Cooperative Motion and Task Planning Under Temporal Tasks

MENG GUO

Licentiate Thesis
Stockholm, Sweden 2014

KTH School of Electrical Engineering
TRITA-EE 2014:005 Automatic Control Lab
ISSN 1653-5146 SE-100 44 Stockholm
ISBN 978-91-7595-016-7 SWEDEN

Akademisk avhandling som med tillstånd av Kungliga Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie licentiatexamen i elektro- och systemteknik torsdag 13 mars 2014, klockan 10.15 i sal L1, Kungliga Tekniska högskolan, Drottning Kristinaväg 30, Stockholm.

Tryck: Universitetsservice US AB

Abstract

Temporal-logic-based languages provide a formal and accurate way to specify complex motion and action missions for autonomous robots, beyond the classic point-to-point navigation task.

The first part of the thesis is devoted to the nominal scenario: an autonomous robot is given a motion task specified as Linear-time Temporal Logic (LTL) formulas. Under the assumption that the workspace is static and fully-known, we provide a systematic and automated scheme to synthesize both the discrete motion and task plan and the hybrid control strategy that drives the robot, such that the resulting trajectory fulfills the given task specification.

Limited knowledge about the workspace model, unforeseen changes in the workspace property and un-modeled dynamical constraints of the robot may render the nominal approach inadequate. Thus in the second part of the thesis we take into account four non-nominal scenarios where: (i) the specified task is not feasible; (ii) the task contains hard and soft constraints; (iii) the workspace model is not fully-known in priori; (iv) the task involves not only robot motion but also actions. The proposed results greatly improve the real-time adaptability and reconfigurability of the nominal scheme.

In the last part, we analyze a team of interconnected autonomous robots with local and independently-assigned tasks. Firstly we consider the case where cooperations among the robots are imposed due to heterogeneity and collaborative tasks. A decentralized coordination scheme is proposed such that the robots' joined plans satisfy their mutual tasks the most. Then a distributed knowledge transfer and update procedure is designed for the networked robots that co-exist within a common but partially-known workspace. It guarantees both the safety and correctness of their individual plans.

Acknowledgments

I would like first of all to express my gratitude to my main advisor Prof. Dimos V. Dimarogonas for his invaluable support, guidance and encouragement, in both life and work; my co-advisor Prof. Karl H. Johansson for his great insight and knowledge.

During the past two years, Automatic Control Lab has been a joyful place to stay. I thank Martin A., Håkan, and Martin J. for fun running and biking routes; Olle for numerous Pentago matches; António for inspiring perspectives on world order; Yuzhe for endless brain teaser puzzles; Burak for movie and novel suggestions; Jana for fruitful discussions; Arda for helpful Ubuntu tips; Hamid and Burak for enormous help with my teaching; Haukur for leading me into the world of Python; and my current officemates Yuzhe, Davide, Farhad, Euhanna, Christian, Hamid, José and Håkan.

Special thanks to collaborators in EU RECONFIG project: Alejandro and Michele for great learning experiences on ROS, computer vision and navigation control and for being such fun companions. Many thanks to Matteo and other researchers at Smart Mobility Lab, for their help and support during the experiment last December.

Thanks also go to the administrators at Automatic Control Lab: Hanna, Kristina, Anneli and Karin for always being helpful and creating a pleasant working atmosphere.

Finally, I would like to thank my girlfriend Wei Li and my family for always believing in me and moral support.

Thank you!
Meng Guo
Stockholm, March 2014

Contents

Chapter 1

Introduction

THE unprecedented development of digital processing units has boosted the manufacture and installation of industrial and especially domestic robots. They become more powerful in terms of computing speed and capacity, and at the same time more affordable. They are expected to accomplish various tasks specified by non-expert end-users autonomously, without or with minimal human intervention.

Additionally, wireless communication technology enables almost all robots to be connected and possibly also with internal or external “smart” sensors, meaning that they can have more accurate and up-to-date information about their operation space. This type of communication should be modeled and encoded in a formal and correct way to save bandwidth and improve efficiency. All these issues bring the need for a new framework for modeling, design and analysis of interconnected multi-robot systems.

1.1 Motivating Examples

To demonstrate the motivation for the methods developed in the thesis,two potential applications are presented: rescue robots and domestic robots.

Rescue Robots

Rescue robots are designed for rescuing people and exploring dangerous and hazardous sites during disasters (as shown in Fig. 1.1). First of all, a formal and concise way to specify the intended task is a key aspect since human languages are hard for machines to understand and full of

Figure 1.1: Illustration of a team of rescue robots. (Courtesy of Wikimedia)

ambiguities (Resnik, 2011). These tasks normally consist of a sequence of robot motion and actions. For instance, one task of the DARPA 2014 competition (DARPA, 2012) is to "open a door, enter a building, climb an industrial ladder, traverse an industrial walkway, and turn on a valve".

Given the task, the robot should be able to synthesize a plan and control strategy that fulfills the task, in an automated way without human intervention. This is critical for practical applications where many rescue robots operate in a degraded environment and communications between the robots and the human operators are un-reliable.

Though a blueprint of the operation site normally exists, the actual workspace might be significantly different after the disaster and full of uncertainties. Thus it is essential for the robot to sense the actual physical workspace and to adapt its own plan and control strategy accordingly.

Due to different sizes, loads, power capacities and functionalities, the robots within the team might be assigned different roles and tasks. However it is very likely that one robot needs the collaboration from others to accomplish part of its own task. A efficient scheme to coordinate collaborations within the team is of great importance.

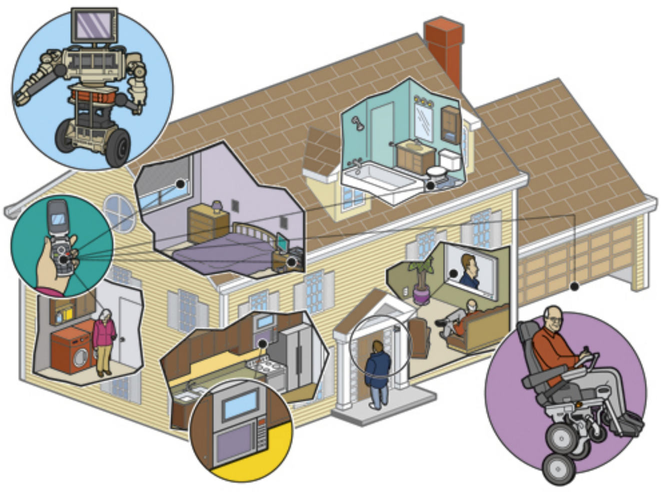


Figure 1.2: Illustration of a future view on domestic robots and smart house. (Courtesy of Creattica (2014))

Domestic Robots

Domestic robots or service robots are used for household duties, like cleaning (Roomba, 2013), surveillance (Gostai, 2013), tele-presence, assistance (Rethink, 2013) and fun (Romo, 2013). It is predicted that about 15.6 million service robots for personal and domestic usage will be sold between 2012 and 2015 (Robotics, 2012). They should be able to accomplish complex tasks including navigation, manipulation, recognition, interaction with humans and so on, as shown in Fig. 1.2. The future perspective on domestic robots is that they will become a nearly ubiquitous part of everyone's day-to-day life.

The key functionality of domestic robots is that they receive daily tasks specified by non-expert end-users and need to synthesize the plan and control strategy by themselves to fulfill the task. Most importantly, they should execute the plan autonomously and adapt to the varying environment within different households. This places numerous challenges on integrating the high-level task planning with the low-level motion and action control.

Furthermore, through Internet or Intranet, the service robots belong to the whole domestic network with other smart sensors. It relieves the robots from being equipped with heavy on-board sensors, under the condition that an efficient information exchange protocol can be designed. As a result, a formal and correct framework that provides all these features will popularize the deployment of domestic robots even further.

1.2 Background

In this section, we introduce some background knowledge related to automated planning, model-checking algorithms and multi-agent systems.

Motion and Task Planning

Motion planning normally refers to the planning problem that involves systems with continuous dynamics given as ordinary differential/difference equations. The planning goal is to find the actuation signal that drives the system from an initial state to a goal state. It is also called planning in continuous statespace by LaValle (2006) or a control problem in Control Theory (Doyle et al., 1992). Despite of the fact that motion planning tasks are normally easy to specify, they are not trivial to solve due to different geometric and dynamic constraints. However there are many well-established methods for navigating an autonomous robot from an initial position to a goal position, while staying within the allowed area, e.g., navigation function (NF) for sphere workspace and obstacles (Koditschek and Rimon, 1990), potential field (PF)-based control algorithm for workspace with triangular partitions (Lindemann et al., 2006), sampling-based motion planning techniques like probabilistic roadmap method (Kavraki et al., 1996), and rapidly-exploring random trees (LaValle, 1998; Karaman and Frazzoli, 2011).

On the other hand, in Artificial Intelligence (Ghallab et al., 2004), the task planning problem is to find a sequence of actions that change the system from an initial state to a goal state. Given a finite set of actions, each action is described by (1) the precondition that has to be fulfilled before the action can be performed; (2) the effect on the system state after performing the action. The system under consideration is modeled as discrete state-transition systems (Dean and Wellman, 1991). Different ways to represent the states may result in various complexities when solving the planning problem. Logic-based representation is currently one of the most popular formalisms used by many planning tools, like STRIPS (Fikes and Nilsson, 1972) and PDDL (McDermott et al., 1998). The solving process is similar to human deliberation that chooses and organizes actions by anticipating their outcomes.

The greatest distinction between motion and task planning is that the statespace in motion planning is continuous and possibly unbounded, making it impossible to enumerate all the states explicitly as in task planning. Furthermore, the set of allowed actions is also infinite given the continuous input space. The notion of action condition and effect is also not well defined since from different initial states the dynamical system may evolve differently under the same actuation signal.

Model Checking for Synthesis

Model checking, also called property checking is a model-based verification technique that exhaustively and automatically checks whether a given model satisfies a given specification (Clarke et al., 1999; Baier et al., 2008; McMillan, 1993). The model can be the actual hardware or software system or its abstraction as a finite transition system. The property specification normally involves security requirements such as absence of bad states and deadlocks. The automaton-based 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 language provides a concise and formal way to specify both propositional and temporal requirements on the system behavior.

Model-checking algorithms have also attracted much interest of applying them for the purpose of synthesis rather than verification. In particular, there has been many recent work that integrates motion planning algorithms with model-checking techniques to treat complex motion tasks specified by temporal logics (Belta et al., 2007; Fainekos et al., 2009; Bhatia et al., 2010). Temporal logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) provide a formal and high-level way of describing motion objectives that are much more complex than the well-studied point-to-point navigation problems. For instance, the task might involve the coverage of several regions, sequential visits, response or surveillance.

When applying model-checking algorithms for synthesis rather than verification, the desired outcome is distinctively different: (1) the purpose of verification is to find any system behavior that violates the property; (2) for synthesis we are interested in the system behavior satisfying the property, and more importantly that is optimized regarding certain cost functions.

Distributed and Networked System

Seldom one autonomous agent is a stand-alone system, but often coexists and interacts with other agents. Networked or multi-agent systems are often deployed for the distributed problem solving (Durfee, 2006), where a global task is decomposed into smaller subtasks and then assigned to each agent. This formulation favors a tightly-coupled and top-down approach, where each agent receives commands from the central planner and executes them in a synchronized manner (Kok and Vlassis, 2006).

There is another type of multi-agent system assuming that each agent is assigned a local task independently and no global tasks. As a result, each agent acts according to its own plan in oder to accomplish the task, and meanwhile it interacts with other agents when necessary, in terms of information exchange and collaborations. This formalism favors loosely-coupled and bottom-up approaches, which resembles many practical systems like the community-based map building and navigation application (Hardawar, 2012) and collaborative consumption (Botsman and Rogers, 2010).

Control or coordination algorithms for a multi-agent system are normally evaluated regarding how decentralized they are (Ren et al., 2005). A centralized approach requires a central unit monitoring the whole team and sending out control commands to every agent. On the contrary, a decentralized approach normally has flexible membership such that agents can join and leave the team freely; requires only local interaction among agents that are nearby; respects privacy that an agent need not reveal all its local information to others.

1.3 Related Work

In this section, we review some existing literature related to this thesis.

Model-checking-based Control Synthesis

The model-checking-based controller synthesis has been applied to both complex dynamical systems and autonomous robots.

Tabuada and Pappas (2006) considers the control strategy synthesis for discrete-time linear systems where the control objective is given by linear temporal logic formulas, beyond the usual stabilization and output regulation objectives. Aydin Gol and Lazar (2013) proposes an optimal control strategy for discrete-time systems under the temporal logic constraints and a quadratic cost function. Yordanov and Belta (2010) and Yordanov et al. (2012) present a computational framework for the a feedback control strategy synthesis of discrete-time piecewise affine systems, where the specification is given as linear temporal logic formulas over linear predicates. Fainekos and Pappas (2009) provides a robustness index for the satisfiability of continuous-time signals under temporal logic specifications. Abbas et al. (2013) synthesizes a generic cyber-physical system with respect to a metric temporal property.

On the other hand, Fainekos et al. (2009) firstly proposes the complete framework of automated controller synthesis for autonomous robots under temporal logic 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 modified model-checking algorithms, which is then implemented by low-level continuous controller. Graphical tools (Srinivas et al., 2013) are also developed for non-expert end-users to express the intended task specifications freely.

Partially-known Workspace

A critical assumption for the formalism above is that the workspace is perfectly known and is correctly represented in the finite transition system. The discrete plan is normally generated off-line and the robot executes the plan no matter what has changed in the workspace, i.e., it does not react to actual observations, as also mentioned in Ding et al. (2011b). Consequently, this framework is lack of reconfigurability and real-time adaptation. Some existing work considers the case when a complete representation of the workspace is not available. In Ding et al. (2011b) and Wolff et al. (2012), the robot's motion and the uncertain workspace are modeled as nondeterministic Markov decision processes, where the goal is to find a control strategy that maximizes the probability of satisfying the specification. In Kress-Gazit et al. (2009) and Wongpiromsarn et al. (2010), a two-player GR(1) game between the robot and the environment is constructed and a receding horizon planning algorithm is introduced. A winning strategy for the robot can be synthesized by exhaustively searching through all possible combinations of the robot movements and the admissible workspaces.

Instead of aiming for an off-line motion plan that takes into account every possible situation, we propose to create a preliminary plan based on the initially available knowledge about the robot and the workspace. Then while implementing the preliminary plan, real-time observation about the workspace and the plan execution status are gathered, based on which the plan is verified and revised (Guo et al., 2013b). Similar ideas appear in Livingston et al. (2012) by locally patching the invalid transitions, which however can not handle unexpected changes in regions' properties.

Infeasible Task

The initial knowledge of the workspace might be partially or even incorrect, which may render the intended task infeasible. As stressed in Fainekos (2011), Kim and Fainekos (2012) and Raman and Kress-Gazit (2011), the above motion planning framework 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.

Fainekos (2011) and Kim and Fainekos (2012) address this problem in a systematic way to find the relaxed specification automaton that is closest to the original one and at the same time can be fulfilled by the system. Raman and Kress-Gazit (2011) 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 (Guo and Dimarogonas, 2013) and the references above is that we put emphasize on how to synthesize the motion plan that fulfills the infeasible task the most, instead of finding and analyzing the infeasible parts of the task. Detailed comparisons can be found at the beginning of Section 3.1.

Tumova et al. (2013b) and Tumova et al. (2013a) take into account the problem of synthesizing a least-violating control strategy under a set of safety rules. However the level of satisfiability is measured differently from our approach. 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 the specifications.

Motion and Action Planning

To solve problems of practical interest it is often necessary to perform various actions at different regions. Thus in Guo et al. (2013a), we propose a generic framework that combines model-checking-based robot motion planning with action planning using action description languages.

Some relevant work can be found that integrates motion planning and action planning. In Smith et al. (2011), since the underlying actions can only be performed at fixed regions, the specification is reinterpreted in terms of regions to visit. In Kress-Gazit et al. (2009), since the actions can be activated and de-activated at any time, independent propositions are created for each action. The above approaches are not be applicable if some actions can only be performed when the workspace and the robot itself satisfy certain conditions, or choices have to be made among all allowed actions.

Multi-agent System with Temporal Tasks

The same model-checking-based control synthesis framework has also been extended to a multi-agent system consisting of autonomous agents. Many existing work can be found on decomposing a global task specification to bi-similar local ones in a top-down manner, which are then executed by the agents in a synchronized (Kloetzer and Belta, 2010) and partially-synchronized (Ding et al., 2011a) manner. Ulusoy et al. (2012) presents an optimal path planning framework for a team of mobile robots under a global task specification and traveling time uncertainties. Karaman and Frazzoli (2011) formulates the multi-UAV routing problem under linear temporal tasks as a mixed integer linear programming problem and the derived paths for all vehicles need to be executed in a synchronized way. There are several drawbacks of this top-down approach: it is centralized as the plans for the whole team are synthesized once by the central unit; it has a high computational complexity subjected to the combinatorial blowup; it is vulnerable to agent failures and contingencies in the workspace.

