(\frac{N}{r^2} - 0.1\varepsilon \right) \|z_i\|^2$. Thus if $\varepsilon < N/(0.1r^2)$, it holds that the quadratic form $\mathbf{z}^T \nabla^2 V \mathbf{z} > 0$, $\forall \mathbf{z} = \mathbf{e}_N \otimes z$ where $\mathbf{e}_N \perp \mathbf{1}_N$, $z \in \mathbb{R}^2$. To conclude, $\nabla^2 V|_{\mathcal{Q}}$ is negative definite by Lemma 6.3 and $\nabla^2 V|_{\mathcal{P}}$ is positive definite, given that ε satisfies the conditions below:

$$\varepsilon < \min\{\varepsilon_0, \frac{N}{0.1r^2}\} \triangleq \varepsilon_1. \tag{6.17}$$

By applying Lemma 3.8 from [126], $\nabla^2 V$ is non-singular at the saddle points $\mathbf{x}_c \in \mathcal{S}^{\neg}$. Thus all critical points within $\mathcal{S}^{\neg}$ are non-degenerate saddle points if $\varepsilon < \varepsilon_1$. ■

Now we focus on showing that all critical points within $\mathcal{S}$ are local minima. First we need the following two lemmas stating that when the system is at a critical point belonging to $\mathcal{S}_i$ of any active robot $i \in \mathcal{N}_a$, then all the other robots are within this goal region π_{ig} and away from their own goal region center by at least distance $r_{\min}$.

Lemma 6.5. *There exists $\varepsilon_2 > 0$ that if $\varepsilon < \varepsilon_2$, the following statements hold: (I) $\mathcal{S}_i \cap \mathcal{S}_j = \varnothing$, $\forall i \neq j$ and $i, j \in \mathcal{N}_a$; (II) If $\mathbf{x}_c \in \mathcal{S}_i$ for any $i \in \mathcal{N}_a$, then $x_j \in \pi_{ig}$, $\forall j \in \mathcal{N}$ and $\|x_j - c_{jg}\| > r_{\min}$, $j \neq i$, $\forall j \in \mathcal{N}_a$.*

Proof. Let ε_2 be given as the solution of

$$r_S(\varepsilon_2) \triangleq \sqrt{3N\,\varepsilon_2} + \sqrt{(N-1)\varepsilon_2 \sqrt{\varepsilon_2}\,\xi} \triangleq r_{\min}, \tag{6.18}$$

where $r_{\min}$ is given in Assumption 6.1. Note that ε_2 is unique as the left-hand side is a function of ε_2 that monotonically increases and has the range $[0, \infty)$. Assume

that $\mathbf{x}_c \in \mathcal{S}_{i^\star}$ for some $i^\star \in \mathcal{N}_a$, i.e., $\|\mathbf{x}_c - \mathbf{1}_N \otimes c_{i^\star \mathrm{g}}\| \le r_S(\varepsilon_2)$. Then $\forall j \neq i^\star$, $j \in \mathcal{N}_a$, it holds that (I) $\|\mathbf{x}_c - \mathbf{1}_N \otimes c_{j\mathrm{g}}\| = \|\mathbf{x}_c - \mathbf{1}_N \otimes c_{i\mathrm{g}} + \mathbf{1}_N \otimes c_{i\mathrm{g}} - \mathbf{1}_N \otimes c_{j\mathrm{g}}\| \ge \sqrt{N}\,\|c_{i\mathrm{g}} - c_{j\mathrm{g}}\| - \|\mathbf{x}_c - \mathbf{1}_N \otimes c_{i\mathrm{g}}\| \ge 2\sqrt{N}\, r_{\min} - r_S(\varepsilon)$, due to that $\|c_{i\mathrm{g}} - c_{j\mathrm{g}}\| > 2r_{\min}$ by Assumption 6.1. Since $\varepsilon < \varepsilon_2$, then $r_S(\varepsilon) < r_S(\varepsilon_2) = r_{\min}$. Thus $\|\mathbf{x}_c - \mathbf{1}_N \otimes c_{j\mathrm{g}}\| > 2\sqrt{N}\, r_{\min} - r_{\min} > r_{\min} = r_S(\varepsilon_2)$, implying that $\mathbf{x}_c \notin \mathcal{S}_j$. (II) $\|x_j - c_{i^\star \mathrm{g}}\| < \|\mathbf{x}_c - \mathbf{1}_N \otimes c_{i^\star \mathrm{g}}\| < r_{\min} < r_{i^\star \mathrm{g}}$, meaning that $x_j \in \pi_{i^\star \mathrm{g}}$, $\forall j \in \mathcal{N}$. Thus, for each active robot $j \in \mathcal{N}_a$, it holds that $\|x_j - c_{j\mathrm{g}}\| = \|x_j - c_{i^\star \mathrm{g}} + c_{i^\star \mathrm{g}} - c_{j\mathrm{g}}\| \ge \|c_{i^\star \mathrm{g}} - c_{j\mathrm{g}}\| - \|x_j - c_{i^\star \mathrm{g}}\| \ge 2r_{\min} - r_{\min} > r_{\min}$. ■

Lemma 6.6. *There exists $\varepsilon_6 > 0$ such that if $\varepsilon < \varepsilon_6$, then for any critical point $\mathbf{x}_c \in \mathcal{S}_i$, $i \in \mathcal{N}_a$, then it holds that $\|p_i\| < \sqrt{0.4\varepsilon}$.*

Proof. Without loss of generality, let $\mathbf{x}_c \in \mathcal{S}_{i^\star}$, where $i^\star \in \mathcal{N}_a$. By summing (6.11) for all $i \in \mathcal{N}$, we get $d_{i^\star}\, p_{i^\star} = -\sum_{j \neq i^\star, j \in \mathcal{N}_a} d_j\, p_j$. Consider the scalar function $f(\|p_j\|) = d_j(\|p_j\|)\|p_j\|$ for $\|p_j\| \ge 0$. It is monotonically increasing for $\|p_j\| \in [0,\, 3.2\sqrt{\varepsilon})$ and decreasing for $\|p_j\| \in [3.2\sqrt{\varepsilon},\, \infty)$. If $\mathbf{x}_c \in \mathcal{S}_{i^\star}$ for $i^\star \in \mathcal{N}_a$, then $\|\mathbf{x}_c - \mathbf{1}_N \otimes c_{i^\star \mathrm{g}}\| \le r_S(\varepsilon_2)$. Moreover, $\|\mathbf{x} - \mathbf{1}_N \otimes c_{i^\star \ell}\| \ge \|\mathbf{1}_N \otimes x_{i^\star} - \mathbf{1}_N \otimes c_{i^\star \mathrm{g}}\| - \|\mathbf{x} - \mathbf{1}_N \otimes x_{i^\star}\| \ge \sqrt{N}\,\|p_{i^\star}\| - \sqrt{(N-1)\varepsilon}\sqrt{\varepsilon}\,\xi$. This implies $\|p_{i^\star}\| \le \sqrt{3\,\varepsilon} + 2\sqrt{\varepsilon\sqrt{\varepsilon}\,\xi}$. Moreover by Lemma 6.5, $\|p_j\| > r_{\min}$, $\forall j \neq i^\star$, $j \in \mathcal{N}_a$. Thus if $r_{\min} > 3.2\sqrt{\varepsilon}$, namely $\varepsilon < 0.07\, r_{\min}^2 \triangleq \varepsilon_3$, it holds that $d_j\,\|p_j\| < 0.5\varepsilon^2/r_{\min}$, $\forall j \neq i^\star$, $j \in \mathcal{N}_a$. Thus $d_{i^\star}\,\|p_{i^\star}\| < 0.5(N_a - 1)\varepsilon^2/r_{\min}$. If the following two conditions hold: (i) $\sqrt{3\,\varepsilon} + 2\sqrt{\varepsilon\sqrt{\varepsilon}\,\xi} < 3.2\sqrt{\varepsilon}$; (ii) $0.5(N_a - 1)\varepsilon^2/r_{\min} < d_j(\sqrt{0.4\varepsilon})\sqrt{0.4\varepsilon}$, then $\|p_{i^\star}\| < \sqrt{0.4\varepsilon}$ since it is shown earlier that function $d_j(\|p_j\|)\|p_j\|$ is monotonically increasing for $\|p_j\| \in [0,\, 3.2\sqrt{\varepsilon})$. Condition (i) above implies that $\varepsilon < 4.1/\xi^2 \triangleq \varepsilon_4$ and condition (ii) holds for any $N_a \le N$ if $\varepsilon < 0.8\, r_{\min}^2/(N-1)^2 \triangleq \varepsilon_5$. To conclude, if $\varepsilon < \varepsilon_6$, where

$$\varepsilon_6 \triangleq \min\{\varepsilon_3,\, \varepsilon_4,\, \varepsilon_5\}, \tag{6.19}$$

then $\mathbf{x}_c \in \mathcal{S}_{i^\star}$ implies $\|p_{i^\star}\| < \sqrt{0.4\varepsilon}$. ■

With the above two lemmas, we can now show that all critical points of $V(t)$ within $\mathcal{S}$ are local minima.

Lemma 6.7. *There exists $\varepsilon_{\min} > 0$ such that if $\varepsilon < \varepsilon_{\min}$, all critical points of V within $\mathcal{S}$ are local minima.*

Proof. A critical point $\mathbf{x}_c \in \mathcal{S}$ can only belong to one set $\mathcal{S}_i$ of an active robot $i \in \mathcal{N}_a$ by Lemma 6.5. Let $\mathbf{x}_c \in S_{i^\star}$, where $i^\star \in \mathcal{N}_a$. Let $\mathbf{z} \in \mathbb{R}^{2N}$ and $\|\mathbf{z}\| = 1$. Set $\mathbf{z} = [z_1^T, z_2^T, \cdots, z_n^T]^T$, where $z_i \in \mathbb{R}^2$, $\forall i \in \mathcal{N}$. Then $\mathbf{z}^T \nabla^2 V\, \mathbf{z}$ at $\mathbf{x}_c$ is computed as:

$$\begin{aligned} \mathbf{z}^T \nabla^2 V \mathbf{z} = &\sum_{i \in \mathcal{N}_a} \left(d_i\, \|z_i\|^2 + d_i'\, |p_i^T z_i|^2 \right) + \\ &\sum_{(i,j) \in E(t)} \left(h_{ij} \|z_{ij}\|^2 + 2\, h_{ij}'\, |x_{ij}^T z_{ij}|^2 \right). \end{aligned} \tag{6.20}$$

where $z_{ij} \triangleq z_i - z_j$. Since $|p_i^T z_i| \le \|p_i\|\|z_i\|$, $d_i > 0$ and $d_i' < 0$, it holds that $d_i\,\|z_i\|^2 + d_i'\,|p_i^T z_i|^2 \ge (d_i + d_i'\|p_i\|^2)\|z_i\|^2$, $\forall i \in \mathcal{N}_a$. It holds that for $j \neq i^\star$ and $\forall j \in \mathcal{N}_a$,

$d_j + d'_j\|p_j\|^2 > \varepsilon^2\hat{g}$ where $\hat{g} \triangleq -2/r_{\min}^2$, since $\|p_j\| > r_{\min}$ by Lemma 6.5; and $d_{i^\star} + d'_{i^\star}\|p_{i^\star}\|^2 > 0.08\varepsilon$ since $\|p_{i^\star}\| > \sqrt{0.4\varepsilon}$ by Lemma 6.6. Regarding the second term of (6.20), since Lemma 6.2 shows that $G(t)$ is a complete graph at $\mathbf{x}_c$ with $h_{ij} > 1/r^2$ and $h'_{ij} > 0$, we get $\sum_{(i,j)\in E}\left(h_{ij}\|z_{ij}\|^2 + 2h'_{ij}|x_{ij}^T z_{ij}|^2\right) \geq \sum_{j\in\mathcal{N}}\|z_{i^\star j}\|^2/r^2$. Thus (6.20) can be bounded by

$$\begin{aligned}
\mathbf{z}^T\nabla^2 V\mathbf{z} &\geq \sum_{i\in\mathcal{N}_a}\left(d_i + d'_i\|p_i\|^2\right)\|z_i\|^2 + \sum_{j\in\mathcal{N}} h_{i^\star j}\|z_{i^\star j}\|^2 \\
&\geq 0.08\,\varepsilon\|z_{i^\star}\|^2 - \varepsilon^2\sum_{j\neq i^\star, j\in\mathcal{N}_a}|\hat{g}|\|z_j\|^2 + \frac{1}{r^2}\sum_{j\in\mathcal{N}}\|z_{i^\star j}\|^2 \\
&\geq \sum_{j\in\mathcal{N}_a}\Big(\frac{1}{r^2} + \frac{0.08\varepsilon}{N}\Big)\|z_{i^\star}\|^2 + \Big(\frac{1}{r^2} - \varepsilon^2|\hat{g}|\Big)\|z_j\|^2 - \frac{2}{r^2}z_{i^\star}^T z_j,
\end{aligned}$$

as $1 \leq N_a \leq N$. If the following condition holds:

$$\Big(\frac{1}{r^2} + \frac{0.08\varepsilon}{N}\Big)\Big(\frac{1}{r^2} - \varepsilon^2|\hat{g}|\Big) > \Big(\frac{1}{r^2}\Big)^2, \tag{6.21}$$

it implies $\mathbf{z}^T\nabla^2 V\mathbf{z} > \left(|z_{i^\star}^T z_j| - z_{i^\star}^T z_j\right)/r^2 \geq 0$, $\forall\,\mathbf{z}\in\mathbb{R}^{2N}$, i.e., $\nabla^2 V$ is positive definite at $\mathbf{x}_c \in \mathcal{S}$. Equation (6.21) is equivalent to $\varepsilon^2 + \frac{N}{0.08\,r^2}\varepsilon - \frac{1}{r^2|\hat{g}|} < 0$. Since $\varepsilon > 0$, this implies that

$$0 < \varepsilon < \frac{\sqrt{\left(\frac{N}{0.08\,r^2}\right)^2 + \frac{4}{r^2|\hat{g}|}} - \frac{N}{0.08\,r^2}}{2} \triangleq \varepsilon_7. \tag{6.22}$$

To conclude, if

$$\varepsilon < \min\{\varepsilon_1, \varepsilon_2, \varepsilon_6, \varepsilon_7\} \triangleq \varepsilon_{\min}, \tag{6.23}$$

where ε_1, ε_2, ε_6 and ε_7 are defined in (6.17), (6.18), (6.19) and (6.22) and are all positive, then all local minima within $\mathcal{S}$ are stable. ■

By summarizing Lemmas 6.3-6.7, we can derive the following convergence result for the controlled closed-loop system:

Theorem 6.8. *Assume that $G(T_s)$ is connected and $\varepsilon < \varepsilon_{\min}$ by (6.23). Then starting from anywhere in the workspace except a set of measure zero, there exists a finite time $T_f \in [T_s, \infty)$ and one robot $i^\star \in \mathcal{N}_a$, such that $x_j(T_f) \in \pi_{i^\star g}$, $\forall\, j \in \mathcal{N}$, while at the same time $\|x_i(t) - x_j(t)\| < r$, $\forall (i,j) \in E(T_s)$ and $\forall t \in [T_s, T_f]$.*

Proof. First of all, the second part follows directly from Theorem 6.1 which guarantees that all edges within $E(T_s)$ will be reserved for all $t > T_s$. Secondly, we have shown that $V(t)$ by (6.6) is lower-bounded and non-increasing after $G(t)$ becomes complete by Lemma 6.2. By LaSalle's invariance principle [82], we only need to find out the largest invariant set within $\dot{V}(t) = 0$. Lemmas 6.3 to 6.7 ensure that the potential function $V(t)$ has only local minima inside $\mathcal{S}$ and saddle points outside $\mathcal{S}$. These saddle points have attractors of measure zero by Lemma 6.4. Thus starting from anywhere in the workspace except a set of measure zero, the system converges

to the set of local minima. Part (I) of Lemma 6.5 shows that a local minimum can not belong to two different $\mathcal{S}_i$ simultaneously. Thus the system converges to the set of local minima within $\mathcal{S}_{i^\star}$ for one active robot $i^\star \in \mathcal{N}_a$. By part (II) of Lemma 6.5, all robots would be inside $\pi_{i^\star \mathrm{g}}$ at a critical point within $\mathcal{S}_{i^\star}$, i.e., $x_j \in \pi_{i^\star \mathrm{g}}$, $\forall j \in \mathcal{N}$. Consequently, by continuity, there exists a finite time $T_f < \infty$ that $x_j(T_f) \in \pi_{i^\star \mathrm{g}}$, $\forall j \in \mathcal{N}$, for exactly one active robot $i^\star \in \mathcal{N}_a$. ■

***Remark* 6.1.** Note that Theorem 6.8 above holds for any number of active robots that $1 \leq N_a \leq N$. In other words, independent of the number of active robots within the team, one of the active robots will reach its goal region first within finite time, while the whole team fulfills the relative-distance constraints at all time. ▲

6.1.4 Control mode switching protocol

In this part, we propose *three* different switching protocols for each robot to decide on its own activity or passivity under there different cases, such that all robots can fulfill their local tasks and at the same time satisfy the relative-distance constraints. Through these protocols, we can integrate the discrete plan execution from Section 6.1.2 and the continuous control laws from Section 6.1.3 into a hybrid control scheme, which monitors the plan execution and motion control online. This hybrid scheme is fully decentralized and only relies on local relative-state measurements.

Switching protocol for sc-LTL

Let us first focus on a case where each local task φ_i, $i \in \mathcal{N}$ is an sc-LTL formula. As introduced in Section 6.1.2, the discrete plan τ_i for robot i can be represented by a finite satisfying prefix of progressive goal regions and the set of services to provide at each region: $\tau_{i,\mathrm{pre}} = (\pi_{i1}, w_{i1}) \cdots (\pi_{ik_i}, w_{ik_i})$, where $\pi_{i1}, \pi_{i2}, \cdots, \pi_{ik_i} \in \Pi_i$ and $w_{i1}, w_{i2}, \cdots, w_{ik_i} \in 2^{AP_i}$. We propose the following *activity switching protocol* for each robot $i \in \mathcal{N}$, which is referred by $\mathbf{P_{sc}}$ in the sequel:

(I) At time $t = 0$, robot i sets $\kappa_i := 1$ and itself as active and sets $\pi_{ig} := \pi_{i\kappa_i}$, namely the first goal region by τ_i. The *active* controller (6.3) is applied to robot i, where the progressive goal region is π_{ig}, i.e., $c_{ig} = c_{i\ell_1}$.

(II) Whenever robot i reaches its current progressive goal region $\pi_{ig} = \pi_{i\kappa_i}$ and $\kappa_i < k_i$, it provides the prescribed set of services $w_{i\kappa_i}$ by τ_i and it sets $\kappa_i := \kappa_i + 1$ and $\pi_{ig} := \pi_{i\kappa_i}$. Then the controller (6.3) for robot i is updated accordingly by setting $c_{ig} = c_{i\ell_{k+1}}$.

(III) Whenever robot i reaches its last progressive goal region $\pi_{ig} = \pi_{ik_i}$, it provides the set of services w_{ik_i} by which it finishes the execution of its finite discrete plan τ_i. Afterwards it remains *passive* and controller (6.4) applies to robot i.

Theorem 6.9. *By following the switching protocol* $\mathbf{P_{sc}}$ *above, it is guaranteed that* $\forall i \in \mathcal{N}$, φ_i *is satisfied by* $\mathbf{x}_i(T)$, *and* $\|x_i(t) - x_j(t)\| < r$, $\forall (i,j) \in E_0(0)$ *and* $\forall t \geq 0$, *where* $T = \infty$.

Proof. At $t = 0$, all robots are active and following the controller (6.3). By Theorem 6.8, all robots converge to one robot's goal region at a finite time $t_1 > 0$. Denote by $i \in \mathcal{N}$ this robot. Then either by step (II) of the protocol robot i updates its active control law by setting $\pi_{ig} = \pi_{i2}$, or by step (III) robot i has completed its plan $\tau_{i,\mathrm{pre}}$ and becomes passive. Since all robots' plans are finite and Theorem 6.8 holds for any number of active robots, we obtain that there exists a finite time instant T_{f_j}, such that one of the robots $j \in \mathcal{N}_a$ finishes executing its plan $\tau_{j,\mathrm{pre}}$, i.e., such that φ_j becomes satisfied. Then by step (III), this robot is passive and following the controller (6.4) for all times $t \in [T_{f_j}, \infty)$. Inductively, we conclude that there exists a time instant T_f, by which all robots complete their plans and all formulas are satisfied. All robots are passive for all $t \in (T_f, \infty)$ and by controller (6.4) they all converge to one point. The second part follows directly from Theorem 6.8. ■

Switching protocol for general LTL

As introduced in Section 6.1.2, if the task specification φ_i is given as a general LTL formula, then the plan τ_i is represented by an infinite sequence of goal regions and services in the prefix-suffix form: $\tau_i = \tau_{i,\mathrm{pre}}(\tau_{i,\mathrm{suf}})^\omega = (\pi_{i1}, w_{i1})(\pi_{i2}, w_{i2})\cdots$, where $\tau_{i,\mathrm{pre}} = (\pi_i, w_{i1})\cdots(\pi_{ik_i}, w_{ik_i})$, for $k_i > 0$ and $\tau_{i,\mathrm{suf}} = (\pi_{ik_i+1}, w_{ik_i+1})\cdots(\pi_{iK_i}, w_{iK_i})$, where the same as before $\pi_{i1}, \pi_{i2}, \cdots, \pi_{iK_i} \in \Pi_i$ is the sequence of goal regions and $w_{i1}, w_{i2}, \cdots, w_{iK_i} \in 2^{AP_i}$ is the associated sequence of services.

The main challenge in this case is to ensure that each robot executes its plan suffix infinitely often. The activity switching protocol $\mathbf{P_{sc}}$ from Section 6.1.4 could not be applied here since all robots should remain active at all time due to the infinite discrete plan. Besides, it is possible that the team may repetitively converge to π_{ig} for one robot $i \in \mathcal{N}$ while never visiting the other robots' progressive goal regions. Hence, we aim here to design a "fair" protocol that enforces a progressive satisfaction towards each robot's local task. Thereto, we first introduce a communication-free reaching-event detector that enables an robot to monitor its neighbors' plan executions, particularly to detect when one of its neighbors has reached that neighbor's progressive goal region. As we have shown in Lemma 6.2, the connectivity graph is complete when the system is at any critical point. Thus every robot could monitor all the other robots' plan execution by the reaching-event detector:

Reaching-event detector. Robot $i \in \mathcal{N}$ can detect when it reaches its own progressive goal region π_{ig} by checking if $x_i(t) \in \pi_{ig}$. However it is also essential that it can detect when another robot $j \in \mathcal{N}$ reaches π_{jg}. Since the connectivity graph is complete, it is sufficient for robot i to detect when a neighboring robot $j \in \mathcal{N}_i(t)$ reaches π_{jg}. Given that the robots satisfy the dynamics by (6.1) and that each robot $i \in \mathcal{N}$ can measure $x_i(t) - x_j(t)$, $\forall j \in \mathcal{N}_i(t)$ in real time, we assume that the robot i can measure or estimate [40] $u_j(t)$, $\forall j \in \mathcal{N}_i(t)$. Let $\Omega_i(j, t) \in \mathbb{B}$ be a Boolean variable indicating that robot i detects its neighboring robot $j \in \mathcal{N}_i(t)$ reaching the goal region π_{jg} at time $t > 0$. We propose a reaching-event detector below inspired by [110]. Simply speaking, the detector checks if within a short time period $[t - \Delta_t, t]$, there exists $j \in \mathcal{N}_i(t)$, such that $u_j(t)$ has changed from a

relatively small value (below a given Δ_u) by a difference larger than certain Δ_d. If so, it indicates that the robot j has reached its progressive goal region π_{jg}. This design is motivated by the following facts: By Theorem 6.7, the system is at a local minimum whenever an active robot is in its progressive goal region. Thus, when the robot j reaches π_{jg} at time t, all control inputs $u_i(t)$ are close to zero for all $i \in \mathcal{N}$ by (6.11). Afterwards, our switching protocol guarantees that *only* robot j switches its control law either to (6.3) to navigate to the next goal region or to (6.4) to become passive. This change is lower-bounded by Δ_d derived using control law (6.3) and Lemmas 6.5, 6.6 as $\Delta_d \triangleq |f(r_{\min}) - f(\sqrt{0.4\varepsilon})|$, where $f(\|p_j\|) = d_j(\|p_j\|)\|p_j\|$ is a scalar function and $d_j(\|p_j\|)$ is defined by (6.5). In contrast, for the other robots $i \neq j$, $i \in \mathcal{N}$, the control input $u_i(t)$ remains unchanged and close to zero.

In this protocol, an robot $i \in \mathcal{N}$ becomes passive if it has made a certain progress towards the satisfaction of its specification, hence giving the other robots an opportunity to advance in execution of their plans. However, once every robot has achieved certain progress, robot i becomes active again to proceed with its infinite plan. We define a *round* as the time period during which each robot has reached at least one of its goal regions according to their plans.

Definition 6.3. For all $m \geq 1$, the **m-th round** is defined as the time interval $[T_{\circlearrowleft_{m-1}}, T_{\circlearrowleft_m})$, where $T_{\circlearrowleft_0} = 0$, $T_{\circlearrowleft_{m-1}} < T_{\circlearrowleft_m}$ and for all $m \geq 1$, $T_{\circlearrowleft_m}$ is the smallest time satisfying the following conditions for all $i \in \mathcal{N}$: $\mathtt{word}_i(T_{\circlearrowleft_m}) = w_{i1}w_{i2}\cdots w_{i\ell}$ for some $\ell \geq 1$ and $\mathtt{word}_i(T_{\circlearrowleft_m}) \neq \mathtt{word}_i(T_{\circlearrowleft_{m-1}})$. ▲

This notion of a round is crucial to the protocol design below. We fistly introduce two local variables $\chi_i \geq 0$ that indicates the starting time of the current round and $\Upsilon_i \in \mathbb{Z}^N$ a vector to record how many progressive goal regions each robot has reached within one round since χ_i. Then the *activity switching protocol* for each robot $i \in \mathcal{N}$, referred by $\mathbf{P_{ge}}$ in the sequel, is as follows:

(I) At time $t = 0$, $\Upsilon_i := \mathbf{0}_N$, $\chi_i := 0$, $\kappa_i := 1$. The robot i is active and follows control law (6.3), where $\pi_{ig} := \pi_{i\kappa_i}$.

(II) Whenever the robot i reaches its current progressive goal region $\pi_{ig} = \pi_{i\kappa_i}$, it provides the prescribed set of services $w_{i\kappa_i}$ by τ_i and updates the current progressive goal region accordingly: If $\kappa_i < K_i$ then $\kappa_i := \kappa_i + 1$, and if $\kappa_i = K_i$ then $\kappa_i := k_i + 1$. Furthermore, $\pi_{ig} := \pi_{i\kappa_i}$, and finally $\Upsilon_i[i] := \Upsilon_i[i] + 1$. Generally speaking, the robot i decides to stay active or to become passive based on the probability function:

$$\mathbf{Pr}(b_i = 1) = \begin{cases} f_{\text{prob}}(\cdot) & \text{if } f_{\text{cond}}(\cdot) = \top, \\ 0 & \text{otherwise,} \end{cases}$$

where $f_{\text{prob}}(\cdot) \in [0, 1]$ and $f_{\text{cond}}(\cdot) \in \{\top, \bot\}$ are functions of time t and the local variables Υ_i and χ_i, subject to the following: given that the current round is the m-th one, there exists a time $T \in (T_{\circlearrowleft_{m-1}}, T_{\circlearrowleft_m})$, such that $f_{\text{cond}}(\cdot) = \bot$

for all $t \in [T, T_{\circlearrowleft_m})$. Whenever $b_i = 1$, the robot i keeps following the control law (6.3) with the updated π_{ig}. Otherwise, it becomes passive and the control law (6.4) is applied.

(III) Whenever robot i detects that $\Omega_i(j,t) = \top$, for some $j \neq i \in \mathcal{N}$, it sets $\Upsilon_i[j] = \Upsilon_i[j] + 1$.

(IV) Whenever it holds that $\Upsilon_i[j] \geq 1$, $\forall j \in \mathcal{N}$, i.e., all elements of Υ_i are positive, then robot i sets $\Upsilon_i \coloneqq \mathbf{0}_N$, $\chi_i \coloneqq t$ and follows the active control law (6.3) to its goal region π_{ig}.

A straightforward choice of the function $\mathbf{Pr}(\cdot)$ is $f_{\text{cond}} = \bot$, for all $t \geq 0$. Then the robot i always becomes passive once it visits π_{ig} and it becomes active again after the current round is completed by step (IV). In this case, the number of active robots gradually decreases within each round. However, a different choice may allow trading the fairness of activity switching with the increased efficiency of plan executions. The switching to passive control mode may be temporarily postponed and as a result, the visits to progressive goal regions may become more frequent. Examples of such a case are given in Section 6.1.6.

Lemma 6.10. *The round* $[T_{\circlearrowleft_{m-1}}, T_{\circlearrowleft_m})$ *is finite,* $\forall m \geq 1$.

Proof. Let $t = T_{\circlearrowleft_{m-1}} = 0$, and thus $\Upsilon_i[j] = 0$, for all $i, j \in \mathcal{N}$ by step (I). By Theorem 6.8, one of the robots reaches its progressive goal region in finite time at $t_1 \geq T_{\circlearrowleft_{j-1}}$. Since there are only finite number of robots and due to the required properties of f_{cond}, there exists a finite time $T_{f_j} \geq 0$, when either the step (IV) applies or when one of the robots $j \in \mathcal{N}_a$ necessarily becomes passive by the function $\mathbf{Pr}(\cdot)$ in step (II) and remains passive till the end of the round. In the former case, $T_{\circlearrowleft_m} = T_{f_j}$, i.e., we directly obtain that the first round is finite. In the latter case, the same argument can be applied to the $N-1$ active robots such that one of them will become passive in finite time. By repeating this process, we obtain that there exists a finite time instant T_f, such that step (IV) applies, i.e., such that $T_{\circlearrowleft_m} = T_f$. Again, we derive that the first round is finite. Inductively, let $m > 1$, $t = T_{\circlearrowleft_{m-1}}$, and $\Upsilon_i[j] = 0$, for all $i, j \in \mathcal{N}$ by step (IV). Using analogous arguments as above, we derive that the m_{th} round $[T_{\circlearrowleft_m}, T_{\circlearrowleft_{m+1}}]$ is finite. ∎

Theorem 6.11. *By following the switching protocol* $\mathbf{P_{ge}}$ *above, it is guaranteed that* $\forall i \in \mathcal{N}$, φ_i *is satisfied by* $\mathbf{x}_i(T)$ *and* $\|x_i(t) - x_j(t)\| < r$, $\forall (i,j) \in E_0(0)$ *and* $\forall t > 0$, *where* $T = \infty$.

Proof. The satisfaction of φ_i follows directly from the correctness of each robot's discrete plan and the fact that each round is finite by Lemma 6.10. At last, the distance constraints are always maintained as shown in Theorem 6.8. ∎

Switching protocol for mixed task specifications

As stated in Section 6.1.1, the tasks $\{\varphi_i, i \in \mathcal{N}\}$ can be of different types. Namely, some tasks are given as sc-LTL formulas (denoted by this set of robots $\mathcal{N}_{sc} \subseteq \mathcal{N}$) and some are given as general LTL formulas (denoted by $\mathcal{N}_{ge} \subseteq \mathcal{N}$).

We firstly show that a new switching protocol is needed for this case, i.e., simply applying the protocol $\mathbf{P_{sc}}$ for robots in $\mathcal{N}_{sc}$ and the protocol $\mathbf{P_{ge}}$ for robots in $\mathcal{N}_{ge}$ is not a valid solution. The reason is that when one robot $j \in \mathcal{N}_{sc}$ has finished executing its plan, it would switch to being passive indefinitely by step (III) of $\mathbf{P_{sc}}$. After that for all robots $i \in \mathcal{N}_{ge}$, one round may never finish in finite time since step (IV) of $\mathbf{P_{ge}}$ will not be reached as $\Upsilon_i[j] = 0$ holds always. Thus it is also important for robot $i \in \mathcal{N}_{ge}$ to detect when one of its neighbors $j \in \mathcal{N}_i$ belonging $\mathcal{N}_{sc}$ has already finished executing its plan and switched to being passive indefinitely. To that end, we propose another event detector as follows:

All-passive detector. Let $\Psi_i(t) \in \mathbb{B}$ be a Boolean variable which indicates that robot $i \in \mathcal{N}_{ge}$ detects that *all* of its neighboring robots are in the passive mode at time $t > 0$. As discussed earlier, when all robots within $\mathcal{N}$ are passive and following the controller (6.4), the closed-loop system dynamics can be described by $\dot{\mathbf{x}} = -(\mathbf{H} \otimes I_2)\mathbf{x}$, where the matrix $\mathbf{H}$ is defined in Section 6.1.3. It has been shown that $\mathbf{H}$ is positive semidefinite with only one zero eigenvalue. As a result, all robots would asymptotically converge to one rendezvous point [122]. In other words, $x_{ij}(t) = x_i(t) - x_j(t) \to 0$ and $u_i(t) \to 0$ as $t \to +\infty$, $\forall (i,j) \in \mathcal{N} \times \mathcal{N}$. Thus we propose that $\Psi_i(t')$ becomes $\top$ if robot i detects that $|u_j| < \Delta_c$ holds, $\forall j \in \mathcal{N}_i$ and $\forall t \in [t' - \Delta_p, t']$, where $\Delta_c > 0$ is the upper bound on the control input and $\Delta_p > 0$ is the monitoring period. Given the appropriately chosen Δ_c and Δ_p, $\Psi_i(t')$ becomes $\top$ only when all robots are passive at time $t = t'$. Without loss of generality, assume there is at least one robot being active in the team at time $t = t'$. Since $|u_i| < \Delta_c$ holds for all time $t \in [t' - \Delta_p, t']$, it means the system stays at the critical point of $V(t)$ associated with one of the active robots for at least the time interval $[t' - \Delta_p, t']$. By the analysis of $V(t)$ from Section 6.1.3, this violates the fact that all active robots should navigate to their individual goal regions by following (6.3). Thus $\Psi_i(t)$ becomes $\top$ only when all robots are passive at time t.

Then the activity switching protocol for this case, denoted by $\mathbf{P_{mx}}$, is designed as follows: for any robot $i \in \mathcal{N}_{sc}$, it simply follows the switching protocol $\mathbf{P_{sc}}$. Namely, it traverses the sequences of goal regions and provides the set of services there according to its finite plan $\tau_{i,\mathrm{pre}}$. After it finishes the execution, it remains passive indefinitely. On the other hand, for any robot $i \in \mathcal{N}_{ge}$, we introduce a new variable $\mathcal{N}_{i,sc}(t) \subseteq \mathcal{N}_i(t)$ to save the set of robot i's neighbors belonging to $\mathcal{N}_{sc}$, which is initialized as empty and maintained locally by robot i. For any robot $i \in \mathcal{N}_{ge}$, the steps (I)-(III) of protocol $\mathbf{P_{ge}}$ remain the same, but (IV) should be modified as follows, and an additional step (V) needs to be added:

(IV) Whenever it holds that $\Upsilon_i[j] \geq 1$, $\forall j \in \mathcal{N}$ and $j \notin \mathcal{N}_{i,sc}$, then robot i sets $\Upsilon_i \coloneqq \mathbf{0}_N$, $\chi_i \coloneqq t$ and follows the active control law (6.3) to its goal region π_{ig}.

(V) Whenever robot i detects that $\Psi_i(t) = \top$, then for each $j \in \mathcal{N}$, if $\Upsilon_i[j] = 0$, add j to $\mathcal{N}_{i,sc}$.

Namely, by step (IV) above, each robot $i \in \mathcal{N}_{ge}$ would reset Υ_i to 0 and start a new round once every robot has made a progress in its plan execution, *except* those belonging to $\mathcal{N}_{i,sc}$, i.e., the robots with sc-LTL formulas and that have accomplished their tasks. Then when $\Psi_i(t) = \top$, it means that all robots are in the passive mode. If the neighbor $j \in \mathcal{N}_i$ also belongs to $\mathcal{N}_{ge}$, by step (II) of protocol $\mathbf{P_{ge}}$ in Section 6.1.4 robot j must have reached its goal region at least once, i.e., $\Upsilon_i[j] \geq 1$. Thus if $\Upsilon_i[j] = 0$ for some neighbor $j \in \mathcal{N}_i$ when $\Psi_i(t) = \top$, it implies $j \in \mathcal{N}_{sc}$ and moreover robot j has finished executing its finite plan according to step (II) of protocol $\mathbf{P_{sc}}$, and it remains passive afterwards with $\Upsilon_i[j]$ being constantly zero. Thus robot i adds j to $\mathcal{N}_{i,sc}$ by step (V) above.

Theorem 6.12. *By following the switching protocol* $\mathbf{P_{mx}}$ *above, it is guaranteed that* $\forall i \in \mathcal{N}$, φ_i *is satisfied by* $\mathbf{x}_i(T)$ *and* $\|x_i(t) - x_j(t)\| < r$, $\forall (i,j) \in E_0(0)$ *and* $\forall t > 0$, *where* $T = \infty$.

Proof. Similar to the proof of Theorem 6.11, we only need to show that in this case one "round" is also finite. Note that now the definition of round differs slightly from Definition 6.3 as here it is only defined for all robots in $\mathcal{N}_{ge}$. Before any robot $j \in \mathcal{N}_{sc}$ finishes executing its plan, one round is clearly finite as it ends once every robot has reached at least one of its progressive goal regions. Consider that one or more robots within $\mathcal{N}_{sc}$ (denoted by $\mathcal{N}_1 \subseteq \mathcal{N}_{sc}$) have finished executing their plans and become passive. All robots within $\mathcal{N}_{ge}$ will reach their goal regions at least once before they become passive in finite time as shown in Lemma 6.10, i.e., $\Upsilon_i[j] \geq 1$, $\forall j \in \mathcal{N}_{ge}$, while the robots in $\mathcal{N}_1$ remain passive since last round, i.e., $\Upsilon_i[j] = 0$, $\forall j \in \mathcal{N}_1$. According to step (V) above, each robot $i \in \mathcal{N}_{ge}$ would detect that all robots are passive and add all robots in $\mathcal{N}_1$ to $\mathcal{N}_{i,sc}$. Then by step (IV), all robots $i \in \mathcal{N}_{ge}$ would reset Υ_i and start a new round, since $\Upsilon_i[j] \geq 1$, $\forall j \in \mathcal{N}$ and $j \notin \mathcal{N}_{i,sc}$, i.e., all neighbors except those in $\mathcal{N}_{i,sc}$ have made a progress in plan execution.

The above procedure repeats itself until all robots in $\mathcal{N}_{sc}$ finish their plan execution and become passive. Then it holds that $\mathcal{N}_{i,sc} = \mathcal{N}_{sc}$, $\forall i \in \mathcal{N}_{ge}$ and the results from Theorem 6.11 apply directly, meaning each robot in $\mathcal{N}_{ge}$ can satisfy its local task. As a result, all robots in both $\mathcal{N}_{sc}$ and $\mathcal{N}_{ge}$ will satisfy their local tasks while fulfilling the relative-distance constraints for all time. ∎

It is obvious that the above three protocols have different applicabilities. Thus considering three cases separately is important such that the users can choose the suitable protocol accordingly.

6.1.5 Real-time discrete plan adaptation

In the aforementioned approaches, the discrete plan of each robot is synthesized only once initially from Section 6.1.2 and executed according to the hybrid control

scheme in real-time, regardless of the robots' actual trajectories. However, due to the relative-distance constraints, one robot's actual trajectory may be different from the planned one, i.e., it may detour to other robot's goal region as stated in Section 6.1.3. Thus given the robot's updated position, its initial plan τ_i might be inefficient in terms of cost defined by (6.2). Thus we propose a discrete plan adaptation algorithm along with the hybrid control scheme proposed earlier, which ensures that the updated plan always fulfills the given task and has the minimal *suffix cost* for any robot i with a general LTL formula defined as follows:

$$\boldsymbol{cost}(\overline{\mathbf{p}}_{i,\mathrm{suf}}(T)) \triangleq \max_{(\pi_s,\,\pi_g)\in\vartheta} \left\{W_i(\pi_s,\pi_g)\right\} \tag{6.24}$$

where $\vartheta = \{(\pi_{i0},\pi_{\ell_1}),(\pi_{\ell_k},\pi_{\ell_{k+1}}),\ \forall\,\pi_{\ell_k},\pi_{\ell_{k+1}} \in \overline{\mathbf{p}}_{i,\mathrm{suf}}(T)\}$; and $\overline{\mathbf{p}}_{i,\mathrm{suf}}(T)$ is the suffix part of the effective path $\overline{\mathbf{p}}_i(T)$. Recall that the plan suffix $\tau_{i,\mathrm{suf}}$ will be repeated infinitely often to satisfy a general LTL formula φ_i. Now assume that at time $t_0 > 0$, robot i finishes executing its current plan suffix $\tau_{i,\mathrm{suf}}$ once. Denote by $\tau^-_{i,\mathrm{suf}}(t_0)$ the plan suffix before the plan update at time $t_0 > 0$ and $\tau^+_{i,\mathrm{suf}}(t_0)$ the plan suffix after the update. Denote by $\mathtt{word}^-_i(0,\,t_0)$ the past sequence of services provided by robot i during the time period $[0,\,t_0]$. Since the corresponding word suffix of $\tau^+_{i,\mathrm{suf}}(t_0)$ is given by $\tau^+_{i,\mathrm{suf}}(t_0)|_{AP_i}$, the planned sequence of services by robot i during the time period $[t_0,\,T)$ is given by $\mathtt{word}^+_i(t_0,\,T) = \tau^+_{i,\mathrm{suf}}(t_0)|_{AP_i}$, where $T = \infty$. Denote by $\mathtt{word}^\star_i(T)$ the complete word from $t = 0$ to $t = T$, which is the complete sequence of services provided by robot i. Given the updated plan $\tau^+_i(t_0)$, $\mathtt{word}^\star_i(T)$ can be computed by concatenating the word during time $t = [0,\,t_0]$ and the word during time $t = [t_0,\,T)$ as follows:

$$\mathtt{word}^\star_i(T) = \mathtt{word}^-_i(0,\,t_0)\,\mathtt{word}^+_i(t_0,\,T). \tag{6.25}$$

On the other hand, $\tau^+_{i,\mathrm{suf}}(t_0)$ determines the suffix of an effective path after time t_0 by $\overline{\mathbf{p}}^+_{i,\mathrm{suf}}(t_0) = \tau^+_{i,\mathrm{suf}}(t_0)|_{\Pi_i}$, of which the suffix cost is given by (6.24). Formally, we consider the following problem:

Problem 6.2. *Find an updated plan suffix $\tau^+_{i,\mathrm{suf}}(t_0)$ for robot i such that: (I) $\mathtt{word}^\star_i(T)$ by* (6.25) *satisfies φ_i; (II) the effective path suffix $\overline{\mathbf{p}}^+_{i,\mathrm{suf}}(t_0)$ has the minimal cost by* (6.24). ▲

The solution consists of two main steps: (I) we compute the set of all product states $Q'_{\mathcal{P},i,t_0} \subseteq Q_{p,i}$ that are reachable from the initial states $Q_{p,i,0}$ given the past effective path $\overline{\mathbf{p}}^-_i(0,\,t_0)$ and the past sequence of services $\mathtt{word}^-_i(0,\,t_0)$ provided by robot i during time $[0,\,t_0)$. In particular, it can be computed by iterating through the sequence of input word by $\mathtt{word}^-_i(0,t_0)$ and computes the set of successors recursively, while at the same time ensuring that it is compliant with the robot's past effective path by $\overline{\mathbf{p}}^-_i(0,\,t_0)$. Note that $Q'_{\mathcal{P},i,t_0}$ can be maintained by each robot along with the plan execution procedure described in Section 6.1.1; (II) we compute firstly the intersection $F' = F_{p,i} \cap Q'_{\mathcal{P},i,t_0}$, which is always non-empty as $Q'_{\mathcal{P},i,t_0}$ contains at least one accepting state in $F_{p,i}$ as robot i has finished executing its plan

suffix once at $t = t_0$. Then the graph search algorithm described in Section 6.1.2 is applied with slight modifications to compute the minimal-bottleneck prefix and cycle, where $Q_{p,i,0}$ is replaced by $Q'_{\mathcal{P},i,t_0}$ and $F_{p,i}$ is replaced by F'. Denote by $\varrho_{i,\text{suf}}$ the minimal-bottleneck cycle from $v_f^\star$ back to itself, which is computed based on $\mathtt{P}_{v_f^\star}$. Then $\tau^+_{i,\text{suf}}(t_0)$ is determined by the projection of $\varrho_{i,\text{suf}}$ onto $\mathcal{T}_i$ and thus $\overline{\mathbf{p}}^+_{i,\text{suf}}(t_0)$ is given as the projection of $\tau^+_{i,\text{suf}}(t_0)$ onto Π_i. Its worst-case computational complexity [89, 97] is $|\delta_{p,i}|\cdot\log|Q_{p,i}|\cdot|Q_{p,i,0}|\cdot|\mathcal{F}_{p,i}|\cdot|Q_{p,i}|)$, where $|Q_{p,i}|$, $|\delta_{p,i}|$ are the number of states and transitions in $\mathcal{A}_{p,i}$.

Lemma 6.13. *$\tau^+_{i,\text{suf}}(t_0)$ derived above solves Problem 6.2.*

Proof. For part (I) of Problem 6.2: by how the reachable set $Q'_{\mathcal{P},i,t_0}$ is computed, we know that for any state $q'_s \in Q'_{\mathcal{P},i,t_0}$, there exists a path in $\mathcal{P}_i$ from one initial state $q'_0 \in Q_{p,i,0}$ to q'_s, which corresponds to $\mathtt{word}_i^-(0, t_0)$. Furthermore, $\tau^+_{i,\text{suf}}(t_0)$ is generated by enforcing its word $\mathtt{word}_i^+(t_0, T)$ corresponds to a path in $\mathcal{P}_i$ which starts from one state $q'_f \in Q'_{\mathcal{P},i,t_0} \cap F'$ and cycles back to itself. By concatenating $\mathtt{word}_i^-(0, t_0)$ and $\mathtt{word}_i^+(t_0, T)$ as in (6.25), it is guaranteed that the complete word $\mathtt{word}_i^\star(T)$ corresponds to a path in $\mathcal{P}_i$ from the initial state q'_0 to an accepting state q'_f and then back to itself, which is an accepting path of $\mathcal{A}_{p,i}$ by definition. As a result, the complete word $\mathtt{word}_i^\star(T)$ satisfies φ_i. Regarding part (II): following the synthesis above, we know that $\tau^+_{i,\text{suf}}(t_0)$ corresponds to the minimal-bottleneck suffix in $\mathcal{A}_{p,i}$ that minimizes the cost by (6.24). In particular, the first term $W_i(\pi'_{i0}, \pi_{\ell_1})$ corresponds to the bottleneck $\mathtt{D}_{v_0}$ for $v_0 \in Q_{p,i,0}$ and $W_i(\pi_{\ell_k}, \pi_{\ell_{k+1}})$, $\forall \pi_{\ell_k}\pi_{\ell_{k+1}} \in \overline{\mathbf{p}}_{i,\text{suf}}(T)$ corresponds to the bottleneck $\mathtt{D}_{v_f}$ for $v_f \in F'$. The total cost by (6.24) is minimized by choosing the best pair of (v_0, v_f), which gives the updated plan suffix $\tau^+_{i,\text{suf}}(t_0)$ and the effective path suffix $\overline{\mathbf{p}}^+_{i,\text{suf}}(t_0)$ with the minimal suffix cost. ∎

Now we discuss how to integrate the above plan adaptation scheme with the switching policies described earlier. We consider here only the policy $\mathbf{P_{ge}}$ from Section 6.1.4 and policy $\mathbf{P_{mx}}$ from Section 6.1.1. We propose that *each* robot with a general LTL task specification updates its plan whenever it finishes executing its plan suffix *once*, by executing the adaptation algorithm above to compute the updated plan suffix $\tau^+_{i,\text{suf}}$. To be more specific, for policy $\mathbf{P_{ge}}$, every robot follows the switching protocol and updates its plan suffix whenever it finishes executing its current plan suffix; for policy $\mathbf{P_{mx}}$, all robots follow the protocol but only the robots within $\mathcal{N}_{ge}$ update its plan suffix whenever it finishes executing its current plan suffix. Then the continuous controller and the switching protocol are updated accordingly given the updated plan suffix.

Lemma 6.14. *For all robot $i \in \mathcal{N}$, the final execution word $\mathtt{word}_i(T)$ satisfies φ_i for $T = \infty$, after applying the plan adaptation scheme described above.*

Proof. For policy $\mathbf{P_{ge}}$, since the plan adaptation is performed by every robot $i \in \mathcal{N}$ when it finishes executing its suffix once, Lemma 6.13 guarantees that $\tau^+_{i,\text{suf}}(t_0)$ is a cyclic suffix containing an accepting state of $\mathcal{A}_{p,i}$ after the update at $t = t_0$.

Thus at least one accepting state of $\mathcal{A}_{p,i}$ is visited between two consecutive plan updates. Moreover, as shown in Lemma 6.10 and Theorem 6.12, any plan suffix has finite length and can be executed in finite time. The set of accepting states of $\mathcal{A}_{p,i}$ will be visited infinitely many times as $T = \infty$. Since the number of accepting states in $\mathcal{A}_{p,i}$ is finite, at least one of the accepting states will be visited infinitely often by $\texttt{word}_i^\star(T)$ as $T = \infty$. Thus $\texttt{word}_i^\star(T)$ satisfies φ_i when $T = \infty$, $\forall i \in \mathcal{N}$. For policy $\mathbf{P_{mx}}$, as any robot $i \in \mathcal{N}_{sc}$ does not update its plan, the result from Theorem 6.12 still holds, while for any robot $i \in \mathcal{N}_{ge}$, the analysis from the previous case holds. ■

Theorem 6.15. *When combining the plan adaption scheme above with the switching protocol* $\mathbf{P_{ge}}$ *or* $\mathbf{P_{mx}}$*, it is guaranteed that* $\forall i \in \mathcal{N}$*, the local task* φ_i *is satisfied by* $\mathbf{x}_i(T)$ *and* $\|x_i(t) - x_j(t)\| < r$*,* $\forall (i, j) \in E_0(0)$ *and* $\forall t > 0$*, where* $T = \infty$*.*

Proof. A detailed proof is omitted here as the first part regarding the task satisfaction is a direct extension of Lemma 6.14 above, while the second part regarding relative-distance constraints can be shown in a similar way as in Theorems 6.11 and 6.12. ■

6.1.6 Case study

In this case study, we simulate a team of four autonomous robots $\mathcal{N} = \{\mathfrak{R}_1, \cdots, \mathfrak{R}_4\}$ subject to the dynamics (6.1) in a bounded, obstacle-free workspace of 40×40 meters (m). Each robot $\mathfrak{R}_i$ is given a local task specified as sc-LTL or LTL formulas φ_i. The simulation stepsize is set to $1ms$.

Workspace description

As shown in Figure 6.1a, the regions of interest are placed in top-left, top-right, bottom-right and bottom-left corners of the workspace and they all satisfy Assumption 6.1 with $c_{\max} = 40$ and $r_{min} = 2$.

(i) *Regions of interest.* Robot $\mathfrak{R}_1$ is an aerial vehicle with four regions of interest, denoted by $\Pi_1 = \{\pi_{1\texttt{tl}}, \pi_{1\texttt{tr}}, \pi_{1\texttt{br}}, \pi_{1\texttt{bl}}\}$ shown in red; robot $\mathfrak{R}_2$ is a ground vehicle with three regions of interest $\Pi_2 = \{\pi_{2\texttt{tl}}, \pi_{2\texttt{tr}}, \pi_{2\texttt{bl}}\}$ shown in green; robot $\mathfrak{R}_3$ is also a ground vehicle with $\Pi_3 = \{\pi_{3\texttt{tr}}, \pi_{3\texttt{br}}, \pi_{3\texttt{bl}}\}$ shown in blue; robot $\mathfrak{R}_4$ is an aerial vehicle with $\Pi_4 = \{\pi_{4\texttt{tl}}, \pi_{4\texttt{tr}}, \pi_{4\texttt{br}}, \pi_{4\texttt{bl}}\}$ shown in cyan. Note that we only consider the planar position of all robots. (ii) *Services.* Robot $\mathfrak{R}_1$ is capable of providing two kinds of services, i.e., surveillance over an area (denoted by σ_{11}) and assistance for ground operations (denoted by σ_{12}). Thus its set of atomic propositions is given by $\Sigma_1 = \{\sigma_{11}, \sigma_{12}\}$. Robot $\mathfrak{R}_4$ can provide the analogous kind of services as $\mathfrak{R}_1$, denoted by $\Sigma_4 = \{\sigma_{41}, \sigma_{42}\}$. Moreover, robot $\mathfrak{R}_2$ is capable of providing three kinds of services, i.e., food delivery (denoted by σ_{21}), water delivery (denoted by σ_{22}), and transportation (denoted by σ_{23}). Thus its set of atomic propositions is $\Sigma_2 = \{\sigma_{21}, \sigma_{22}, \sigma_{23}\}$. Robot $\mathfrak{R}_3$ can provide the analogous kind of services as $\mathfrak{R}_2$, denoted by $\Sigma_3 = \{\sigma_{31}, \sigma_{32}, \sigma_{33}\}$. (iii) *Region labeling.* The aerial assistance service is available at two regions of interest of $\mathfrak{R}_1$ while the surveillance service is available at the

other two regions. Namely, $L_1(\pi_{1\mathtt{tl}}) = L_1(\pi_{1\mathtt{br}}) = \{\sigma_{11}\}$ and $L_1(\pi_{1\mathtt{tr}}) = L_1(\pi_{1\mathtt{bl}}) = \{\sigma_{12}\}$. Similar statements hold for robot $\mathfrak{R}_4$, i.e., $L_4(\pi_{4\mathtt{tl}}) = L_4(\pi_{4\mathtt{tr}}) = \{\sigma_{41}\}$, $L_4(\pi_{4\mathtt{bl}}) = L_4(\pi_{4\mathtt{br}}) = \{\sigma_{42}\}$. While for robot $\mathfrak{R}_2$, the food delivery, water delivery and transportation services are available at its regions of interest, respectively. Namely, $L_2(\pi_{2\mathtt{tl}}) = \{\sigma_{21}\}$, $L_2(\sigma_{2\mathtt{tr}}) = \{\sigma_{22}\}$, $L_2(\pi_{2\mathtt{bl}}) = \{\sigma_{23}\}$. Similar statements hold for robot $\mathfrak{R}_3$, i.e., $L_3(\pi_{3\mathtt{tr}}) = \{\sigma_{31}\}$, $L_3(\pi_{3\mathtt{br}}) = \{\sigma_{32}\}$, $L_3(\pi_{3\mathtt{bl}}) = \{\sigma_{33}\}$. (iv) *Network graph.* The robots have a uniform neighboring radius as $r = 8m$ and the design parameter needed in Definition 6.2 is $\kappa = 0.5m$. They start from $[25, 15]$, $[20, 15]$, $[15, 20]$ and $[20, 25]$ in the 2D workspace. Thus the initial edge set of $G(0)$ is given by $E_0(0) = \{(\mathfrak{R}_1, \mathfrak{R}_2), (\mathfrak{R}_2, \mathfrak{R}_3), (\mathfrak{R}_3, \mathfrak{R}_4)\}$. The upper bound by (6.23) is $\varepsilon < \varepsilon_{\min} \approx 0.031$ and we choose $\varepsilon = 0.03$.

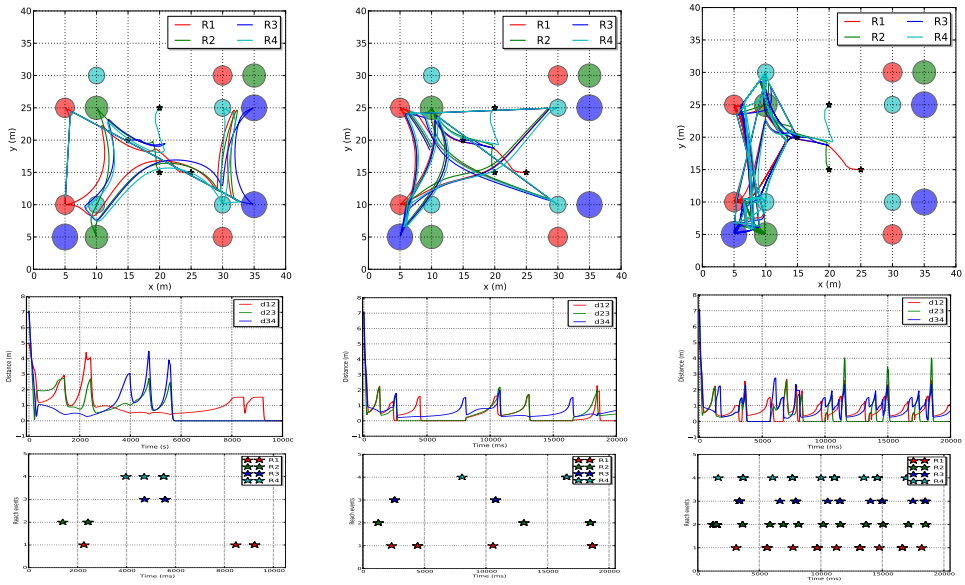
Task specification and simulation results

We consider two cases of the robot task specifications: one with sc-LTL formulas and one with general LTL formulas.

(I) ***sc-LTL Task Specifications***. The finite-time local task for robot $\mathfrak{R}_1$ or $\mathfrak{R}_4$ is to first provide the surveillance, assistance service to the ground vehicles and then another surveillance service in this sequence to regions required. Namely, $\varphi_1^s = \Diamond(\sigma_{12} \wedge \Diamond(\sigma_{11} \wedge \Diamond\sigma_{12}))$ and $\varphi_4^s = \Diamond(\sigma_{42} \wedge \Diamond(\sigma_{41} \wedge \Diamond\sigma_{42}))$. On the other hand, the finite-time local task for robot $\mathfrak{R}_2$ or $\mathfrak{R}_3$ is to first deliver food or water and then provide transportation service, which is formalized as $\varphi_2^s = \Diamond(\sigma_{21} \vee \sigma_{22}) \wedge \Diamond\sigma_{23}$ and $\varphi_3^s = \Diamond(\sigma_{31} \vee \sigma_{32}) \wedge \Diamond\sigma_{33}$.

The synthesized discrete plans derived by the algorithm described in Section 6.1.2 are as follows: robot $\mathfrak{R}_1$ needs to provide surveillance at region $\pi_{1\mathtt{bl}}$, assistance at region $\pi_{1\mathtt{tl}}$ and then surveillance at region $\pi_{1\mathtt{bl}}$, i.e., $\tau_1 = (\pi_{1\mathtt{bl}}, \{\sigma_{12}\})(\pi_{1\mathtt{tl}}, \{\sigma_{11}\})(\pi_{1\mathtt{bl}}, \{\sigma_{12}\})$; robot $\mathfrak{R}_2$ would supply food at region $\pi_{2\mathtt{tl}}$ and transportation at region $\pi_{2\mathtt{bl}}$ while robot $\mathfrak{R}_3$ would supply food at region $\pi_{3\mathtt{tl}}$ and transportation at region $\pi_{3\mathtt{bl}}$. Namely, $\tau_2 = (\pi_{2\mathtt{tl}}, \{\sigma_{21}\})(\pi_{2\mathtt{bl}}, \{\sigma_{23}\})$ and $\tau_3 = (\pi_{3\mathtt{tr}}, \{\sigma_{31}\})(\pi_{3\mathtt{br}}, \{\sigma_{33}\})$; At last, robot $\mathfrak{R}_4$ needs to provide surveillance at region $\pi_{4\mathtt{bl}}$, assistance at region $\pi_{4\mathtt{tl}}$ and then surveillance at region $\pi_{4\mathtt{bl}}$, i.e., $\tau_4 = (\pi_{4\mathtt{br}}, \{\sigma_{41}\})(\pi_{4\mathtt{tr}}, \{\sigma_{42}\})(\pi_{4\mathtt{tl}}, \{\sigma_{41}\})$. It can be verified that they all satisfy the respective local tasks. At $t = 0$, the switching policy $\mathbf{P_{sc}}$ from Section 6.1.4 is applied. It takes around $9s$ for all robots to accomplish the execution of their local plans. The complete robot trajectories are shown in Figure 6.1a, where the distances between the neighboring robots along with times of reaching the robots' respective progressive goal regions are also plotted. In addition, the time instants when each robot reaches their goal regions are shown to illustrate the progressive plan execution.

(II) ***General LTL task specifications***. In this case, all robots' local tasks are specified as general LTL formulas over the services. The task of robot $\mathfrak{R}_1$ is to periodically provide both the surveillance and assistance services σ_{11} and σ_{12} at the required regions, which is represented by $\phi_1 = \Box \Diamond \sigma_{11} \wedge \Box \Diamond \sigma_{12}$; the task of robots $\mathfrak{R}_2$ and $\mathfrak{R}_3$ are similar, which is to periodically provide either food, water supply or transportation service at desired regions, which is formalized



(a) (b) (c)

Figure 6.1: (a) Top: robots' respective regions of interest in red, green, blue and cyan respectively and their trajectories under policy $\mathbf{P_{sc}}$. All robots accomplish their sc-LTL tasks after $9s$. Middle: the evolution of pair-wise distances $\|x_{12}\|, \|x_{23}\|, \|x_{34}\|$, which all satisfy the distance constraints (below $7.5m$). Bottom: the time instants when the robots reach their goal regions and provide the set of planned services. (b) robots' trajectories under policy $\mathbf{P_{ge}}$ with general LTL tasks. (c) robots' trajectories under policy $\mathbf{P_{ge}}$ with general LTL tasks, after incorporating the plan adaptation algorithm. The bottom figure can be compared with Figure 6.1b).

as $\phi_2 = \square \diamondsuit (\sigma_{21} \vee \sigma_{22} \vee \sigma_{23})$ and $\phi_3 = \square \diamondsuit (\sigma_{31} \vee \sigma_{32} \vee \sigma_{33})$; at last, the task of robot $\mathfrak{R}_4$ is to periodically provide both the surveillance and assistance services at the required regions, which is represented by $\phi_4 = \square \diamondsuit \sigma_{41} \wedge \square \diamondsuit \sigma_{42}$.

The synthesized discrete plans from Section 6.1.2 are as follows: robot $\mathfrak{R}_1$ would provide assistance at region $\pi_{\mathtt{1b1}}$ and then surveillance at region $\pi_{\mathtt{1t1}}$, which is repeated infinitely often, i.e., $\tau_1 = \big((\pi_{\mathtt{1b1}}, \{\sigma_{12}\})\big((\pi_{\mathtt{1t1}}, \{\sigma_{11}\})\big)\big)^{\omega}$; robot $\mathfrak{R}_2$ would supply food at region $\pi_{\mathtt{2t1}}$ repetitively and robot $\mathfrak{R}_3$ would provide transportation service at region $\pi_{\mathtt{3b1}}$ repetitively. Namely, $\tau_2 = (\pi_{\mathtt{2t1}}, \{\sigma_{21}\})^{\omega}$ and $\tau_3 = (\pi_{\mathtt{3b1}}, \{\sigma_{33}\})^{\omega}$; at last, robot $\mathfrak{R}_4$ would provide surveillance at region $\pi_{\mathtt{4br}}$ and then assistance at region $\pi_{\mathtt{4tr}}$, which is repeated infinitely often, i.e., $\tau_4 = \big((\pi_{\mathtt{4br}}, \{\sigma_{41}\})(\pi_{\mathtt{4tr}}, \{\sigma_{42}\})\big)^{\omega}$.

The simulation results for the activity switching protocol $\mathbf{P_{ge}}$ from Section 6.1.4 are illustrated in Figure 6.1b. The functions f_{prob} and f_{cond} were chosen by $\mathbf{Pr}(b_i = 1) = e^{-\alpha_i \Upsilon_i[i](t-\chi_i)}$ if $\Upsilon_i[i] \cdot (t - \chi_i) < \bar{\chi}_i$; and $\mathbf{Pr}(b_i = 1) = 0$ if $\Upsilon_i[i] \cdot (t - \chi_i) \geq \bar{\chi}_i$, where $\bar{\chi}_i = 5$ and $\alpha_i = 1$. The probability of remaining active decreases with the increasing time elapsed since the current round started and with the increasing

number robot $\mathfrak{R}_i$'s own progressive goal region was visited. Note that there exists a finite $T \in (T_{\circlearrowleft_{m-1}}, T_{\circlearrowleft_m})$, such that $\Upsilon_i[i] \cdot (t - \chi_i) \geq \bar{\chi}_i$ for all $t \in [T, T_{\circlearrowleft_m}]$, hence each robot $\mathfrak{R}_i$ is guaranteed to be switched to passive control mode eventually.

At last, to demonstrate the effectiveness of the local plan adaptation technique proposed in Section 6.1.5, we combine the switching protocol $\mathbf{P_{ge}}$ and the real-time adaptation algorithm. The results are shown in Figure 6.1c, which is significantly different from the results in Figure 6.1b: Robot $\mathfrak{R}_4$ adapts its plan to visit $\pi_{4\mathtt{t1}}$ with service σ_{42} while robot $\mathfrak{R}_1$ is reaching $\pi_{1\mathtt{t1}}$ with service σ_{11}. Then robot $\mathfrak{R}_4$ adapts its plan to visit region $\pi_{4\mathtt{b1}}$ and provide the surveillance service there and robot $\mathfrak{R}_2$ adapts its plan to visit region $\pi_{2\mathtt{b1}}$ and supply water there while robot $\mathfrak{R}_1$ is reaching the region $\pi_{1\mathtt{b1}}$. Consequently, the robots reach their goal regions much more often than before when the real-time adaptation algorithm is not applied, which can be confirmed by comparing from the time instants when each robot reaches its goal region in Figures 6.1b and 6.1c, respectively.