Thus we, from an opposite viewpoint, assume that the local task specifications are assigned locally to each agent and there is no specified global task. Namely we consider a team of cooperative agents with different, independently-assigned, even conflicting individual tasks (Guo and Dimarogonas, 2013, 2014a). This loosely-coupled and bottom-up approach allows more flexibility of the system, where plans and decisions are made locally by each agent; conflicts and collaborations are resolved during execution time. Filippidis et al. (2012) adopts a similar formalism where a framework for decentralized verification is proposed. However the way to resolve potentially-conflicting tasks is not considered there.

1.4 Main Contributions

In the first contribution we propose a novel framework for real-time motion planning based on model checking and revision. We firstly modify the existing nested-DFS algorithm to search for an accepting path of a directed graph, which gives a preliminary motion plan. Then we classify three types of real-time information that might be obtained during real-time execution, and show how they can be used to update the system model. We provide a criterion to verify whether the current motion plan remains valid for the updated system. If not, an iterative revision algorithm is designed to revise the plan locally such that it becomes valid and fulfills the task specification. This framework is particularly useful for operation in partially-known workspaces and workspaces with uncertainties. The above results have been published in the following proceeding:

- 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, May 2013.

The second contribution is a generic approach to derive the complete description of the robot's functionalities within a certain workspace, such that any LTL task specification in terms of desired motions and actions can be treated. We propose an approach to combine model-checking-based motion planning with action planning using action description languages, in order to tackle tasks involving not only regions to visit but also desired actions at these regions. The above results have been published in the following proceeding:

- 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, November 2013.

In the third contribution we analyze single- and multi-agent systems under local LTL tasks that are infeasible and describe how to synthesize the motion plan that fulfills the infeasible task the most and how the infeasible task could be relaxed. We allow the user-defined relative weighting between the implementation cost of the motion plan and how much this plan fulfills the original task specification. Multi-agent systems are also exploited and a decentralized approach is proposed by considering the dependency and priority relations. The above results have been published in the following proceeding:

- 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, December 2013.

In the fourth contribution, we propose a cooperative plan reconfiguration scheme for networked autonomous agents under partially-known workspace. Each agent has a locally-assigned and possibly infeasible task specification as LTL formulas, containing hard sub-specifications for safety and soft ones for performance. A real-time knowledge transfer and update scheme is designed, which guarantees not only safety and correctness of individual plans but also fast convergence to the optimal plan. The above results have been published in the following proceedings:

- 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, May 2014. To appear.

- M. Guo and D. V. Dimarogonas. Multi-agent Plan Reconfiguration under Local LTL Specifications. *International Journal of Robotics Research*. Submitted.

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*, Sep 2013.

- M. Guo and D. V. Dimarogonas. Consensus with Quantized Relative State Measurements. *Automatica*, 49(8): 2531–2537, Aug 2013.

- M. Guo and D. V. Dimarogonas. Nonlinear Consensus via Continuous, Sampled, and Aperiodic Updates. *International Journal of Control*, 86(4): 567-578, Jan 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, June 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, FL, USA, December 2011.

1.5 Thesis Outline

The structure of this thesis is as follows. In Chapter 2, we introduce the theoretical tools and preliminary knowledge needed to formulate the nominal motion and task planning problem. Then the generic framework for solving this problem is proposed. Chapter 3 contains four extensions to the nominal scenario, where reconfigurability and real-time adaptation is the main focus. Chapter 4 addresses the cooperative planning and reconfiguration problem for networked autonomous agents with both independent and dependent tasks. At last, we present the final conclusions and some future research directions in Chapter 5.

Chapter 2

Motion and Task Planning

THIS chapter introduces the key ingredients in the model-checking-based motion and task planning framework. A finite-state transition system serves as the discrete abstraction of the continuous robot motion within a workspace, while a linear-temporal-logic formula specifies the task requirement over the transition system. The aim is to firstly find a discrete motion and task plan, which satisfies the task and at the same time minimizes a cost function. Then the discrete plan is implemented by a hybrid control strategy that navigates the robot within the real workspace.

2.1 Finite-state Transition System

The robot's continuous motion within a certain workspace is abstracted as a finite-transition system (Baier et al., 2008). This finite-state transition system is constructed by integrating two aspects: (i) the workspace model; (ii) the robot's dynamics and its navigation technique.

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 $\pi_i \subset \mathcal{W}_0$, $\forall i = 1, \cdots, W$. 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 $\pi_i \cap \pi_j = \emptyset$, $\forall i, j = 1, \ldots, W$ and $i \neq j$.

Atomic propositions are boolean variables that can be either `True` or `False`. They are used here to express known propterties 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

$$a_{r,i} = \begin{cases} \texttt{True} & \text{if the robot is in region } \pi_i, \\ \texttt{False} & \text{otherwise,} \end{cases} \tag{2.1}$$

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 $AP_p = \{a_{p,1}, \cdots, a_{p,M}\}$ 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 2.1. *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. Moreover, $a_{r,i} \in L(\pi_i)$ 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 2.1. An office-like workspace has six rooms and one corridor, which gives the partition in Fig. 2.1. 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 (Lee and Schachter, 1980) and Voronoi diagram (LaValle, 2006). Note that this level of partition is preliminary and not robot-specific.

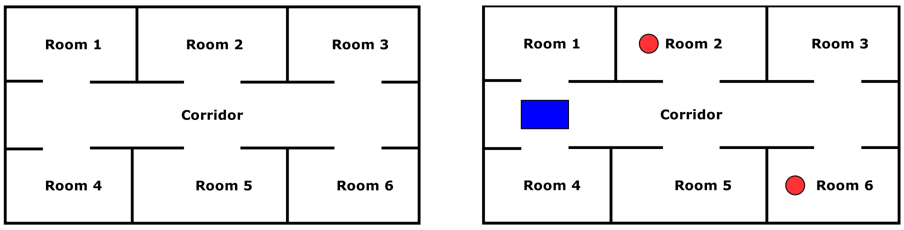


Figure 2.1: Left: the office-like partition of six rooms and one corridor. Right: after adding in AP_p, with two balls (in red) and one basket (in blue).

Robot Dynamics and Navigation Technique

We assume the robot satisfies the following continuous dynamics:

$$\dot{x} = f(x(t), u(t)), \tag{2.2}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ are the position and control signal; $f(\cdot)$ is Lipschitz continuous (Khalil, 2002). 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 autonomous robots with various dynamics.

Given the preliminary partition from Section 2.1 and the agent dynamics by (2.2), 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 2.2. *There is a transition from π_i to π_j if an admissible navigation controller $\mathcal{U} : \mathbb{R}^n \times \Pi \times \Pi \to \mathbb{R}^m$:*

$$u(t) = \mathcal{U}(x(t), \pi_i, \pi_j), \qquad t \in [t', t''] \tag{2.3}$$

exists that could drive the system (2.2) *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 2.5. Two main navigation techniques are discussed in the thesis: (i) Lindemann et al. (2006) 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) Koditschek and Rimon (1990) 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 (Loizou and Jadbabaie, 2008) and multi-agent (Dimarogonas and Kyriakopoulos, 2007) navigation control under different geometric constraints, like single-integrator (Loizou and Kyriakopoulos, 2002), double-integrator (Dimarogonas and Kyriakopoulos, 2005) 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 2.2 shows some cases where the preliminary partition is modified after incorporating the robot dynamics and navigation techniques. Note that this might lead to an over-approximation or under-approximation (Saïdi and Shankar, 1999) of the actual workspace as some regions are shrunk or expanded during the process. In some cases, another navigation technique needs to be considered when it is infeasible to incorporate one navigation technique to a given workspace model.

Example 2.2. As shown in Fig. 2.2, the office-like workspace is further partitioned, since the underlying navigation technique relies on triangular partition. The irregular partitions are approximated by circular areas due to the navigation-function construction. ▲

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 the state $\pi_i = \{\texttt{the robot is at region } \pi_i\}$, $i = 1, \cdots, N$. The states 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

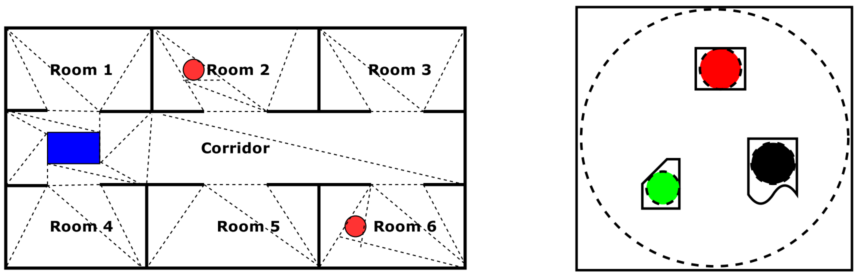


Figure 2.2: Further partitions resulting from incorporating different navigation techniques.

that the transition relation is not necessarily the adjacency relation in the geometrical sense, but by Definition 2.2. Formally the control-driven and weighted finite-state transition system (FTS) is defined below:

Definition 2.3. *The control-driven weighted FTS is a tuple* $\mathcal{T}_c = (\Pi, \longrightarrow_c, \Pi_0, AP, L_c, W_c)$*, where*

- $\Pi = \{\pi_i,\, i = 1, \ldots, W\}$ *is the set of states;*
- $\longrightarrow_c \subseteq \Pi \times \Pi$ *is the transition relation by Definition 2.2. 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 2.1*
- $W_c : \Pi \times \Pi \to \mathbb{R}^+$ *is the weight function, representing 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 (Baier et al., 2008).

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.

2.2 Linear Temporal Logic

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.

Syntax and Semantics

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

$$\varphi ::= \texttt{True} \mid a \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2, \tag{2.4}$$

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 Baier et al. (2008). An infinite *word* over the alphabet 2^{AP} is an infinite sequence $\sigma \in (2^{AP})^{\omega}$, with the following structure $\sigma = S_0S_1S_2\cdots$, 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 formula for an infinite word σ is given via a doubly-recursive definition of the relation $(\sigma,\, k) \models \varphi$, meaning that the word σ *satisfies* φ at time step k.

Definition 2.4. *The semantics of LTL is defined as follows:*

$$\begin{aligned}
(\sigma,\, k) \models a \quad &\leftrightarrow \quad a \in S_k\\
(\sigma,\, k) \models \neg\,\varphi \quad &\leftrightarrow \quad (\sigma,\, k) \not\models \varphi\\
(\sigma,\, k) \models \bigcirc \varphi \quad &\leftrightarrow \quad (\sigma,\, k+1) \models \varphi\\
(\sigma,\, k) \models \varphi_1 \vee \varphi_2 \quad &\leftrightarrow \quad (\sigma,\, k) \models \varphi_1 \ \textit{or}\ (\sigma,\, k) \models \varphi_2\\
(\sigma,\, k) \models \varphi_1 \,\mathsf{U}\, \varphi_2 \quad &\leftrightarrow \quad \exists k' \in [k,\, +\infty],\, (\sigma,\, k') \models \varphi_2 \ \textit{and}\\
&\qquad \forall k'' \in (k,\, k'),\, (\sigma,\, k'') \models \varphi_1\,.
\end{aligned}$$

▲

Any LTL formula φ is satisfied by σ at time step 0 if $(\sigma, 0) \models \varphi$ (for simplicity we denote by $\sigma \models \varphi$). The *words* of φ is defined as the set of words that satisfy φ at time step 0, i.e., $\texttt{Words}(\varphi) = \{\sigma \in (2^{AP})^\omega \,|\, \sigma \models \varphi\}$.

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).

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 (2.4).

Definition 2.5. *An infinite path τ **satisfies** φ, i.e., $\tau \models \varphi$ if its trace $\texttt{trace}(\tau) \models \varphi$. A satisfying path is also called a **plan** for φ.*

2.3 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, it would be of interest 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:

$$\tau = \langle \tau_{\text{pre}}, \tau_{\text{suf}} \rangle = \tau_{\text{pre}}\, [\tau_{\text{suf}}]^\omega \tag{2.5}$$

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 (2.5). This structure is also called *lasso-shapeed* in Schuppan and Biere (2005), namely, τ has the stem τ_{pre} and the loop τ_{suf}.

Definition 2.6. *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 2.1. *Given the control-driven wFTS $\mathcal{T}_c$ and an LTL formula φ over AP, (i) find a plan with the prefix-suffix structure by* (2.5)*; (ii) construct the hybrid control strategy based on* (2.3) *to execute the derived plan.*

2.4 Discrete Plan Synthesis

In this section, we describe in detail how to synthesize the discrete plan that solves the first part of Problem 2.1.

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

$$\mathcal{A}_\varphi = (Q,\, 2^{AP},\, \delta,\, Q_0,\, \mathcal{F}), \tag{2.6}$$

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, $r = q_0q_1q_2\cdots$, 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 $\texttt{Inf}(r) \cap \mathcal{F} \neq \emptyset$, where $\texttt{Inf}(r)$ is the set of states that appear in r infinitely often. Similar as before, we denote the successors of $q_m \in Q$ by $\texttt{Post}(q_m) = \{q_n \,|\, \exists S \in 2^{AP},\ q_n \in \delta(q_m,\, S)\}$.

Definition 2.8. *Given an infinite word $\sigma = S_0S_1S_2\cdots$ over 2^{AP}, its **resulting run** in $\mathcal{A}_\varphi$ is denoted by $r_\sigma = q_0q_1q_2\cdots$, which satisfies: (i) $q_0 \in Q_0$; (ii) $q_{i+1} \in \delta(q_i,\, S_i)$, $\forall i = 1, 2, \cdots, \infty$.*

*Similarly, given a finite word $\bar{\sigma} = S_0S_1S_2\cdots S_{N+1}$ over 2^{AP}, its **resulting run** is denoted by $r_{\bar{\sigma}} = q_0q_1q_2\cdots q_N$, which satisfies: (i) $q_0 \in Q_0$; (ii) $q_{i+1} \in \delta(q_i,\, S_i)$, $\forall i = 1, 2, \cdots, N$.* ▲

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., $\mathcal{L}_w(\mathcal{A}_\varphi) = \{\sigma \in (2^{AP})^\omega \,|\, r_\sigma \text{ is an accepting run}\}$.

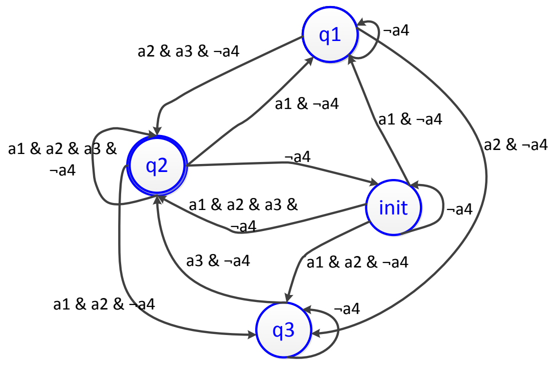


Figure 2.3: The NBA corresponds to $\varphi = (\Box\Diamond a_1) \wedge (\Box\Diamond a_2) \wedge (\Box\Diamond a_3) \wedge (\Box\neg a_4)$ (from Gastin and Oddoux (2001)). The transition from state q_1 to q_3 is given by $q_3 \in \delta(q_1, l)$ where the boolean expression $l = (a_2 \,\&\, \neg a_4)$ encodes four input alphabets $\{a_2\}$, $\{a_2,\, a_1\}$, $\{a_2,\, a_3\}$, $\{a_2,\, a_1,\, a_3\}$.

Lemma 2.1. $\mathcal{L}_w(\mathcal{A}_\varphi) = \texttt{Words}(\varphi)$

Proof. See proof of Theorem 5.37 in Baier et al. (2008) ∎

The translation process from an LTL formula to its corresponding NBA can be done in time and space $2^{\mathcal{O}(|\varphi|)}$ (Baier et al., 2008). However, there are fast translation algorithms (Gastin and Oddoux, 2001), 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 Gastin and Oddoux (2001) generates a boolean expression for each transition, which accepts all alphabets that enable this transition (as shown in Fig. 2.3). Binary decision diagrams (BDD) are well-known for their efficiency to represent and evaluate boolean functions (Akers, 1978) . As a result, an NBA can be encoded symbolically and efficiently. More details can be found in Section 4.3.

Product Büchi Automaton

The automaton-based model-checking algorithm can be found in Vardi and Wolper (1986) and Algorithm 11 of Baier et al. (2008). 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.

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 2.9. *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.*
$$W_p(\langle \pi_i, q_m\rangle, \langle \pi_j, q_n\rangle) = W_c(\pi_i, \pi_j), \tag{2.7}$$
where $\langle \pi_j, q_n\rangle \in \delta'(\langle \pi_i, q_m\rangle)$. ▲

Since $\mathcal{A}_p$ remains a Büchi automaton (Baier et al., 2008), 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 2.2. *If there exists an infinite path τ of $\mathcal{T}_c$ such that $\tau \models \varphi$, then* ***at least*** *an accepting run of $\mathcal{A}_p$ exists.*

Proof. See the proof of Theorem 4.63 from Baier et al. (2008). ■

Lemma 2.3. *Given an accepting run R of $\mathcal{A}_p$, then $R|_\Pi \models \varphi$.*

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 $R|_\Pi \models \varphi$ by Definition 2.5. It completes the proof. ∎

Cost of Accepting Run

We intended to find an infinite path τ of $\mathcal{T}_c$ with the prefix-suffix structure by (2.5), such that $\tau \models \varphi$.