6.2 EGGs-based hybrid control

In this section, we consider again a team of robots but with the unicycle dynamics, each of which is given a local LTL task as the desired motion and actions. Besides, dynamic constraints with neighboring robots are addressed, including the inter-robot collision avoidance and connectivity maintenance of the communication network. We propose a distributed and cooperative motion and action control scheme which integrates two main components: Embedded Graph Grammars (EGGs) to specify the local interaction rules and switching control modes of the robots; and the discrete plan synthesis for desired motion and actions. It is ensured that all local tasks are satisfied while the dynamic constraints are obeyed at all time.

6.2.1 Problem formulation

Consider a team of N autonomous robots with identities $i \in \mathcal{N} = \{1, 2, \cdots, N\}$. Now each robot $i \in \mathcal{N}$ satisfies the unicycle dynamics [97]:

$$\dot{x}_i = v_i \cos(\theta_i), \quad \dot{y}_i = v_i \sin(\theta_i), \quad \dot{\theta}_i = w_i, \tag{6.26}$$

where $s_i = (x_i, y_i, \theta_i) \in \mathbb{R}^3$ is the continuous state of robot i containing its coordinates $p_i = (x_i, y_i)$ and orientation θ_i; and $u_i = (v_i, w_i) \in \mathbb{R}^2$ is the control input as linear and angular velocities, which are bounded by $v_{\max}$ and $w_{\max}$. Moreover, robot i has a reference linear velocity $V_i < v_{\max}$ and a reference angular velocity $W_i < w_{\max}$. Each robot occupies a disk area of $\{p \in \mathbb{R}^2 \,|\, \|p - p_i\| \leq r\}$, where $r > 0$ is the radius of its physical volume. A *safety distance* $\underline{d} > 2r$ is predefined as the allowed minimal inter-robot distance to avoid collisions. Each robot $i \in \mathcal{N}$ can only communicate with another robot $j \in \mathcal{N}$ if $\|p_i - p_j\| < \overline{d}$, where $\overline{d} > \underline{d}$ is the predefined communication radius.

Definition 6.4. Robots $i, j \in \mathcal{N}$ are: in **collision** if $\|p_i(t) - p_j(t)\| \leq \underline{d}$; **neighbors** if $\|p_i(t) - p_j(t)\| \leq \overline{d}$. ▲

Given the robot states, an embedded graph $\gamma(t)$ is defined as $\gamma(t) = (G(t), p(t))$, where $G(t) = (\mathcal{N}, E(t))$ with $(i, j) \in E(t)$ if $\|p_i(t) - p_j(t)\| < \overline{d}$, $\forall i, j \in \mathcal{N}$, $i \neq j$; $p(t)$ is the stack vector of all $p_i(t)$. Then we define the set of *allowed* embedded graphs Γ_d as follows:

Definition 6.5. An embedded graph $\gamma(t) = (G(t), p(t))$ is allowed that $\gamma(t) \in \Gamma_d$ if (i) $\|p_i(t) - p_j(t)\| > \underline{d}$, $\forall i, j \in \mathcal{N}$, $i \neq j$; (ii) the graph $G(t)$ is connected. ▲

Embedded Graph Grammars (EGGs)

Here we review the basics of EGGs. For a detailed description, we refer the readers to [113, 114]. Let Σ be a set of pre-defined labels. A labeled graph is defined as the quadruple $G = (V, E, l, e)$, where V is a set of vertices, $E \subset V \times V$ is a set of edges, $l : V \to \Sigma$ is a vertex labeling function, and $e : E \to \Sigma$ is an edge labeling function. Then consider a labeled graph consisting of N robots as its nodes with identical continuous state space X. Then an embedded graph is given by $\gamma = (G, x)$, where G is a labeled graph and $x : V \to X$ is a realization function. We use G_γ, x_γ to denote the labeled graph and continuous states associated with γ. The set of all allowed embedded graphs is denoted by Γ. Furthermore, an embedded graph transition is a relation $\mathcal{A} \subset \Gamma \times \Gamma$ such that $(\gamma_1, \gamma_2) \in \mathcal{A}$ implies $x_{\gamma_1} = x_{\gamma_2}$ and $G_{\gamma_1} \neq G_{\gamma_2}$. The associated rules and conditions are called grammars.

Task specification

For each robot $i \in \mathcal{N}$, there is a set of points of interest in the workspace, denoted by $Z_i = \{z_{i1}, z_{i2}, \cdots, z_{iM_i}\}$, where $z_{i\ell} \in \mathbb{R}^2$, $\forall \ell = 1, 2, \cdots, M_i$, where $M_i > 0$. Each point satisfies different properties. Furthermore, it is capable of performing a set of actions, described by the action primitives $\Sigma_i = \{a_1, a_2, \cdots, a_{K_i}\}$. Each action has conditions on the workspace property that should be satisfied to perform it and also an effect on the workspace after performing it. By Section 5.1.1, we can derive a complete motion and action model for robot i as a wFTS $\mathcal{R}_i = (\Pi_i, \longrightarrow_i, \Pi_{i,0}, AP_i, L_i, W_i)$, where $\Pi_i = Z_i \times \Sigma_i$ is the set of states; AP_i is the set of atomic propositions of interest and the other notations are similar to (5.5). Moreover, the local task for each robot $i \in \mathcal{N}$ is specified as an LTL formula φ_i over AP_i, with syntax and semantics described in Section 3.2. Particularly, the satisfaction of task φ_i is defined as follows:

Definition 6.6. The task φ_i is satisfied if there *exists* a sequence of time instants $t_{i0}\, t_{i1}\, t_{i2} \cdots$ and a sequence of states $\pi_{i\ell_0} \pi_{i\ell_1} \pi_{i\ell_2} \cdots$ such that: $\pi_{i\ell_k} = (z_{i\ell_k}, a_{i\ell_k})$ where $z_{i\ell_k} \in Z_i$ and $a_{i\ell_k} \in \Sigma_i$, $\|p_i(t_{ik}) - z_{i\ell_k}\| \leq c_i$ where $c_i > 0$ is a given threshold for reaching a point of interest and the action $a_{i\ell_k}$ is performed at $z_{i\ell_k}$, $\forall k = 0, 1, 2, \cdots$; and $L_i(\pi_{i\ell_0}) L_i(\pi_{i\ell_1}) L_i(\pi_{i\ell_2}) \cdots \vDash \varphi_i$. ▲

Some examples of the task specification are: "Infinitely often pick up object `A` in point 1 and then drop it to point 2"; "Surveil points 3 and 4 by taking pictures there"; "Go to point 5 and operate machine `M`, then go to point 5 and charge the battery", which all involve motion and actions.

Problem 6.3. *Design a distributed motion control scheme such that φ_i is satisfied, $\forall i \in \mathcal{N}$, while at the same time $\gamma(t) \in \Gamma_d$, $\forall t \geq 0$.* ▲

The proposed solution consists of two major parts: the embedded graph grammars (EGGs) design and the local task coordination, of which the details are given in the sequel. Then we combine them as the complete hybrid control structure, where we also prove the correctness formally.

6.2.2 EGGs design

The design of EGGs involves three components: (i) the workspace discretization, (ii) the graph transition rules, and (iii) the associated control modes.

Workspace discretization

The 2-D workspace is discretized into uniform grids by a quantization function, through which we transform the collision avoidance and connectivity constraints into relative grids positions.

Definition 6.7. Given a point $(x, y) \in \mathbb{R}^2$, its grid position is given by the function $\textsc{Grid} : \mathbb{R}^2 \to \mathbb{Z}^2$:

$$(g_x, g_y) \triangleq \textsc{Grid}(x, y) \triangleq ([\frac{x}{\underline{d}}], [\frac{y}{\underline{d}}]), \tag{6.27}$$

where $[\cdot]$ is the *round* function that returns the closest integer ($[0.5] = 1$) and $\underline{d}$ is the safety distance introduced earlier. ▲

Given that $p_i(t) = (x_i(t), y_i(t))$ at time $t > 0$, the grid position of robot i is given by $g_i(t) \triangleq (g_i^x(t), g_i^y(t)) = \textsc{Grid}(x_i(t), y_i(t))$. Now consider two robots i and j whose grid positions are given by $g_i(t)$ and $g_j(t)$.

Definition 6.8. The collision function $\textsc{Collide} : \mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{B}$ satisfies:

$$\textsc{Collide}(g_i(t), g_j(t)) \triangleq \bot,$$

if it holds that $|g_i^x - g_j^x| \geq 2$ or $|g_i^y - g_j^y| \geq 2$; otherwise, $\textsc{Collide}(g_i(t), g_j(t)) \triangleq \top$. The neighboring function $\textsc{Neighbor} : \mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{B}$ satisfies:

$$\textsc{Neighbor}(g_i(t), g_j(t)) \triangleq \top,$$

if it holds that $\|(|g_i^x - g_j^x| + 1, |g_i^y - g_j^y| + 1)\| \leq \lambda_d$, where $\lambda_d \triangleq \overline{d}/\underline{d} > 1$; otherwise, $\textsc{Neighbor}(g_i(t), g_j(t)) \triangleq \bot$. ▲

Lemma 6.16. *By Definition 6.4 robots i and j are collision-free at time $t > 0$ if* $\textsc{Collide}(g_i(t), g_j(t)) = \bot$*; they are connected if* $\textsc{Neighbor}(g_i(t), g_j(t)) = \top$.

Proof. For $p_i(t), p_j(t) \in \mathbb{R}^2$, by (6.27) it holds that if $|g_i^x(t) - g_j^x(t)| \geq 2$, then $|x_i - x_j| > \underline{d}$ and $\|p_i(t) - p_j(t)\| \geq |x_i - x_j| + |y_i - y_j| > \underline{d}$, i.e., robots i and j are collision-free by Definition 6.4. The same arguments hold when $|g_i^y - g_j^y| \geq 2$. For any $p_i(t), p_j(t) \in \mathbb{R}^2$, by (6.27) it holds that $|x_i - x_j| < \underline{d} \cdot (|g_i^x - g_j^x| + 1)$ and $|y_i - y_j| < \underline{d} \cdot (|g_i^y - g_j^y| + 1)$. Then $\|p_i(t) - p_j(t)\| = \|(x_i - x_j, y_i - y_j)\| < \underline{d} \cdot \|(|g_i^x - g_j^x| + 1, |g_i^y - g_j^y| + 1)\| < \underline{d} \cdot \lambda_d = \overline{d}$, implying that robots i and j are neighbors by Definition 6.4. ■

Building blocks

We first introduce the set of labels for the embedded graph. Then we present some building blocks for the graph transition rules. At last, we state the set of transition rules that are executed locally.

(I) ***Labels on vertices and edges***. The *first* building block is the modified embedded graph $\gamma(t) = (G(t), p(t))$ where $G(t) = (\mathcal{N}, E(t), l, e)$, where l and e are the vertex and edge labeling functions. Each vertex has a label with three named fields $\{\texttt{id}, \texttt{mode}, \texttt{data}\}$, where `id` is the robot ID; `mode` is the robot status, including $\{\texttt{check}, \texttt{static}, \texttt{move}\}$; and `data` stores data for the execution, which has three sub-fields $\{\texttt{nb}, \texttt{pt}, \texttt{gi}\}$, where `nb` saves the a set of other robots' IDs; `pt` saves a tentative path; and `gi` saves a positive gain parameter. Moreover, the edge between neighbors has the named field `id`, i.e., the edge from robot i to j has the ID (i, j). For brevity, we omit the definitions of l and e that map $\mathcal{N}$ and $E(t)$ to the set of labels, which is a cartesian product of the named fields defined above. We use dot notation to indicate the value of label fields. For instance, "$i.\texttt{mode} = \texttt{move}$" means that robot i has the mode being `move`. We call an robot *static* if its mode is `static` and *active* if its mode is `move`.

To start with, we need the notion of a local sub-graph for robot $i \in \mathcal{N}$, denoted by $G_i(t) = (V_i(t), E_i(t))$, where (i) $V_i(t) = \{i\} \cup \mathcal{N}_i(t)$, where $\mathcal{N}_i(t) = \{j \in \mathcal{N} \mid (i, j) \in E(t)\}$; (ii) $(j, k) \in E_i(t)$ if $(j, k) \in E(t)$, $\forall j, k \in V_i(t)$. Clearly, $G_i(t)$ is a sub-graph of $G(t)$ and it can be constructed locally by robot i. Clearly if $G(t)$ is connected, then $G_i(t)$ is connected, $\forall i \in \mathcal{N}$.

(II) ***Neighbor marking scheme***. The *second* building block is the mechanism to maintain graph connectivity while the robots are moving. The main idea is to choose locally some robots to be static and the others be active; and more importantly ensure that the active robots remain connected to its static neighbors while moving. The most straightforward way is to allow only one robot move at a time, which is extremely inefficient as a system. Here we propose a local marking scheme to choose static and active robots, which allows more robots being active simultaneously. Assume that robot $i \in \mathcal{N}$ satisfies $i.\texttt{mode} = \texttt{active}$. Given its local graph $G_i(t)$ at time $t > 0$, robot i can communicate with its neighbor $j \in \mathcal{N}_i(t)$ regarding its `mode`. We denote by $\mathcal{N}_i^s(t) = \{j \in \mathcal{N}_i(t) \mid j.\texttt{mode} = \texttt{static}\}$ the set of static neighbors; $\mathcal{N}_i^a(t) = \{j \in \mathcal{N}_i(t) \mid j.\texttt{mode} = \texttt{move}\}$ the set of active neighbors; and

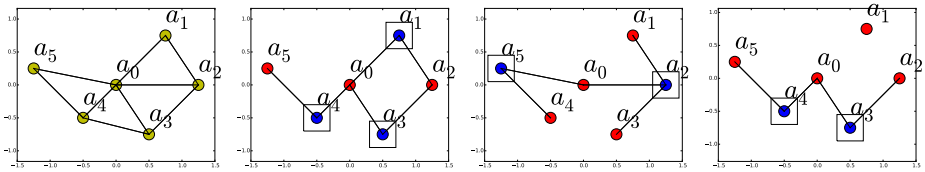


Figure 6.2: Examples of marking schemes for robot a_0 locally: (a) Local graph G_0, consisting of neighbors a_1, a_2, a_3, a_4, a_5; (b) one allowed marking scheme where a_1, a_3, a_4 are marked, with the associated marked sub-graph of G_0^m,G_2^m,G_5^m; (c) another allowed marking scheme where a_2, a_5 are marked; (d) an *not*-allowed marking scheme where a_3, a_4 are marked as a_1 is neither marked nor connected to a marked robot.

the others are in the `check` mode. A marking scheme of robot i at time $t > 0$ marks a subset of its neighbors, denoted by $\mathcal{N}_i^m(t) \subseteq \mathcal{N}_i(t)$, as the potential robots to become static. Then a marking scheme should satisfy the following:

Definition 6.9. The marked set of neighbors $\mathcal{N}_i^m(t)$ is *allowed* if: (i) for any neighbor $j \in \mathcal{N}_i(t)$, it holds that either $j \in \mathcal{N}_i^m(t)$ or there exists $g \in \mathcal{N}_i^m(t)$ that $(j, g) \in E_i(t)$; (ii) $\mathcal{N}_i^s(t) \subseteq \mathcal{N}_i^m(t)$ and $\mathcal{N}_i^m(t) \cap \mathcal{N}_i^a(t) = \varnothing$. ▲

The first condition requires that any neighbor is either marked or *directly* connected to a marked robot, while the second condition says that all static and no active neighbors should be marked. Examples of different marked schemes are shown in Figure 6.2. Given the set of marked robots $\mathcal{N}_i^m(t) \subseteq \mathcal{N}_i(t)$, the *marked* sub-graph of $G_i(t)$ is defined as:

Definition 6.10. The marked sub-graph $G_i^m(t) \triangleq (V_i^m(t), E_i^m(t))$ has $V_i^m(t) = \{i\} \cup \mathcal{N}_i^m(t)$ and $(j, k) \in E_i^m(t)$ if $(j, k) \in E_i(t)$, $\forall j, k \in V_i^m$. ■

(III) *Potential path synthesis*. The *third* building block is the synthesis algorithm to derive a local path for an active robot $i \in \mathcal{N}$ to move towards its point of interest $z_{i\ell} = (z_{i\ell}^x, z_{i\ell}^y) \in Z_i$ while keeping connected and collision-free to all its marked neighbors in $\mathcal{N}_i^m$. Denote by $\mathbf{p}_i$ the *tentative* discrete path of robot i with length $L_i \geq 1$ that obeys the following structure:

$$\mathbf{p}_i = q_i^0\, q_i^1 \cdots q_i^l \cdots q_i^{L_i} \tag{6.28}$$

where $q_i^l = (s_i^l, t_i^l, v_i^l)$ is a 3-tuple with the desired state $s_i^l = (x_i^l, y_i^l, \theta_i^l) \in \mathbb{R}^3$, the approximated time t_i^l when s_i^l will be reached, and the linear velocity v_i^l at q_i^l when heading towards q_i^{l+1}, $\forall l = 0, 1, \cdots, L_i$. Notice that $q_i^0 \triangleq (s_i(t), 0, V_i)$ initially, where V_i is the reference linear velocity. Moreover, the position $p_i^l = (x_i^l, y_i^l)$ of s_i^l should correspond to the center of a grid $g_i^l = \textsc{Grid}(p_i^l)$ and two consecutive positions p_i^l, p_i^{l+1} correspond to two adjacent grids, $\forall l = 0, 1, \cdots, L_i - 1$. Given the current state $s_i(t)$ of robot i, the potential *cost* of $\mathbf{p}_i$ is defined as:

$$\textsc{Cost}(\mathbf{p}_i) \triangleq \sum_{l=0}^{L_i-1} \left(\|p_i^l - p_i^{l+1}\| + \alpha \cdot |\theta_i^l - \theta_i^{l+1}| \right), \tag{6.29}$$

where the first term is the total traveled distance and the second term is the total turned angles; $\alpha > 0$ is the chosen weight on turning cost. To synthesize the tentative path $\mathbf{p}_i$, we consider the following optimization problem:

$$\begin{aligned} \mathbf{min}_{\mathbf{p}_i} \quad & \|(p_{ix}^{L_i} - z_{i\ell}^x,\, p_{iy}^{L_i} - z_{i\ell}^y)\| + \beta \cdot \textsc{Cost}(\mathbf{p}_i) \\ \mathbf{s.t.} \quad & G_i^m(t) \text{ remains connected if } p_i = p_i^l, \\ & \textsc{Collide}(g_i^l,\, g_j(t)) = \bot, \\ & \forall l = 0, 1, \cdots, L_i \text{ and } \forall j \in \mathcal{N}_i^m(t), \end{aligned} \tag{6.30}$$

where the first term is the tentative progress as the distance from $p_i^{L_i} = (p_{ix}^{L_i}, p_{iy}^{L_i})$ to $(z_{i\ell}^x,\, z_{i\ell}^y)$; and $\beta > 0$ is a tuning parameter; along $\mathbf{p}_i$ robot i should remain connected and collision-free to all robots in G_i^m.

The above problem can be solved in four steps: (i) determine the general search area. Given the positions of the marked robots, the general search area $\mathcal{S}_i \subset \mathbb{Z}^2$ satisfies that $g_s = (g_s^x, g_s^y) \in \mathcal{S}_i$ if $\textsc{Neighbor}(g_b,\, g_j(t)) = \top$, for at least one neighbor $j \in \mathcal{N}_i^m(t)$; (ii) remove any grid $g_s \in \mathcal{S}_i$ that $G_i^m(t)$ is not connected if $g_i = g_s$ or $\textsc{Collide}(g_s,\, g_j(t)) = \top$ for any neighbor $j \in \mathcal{N}_i^m(t)$. Thus all elements of $\mathbf{p}_i$ should belong to this general search area; (iii) the augmented-graph construction. Construct a graph $\Xi = (\mathbf{n}_i,\, \mathbf{e}_i,\, \mathbf{w}_i)$ where $\mathbf{n}_i = \mathcal{S}_i \times \{0, \pm\frac{\pi}{2}, \boldsymbol{\pi}\}$ is the set of nodes; $\mathbf{e}_i \subset \mathcal{S}_i \times \mathcal{S}_i$ is the set of edges $(n_1,\, n_2) \in e_i$ if $n_1 = (g_1,\, \theta_1)$, $n_1 = (g_2,\, \theta_2)$ where the grids g_1 and g_2 are adjacent; $\mathbf{w}_i : e_i \to \mathbb{R}^+$ is the weighting function, where $\mathbf{w}_i((g_1,\, \theta_1),\, (g_2,\, \theta_2)) = \underline{d} + \alpha \cdot |\theta_1 - \theta_2|$, where α is defined in (6.29); (iv) shortest path search. Firstly, locate the initial node $n_0 = (g_0,\, \theta_0) \in \mathbf{n}_i$ that is closest to the current state $s_i(t)$. Then construct the shortest paths from n_0 to every other node in $\mathbf{n}_i$ by Dijkstra's algorithm. At last, find the destination $n_d^\star \in \mathbf{n}_i$ that minimizes the cost in (6.30). Denote the shortest path from n_0 to $n_d^\star$ by $\mathbf{p}_i^\Xi = n_0 n_1 n_2 \cdots n_{L_i - 1} n_d^\star$, where $n_l = (g_l,\, \theta_l) \in \mathbf{n}_i$ and L_i is the total length. An example is shown in Figure 6.3.

Give the shortest path $\mathbf{p}_i^\Xi$, the element $q_i^l = (s_i^l,\, t_i^l,\, v_i^l)$ of $\mathbf{p}_i$ can be derived by setting $s_i^l = (g_l^x \cdot \underline{d},\, g_l^y \cdot \underline{d},\, \theta_l)$ and $v_i^l = V_i$, $\forall l = 0, 1, \cdots, L_i$, and t_i^l is computed by:

$$t_i^{l+1} = t_i^l + \frac{\underline{d}}{v_i^l} + \frac{|\theta_i^{l+1} - \theta_i^l|}{W_i}, \quad \forall l = 1, 2, \cdots, L_i, \tag{6.31}$$

which accumulates the time for robot i to move from s_i^l to s_i^{l+1} with linear velocity v_i^l and angular velocity W_i. If a solution to (6.30) exists, the resulting $\mathbf{p}_i$ is the tentative path of robot i with the associated marked set $\mathcal{N}_i^m$. Moreover, its tentative *gain* is given by $\chi_i = \|p_i^{L_i} - z_{i\ell}\| - \|p_i(t) - z_{i\ell}\|$. For the ease of notation, we denote this local path synthesis procedure by a single function:

$$(\mathbf{p}_i,\, \chi_i) = \textsc{Check}(s_i(t),\, \mathcal{N}_i(t),\, z_{i\ell},\, \mathcal{N}_i^m). \tag{6.32}$$

As a result, robot i *executes* $\mathbf{p}_i$ by following and staying within the sequence of grids along $\mathbf{p}_i$. More details are given in the fifth block.

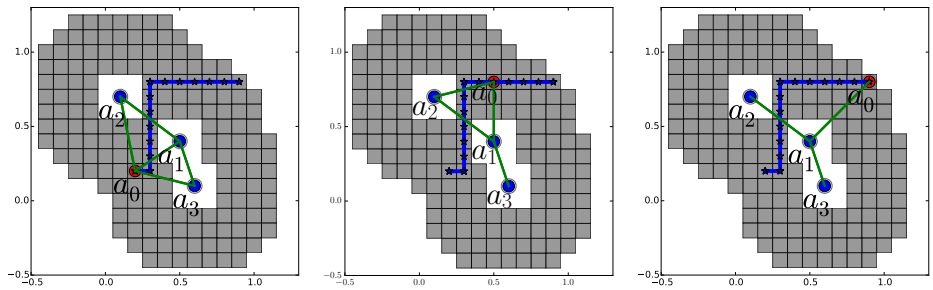


Figure 6.3: Gray grids indicate the allowed search area. The blue star-marked path is the optimal discrete path $\mathbf{p}_0$ by (6.30) for robot a_0, given its marked neighbors a_1, a_2, a_3 and its goal. Notice the change of graph topology $G_0^m(t)$ and the fact that it remains connected while a_0 moves along $\mathbf{p}_0$.

Lemma 6.17. *Assume that* (6.30) *has a solution at time* $t_0 > 0$. *If all marked neighbors in* $\mathcal{N}_i^m$ *remain static and robot* i *executes* $\mathbf{p}_i$ *until* $t_1 > t_0$, *then* $G_i^m(t)$ *remains connected and all robots within* $V_i^m(t)$ *are collision-free,* $\forall t \in [t_0, t_1]$.

Proof. Since all marked neighbors in $\mathcal{N}_i^m$ stay static, robot i is the only moving robot within V_i^m. Initially $G_i^m(t_0)$ is connected and any two robots within V_i^m are collision-free. While robot i executes $\mathbf{p}_i$, the formulation of (6.30) ensures that $G_i^m(t)$ remains connected and robot i is collision-free with any marked neighbor. This holds until robot i finishes executing $\mathbf{p}_i$ by reaching $q_i^{L_i}$ at time t_1. ■

(IV) ***Path adaptation***. The *fourth* building block is the path adaptation algorithm for any active robot while executing its tentative path. Assume that at time $t > 0$ an active robot i may detect another robot $j \in \mathcal{N}$ that does not belong to $\mathcal{N}_i^m$, when its state $s_i(t)$ corresponds to $q_i^{w_0} \in \mathbf{p}_i$ in (6.28), where $0 < w_0 < L_i$. We consider the two cases:

If j.`mode` = `static`, then robot i only needs to check if its future path segment is in collision with this static robot j. Its future path segment is given by $\mathbf{p}_i[w_0{:}L_i] = q_i^{w_0} q_i^{w_0+1} \cdots q_i^{L_i}$, where $q_i^l = (s_i^l, t_i^l, v_i^l)$ is defined in (6.28). Therefore if $\textsc{Collide}(g_i^w, g_j(t)) = \perp$, $\forall w = w_0, w_0+1, \cdots, L_i$, it means they will *not* collide and $\mathbf{p}_i$ remains unchanged; otherwise, $\mathbf{p}_i$ is adapted by repeating the synthesis procedure by (6.32), but with the new neighboring set $\mathcal{N}_i(t)$.

If j.`mode` = `move`, then robot j is also moving and executing its path $\mathbf{p}_j$. In this case, it is more complicated to check whether they will be in collision. To begin with, we assume that robot j's position $s_j(t)$ corresponds to $q_j^{v_0} \in \mathbf{p}_j$, where $0 < v_0 < L_j$. Its future path segment is given by $\mathbf{p}_j[v_0{:}L_j] = q_j^{v_0} q_j^{v_0+1} \cdots q_j^{L_j}$, where $q_j^l = (s_j^l, t_j^l, v_j^l)$ from (6.28). Given $\mathbf{p}_i[w_0{:}L_i]$ and $\mathbf{p}_j[v_0{:}L_j]$, a potential collision between robots i and j can be detected by the function:

$$\textsc{CollidePath}(\mathbf{p}_i, \mathbf{p}_j) = \perp. \tag{6.33}$$

if $\textsc{Collide}(p_i^w, p_j^v) = \bot$ and $|t_i^w - t_j^v| < \Delta_t$, for any $p_i^w \in \mathbf{p}_i[w_0{:}L_i]$ and any $p_j^v \in \mathbf{p}_j[v_0{:}L_j]$, where $\Delta_t > 0$ is a design parameter as the allowed time difference, which depends on the estimation accuracy of the time sequences $\{t_i^w\}$ and $\{t_j^v\}$ by (6.31). Then robots i and j keep their current paths unchanged; otherwise, $\textsc{CollidePath}(\mathbf{p}_i, \mathbf{p}_j) = \top$, meaning that they may collide by executing their respective paths. Thus at least one of them should modify its current path, the choice of which robot will be presented later. For now, we assume that robot i is chosen to change its path $\mathbf{p}_i$. Let $w_c \in \{w_0, w_0+1, \cdots, L_i\}$ be the *smallest* index within $\mathbf{p}_i[w_0{:}L_i]$ that a potential collision could happen by (6.33) and the associated index within $\mathbf{p}_j[v_0{:}L_j]$ is $v_c \in \{v_0, v_0+1, \cdots, L_j\}$. Then robot i would avoid this collision by reducing its speed within the segment $\mathbf{p}_i[w_0{:}w_c]$, while $\mathbf{p}_i[w_c{:}L_i]$ remains unchanged. To find a suitable linear velocity $\nu < v_{\max}$ for elements in $\mathbf{p}_i[w_0{:}w_c]$, we consider the following optimization problem:

$$\begin{aligned} &\mathbf{min}_{0<\nu<v_{\max}} \quad |V_i - \nu| \\ &\mathbf{s.t.} \quad v_i^l = \nu,\ \forall l = w_0, \cdots, w_c. \\ &\qquad \textsc{Collide}(p_i^w, p_s^v) = \bot, \text{ and } |t_i^w - t_s^v| < \Delta_t, \\ &\qquad \forall p_i^w \in \mathbf{p}_i[w_0{:}L_i],\ \forall p_j^v \in \mathbf{p}_j[v_0{:}L_j]. \end{aligned} \tag{6.34}$$

where V_i is the reference velocity. The conditions above ensure that after adjusting the linear velocity, $\mathbf{p}_i$ and $\mathbf{p}_j$ will not collide by (6.33). The above problem can be solved as follows: firstly, choose $\nu = \max_{l\in[w_0:w_c]}\{v_i^l\}$ and a proper step size $\delta_v > 0$. Then gradually decrease ν by δ_v and check if the conditions within (6.34) are fulfilled. If not, repeat this procedure until $\nu = \nu^\star$ is small enough and all conditions within (6.34) are fulfilled. As a result, $\nu^\star$ is the suitable linear velocity for $\mathbf{p}_i[w_0{:}w_c]$. Moreover, the time instants $\{v_i^w\}$ within $\mathbf{p}_i[w_0{:}L_i]$ are updated according to (6.31). If $\nu < 0$ and no solution can be found, it means that the initial position of robot i is in collision with parts of robot j's path.

Now consider that while executing the adjusted path, robot i may meet with another moving robot, say $k \in \mathcal{N}_i(t_1)$ at time $t_1 > 0$. Now its corresponding index within $\mathbf{p}_i$ is $w_0' > w_0$. Similar as before, robots i and k exchange their paths $\mathbf{p}_i$ and $\mathbf{p}_k$. Function $\textsc{CollidePath}(\mathbf{p}_i, \mathbf{p}_k)$ can be used to check if they will collide in the future. If so, assume that robot i is chosen to adapt its path again and the potential collision is estimated to happen at index w_c' of $\mathbf{p}_i$. Consider the relative position of $q_i^{w_c'}$ and $q_i^{w_c}$ from the previous adjustment: (i) if $w_c' \le w_c$, robot i would reduce its linear velocity within $\mathbf{p}_i[w_0'{:}w_c']$ by the same formulation as (6.34); (ii) if $w_c' > w_c$, robot i would instead reduce its linear velocity within $\mathbf{p}_i[w_c{:}w_c']$ by the same formulation as (6.34). For the ease of notation, we denote this process of adjusting linear velocity by a single function:

$$\mathbf{p}_i = \textsc{SlowDown}(s_i(t), \mathbf{p}_i, \mathbf{p}_j), \tag{6.35}$$

which is only applied to the robot that adapts its path. Figure 6.4 shows an example of applying the above function. Note that the sequence of $\{s_i^l\}$ within $\mathbf{p}_i$ remains unchanged and the collision is avoided by adjusting only the velocity.

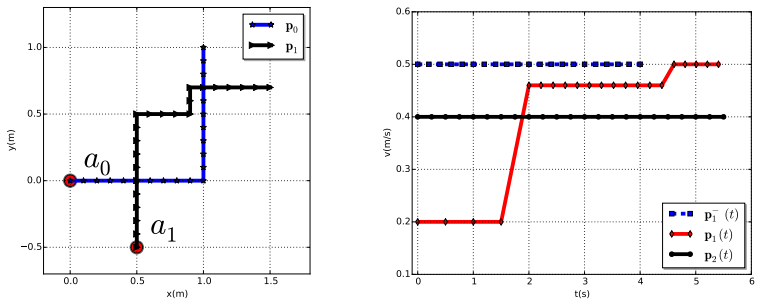


Figure 6.4: The left image shows that robots a_0, a_1 have a potential collision given their paths $\mathbf{p}_1, \mathbf{p}_2$ with velocities $0.5m/s, 0.4m/s$. After applying SLOWDOWN$(\cdot)$ by (6.35), the resulting velocity of a_0 is shown in the right and the potential collision is avoided.

(V) ***Continuous control for tracking***. The *fifth* building block is the continuous controller for an active robot to track its synthesized path. We rely on the nonlinear control scheme from [101] for unicycle models that handles bounded control inputs and ensures the tracking of a reference trajectory with a provable bounded tracking error. In particular, consider that an active robot i needs to execute its path $\mathbf{p}_i$ by (6.28) and assume that it is going from q_i^l to q_i^{l+1} at time $t_0 > 0$. We first construct the reference trajectory $(x_r(t), y_r(t), \theta_r(t))$ as follows: (i) rotate to the desired orientation while staying at the same position. For $t \in [t_0, t_1)$, we set $x_r(t) = x_i^l$, $y_r(t) = y_i^l$, $\theta_r(t) = \theta_i^l + W_i \cdot \text{sgn}(\theta_i^{l+1} - \theta_i^l)(t - t_0)$ and $w_r(t) = W_i$, $v_r(t) = 0$, where $t_1 = t_0 + |\theta_i^{l+1} - \theta_i^l|/W_i$; (ii) forward towards the next grid while keeping the same orientation. For $t \in [t_1, t_2]$, we set $x_r(t) = x_i^l + v_i^l \cdot \cos(\theta_r) \cdot (t - t_1)$, $y_r(t) = y_i^l + v_i^l \cdot \sin(\theta_r) \cdot (t - t_1)$, $\theta_r(t) = \theta_i^{l+1}$ and $w_r(t) = 0$, $v_r(t) = v_i^l$, where $t_2 = \underline{d}/v_i^l$. Denote the saturation function by $\text{Sat}_\delta(x) = x$, $\forall |x| \leq \delta$ and $\text{Sat}_\delta(x) = \text{sgn}(x)\delta$, $\forall |x| > \delta$. Then the nonlinear control laws are given as follows: $v_i = v_r \cos(\theta_e) - \text{Sat}_a(k_0 x_e)$ and $w_i = w_r + \frac{f_1(x_e, y_e, \theta_e, t)}{f_2(x_e, y_e, t)} + \text{Sat}_b(k_1 \bar{\theta}_0)$, where $a = v_{\max} - v_i^l$; $x_e = \cos(\theta)(x - x_r) + \sin(\theta)(y - y_r)$; $y_e = -\sin(\theta)(x - x_r) + \cos(\theta)(y - y_r)$; $\theta_e = \theta_r - \theta$; $b > 0$ is chosen such that $|w_i| < w_{\max}$; $k_1, k_2 > 0$; $\bar{\theta}_0 = \theta_0 + f_3(x_e, y_e, t)\, y_e$; the actual expressions of functions $f_1(\cdot)$, $f_2(\cdot)$ and $f_3(\cdot)$ can be found in Section III of [101]. The guarantees for convergence and bounded tracking error are shown in Theorem 1 of [101]. For brevity, we denote this control scheme by the function:

$$(v_i, w_i) = \text{MOVE}(s_i(t), \mathbf{p}_i). \tag{6.36}$$

which can be either of the above approaches depending on the particular application. Examples of the control scheme can be found in Section 6.2.5.

EGGs description

With the above building blocks, we now present the complete graph grammars for the embedded graph $\gamma(t)$, which includes the set of local transition rules with the

associated conditions and control modes, which can be applied locally by each robot.

[R.0] At $t = 0$, each robot $i \in \mathcal{N}$ initializes its label by setting $i.\texttt{id} = i$, $i.\texttt{mode} = \texttt{check}$ or $i.\texttt{mode} = \texttt{static}$ randomly, and $i.\texttt{data.nb} = \varnothing$, $i.\texttt{data.pt} = [\,]$, $i.\texttt{gi} = 0$, where $[\,]$ denotes an empty sequence. Moreover, for any robot $j \in \mathcal{N}_i(0)$, it sets $(i, j).\texttt{id} = (i, j)$.

After the system starts at $t > 0$, *each* robot $i \in \mathcal{N}$ reconstructs its local graph $G_i(t)$ and applies the rules below: **[R.1]** If $i.\texttt{mode} = \texttt{check}$, robot i first communicates with *every* neighbor $j \in \mathcal{N}_i(t)$ and checks if $j.\texttt{mode} = \texttt{active}$ and $i \in j.\texttt{data.nb}$. If so, it sets $i.\texttt{mode} = \texttt{static}$ and adds robot j to $i.\texttt{data.nb}$.

After that, if $i.\texttt{mode} = \texttt{check}$ still holds, robot i chooses an allowed marked scheme $\mathcal{N}_i^m$ given $G_i(t)$ and calls the function $\textsc{Check}(s_i(t), \mathcal{N}_i(t), z_i, \mathcal{N}_i^m)$ in (6.32). If (6.30) has a solution as the tentative path $\mathbf{p}_i$ and the potential gain χ_i. If $\chi_i > 0$, then it sets $i.\texttt{mode} = \texttt{move}$ and $i.\texttt{data.nb} = \mathcal{N}_i^m(t)$, $i.\texttt{data.gi} = \chi_i$, $i.\texttt{data.pt} = \mathbf{p}_i$. Otherwise if $\chi_i \le 0$, it sets $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \varnothing$. Otherwise if no solutions to (6.30) exist or $\chi_i \le 0$, it sets $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \varnothing$.

[R.2] If $i.\texttt{mode} = \texttt{static}$, robot i stays static by setting $v_i = w_i = 0$. Then it communicates with each neighbor $j \in \mathcal{N}_i(t)$ and checks that if $j.\texttt{mode} = \texttt{active}$, $i \in j.\texttt{data.nb}$, and $j \notin i.\texttt{data.nb}$ hold. If so, it adds robot j to $i.\texttt{data.nb}$. Moreover, for each robot $j \in i.\texttt{data.nb}$, it checks whether $i \in j.\texttt{data.nb}$ still holds. If not, it removes robot j from $i.\texttt{data.nb}$. At last, it checks if $i.\texttt{data.nb} = \varnothing$. If so, it sets $i.\texttt{mode} = \texttt{check}$.

[R.3] If $i.\texttt{mode} = \texttt{move}$, robot i first checks if $j.\texttt{mode} = \texttt{static}$, $\forall j \in i.\texttt{data.nb}$. If not, it stops moving by setting $i.\texttt{mode} = \texttt{check}$ and $i.\texttt{data.nb} = \varnothing$. Otherwise, it executes its tentative path $\mathbf{p}_i$ via the motion controller $(v_i, w_i) = \textsc{Move}(s_i(t), \mathbf{p}_i)$ by (6.36). As discussed earlier, robot i may encounter other robots, e.g., $j \in \mathcal{N}_i(t)$:

(i) if $j.\texttt{mode} = \texttt{move}$, they exchange their respective gains and tentative paths. Then the robot with higher gain is given higher priority. Assume for now $i.\texttt{data.gi} < j.\texttt{data.gi}$, implying robot j has higher priority. Then the robot with lower priority, i.e., robot i, calls $\textsc{CollidePath}(\mathbf{p}_i, \mathbf{p}_j)$ by (6.33) to check if $\mathbf{p}_i$ and $\mathbf{p}_j$ will collide. If so, robot i calls $\textsc{SlowDown}(s_i(t), \mathbf{p}_i, \mathbf{p}_j)$ by (6.35). If it has a solution, robot i updates its path $\mathbf{p}_i$ by slowing down; otherwise, robot i stops moving by setting $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \varnothing$.

(ii) if $j.\texttt{mode} = \texttt{static}$, robot i checks if it would collide with robot j given its current path $\mathbf{p}_i$. If so, it stops moving by setting $i.\texttt{mode} = \texttt{check}$ and $i.\texttt{data.nb} = \varnothing$.

[R.4] If $i.\texttt{mode} = \texttt{move}$ and $\|p_i(t) - z_{i\ell}\| < c_i$, where $c_i > 0$ is the threshold from Definition 6.6, robot i has reached its goal point. Then robot i stops moving and resets $i.\texttt{mode} = \texttt{static}$ and $i.\texttt{data.nb} = \varnothing$.

It is worth mentioning that the gain comparison in **[R.3]** introduces a fixed priority among the active robots. It means that in the worst-case scenario all robots will slow down or be static except the one with the highest gain.

6.2.3 Local plan synthesis

The previous section solves how each robot could move to its current goal point, while obeying the motion constraints. Here we tackle how each robot should choose and update its goal point, in order to fulfill the local task φ_i. The solution was presented in Section 5.1: (i) recall that the complete motion and action model $\mathcal{R}_i$ is given in Section 6.2.1, (ii) then we derive the NBA $\mathcal{A}_{\varphi_i}$ associated with φ_i by the fast translation tools [41]; (iii) now we construct the product automaton $\mathcal{A}_{p,i} = \mathcal{T}_i \times \mathcal{A}_{\varphi_i}$ by Definition 3.9; (iv) lastly a nested Dijkstra's shortest path algorithm by Algorithm 3.1 is applied to $\mathcal{A}_{p,i}$, to find its strongly connected component with the minimal summation cost. The discrete plan denoted by τ_i has the prefix-suffix structure:

$$\tau_i = \pi_{i,0}\pi_{i,1}\cdots\pi_{i,k_i-1}(\pi_{i,k_i}\pi_{i,k_i+1}\cdots\pi_{i,K_i})^{\omega}, \tag{6.37}$$

where $\pi_{i,k} = (z_{i,k}, a_{i,k}) \in \Pi_i$ where $z_{i,k} \in Z_i$ and $a_{i,k} \in \Sigma_i$, $\forall i = 0, 1, \cdots, K_i$ and $K_i > 0$ is the total length of the prefix and suffix. Note that since the suffix is repeated infinitely often, τ_i has an infinite length. Given these locally-synthesized plans, we need the following assumption:

Assumption 6.2. *The plans* $\{\tau_i, i \in \mathcal{N}\}$ *are* feasible *as a whole if* $\gamma(t)$ *is allowed by Definition 6.5 when* $p(t)$ *satisfies* $p_i(t) = z_{i,k}$, $\forall i \in \mathcal{N}$ *and* $\forall k = 0, 1, \cdots$. ▲

6.2.4 Overall structure

In this part, we present the complete solution that combines the EGGs and the local plan synthesis scheme from above. When the system starts, each robot $i \in \mathcal{N}$ derives it local plan τ_i by (6.37) and sets its current goal point $z_{i\ell} = z_{i,0}$; then it follows the transition rules and control laws from the EGGs; by **[R.4]** after it reaches $z_{i,0}$, it then performs the action $a_{i,k}$ according to the plan τ_i; after the action is accomplished, it remains static until all other robots have reached their respective goal points and finished the corresponding actions. This can be detected through the communication network that all robots are static. Then each robot would update its goal point by $z_{i\ell} = z_{i,1}$ and sets $i.\texttt{mode} = \texttt{check}$, $\forall i \in \mathcal{N}$. Then all robots follow the EGGs to make progress towards this new goal point. This procedure repeats indefinitely as the discrete plans have infinite length. Note that after robot $i \in \mathcal{N}$ reaches z_{i,K_i}, it should set $z_{i\ell} = z_{i,k_i}$ to repeat the plan suffix by (6.37).

Lemma 6.18. *Given that* $G(0)$ *is connected initially, it is guaranteed that* $G(t)$ *remains connected for* $t \geq 0$.

Proof. Since $G(0)$ is connected, there exists at least one path of length N that connects all robots in $G(0)$. Denote by this path $\zeta_0 = a_0 a_1 \cdots a_N$, where robots a_i and a_{i+1} are directly connected by an edge and $a_i \in \mathcal{N}_{i+1}(0)$, $\forall i = 0, 1, \cdots, N-1$. Denote by $t_1 > 0$ as the smallest time instance that one of the consecutive robot pairs within ζ_0 is not directly connected anymore. Without loss of generality, let the pair be robots i and j. Notice that $j \notin \mathcal{N}_i(t_1)$ can only happen in one of

the following cases: (i) robot i is moving while robot j is static during $[0, t_1]$. Given the marked neighbors $\mathcal{N}_i^m(0)$, by Definition 6.9 it holds that $j \in \mathcal{N}_i^m(0)$. Given robot i's path $\mathbf{p}_i$ as derived by (6.30), Lemma 6.17 ensures that the subgraph $G_i^m(t)$ remains connected for $t \in [0, t_1]$ while robot i executes $\mathbf{p}_i$. Thus even though robots i and j are not connected directly at time t_1, they are still connected indirectly within $G_i^m(t_1)$; (ii) both robots i and j are moving during $[0, t_1]$. Given their marked neighbors $\mathcal{N}_i^m(0)$ and $\mathcal{N}_j^m(0)$, by Definition 6.9 there must exist a static robot $k \in \mathcal{N}_i(0)$ that $k \in \mathcal{N}_i^m(0)$ and $k \in \mathcal{N}_j^m(0)$. Given their paths $\mathbf{p}_i$, $\mathbf{p}_j$ as derived by (6.30), by the same analysis as in case (i), robots i, k remain connected and robots j, k remain connected during $[0, t_1]$, yielding that robots i, j remain connected indirectly. We conclude that robots i, j remain connected indirectly after becoming disconnected directly at time t_1. Since the other consecutive pairs in ζ_0 remain connected directly, $G(t_1)$ remains connected for $t \in [0, t_1]$. Now denote by ζ_1 the new path of length N that connects all robots within $G(t_1)$ at time t_1. By repeating the same analyses for ζ_0 above, we can show that $G(t)$ remains connected for $t \in [t_1, t_2]$, where t_2 is the smallest time instance that one of the consecutive robot pairs in ζ_1 becomes disconnected directly. Thus by recursive reasoning we can show that $G(t)$ remains connected, $\forall t \geq 0$. ■

Theorem 6.19. *All local tasks φ_i, $i \in \mathcal{N}$ are satisfied while $\gamma(t) \in \Gamma_d$, $\forall t > 0$.*

Proof. Since the workspace is assumed to be unbounded and free of obstacles, at least one robot within $\mathcal{N}$ can be active and make a progress towards its current goal point. The connectivity of $G(t)$ is proved above and the collision avoidance is ensured by the formulation of (6.30) and (6.34). Moreover, Assumption 6.2 ensures that the intermediate configuration of all robots' goal points is feasible and can be reached. At last, by construction, the execution of τ_i guarantees the satisfaction of φ_i, we ensure that the local task φ_i is satisfied, $\forall i \in \mathcal{N}$. ■

6.2.5 Case study

In this section, we present the simulation results of the EGGs-based hybrid control scheme presented above. The message passing among the robots are handled by the Robot Operating System (ROS) and each robot is launched as a ROS node.

Workspace and robot description

The six robots are labeled a_0, $a_1, \cdots, a_5$ and each occupies a disk area of radius $0.05m$. As shown in Figure 6.5, the communication range $\overline{d}$ is uniformly set to $0.9m$, while the safety distance $\underline{d}$ is set to $0.15m$. Moreover, their reference linear velocity is set to between $0.1m/s$ and $0.3m/s$, under the maximal $0.4m/s$. The angular velocity is set to between $0.4rad/s$ and $0.5rad/s$, under the maximal $0.7rad/s$.

The robots' points of interest and local task specifications are defined as follows: robots a_0, a_1 have the local task as surveillance. Robot a_0 has four points of interest at $(1.5, 1.5)$, $(-0.2, 1.5)$, $(0, 0)$, $(1.6, 0)$ with labels $\{r_1\}$, $\{r_2\}$, $\{r_3\}$, $\{r_4\}$.

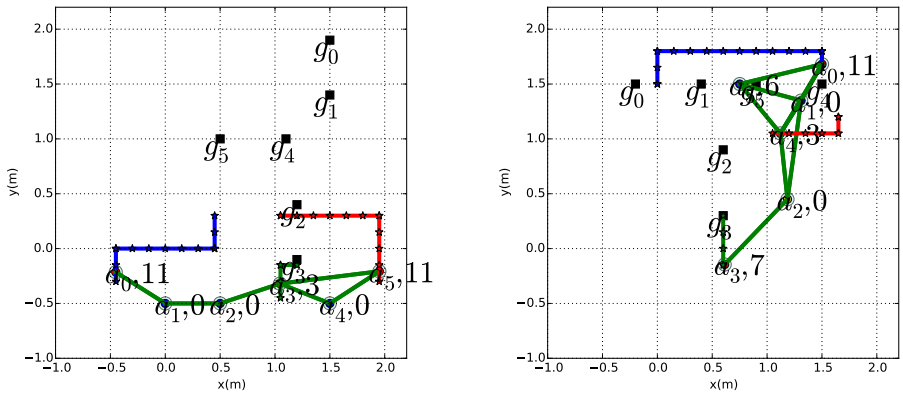


Figure 6.5: Snapshots of the simulation. Moving robots are denoted by red circles while static ones are in blue, labeled by a_i and a_i.gi. Lines marked by stars are tentative paths of the active robots. Black squares represent the goal points, labeled by g_i, $\forall i \in \mathcal{N}$.

Its local task is to surveil r_1, r_2, r_3, r_4 in any order, which can be specified as the LTL formula $\varphi_0 = \wedge_{i=1,\cdots,4} \square \diamond r_i$. Robots a_1 has points of interest close to a_0's and its local tasks is similar to φ_0. Robots a_2, a_3 have the local tasks for providing services. Robot a_2 has three points of interest at $(1.2, 0.4)$, $(0.6, 0.6)$, $(0.6, 0.9)$ with labels $\{s_1\}$, $\{s_3\}$, $\{s_2\}$. Its local task is to provide services s_1, s_2, s_3 in sequence, namely $\varphi_2 = \square \diamond (s_1 \wedge \diamond(s_2 \wedge \diamond s_3))$. Robots a_3 has points of interest close to a_3's and its task is similar to φ_2. At last, robots a_4, a_5 are responsible for transporting goods between goal points. Robot a_4 has three points of interest $(1.1, 1.0)$, $(1.5, 1.5)$, $(1.0, 1.0)$ with labels $\{b\}$, $\{g_1\}$, $\{g_2\}$. Its local task is to transport goods g_1 and g_2 to the base b, i.e., $\varphi_4 = \wedge_{i=1,2} \square \diamond (b \Rightarrow (\neg b \, \mathsf{U} \, g_i))$. Robot a_5 has three points of interest close to a_4's and its local task is similar to φ_4. Initially, the robots start from (-0.5, -0.5), (0, -0.5), (0.5, -0.5), (1.0, -0.5), (1.5, -0.5), (2.0, -0.5), which forms a line graph.

Simulation results

After the system starts, each robot first synthesizes its discrete plan τ_i as described in Section 6.2.3. To give an example, robot a_0's discrete plan is to visit r_1, r_2, r_3, r_4 in sequence and repeat, while robot a_4's plan is to visit b, g_1, b, g_2 in sequence and repeat. Then they follow the EGGs as described in Section 6.2.2. Most of the time there are $3-4$ robots moving. Figures 6.5 show some snapshots of how $G(t)$ changes with time. At $t = 92.5s, 160.5s$, all robots update their goal points according to their discrete plans. This procedure continues indefinitely and we simulate the system until $t = 240.5s$ when they have reached the forth goal point. To verify that all motion constraints are fulfilled, the left image of Figure 6.6 shows the evolution of the maximal length of the shortest path between any two vertices within $G(t)$ (i.e., the diameter). It is always less than 6, meaning that $G(t)$ remains connected. The

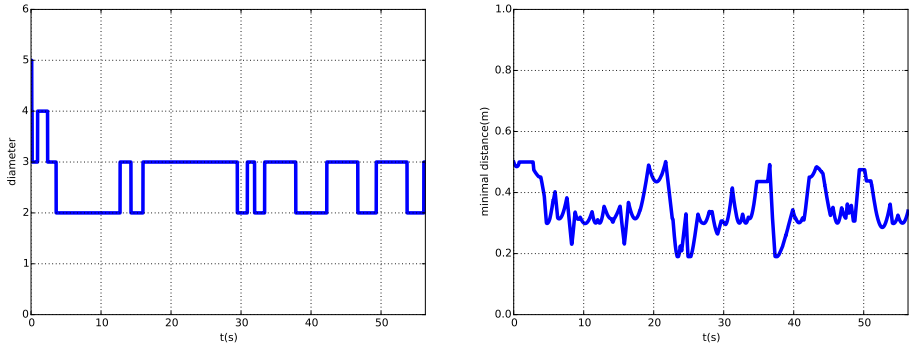


Figure 6.6: Evolution of the graph diameter (left) and the minimal distance among the robots (right). $G(t)$ remains connected as its diameter is always lower than 6; no collision occur as the minimal distance is always above $0.15m$.

right image of Figure 6.6 shows the evolution of the minimal distance between any two robots, which is always larger than the safety distance $0.15m$, meaning that no collision happens. The complete simulation video can be found in [65].

6.3 Summary

In this chapter, we first proposed a potential-field-based control scheme for multi-robot systems to fulfil locally-assigned tasks as general or sc-LTL formulas, while subject to relative-distance constraints. Then we considered the multi-robot system with a different dynamical model under both collision avoidance and connectivity maintenance constraints. The proposed solution relies on the EGGs to specify local communication and interaction rules among the robots. It has been shown that both approaches are distributed and can guarantee the satisfaction of all local tasks while the relative-motion constraints are obeyed at all time.

Chapter 7

Contingent service and formation tasks

TWO COMMON types of cooperative robotic tasks are the service and formation tasks. Particularly, the service request is a short-term task provided by one robot to another, while the formation task is a relative deployment requirement among the robots with predefined transient responses imposed by an associated performance function. These tasks are often requested and exchanged among the robots during run time. In this chapter, we address this issue by proposing a hybrid control strategy that handles the contingent service or formation tasks along with the local task of each robot. It involves monitoring the real-time events that are critical to the plan execution, adjusting the local plan to satisfy the contingent requests and switching the low-level continuous control mode. It is shown that both local tasks and contingent tasks are satisfied for all robots. Numerical simulations are provided in the end to validate the proposed strategy.

7.1 Problem formulation

In this section, we formally state the problem considered in this chapter, i.e., the robot model, the contingent service and formation tasks and the local task specifications given as RTL formulas.

Preliminaries

Maximal solution of dynamical systems

Consider the initial value problem:

$$\dot{\psi}(t) = H(t, \psi), \quad \psi(0) = \psi_0 \in \Omega_\psi, \tag{7.1}$$

where $\psi \in \mathbb{R}^n$ is the state, $H : \mathbb{R}^+ \times \Omega_\psi \to \mathbb{R}^n$ and $\Omega_\psi \subset \mathbb{R}^n$ is a non-empty open set. A solution $\psi(t)$ of the aforementioned initial value problem is *maximal* if it has no proper right extension that is also a solution of (7.1). Moreover, the following theorem and proposition will be employed in the sequel.

Theorem 7.1. *[133] Assume that $H(t, \psi)$ is: (i) locally Lipschitz on ψ for almost all $t > 0$; (ii) piecewise continuous on t for each fixed $\psi \in \Omega_\psi$; and (iii) locally integrable on t for each $\psi \in \Omega_\psi$. Then there exits a maximal solution $\psi(t)$ of (7.1) on the time interval $[0, t_{\max})$ with $t_{\max} > 0$ such that $\psi(t) \in \Omega_\psi$, $\forall t \in [0, t_{\max})$.* ▲

Proposition 7.2. *[133] Assume that the hypotheses of Theorem 7.1 above hold. For a maximal solution $\psi(t)$ of system (7.1) on the time interval $[0, t_{\max})$ with $t_{\max} < \infty$ and for any compact set $\Omega'_\psi \subset \Omega_\psi$, there exists a time instant $t' \in [0, t_{\max})$ such that $\psi(t') \notin \Omega'_\psi$.* ▲

.

Real-time temporal logic

In order to analyze properties of real-time signals, we also consider a real-time extensions of LTL described in Section 3.2, i.e., the real-time temporal logic (RTL), originally introduced in [124]. Its syntax is similar to LTL as introduced in Section 3.2, but its semantics is defined over *continuous-time* Boolean signals. Consider a continuous-time Boolean signal $x : \mathbb{R}_{\geq 0} \to 2^{AP}$ over the set of atomic propositions AP, where $x(t) \subseteq AP$ is the set of propositions that x satisfies at time $t \geq 0$. Given an RTL formula φ over AP, the satisfiability relation $(x, t) \vDash \varphi$, i.e., whether signal x satisfies φ at time $t \geq 0$, is determined according to the following recursive definition: $(x, t) \vDash a \leftrightarrow a \in x(t)$; $(x, t) \vDash \neg\varphi \leftrightarrow (x, t) \nvDash \varphi$; $(x, t) \vDash \varphi_1 \vee \varphi_2 \leftrightarrow (x, t) \vDash \varphi_1$ or $(x, t) \vDash \varphi_2$; $(x, t) \vDash \varphi_1 \mathsf{U} \varphi_2 \leftrightarrow \exists t' \geq t$, $(x, t') \vDash \varphi_2$ and $\forall t'' \in (t, t')$, $(x, t'') \vDash \varphi_1$. We say that an RTL formula φ is satisfied by x if $(x, 0) \vDash \varphi$ or simply $x \vDash \varphi$. Moreover, similarly to sc-LTL, there exists a particular class of RTL, called sc-RTL that can be satisfied by a real-time Boolean signal in a finite time. It only contains the U and $\Diamond$ temporal operators and is written in positive normal form [36]. Finally since we consider RTL and LTL with the same syntax but interpreted with different semantics, we define the following correspondence between an RTL formula and an LTL formula:

Definition 7.1. Given an RTL formula φ, the associated LTL formula, denoted by $[\varphi]$, has the same expression as φ but evaluated under the LTL semantics. ▲

7.1.1 System description

We consider a team of N autonomous robots with identities $i \in \mathcal{N} \triangleq \{1, 2, \cdots, N\}$ obeying the following single-integrator dynamics on a 2D workspace:

$$\dot{p}_i(t) = u_i(t), \tag{7.2}$$

where $p_i(t)$, $u_i(t) \in \mathbb{R}^2$ denote the i_{th} robot's position and control input at time $t \geq 0$, $\forall i \in \mathcal{N}$. It should be noticed that the robots are modeled as point masses without volume, hence no that inter-robot collisions are not considered. Each robot has a sensing radius $r > 0$, which is assumed to be identical for all robots, i.e., robot i can only exchange information with another robot j if their relative distance satisfies

$\|p_i(t) - p_j(t)\| \leq r$. We denote by $\mathbf{p}_i(t) : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ the trajectory of robot i during the time interval $[0, t)$. Moreover, $\mathbf{p}_i([t_1, t_2])$ stands for the trajectory segment during time $[t_1, t_2]$. Finally, each robot i has a predefined neighboring set $\mathcal{N}_i \subseteq \mathcal{N}$, $i \in \mathcal{N}$, which is assumed to be nonempty, and $j \in \mathcal{N}_i$ implies $i \in \mathcal{N}_j$, $\forall j \in \mathcal{N}_i$.

Moreover, there exists a set of regions of interest within the 2-D workspace $\Pi \triangleq \{\pi_1, \pi_2, \cdots, \pi_M\}$, where $M \geq 1$. They are circular area around the points of interest: $\pi_\ell \triangleq \mathcal{B}(c_\ell, r_\ell) = \{p \in \mathbb{R}^2 | \|p - c_\ell\| \leq r_\ell\}$, where $\ell = 1, 2, \cdots, M$, $c_\ell \in \mathbb{R}^2$ is the center and $r_\ell \geq r_{\min}$ is the radius, with $r_{\min}$ being a given constant as the minimal radius of all regions. We assume that Π is included within a large sphere region denoted by $\pi_B = \mathcal{B}(c_B, r_B)$, which stands for the allowed workspace. Finally, there is a set of *local* atomic propositions $R = \{R_\ell, \ell = 1, 2, \cdots, M\}$ representing the property fulfilled at each region. Hence given robot's state $p_i(t)$ at time $t \geq 0$, it holds that $R_\ell(t) = \top$ if $p_i(t) \in \pi_\ell$ and $R_\ell(t) = \bot$ otherwise, for $\ell = 1, 2, \cdots, M$.

7.1.2 Local task with contingent requests

In this part, we introduce the definition of contingent requests for formation and service, based on which we can then formulate the local task of each robot.

Contingent request

As mentioned earlier, the robots can exchange information with their neighbors when their relative distance is less than r. Via this communication protocol, we allow each robot to send contingent requests of the following two types:

(I) ***Service***: robot i requests its neighbor $j \in \mathcal{N}_i$ at time t to accomplish a short-term service described by a sc-RTL formula $\varphi^{\mathtt{s}}_{ij,t}$ over R. Note that $\varphi^{\mathtt{s}}_{ij,t}$ is predefined for each neighbor j and may be different at different request time t, which is assumed to be always feasible for robot j. This request can be communicated by sending the formula $\varphi^{\mathtt{s}}_{ij,t}$ directly to robot j.

(II) ***Formation***: robot i requests its neighbor $j \in \mathcal{N}_i$ at time t to converge to a desired relative-position formation by $c_{ij} \in \mathbb{R}^2$ with a predefined transient response imposed by the corresponding performance function $\rho_{ij}(t) : \mathbb{R}^+ \to \mathbb{R}^+$. This formation has to be kept until robot i accomplishes a short-term formation task described by a sc-RTL formula $\varphi^{\mathtt{f}}_{ij,t}$ over R and a release message is sent to robot j afterwards. The formation task $\varphi^{\mathtt{f}}_{ij,t}$ is also predefined for robot i and may be different at different request time t. The relative formation error is given by:

$$e_{ij}(t) \triangleq p_i(t) - p_j(t) - c_{ij}. \tag{7.3}$$

Here let us define $\mu_{ij} \triangleq e_{ij}^T e_{ij}$ as a scalar measure of the formation error and

$$\widehat{\mu}_{ij}(t) \triangleq \frac{\mu_{ij}(t)}{\rho_{ij}(t)} \tag{7.4}$$

as the normalized error with respect to the performance specifications introduced by the corresponding performance function $\rho_{ij}(t)$, which is a smooth, bounded and

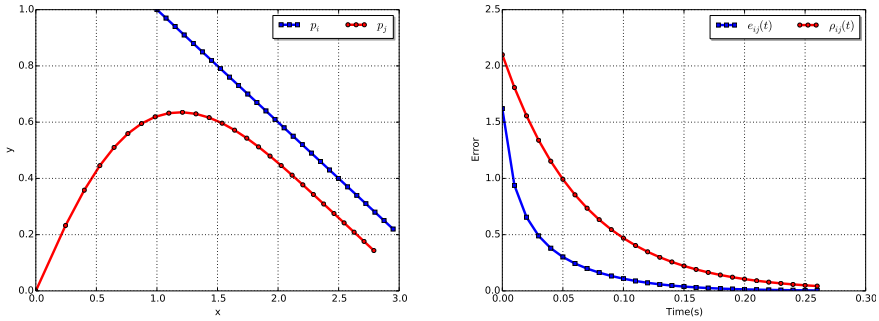


Figure 7.1: Robot i starts from $(1, 1)$ and its neighbor j starts from $(0, 0)$. The desired relative position is $c_{ij} = [0.1, 0.1]$ and the corresponding performance function is set to $\rho_{ij}(t) = 2.995\,\mathrm{e}^{-0.5t} + 0.005$. Robot i moves with constant velocity while robot j aims at establishing the desired formation with prescribed performance. The left figure shows the trajectory of both robots within the workspace while the right one depicts the evolution of $e_{ij}(t)$, along with the performance function $\rho_{ij}(t)$.

strictly positive function. We consider exponential performance functions here:

$$\rho_{ij}(t) \triangleq (\rho_{ij,0} - \rho_{ij,\infty})\,\mathrm{e}^{-l_{ij}t} + \rho_{ij,\infty}, \tag{7.5}$$

where $l_{ij} > 0$ specifies the decreasing rate of $\rho_{ij}(t)$, $\rho_{ij,0} > 0$ is the initial value of $\rho_{ij}(t)$ at time 0, chosen such that $\rho_{ij,0} > \mu_{ij}(0)$; and $\rho_{ij,\infty} > 0$ reflects the maximum allowed steady state error. This request can be communicated by sending the formation vector c_{ij} and the performance function $\rho_{ij}(t)$ to robot j.

In order to monitor the performance of the formation, we define a set of controllable atomic propositions: $H_i \triangleq \{h_{ij},\, j \in \mathcal{N}_i\}$, where

$$h_{ij}(t) \triangleq \begin{cases} \top & \text{if} \quad \widehat{\mu}_{ij}(t) \in D_{ij} \equiv (0,\, 1), \\ \bot & \text{otherwise.} \end{cases} \tag{7.6}$$

Notice that robot j needs to satisfy the prescribed formation performance, i.e., $h_{ij}(t) = \top$ until it receives a release message from robot i. An example of formation control with prescribed performance is shown in Figure 7.1.