Lemma 2.4. *Given that there exists an infinite path τ with the prefix-suffix structure by* (2.5) *and $\tau \models \varphi$, then* ***at least*** *one accepting run of $\mathcal{A}_p$ exists with the prefix-suffix structure.*

Proof. Since $\tau \models \varphi$, then $\sigma = \texttt{trace}(\tau) \in \texttt{Words}(\varphi)$. Furthermore as $\mathcal{L}_w(\mathcal{A}_\varphi) = \texttt{Words}(\varphi)$ by Lemma 2.1, $\sigma \in \mathcal{L}_w(\mathcal{A}_\varphi)$, meaning that the resulting run r_σ in $\mathcal{A}_\varphi$ by Definition 2.8 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 following *prefix-suffix* structure:

$$\begin{aligned} R = \langle R_{\text{pre}},\, R_{\text{suf}}\rangle &= q'_0\, q'_1 \cdots q'_f \, \Big[\, q'_f\, q'_{f+1} \cdots\cdots q'_n \,\Big]^\omega \\ &= \langle \pi_0, q_0\rangle \cdots \langle \pi_{f-1}, q_{f-1}\rangle \, \big[\, \langle \pi_f, q_f\rangle \langle \pi_{f+1}, q_{f+1}\rangle \cdots\cdots \langle \pi_n, q_n\rangle \,\big]^\omega \,, \end{aligned} \tag{2.8}$$

Algorithm 1: Construct full product automaton, `FullProd( )`

```
   Input: $\mathcal{T}_c$, $\mathcal{A}_\varphi$
   Output: $\mathcal{A}_p$
1  foreach $\pi_i \in \Pi$, $q_m \in Q$ do
2      $q'_s = \langle \pi_i,\, q_m \rangle \in Q'$
3      if $q_m \in Q_0$ and $\pi_i \in \Pi_0$ then
4          $q'_s \in Q'_0$
5      if $q_m \in \mathcal{F}$ then
6          $q'_s \in \mathcal{F}'$
7      foreach $\pi_j \in \texttt{Post}(\pi_i)$, $q_n \in \texttt{Post}(q_m)$ do
8          $q'_g = \langle \pi_j,\, q_n \rangle \in Q'$
9          $d = \texttt{CheckTranB}(q_m,\, L_c(\pi_i),\, q_n,\, \mathcal{A}_\varphi)$
10         if $d \geq 0$ then
11             $q'_g \in \delta'(q'_s)$
12             $W_p(q'_s,\, q'_g) = W_c(\pi_i,\, \pi_j) + \alpha \cdot d$
13 return $\mathcal{A}_p$
```

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 (2.8), there is a finite set of transitions appearing in R:

$$\texttt{Edge}(R) = \{(q'_i,\, q'_{i+1}),\ i = 0, 1 \cdots, (n-1)\} \cup \{(q'_n,\, q'_f)\}. \tag{2.9}$$

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

$$\begin{aligned} \texttt{Cost}(R,\, \mathcal{A}_p) &= \sum_{i=0}^{f-1} W_p(q'_i,\, q'_{i+1}) + \gamma \sum_{i=f}^{n-1} W_p(q'_i,\, q'_{i+1}) \\ &= \sum_{i=0}^{f-1} W_c(\pi_i,\, \pi_{i+1}) + \gamma \sum_{i=f}^{n-1} W_c(\pi_i,\, \pi_{i+1}) \end{aligned} \tag{2.10}$$

where the first summation in (2.10) represents the accumulated weights of transitions along the prefix and the second is the summation along the suffix.

Algorithm 2: Validate transitions of $\mathcal{A}_\varphi$, `CheckTranB( )`

```
  Input: $(q_m,\, l,\, q_n,\, \mathcal{A}_\varphi)$, $q_m,\, q_n \in Q$, $l \in 2^{AP}$
  Output: distance $d$
1 $d = -1$
2 if $q_n \in \delta(q_m,\, l)$ then
3     $d = 0$
4 return $d$
```

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

Problem 2.2. *Given the product automaton $\mathcal{A}_p$, find its accepting run with the prefix-suffix structure that **minimizes** the cost defined by* (2.10)*.*

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 2.1. τ_{opt} might not be the actual optimal plan with the prefix-suffix structure, whose cost is defined similarly as (2.10). As pointed out in Schuppan and Biere (2005), 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 Clarke et al. (1994) 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 Schuppan and Biere (2005). ▲

Optimal Run Search Algorithms

In this part we present two methods to construct the product automaton and 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 2.9,

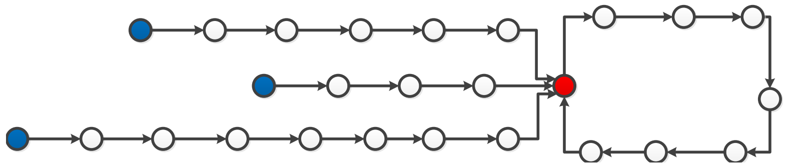


Figure 2.4: For every pair of initial state $q'_0 \in Q'_0$ (in blue) and accepting state $q'_f \in \mathcal{F}'$ (in red), the shortest path from q'_0 to q'_f and the shortest cycle containing q'_f are computed.

which is presented in Algorithm 1. 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 $q_n \in \delta(q_m, L(\pi_i))$. This condition is evaluated by calling Algorithm 2 in Line 9: $d = -1$ if $q_n \notin \delta(q_m, L_c(\pi_i))$; $d = 0$ if $q_n \in \delta(q_m, L_c(\pi_i))$. The weight of this transition is given by (2.7) in Line 12, where α is set to 1 for now.

After constructing $\mathcal{A}_p$ fully, Algorithm 3 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 utilizes Dijkstra's algorithm (LaValle, 2006) for computing the shortest path from a single source node to a set of target nodes within a weighted graph. In particular, function $\texttt{DijksTargets}(\mathcal{A}_p, q'_S, Q'_T)$ 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 $\texttt{DijksCycle}(\mathcal{A}_p, q'_S)$ is used to compute shortest cycle from the source state q'_S back to itself. As shown in Figure 2.4, 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 where $\texttt{DijksTargets}(\cdot)$ is called while the shortest cycle containing q'_f is obtained from line 2 of Algorithm 3 where $\texttt{DijksCycle}(\cdot)$ is called. At last, the pair $(q'_{0,\star}, q'_{f,\star})$ that minimizes the total cost defined by (2.10) 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 $q'_{f,\star}$.

Algorithm 1 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 is given by $\mathcal{O}(|\mathcal{A}_p| \cdot \log |\mathcal{A}_p| \cdot (|Q'_0| + |\mathcal{F}'|))$ as essentially the Dijkstra algorithm has to be called over $\mathcal{A}_p$ by the number of times equal to $|Q'_0| + |\mathcal{F}'|$.

Algorithm 3: Search for optimal run, `OptRun( )`.

```
Input: $\mathcal{A}_p$, $S' = Q'_0$ by default
Output: $R_{\text{opt}}$
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 (2.10).
5. Optimal accepting run $R_{\text{opt}}$, prefix: the shortest path from $q'_{0,\star}$ to $q'_{f,\star}$; suffix: the shortest cycle from $q'_{f,\star}$ and back to itself.
return $R_{\text{opt}}$
```

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 and even the revision Algorithm 14 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 $q'_s \in Q'$ and calls Algorithm 4 for the adjacency relation of q'_s, Algorithm 4 iterates 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 (Lines 2-4); if q'_s is marked "unvisited" or π_i belongs to $\widehat{\Pi}$, they need to be constructed by Definition 2.7 in Lines 5-12; $\widehat{\Pi}$ is defined in (3.20), which can be treated as $\emptyset$ for now.

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 in $\mathcal{T}_c$, as later described in Algorithm 14 in Section 3.3. Then Algorithm 3 can be easily modified such that it takes the adjacency relation of $\mathcal{A}_p$ as an input, instead of the full $\mathcal{A}_p$. Functions `DijksTargets`$(\cdot)$ and `DijksCycle`$(\cdot)$ still work in the same way as Dijkstra algorithm only needs the adjacency relation in the breadth-first structure and the associated weight. Even though the worst-case computational

Algorithm 4: Build adjacency relation of $\mathcal{A}_p$ on-the-fly, `AdjProd( )`

```
Input: $q'_s$, $\mathcal{T}_c$, $\mathcal{A}_\varphi$, $\widehat{\Pi}$, $\mathcal{A}_p$
Output: $q'_g \in \delta'(q'_s)$, $W_p(q'_s, q'_g)$
1  $q'_s = \langle \pi_i, q_m \rangle \in Q'$
2  if $q'_s$ is marked "visited" then
3      foreach $q'_g \in \delta'(q'_s)$ do
4          yield transition $q'_g \in \delta'(q'_s)$ and its weight
5  else if $q'_s$ is marked "unvisited" or $\pi_i \in \widehat{\Pi}$ then
6      empty $\delta'(q'_s)$
7      foreach $\pi_j \in$ Post($\pi_i$), $q_n \in$ Post($q_m$) do
8          $q'_g = \langle \pi_j, q_n \rangle \in Q'$
9          $d =$ CheckTranB($q_m$, $L_c(\pi_i)$, $q_n$, $\mathcal{A}_\varphi$)
10         if $d \geq 0$ then
11             $q'_g \in \delta'(q'_s)$
12             $W_p(q'_s, q'_g) = W_c(\pi_i, \pi_j) + \alpha \cdot d$
13             yield transition $q'_g \in \delta'(q'_s)$ and its weight
14     mark $q'_s$ as "visited"
```

complexity of Algorithm 3 when combined with Algorithm 4 remains the same as the full construction by Algorithm 1, the on-the-fly construction of $\mathcal{A}_p$ is essential for the partially-known and dynamic workspace presented in Section 3.3 where $\mathcal{T}_c$ has to be updated frequently.

Example 2.3. 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 (2.15). 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 Algorithm 1; (ii) t_{syn}, the time taken to find the optimal accepting run by Algorithm 3 over the fully-constructed $\mathcal{A}_p$; (iii) t_{fly}, the time taken to construct $\mathcal{A}_p$ on-the-fly by Algorithm 4 along the optimal search by Algorithm 3. They are measured by CPU time in

Table 2.1: Numerical Results for Example 2.3

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

seconds. This numerical analysis is carried out 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 2.1, 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. ▲

2.5 Hybrid Controller Synthesis

Assume the optimal run R_{opt} from Algorithm 3 has the following format:

$$\begin{aligned} R_{\text{opt}} &= \langle R_{\text{pre}},\, R_{\text{suf}} \rangle \\ &= R_{\text{pre},1} R_{\text{pre},2} \cdots R_{\text{pre},N_{\text{pre}}} \big[R_{\text{suf},1} R_{\text{suf},2} \cdots R_{\text{suf},N_{\text{suf}}} \big]^{\omega}, \end{aligned} \tag{2.11}$$

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. 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}, Algorithm 5 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 6 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

Algorithm 5: Find next goal state, `NextGoal( )`

```
    Input: $q'_{\text{cur}}$, $R_{\text{opt}}$
    Output: $q'_{\text{next}}$
1   $q'_{\text{cur}} = R_{seg,k}$
2   if $seg$ = "pre", $k < N_{\text{pre}}$ then
3       $k = k + 1$, $seg$="pre"
4   if $seg$ = "pre", $k = N_{\text{pre}}$ then
5       $k = 1$, $seg$="suf"
6   if $seg$ = "suf", $k < N_{\text{suf}}$ then
7       $k = k + 1$, $seg$="suf"
8   if $seg$ = "suf", $k = N_{\text{suf}}$ then
9       $k = 1$, $seg$="suf"
10  return $q'_{\text{next}} = R_{seg,k}$
```

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 from Algorithm 5 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 6 can be running for infinitely long time as the the suffix segment is repeated infinitely many times.

Remark 2.2. The condition in Line 3 of Algorithm 6 greatly effects the resulting behavior of the robot, regarding when it stops moving towards the current goal region and switches to the next.

Case Study

In this case study, we validate the proposed framework by both simulation and experiment results.

Workspace Abstraction

The workspace is a testbed with size $2.4m \times 2.1m$ representing the office environment shown in Fig. 2.5, consisting of three rooms on each

Algorithm 6: Off-line plan execution by hybrid control, `HybCon( )`

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

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,$\cdots$, r6, c1, c2, c3". There are one red ball, one green ball and 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): ($r1$, {r1}), ($r2$, {r2, basket}), ($r3$, {r3, gball}), ($c1$, {c1}), ($c2$, {c2}), ($c3$, {c3}), ($r4$, {r4, basket}), ($r5$, {r5, rball}), ($r6$, {r6}).

Robot Description

The robot we deployed is a NAO robot (Aldebaran, 2014) with size $0.5m \times 0.2m \times 0.3m$, 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. Although its inner kinematics and dynamics are hard to analyze, it is equipped with a development toolkit called NAOqi motion API, making it fairly straightforward to design motion controllers that drive the robot to a goal point relative to its local coordinate. In particular, the basic control modules provided are "move_x()", "move_y()" and "turn()". Namely, it can move forward in its local x-axis by the given speed, move sideways in its local

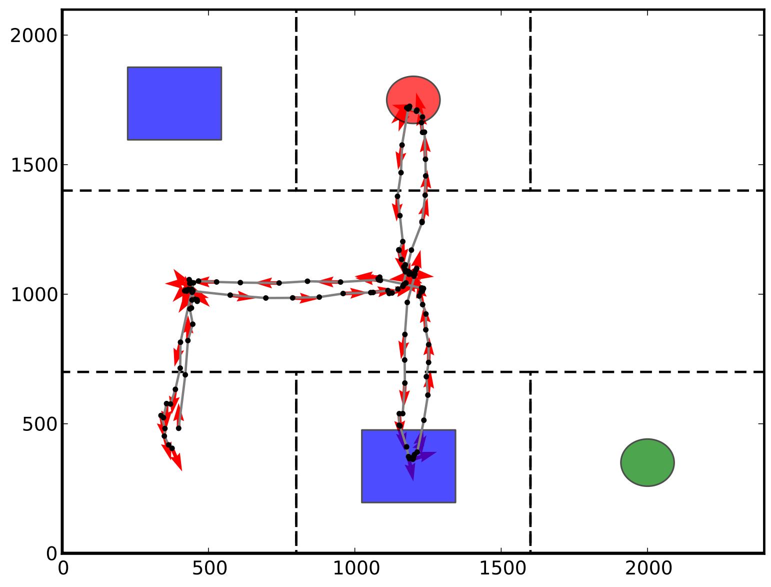


Figure 2.5: the actual trajectory that fulfills φ_1 by (2.13).

y-axis by the given speed and it can turn itself by the given angular speed. Those modules are blocking in the sense that only one module can be running every time instant and moving forward is much more precise than moving sideways. Since they are not free of actuation noises and disturbances, we design the following turn-and-forward feedback controller:

$$u = \begin{cases} \mathsf{move\_x}\big(\kappa_1 \cdot \|(x_g - x_r,\, y_g - y_r)\|\big), & \text{if } |\theta_{\text{dif}}| > \bar{\theta} \\ \begin{cases} \mathsf{turn}\big(\kappa_2 \cdot \theta_{\text{dif}}\big), & \text{if } \bar{\theta} < |\theta_{\text{dif}}| \leq \pi \\ \mathsf{turn}\big(-\kappa_2 \cdot \mathtt{sign}(\theta_{\text{dif}}) \cdot (2\,\pi - \mathsf{abs}(\theta_{\text{dif}}))\big), & \text{if } |\theta_{\text{dif}}| > \pi \end{cases} \end{cases} \tag{2.12}$$

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

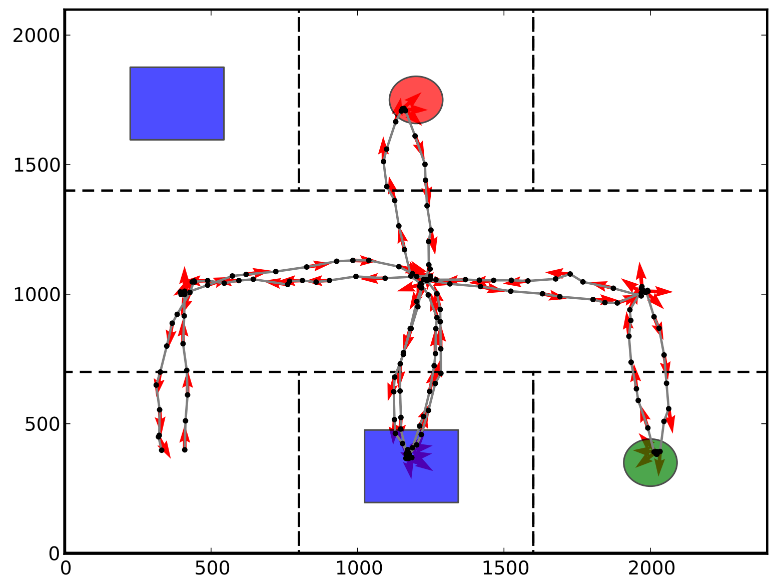


Figure 2.6: the actual trajectory that fulfills φ_3 by (2.14).

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 (2.3) can be obtained by setting the way point in (2.12) 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 (2.12).

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

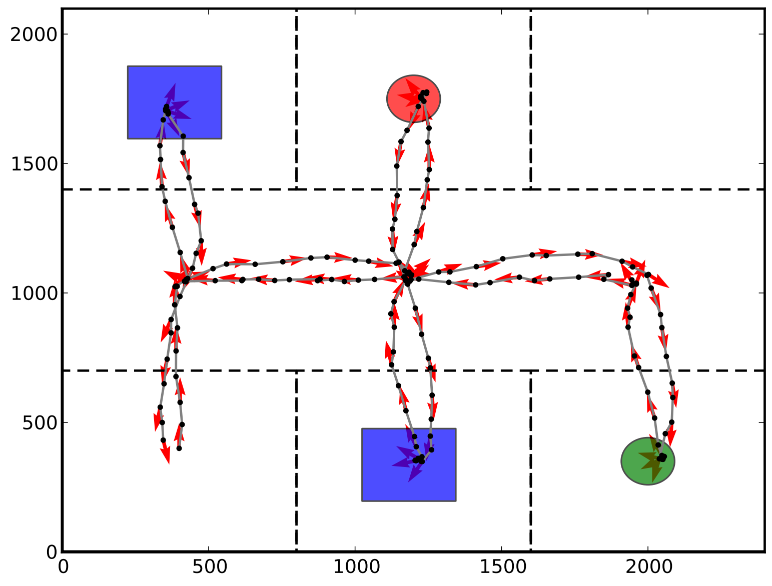


Figure 2.7: the actual trajectory that fulfills φ_4 by (2.15).

baskets and then stay at room one. It is specified as the LTL formula:

$$\varphi_1 = \Diamond(\,\mathsf{rball} \wedge \Diamond\,\mathsf{basket}) \wedge \Diamond\Box\,\mathsf{r1}, \tag{2.13}$$

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 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 6 and the final trajectory is shown in Fig. 2.5. The experiment video that demonstrates this case can be found online (Guo and Colledanchise, 2014).

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.

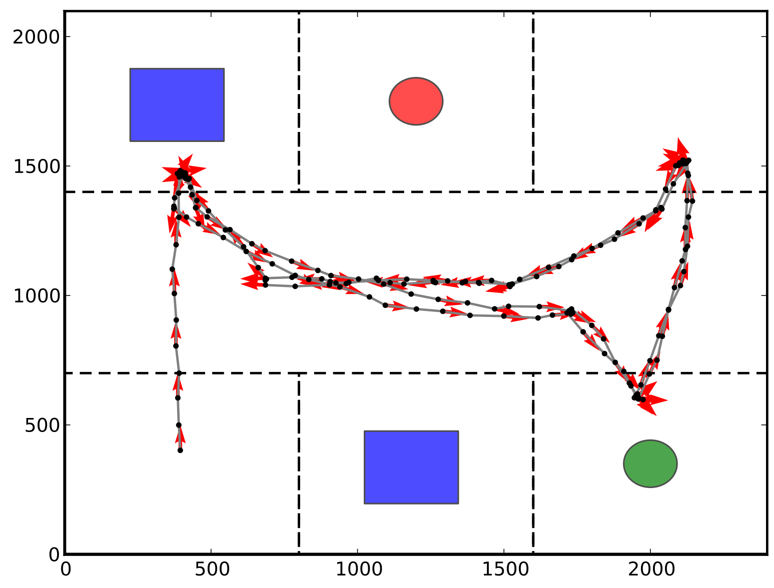


Figure 2.8: the actual trajectory that fulfills φ_5 by (2.16).

The task is specified by the LTL formula:

$$\begin{aligned}\varphi_2 = &\Diamond(\mathsf{rball} \wedge \Diamond\,\mathsf{basket}) \wedge \Diamond(\mathsf{gball} \wedge \Diamond\,\mathsf{basket}) \wedge \Diamond\Box\,\mathsf{r1}\\ &\wedge \Box(\mathsf{rball} \Rightarrow \bigcirc(\neg\mathsf{gball}\ \mathsf{U}\ \mathsf{basket}))\\ &\wedge \Box(\mathsf{gball} \Rightarrow \bigcirc(\neg\mathsf{rball}\ \mathsf{U}\ \mathsf{basket})),\end{aligned} \tag{2.14}$$

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 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 Fig. 2.6. 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 constraint.

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:

$$\begin{aligned}\varphi_3 = &\Diamond(\,\mathsf{rball} \wedge \Diamond\,(\,\mathsf{basket} \wedge \mathsf{r2}))\;\wedge\;\Diamond(\,\mathsf{gball} \wedge \Diamond\,(\,\mathsf{basket} \wedge \mathsf{r4}))\\ &\wedge\;\Box(\,\mathsf{rball} \Rightarrow \bigcirc(\,\neg\mathsf{gball}\ \mathsf{U}\ \mathsf{basket}))\\ &\wedge\;\Box(\,\mathsf{gball} \Rightarrow \bigcirc(\,\neg\mathsf{rball}\ \mathsf{U}\ \mathsf{basket}))\;\wedge\;\Diamond\Box\,\mathsf{r1},\end{aligned} \tag{2.15}$$

where the location for each basket is specified. It took $1.39s$ for Algorithm 3 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 Fig. 2.7. 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:

$$\varphi_4 = (\Box\Diamond\,\mathsf{r3}) \wedge (\Box\Diamond\,\mathsf{r4}) \wedge (\Box\Diamond\,\mathsf{r6}). \tag{2.16}$$

It took $0.01s$ for Algorithm 3 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 Fig. 2.8. It can be seen that the robot patrols these three rooms as required.

2.6 Discussion

In this chapter, we present the framework to synthesize the hybrid control strategy that navigates an autonomous robot such that a high-level task specification is fulfilled. We start from constructing the finite abstraction of the robot's motion within the workspace. Then we propose an fully-automated scheme to synthesize the discrete motion and task plan satisfying the specified task. At last, a hybrid control strategy is designed that executes the discrete plan off-line.

Chapter 3

Reconfiguration and Real-time Adaptation

THE framework presented in Chapter 2 needs the critical assumption that the workspace is fully-known in priori and remains static. The plan is synthesized once and executed off-line by the hybrid controller as it is. This renders the approach lack of reconfigurability and real-time adaptation. In this chapter, we mainly address this issue.

3.1 Potentially Infeasible Task

An intriguing question to ask about the framework introduced in Chapter 2 is what if the given task specification is infeasible. This could happen when either the task is actually infeasible or the task is feasible but the initial workspace model is incomplete or incorrect.

Problem 3.1. *Assume the given task specification is* ***infeasible*** *by Definition 2.6, how should the specification be* ***relaxed*** *and more importantly how to synthesize the motion and task plan that satisfies the original specification* ***as much as possible****?*

An approximate algorithm is provided in Kim and Fainekos (2012) that 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 Kim and Fainekos (2012)). Then the discrete plan can be synthesized by following the procedure as described in Section 2.4. However there are often more than one accepting run within $\mathcal{T}_c \otimes \mathcal{A}'_\varphi$ and they may fulfill the original task specification to different extents. Instead we aim to firstly find the motion and task plan that fulfills the task the most regarding certain criterion, based on which then the relaxed specification automaton is constructed.

Relaxed Product

By Definition 2.6, φ is infeasible when the standard product automaton $\mathcal{A}_p$ does not have an accepting run. Thus we need to relax the constraints imposed by $\mathcal{A}_\varphi$ to allow more transitions within $\mathcal{A}_p$.

Definition 3.1. *The relaxed product Büchi automaton* $\mathcal{A}_r = \mathcal{T}_c \times \mathcal{A}_\varphi = (Q',\, 2^{AP},\, \delta',\, Q'_0,\, \mathcal{F}',\, W_r)$ *is defined as follows:*

- $Q' = \Pi \times Q = \{\langle \pi,\, q\rangle \,|\, \forall \pi \in \Pi,\, \forall q \in Q\}$.
- 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* $q_n \in \delta(q_m, l)$.
- $Q'_0 = \Pi_0 \times Q_0$ *is the set of initial states.* $\mathcal{F}' = \Pi \times \mathcal{F}$ *is the set of 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 2.9 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 $\texttt{Eval} : 2^{AP} \to \{0,\, 1\}^K$:

$$\texttt{Eval}(l) = \nu \Longleftrightarrow [\,\nu_i\,] = \begin{cases} 1 & \text{if } a_i \in l, \\ 0 & \text{if } a_i \notin l, \end{cases} \tag{3.1}$$

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:

$$\rho(l,\, l') = \|\nu - \nu'\|_1 = \sum_{i=1}^{K} |\,\nu_i - \nu'_i\,|, \tag{3.2}$$

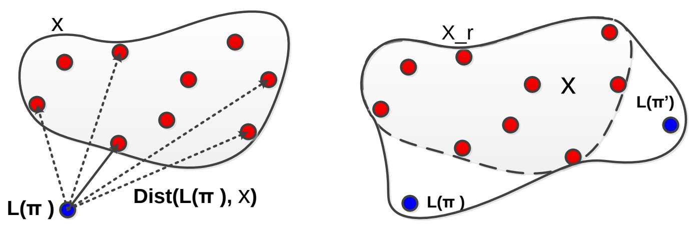


Figure 3.1: Left: the distance of $L_c(\pi)$ to a set of input alphabets χ (the solid line). Right: the input alphabets are revised by adding more elements.

where $\nu = \texttt{Eval}(l)$, $\nu' = \texttt{Eval}(l')$ 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 $\chi \subseteq 2^{AP} (\chi \neq \emptyset)$ (Boyd and Vandenberghe, 2004):

$$\texttt{Dist}(l,\, \chi) = \begin{cases} 0 & \text{if} \quad l \in \chi, \\ \min_{l' \in \chi}\ \rho(l,\, l') & \text{otherwise.} \end{cases} \tag{3.3}$$

Note that $\texttt{Dist}(l,\, \chi)$ is not defined for $\chi = \emptyset$. An example of computing $\texttt{Dist}(\cdot)$ is given in Fig. 3.1. Now we give the formal definition of W_r:

$$\begin{aligned} &W_r(\langle \pi_i,\, q_m\rangle,\, \langle \pi_j,\, q_n\rangle) \\ &= W_c(\pi_i,\, \pi_j) + \alpha\, \cdot \texttt{Dist}(L_c(\pi_i),\, \chi(q_m,\, q_n)), \end{aligned} \tag{3.4}$$

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

$$\chi(q_m,\, q_n) = \{l \in 2^{AP} |\, q_n\ \in \delta(q_m,\, l)\} \tag{3.5}$$

consists of all input alphabets that enable the transition from q_m to q_n in $\mathcal{A}_\varphi$. By Definition 3.1 there always exists $l \in 2^{AP}$ that $q_n \in \delta(q_m, l)$, thus $\chi(q_m,\, q_n) \neq \emptyset$ is ensured. $W_c(\pi_i,\, \pi_j)$ is the implementation 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. Being 0 means that $\mathcal{A}_\varphi$ is not violated, while the larger the distance is the more $\mathcal{A}_\varphi$ is violated. The design parameter α is used to reflect the relative penalty on violating the original specification, and also the user's preference on a plan that has less implementation cost or that fulfills the task specification more. The penalty on violating $\mathcal{A}_\varphi$ is increased when α is larger.

Example 3.1. Consider the NBA in Fig. 2.3 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 (2.8):

$$\begin{aligned} R &= q'_0\, q'_1 \cdots q'_{f-1} [\, q'_f\, q'_{f+1} \cdots\cdots q'_n\,]^\omega \\ &= \langle \pi_0, q_0 \rangle \cdots \langle \pi_{f-1}, q_{f-1} \rangle \, \big[\, \langle \pi_f, q_f \rangle \langle \pi_{f+1}, q_{f+1} \rangle \cdots\cdots \langle \pi_n, q_n \rangle \,\big]^\omega , \end{aligned} \tag{3.6}$$

where $q'_0 = \langle \pi_0, q_0 \rangle \in Q'_0$ and $q'_f = \langle \pi_f, q_f \rangle \in \mathcal{F}'$. However, the total cost by (2.10) is interpreted differently:

$$\begin{aligned} \texttt{Cost}(R,\, \mathcal{A}_r) &= \sum_{i=0}^{f-1} W_r(q'_i,\, q'_{i+1}) + \gamma \sum_{i=f}^{n-1} W_r(q'_i,\, q'_{i+1}) \\ &= \sum_{i=0}^{f-1} \big(W_c(\pi_i,\, \pi_{i+1}) + \alpha \cdot \texttt{Dist}(L_c(\pi_i), \chi(q_i, q_{i+1})) \big) \\ &\quad + \gamma \sum_{i=f}^{n-1} \big(W_c(\pi_i,\, \pi_{i+1}) + \alpha \cdot \texttt{Dist}(L_c(\pi_i), \chi(q_i, q_{i+1})) \big) \\ &= \texttt{cost}_\tau + \alpha \cdot \texttt{dist}_\varphi , \end{aligned} \tag{3.7}$$

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

$$\texttt{cost}_\tau = \big(\sum_{i=0}^{f-1} + \gamma \sum_{i=f}^{n-1} \big)\, W_c(\pi_i,\, \pi_{i+1}); \tag{3.8}$$

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

$$\texttt{dist}_\varphi = \big(\sum_{i=0}^{f-1} + \gamma \sum_{i=f}^{n-1} \big)\, \texttt{Dist}(L_c(\pi_i), \chi(q_i, q_{i+1})); \tag{3.9}$$

```
Algorithm 7: Validate transitions of relaxed $\mathcal{A}_\varphi$, CheckTranR( )
    Input: $(q_m,\, l,\, q_n,\, \mathcal{A}_\varphi)$, $q_m,\, q_n \in Q$, $l \in 2^{AP}$
    Output: distance $d$
1   $d =$ Dist$(l,\, \chi(q_m,\, q_n))$ by (3.3)
2   return $d$
```

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

Problem 3.2. *Find the accepting run of $\mathcal{A}_r$ that minimizes the cost by* (3.7)*.*

We call the solution to Problem 3.2 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 3.1. As mentioned in Section 2.4, each transition of the NBA is encoded by a boolean expression accepting all alphabets that enable this transition. This boolean expression is represented as a binary decision diagram (BBD), where the distance function by (3.3) can be readily integrated. The basic 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 its left and right branches. Thus it is not necessary to enumerate all input alphabets to evaluate the distance. More details can be found in Section 4.3.

Balanced Plan Synthesis

Given the values of α and γ, $\mathcal{A}_r$ can be either constructed fully by Algorithm 1 or on-the-fly by Algorithm 4. But Algorithm 2 needs to be replaced by Algorithm 7, where the distance is computed by (3.3). Consequently, given the value of α, γ, Algorithm 3 can be called with respect to the full construction of $\mathcal{A}_r$ or its adjacency relation, by which the balanced accepting run R_{bal} can be obtained. Then the corresponding balanced plan is $\tau_{\text{bal}} = R_{\text{bal}}|_\Pi$.

Remark 3.2. 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.

```
Algorithm 8: Feedback for single robot, SingleFB( )
  Input: $R_{\text{bal}}$, $\mathcal{T}_c$, $\mathcal{A}_\varphi$
  Output: $\mathcal{A}'_\varphi$, $\texttt{cost}_\tau$, $\texttt{dist}_\varphi$
  1. Initialization: $\mathcal{A}'_\varphi = \mathcal{A}_\varphi$. $\texttt{cost}_\tau = \texttt{dist}_\varphi = 0$.
  2. For each transition $(q'_i,\, q'_{i+1}) \in \texttt{Edge}(R_{\text{bal}})$, perform steps 3-5:
  3. Let $q'_i = \langle \pi_i,\, q_m \rangle$ and $q'_{i+1} = \langle \pi_j,\, q_n \rangle$.
  4. $\texttt{cost}_\tau = \texttt{cost}_\tau + W_c(\pi_i,\, \pi_j)$.
  5. $d = \texttt{CheckTranR}(q_m,\, L_c(\pi_i),\, q_n, \mathcal{A}_\varphi)$. If $d > 0$, add $q_n$ to
  $\delta(q_m,\, L_c(\pi_i))$ of $\mathcal{A}'_\varphi$. $\texttt{dist}_\varphi = \texttt{dist}_\varphi + d$.
  return $\mathcal{A}'_\varphi$, $\texttt{cost}_\tau$, $\texttt{dist}_\varphi$
```

Furthermore, Algorithm 8 takes as inputs R_{bal}, $\mathcal{T}_c$ and $\mathcal{A}_\varphi$ and 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} by (2.9), it constructs $\mathcal{A}'_\varphi$ by adding new transitions to $\mathcal{A}_\varphi$; it accumulates $\texttt{cost}_\tau$ and $\texttt{dist}_\varphi$ as defined in (3.8) and (3.9). It can be verified that the obtained $\mathcal{A}'_\varphi$ is a valid relaxation of $\mathcal{A}_\varphi$ (Kim and Fainekos, 2012). Note each R_{bal} corresponds to an unique balanced plan τ_{bal} and a revised specification automaton $\mathcal{A}'_\varphi$.