Upon receiving either a service or formation request from robot i, we assume that robot j will reply immediately to either confirm or refuse this request. Thus we introduce a set of *observational* atomic propositions for each robot $i \in \mathcal{N}$: $O_i \triangleq \{o^{\mathtt{s}}_{ij},\, o^{\mathtt{f}}_{ij},\, j \in \mathcal{N}_i\}$, where $o^{\mathtt{s}}_{ij}(t) = \top$ if the service request sent by robot i is *confirmed* by robot j at time t; and $o^{\mathtt{s}}_{ij}(t) = \bot$ otherwise ($o^{\mathtt{f}}_{ij}(t)$ is defined similarly for formation requests). Additionally, we also introduce a set of *releasing* atomic propositions for robot $i \in \mathcal{N}$: $Z_i \triangleq \{z_{ij},\, j \in \mathcal{N}_i\}$, where $z_{ij}(t) = \top$ if robot i sends a release message to its neighbor $j \in \mathcal{N}_i$ at time t; and $z_{ij}(t) = \bot$ otherwise.

Local task specifications

Denote by $AP_i = R \cup O_i \cup H_i \cup Z_i$ the complete set of atomic propositions of each robot $i \in \mathcal{N}$. Each robot i is assigned a local task specified as an RTL formula φ_i over AP_i, which has the following structure:

$$\varphi_i \triangleq \varphi_i^{\mathtt{l}} \wedge \varphi_i^{\mathtt{s}} \wedge \varphi_i^{\mathtt{f}} \tag{7.7}$$

where $\varphi_i^{\mathtt{l}}$ denotes the static task specification as a general RTL formula over R concerning only the *local* motions of robot i; $\varphi_i^{\mathtt{s}}$ is the contingent task specification regarding the *service* requests that robot i may receive from robot j with $i \in \mathcal{N}_j$:

$$\varphi_i^{\mathtt{s}} \triangleq \bigwedge_{i \in \mathcal{N}_j,\, j \in \mathcal{N}} \square\,(o_{ij}^{\mathtt{s}} \rightarrow \varphi_{ji,t}^{\mathtt{s}}). \tag{7.8}$$

Hence whenever robot i confirms a service request $\varphi_{ji,t}^{\mathtt{s}}$ from robot j, then it should fulfill this service task afterwards; finally $\varphi_i^{\mathtt{f}}$ is the contingent task specification regarding the *formation* requests that robot i may send to its neighbor $j \in \mathcal{N}_i$:

$$\varphi_i^{\mathtt{f}} \triangleq \bigwedge_{j \in \mathcal{N}_i} \square\Big(o_{ji}^{\mathtt{f}} \rightarrow \big(\varphi_{ij,t}^{\mathtt{f}} \wedge (h_{ij} \,\mathsf{U}\, z_{ij})\big)\Big). \tag{7.9}$$

Therefore whenever a formation request from robot i is confirmed by robot j, the corresponding short-term formation task $\varphi_{ij,t}^{\mathtt{f}}$ should be fulfilled afterwards and moreover the formation controllable proposition by h_{ij} should be always kept true until a release message is sent to robot j. In other words, it requires that once robot j confirms a formation request from robot i, it should achieve the desired relative position with prescribed performance as imposed by the corresponding performance function $\rho_{ij}(t)$, until robot i has accomplished its formation task $\varphi_{ij,t}^{\mathtt{f}}$.

***Remark* 7.1.** While the local task $\varphi_i^{\mathtt{l}}$ as the static task specification is commonly-seen in the related literature [15, 56], main novelty of this work lies in the contingent service task $\varphi_i^{\mathtt{s}}$ by (7.8) and the formation task specification $\varphi_i^{\mathtt{f}}$ by (7.9) that have not been tackled at all for multi-robot systems. Moreover notice that all services and formation requests are exchanged in real-time and hence cannot be known a priori. The validity of the observational and controllable propositions can only be determined in real-time. Thus, it is not possible to acknowledge the complete specification φ_i before the system initializes. ▲

Thus, the problem confronted in this work is formulated as follows:

Problem 7.1. *Consider a team of robots obeying the dynamics* (7.2)*, with each assigned a local task defined by* (7.7)-(7.9)*. The goal is to synthesize a distributed hybrid control protocol that the RTL formulas φ_i are satisfied, $\forall i \in \mathcal{N}$.* ▲

The proposed hybrid control scheme consists of four major components as shown in Figure 7.2, i.e., the communication block, the event monitoring block, the discrete planning block and the hybrid control block. In the following, we first present two

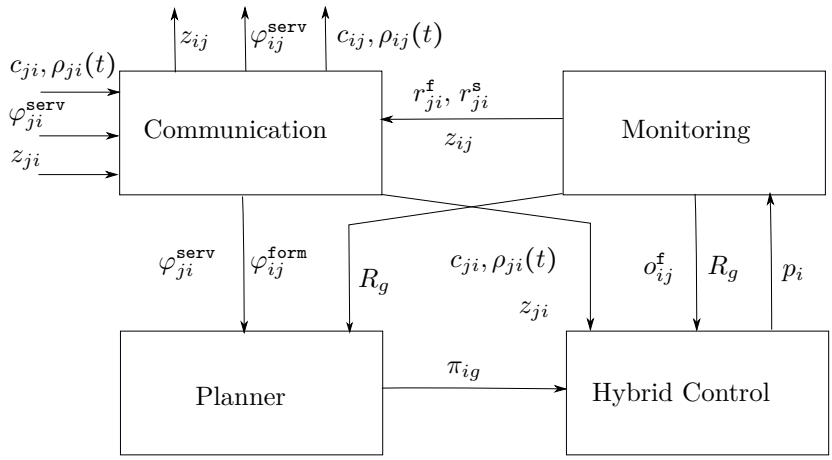


Figure 7.2: The overall structure of the hybrid control scheme. The arrows indicate the information flow. Detailed descriptions are given in Section 7.5.

continuous motion control schemes regarding the navigation and the formation tasks respectively; subsequently we describe the real-time event monitoring module that handles critical events during the system operation as well as the established communication protocol among the robots regarding contingent requests. Afterwards, we show the discrete plan synthesis and adaptation algorithms that handle the contingent service and formation requests. Finally, the overall hybrid control scheme is synthesized based on these components and its correctness is proven.

7.2 Controller design under prescribed performance

In this part, we describe the continuous control design for two different objectives, i.e., to navigate an robot to its goal region, without crossing any undesirable regions; and to establish a desired formation with a prescribed transient response.

7.2.1 Navigation control

Recall that the set of sphere regions of interest within the workspace boundary π_B is given by $\Pi = \{\pi_1, \pi_2, \cdots, \pi_M\}$. Since there are no explicit representations of "obstacles" and no fixed initial or goal regions, we denote by $\pi_g = \mathcal{B}(c_g, r_g) \in \Pi$ the goal region and $\pi_s = \mathcal{B}(c_s, r_s) \in \Pi$ the initial region. In this work, we rely on the navigation function approach proposed by Rimon and Koditschek in [87] to navigate robot i from π_s to π_g, without crossing other undesirable regions $\pi_j \in \Pi$ with $j \neq s, g$. In particular, the navigation function can be constructed as

$$\Phi_i(p_i) \triangleq \frac{\gamma_g}{(\gamma_g^k + \beta_{gs})^{\frac{1}{k}}}, \tag{7.10}$$

where $k > 0$ is a design parameter, $p_i \in \mathbb{R}^2$ and $\Phi_i \in [0, 1]$; $\gamma_g \triangleq \|p_i - c_g\|^2$ represents the attractive potential field to the goal $c_g \in \mathbb{R}^2$; $\beta_{gs} \triangleq \beta_B \prod_{j=1, j\neq s,g}^{M} \beta_j$ is the repulsive potential field by the workspace boundary and the undesirable regions that should be avoided, with $\beta_B \triangleq r_B^2 - \|p_i - c_B\|^2$ and $\beta_j \triangleq \|p_i - c_j\|^2 - r_j^2$. For brevity, we denote by $\Pi_{\text{avoid}} \triangleq \cup_{j=1, j\neq s,g}^{M} \pi_j$ the set of regions to be avoided and by $\pi_B \backslash \Pi_{\text{avoid}}$ the free space. It is assumed that π_B and the sphere regions in Π satisfy the condition of a valid workspace from [87], i.e., $\pi_m \subset \pi_B$ and $\pi_m \cap \pi_n = \varnothing$, $\forall m, n = 1, 2, \cdots, M$.

It has been proved [87] that $\Phi_i(p_i)$ has only one global minimum at $p = c_g$ and $M - 2$ saddle points within the allowed freespace with zero set-measure. Hence a feasible path within the free workspace that leads an robot from its initial position in π_s to its goal region π_g can be generated by following the negated gradient of $\Phi_i(p_i)$ or equivalently by adopting the subsequent control law:

$$u_i = -\nabla_{p_i} \Phi_i(p_i) \tag{7.11}$$

Based on [105], it is proven that $\gamma_g \to 0$ as $t \to \infty$ and $\beta_{gs} > 0$ holds, $\forall t \geq 0$, for a sufficiently large constant k. Moreover, a collision free path is ensured for almost any initial position in the free space (except a set of measure zero) to any goal region in the free space. Furthermore, its asymptotic stability guarantees the convergence to the neighborhood of p_g (i.e., the goal region π_g) in finite time [105].

Lemma 7.3. *The navigation control law* (7.11) *remains bounded, i.e.,* $\|u_i\| < u_{\max}$, *where* $u_{\max} > 0$ *is a finite upper bound given the workspace configuration.*

Proof. By computing explicitly the gradient of $\Phi_i(p_i)$, we get:

$$\nabla_{p_i} \Phi = \frac{\beta_{gs} \nabla_{p_i} \gamma_g - \frac{1}{k} \gamma_g \nabla_{p_i} \beta_{gs}}{\left(\gamma_g^k + \beta_{gs}\right)^{\frac{1}{k}+1}}, \tag{7.12}$$

where $\nabla_{q_i} \gamma_g = 2(p_i - c_g)$ and $\nabla_{q_i} \beta_{gs} = \left(\sum_{j=1, j\neq s,g}^{M} 2(p_i - c_j) \beta_B \prod_{k=1, k\neq j,s,g}^{M} \beta_k\right) - \left(2(p_i - c_B) \prod_{j=1, j\neq s,g}^{M} \beta_j\right)$. Owing to the fact that q_i remains in the allowed free space without crossing Π_{avoid}, both $\nabla_{p_i} \gamma_g$ and $\nabla_{p_i} \beta_{gs}$ have bounded magnitude and $\beta_{gs} > 0$, which implies that $\|u_i\| = \|\nabla_{p_i} \Phi_i\| \leq u_{\max}$ with an upper bound $u_{\max} > 0$ depending on the configuration of Π. ∎

7.2.2 Prescribed formation control

As described in Section 7.1.2, a contingent request may also involve a formation task that should be executed with prescribed performance, i.e., robot i requests robot $j \in \mathcal{N}_i$ to converge to a desired formation described by the relative position c_{ij} while satisfying predefined transient constraint introduced by the corresponding prescribed performance function $\rho_{ij}(t)$ as defined in (7.5). Moreover, this formation should be maintained until the associated short-term formation task $\varphi_{ij,t}^{\mathbf{f}}$ is satisfied by robot i. Since robot i may send a formation request to more than one of its neighbors, we denote by $\mathcal{N}_i^{\mathbf{f}} \subseteq \mathcal{N}_i$ the subset of robots that have confirmed its

request. In the sequel, we propose the motion control scheme for both robot i that sends the formation request and robot $j \in \mathcal{N}_i^{\mathtt{f}}$ that confirmed it.

Notice that at the request time t in case robot i lies inside a region $\pi_s \in \Pi$ then the associated navigation function can be constructed similarly to (7.10). Otherwise the repulsive potential β_{gs} should be slightly modified since all regions in Π except π_g should be treated as undesirable regions to be avoided. Denoting the corresponding navigation function by $\Phi_i^{\mathtt{f}}(p_i)$, the proposed motion controller is given by:

$$u_i \triangleq -\nabla_{p_i} \Phi_i^{\mathtt{f}}(p_i). \tag{7.13}$$

On the other hand, for each robot $j \in \mathcal{N}_i^{\mathtt{f}}$ that has confirmed the formation request of robot i, we design a motion control law that ensures the convergence to the desired relative formation under the prescribed performance as follows:

$$u_j \triangleq K_{ij} \frac{\varepsilon_{ij}}{\rho_{ij}(t)} e_{ij}, \tag{7.14}$$

where $K_{ij} > 0$ is a design parameter; $e_{ij} = p_i - p_j - c_{ij}$; and

$$\varepsilon_{ij} \triangleq \ln\Big(\frac{1}{1-\widehat{\mu}_{ij}}\Big), \tag{7.15}$$

where $\widehat{\mu}_{ij} = \frac{e_{ij}^T e_{ij}}{\rho_{ij}(t)}$ is defined by (7.4) as the normalized formation error.

Lemma 7.4. *Under the motion control laws by* (7.13) *and* (7.14)*, initialized at time $t = T_0$, robot i will arrive its goal region π_g within finite time $T_g > T_0$, while all robots $j \in \mathcal{N}_i^f$ that confirmed its formation request will satisfy $e_{ij}^T(t) e_{ij}(t) < \rho_{ij}(t), \forall t \in [T_0, T_g]$, thus fulfilling the formation request with prescribed performance as imposed by the associated performance functions $\rho_{ij}(t), j \in \mathcal{N}_i^f$.*

Proof. Following similar analysis within Section 7.2.1, we may show that robot i arrives its goal region π_g within finite time $T_g > T_0$, while avoiding the undesirable regions included in Π_{avoid} Regarding the robots $j \in \mathcal{N}_i^{\mathtt{f}}$ that have confirmed the formation request, the dynamics of the formation error is calculated as follows:

$$\frac{\partial \widehat{\mu}_{ij}}{\partial t} = \frac{2\, e_{ij}^T\, u_{ij}\, \rho_{ij}(t) - e_{ij}^T\, e_{ij}\, \dot{\rho}_{ij}(t)}{\rho_{ij}^2(t)},$$

where $u_{ij}(t) = u_i(t) - u_j(t)$. Substituting (7.14) and $e_{ij}^T\, e_{ij} = \rho_{ij}\, \widehat{\mu}_{ij}$, we obtain:

$$\begin{aligned} \frac{\partial \widehat{\mu}_{ij}}{\partial t} &= -\frac{\widehat{\mu}_{ij}}{\rho_{ij}(t)}\big[K_{ij}\, \varepsilon_{ij} + \dot{\rho}_{ij}(t)\big] + \frac{2\, e_{ij}^T\, u_i(t)}{\rho_{ij}(t)} \\ &= -\frac{\widehat{\mu}_{ij}}{\rho_{ij}(t)}\big[K_{ij} \ln\Big(\frac{1}{1-\widehat{\mu}_{ij}}\Big) + \dot{\rho}_{ij}(t)\big] + \frac{2\, e_{ij}^T\, u_i(t)}{\rho_{ij}(t)}. \end{aligned} \tag{7.16}$$

Since the right-hand side of (7.16) is continuous on t and locally Lipschitz on $\widehat{\mu}_{ij}$ over D_{ij}, we may apply Theorem 7.1 to deduce that a maximal solution of (7.16) on a time interval $[T_0, \tau_{\max})$ such that $\widehat{\mu}_{ij}(t) \in D_{ij},\ \forall t \in [T_0, \tau_{\max})$. Let us also consider the following Lyapunov-like function: $V_{ij}(t) \triangleq \frac{1}{2}\,\varepsilon_{ij}^2$, which owing to (7.15) is well-defined, $\forall t \in [T_0, \tau_{\max})$. Its time derivative is given by:

$$\dot{V}_{ij} = \varepsilon_{ij}\,\frac{\partial \varepsilon_{ij}}{\partial t} = \frac{\varepsilon_{ij}}{1-\widehat{\mu}_{ij}}\,\frac{\partial \widehat{\mu}_{ij}}{\partial t}. \tag{7.17}$$

Invoking (7.16), $\dot{V}_{ij} = -h_{ij}\Big[\widehat{\mu}_{ij}\big(K_{ij}\,\varepsilon_{ij}+\dot{\rho}_{ij}(t)\big)-2\,e_{ij}^T\,u_i(t)\Big]$, where $h_{ij}(t) = \frac{\varepsilon_{ij}}{(1-\widehat{\mu}_{ij})\,\rho_{ij}(t)} > 0$. Since $\varepsilon_{ij} > 0$, $\rho_{ij}(t) > 0$ and $|e_{ij}^T\,u_i(t)| \le \|e_{ij}\|\|u_i(t)\|$, we have

$$\begin{aligned}\dot{V}_{ij} &\le -h_{ij}\Big[K_{ij}\,\widehat{\mu}_{ij}\,\varepsilon_{ij} + \widehat{\mu}_{ij}\,\dot{\rho}_{ij}(t) - 2\,\|e_{ij}\|\|u_i(t)\|\Big]\\ &\le -h_{ij}\Big[K_{ij}\,\widehat{\mu}_{ij}\,\varepsilon_{ij} - |\dot{\rho}_{ij}(t)| - 2\sqrt{\rho_{ij}(t)}\,\|u_i(t)\|\Big],\end{aligned}$$

where we use the fact that $0 \le \widehat{\mu}_{ij} < 1$ and $\|e_{ij}\| = \sqrt{\mu_{ij}} = \sqrt{\widehat{\mu}_{ij}\,\rho_{ij}(t)} \le \sqrt{\rho_{ij}(t)}$. Hence, since $\widehat{\mu}_{ij} = 1 - \mathbf{e}^{-\varepsilon_{ij}}$, we conclude that $\dot{V}_{ij} < 0,\ \forall t \in [0, \tau_{\max})$ when

$$\varepsilon_{ij}\big(1-\mathbf{e}^{-\varepsilon_{ij}}\big) > \frac{\sup_{t\in[T_0,\,\tau_{\max})}\{|\dot{\rho}_{ij}(t)| + 2\sqrt{\rho_{ij}(t)}\,\|u_i(t)\|\}}{K_{ij}} \triangleq b_{ij}. \tag{7.18}$$

Notice that b_{ij} is finite as $K_{ij} > 0$, $\dot{\rho}_{ij}(t)$, $\rho_{ij}(t)$ are bounded by construction and $\|u_i(t)\| < u_{\max}$ as proven in Lemma 7.3. Consider now the smooth function $g(x) = x(1-\mathbf{e}^{-x})$, which is monotonically increasing for $x > 0$ with $g(0) = 0,\ \forall x > 0$. Let $\varepsilon_{ij}^\star$ be a constant that satisfies $\varepsilon_{ij}^\star(1-\mathbf{e}^{-\varepsilon_{ij}^\star}) = b_{ij}$, which exists and is unique owing to the monotonicity of $g(x)$. Hence $\dot{V}_{ij} < 0$ when $\varepsilon_{ij} > \varepsilon_{ij}^\star$, from which we conclude: $\varepsilon_{ij}(t) < \max\{\varepsilon_{ij}^\star,\, \varepsilon_{ij}(T_0)\} \triangleq \overline{\varepsilon}_{ij},\ \forall t \in [T_0, \tau_{\max}]$ and consequently:

$$\widehat{\mu}_{ij}(t) < 1 - \mathbf{e}^{-\overline{\varepsilon}_{ij}} \triangleq \overline{\widehat{\mu}}_{ij} < 1,\ \ \forall t \in [T_0, \tau_{\max}).$$

Thus we deduce that $\widehat{\mu}_{ij}(t) \in [0, \overline{\widehat{\mu}}_{ij}] \triangleq D'_{ij},\ \forall t \in [T_0, \tau_{\max}]$, where D'_{ij} is a nonempty and compact subset of D_{ij}. Finally, what remains to be shown is that $\tau_{\max}$ can be extended to ∞. Therefore, assume that $\tau_{\max} < \infty$. Since $D'_{ij} \subset D_{ij}$ and $\widehat{\mu}_{ij}(t)$ is a maximal solution of (7.16) over $[T_0, \tau_{\max})$, Proposition 7.2 dictates the existence of a time instant $t' \in (T_0, \tau_{\max})$ such that $\widehat{\mu}_{ij}(t) \notin D'_{ij}$, which clearly contradicts the fact that $\widehat{\mu}_{ij}(t) \in D'_{ij},\ \forall t \in [T_0, \tau_{\max})$. Thus $\tau_{\max} = \infty$ and $\widehat{\mu}_{ij}(t) \in [T_0, \overline{\widehat{\mu}}_{ij}] \subset D_{ij}$, $\forall t \in [T_0, \infty)$, which ensures that: $e_{ij}^T(t)e_{ij}(t) < \rho_{ij}(t), \forall t \in [T_0, T_g],\ \forall j \in \mathcal{N}_i^{\mathbf{f}}$. ∎

7.3 Triggering events and communication protocol

In this section, we first present the set of real-time events that are crucial for the system execution, and then describe the communication protocol among the robots to handle contingent requests.

7.3.1 Real-time event monitoring scheme

It is clear from the problem formulation that monitoring several real-time events plays a crucial role in the satisfaction of the individual local tasks. In particular, for each robot $i \in \mathcal{N}$, four events need to be monitored in real time:

(a) ***Region cross event***. It is the event that occurs when an robot enters or leaves a region in Π. More precisely, robot i enters the region $\pi_\ell \in \Pi$ at time $t_0 > 0$ if $p_i(t_0^-) \notin \pi_\ell$ and $p_i(t_0) \in \pi_\ell$, where $\delta_d < t_0 - t_0^- < \delta_t$, with $\delta_d, , \delta_t > 0$ as two design parameters and δ_d is similar to the notion of dwelling time [40]. Robot i leaves the region $\pi_\ell \in \Pi$ at time $t_0 > 0$ if $p_i(t_0^-) \in \pi_\ell$ and $p_i(t_0) \notin \pi_\ell$, where $\delta_d < t_0 - t_0^- < \delta_t$. This event is directly related to the validity of proposition $R_\ell \in R$.

(b) ***Request and reply event***. It is the event that occurs when robot i sends a service or formation request to its neighbor $j \in \mathcal{N}_i$ at time $t_0 > 0$ and robot j replies this request at the same time. If robot j confirms this service request, the observational proposition $o_{ij}^{\mathtt{s}} \in O_i$ satisfies $o_{ij}^{\mathtt{s}}(t) = \top$, $\forall t \in [t_0, t_0 + \delta_d)$ and $o_{ij}^{\mathtt{s}}(t) = \bot$, $\forall t \in [t_0 + \delta_d, t_1]$, where $t_1 > t_0 + \delta_d$ is the next time instant when robot j receives a request from robot i; otherwise, if robot j denies this request, $o_{ij}^{\mathtt{s}}(t) = \bot$, $\forall t \in [t_0, t_1]$. Same rules apply to propositions related to formation requests $\{o_{ij}^{\mathtt{f}}, \forall j \in \mathcal{N}_i\}$. In addition, if robot i sends a release message to its neighbor $j \in \mathcal{N}_i$ at t_0, then $z_{ij}(t) = \top$, $\forall t \in [t_0, t_0 + \delta_d)$ and $z_{ij}(t) = \bot$, $\forall t \in [t_0 + \delta_d, t_1]$, where t_1 is the next time instant when robot i sends a release message to robot j.

(c) ***Service finish event***. It is the event that occurs when robot i has finished a service request that it has confirmed. For instance, suppose that $o_{ji}^{\mathtt{s}}(t_0) = \top$ and robot i confirms the service request $\varphi_{ji,t_0}^{\mathtt{s}}$ by robot j at time $t_0 > 0$. Then $\varphi_{ji,t_0}^{\mathtt{s}}$ is satisfied by robot i at time $t_f \geq t_0$, if its trajectory $\mathbf{p}_i([t_0, t_f])$ satisfies $\varphi_{ji,t_0}^{\mathtt{s}}$ on the basis of the semantics presented in Section 7.1. Since all service tasks are assumed to be feasible, the finishing time $t_f > t_0$ is finite for each service request.

(d) ***Formation finish event***. It is the event that occurs when robot i has finished a formation task that has been confirmed. For instance, suppose that $o_{ji}^{\mathtt{f}}(t_0) = \top$ and robot j confirms the formation request by robot i at time $t_0 > 0$. The corresponding formation task for robot i is given by $\varphi_{ij,t_0}^{\mathtt{f}}$. Then $\varphi_{ij,t_0}^{\mathtt{f}}$ is satisfied at time $t_f \geq t_0$ if $\mathbf{p}_i([t_0, t_f])$ satisfies $\varphi_{ij,t_0}^{\mathtt{f}}$ on the basis of the semantics presented in Section 7.1. Since all formation tasks are also assumed to be feasible in finite time, the time t_f is also finite for each formation request.

Summarizing the events (a) and (b) can be easily monitored in real-time based on their definitions. On the other hand, it is not trivial to monitor the events (c) and (d) in an efficient way, especially when multiple service or formation requests occur. Nevertheless, since event (c) and (d) are closely related to the discrete plan synthesis and adaptation module, more details will be given there.

7.3.2 Protocol for exchanging contingent requests

Each robot $i \in \mathcal{N}$ will receive, send and confirm various contingent requests from its neighbors during the system operation. Hence, denote by $r_{ji}^s(t) : \mathbb{R}^+ \to \mathbb{B}$ the

continuous-time Boolean variable indicating whether robot i is serving a service request from robot $j \in \mathcal{N}_i$ at time $t > 0$. In addition, let $r^{\mathtt{f}}_{ji}(t) : \mathbb{R}^+ \to \mathbb{B}$ be the variable indicating whether robot i is serving a formation request from robot $j \in \mathcal{N}_i$ at time $t > 0$. Initially, $r^{\mathtt{s}}_{ji}(0) = \bot$ and $r^{\mathtt{f}}_{ji}(0) = \bot$, $\forall j \in \mathcal{N}_i$. Moreover, $r^{\mathtt{f}}_{ii}(t) : \mathbb{R}^+ \to \mathbb{B}$ indicates robot i is currently executing the task associated with its own formation requests, which is also initialized as false, i.e., $r^{\mathtt{f}}_{ij}(0) = \bot$.

Robot i may send the predefined service formula $\varphi^{\mathtt{s}}_{ij,t}$ to its neighbor $j \in \mathcal{N}_i$ at time $t > 0$ only if $r^{\mathtt{f}}_{ij}(t) = \bot$, i.e., robot j is not still serving the previous service request from robot i. Similarly robot i may send the formation request described by c_{ij} and $\rho_{ij}(t)$ at time $t > 0$, only if $r^{\mathtt{f}}_{gi} = \bot$, $\forall g \in \mathcal{N}_i$, i.e., robot i itself is not currently serving any of its neighbor's formation request and $r^{\mathtt{f}}_{ij}(t) = \bot$,i.e., robot j is not still serving the previous formation request from robot i. After sending a request, robot i needs to wait for robot j's reply. Once its request is confirmed by robot j, the observational proposition $o^{\mathtt{s}}_{ij}(t)$ or $o^{\mathtt{f}}_{ij}(t)$ is updated according to part (b) of Section 7.3.1. On the other hand, whenever robot i *receives* a request from its neighbor $j \in \mathcal{N}_i$ at time $t > 0$, it checks first whether $r^{\mathtt{f}}_{ii}(t) = \top$ or $r^{\mathtt{f}}_{ji}(t) = \top$, for any $j \in \mathcal{N}_i$, i.e., robot i is currently executing its own formation task or serving one of its neighbor's formation request. If so, robot i denies this request; otherwise, robot i confirms its request and starts serving it. Alternatively, other approaches like direct user triggering could also be incorporated.

From the above, we see that each robot $i \in \mathcal{N}$ may serve multiple service requests from its neighbors but only one formation request at a time. This is because a new service or formation request can only be sent if the previous service or formation request is fulfilled. Moreover, an robot can confirm multiple service requests from its neighbors, but only one formation request as long as it is not currently serving another formation request. On the other hand it will deny any service or formation requests while it is serving a formation request, i.e., $r^{\mathtt{f}}_{ji}(t) = \top$, $\forall j \in \mathcal{N}_i$.

7.4 Discrete plan synthesis and adaptation

In this section, we demonstrate how to synthesize the initial discrete plan for each robot based on its own local task specifications and how to adapt it in real-time upon receiving contingent service and formation requests from its neighboring robots.

7.4.1 Initial plan synthesis

At time $t = 0$, we assume that no requests have been sent yet, i.e., $o^{\mathtt{s}}_{ij}(0) = o^{\mathtt{f}}_{ij}(0) = \bot$, $\forall j \in \mathcal{N}_i$ and $i \in \mathcal{N}$. As a result, only the static task $\varphi^{\mathtt{l}}_i$ of the RTL formula φ_i defined in (7.7) is initially pursued. Hence the initial plan synthesis aims at finding a discrete plan that satisfies $\varphi^{\mathtt{l}}_i$.

Our approach relies on the model-checking algorithm [11] as introduced in Chapter 3. The motion of robot i within the workspace is abstracted by a wFTS: $\mathcal{T}_i \triangleq (\Pi_i, \longrightarrow, R, L, \Pi_{i,0}, W)$, where $\Pi_i \triangleq \{\pi_0\} \cup \Pi$ is the set of states with $\pi_0 \triangleq \mathcal{B}(p_i(0), 0)$; $\longrightarrow \triangleq (\Pi \times \Pi) \cup (\{\pi_0\} \times \Pi)$ denotes the transition relation; $L : \Pi_i \to 2^R$ with

$L(\pi_\ell) \triangleq R_\ell$, $\forall \ell \in \{1, 2, \cdots, M\}$ is the labeling function; $\Pi_{i,0} \triangleq \{\pi_0\}$ is the initial state; and $W :\longrightarrow\!\!\!\!\!\rightarrow \mathbb{R}^+$ computes the cost of each transition with $W(\pi_m, \pi_n) \triangleq \|c_m - c_n\|$, $\forall(\pi_m, \pi_n) \in \longrightarrow$. Notice that only the local propositions of R are allowed in $\mathcal{T}_i$ and that the initial state π_0 is solely determined by the initial position of robot i. A path of $\mathcal{T}_i$ is given by $\tau = \pi_0\pi_1\pi_2\cdots$, where $(\pi_k, \pi_{k+1}) \in \longrightarrow$, $\forall k \geq 0$ and its associated trace,which corresponds to a discrete-time Boolean signal over R, is defined as the sequence of propositions that are true along the path, i.e., $\texttt{trace}(\tau) = L(\pi_0)L(\pi_1)\cdots$, which corresponds to a discrete-time Boolean signal over R.

On the other hand, φ_i^1 is a general RTL formula over R with a corresponding LTL formula denoted by $[\varphi_i^1]$ based on Definition 7.1. Thus following the notations defined in Section 3.2, we can construct the NBA associated with $[\varphi_i^1]$, denoted by $\mathcal{B}_{i,s} = (Q_s, 2^R, \delta_s, Q_{s,0}, F_s)$. Subsequently, given $\mathcal{T}_i$ and $\mathcal{B}_{i,s}$, we may construct the weighted product Büchi automaton (wPBA) similarly to Definition 3.9. The weighted PBA $\mathcal{A}_{p,i} = \mathcal{T}_i \times \mathcal{B}_{i,s} = (Q_{p,i}, \delta_{p,i}, Q_{p,i,0}, F_{p,i}, W_{p,i})$ is defined by $Q_{p,i} = \Pi_i \times Q_s$ with $q_p' = \langle \pi, q\rangle \in Q_{p,i}$, $\forall \pi \in \Pi_i$ and $\forall q \in Q_s$; $\delta_{p,i} : Q_{p,i} \to 2^{Q_{p,i}}$ with $\langle \pi_d, q_n\rangle \in \delta_s'(\langle \pi_c, q_m\rangle)$ if and only if $(\pi_c, \pi_d) \in \longrightarrow$ and $q_n \in \delta_s(q_m, L(\pi_c))$; $Q_{p,i,0} = \Pi_{i,0} \times Q_{s,0}$ is the set of initial states; $F_{p,i} = \Pi_i \times F_s$ is the set of accepting states; $W_{p,i} : \delta_{p,i} \to \mathbb{R}^+$ with $W_{p,i}(\langle \pi_c, q_m\rangle, \langle \pi_d, q_n\rangle) = W(\pi_c, \pi_d)$, $\forall(\langle \pi_c, q_m\rangle, \langle \pi_d, q_n\rangle) \in \delta_{p,i}$.

After $\mathcal{A}_{p,i}$ is constructed, we search for one of its accepting runs denoted by R_i that: (i) has a prefix-suffix structure, i.e., $R_i = R_{i,\text{pre}}\,(R_{i,\text{suf}})^\omega$; and (ii) minimizes the total cost $\texttt{cost}(R_i, \mathcal{A}_{p,i}) = \texttt{cost}(R_{i,\text{pre}}) + \texttt{cost}(R_{i,\text{suf}})$, where $\texttt{cost}(R_{i,\text{pre}})$ and $\texttt{cost}(R_{i,\text{suf}})$ are simply the accumulated weight of the transitions along the finite sequence of product states in $R_{i,\text{pre}}$ and $R_{i,\text{suf}}$. In this aspect, since $\mathcal{A}_{p,i}$ may be viewed as a directed graph with initial and accepting states, a variation of the Dijkstra's shortest path algorithm can be used to find such an optimal accepting run. For algorithmic details, we refer the readers to Algorithm 3.1 and Algorithm 2 in [51]. Denote by this optimal accepting run by $R_{i,0}$ and it can be then projected back onto Π_i, yielding the initial discrete plan of robot i as $\tau_{i,0} = R_{i,0}|_{\Pi_i}$. Note that $\tau_{i,0}$ also has the prefix-suffix structure and the trace of $\tau_{i,0}$ satisfies $[\varphi_i^1]$ automatically [11]. In this way $\tau_{i,0}$ obtains the following sequence:

$$\tau_{i,0} = \pi_{i0}\pi_{i1}\cdots\left(\pi_{ik_i}\,\pi_{i(k_i+1)}\cdots\pi_{iK_i}\right)^\omega, \tag{7.19}$$

with trace given by $\texttt{trace}(\tau_{i,0}) = L(\pi_{i0})L(\pi_{i1})\cdots$, where $\pi_{i\ell} \in \Pi_i$, $\forall \ell = 1, 2, \cdots, K_i$; $\pi_{i0}\pi_{i1}\cdots\pi_{ik_i}$ is the plan prefix defined as a finite sequence of goal regions to reach; and $\pi_{ik_i}\pi_{i(k_i+1)}\cdots\pi_{iK_i}$ is the plan suffix which is also finite but should be repeated infinitely often, in order to satisfy $[\varphi_i^1]$.

If no contingent requests are sent or confirmed by robot i for all $t > 0$, its initial discrete plan $\tau_{i,0}$ should remain unchanged and can be executed as follows: initializing at $p_i(0)$, robot i moves to and enters region π_{i1} employing the motion controller described in Section 7.2.1. Then an event that robot i entered region π_{i1} should be detected. Afterwards, it leaves region π_{i1} and moves to region π_{i2} under the same control scheme. Such procedure is repeated until it reaches π_{iK_i}, after which the next goal region is set at the beginning of the plan suffix π_{ik_i} and continues

to π_{iK_i} and back to π_{ik_i} again. In this way, the plan suffix is repeated infinitely often as $t \to \infty$. Denote by $\mathbf{p}_i(t)$ the resulting trajectory of robot i after the above execution. Thus there exists an infinite sequence of time instants $0\, t_1^1\, t_1^2\, t_2^1\, t_2^2 \cdots t_k^1\, t_k^2 \cdots$, with $t_{k+1}^2 > t_{k+1}^1 > t_k^2 > t_k^1 > 0$, $\forall k \geq 1$, such that

$$p_i(t) \in \pi_{ik}, \quad \forall t \in [t_k^1, t_k^2), \ \ \forall k \geq 1, \tag{7.20}$$

where note that when $k > K_i$, $\pi_{ik} \triangleq \pi_{ik'}$ for $k' \triangleq \texttt{mod}(k - k_i, K_i - k_i) + k_i$, where $\texttt{mod}$ is the modulo operation. In this way, the trajectory $\mathbf{p}_i$ intersects with the infinite sequence of goal regions as specified by $\tau_{i,0}$.

Theorem 7.5. *If no contingent requests are sent or confirmed by robot i, i.e., $o_{ij}^{\mathtt{s}}(t) = o_{ij}^{f}(t) = \bot$, $\forall j \in \mathcal{N}_i$ and $\forall t \geq 0$, then the trajectory $\mathbf{p}_i(t)$ generated by executing the initial plan $\tau_{i,0}$ satisfies the RTL formula φ_i.*

Proof. First of all, if $o_{ij}^{\mathtt{s}}(t) = o_{ij}^{\mathtt{f}}(t) = \bot$, $\forall j \in \mathcal{N}_i$ and $\forall t \geq 0$, the service task $\varphi_i^{\mathtt{s}}$ and formation task $\varphi_i^{\mathtt{f}}$ defined by (7.8) and (7.9) may be ignored owing to the semantics of the implication operator. In particular, if $o_{ij}^{\mathtt{s}}(t) = \bot$, $\forall t \geq 0$, $\varphi_{ij,t}^{\mathtt{s}}$ need not be true to satisfy $\varphi_i^{\mathtt{s}}$. Similarly, if $o_{ij}^{\mathtt{f}}(t) = \bot$, $\forall t \geq 0$, then $\varphi_{ij,t}^{\mathtt{f}} \wedge (h_{ij} \,\mathsf{U}\, z_{ij})$ need not be true to satisfy $\varphi_i^{\mathtt{f}}$. Thus φ_i is equivalent to $\varphi_i^{\mathtt{l}}$ and $[\varphi_i]$ is equivalent to $[\varphi_i^{\mathtt{l}}]$.

Notice also that $\varphi_i^{\mathtt{l}}$ is specified over R and hence depends solely on the robot's trajectory, since it does not contain any observational or controllable propositions. Moreover, we have shown that the trace of $\tau_{i,0}$ satisfies the LTL formula $[\varphi_i^{\mathtt{l}}]$ as well as that the trajectory $\mathbf{p}_i(t)$ satisfies (7.20). Hence we also need to show that $\mathbf{p}_i(t)$ satisfies the RTL formula $\varphi_i^{\mathtt{l}}$.

Let us define $w = \texttt{trace}(\tau_{i,0})$, which is a discrete-time Boolean signal over R with $(w, 0) \vDash [\varphi_i^{\mathtt{l}}]$. From the semantics of LTL and RTL presented in Sections 3.2 and 7.1, it is easy to show that (i) if $(w, 0) \vDash [R_\ell]$ for $R_\ell \in R$, i.e., robot i needs to start from region π_ℓ, then (7.20) guarantees that there exists a time interval $[0, t_1)$ with $t_1 > 0$ such that $p_i(t) \in \pi_\ell$, $\forall t \in [0, t_1)$. Thus, invoking the RTL semantics we conclude that $(\mathbf{p}_i, 0) \vDash R_\ell$. Similar arguments may also apply for $[\neg R_\ell]$ with $R_\ell \in R$; (ii) if $(w, 0) \vDash [R_{\ell_1} \vee R_{\ell_2}]$ for $R_{\ell_1}, R_{\ell_2} \in R$, i.e., robot i needs to start either from region π_{ℓ_1} or π_{ℓ_2}, then (7.20) guarantees that there exists a time interval $[0, t_1)$ with $t_1 > 0$ such that $p_i(t) \in \pi_{\ell_1}$ or $p_i(t) \in \pi_{\ell_2}$, $\forall t \in [0, t_1)$. Thus, invoking the RTL semantics we also conclude that $(\mathbf{p}_i, 0) \vDash R_{\ell_1} \vee R_{\ell_2}$. (iii) if $(w, 0) \vDash [R_{\ell_1} \mathsf{U} R_{\ell_2}]$ for $R_{\ell_1}, R_{\ell_2} \in R$, i.e., robot i needs to stay at region π_{ℓ_1} before it moves to region π_{ℓ_2}, then (7.20) guarantees that there exists a time interval $[0, t_1)$ with $t_1 > 0$ such that $p_i(t) \in \pi_{\ell_1}$ $\forall t \in [0, t_1)$ and a subsequent one $[t_1, t_2)$ with $t_2 > t_1$ such that $p_i(t) \in \pi_{\ell_2}$, $\forall t \in [t_1, t_2)$. Thus, invoking the RTL semantics we also conclude that $(\mathbf{p}_i, 0) \vDash R_{\ell_1} \mathsf{U} R_{\ell_2}$. Similar arguments can be applied to other operators like $\Diamond$, $\rightarrow$ and $\Box$ through induction. Thus since $(\texttt{trace}(\tau_{i,0}), 0) \vDash [\varphi_i]$, we conclude that $(\mathbf{p}_i(t), 0) \vDash \varphi_i$. ∎

The aforementioned results are valid only if no contingent requests are exchanged while the multi-robot system operates, which however is not the case in this work.

Thus, in the sequel we study how service or formation requests should be handled locally by adapting the discrete plans of the robots.

7.4.2 Event-based plan adaptation

Initially, we show how to handle contingent service requests. Based on our previous analysis, an robot may confirm and serve multiple service requests simultaneously. In such case, robot i needs to satisfy three kinds of tasks: (i) the static local task; (ii) all the service requests received so far that have not been satisfied; and (iii) the newly-received service request. These tasks need to be treated differently since the first kind should be satisfied by the whole trajectory from $t = 0$; the second kind by the trajectory starting from the time the service requests are confirmed; the third kind by the future trajectory. Thus it is important to keep track of how much the local task as well as the past and current service requests have been satisfied. In that respect, we consider three different cases: (I) robot i receives the first service request; (II) robot i receives a new service request while carrying out an old one from another neighbor; and (III) robot i receives a new service request from the same neighbor after its previous service request has been satisfied.

Case I: Suppose the first service request $\varphi^{\mathtt{s}}_{ji,t_j}$ received by robot i at time $t_j > 0$ was sent by robot j. Robot i needs to incorporate $\varphi^{\mathtt{s}}_{ji,t_j}$ into its static task specification $\varphi^{\mathtt{l}}_i$ and update its current plan $\tau_{i,0}$ to satisfy this request.

As mentioned earlier, the service task specification from (7.7) requires $\varphi^{\mathtt{s}}_{ji,t_j}$ to be satisfied in a timely manner. The associated LTL formula is denoted by $[\varphi^{\mathtt{s}}_{ji,t_j}]$ and $\mathcal{B}_{[\varphi^{\mathtt{s}}_{ji,t_j}]} = (Q_j, 2^R, \delta_j, Q_{j,0}, F_j)$ is the corresponding NBA, where the notations are defined similarly as in Section 3.2. Recall that $\mathcal{B}_{[\varphi^{\mathtt{l}}_i]} = (Q_s, 2^R, \delta_s, Q_{s,0}, F_s)$ is the NBA associated with $[\varphi^{\mathtt{l}}_i]$. Assume that at time t_j the corresponding product state of robot i in $\mathcal{A}'_{p,i}$ is $q'_{p,t_j} \in Q'_p$ and the associated Büchi state in $\mathcal{B}_{[\varphi^{\mathtt{l}}_i]}$ is $q_{s,t_j} = q'_{p,t_j}|_{Q_s}$. Thus the request-prioritized and layered intersection of $[\varphi^{\mathtt{s}}_{ji,t_j}]$ and $[\varphi^{\mathtt{l}}_i]$ is:

Definition 7.2. The intersection of $\mathcal{B}_{[\varphi^{\mathtt{s}}_{ji,t_j}]}$ and $\mathcal{B}_{[\varphi^{\mathtt{l}}_i]}$ is an NBA defined by:

$$\mathcal{A}_{[\varphi_i]} = (Q, 2^R, \delta, Q_0, F), \tag{7.21}$$

where $Q = Q_j \times Q_s \times \{1, 2\}$; $Q_0 = Q_{j,0} \times \{q_{s,t_j}\} \times \{1\}$; $F = F_j \times F_s \times \{2\}$; $\delta : Q \times 2^R \to 2^Q$, with $\langle \check{q}_j, \check{q}_s, \check{c}\rangle \in \delta(\langle q_j, q_s, c\rangle, l)$ when the following conditions hold: (i) $\langle q_j, q_s, c\rangle$, $\langle \check{q}_j, \check{q}_s, \check{c}\rangle \in Q$; (ii) $\check{q}_j \in \delta_j(q_j, l)$ and $\check{q}_s \in \delta_s(q_s, l)$; (iii) $q_j \notin F_j$ and $\check{c} = c = 1$; *or* $q_j \in F_j$, $c = 1$ and $\check{c} = 2$; *or* $\check{c} = c = 2$. ▲

Different from the conventional way of computing intersections of Büchi automata [11], $[\varphi^{\mathtt{l}}_i]$ is a general LTL formula that should be satisfied at $t = 0$ while $[\varphi^{\mathtt{s}}_{ji,t_j}]$ is a sc-LTL formula that should be satisfied after it is received. In particular, $\mathcal{A}_{[\varphi_i]}$ has two layers and it transits from the first layer to the second only if it reaches F_j, i.e., $[\varphi^{\mathtt{s}}_{ji,t_j}]$ is satisfied; afterwards it stays at the second layer in order

to satisfy $[\varphi_i^{\mathsf{l}}]$. In the following lemma, we prove the correctness of Definition 7.2 by showing that $\mathcal{A}_{[\varphi_i]}$ accepts all words that satisfy both $[\varphi_i^{\mathsf{l}}]$ and $[\varphi_{ji,t_j}^{\mathsf{s}}]$.

Lemma 7.6. *If there exists an ω-word $w \in R^\omega$ such that $w \vDash [\varphi_{ji,t_j}^{\mathsf{s}}]$ and $w \vDash [\varphi_i^{\mathsf{l}}]$, then $\mathcal{A}_{[\varphi_i]}$ has at least one accepting run.*

Proof. Owing to the fact that $w \vDash [\varphi_{ji,t_j}^{\mathsf{s}}]$, at least one of the resulting runs of w in $\mathcal{B}_{[\varphi_{ji,t_j}^{\mathsf{s}}]}$ is an accepting run, denoted by $r_j = q_{j,0}q_{j,1}\cdots q_{j,K_j}(q_{j,K_j})^\omega$, where $q_{j,0}q_{j,1}\cdots q_{j,K_j}$ is a finite sequence from an initial state $q_{j,0} \in Q_{j,0}$ and cycles from $q_{j,K_j} \in F_j$ and $(q_{j,K_j})^\omega$ is a repetitive suffix over q_{j,K_j}. Such argument holds true since all accepting states of $\mathcal{B}_{[\varphi_{ji,t_j}^{\mathsf{s}}]}$ have a self-cycle that accepts any input alphabet (see Remark 4.31 of [11]). On the other hand, given that $w \vDash [\varphi_i^{\mathsf{l}}]$, then w results in at least one accepting run of $\mathcal{B}_{[\varphi_i^{\mathsf{l}}]}$, denoted by $r_s = q_{s,0}q_{s,1}\cdots(q_{s,K_f}\, q_{s,K_f+1}q_{s,K_f+2}\cdots q_{s,K_f+K_s})^\omega$ that starts from an initial state $q_{s,0} \in Q_{s,0}$ to an accepting state $q_{s,K_f} \in F_s$ and $q_{s,K_f}q_{s,K_f+1}q_{s,K_f+2}\cdots q_{s,K_f+K_s}$ is a finite cyclic suffix that cycles from q_{s,K_f} and back to itself. Without loss of generality, we assume $K_f > K_j$. If not so, the suffix of r_s can be extended by repeating itself until $K_f > K_j$. In this sense, we can easily verify that a resulting run of w in $\mathcal{A}_{[\varphi_i]}$ can be constructed as follows:

$$\begin{aligned} r = {} & (q_{j,0}, q_{s,0}, 1)\cdots(q_{1,K_j}, q_{s,K_j}, 1)(q_{1,K_j}, q_{s,K_j+1}, 2) \\ & (q_{1,K_j}, q_{s,K_j+2}, 2)(q_{1,K_j}, q_{s,K_j+3}, 2)\cdots \\ & \Big((q_{1,K_j}, q_{s,K_f}, 2)(q_{1,K_j}, q_{s,K_f+1}, , 2)\cdots(q_{1,K_j}, q_{s,K_f+K_s}, , 2)\Big)^\omega. \end{aligned} \tag{7.22}$$

Consequently, since the first state $(q_{1,0}, q_{s,0}, 1) \in Q_0$ and the state within the repetitive suffix $(q_{1,K_j}, q_{s,K_f}, 2) \in F$, then by definition r is an accepting run of $\mathcal{A}_{[\varphi_i]}$ and hence $\mathcal{A}_{[\varphi_i]}$ accepts w. ∎

It can be seen that r reaches the accepting states in F_j that satisfies $[\varphi_{ji,t_j}^{\mathsf{s}}]$ before it repeats the suffix that satisfies $[\varphi_i^{\mathsf{l}}]$. Thus the service request has a higher priority than the static specification and is satisfied earlier. Moreover, the layered structure of $\mathcal{A}_{[\varphi_i]}$ allows us to track the satisfaction of $[\varphi_i^{\mathsf{l}}]$ and $[\varphi_{ji,t_j}^{\mathsf{s}}]$ separately.

Case II: Suppose that robot i has confirmed m service requests $[\varphi_{gi,t_g}^{\mathsf{s}}]$ at time t_g from robot g, $\forall g \in \mathcal{N}_i^{\mathsf{s}} \subseteq \mathcal{N}_i$. Without loss of generality, we can re-order the neighbors in $\mathcal{N}_i^{\mathsf{s}}$ by $\{1, 2, \cdots, m\} \triangleq \mathcal{N}_i^{\mathsf{s}}$, according to the time their service requests were received, e.g., $[\varphi_{1i,t_1}^{\mathsf{s}}]$ denotes the earliest and $[\varphi_{mi,t_m}^{\mathsf{s}}]$ denotes the latest received request. Furthermore, let $\mathcal{B}_{[\varphi_{gi,t_g}^{\mathsf{s}}]} = (Q_g, 2^R, \delta_g, Q_{g,0}, \mathcal{F}_g)$ be the NBA associated with $[\varphi_{gi,t_g}^{\mathsf{s}}]$, $\forall g \in \mathcal{N}_i^{\mathsf{s}}$, where the notations are defined as in Section 3.2.

Additionally, assume that at time t_m the corresponding product state of robot i in $\mathcal{A}_{p,i}$ is $q'_{p,t_g} \in Q'_p$. In that respect, its associated Büchi state in $\mathcal{B}_{[\varphi_i^{\mathsf{l}}]}$ is given by $q_{s,t_m} = q'_{p,t_m}|_{Q_s}$ and the associated Büchi state in each $\mathcal{B}_{[\varphi_{gi,t_g}^{\mathsf{s}}]}$ is given by $q_{g,t_m} = q'_{p,t_m}|_{Q_g}$, $\forall g \in \mathcal{N}_i^{\mathsf{s}}$. Thus we may define the request-prioritized and layered intersection of $[\varphi_{gi,t_g}^{\mathsf{s}}]$, $\forall g \in \mathcal{N}_i^{\mathsf{s}}$ and $[\varphi_i^{\mathsf{l}}]$ as follows:

Definition 7.3. The intersection of $\mathcal{B}_{[\varphi^{\mathbf{s}}_{gi,t_g}]}$, $\forall g \in \mathcal{N}^{\mathbf{s}}_i$ and $\mathcal{B}_{[\varphi^{\mathbf{l}}_i]}$ is an NBA by:

$$\mathcal{A}_{[\varphi_i]} = (Q, 2^R, \delta, Q_0, \mathcal{F}), \tag{7.23}$$

where $Q = Q_1 \times Q_2 \cdots \times Q_m \times Q_s \times \{1, 2, \cdots, m+1\}$; $Q_0 = \{q_{1,t_m}\} \times \{q_{2,t_m}\} \cdots \times \{q_{m-1,t_m}\} \times Q_{m,0} \times \{q_{s,t_m}\} \times \{1\}$; $\mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2 \cdots \times \mathcal{F}_m \times \mathcal{F}_{s,0} \times \{m+1\}$; $\delta : Q \times 2^R \to 2^Q$, with $\langle \check{q}_1, \check{q}_2, \cdots, \check{q}_m, \check{q}_s, \check{c}\rangle \in \delta(\langle q_1, q_2, \cdots, q_m, q_s, c\rangle, l)$ when the following conditions hold: (i) $\langle q_1, q_2, \cdots, q_m, q_s, c\rangle$, $\langle \check{q}_1, \check{q}_2, \cdots, \check{q}_m, \check{q}_s, \check{c}\rangle \in Q$; (ii) $\check{q}_j \in \delta_j(q_j, l)$, $\forall j \in \mathcal{N}^{\mathbf{s}}_i$; and $\check{q}_s \in \delta_s(q_s, l)$; (iii) $q_c \notin \mathcal{F}_c$ and $\check{c} = c$; *or* $q_c \in \mathcal{F}_c$ and $\check{c} = c + 1$; *or* $\check{c} = c = m + 1$. ▲

Notice that $\mathcal{A}_{[\varphi_i]}$ has $m+1$ layers and transits to the $(c+1)_{th}$ layer only if the set of accepting states F_c is reached for $c = 1, 2, \cdots, m$, i.e., $[\varphi^{\mathbf{s}}_{ci,t_c}]$ is satisfied; and then it stays at the $(m+1)_{th}$ layer in order to satisfy $\mathcal{B}_{[\varphi^{\mathbf{l}}_i]}$. The definition of Q_0 ensures that the past progress of serving $[\varphi^{\mathbf{l}}_i]$ and $[\varphi^{\mathbf{s}}_{gi,t_g}]$ for $g \in \mathcal{N}^{\mathbf{s}}_i$ is preserved, while the definition of δ allows us to keep track of such progress separately. Finally, the correctness of the aforementioned definition relies on the following lemma:

Lemma 7.7. *If there exists an ω-word $w \in R^\omega$ such that $w \vDash [\varphi^{\mathbf{s}}_{gi,t_g}]$, $\forall g \in \mathcal{N}^{s}_i$ and $w \vDash [\varphi^{\mathbf{l}}_i]$, then $\mathcal{A}_{[\varphi_i]}$ has at least one accepting run.*

Proof. Similar to the proof with Lemma 7.6, we may construct an accepting run of $\mathcal{A}_{[\varphi_i]}$ by employing the resulting runs of w over $[\varphi^{\mathbf{l}}_i]$ and $[\varphi^{\mathbf{s}}_{gi,t_g}]$, $\forall g \in \mathcal{N}^{\mathbf{s}}_i$. Similarly to (7.22), such accepting run reaches an accepting state of $\mathcal{B}_{[\varphi^{\mathbf{s}}_{2i,t_2}]}$ first, transits to the second layer, reaches an accepting state of $\mathcal{B}_{[\varphi^{\mathbf{s}}_{2i,t_2}]}$, and transits to the third layer and so on. This process is repeated until it reaches the $(m+1)_{th}$ layer and stays there. Afterwards its suffix is repeated infinitely often to satisfy $[\varphi^{\mathbf{l}}_i]$. ■

In other words, the fact that $\mathcal{A}_{[\varphi_i]}$ accepts the common words of $[\varphi^{\mathbf{l}}_i]$ and all the service requests $[\varphi^{\mathbf{s}}_{gi,t_g}]$, $\forall g \in \mathcal{N}^{\mathbf{s}}_i$. Note that $\mathcal{A}_{[\varphi_i]}$ is updated recursively by Definition 7.3 whenever robot i confirms new requests from its neighbors.

Case III: Assume that robot i has confirmed a service request $\varphi^{\mathbf{s}}_{li,t^1_l}$ from its neighbor $l \in \mathcal{N}_i$ at time $t^1_l > 0$ and accomplished it at $t^2_l > t^1_l$. Afterwards, at time $t^3_l > t^2_l$, robot i confirms a new service request $\varphi^{\mathbf{s}}_{li,t^3_l}$ from the same neighbor l. In the sequel we discuss how we should adjust $\mathcal{A}_{[\varphi_i]}$ to replace the old request $\varphi^{\mathbf{s}}_{li,t^1_l}$ by the new request $\varphi^{\mathbf{s}}_{li,t^3_l}$. Let us denote by $\mathcal{N}^{\mathbf{s}}_i$ the set of neighbors whose service requests robot i has to satisfy and by $\mathcal{A}_{[\varphi_i]}$ the associated intersection NBA. Since the service request from robot l has changed from $\varphi^{\mathbf{s}}_{li,t^1_l}$ to $\varphi^{\mathbf{s}}_{li,t^3_l}$, $\mathcal{A}_{[\varphi_i]}$ needs to be updated as follows. First, the service requests need to be re-organized according to the time they were received. Thus, the service request of robot l should be assigned the index $|\mathcal{N}^{\mathbf{s}}_i|$ as it was he latest received one. Subsequently, given the product state q'_{p,t^3_l} of robot i at time t^3_l, the Büchi state associated with each $[\varphi^{\mathbf{s}}_{gi,t_g}]$ is derived by the projection $q'_{p,t^3_l}|_{Q'_g}$, $\forall g \in \mathcal{N}^{\mathbf{s}}_i$ and $g \neq l$. Thus $\mathcal{A}_{[\varphi_i]}$ can be recomputed by Definition 7.3, with the service request of robot l being moved to the $|\mathcal{N}^{\mathbf{s}}_i|_{th}$ layer.

Given the updated intersection automaton $\mathcal{A}_{[\varphi_i]}$ from Cases (I)-(III), the corresponding product automaton $\mathcal{P}_i$ should be updated following Definition 3.9. Thus we should search for an accepting path of the updated $\mathcal{P}_i$ that minimizes the cost function mentioned in Section 7.4.1. Subsequently, this accepting run can be projected onto Π, thus yielding the updated discrete plan of robot i. Finally, notice that the local plan should also be adapted whenever $\mathcal{A}_{[\varphi_i]}$ is updated in any of the aforementioned cases. In the sequel, we prove the correctness of the proposed discrete plan adaptation approach. In this respect, assume that robot i receives a service request $\varphi^{\mathtt{s}}_{gi,t_g}$ from neighbor $g \in \mathcal{N}^{\mathtt{s}}_i$ at time $t_g > 0$. After that, at time $t > t_g > 0$, robot i has crossed the sequence of goal regions $\pi_{i,1}\pi_{i,2}\cdots\pi_{i,K}$ during $[t_g, t]$, which is uniquely determined by its discrete plan $\tau_i([t_g, t])$, where $\pi_{i,k} \in \Pi$, $\forall i = 1, 2, \cdots, K$. Then the corresponding trace is defined by $\mathtt{trace}_i([t_g, t]) = L_i(\pi_{i,0})L_i(\pi_{i,1})\cdots L_i(\pi_{i,K})$. Furthermore, let $q'_{i,t} \in Q'$ be the corresponding product state in $\mathcal{P}_i$ at time t.

Theorem 7.8. *Given that φ^{s}_{gi,t_g} is feasible, there exists a finite time $t > t_g$ such that the trajectory $\mathbf{p}_i([t_g, t])$ of robot i satisfies the RTL formula φ^{s}_{gi,t_g}.*

Proof. Since $\varphi^{\mathtt{s}}_{gi,t_g}$ is feasible, Lemma 7.7 ensures that $\mathcal{A}_{[\varphi_i]}$ accepts the satisfying words of $[\varphi^{\mathtt{s}}_{gi,t_g}]$. Irrespective of the layer $[\varphi^{\mathtt{s}}_{gi,t_g}]$ is in $\mathcal{A}_{[\varphi_i]}$, since all service requests are co-safe, there exists a finite time $t > t_g$ when $q'_{i,t}|_Q$ reaches the accepting state F_g From the definition of $\mathcal{A}_{[\varphi_i]}$, we know that if $q_g \in F_g$ for any $g \in \mathcal{N}^{\mathtt{s}}_i$, then the past trace during $[t_g, t]$ (i.e., $\mathtt{trace}_i([t_g, t])$) results in an accepting path of $\mathcal{B}_{[\varphi^{\mathtt{s}}_{gi,t_g}]}$ from an initial state to the accepting state q_g. Thus $\mathtt{trace}_i([t_g, t])$ satisfies $[\varphi^{\mathtt{s}}_{gi,t_g}]$. Therefore the fact that the resulting trajectory $\mathbf{p}_i([t_g, t])$ satisfies the RTL formula $\varphi^{\mathtt{s}}_{gi,t_g}$ can be easily deduced following the line of proof of Theorem 7.5. ∎

This proof also provides a straightforward way for robot i to monitor the satisfaction of each request specification $\varphi^{\mathtt{s}}_{gi,t_g}$, $\forall g \in \mathcal{N}^{\mathtt{s}}_i$. Notice that given the projection $q_{i,t} = q'_{i,t}|_Q$, if $q_{i,t}|_{Q_g} \in F_g$ for any $g \in \mathcal{N}^{\mathtt{s}}_i$, then $[\varphi^{\mathtt{s}}_{gi,t_g}]$ is satisfied by $\mathtt{trace}_i([t_g, t])$. This issue plays an important role for the event monitoring scheme and the communication protocol presented in Section 7.3.

Finally, while serving its service requests, robot i may send formation requests to its neighbors in $\mathcal{N}_i$ that may be confirmed. In this sense, assume that robot $j \in \mathcal{N}_i$ confirms a formation request at time t_j. The formation between robots i and j should be kept until the associated formation task $\varphi^{\mathtt{f}}_{ij,t_j}$ is fulfilled by robot i. In order to achieve that, robot i first needs to incorporate $\varphi^{\mathtt{f}}_{ij,t_j}$ into φ_i employing the same procedure as with the new service requests case described previously. More specifically, the NBA associated with $[\varphi^{\mathtt{f}}_{ij,t_j}]$ is computed as $\mathcal{B}_{[\varphi^{\mathtt{f}}_{ij,t_j}]}$ and the intersection automaton $\mathcal{A}_{[\varphi_i]}$ is recomputed by adding one extra level for $\mathcal{B}_{[\varphi^{\mathtt{f}}_{ij,t_j}]}$ in Definition 7.3. Thus similar to Theorem 7.8, since $\varphi^{\mathtt{f}}_{ij,t_j}$ is a feasible sc-RTL formula, it can be shown that $\varphi^{\mathtt{f}}_{ij,t_j}$ will be satisfied at a finite time instant $t > t_j$, after which, robot i sends a release message to robot j and then the formation between them is removed by the continuous controller switch as follows.

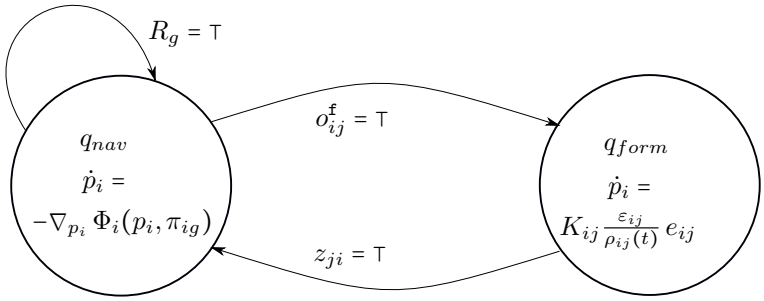


Figure 7.3: The hybrid controller module. The arrows indicate the discrete transition labeled by the associated guards.

After robot j confirms a formation request from its neighbor $i \in \mathcal{N}_j$ at time $t_j > 0$, it should converge to the desired formation described by the relative position c_{ij} under the prescribed performance function $\rho_{ij}(t)$ until the release message is received from robot i at $t > t_j$. Thus at time t_j, robot j needs to switch from the navigation control law (7.10) to the formation control law (7.14). Then at time t, it needs to switch back to the navigation controller to execute its original discrete plan.

Theorem 7.9. *Given that φ^{f}_{ij,t_j} is feasible for robot i and robot j confirms the formation request at time t_j, there exists a finite time $t > t_j$ such that: (i) the trajectory $\mathbf{p}_i([t_j, t])$ of robot i satisfies the formation task φ^{f}_{ij,t_j}; (ii) robot j achieves the desired formation described by c_{ij} with prescribed performance imposed by the performance function $\rho_{ij}(t)$ during the time interval $[t_j, t]$; and (iii) a release signal is sent to robot j at time t.*

Proof. The first part of the theorem can be proved in a similar way to Theorem 7.8. In particular, the newly-confirmed formation task is incorporated into the intersection automaton $\mathcal{A}_{[\varphi_i]}$. Since it is co-safe and feasible, there exists a finite time $t > t_j$ that $q'_{i,t}|_Q$ reaches the accepting states of $\mathcal{B}_{\varphi^{\mathtt{f}}_{ij,t_j}}$ and thus $\mathtt{trace}_i([t_j, t])$ satisfies $[\varphi^{\mathtt{f}}_{ij,t_j}]$. Subsequently the fact that the resulting trajectory $\mathbf{p}_i([t_g, t])$ satisfies the RTL formula $\varphi^{\mathtt{f}}_{ij,t_j}$ can be shown as in Theorem 7.5. At the same time, a confirmation message is sent from robot i to robot j. Regarding the second part, Lemma 7.4 indicates that the formation control law (7.14) guarantees that robot j will converge to the desired relative formation described by c_{ij} with prescribed performance until a release signal is received at time t, which completes the proof. ∎

7.5 Overall structure

In this subsection, we formally construct the overall hybrid control architecture, based on the aforementioned analysis. As depicted in Figure 7.2, the architecture is

organized in four interconnected modules that run concurrently in real time.

Communication module. It receives the contingent service request $\varphi_{ji}^{\mathtt{s}}$ and formation requests c_{ji}, $\rho_{ji}(t)$ from all its neighbor $j \in \mathcal{N}_i$ as well as the events concerning $r_{ji}^{\mathtt{f}}$, $r_{ji}^{\mathtt{s}}$ and z_{ij} from the event monitoring module. It also confirms or refuses these requests based on the communication protocol described in Section 7.3. Thus the confirmed service request $\varphi_{ji}^{\mathtt{s}}$ is passed to the discrete planner module and the confirmed formation request by c_{ji}, $\rho_{ji}(t)$ is passed to the hybrid controller module. Moreover, it is also responsible for sending the contingent service requests $\varphi_{ij}^{\mathtt{s}}$ and formation requests c_{ij}, $\rho_{ij}(t)$ to all its neighbors $j \in \mathcal{N}_i$. Additionally, in case a formation request is confirmed by robot j, the associated formation task $\varphi_{ij}^{\mathtt{f}}$ is sent to the discrete planner module. Finally, the release message z_{ji} received from neighbor j is sent directly to the hybrid control module.