Lemma 3.1. *If* $\texttt{dist}_\varphi = 0$*, then* $\tau_{\text{bal}} \models \varphi$.

Proof. Since $\texttt{Dist}(\cdot) \geq 0$ by (3.3), 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 2.9. Then its corresponding plan τ_{bal} satisfies φ by Lemma 2.3. ■

However it may not be trivial to determine the appropriate value of α for the desired balance between the implementation cost and distance to the task specification. As an extension, Algorithm 3 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 implemented off-line by constructing the hybrid controller as proposed in Algorithm 6.

Theorem 3.1. *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* (3.10).

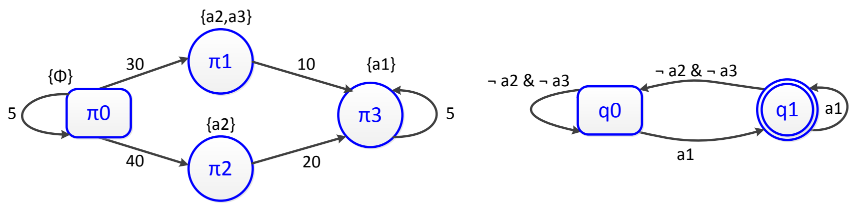


Figure 3.2: Left: the FTS $\mathcal{T}_c$ has four states, labeled by the propositions they satisfy. Transitions are labeled by the costs. Right: the NBA $\mathcal{A}_\varphi$ associated with $\varphi = \Diamond\Box a_1 \wedge \Box\neg(a_2 \wedge a_3)$.

Proof. If φ is feasible over $\mathcal{T}_c$, Algorithms 3 and 2 return the optimal accepting run R_{opt} with the total cost (under the same γ) by (2.10):

$$\texttt{Cost}(R_{\text{opt}}, \mathcal{A}_p) = \underline{\alpha}. \tag{3.10}$$

It is easy to show that 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 (3.9). As a result, the total cost of R_{bal} by (3.7) satisfies:

$$\texttt{Cost}(R_{\text{bal}}, \mathcal{A}_r) > \alpha \cdot \texttt{dist}_\varphi > \alpha. \tag{3.11}$$

Since $\alpha > \underline{\alpha} = \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_p) = \texttt{Cost}(R_{\text{opt}}, \mathcal{A}_r)$, (3.11) 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 $\mathcal{T}_c$ without any modification but choosing a large enough α. Algorithms 3 and 7 will automatically select the accepting run that satisfies φ, i.e., $\texttt{dist}_\varphi = 0$. ∎

Example 3.2. As shown in Fig. 3.2, 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 motion plans are obtained by varying α ($\gamma = 5$), as shown in Fig. 3.3: (i) when the penalty on violating φ is low, $\mathcal{A}_\varphi$ is revised by adding

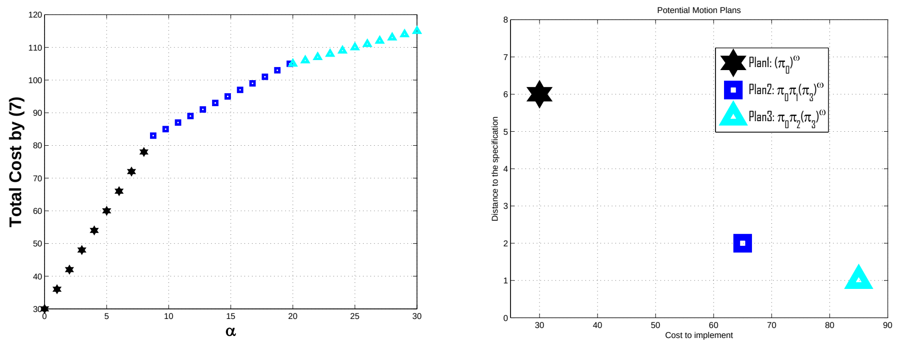


Figure 3.3: Left: the total cost of the balanced accepting run when $\gamma = 5$ under different α. Right: the unique balanced runs, located by their $\texttt{cost}_\mathcal{T}$ (x-axis) and $\texttt{dist}_\varphi$ (y-axis).

q_1 to $\delta(q_0, \emptyset)$, q_1 to $\delta(q_1, \emptyset)$ and the balanced plan is $[\pi_0]^\omega$ (black hexagram, $\texttt{cost}_\mathcal{T}$ 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, $\texttt{cost}_\mathcal{T}$ 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 is $\pi_0\,\pi_2\,[\pi_3]^\omega$ (cyan triangle, $\texttt{cost}_\mathcal{T}$ 85, $\texttt{dist}_\varphi$ 1). Note that in (iii) the agent passes through π_2 which satisfies only a_2, instead of π_1 which satisfies both a_2 and a_3. ▲

3.2 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 might not be feasible and 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 2.3. Its task specification remains an LTL formula over AP, but with the

Algorithm 9: Relaxed intersection, `RelaxInt( )`

```
    Input: $\mathcal{A}^{\text{soft}}$, $\mathcal{A}^{\text{hard}}$
    Output: $\tilde{\mathcal{A}}_{\varphi}$
 1  foreach $q_1 \in Q_1,\, q_2 \in Q_2,\, t \in \{1, 2\}$ do
 2      $q_m = \langle q_1, q_2, t\rangle \in Q$
 3      if $q_1 \in Q_{1,0},\, q_2 \in Q_{2,0},\, t \in \{1\}$ then
 4          $q_m \in Q_0$
 5      if $q_1 \in \mathcal{F}_1,\, t \in \{1\}$ then
 6          $q_m \in \mathcal{F}$
 7      foreach $\check{q}_1 \in \texttt{Post}(q_1),\, \check{q}_2 \in \texttt{Post}(q_2),\, \check{t} \in \{1, 2\}$ do
 8          $q_n = \langle \check{q}_1, \check{q}_2, \check{t}\rangle \in Q$
 9          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
10              $q_n \in \delta(q_m, l),\, \forall l \in \chi_1(q_1,\, \check{q}_1)$
11  return $\tilde{\mathcal{A}}_{\varphi}$
```

following structure:

$$\varphi = \varphi^{\text{soft}} \wedge \varphi^{\text{hard}}, \tag{3.12}$$

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 3.3 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 3.3. *Given the task specification by* (3.12)*, how to synthesize the motion and task plan such that φ^{hard} is satisfied* ***fully****, while φ^{soft} is satisfied* ***the most****?*

Safety-ensured Synthesis

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

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 2.4. The functions $\chi_1(\cdot)$ of $\mathcal{A}^{\text{hard}}$ and $\chi_2(\cdot)$ of $\mathcal{A}^{\text{soft}}$ are defined analogously as (3.5). Now we propose a way to construct the relaxed but safety-ensured intersection of $\mathcal{A}^{\text{hard}}$ and $\mathcal{A}^{\text{soft}}$.

Definition 3.3. *The relaxed intersection of $\mathcal{A}^{\text{hard}}$ and $\mathcal{A}^{\text{soft}}$ is defined by:*

$$\tilde{\mathcal{A}}_\varphi = (Q,\, 2^{AP},\, \delta,\, Q_0,\, \mathcal{F}), \tag{3.13}$$

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) $\chi_2(q_2,\, \check{q}_2) \neq \emptyset$; (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 9 constructs $\tilde{\mathcal{A}}_\varphi$ by Definition 3.3. 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 Baier et al. (2008)). 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 3.2 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 3.2. *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 3.3, 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 2.8, 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.

Definition 3.4. *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* $q_n \in \delta(q_m,\, L_c(\pi_i))$.
- $Q'_0 = \Pi_0 \times Q_0$ *is the set of initial states.* $\mathcal{F}' = \Pi \times \mathcal{F}$ *is the set of accepting states.*
- $W_r : Q' \times Q' \to \mathbb{R}^+$ *is the weight function.*

$$\begin{aligned} &W_r(\langle \pi_i,\, q_m\rangle,\, \langle \pi_j,\, q_n\rangle) \\ &= W_c(\pi_i,\, \pi_j) + \alpha \cdot \texttt{Dist}(L_c(\pi_i),\, \chi_2(q_2,\, \check{q}_2)) \end{aligned} \tag{3.14}$$

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* (3.3). ▲

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 3.3. *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 3.2.*

Proof. The fact that τ is valid can be verified from the definition of δ': every transition in δ' when projected onto Π is a valid transition within $\longrightarrow_c$. Secondly, since R is an accepting run of $\tilde{\mathcal{A}}_r = \mathcal{T}_c \times \tilde{\mathcal{A}}_\varphi$, then $\texttt{trace}(\tau) \in \mathcal{L}_\omega(\tilde{\mathcal{A}}_\varphi)$, which implies $\texttt{trace}(\tau) \in \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$ by Theorem 3.2. Since $\texttt{Words}(\varphi^{\text{hard}}) = \mathcal{L}_\omega(\mathcal{A}^{\text{hard}})$ by Lemma 2.1, $\texttt{trace}(\tau) \in \texttt{Words}(\varphi^{\text{hard}})$, which implies $\tau \models \varphi^{\text{hard}}$, thus τ is also safe. ■

Algorithm 10: Validate transitions of $\tilde{\mathcal{A}}_\varphi$, `CheckTranS( )`

```
    Input: $(q_m,\, l,\, q_n,\, \tilde{\mathcal{A}}_\varphi)$, $q_m,\, q_n \in Q$, $l \in 2^{AP}$
    Output: distance $d$
1   $q_m = \langle q_1,\, q_2,\, t\rangle$, $q_n = \langle \check{q}_1,\, \check{q}_2,\, \check{t}\rangle$
2   $d = -1$
3   if $q_n \in \delta(q_m,\, l)$ then
4       $d =$ Dist$(l,\, \chi_2(q_2,\, \check{q}_2))$ by (3.3)
5   return $d$
```

Same as before, in order 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 (3.6). However its total cost is defined with respect to the soft specification:

$$\begin{aligned}\texttt{Cost}(R,\, \tilde{\mathcal{A}}_r) &= \sum_{i=0}^{f-1} W_r(q'_i,\, q'_{i+1}) + \gamma \sum_{i=f}^{n-1} W_r(q'_i,\, q'_{i+1}) \\ &= \texttt{cost}_\tau + \alpha \cdot \texttt{dist}_{\varphi^{\text{soft}}}\,,\end{aligned} \tag{3.15}$$

where $\gamma \geq 0$; the accumulated 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 3.4. *Find the accepting run of $\tilde{\mathcal{A}}_r$ that **minimizes** the total cost by* (3.15).

We call the solution to Problem 3.4 as the *safe* accepting run of $\tilde{\mathcal{A}}_r$, denoted by R_{safe}. The corresponding *safe* plan is $\tau_{\text{safe}} = R_{\text{safe}}|_\Pi$.

Safe Plan Synthesis

Given the values of α and γ, $\tilde{\mathcal{A}}_r$ can be either constructed fully by Algorithm 1 or on-the-fly by Algorithm 4. But Algorithm 2 needs to be replaced by Algorithm 10 where 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 with respect to $\tilde{\mathcal{A}}_r$. By Theorem 3.3, its corresponding plan τ_{safe} is always valid and safe no matter how the value of α and γ are chosen.

Same as in Section 3.1, the value of α could be tuned by calling Algorithm 3 under different α to generate candidates of R_{safe}. The associated $\texttt{cost}_\tau$ and $\texttt{dist}_{\varphi^{\text{soft}}}$ can be computed similarly as in Algorithm 8. After deciding the safe accepting run, its corresponding plan τ_{safe} can be implemented by the hybrid control strategy following Algorithm 6.

Lemma 3.2. *If* $\texttt{dist}_{\varphi^{\text{soft}}} = 0$, *then* $\tau_{\text{safe}} \models \varphi$.

Proof. By Lemma 3.1, 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 2.9, where $\mathcal{A}_\varphi$ is the un-relaxed intersection (Baier et al., 2008) of $\mathcal{A}_{\varphi^{\text{soft}}}$ and $\mathcal{A}_{\varphi^{\text{hard}}}$. Thus its corresponding plan τ_{safe} satisfies φ by Lemma 2.3. ■

3.3 Partially-known Workspace

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 planning and reconfiguration framework is needed, where the planning and execution are interleaved as shown in Fig. 3.4. If the actual workspace is different from the workspace model embodied in the transition system, it is crucial to put the planner on-line and make it dynamic, such that it can monitor the execution of the plan, update the system model based on the real-time observation, and validate or revise the current plan.

Thus in this section we analyze how the transition system could be updated based on the robot's observations. The observations could come from both its sensing ability and the communication functionality. We

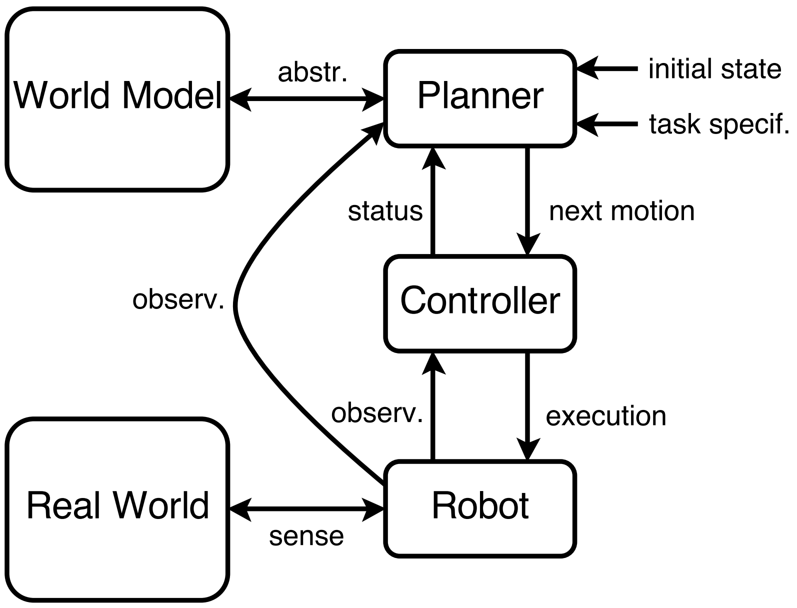


Figure 3.4: Diagram for dynamic planning, where the planning and execution are interleaved

denoted by $\mathcal{T}_c^t$ the transition system at time $t \geq 0$, particularly

$$\mathcal{T}_c^t = (\Pi,\, \longrightarrow_c^t,\, \Pi_0,\, AP,\, L_c^t,\, W_c^t), \tag{3.16}$$

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.

Furthermore its task specification consists of hard and soft parts: $\varphi = \varphi^{\text{soft}} \wedge \varphi^{\text{hard}}$, as introduced in (3.12), which is invariant after the system starts. Denote by $\tilde{\mathcal{A}}_\varphi$ the relaxed intersection automaton from Algorithm 9.

Problem 3.5. *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****.*

Initial Synthesis

At $t = 0$, the initial motion and task plan can be obtained by following the approach proposed in Section 3.2, 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 $t \geq 0$, respectively. In this section, we assume that $\tilde{\mathcal{A}}_r^t$ is constructed on-the-fly by Algorithms 4 and 10 and R^t is obtained by Algorithm 3 for $\tilde{\mathcal{A}}_r^t$. Note that the soft specification may not be feasible initially but it is guaranteed by Theorem 3.3 that τ^0 is always safe and valid for $\mathcal{T}_c^0$.

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:

$$\mathbf{Sense}^t = \{\big((\pi,\, S,\, S_\neg),\, E,\, E_\neg\big)\}, \tag{3.17}$$

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 $(\pi_i,\, \pi_j)$ 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 $(\pi_i,\, \pi_j)$ needs to be removed from $\longrightarrow_c^t$. $\mathbf{Sense}^t$ reflects the actual workspace at time t.

Example 3.3. The sensing info $\big((\pi_1, \{a_1,\, a_3\}, \{a_2\}),\, (\pi_1,\, \pi_2,\, 10),\, (\pi_1,\, \pi_3)\big) \in \mathbf{Sense}^t$ 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 necessary to abstract the essential information for (3.17) 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:

$$\mathbf{Request}^t = \varphi|_{AP}\,, \tag{3.18}$$

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

$$\mathbf{Reply}^t = \{(\pi',\, S',\, S'_{\neg})\}, \tag{3.19}$$

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 3.4. 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 agent 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. The above descriptions are summarized in Algorithm 11: $\widetilde{\Pi} \subseteq \Pi$ is used to store the set of regions within Π of which the labeling function is changed during the update; $\widehat{\Pi}$ in Line 8 stores the set of regions of which the adjacency relation needs to be reconstructed in Line 5 of Algorithm 4. Note that if both $\mathbf{Sense}^t$ and $\mathbf{Reply}^t$ are empty, $\mathcal{T}_c^{t^+}$ remains the same as $\mathcal{T}_c^{t^-}$.