Discrete planner module. It computes the initial discrete plan $\tau_{i,0}$ as described in Section 7.4.1. Afterwards, each time a confirmed service request $\varphi_{ji}^{\mathtt{s}}$ or a formation task $\varphi_{ij}^{\mathtt{f}}$ is received from the communication module, it reconstructs the intersection automaton $\mathcal{A}_{[\varphi_i]}$ and recomputes the production automaton $\mathcal{A}_{p,i}$, based on which the discrete plan τ_i is updated, as shown in Section 7.4.2. Moreover, it updates the next goal region based on the reaching event generated by the event monitoring module. Finally, it sends the next goal $\pi_{ig} \in \Pi_i$ to the hybrid controller module.

Event monitoring module. It monitors and passes the reaching event to the discrete planner and the hybrid controller modules. Moreover, events related to the satisfaction of service or formation tasks (i.e., $r_{ji}^{\mathtt{f}}$, $r_{ji}^{\mathtt{s}}$ and z_{ij}) are also acknowledged to the communication and hybrid controller modules.

Hybrid controller module. It is in charge of switching between the navigation and formation control modes. As shown in Figure. 7.3, the hybrid control automaton is defined as a tuple $H_{cont} \triangleq (Q, P, \text{Init}, f, D, O, E, G)$, where $Q = \{q_{nav}, q_{form}\}$ is the set of discrete states; $P \subseteq \pi_{i0}$ is the continuous state; $\text{Init} = q_{nav} \times p_i(0)$ is the initial state; $f(q_{nav}, p_i) = -\nabla_{q_i} \Phi_i(p_i, \pi_{ig})$ is the continuous dynamics by (7.10) within the discrete state q_{nav}, where π_{ig} is the goal region by the discrete planner; $f(q_{form}, p_i) = K_{ji}\, \varepsilon_{ji}\, e_{ji}/\rho_{ji}(t)$ is the continuous dynamics by (7.14) within the discrete state q_{form}, where c_{ji}, $\rho_{ji}(t)$ are the formation request from the communication module; $D(q_{nav}) = D(q_{form}) \subseteq \pi_{i0}$ are the domains of the continuous state; $O = \{o_1, o_2, o_3\}$ is the set of external events, where o_1 is "$R_g = \top$", o_2 is "$o_{ij}^{\mathtt{f}} = \top$", and o_3 is "$z_{ji} = \top$"; $E = \{(q_{nav}, q_{nav}), (q_{nav}, q_{form}), (q_{form}, q_{nav})\}$ involves the allowed discrete transition edges; $G(q_{nav}, q_{nav}) = o_1$, $G(q_{nav}, q_{form}) = o_2$, $G(q_{form}, q_{nav}) = o_3$ indicate the guard over each discrete transition.

Theorem 7.10. *Given the aforementioned local hybrid control architecture and under the assumption that all service and formation requests are feasible, then each local task φ_i is satisfied by the trajectory of robot i as $t \to \infty$, $\forall i \in \mathcal{N}$.*

Proof. As mentioned in Section 7.1.2, each local task φ_i consists of three parts, i.e., $\varphi_i^{\mathtt{l}}$, $\varphi_i^{\mathtt{s}}$ and $\varphi_i^{\mathtt{f}}$. At first, notice that Theorem 7.5 guarantees that if no service or formation requests are confirmed by robot i, then its trajectory would satisfy the

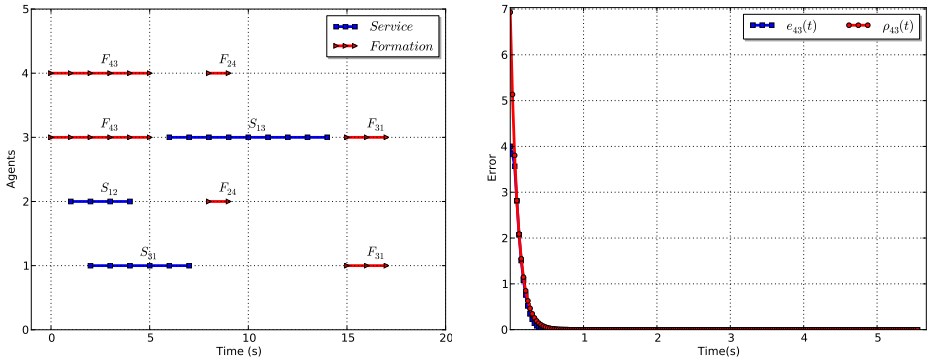


Figure 7.4: Left: the service or formation tasks each robot is engaged. The label S_{ij} indicates that robot j is performing a service request from robot i and F_{ij} indicates that robot j is during a formation request from robot i. Right: evolution of the formation error along with the performance function for the formation request $\varphi^{\mathsf{f}}_{43}$ during $[0s, 5.5s]$.

local task φ^{l}_i. Subsequently, we show that both service requests φ^{s}_i and formation requests φ^{f}_i are fulfilled as long as robot i exchanges requests with its neighbors in real-time. In case robot i confirms a service request $\varphi^{\mathsf{s}}_{gi,\,t_g}$ from robot g, then Theorem 7.8 ensures that there exists a finite time $t_i s > t_g$ such that the trajectory $\mathbf{p}_i([t_g,\, t_i^s])$ satisfies $\varphi^{\mathsf{s}}_{gi,\,t_g}$. Moreover, notice that the same also holds for all services requests confirmed by robot i, which implies that φ^{s}_i is fulfilled. Alternatively, if a formation request $\varphi^{\mathsf{f}}_{gi,\,t_g}$ is confirmed, then Theorem 7.9 ensures that there exists a finite time $t_i^f > t_g$ such that the trajectory $\mathbf{p}_g([t_g,\, t_i^f])$ of robot g satisfies the formation task $\varphi^{\mathsf{f}}_{gi,t_g}$ in a finite time $t > t_g$ and meanwhile the formation between robot i and g is kept during time $[t_g,\, t_i^f]$. Furthermore, since the same holds for all formation requests confirmed by robot i, it implies that φ^{f}_i is fulfilled. Thus, all three parts of φ_i are fulfilled and thus φ_i is satisfied for each robot $i \in \mathcal{N}$. ■

7.6 Case study

In this section, we present a simulated paradigm of four autonomous robots with heterogeneous capabilities. The proposed algorithms are implemented in Python 2.7 and all simulations are carried out on a desktop computer.

Workspace and task description

The area of size $4m \times 4m$ is shown in Figure 7.6, within which there are seven sphere regions $\pi_0, \pi_1, \cdots, \pi_6$ of interests with different radius. They are labeled by the propositions $R_0, R_1, \cdots, R_6$. Moreover the workspace is bounded by the circle $\mathcal{B}([2, 2],\, 2.35)$. Additionally, denote by $\mathfrak{R}_1$, $\mathfrak{R}_2$, $\mathfrak{R}_3$, $\mathfrak{R}_4$ the team of four robots that satisfy (7.2). Their neighboring set is defined by: $\mathcal{N}_1 = \{\mathfrak{R}_2, \mathfrak{R}_3\}$, $\mathcal{N}_2 = \{\mathfrak{R}_1, \mathfrak{R}_4\}$,

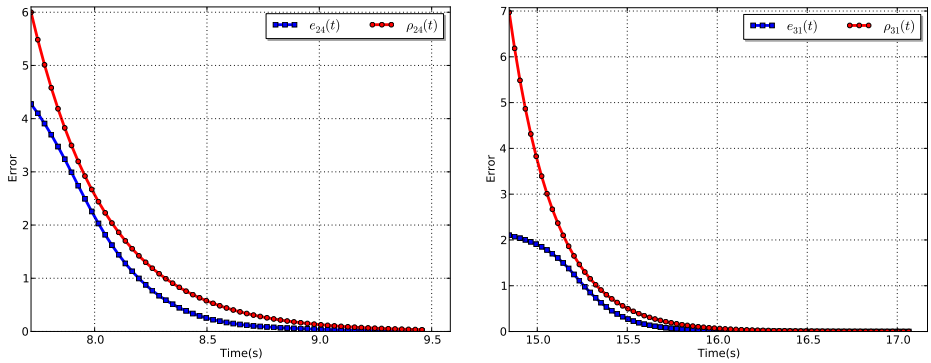


Figure 7.5: Evolution of the formation error along with the associated performance function, for the formation requests $\varphi^{\mathsf{f}}_{24}$ during $[7.7s, 9.4s]$ and $\varphi^{\mathsf{f}}_{31}$ during $[14.8s, 17.1s]$.

$\mathcal{N}_3 = \{\mathfrak{R}_1, \mathfrak{R}_4\}$, $\mathcal{N}_4 = \{\mathfrak{R}_2, \mathfrak{R}_3\}$. They initialize from $(0.3, 0.5)$, $(3.8, 0.5)$, $(2.0, 3.5)$ and $(0.3, 3.5)$, respectively and their communication radius is set to $1m$.

For robot $\mathfrak{R}_1$, the local task $\varphi^{\mathsf{l}}_1 = \big(\Diamond(R_2 \wedge \Diamond R_5)\big) \wedge \big(\Box\Diamond R_3 \wedge \Box\Diamond R_6\big)$ requires that it first visits region π_2, then π_5 and surveils over regions π_3 and π_6. Its predefined service requests concern $\mathfrak{R}_2$ (i.e., $\varphi^{\mathsf{s}}_{12} = \Diamond R_6$) and $\mathfrak{R}_3$ (i.e., $\varphi^{\mathsf{s}}_{13} = \Diamond(R_2 \wedge \Diamond R_5)$), while there are no formation requests to either $\mathfrak{R}_2$ or $\mathfrak{R}_3$. Robot $\mathfrak{R}_2$ has the local task $\varphi^{\mathsf{l}}_2 = \Box \Diamond (R_2 \wedge \Diamond R_5) \wedge \Box\neg R_0$ indicating that it should surveil over regions π_2 and then π_5, and avoids π_0 all the time. Moreover, there are no service requests to $\mathfrak{R}_1$ or $\mathfrak{R}_4$, while the formation request to $\mathfrak{R}_4$ is given by $c_{24} = (-0.5, 0)$, $\rho_{24,0} = 6$, $\rho_{24,\infty} = 0.0001$ and $l_{24} = 3$. under formation task $\varphi^{\mathsf{f}}_{24,} = \Diamond R_3$. Robot $\mathfrak{R}_3$ has the local task $\varphi^{\mathsf{l}}_3 = (\Diamond R_3) \wedge (\Diamond R_4) \wedge (\Diamond \Box R_5) \wedge (\Box\neg R_0)$, i.e., it should visit regions π_3, π_4 and π_5 in any order, and avoid π_0 all the time. The predefined service request to $\mathfrak{R}_1$ is given by $\varphi^{\mathsf{s}}_{31} = \Diamond(R_1 \wedge \Diamond R_2)$ and the formation requests to $\mathfrak{R}_1$ is given by $c_{31} = (0.5, 0)$, $\rho_{31,0} = 7$, $\rho_{31,\infty} = 0.0001$ and $l_{31} = 4$. There are no service or formation requests to $\mathfrak{R}_4$. Finally, robot $\mathfrak{R}_4$ has the local task $\varphi^{\mathsf{l}}_4 = \Box \Diamond R_3 \wedge \Box \Diamond R_4$ that requires it surveil regions π_3 and π_4. There are no service requests to $\mathfrak{R}_2$ or $\mathfrak{R}_3$, while the formation request to $\mathfrak{R}_3$ is defined as $c_{43} = (0.3, 0)$, $\rho_{43,0} = 7$, $\rho_{43,\infty} = 0.0001$ and $l_{43} = 10$ under the formation task $\varphi^{\mathsf{f}}_{43} = \Diamond R_3$.

Simulation results

The system was simulated for $20s$ and it should be noted that all service and formation requests defined earlier were exchanged and accomplished by the robots after $17.1s$. More specifically, at $t = 0$, the initial synthesis of the discrete plan is done locally by each robot. The initial plan of $\mathfrak{R}_1$ is given by $\tau_1 = \pi_2\pi_5\pi_3\pi_6(\pi_0\pi_3)^\omega$; the initial plan of $\mathfrak{R}_2$ is $\tau_2 = \pi_3(\pi_2\pi_5)^\omega$; the initial plan of $\mathfrak{R}_3$ is $\tau_3 = \pi_5\pi_4\pi_3(\pi_5)^\omega$; the initial plan of $\mathfrak{R}_4$ is $\tau_4 = \pi_6(\pi_4\pi_3)^\omega$. It can be easily verified that all satisfy the respective local tasks. Snapshots of the robots' trajectories at key time instants are shown in Figure 7.6. In particular, at $t = 0.1s$, $\mathfrak{R}_3$ receives $\mathfrak{R}_4$'s formation request

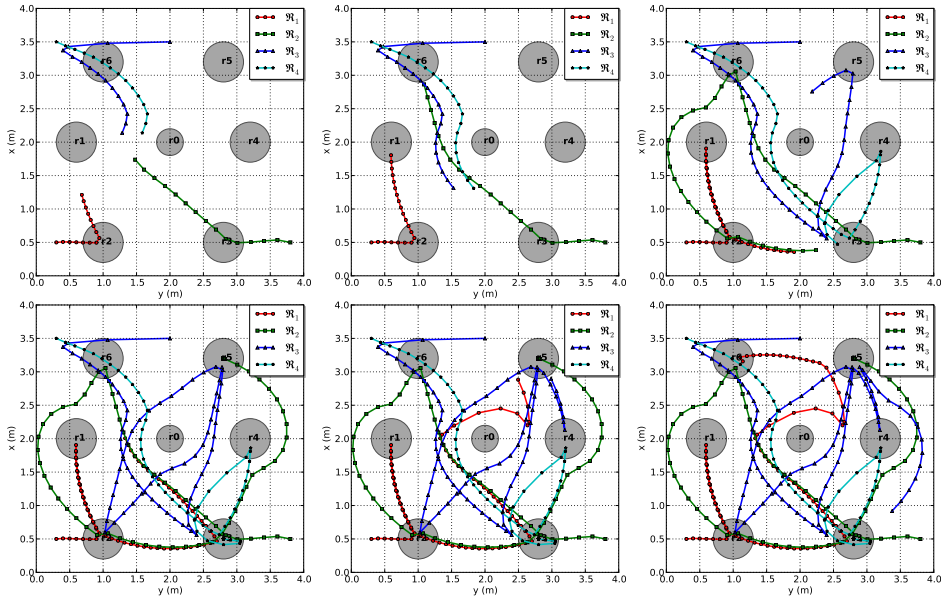


Figure 7.6: Snapshots of the robots' trajectories at $3s$, $4.2s$, $9.1s$, $10.5s$, $17s$ and $20s$. Descriptions of these events can be found in Section 7.6.

and executes it until $t = 5.5s$, when $\mathfrak{R}_4$ finishes the formation task $\Diamond R_3$ by reaching π_3. At $0.6s$, $\mathfrak{R}_2$ confirms service request $\Diamond R_6$ from $\mathfrak{R}_1$ and changes its local plan to visit π_6 first, which is done at $4.2s$. At the same time, $\mathfrak{R}_1$ confirms service request $\Diamond(R_1 \wedge \Diamond R_2)$ from $\mathfrak{R}_3$ and changes its local plan to visit π_1 and then π_2 first, which is done at $7.1s$. During $[7.7s, 9.4s]$, $\mathfrak{R}_4$ carries out a relative formation task with $\mathfrak{R}_2$ until $\mathfrak{R}_2$ reaches π_2. Then $\mathfrak{R}_3$ confirms the service request $\Diamond(R_2 \wedge \Diamond R_5)$ from $\mathfrak{R}_1$ at $5.5s$. This service is accomplished at $14.8s$ by $\mathfrak{R}_3$'s detour to π_2 first and π_5 afterwards. At last, $\mathfrak{R}_1$ establishes a relative formation with $\mathfrak{R}_4$ at $14.8s$ until robot $\mathfrak{R}_4$ finishes the formation task $\Diamond R_3$ by reaching π_3 at $17.1s$. By then, all predefined contingent tasks were fulfilled and no further contingent tasks will be exchanged. Subsequently, each robot continues executing its local plans to fulfill its local task. Finally, regarding the formation requests $\varphi^{\mathrm{f}}_{43}$, $\varphi^{\mathrm{f}}_{24}$ and $\varphi^{\mathrm{f}}_{31}$, the evolution of the formation errors as depicted in Figures 7.4 and 7.5 meets the specifications imposed by the performance functions. An accompanying video may be found in [53].

7.7 Summary

In this chapter we presented a hybrid control strategy for multi-robot systems under static local task and contingent temporal tasks. It has been shown that both the contingent service and formation tasks are fulfilled, while the prescribed formation constraints are always respected.

Chapter 8

Software implementation and experiments

This chapter presents the software implementation for some algorithms we have described in the previous chapters. Then we show some experiment results based on this implementation over different robotic platforms.

8.1 Software package P-MAS-TG

There are some existing model-checking-based motion planning software packages, e.g., the LTL motion planner (LTLMoP) by [39] and the LTL robust multi-robot planner (LROMP) by [142]. However since both are simulation-oriented, it is not straightforward to use them for controlling physical robots that interact with real-time measurements. The software package developed in this thesis is named by "planner for multi-agent systems with temporal goals" (P-MAS-TG), available at [50]. It features both the motion planning and action planning modules described in Chapters 3 and 5, and the partially-feasible tasks addressed in Chapter 4.

8.1.1 Robot control architecture

Robot operating system (ROS) is currently the most popular operating system for a large variety of robotic platforms [128]. ROS is a flexible framework for writing robot softwares. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It keeps a minimum and simple interfaces between different functional packages. More importantly, the real-time message passing and handling services of ROS allow multiple ROS packages to be running in parallel, which is extremely beneficial for real-time applications. For instance, the planning algorithm can be running to revise, update and improve the plan while the robot is executing the continuous controller, in contrary to the blocking scenario where the robot can only move after it has finished the planning phase.

A running ROS consists of a single ROS core and several running processes as ROS nodes. The ROS core provides a distributed computing environment allowing multiple

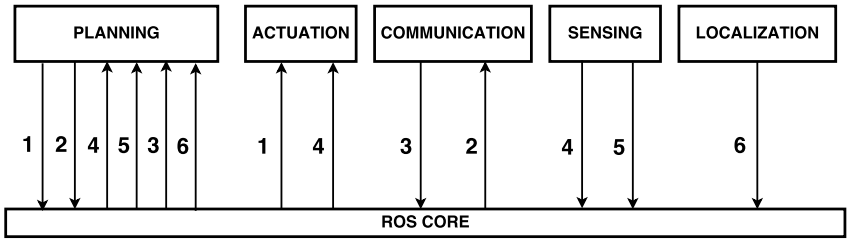


Figure 8.1: Architecture for the ROS-based implementation. Name of the topics: 1. next goal region or next action; 2. request to external source; 3. reply from communication; 4. observation from sensing; 5. confirmation for motion or action; 6. robot's position.

processes to be running simultaneously. They communicate via ROS messaging services and can be running on different robots, computers and processing units even with different programming languages. A ROS node can be both publisher and subscriber: it can subscribe to as many topics and publish to as many topics. Once a ROS node publishes a message to a topic, the ROS core will distribute this message to all nodes that have subscribed to this topic. Normally once a ROS node receives a new message under a topic, a pre-defined callback function is called automatically such that this message can actually be used by this node.

We divide the essential functionalities for the motion and task planning scheme into five modules: planning, actuation, sensing, localization and communication. Ideally they are located in five different ROS nodes, as shown in Figure 8.1. Each node is connected to the ROS core by directed arrows as the message flow, above which is the topic name. Outgoing arrows from a node indicate that this node can publish messages over the topics above the arrows, while incoming arrows indicate that this node is subscribed to those topics above the arrows. Thus the integration process only involves agreement on the topic names and the message structures. Note that the five-module structure may vary for different applications. For example, the sensing and localization nodes might be merged if they rely on the same hardware and are outputs of the same program.

8.1.2 ROS node for planning

The main contribution lies in our ROS-node package for the motion and task planning algorithms presented in Chapters 3, 4, and 5. The whole package is written in Python due to its fast prototyping and multi-platform accessibility. We use the ROS library "Rospy" as the interface, which is a client library that enables the creation of ROS topics, services and parameters for python-based programs [128]. The structure of the ROS node for planning is shown in Figure 8.2. In the initialization step, the initial finite transition system is encoded by the NetworkX graph structure [70]; the hard and soft task specifications are given as LTL formulas in the string format. When the planner node starts running, it firstly translates the hard and soft specification

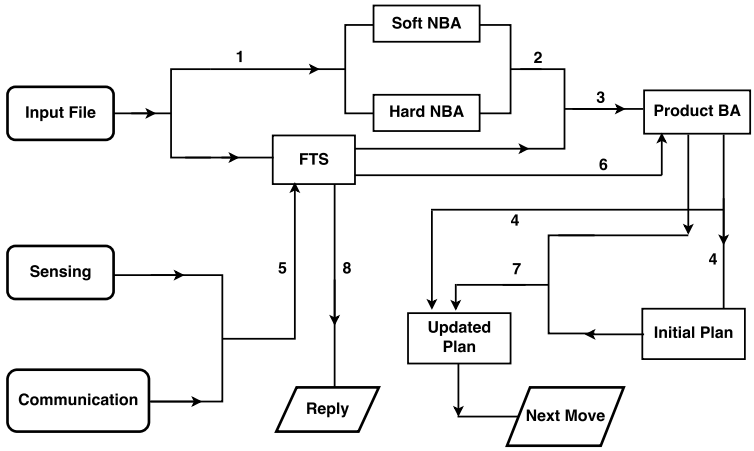


Figure 8.2: Data flow within planning package. Input data is in rounded rectangular; internal variables are in rectangular; output data is in parallelogram. Texts on the arrows are algorithms: 1. LTL formulas to Büchi automaton and encodings; 2. automaton intersection by Definition 4.3; 3. product automaton by Definition 4.1; 4. plan synthesis by Algorithm 3.1; 5. transition system update; 6. product automaton update; 7. validate and revise plan by Algorithm 4.4; 8. reply to requests by Algorithm 4.5.

formulas into Büchi automaton by running the executable file from [41]. The output text file is then parsed and encoded as a NetworkX graph. The propositional logic formula representing all input alphabets for each transition is encoded by binary decision diagram (BDD) using the Lex parser [74]. It allows for: (i) fast encoding and evaluation of propositional formulas when constructing the product automaton; (ii) efficient integration with the distance evaluation. Both the full construction and on-the-fly construction are implemented.

For complex tasks of practical interest, it may take several minutes to compute the initial plan, which is not desirable for real-time applications. Since in ROS all nodes can be running simultaneously, it allows us to exploit the notion of *any-time* graph search algorithm [102, 144]. Normally anytime planning algorithms find a preliminary, possibly highly suboptimal solution quickly within a bounded time. While this preliminary plan is being executed, the planner continues to improve this plan until the optimal one is found. This approach works particularly well for static and fully-known workspaces. However if the workspace is dynamic, the robot has to replan frequently during the execution. Then the anytime planner may loss its anytime property as it needs to generate new and suboptimal plans from scratch frequently. Instead, the local revising algorithm by Algorithm 4.4 and the event-based optimality check are more suitable.

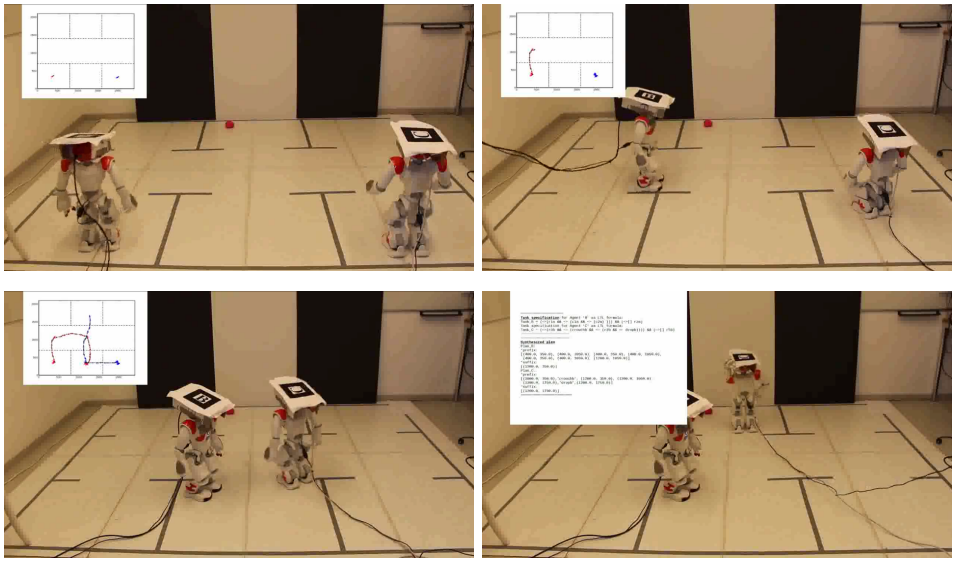


Figure 8.3: The snapshots of two NAO robots fulfilling their local tasks.

8.2 Experiment demonstrations

In this section, we present four different experiment setups and results where the P-MAS-TG package serves as the main planning module.

Experiment one

In the *first* experiment we demonstrate the general motion and action planning framework proposed in Chapters 3 and 5.

Experiment setup. The workspace represents an office environment as shown in Figure 8.3, consisting of six rooms and a corridor in the middle. The corridor is partitioned into three smaller parts, i.e., this workspace has nine regions, represented by labels "r1,⋯, r6" and "c1, c2, c3". The object of interest is a red ball, represented by the label "ball". The cost of moving from one partition to another is estimated by the Euclidean distance between their centers. The labeling function is given by the region and labels (from bottom to top, from left to right): (π_1, {r1}), (π_2, {r2}), (π_3, {r3}), (π_4, {c1}), (π_5, {c2}), (π_6, {c3}), (π_7, {r4}), (π_8, {r8, ball}), (π_9, {r9}). The robots we deployed are two NAO robots as shown in Figure 8.3. It is equipped with three basic control modules "move_x(·)", "move_y(·)" and "turn(·)". Namely, it can move forward, sideways and turn itself by the given speed. Besides these motion primitives, we have implemented various NAO action primitives, such as "crouch" to crouch; and "wave" to wave one hand. They are implemented using the Behavior Tree control scheme from [109]. We rely on the ARToolKit single camera system to track real-time positions of the NAO robots.

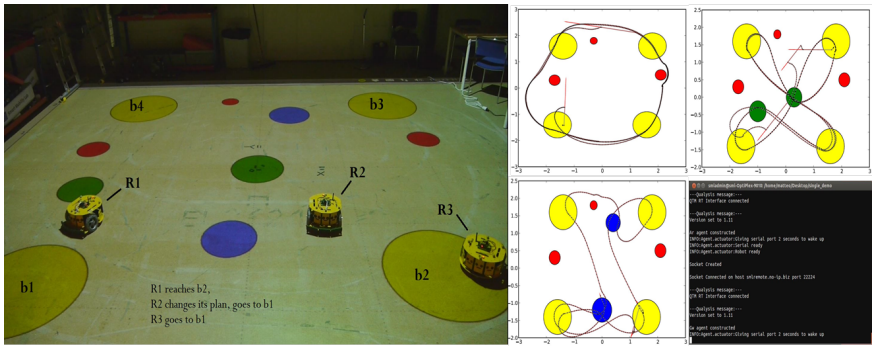


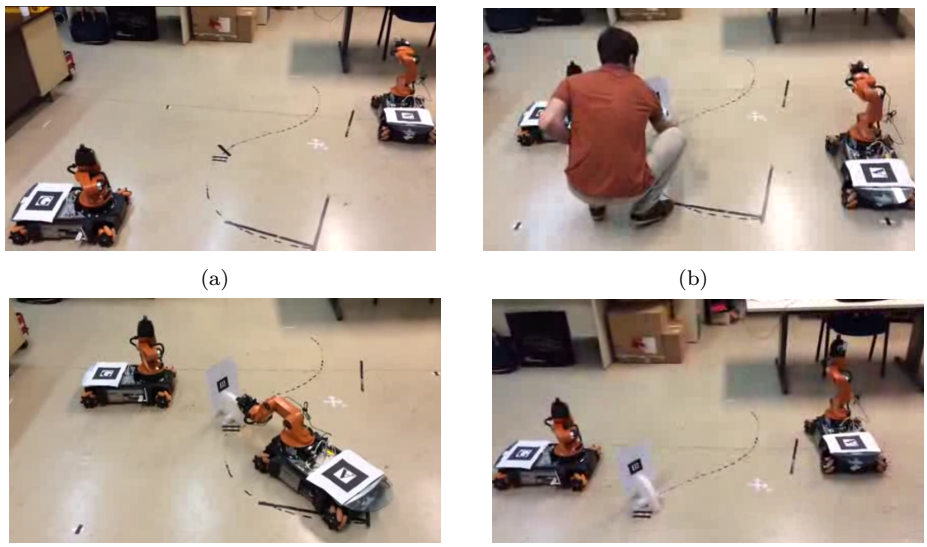
Figure 8.4: Left: the workspace with three Nexus vehicles. Right: the recorded panels, including the message log and vehicles' trajectories.

Results description. This section describes the experiment results when the local tasks are given as: $\varphi_1 = (\diamondsuit(r1 \wedge \diamondsuit(c1 \wedge \diamondsuit(c2)))) \wedge (\diamondsuit \square r2)$ and $\varphi_2 = (\diamondsuit(r3 \wedge \diamondsuit(\mathsf{crouch} \wedge \diamondsuit\mathsf{wave}))) \wedge (\diamondsuit \square r5)$. Namely, the NAO robot to the right needs to crouch first, traverse the workspace, wave their hand when detecting a ball and then stays at that region; another NAO robot needs to cross the corridor and reach the adjacent room. Note that the inter-robot collision avoidance is integrated with the navigation control scheme, i.e., one robot would stop and wait until another robot passes. It took $0.04s$ to synthesize the optimal plans: $\tau_1 = \pi_1 \pi_4 \pi_5 (\pi_2)^\omega$ and $\tau_2 = \pi_3 \, \mathtt{crouch} \, \pi_6 \pi_5 \pi_8 \, \mathtt{wave} \, (\pi_8)^\omega$. A potential collision is resolved locally at region π_5. The real-time behaviors and simulated trajectories are shown in Figure 8.3 at different time instants. It can be seen that both tasks are fulfilled. The complete video for this experiment can be found in [60].

Experiment two

The *second* experiment demonstrates the real-time knowledge transfer and plan adaptation scheme proposed in Chapter 4.

Experiment setup. The experiment setup is similar to the simulated case presented in Section 4.5: three nexus ground vehicles are deployed in the Smart Mobility Lab at KTH, as shown in Figure 8.4. The lab workspace is $6m \times 6m$, within which there are four base stations at four corners in yellow, two blue regions, two green regions, three obstacle-regions in red. The local tasks are specified as follows: vehicle $\mathcal{R}_1$ has the surveillance task over four base stations and avoids all obstacles; vehicle $\mathcal{R}_2$ needs to visit the blue regions and all base stations infinitely often in an interleaved order while avoiding all obstacles; vehicle $\mathcal{R}_3$ needs to visit the green regions and all base stations in an interleaved manner. The corresponding LTL formulas are formulated similarly as in Section 4.5. The real-time feedback of each vehicle's position is obtained from the motion capture system "Qualisys". Obstacle detection are modeled by the on-board sonar sensors. Communication between the



(a) (b)

(c) (d)

Figure 8.5: Experiment snapshots for the hybrid control of two youBots.

control PC and vehicles are through wireless modules.

Results description. Note that the workspace model is only partially known to each robot initially. In particular, $\mathcal{R}_1$ knows the location of all base stations but none of the obstacle regions while $\mathcal{R}_2$ and $\mathcal{R}_3$ only know the location of one base station and one green or blue regions. It means that all local tasks are initially *infeasible* and the planning scheme proposed in Algorithm 4.6 is needed, including the initial balanced plan synthesis, the knowledge-transfer module and the real-time plan adaptation algorithm. The experiment results are similar to the simulated case in Section 4.5. Figure 8.4 shows the graphic interface to monitor the progress. The experiment video is available online [53].

Experiment three

The *third* experiment demonstrates the collaborative motion and action planning module in Chapter 5 and the real-time plan adaptation scheme through knowledge transfer from Chapter 4.