Algorithm 11: Transition system update, `UpdaT( )`

```
   Input: $\mathcal{T}_c^{t^-}$, $\mathbf{Sense}^t$, $\mathbf{Reply}^t$.
   Output: $\mathcal{T}_c^{t^+}$, $\widetilde{\Pi}$, $\widehat{\Pi}$
1  $\mathcal{T}_c^{t^+} = \mathcal{T}_c^{t^-}$
2  foreach $(\pi,\, S,\, S_\neg) \in \mathbf{Sense}^t$ or $\mathbf{Reply}^t$ do
3      if $\pi \in \Pi$ then
4          $L_c^{t^+}(\pi) = L_c^{t^-}(\pi) \cup S \setminus S_\neg$
5          if $L_c^{t^+}(\pi) \neq L_c^{t^-}(\pi)$ then
6              add $\pi$ to $\widetilde{\Pi}$
7  remove $(\pi_i,\, \pi_j)$ from $\longrightarrow_c^{t^+}$, $\forall (\pi_i,\, \pi_j) \in E_\neg$
8  add $(\pi_i,\, \pi_j)$ to $\longrightarrow_c^{t^+}$, $W_c^{t^+}(\pi_i,\, \pi_j) = w$, $\forall (\pi_i,\, \pi_j,\, w) \in E$
9  $\widehat{\Pi} = \widetilde{\Pi} \cup \{\pi_i \,|\, (\pi_i,\, \pi_j) \in E,\ or\ (\pi_i, \pi_j, w) \in E_\neg\}$
10 return $\mathcal{T}_c^{t^+}$, $\widetilde{\Pi}$, $\widehat{\Pi}$
```

Real-time 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 3.2 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 Algorithm 1 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 Guo et al. (2013b). 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 14 later.

Thus we propose to incorporate the update information including $\widetilde{\Pi}$, E and $E_\neg$ into the adjacency relation function of $\tilde{\mathcal{A}}_r^t$ in Algorithm 4. As shown in Line 8 of Algorithm 11, for any region $\pi_i \in \widetilde{\Pi}$ whose labeling function or adjacency relation has been changed, they are stored in a new set $\widehat{\Pi}$, namely

Algorithm 12: Validate the current run, `ValidRun( )`

```
   Input: $\tilde{\mathcal{A}}_r^{t^+}$, $R^t$, $\widetilde{\Pi}$, $E_\neg$
   Output: $\Xi$, $\aleph$
1  $\Xi = \aleph = \emptyset$
2  forall $(q_s',\, q_{s+1}') \in$ Edge$(R^t)$ do
3      $q_s' = \langle \pi_i,\, q_m\rangle$, $q_{s+1}' = \langle \pi_j,\, q_n\rangle$
4      if $(\pi_i,\, \pi_j) \in E_\neg$ then
5          add $(q_s',\, q_{s+1}')$ to $\Xi$
6      else if $\pi_i \in \widetilde{\Pi}$ then
7          $d =$ CheckTranS$(q_m,\, L_c^{t^+}(\pi_i),\, q_n,\, \tilde{\mathcal{A}}_\varphi)$
8          if $d < 0$ then
9              add $(q_s',\, q_{s+1}')$ to $\aleph$
10 return $\Xi$, $\aleph$
```

$$\widehat{\Pi} = \widetilde{\Pi} \cup \{\pi_i \,|\, (\pi_i,\, \pi_j) \in E,\ or\ (\pi_i,\, \pi_j, w) \in E_\neg\}. \tag{3.20}$$

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. This is implemented by Line 5 of Algorithm 4. In this way, the update information is used only when q_s' is revisited by the revision Algorithm 14 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?

Validity and Safety

Recall that the current accepting run R^t has a finite set of transitions appearing in it, i.e., `Edge`(R^t) by (2.9). By checking `Edge`(R^t), we could validate if the current plan τ^t is still valid or safe.

Definition 3.5. *Given the updated transition system $\mathcal{T}_c^{t^+}$ from Algorithm 11, any transition $(\langle \pi_i,\, q_m\rangle,\, \langle \pi_j,\, q_n\rangle) \in$ `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 $\mathcal{A}^{\text{hard}}$. The notations Ξ and $\aleph$ in Algorithm 12 are used to store the sets of invalid and unsafe transitions in R^t. Algorithm 12 iterates 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 $\mathcal{T}_c^{t^+}$ (line 3) or if the changed label $L_c^{t^+}(\pi_i)$ would make this transition unsafe (line 6), where function $\texttt{CheckTranS}(\cdot)$ by Algorithm 10 is called.

Theorem 3.4. *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 obtained from Algorithm 12. Then (i) τ^t remains **valid** if and only if $\Xi = \emptyset$; (ii) τ^t remains **safe** if $\aleph = \emptyset$.*

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 3.3. If $\Xi = \emptyset$ and $\aleph = \emptyset$, $\texttt{Edge}(R^t)$ 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 3.3, τ^t is valid since R^t remains an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$. "Only if" part of (i): if $\Xi \neq \emptyset$, τ^t contains at least one invalid transition, thus not valid by Definition 3.2. "If" part of (ii): by Theorem 3.3, τ^t is safe since R^t remains an accepting run of $\tilde{\mathcal{A}}_r^{t^+}$. ∎

Safety-ensured Plan Revision

If Ξ or $\aleph$ are not empty from Algorithm 12, 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 with respect to $\mathcal{T}_c^{t^+}$ and $\mathcal{A}_\varphi$, but using the robot's current region π_{cur} from Algorithm 6 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 3.3, 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 6, its complete trajectory is obtained by concatenating τ_{past} with τ_{new}, namely

$$\tau_{\text{comp}} = \tau_{\text{past}} + \tau_{\text{new}}. \tag{3.21}$$

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 6 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 $\tilde{\mathcal{A}}_r^{t^+}$.

Algorithm 13: Corresponding product states, `CorProd( )`

```
  Input: $\tilde{\mathcal{A}}_r^t$, $\tau_{\text{past}}$
  Output: $Q'_{\tau_{\text{past}}}$
1 $S_1 = \{\langle \tau_{\text{past}}[1], q_0 \rangle \,|\, q_0 \in Q_0\}$
2 for $k = 2 : |\tau_{\text{past}}|$ do
3     $S_2 = \emptyset$
4     forall $q'_s \in S_1$ do
5         forall $q'_g \in \delta'(q'_s)$ do
6             if $q'_g|_\Pi = \tau_{\text{past}}[k]$ then
7                 add $q'_g$ to $S_2$
8         $S_1 = S_2$
9 return $Q'_{\tau_{\text{past}}} = S_1$
```

Problem 3.6. *Assume Ξ or $\aleph$ are not empty from Algorithm 12. Given the robot's past trajectory τ_{past}, how to find the* ***new plan*** *τ_{new} such that its complete trajectory by* (3.21) *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 2.8, 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 $r_{\tau_{\text{past}}}$:

$$r_{\tau_{\text{past}}} = \{r_{\bar{\sigma}} \text{ by Def. 2.8 for } \tilde{\mathcal{A}}_\varphi \,|\, \bar{\sigma} = \texttt{trace}(\bar{\tau}_{\text{past}})\}, \tag{3.22}$$

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 finite runs in $\tilde{\mathcal{A}}_r^{t^+}$ is given by

$$R_{\tau_{\text{past}}} = \{R \,|\, R|_\Pi = \tau_{\text{past}},\ R|_Q \in r_{\tau_{\text{past}}}\}, \tag{3.23}$$

where each finite run R is the synchronized product of τ_{past} and any run belonging to $r_{\tau_{\text{past}}}$ by (3.22). 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

$$Q'_{\tau_{\text{past}}} = \{\texttt{last}(R) \,|\, R \in R_{\tau_{\text{past}}}\}, \tag{3.24}$$

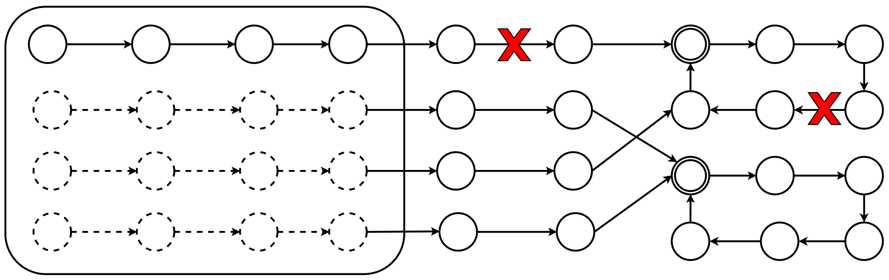


Figure 3.5: Locally revise the invalid transitions in the current accepting run R^t. The sequences of states in dashed line represent the corresponding runs given the robot's past trajectory.

where `last`(R) is the last state of the finite run R belonging to $R_{\tau_{\text{past}}}$. Clearly, $R_{\text{past}} \in R_{\tau_{\text{past}}}$ and $q'_{\text{cur}} \in Q'_{\tau_{\text{past}}}$. Given τ_{past}, the corresponding $Q'_{\tau_{\text{past}}}$ can be derived by Algorithm 13.

To solve Problem 3.6, we propose an algorithm that is similar to Algorithm 5 in Guo et al. (2013b). Denote by R^{t^-}, R^{t^+}, τ^{t^-}, τ^{t^+} the accepting run and corresponding plan before and after the revision at time t, 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^-} from Algorithm 12, it belongs to either $R^{t^-}_{\text{pre}}$ or $R^{t^-}_{\text{suf}}$. In Algorithm 14, it firstly tries to locally finds a "bridging" segment that make up this transition by breadth-first search (as shown in Fig. 3.5). `tail`$(q'_{s+1}, R^{t^-}_{\text{pre}})$ is the segment 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`$(\textit{bridge})$ is the last state on the path *bridge*. Function `DijksTarget`$(\tilde{\mathcal{A}}^{t^+}_r,\, q'_S,\, Q'_T)$ is defined similarly as function `DijksTargets`$(\cdot)$ in Algorithm 3, which returns shortest path from the single source state $q'_S \in Q'$ to *any* target state belonging to the set $Q'_T \subseteq Q'$; $\tilde{\mathcal{A}}^{t^+}_r$ is the adjacency function from Algorithm 4. Thus it returns the shortest path once one of the targets is reached.

At last, if the call to 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 Algorithm 3 is called to search for the balanced accepting run of $\tilde{\mathcal{A}}^{t^+}_r$, but using $Q'_{\tau_{\text{past}}}$ from Algorithm 13 as the set of initial states, instead

Algorithm 14: Revise the current plan, `Revise( )`

```
   Input: $\Xi$, $\aleph$, $R^{t^-}$, $\tilde{\mathcal{A}}_r^{t^+}$, $Q'_{\tau_{\text{past}}}$
   Output: $R^{t^+}$
1  forall $(q'_s, q'_{s+1}) \in (\Xi \cup \aleph)$ do
2      if $(q'_s, q'_{s+1}) \in R^{t^-}_{\text{pre}}$ then
3          $bridge$=DijksTarget($\tilde{\mathcal{A}}_r^{t^+}$, $q'_{s-1}$, tail($q'_s$, $R^{t^-}_{\text{pre}}$))
4          if $bridge \neq \emptyset$ then
5              $R^{t^-}_{\text{pre}}$ = head($q'_{s-1}$, $R^{t^-}_{\text{pre}}$)+$bridge$+tail(last($bridge$), $R^{t^-}_{\text{pre}}$)
6          else
7              $R^{t^+}$ = OptRun($\tilde{\mathcal{A}}_r^{t^+}$, $Q'_{\tau_{\text{past}}}$)
8              return $R^{t^+}$
9      if $(q'_s, q'_{s+1}) \in R^{t^-}_{\text{suf}}$ then
10         repeat line 3-8 but replace $R^{t^-}_{\text{pre}}$ by $R^{t^-}_{\text{suf}}$
11 return $R^{t^+} = \langle R^{t^-}_{\text{pre}}, R^{t^-}_{\text{suf}} \rangle$
```

of the default Q'_0. Note that 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 (lines 2 and 9).

Theorem 3.5. *The new plan τ_{new} can be obtained from Algorithm 14 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 2.2 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 14 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 in Line 3 that revise the current accepting run 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 17 later, where the accepting run R^t is updated by both the revision Algorithm 14 and the optimal search Algorithm 3. As a result, the next goal region for the hybrid controller also changes accordingly.

Case Study

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 Fig. 3.6, which consists of 12 polygonal regions. The continuous controller that drives the robot from an region to any geometrically adjacent region is based on Lindemann et al. (2006), 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 “$\varphi = (\Box\Diamond a_1) \wedge (\Box\Diamond a_2) \wedge (\Box\Diamond a_3) \wedge (\Box\neg a_4)$” and its associated NBA is shown in Fig. 2.3.

Simulation Results

The preliminary workspace is initialized as obstacles free and the associated $\mathcal{T}_c^0$ is constructed by Definition 2.3. The actual workspace is shown in Fig. 3.6, 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 and 6 (arrowed red line in Fig. 3.6), 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 $(\pi_9,\, \{a_4\},\, \emptyset)$, namely region 9 satisfies a_4. The updated motion plan is illustrated by the arrowed red lines in Fig. 3.6. 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. 32 transitions are removed in both cases. But the motion and task plan remains

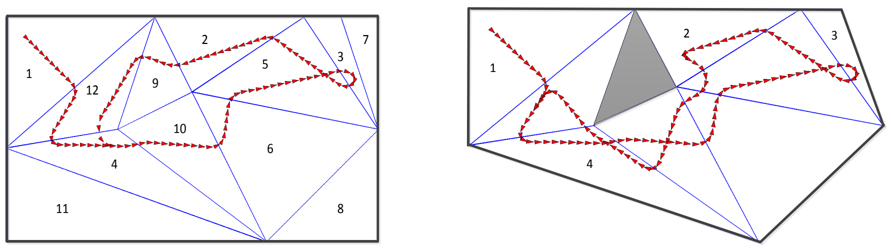


Figure 3.6: Left: Initial workspace and the the preliminary motion and task plan. Right: actual workspace and the final trajectory.

valid because its corresponding accepting run remains valid. The final trajectory is shown in Fig. 3.6.

3.4 Motion and Action Planning

Until now all the task and motion plans only involve a sequence of regions to visit, such that the robot's trajectory satisfies the specification. However, to solve problems of practical interest 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. Action description language (Gelfond and Lifschitz, 1998) provides an intuitive and powerful way for describing the preconditions and effects of different actions. Moreover, we propose to separate the domain-specific knowledge (Van Harmelen et al., 2008) 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.

In this section, to distinguish the finite transition systems for mobility and action, we replace the wFTS $\mathcal{T}_c$ in Definition 2.3 with $\mathcal{M}$. Namely, the abstraction of agent's *mobility* is given by:

$$\mathcal{M} = (\Pi_{\mathcal{M}},\, act_{\mathcal{M}},\, \longrightarrow_{\mathcal{M}},\, \Pi_{\mathcal{M},0},\, \Psi_{\mathcal{M}},\, L_{\mathcal{M}},\, W_{\mathcal{M}}), \tag{3.25}$$

which is defined in the same way as $\mathcal{T}_c$; $act_{\mathcal{M}}$ stands for the control strategy (2.3) symbolically; $\Psi_{\mathcal{M}} = \Psi_r \cup \Psi_p$ where $\Psi_r = \{\Psi_{r,i}\}$, $i = 1, \cdots, N$, is the set of propositions indicating the agent position; $\Psi_p = \{\Psi_{p,1}, \cdots, \Psi_{p,I}\}$ is the finite set of propositions indicating 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".

Action Description Language

Classic planning formalisms, like STRIPS Fikes and Nilsson (1972), ADL Pednault (1989), 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 $\{act_{\mathcal{B},1}, \cdots, act_{\mathcal{B},K}\}$, implementable by the corresponding low-level controllers $\{\mathcal{K}_k\}$, $k = 1, \cdots, K$. Denote by $Act_{\mathcal{B}} = \{act_{\mathcal{B},0}, \cdots, act_{\mathcal{B},K}\}$, where $act_{\mathcal{B},0} \triangleq \texttt{None}$ 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 $\Psi_{b,k} = \texttt{True}$ 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" of the robot. With Ψ_p, Ψ_s and Ψ_b, we can describe each action in $Act_{\mathcal{B}}$ by the precondition and effect functions.

Precondition and Effect

The precondition function

$$\mathsf{Cond} : Act_{\mathcal{B}} \times 2^{\Psi_p} \times 2^{\Psi_s} \longrightarrow \mathsf{True/False}, \tag{3.26}$$

Table 3.1: Action Description for Section 3.4

Action	**Condition**	**Effect**
$act_{\mathcal{B},0}$	`True`	$\Psi_{b,0} = \mathsf{T}$, $\Psi_{b,\sim 0} = \mathsf{F}$
$act_{\mathcal{B},1}$	$\Psi_{p,1} \,\&\, \neg\Psi_{s,1} \,\&\, \neg\Psi_{s,2}$	$\Psi_{s,1} = \mathsf{T}$, $\Psi_{b,1} = \mathsf{T}$, $\Psi_{b,\sim 1} = \mathsf{F}$
$act_{\mathcal{B},2}$	$\Psi_{s,1}$	$\Psi_{s,1} = \mathsf{F}$, $\Psi_{b,2} = \mathsf{T}$, $\Psi_{b,\sim 2} = \mathsf{F}$
$act_{\mathcal{B},3}$	$\Psi_{p,2} \,\&\, \neg\Psi_{s,2} \,\&\, \neg\Psi_{s,1}$	$\Psi_{s,2} = \mathsf{T}$, $\Psi_{b,3} = \mathsf{T}$, $\Psi_{b,\sim 3} = \mathsf{F}$
$act_{\mathcal{B},4}$	$\Psi_{s,2}$	$\Psi_{s,2} = \mathsf{F}$, $\Psi_{b,4} = \mathsf{T}$, $\Psi_{b,\sim 4} = \mathsf{F}$
$act_{\mathcal{B},5}$	`True`	$\Psi_{b,5} = \mathsf{T}$, $\Psi_{b,\sim 5} = \mathsf{F}$

takes one action in $Act_{\mathcal{B}}$, 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 Ψ_p and the robot's internal states Ψ_s have to be fulfilled. For instance, the action "pickup A" can only be performed when "the room has product A". While some actions like "take pictures" might be performed without such constraints and then the condition is simply `True`. Note the condition for $act_{\mathcal{B},0}$ is always `True` by definition.

The effect function

$$\texttt{Eff} : Act_{\mathcal{B}} \times (2^{\Psi_s} \times \Psi_b) \longrightarrow (2^{\Psi_s} \times \Psi_b), \tag{3.27}$$

represents the effect of the actions. As a result of performing action $act_{\mathcal{B},k}$, the robot's internal states Ψ_s might be changed and Ψ_b is changed to indicate which action is performed. More specifically,

(i) $\texttt{Eff}(act_{\mathcal{B},0}, w_s, \Psi_{b,k}) = (w_s,\, \Psi_{b,0})$, where $w_s \subseteq 2^{\Psi_s}$ and $\forall \Psi_{b,k} \in \Psi_b$. Performing $act_{\mathcal{B},0}$ does not change the robot's internal state and all elements in Ψ_b except $\Psi_{b,0}$ are set to `False`;

(ii) $\texttt{Eff}(act_{\mathcal{B},k},\, w_s,\, \Psi_{b,l}) = (w'_s,\, \Psi_{b,k})$, where $w_s,\, w'_s \subseteq 2^{\Psi_s}$ and $\Psi_{b,l},\, \Psi_{b,k} \in \Psi_b$, is the effect function of $act_{\mathcal{B},k}$ 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.

Action Map

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

$$\mathcal{B} = (\Pi_{\mathcal{B}},\, Act_{\mathcal{B}},\, \Psi_p,\, \hookrightarrow_{\mathcal{B}},\, \Pi_{\mathcal{B},0},\, \Psi_{\mathcal{B}},\, L_{\mathcal{B}},\, W_{\mathcal{B}}), \tag{3.28}$$

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) $\alpha_{\mathcal{B}} \in Act_{\mathcal{B}}$, $\pi_{\mathcal{B}}, \pi'_{\mathcal{B}} \in \Pi_{\mathcal{B}}$; (2) $\texttt{Cond}\,(\alpha_{\mathcal{B}}, 2^{\Psi_p}, \pi_{\mathcal{B}}) = \texttt{True}$; (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 3.3. 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 (Baier et al., 2008) 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 based on the definition of $\hookrightarrow_{\mathcal{B}}$.

Full Functionality Model

As mentioned earlier, the mobility abstraction $\mathcal{M}$ from (3.25) and the action map $\mathcal{B}$ from (3.28) are adequate for the controller synthesis within certain problem domain. However, in order to consider richer and more complex tasks involving both regions to visit and actions to perform within these regions, we need a *complete model* of robot's functionalities that combines these two parts. We propose the following way to compose $\mathcal{M}$ and $\mathcal{B}$:

$$\mathcal{R} = (\Pi_{\mathcal{R}},\, Act_{\mathcal{R}},\, \longrightarrow_{\mathcal{R}},\, \Pi_{\mathcal{R},0},\, \Psi_{\mathcal{R}},\, L_{\mathcal{R}},\, W_{\mathcal{R}}), \tag{3.29}$$

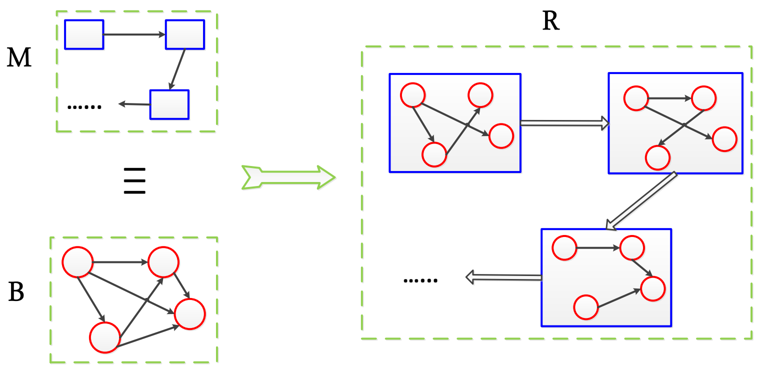


Figure 3.7: The action map $\mathcal{B}$ is composed with each region of $\mathcal{M}$, giving a complete description $\mathcal{R}$ of robot's functionalities.

where (i) $\Pi_{\mathcal{R}} = \Pi_{\mathcal{M}} \times \Pi_{\mathcal{B}}$ is the set of states; (ii) $Act_{\mathcal{R}} = act_{\mathcal{M}} \cup Act_{\mathcal{B}}$ is the set of actions; (iii) $\longrightarrow_{\mathcal{R}} \subseteq \Pi_{\mathcal{R}} \times Act_{\mathcal{R}} \times \Pi_{\mathcal{R}}$ is the transition relation, defined by the following rules:

(1) $\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle \xrightarrow{act_{\mathcal{M}}}_{\mathcal{R}} \langle \pi'_{\mathcal{M}}, \pi'_{\mathcal{B}} \rangle$ if $\pi_{\mathcal{M}} \xrightarrow{act_{\mathcal{M}}}_{\mathcal{M}} \pi'_{\mathcal{M}}$ and $\pi_{\mathcal{B}} \xrightarrow{act_{\mathcal{B},0}}_{\mathcal{B}} \pi'_{\mathcal{B}}$;

(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 $\alpha_{\mathcal{B}} \in Act_{\mathcal{B}}$;

(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) $W_{\mathcal{R}}(\langle \pi_{\mathcal{M}}, \pi_{\mathcal{B}} \rangle, act_{\mathcal{M}}, \langle \pi'_{\mathcal{M}}, \pi'_{\mathcal{B}} \rangle) = W_{\mathcal{M}}(\pi_{\mathcal{M}}, act_{\mathcal{M}}, \pi'_{\mathcal{M}})$;

(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 $\alpha_{\mathcal{R}} \in Act_{\mathcal{B}}$.

Remark 3.4. In the definition of $\hookrightarrow_{\mathcal{B}}$, $act_{\mathcal{B},0}$ is released automatically whenever the controller (2.3) is activated, because whenever the robot moves to a new region, this automatically indicates that no actions within $act_{\mathcal{B},k}$ are performed as we assume non-concurrent actions. ▲

Figure 3.7 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.

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 (Bhatia et al., 2010; Kloetzer and Belta, 2010; Smith et al., 2011), 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 $\varphi = \Box\Diamond\,(\Psi_{r,1} \wedge \Psi_{b,2})$, 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 to "pickup A" as it is modeled in the action map that to "drop A" the robot has to "pickup A" first at some regions that have A. We now state the problem we consider in this section:

Problem 3.7. *Given the full functionality description $\mathcal{R}$ and the task φ, find its motion and action plan that **minimizes** the cost defined by* (2.10) *and construct the hybrid control strategy that executes this plan.*

Since $\mathcal{R}$ remains a standard finite transition system, the framework proposed in Chapter 2 can be applied directly to find the infinite optimal motion and action sequence that has a finite representation and minimizes the cost by 2.10. We denote by R_{moac} the optimal accepting run of the product $\mathcal{R} \times \mathcal{A}_{\varphi}$ from Algorithm 3, which can be projected onto $\mathcal{R}$ as the motion and action plan $\tau_{\text{moac}} = R_{\text{moac}}|_{\Pi_{\mathcal{R}}}$.

For each pair of sequential states $(\pi_{\mathcal{R},i},\, \pi_{\mathcal{R},i+1})$ in τ_{moac} there exists an action $\alpha_{\mathcal{R}} \in Act_{\mathcal{R}}$ such that $(\pi_{\mathcal{R},i},\, \alpha_{\mathcal{R}},\, \pi_{\mathcal{R},i+1}) \in \longrightarrow_{\mathcal{R}}$ from (3.29). 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 6. In particular, if $\alpha_{\mathcal{R}} = act_{\mathcal{B},k}$, the controller $\{\mathcal{K}_k\}$ that implements $act_{\mathcal{B},k}$ is activated. If $\alpha_{\mathcal{R}} = act_{\mathcal{M}}$, the navigation controller (2.3) is applied to drive the agent from one point in the starting region to one point in the goal region. Note that external confirmation messages are needed for the completion of both motion and action.

Overall Framework

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 the motion and action plan τ_{moac}; (iv) execute the plan by the hybrid control strategy. It is worth mentioning that $\mathcal{M}$, $\mathcal{B}$ 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 reconstructed but the action map $\mathcal{B}$ remains the same and can be reused.

Case Study

In this case study, we consider an autonomous robot that repetitively delivers various products from a source region to destination regions, meanwhile avoids the prohibited regions and surveils over certain regions.

System Model

We take into account a 2-D workspace for better visualization of the results. The workspace (as shown in Fig. 3.8) is bounded by region 0: $\pi_0 = \mathcal{B}_1([\,0.5\;\;0.5\,]^T)$, where $[\,0.5\;\;0.5\,]^T \in \mathbb{R}^2$ is the center point and 1 is the radius. Within π_0 there exist five sphere regions of interest: region 1: $\pi_1 = \mathcal{B}_{0.1}([\,0\;\;0\,]^T)$, region 2: $\pi_2 = \mathcal{B}_{0.1}([\,1\;\;0\,]^T)$, region 3: $\pi_3 = \mathcal{B}_{0.1}([\,1\;\;1\,]^T)$, region 4: $\pi_4 = \mathcal{B}_{0.1}([\,0\;\;1\,]^T)$, region 5: $\pi_5 = \mathcal{B}_{0.15}([\,0.5\;\;0.5\,]^T)$. $\Psi_r = \{\Psi_{r,i}\}$ reflects the robot's position, $i = 1, \cdots, 5$. The robot satisfies the single-integrator dynamics, namely $\dot{x} = u$. Each region is potentially connected to any other region and the transition cost is estimated by the straight-line distance between them. There are three properties of concern: $\Psi_{p,1}$={this region has product A}, $\Psi_{p,2}$={this region has product B}, $\Psi_{p,3}$={this region is a office area}. It is assumed that region 1 has product A and B and region 5 is a office area. The robot starts from region 1. Then $\mathcal{M}$ can be easily constructed by (3.25). The navigation controller is derived by constructing the exact potential filed over this sphere workspace Koditschek and Rimon (1990). Details can be found in Guo et al. (2013a).

The robot is assumed to be capable of five actions: $act_{\mathcal{B},1}$={pickup A}, $act_{\mathcal{B},2}$={drop A}, $act_{\mathcal{B},3}$={pickup B}, $act_{\mathcal{B},4}$={drop B}, $act_{\mathcal{B},5}$={take pictures}, and $act_{\mathcal{B},0}$={*None*} by definition. The associated costs are 20, 20, 20, 20, 15, 5, respectively. Ψ_b indicates which action is performed. Two

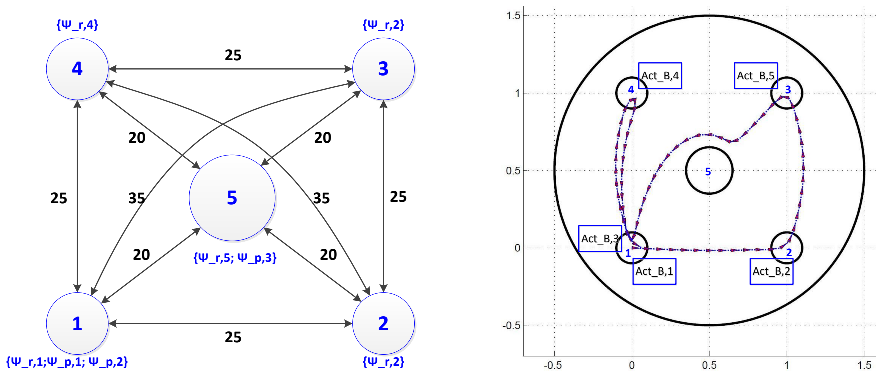


Figure 3.8: Left: the wFTS for the robot's mobility. Right: the final action and motion trajectories fulfills the task specification. The robot stays within the workspace π_0 during the whole mission. Inside the blue boxes are the actions to perform at different regions.

propositions reflecting the robot's internal states are given by $\Psi_{s,1}$={the robot has A}, $\Psi_{s,2}$={the robot has B}. The effect and condition functions paired with each action in $Act_{\mathcal{B}}$ are listed in Table 3.1 after Section 3.4. $act_{\mathcal{B},0}$ and $act_{\mathcal{B},5}$ can be performed anytime while the others have conditions on Ψ_p and/or Ψ_s. Note that the conditions on $act_{\mathcal{B},1}$ and $act_{\mathcal{B},3}$ indicated that the robot can not hold product A and B at the same time. Assume that the robot initially has no product A or B. The resulting action map $\mathcal{B}$ is then constructed by (3.28), which has $6 \times 2^2 = 24$ states. We omit the diagrams of $\mathcal{M}$ and $\mathcal{B}$ here due to limited space.

Task Specification

In plain English, the given task is to repeatedly transport product A to from region 1 to region 2 and product B from region 1 to region 4, while at the same time region 3 need to be under surveillance. Moreover, all office areas should be avoided during the whole mission. Related the propositions we have defined and the LTL, the task is reinterpreted as "**Infinitely often**, drop A in region 2, drop B in region 4, take pictures within region 3. **Always**, avoid office areas". The above task can be expressed in LTL format as

$$\begin{aligned}\varphi &= \Box\Diamond(\Psi_{r,2} \wedge \Psi_{b,2}) \wedge \Box\Diamond(\Psi_{r,4} \wedge \Psi_{b,4}) \\ &\quad \wedge \Box\Diamond(\Psi_{r,3} \wedge \Psi_{b,5}) \wedge \Box(\neg\Psi_{p,3}).\end{aligned}$$

The NBA $\mathcal{A}_\varphi$ corresponding to φ above is obtained from Gastin and Oddoux (2001), which has 4 states and 13 transitions. As can be seen here, the size of the resulting $\mathcal{A}_\varphi$ is relatively small even though the desired action is quite complex. The main reason is that we do not need to specify in φ that where the robot should go to pickup A and B.

Simulation Results

The composition $\mathcal{R} = \mathcal{M} \,|||\, \mathcal{B}$ is constructed by (3.29), which has 90 states and 606 transitions (compared with 2^{15} states when unfolding $\Psi_\mathcal{R}$ blindly). The product automaton $\mathcal{A}_\mathcal{P} = \mathcal{R} \otimes \mathcal{A}_\varphi$ is constructed fully by Algorithm 1, which has 480 states, out of which 120 are accepting states. Then Algorithm 3 is applied to $\mathcal{A}_\mathcal{P}$ to find its optimal accepting run. The final discrete plan is obtained by projecting this accepting run onto $\mathcal{R}$, which is interpreted in terms of the following sequence of actions in $\mathcal{R}$: pickup A in region 1 $\longrightarrow$ move to region 2 $\longrightarrow$ drop A $\longrightarrow$ move to region 3 $\longrightarrow$ take pictures $\longrightarrow$ move to region 1 $\longrightarrow$ pickup B $\longrightarrow$ move to region 4 $\longrightarrow$ drop B $\longrightarrow$ move to region 1. Note that this sequence is cyclic and can be repeated as many times as needed. The total cost of this motion and action plan is 230. The corresponding hybrid control strategy is synthesized based on Algorithm 6. In Figure 3.8, the final trajectories are shown by the red arrowed lines and the actions performed during the motion are indicated by action names in the blue boxes.

3.5 Discussion

In this chapter, we discuss about four non-nominal scenarios as extensions to the nominal motion and task planning scheme presented in Chapter 2. In particular, we present a method to synthesize the balanced plan for potentially infeasible tasks and the safe plan for tasks with hard and soft constraints. Then we present a dynamic planning and revising scheme for partially-known workspace, which guarantees safety and correctness. At last, we propose the motion and action planning scheme to handle broader task specifications.

Chapter 4

Multi-agent System with Locally-assigned Tasks

The multi-agent system we consider consists of a team of cooperative agents with different and locally-assigned individual tasks, which might be dependent or independent. Compared with the multi-agent systems designed for solving global tasks (Kloetzer and Belta, 2010; Ding et al., 2011a), this kind of distributed system favors a loosely-coupled and bottom-up formulation. In this chapter, we start from the reconfiguration problem for multi-agent systems, where some local tasks are dependent due to heterogeneity and collaborative tasks. Potential conflicts are resolved by distributed coordination. Then we propose a real-time knowledge transfer and update scheme for a team of robots coexisting within the partially-known workspace. At last, we describe the software implementation of our motion and task planning framework.