Experiment setup. The experiment setup involves two youBots, a localization system, one object of interest as shown in Figure 8.5. Each youBot is equipped with one gripper and can be controlled to move freely in the obstacle-free workspace. We rely on the same localization system as for the NAO robots. There are some predefined regions of interest for each youBot: the youBot with marker "G" has one in the middle as "r_3" and two on the left as "r_1, r_2", while the youBot with marker "A" has one in the middle as "r_4" and two on the right as "r_5, r_6". The

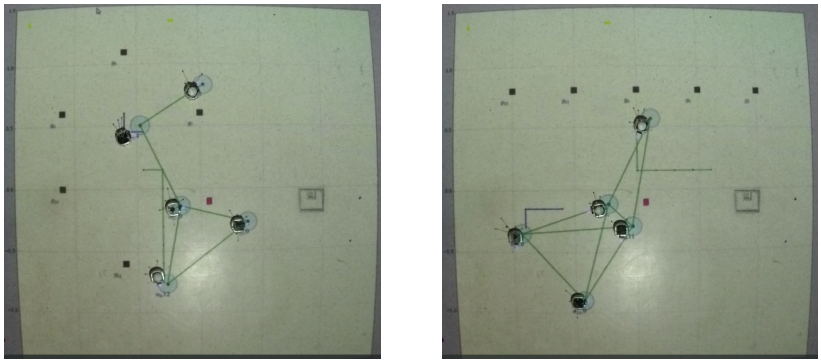


Figure 8.6: Snapshots of experiment four at $25s$ and $230s$.

object of interest to youBot "A" is in one of its known regions. The precondition and effect of actions "`pick`" and "`drop`" are omitted here for brevity. The local tasks are defined as follows: youBot "G" is required to surveil over its two regions of interest on the left side, or by the LTL formula $\varphi_G = \square \diamond r_1 \wedge \square \diamond r_2$; youBot "A" is required to go to region r_4, pick up the object of interest and transport it to region r_5, then surveil over regions r_5 and r_6, or by the formula $\varphi_A = \diamond\big((r_4 \wedge \diamond \texttt{pick}) \wedge \diamond(r_5 \wedge \diamond \texttt{drop})\big) \wedge (\square \diamond r_5 \wedge \square \diamond r_6)$.

Results description. Each youBot synthesizes its balanced plan initially and starts executing it. Since no object of interest is at region r_4 initially when the system starts, youBot "A" can not perform the action "`pick`" there. Instead, it surveils over regions r_5 and r_6 according to its balanced plan, as shown in Figures 8.5a-8.5b. For youBot "G", it fulfills its task specification by visiting regions r_1 and r_2 repetitively, as shown in Figures 8.5a-8.5d. After some time, the object of interest is placed *manually* close to region r_3, as shown in Figure 8.5b. Afterwards, youBot "G" discovers this object in r_3 and then broadcasts this knowledge to youBot "A". As a result, youBot "A" incorporates this information into its workspace model and updates its local plan to improve the satisfiability. According to the revised plan, it navigates to region r_4 and perform the action "`pick`", as shown in Figure 8.5c. After the successful grasp, this object is then transported to region r_5 as specified in φ_A, which is shown in Figure 8.5d. Afterwards, youBot "A" continues visiting regions r_5 and r_6 repetitively. The complete experiment video with more detailed descriptions can be found online [64].

Experiment four

In the *forth* experiment, we demonstrate the EGGs-based hybrid control scheme proposed in Section 6.2, which corresponds to the simulated case in Section 6.2.5.

Experiment setup. We implement the EGGs-based control scheme and the navigation controller proposed in Section 6.2 on a team of five Khepera II robots at the GRITS Lab of Georgia Tech, as shown in Figure 8.6. They are differential-driven

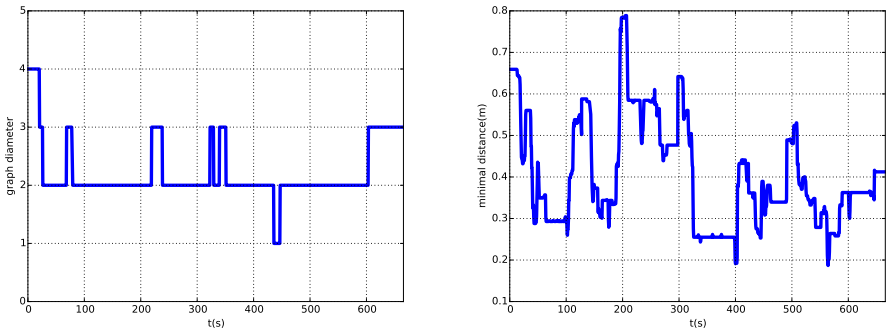


Figure 8.7: Evolution of the graph diameter (left) and the minimal distance among the robots (right). Same conclusions can be drawn as in Figure 6.6.

wheeled robots that communicates wirelessly with the base station computer. Their positions and orientations are tracked in real-time by the OptiTrack system. The message exchange among the robots, between the robots and OptiTrack system are all handled by ROS as described earlier. The robots' points of interest are scattered within the $3m \times 3m$ workspace and designed to be feasible by Assumption 6.2. The communication radius and safety distance for the multi-robot team are set to $0.9m$ and $0.15m$, respectively. The navigation controller in (6.36) is tuned properly to ensure that the robot can track the synthesized path consisting of discrete grid points. We omit the task description for each robot here since it is the same as the simulated case in Section 6.2.5 and described in detail in the experiment video [66]. Note that no robot actions are modeled and the synthesis of the motion plan relies on the package described earlier.

Results description. We run the system for 11 minutes and the robots have reached the fourth goal point in their discrete plans. The whole experiment is recorded by an overhead camera. Two snapshots of the experiment are shown in Figure 8.6, where the layout of the robots' goal points have been changed. We plot the diameter of $G(t)$ and the minimal distance between any two robots during the time period $[0s, 660s]$ in Figure 8.7. Based on this figure, we can verify that both continuous constraints are satisfied since $G(t)$ remains connected as its diameter is always lower than 6; no collision occur as the minimal distance is always above $0.15m$. The overall progress is similar to the simulated case from Section 6.2.5 and the complete experiment video can be found in [66].

8.3 Summary

We presented in this chapter the software implementation for some algorithms we have proposed in the previous chapters. We then demonstrated its application to various robotic platforms.

Chapter 9

Conclusion and future work

This chapter summarizes the content of the thesis and provides some future research directions.

9.1 Conclusion

At the beginning of this thesis, we provided some motivating applications from the industrial areas of service robots, autonomous cars and drones, where we also outlined several main challenges that arise when designing coordination and control strategies for networked multi-robot systems. Particularly, we emphasized the need for an intuitive but formal way for the end-users to specify high-level tasks, the importance of an automated scheme to control the robot to satisfy this task, the benefits of inter-robot communication to the planning process, and the advantage of coordinated collaboration among the robots. Then we presented the theoretical background and a review of some relevant work in Chapter 2.

In Chapter 3 we introduced the nominal framework for motion and task planning of a single dynamical robot given its motion task specified as a LTL formula. Under the assumption that the workspace is static and fully-known, we provided a systematic and automated scheme to synthesize firstly the discrete motion and task plan, and then the hybrid control strategy that drives the robot to execute this plan such that its resulting trajectory fulfills the given task.

Then in Chapter 4 we extended this nominal framework to a multi-robot system where a team of dynamical robots coexist within the same workspace and are interconnected by communication links. Each robot has a locally-assigned individual task specified as LTL formulas, which now contain hard and soft constraints. Due to limited knowledge about the workspace model and un-modeled dynamical constraints of the robots, the nominal approach becomes inadequate. Thus we proposed a cooperative knowledge-transfer scheme: while the system runs, each robot updates its knowledge about the workspace via its sensing capability and shares this knowledge with its neighbouring robots. Based on the knowledge update, each robot then verifies and revises its local plan in real time. It has been shown that the hard

specification is always fulfilled for safety and the satisfaction of the soft specification is improved gradually for performance.

It has been emphasized that collaborations among the robots can greatly improve the capability and efficiency of a multi-robot system. In Chapter 5, the robots' local tasks are specified as LTL formulas over both robot motion and actions, which are dependent due to the existence of collaborative actions. The proposed coordination scheme is distributed and guarantees that all local tasks are fulfilled. Any decision is made locally by each robot based on local computation and communication with neighboring robots. It is scalable and resilient to robot failures.

In Chapter 6, we considered inter-robot continuous constrains such as relative-distance constrains, collision avoidance and connectivity of the communication network. These constraints are closely related to the stability, safety and integrity of the overall team. We proposed two different hybrid control techniques to handle the relative-motion constrains along with the local temporal tasks: the first approach is based on the potential field and the second approach uses Embedded Graph Grammars (EGGs) as the main tool. Both approaches are shown to ensure the satisfaction of all local tasks and the relative-motion constraints.

Two common types of cooperative robotic tasks, namely service and formation tasks, were addressed in Chapter 7. The robots are allowed to request and exchange these tasks in real-time, with additional prescribed performance constraints. The proposed hybrid control strategy has been shown to guarantee that all local tasks are satisfied, while at the same time the service requests are fulfilled by one robot accomplishing a short-term task for another robot, and the formation requests are fulfilled by two robots keeping a relative-deployment requirement with predefined transient response imposed by an associated performance function.

Last but not least, we presented in Chapter 8 the software package that accompanies some algorithms proposed in the thesis and some experiment demonstrations over different robotic platforms.

9.2 Future work

In the future research, I would like to continue the research direction on hybrid control of networked multi-robot systems under complex temporal tasks, but with particular interest in the following aspects:

Distributed workspace abstraction

All the general hybrid control schemes mentioned in the thesis consist of three major steps: the finite abstraction of robot motion as a FTS; the discrete plan synthesis by graph search; and the hybrid controller construction, among which only the first step is not automated. In other words, it relies on a manual definition of the FTS based on the workspace model and the robot dynamics. Thus it is highly desirable to develop an automated abstraction method that achieves the same results. More importantly, given a team of dynamical robots, it would be beneficial for them to

exchange and *improve* their abstraction results through inter-robot communication, owing to the fact that each robot has a more accurate abstraction of the area where it is located. Especially between homogeneous robots, one robot's abstraction results can be used *directly* by another due to their identical dynamics. However it becomes more challenging to explore how to transfer abstraction results between heterogeneous robots given that their dynamics are different.

Task swapping and merging

Task swapping means that two robots exchange parts of their task specifications, while task merging means that one robot merges parts of another robot's task to its own. Both are powerful means of collaborative task execution for multi-robot systems, in order to either reduce the total execution cost or render the originally-infeasible tasks feasible for the whole team [59]. The direct question to ask is which two robots should swap or merge which parts of their task specifications, while ensuring that the temporal characteristics of the original local tasks are still respected. To answer that, we first need to address the essential notion of equivalence for task executions by *different* robots, which has not been investigated for neither homogeneous nor heterogeneous robots in literature. Additionally, as a side effect, it can be observed that there is often a trade-off between the potential cost reduction and the extra time delay caused by the synchronization during the task swapping or merging.

Dynamic recruitment among groups

As discussed in Chapter 5, recruitment is an essential tool for multi-robot systems where collaborations among the robots are required to fulfill all local tasks. The straightforward formulation is to define inter-robot dependency directly on robot identities, which often leads to a rigid coordination scheme that is neither scalable nor robust. Chapter 5 has shown that dependency by collaborative actions provides a more intuitive and flexible way to formulate collaborative tasks. The proposed coordination framework features a dynamic recruitment scheme that allows flexible membership and is resilient to robot failures, which is particularly suitable for large-scale multi-robot systems. To continue exploring this direction, it is worth considering multi-robot systems that consist of heterogeneous robots within which there are several *groups* of homogeneous robots, particularly to find out how the coordination scheme among different groups could be different from that within the same group. Also it would be interesting to investigate the possibility of introducing group leaders for the groups of homogeneous robots, and how this would affect the overall complexity and performance for coordination.

Information-driven control and coordination

Most of the motion control and coordination schemes mentioned in the thesis are task-driven in the sense that the main objective is to satisfy local tasks in a cost-

efficient way, while the availability and quality of real-time information, including sensory inputs and inter-robot communication, are taken for guaranteed. It however might not be the case in many practical applications, e.g., the sensory information might be uncertain when abstracting features in the workspace or the inter-robot communication might be subject to package loss, channel failure and delays. Thus it would be of practical interest to consider improving real-time information quality as an additional control objective, along with the satisfaction of high-level tasks. For instance, due to the existence of blocking obstacles, two robots may *actively* move towards each other to improve their communication quality, even if it means to deviate from their best routes; one robot may visit a region that is not in its optimal plan, in order to reduce uncertainty about that region for other robots. Suitable metrics to measure information quality should be investigated and then incorporated into the existing motion and task coordination framework.

Timed temporal tasks

Besides temporal properties that can be specified in LTL, another important performance metric of dynamical systems is the response time. Namely, it is not only desired the robot will accomplish a task, but also it should happen within a *desired time*. For example, instead of "stay in room A and then go to room B", the task specification may be "stay in room A for 5 minutes and then arrive room B in 10 minutes". This aspect will introduce two main challenges: firstly, how to modify the existing model-checking-based synthesis framework such that it incorporates *physical time*, instead of virtual discrete time or sampled time; secondly, how this timed metric will effect the design of the underlying continuous controller. In particular, the controller now needs to ensure both convergence as the traditional control objective and the response time as required. These two challenges may have to be tackled simultaneously as they are closely coupled, i.e., a reliable controller design with guaranteed response time would greatly simply the procedure for the synthesis of the timed discrete plan.

Bibliography

[1] A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 16(6): 624–641, 2010.

[2] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2):95, 2013.

[3] Aethon. The tug smart autonomous mobile robot. `http://www.aethon.com/tug/benefits/`, 2015.

[4] S. B. Akers. Binary decision diagrams. *Computers, IEEE Transactions on*, 100(6):509–516, 1978.

[5] Aldebaran. Robot nao. `http://www.aldebaran-robotics.com/en/`, 2014.

[6] Aldebaran. Robot pepper. `https://www.aldebaran.com/en/a-robots/who-is-pepper`, 2015.

[7] R. Alur, T. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[8] Amazon. Amazon prime air delivery system. `http://www.amazon.com/b?node=8037720011`, 2015.

[9] T. Arai, E. Pagello, and L. E. Parker. Editorial: Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, 18(5):655–661, 2002.

[10] E. Aydin Gol and M. Lazar. Temporal logic model predictive control for discrete-time systems. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pages 343–352. ACM, 2013.

[11] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press Cambridge, 2008.

[12] C. P. Bechlioulis and K. J. Kyriakopoulos. Robust model-free formation control with prescribed performance and connectivity maintenance for nonlinear multi-agent systems. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 4509–4514, 2014.

[13] C. P. Bechlioulis, G. Rovithakis, et al. Robust adaptive control of feedback linearizable mimo nonlinear systems with prescribed performance. *Automatic Control, IEEE Transactions on*, 53(9):2090–2099, 2008.

[14] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics. *Robotics & Automation Magazine, IEEE*, 14(1):61–70, 2007.

[15] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2689–2696, 2010.

[16] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[17] E. P. Chan and Y. Yang. Shortest path tree computation in dynamic graphs. *Computers, IEEE Transactions on*, 58(4):541–557, 2009.

[18] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *Robotics, IEEE Transactions on*, 28 (1):158–171, 2012.

[19] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at ltl model checking. In *Computer Aided Verification*, pages 415–427. Springer, 1994.

[20] E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

[21] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT press, 1999.

[22] T. L. Dean and M. P. Wellman. *Planning and control*. Morgan Kaufmann Publishers Inc., 1991.

[23] D. V. Dimarogonas and K. J. Kyriakopoulos. Decentralized motion control of multiple agents with double integrator dynamics. In *16th IFAC World Congress*, 2005.

[24] D. V. Dimarogonas and K. J. Kyriakopoulos. Decentralized navigation functions for multiple robotic agents with limited sensing capabilities. *Journal of Intelligent and Robotic Systems*, 48(3):411–433, 2007.

[25] D. V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos, and M. M. Zavlanos. A feedback stabilization and collision avoidance scheme for multiple independent non-point agents. *Automatica*, 42(2):229–243, 2006.

[26] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta. Automatic deployment of robotic teams. *Robotics & Automation Magazine*, 18(3):75–86, 2011.

[27] X. C. Ding, S. L. Smith, C. Belta, and D. Rus. Mdp optimal control under temporal logic constraints. In *Decision and Control and European Control Conference (CDC-ECC), IEEE Conference on*, pages 532–538, 2011.

[28] J. C. Doyle, B. A. Francis, and A. Tannenbaum. *Feedback control theory*. Macmillan Publishing Company New York, 1992.

[29] E. H. Durfee. Distributed problem solving and planning. In *Multi-agent systems and applications*, pages 118–149. Springer, 2006.

[30] M. Egerstedt and X. Hu. Formation constrained multi-agent control. *Robotics and Automation, IEEE Transactions on*, 17(6):947–951, 2001.

[31] M. Egerstedt, X. Hu, and A. Stotsky. Control of mobile platforms using a virtual vehicle approach. *Automatic Control, IEEE Transactions on*, 46(11): 1777–1782, 2001.

[32] Engadget. Self-driving taxis will begin trials in japan next year. `http://www.engadget.com/2015/10/01/robot-taxi-japan-2016/`, 2015.

[33] C. Exist. Don't worry: Drones are making sure you'll never have to go without wine. `http://www.fastcoexist.com/3040473/dont-worry-drones-are-making-sure-youll-never-have-to-go-without-wine`, 2015.

[34] G. E. Fainekos. Revising temporal logic specifications for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 40–45, 2011.

[35] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[36] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[37] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[38] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos. Decentralized multi-agent control from local ltl specifications. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 6235–6240, 2012.

[39] C. Finucane, G. Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1988–1993, 2010.

[40] M. Franceschelli, M. Egerstedt, and A. Giua. Motion probes for fault detection and recovery in networked control systems. In *American Control Conference (ACC), 2008*, pages 4358–4363, 2008.

[41] P. Gastin and D. Oddoux. Fast ltl to büchi automata translation. In *Computer Aided Verification*, pages 53–65. Springer, 2001.

[42] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16), 1998.

[43] M. Ghallab, D. Nau, and P. Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[44] A. Girard. Approximately bisimilar finite abstractions of stable linear systems. pages 231–244, 2007.

[45] C. Godsil and G. F. Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

[46] Google. Google self-driving car project. `http://www.google.com/selfdrivingcar/`, 2015.

[47] Gostai. Gostai by jazz security. `http://www.gostai.com/security/`, 2013.

[48] M. Guo. Cooperative motion and task planning under temporal tasks. *Licentiate Degree Thesis*, 2014.

[49] M. Guo. Cooperative motion and task planning under temporal tasks. 2014.

[50] M. Guo. P-mas-tg. `https://github.com/MengGuo/P_MAS_TG`, 2015.

[51] M. Guo and D. V. Dimarogonas. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local ltl specifications. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, 2013.

[52] M. Guo and D. V. Dimarogonas. Distributed plan reconfiguration via knowledge transfer in multi-agent systems under local ltl specifications. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[53] M. Guo and D. V. Dimarogonas. Multi-agent cooperative motion and task planning. `https://youtu.be/leTuzy3TIhI`, 2014.

[54] M. Guo and D. V. Dimarogonas. Simulation video of nominal scenario for cooperative motion and task planning. `https://vimeo.com/142983863`, 2015.

[55] M. Guo and D. V. Dimarogonas. Simulation of fault recovery scenario for cooperative motion and task planning. `https://vimeo.com/142984081`, 2015.

[56] M. Guo and D. V. Dimarogonas. Multi-agent plan reconfiguration under local ltl specifications. *The International Journal of Robotics Research*, 34(2): 218–235, 2015.

[57] M. Guo and D. V. Dimarogonas. Bottom-up motion and task coordination for loosely-coupled multi-agent systems with dependent local tasks. In *Automation Science and Engineering (CASE), IEEE International Conference on*, 2015. To appear.

[58] M. Guo, K. H. Johansson, and D. V. Dimarogonas. Motion and action planning under ltl specifications using navigation functions and action description language. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 240–245, 2013.

[59] M. Guo, K. H. Johansson, and D. V. Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5025–5032, 2013.

[60] M. Guo, M. Colledanchise, and A. Marzinotto. Experiment video for hybrid control of nao robot. `https://youtu.be/a75iwD5dFYY`, 2014.

[61] M. Guo, J. Tumova, and D. V. Dimarogonas. Cooperative decentralized multi-agent control under local ltl tasks and connectivity constraints. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 75–80, 2014.

[62] M. Guo, M. M. Zavlanos, and D. V. Dimarogonas. Controlling the relative agent motion in multi-agent formation stabilization. *Automatic Control, IEEE Transactions on*, 59(3):820–826, 2014.

[63] M. Guo, C. P. Bechlioulis, K. J. Kyriakopoulos, and D. V. Dimarogonas. Hybrid control of multi-agent systems with contingent temporal tasks and prescribed formation constraints. *Control of Network Systems, IEEE Transactions on. Submitted.*, 2015.

[64] M. Guo, M. Colledanchise, and A. Marzinotto. Experiment video for the hybrid control of youbot robot. `https://youtu.be/eWviu8We-vk`, 2015.

[65] M. Guo, M. Egerstedt, and D. V. Dimarogonas. Simulation video for eggs-based control. `https://vimeo.com/136210841`, 2015.

[66] M. Guo, M. Egerstedt, and D. V. Dimarogonas. Experiment video for eggs-based control. `https://vimeo.com/137872185`, 2015.

[67] M. Guo, J. Tumova, and D. V. Dimarogonas. Communication-free multi-agent control under local temporal tasks and relative-distance constraints. *Automatic Control, IEEE Transactions on. Conditionally accepted.*, 2015.

[68] M. Guo, J. Tumova, and D. V. Dimarogonas. Hybrid control of multi-agent systems under local temporal tasks and relative-distance constraints. In *Decision and Control (CDC), 2015 IEEE 54rd Annual Conference on*, 2015. To appear.

[69] M. Guo, M. Egerstedt, and D. V. Dimarogonas. Hybrid control of multi-robot systems using embedded graph grammars. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 2016. Sumbitted.

[70] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. *Technical Report*, 2008.

[71] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi. Towards manipulation planning with temporal logic specifications. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 346–352, 2015.

[72] W. Heemels, K. H. Johansson, and P. Tabuada. An introduction to event-triggered and self-triggered control. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 3270–3285, 2012.

[73] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[74] D. Horrigan. Lex parser. `https://github.com/pyrocms/lex`, 2013.

[75] IEEE. Look mom, no hands! `http://www.ieee.org/about/news/2012/5september_2_2012.html`, 2012.

[76] M. Ji and M. B. Egerstedt. Distributed coordination control of multi-agent systems while preserving connectedness. *Georgia Institute of Technology*, 2007.

[77] M. Ji, G. Ferrari-Trecate, M. Egerstedt, and A. Buffa. Containment control in mobile networks. *Automatic Control, IEEE Transactions on*, 53(8):1972–1975, 2008.

[78] S. Karaman and E. Frazzoli. Linear temporal logic vehicle routing with applications to multi-uav mission planning. *International Journal of Robust and Nonlinear Control*, 21(12):1372–1395, 2011.

[79] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[80] Y. Karayiannidis, D. V. Dimarogonas, and D. Kragic. Multi-agent average consensus control with prescribed performance guarantees. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 2219–2225, 2012.

[81] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[82] H. K. Khalil. *Nonlinear systems*, volume 3. Prentice Hall, 2002.

[83] K. Kim and G. E. Fainekos. Approximate solutions for the minimal revision problem of specification automata. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 265–271, 2012.

[84] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1):287–297, 2008.

[85] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *Robotics, IEEE Transactions on*, 26(1):48–61, 2010.

[86] M. Kloetzer, X. C. Ding, and C. Belta. Multi-robot deployment from ltl specifications with reduced communication. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 4867–4872, 2011.

[87] D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412–442, 1990.

[88] J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *The Journal of Machine Learning Research*, 7:1789–1828, 2006.

[89] R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[90] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Robotics and Automation (ICRA), 2007 IEEE International Conference on*, pages 3116–3121, 2007.

[91] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25(6): 1370–1381, 2009.

[92] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[93] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[94] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In *AAAI Conference on Artificial Intelligence*, 2015.

[95] J.-C. Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[96] S. M. LaValle. Rapidly-exploring random trees a new tool for path planning. *Technical Report 98-11*, 1998.

[97] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[98] S. M. LaValle and S. Hutchinson. Optimal motion planning for multiple robots having independent goals. *Robotics and Automation, IEEE Transactions on*, 14(6):912–925, 1998.

[99] S. M. LaValle and J. J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[100] D.-T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9 (3):219–242, 1980.

[101] T.-C. Lee, K.-T. Song, C.-H. Lee, and C.-C. Teng. Tracking control of unicycle-modeled mobile robots using a saturation feedback controller. *Control Systems Technology, IEEE Transactions on*, 9(2):305–318, 2001.

[102] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.

[103] S. R. Lindemann, I. I. Hussein, and S. M. LaValle. Real time feedback control for nonholonomic mobile robots with obstacles. In *Decision and Control (CDC), 2006 45th IEEE Conference on*, pages 2406–2411, 2006.

[104] S. C. Livingston, R. M. Murray, and J. W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5163–5170, 2012.

[105] S. G. Loizou and A. Jadbabaie. Density functions for navigation-function-based systems. *Automatic Control, IEEE Transactions on*, 53(2):612–617, 2008.

[106] S. G. Loizou and K. J. Kyriakopoulos. Closed loop navigation for multiple holonomic vehicles. In *Intelligent Robots and Systems (IROS), 2002 IEEE/RSJ International Conference on*, volume 3, pages 2861–2866, 2002.

[107] S. G. Loizou and K. J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on ltl specifications. In *Decision and Control (CDC), 2004. 43rd IEEE Conference on*, volume 1, pages 153–158, 2004.

[108] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *International Conference on Hybrid Systems: Computation and Control*, pages 353–362, 2013.

[109] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren. Towards a unified behavior trees framework for robot control. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5420–5427, 2014.

[110] M. Mazo and P. Tabuada. Decentralized event-triggered control over wireless sensor/actuator networks. *Automatic Control, IEEE Transactions on*, 56(10): 2456–2461, 2011.

[111] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. *Tech. Report CVC TR–98–003/DCS TR–1165*, 1998.

[112] K. L. McMillan. *Symbolic model checking*. Springer, 1993.

[113] J.-M. McNew and E. Klavins. Locally interacting hybrid systems with embedded graph grammars. In *Decision and Control (CDC), 2006 45th IEEE Conference on*, pages 6080–6087, 2006.

[114] J.-M. McNew, E. Klavins, and M. Egerstedt. Solving coverage problems with embedded graph grammars. In *Hybrid Systems: Computation and Control*, pages 413–427. 2007.

[115] S. Misra and B. J. Oommen. Dynamic algorithms for the shortest path routing problem: learning automata-based solutions. *Systems, Man, and Cybernetics, IEEE Transactions on*, 35(6):1179–1192, 2005.

[116] P. Ögren. Increasing modularity of uav control systems using computer game behavior trees. In *AIAA Guidance, Navigation and Control Conference, Minneapolis, MN*, 2012.

[117] P. Ögren, M. Egerstedt, and X. Hu. A control lyapunov function approach to multi-agent coordination. In *Decision and Control (CDC), 2001. Proceedings of the 40th IEEE Conference on*, pages 1150–1155, 2001.

[118] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *Automatic Control, IEEE Transactions on*, 51(3):401–420, 2006.

[119] E. P. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. Morgan Kaufmann Publishers Inc., 1989.

[120] D. Pickem, M. Egerstedt, and J. S. Shamma. Complete heterogeneous self-reconfiguration: Deadlock avoidance using hole-free assemblies. In *Estimation and Control of Networked Systems*, volume 4, pages 404–410, 2013.

[121] V. Raman and H. Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using ltlmop. In *Computer Aided Verification*, pages 663–668. Springer, 2011.

[122] W. Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multi-agent coordination. In *American Control Conference (ACC), 2005. Proceedings of the*, pages 1859–1864, 2005.

[123] Rethink. Rethink robotics. `http://www.rethinkrobotics.com/`, 2013.

[124] M. Reynolds. Continuous temporal models. In *AI 2001: Advances in Artificial Intelligence*, pages 414–425. Springer, 2001.

[125] S. Riko. Robot bear. `http://www.digitaltrends.com/cool-tech/riken-robear/`, 2015.

[126] E. Rimon and D. E. Koditschek. Exact robot navigation using cost functions: the case of distinct spherical boundaries in e^n. In *Robotics and Automation (ICRA), Proceedings of 1988 IEEE International Conference on*, pages 1791–1796, 1988.

[127] N. Robotics. Professional service robots: continued increase. `http://www.worldrobotics.org/index.php?id=home&news_id=262`, 2012.

[128] ROSWiki. Ros wikipedia. `http://wiki.ros.org/ROS/Introduction`, 2013.

[129] H. Saïdi and N. Shankar. Abstract and model check while you prove. In *Computer Aided Verification*, pages 443–454. Springer, 1999.

[130] V. Schuppan and A. Biere. *Shortest counterexamples for symbolic model checking of LTL with past*. Springer, 2005.

[131] B. Smith, A. Howard, J.-M. McNew, J. Wang, and M. Egerstedt. Multi-robot deployment and coordination with embedded graph grammars. *Autonomous Robots*, 26(1):79–98, 2009.

[132] S. L. Smith, J. Tumova, C. Belta, and D. Rus. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[133] E. D. Sontag. *Mathematical control theory: deterministic finite dimensional systems*, volume 6. Springer Science & Business Media, 2013.

[134] S. Srinivas, R. Kermani, K. Kim, Y. Kobayashi, and G. Fainekos. *A Graphical Language for LTL Motion and Mission Planning*. PhD thesis, Arizona State University, 2013.

[135] P. Tabuada and G. J. Pappas. Model checking ltl over controllable linear systems is decidable. pages 498–513, 2003.

[136] P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *Automatic Control, IEEE Transactions on*, 51(12):1862–1877, 2006.

[137] N. Y. Times. Some germans balk at plan to use drones to fight graffiti. `http://www.nytimes.com/2013/05/29/world/europe/in-germany-unease-at-plan-to-use-drones-to-fight-graffiti.html?_r=0`, 2013.

[138] TU/e. Amigo robots. `http://www.roboticopenplatform.org/wiki/AMIGO`, 2015.

[139] J. Tumova and D. V. Dimarogonas. A receding horizon approach to multi-agent planning from local ltl specifications. In *American Control Conference (ACC), 2014*, pages 1775–1780, 2014.

[140] J. Tumova, L. I. R. Castro, S. Karaman, E. Frazzoli, and D. Rus. Minimum-violation ltl planning with conflicting specifications. *arXiv preprint arXiv:1303.3679*, 2013.

[141] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus. Least-violating control strategy synthesis with safety rules. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2013.

[142] A. Ulusoy, S. L. Smith, X. C. Ding, and C. Belta. Robust multi-robot optimal path planning with temporal logic constraints. In *Robotics and Automation, 2012 IEEE International Conference on*, pages 4693–4698, 2012.

[143] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911, 2013.

[144] J. Van Den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Robotics and Automation (ICRA), Proceedings 2006 IEEE International Conference on*, pages 2366–2371, 2006.

[145] J. Van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation (ICRA), 2008 IEEE International Conference on*, pages 1928–1935, 2008.

[146] F. Van Harmelen, V. Lifschitz, and B. Porter. *Handbook of knowledge representation*. Elsevier, 2008.

[147] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*. IEEE Computer Society, 1986.

[148] Wikipedia. Moore's law. `https://en.wikipedia.org/wiki/Moore's_law`, 2015.

[149] Wired. Robots to live-stream night-time museum explorations. `http://www.wired.co.uk/news/archive/2014-08/12/robots-tate-britain`, 2014.

[150] Wired. The internet of things is far bigger than anyone realizes. `http://www.wired.com/insights/2014/11/the-internet-of-things-bigger/`, 2014.

[151] E. M. Wolff, U. Topcu, and R. M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *Decision and Control (CDC), IEEE 51st Conference on*, pages 3372–3379, 2012.

[152] L. A. Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.

[153] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 101–110, 2010.

[154] B. Yordanov and C. Belta. Formal analysis of discrete-time piecewise affine systems. *Automatic Control, IEEE Transactions on*, 55(12):2834–2840, 2010.

[155] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta. Temporal logic control of discrete-time piecewise affine systems. *Automatic Control, IEEE Transactions on*, 57(6):1491–1504, 2012.

[156] M. Zamani, P. Mohajerin Esfahani, R. Majumdar, A. Abate, and J. Lygeros. Symbolic control of stochastic systems via approximately bisimilar finite abstractions. *Automatic Control, IEEE Transactions on*, 59(12):3135–3150, 2014.

[157] M. M. Zavlanos, A. Jadbabaie, and G. J. Pappas. Flocking while preserving network connectivity. In *Decision and Control (CDC), 46th IEEE Conference on*, pages 2919–2924, 2007.

[158] M. M. Zavlanos, M. B. Egerstedt, and G. J. Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9): 1525–1540, 2011.