4.1 Dependent Local Tasks

The reconfiguration of multi-agent systems under local infeasible LTL specifications is more difficult than the single-agent case discussed in Sections 3.1 and 3.2, due to the following reasons: (i) the joined execution of multiple agents' tasks may not be mutually feasible even though the individual one is; (ii) the priority of each agent plays an important role when deciding whose tasks should be changed. The first aspect is because these tasks are assigned independently and some cooperative tasks have not been fully agreed before the deployment. The second aspect is because some agents' tasks are safety or security critical and have to be fulfilled all the time, meaning that other agents have to comply when there are conflicts.

Assume the system we consider consists of N agents, denoted by agent $i \in \mathcal{N} = \{1, 2 \cdots, N\}$. Moreover, we denote the finite transition system of agent i by

$$\mathcal{T}_i = (\Pi_i,\, \longrightarrow_i,\, \Pi_{i,0},\, AP_i,\, L_i,\, W_i), \tag{4.1}$$

of which the notations follow Definition 2.3; $\mathcal{T}_i$ abstracts agent i's motion within its workspace Π_i; AP_i reflects the properties concerning agent i in $\mathcal{T}_i$.

Denote by $AP_{\mathcal{N}} = AP_1 \cup AP_2 \cdots \cup AP_N$ the set of all propositions allowed. For each agent, its task specification φ_i can be specified over not only AP_i but $AP_{\mathcal{N}}$. The specification automaton for φ_i is

$$\mathcal{A}_{\varphi_i} = (Q_i,\, 2^{AP_{\mathcal{N}}},\, \delta_i,\, Q_{i,0},\, \mathcal{F}_i), \tag{4.2}$$

of which the notations follow Definition 2.7; $\chi_i(q_j,\, q'_j) = \{l \in 2^{AP_{\mathcal{N}}} | \, q'_j \in \delta(q_j,\, l)\}$ is the set of input alphabets enabling the transition from q_j to q'_j, as in (3.5).

Dependency and Mutual Feasibility

The dependencies implied by the task assignments can be defined as follows:

Definition 4.1. *Agents i and j are called **dependent** when one of the following conditions holds: (1) agent i depends on agent j if $AP_{\varphi_i} \cap AP_j \neq \emptyset$; (2) agent j depends on agent i if $AP_{\varphi_j} \cap AP_i \neq \emptyset$.* ▲

These conditions can be checked by comparing the elements within sets AP_{φ_i} and AP_j (also AP_{φ_j} and AP_i), as also proposed in Filippidis et al. (2012). Based on the dependency relation, we may define the *dependency graph* of the multi-agent system associated with tasks φ_i, $i = 1, \cdots, N$.

Definition 4.2. *The **dependency graph** $G_d = (V,\, E)$ consists of: the set of vertices $V = 1, 2 \cdots, N$ representing the agents; the set of edges $E \subseteq V \times V$ where $(i,\, j) \in E$ and $(j,\, i) \in E$ if agent i and j are dependent by Definition 4.1, $\forall i \neq j$ and $i,\, j \in V$.* ▲

Definition 4.3. *$\Theta \subseteq V$ forms a **dependency cluster** if and only if $\forall i,\, j \in \Theta$ there is a path from i to j in the dependency graph G_d.* ▲

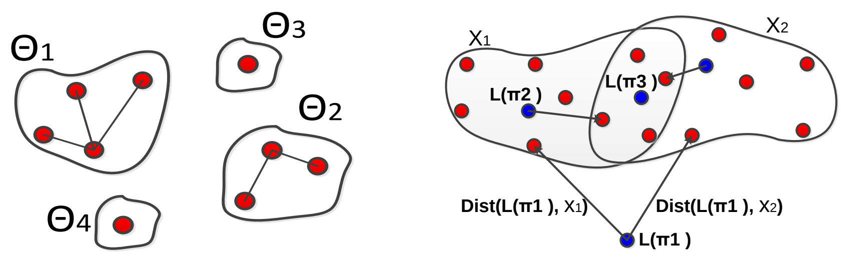


Figure 4.1: Left: the dependency graph of a system with 9 agents and 4 clusters. Right: different relative distances between $L(\pi)$ and χ_1, χ_2.

A cluster contains at least one agent, which happens when this single agent is not dependent on any of the other agents. Loosely speaking, two agents belong to the same cluster when they are directly dependent or transitively dependent by a dependency chain. An example of a dependency graph and dependency clusters are shown in Figure 4.1.

Without loss of generality, we first solve the reconfiguration problem within one cluster $\Theta = \{1, 2, \cdots, M\}$. Given the individual transition system $\mathcal{T}_i$ by (4.1), $\forall i \in \Theta$, the composed finite transition system for this cluster Θ is constructed by

$$\mathcal{T}_\Theta = (\Pi_\Theta,\, \longrightarrow_\Theta,\, \Pi_{\Theta,0},\, AP_\Theta,\, L_\Theta,\, W_\Theta), \tag{4.3}$$

where $\Pi_\Theta = \Pi_1 \times \Pi_2 \cdots \times \Pi_M$; $\langle \pi_1,\, \pi_2 \cdots, \pi_M \rangle \longrightarrow_\Theta \langle \pi'_1,\, \pi'_2 \cdots, \pi'_M \rangle$ if and only if $\pi_i \longrightarrow_i \pi'_i$, $i = 1, 2, \cdots, M$; $\Pi_{\Theta,0} = \Pi_{1,0} \times \Pi_{2,0} \cdots \times \Pi_{M,0}$; $AP_\Theta = AP_1 \cup AP_2 \cdots \cup AP_M$; $L_\Theta(\langle \pi_1,\, \pi_2 \cdots, \pi_M \rangle) = L_1(\pi_1) \cup L_2(\pi_2) \cdots \cup L_M(\pi_M)$; $W_\Theta(\langle \pi_1,\, \pi_2 \cdots, \pi_M \rangle, \langle \pi'_1,\, \pi'_2 \cdots, \pi'_M \rangle) = W_1(\pi_1,\, \pi'_1) + \cdots + W_M(\pi_M,\, \pi'_M)$.

We denote the mutual specification by

$$\varphi_\Theta = \varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_M, \tag{4.4}$$

which is the conjunction of all individual task specifications. $\mathcal{A}_{\varphi_\Theta}$ is the NBA associated with φ_Θ. Then $\{\varphi_i, \forall i \in \Theta\}$ are called *mutually infeasible* if φ_Θ is infeasible over $\mathcal{T}_\Theta$ by Definition 2.6.

Problem 4.1. *Given the cluster Θ, if $\{\varphi_i, \forall i \in \Theta\}$ are **mutually infeasible**, how should the individual motion and task plan be synthesized such that the mutual specification is satisfied **the most**?*

Decentralized Coordination

Denote by $AP_{\varphi_\Theta} = AP_{\varphi_1} \cup AP_{\varphi_2} \cdots \cup AP_{\varphi_M}$ the set of all propositions appearing in φ_Θ by (4.4). Note that $AP_{\varphi_\Theta} \subseteq AP_\Theta \subset AP_{\mathcal{N}}$. Since φ_Θ is infeasible over $\mathcal{T}_\Theta$, we need to relax the requirement that every φ_i has to be fulfilled simultaneously. Thus we define the relaxed intersection of the individual specification automaton $\mathcal{A}_{\varphi_i}$.

Definition 4.4. *Given the M Büchi automata $\mathcal{A}_{\varphi_1}, \mathcal{A}_{\varphi_2} \cdots, \mathcal{A}_{\varphi_M}$, their relaxed intersection is given by $\tilde{\mathcal{A}}_{\varphi_\Theta} = (Q,\, 2^{AP_{\varphi_\Theta}},\, \delta,\, Q_0,\, \mathcal{F})$, where $Q = Q_1 \times \cdots \times Q_M \times \{1, 2 \cdots, M\}$; $Q_0 = Q_{1,0} \times \cdots \times Q_{M,0} \times \{1\}$; $\mathcal{F} = \mathcal{F}_1 \times Q_2 \cdots \times Q_M \times \{1\}$; $\delta : Q \to 2^Q$. $\langle q_1', \cdots, q_M', t'\rangle \in \delta(\langle q_1, \cdots, q_M, t\rangle)$ when: (1) $\langle q_1, q_2 \cdots, q_M, t\rangle, \langle q_1', q_2' \cdots, q_M', t'\rangle \in Q$; (2) $\exists\, l_i \in 2^{AP_{\varphi_\Theta}}$ such that $q_i' \in \delta_i(q_i, l_i)$, $\forall i \in \Theta$; (3) $q_t \notin \mathcal{F}_t$ and $t' = t$,* or *$q_t \in \mathcal{F}_t$ and $t' = \texttt{mod}(t, M) + 1$, where* `mod` *is the modulo operation.* ▲

The conventional definition of Büchi automaton intersection (Clarke et al., 1999) is obtained by replacing the second constraint "$\exists\, l_i \in 2^{AP_{\varphi_\Theta}}$ such that $q_i' \in \delta_i(q_i, l_i)$, $\forall i \in \Theta$" by "$\exists\, l \in 2^{AP_{\varphi_\Theta}}$ such that $q_i' \in \delta_i(q_i, l)$, $\forall i \in \Theta$". Namely, we relax the requirement that there should exist a common input alphabet that enable the transitions from q_i to q_i' in $\mathcal{A}_{\varphi_i}$, $\forall i \in \Theta$. The last component $t \in \{1, 2, \cdots, M\}$ in the state ensures that at least one accepting state of every $\mathcal{A}_{\varphi_i}$ is visited infinitely often.

Definition 4.5. *The relaxed product automaton $\mathcal{A}_{r,\Theta} = \mathcal{T}_\Theta \times \tilde{\mathcal{A}}_{\varphi_\Theta} = (Q', \delta', Q_0', \mathcal{F}', W_r)$ is defined as follows:*

- *$Q' = \Pi_\Theta \times Q = \{\langle \pi_\Theta, q\rangle \,|\, \forall \pi_\Theta \in \Pi_\Theta, \forall q \in Q\}$.*
- *$\delta' \subseteq Q' \times Q'$. $(\langle \pi_\Theta, q_a\rangle, \langle \pi_\Theta', q_b\rangle) \in \delta'$ iff $(\pi_\Theta, \pi_\Theta') \in \longrightarrow_\Theta$ and $q_b \in \delta(q_a)$.*
- *$Q_0' = \Pi_{\Theta,0} \times Q_0$ is the set of initial states. $\mathcal{F}' = \Pi_\Theta \times \mathcal{F}$ is the set of accepting states.*
- *$W_r : \delta' \to \mathbb{R}^+$ is the weight function, defined as*

$$\begin{aligned} &W_r(\langle \pi_\Theta, q_1, \cdots, q_M, t\rangle, \langle \pi_\Theta', q_1', \cdots, q_M', t'\rangle) \\ &= W_\Theta(\pi_\Theta, \pi_\Theta') + \alpha \sum_{i=1}^{M} \beta_i\, \texttt{Dist}(L_\Theta(\pi_\Theta),\, \chi_i(q_i, q_i')) \end{aligned} \tag{4.5}$$

where $\alpha, \beta_1, \beta_2 \cdots, \beta_M \geq 0$ *are design parameters; the function* `Dist`$(\cdot)$ *is defined in* (3.9)*;* $\langle \pi'_\Theta, q'_1, \cdots, q'_M, t' \rangle \in \delta'(\langle \pi_\Theta, q_1, \cdots, q_M, t \rangle)$*;* $\chi_i(q_i, q'_i) = \{l \in 2^{AP_\Theta} \,|\, q'_i \in \delta_i(q_i, l)\}$ *consists of all input alphabets that enable the transition from* q_i *to* q'_i *in* $\mathcal{A}_{\varphi_i}$*,* $\forall i \in \Theta$*.* ▲

Denote by $\beta = \{\beta_i, i \in \Theta\}$. As "$\exists l_i \in 2^{AP_{\varphi_\Theta}}$ such that $q'_i \in \delta_i(q_i, l_i)$, $\forall i \in \Theta$" by Definition 4.4, $\chi_i(q_i, q'_i) \neq \emptyset$. Figure 4.1 illustrates the relative distances between $L_\Theta(\pi_\Theta)$ and two sets of input alphabets χ_1, χ_2. The definition of W_r can be interpreted similarly as the one in (3.4). The first item represents the implementation cost of the transition from π_Θ to π'_Θ in $\mathcal{T}_\Theta$; α reflects the relative weighting between the implementation cost of the motion plan and how much the plan fulfills the mutual specification automaton $\mathcal{A}_{\varphi_\Theta}$; However, β plays the role as the "priority" index for each agent, i.e., the larger β_i is, the higher the priority agent i has. For example, if agent i has the highest priority with important tasks, β_i can be set very large such that the penalty of violating $\mathcal{A}_{\varphi_i}$ is severe. On the other hand, if it plays the role as an assisting robot, β_i can be chosen close to zero.

Problem 4.2. *Given the relaxed product automaton* $\mathcal{A}_{r,\Theta}$*, (i) find its balanced accepting run that* ***minimizes*** *the total cost by* (3.7)*; (ii) synthesize and execute the individual plan for each agent.*

Given the value of α and β, $\mathcal{A}_{r,\Theta}$ by Definition 4.5 can be either constructed fully or on-the-fly as described in Section 2.4. Then by following the same approach from Section 3.1, the balanced accepting run with the prefix-suffix structure and minimizes the total cost by (3.7) can be found by Algorithm 3. Denote by $R_{\text{bal},\Theta}$ the balanced accepting run for the cluster.

Remark 4.1. It is possible to split $W_\Theta(\pi_\Theta, \pi'_\Theta)$ in (4.5) into M parts, i.e., the implementation cost of each agent. Relative weighting among these costs can also be added in case of different power capacities among the agents. ▲

Synthesis and Execution of Individual Plan

First of all, agents within one cluster should agree upon the value of α according to the intended relative weighting between the implementation cost and the distance to the mutual tasks, and also the value of β based on their priorities within the cluster. In the absence of a central authority, α and β can either be determined by the designer prior to the deployment or

Algorithm 15: Feedback function for clusters, `ClusterFB( )`

```
**Input**: $R_{\text{bal},\Theta}$ of $\mathcal{A}_{r,\Theta}$; $\mathcal{T}_i$, $\mathcal{A}_{\varphi_i}$
**Output**: $\texttt{cost}_{\tau_i}$, $\texttt{dist}_{\varphi_i}$, $\mathcal{A}'_{\varphi_i}$
1. Initialization: $\mathcal{A}'_{\varphi_i} = \mathcal{A}_{\varphi_i}$, $\texttt{cost}_{\tau_i} = \texttt{dist}_{\varphi_i} = 0$.
2. For each $(q'_j,\, q'_{j+1}) \in \texttt{Edge}(R_{\text{bal},\Theta})$, perform steps 3-5 as follows:
3. Let $q'_j = \langle \pi_\Theta,\, q_1,\, \cdots, q_M, t\rangle$ and $q'_{j+1} = \langle \pi'_\Theta,\, q'_1,\, \cdots, q'_M, t'\rangle$.
4. Project $(\pi_\Theta,\, \pi'_\Theta)$ onto $\mathcal{T}_i$ and save the projection $(\pi_{i,\Theta},\, \pi'_{i,\Theta})$ in $\tau_i$.
$\texttt{cost}_{\tau_i} = \texttt{cost}_{\tau_i} + W_i(\pi_{i,\Theta},\, \pi'_{i,\Theta})$;
5. $d = \texttt{CheckTranR}(q_i,\, L_\Theta(\pi_\Theta),\, q'_i, \mathcal{A}_{\varphi_i})$. If $d > 0$, add $q'_i$ to
$\delta_i(q_i,\, L_\Theta(\pi_\Theta)|_{AP_i})$ of $\mathcal{A}'_{\varphi_i}$. $\texttt{dist}_{\varphi_i} = \texttt{dist}_{\varphi_i} + d$.
**return** $\texttt{cost}_{\tau_i}$, $\texttt{dist}_{\varphi_i}$, $\mathcal{A}'_{\varphi_i}$
```

a consensus algorithm on the value of α and β within the cluster might be needed. Many distributed consensus algorithms can be applied, e.g., Ren et al. (2005); Olfati-Saber and Shamma (2005); Cortés (2008).

Then Algorithm 3 is called to generate the balanced accepting run $R_{\text{bal},\Theta}$. The cooperative motion plan τ_Θ is the projection of $R_{\text{bal},\Theta}$ onto $\mathcal{T}_\Theta$. Then Algorithm 15 is used to interpret $R_{\text{bal},\Theta}$ of $\mathcal{A}_{r,\Theta}$ for each agent i: (i) its individual motion plan τ_i, as the projection of τ_Θ onto Π_i; (ii) the associated revised specification automaton $\mathcal{A}'_{\varphi_i}$, obtained by adding new transitions to $\mathcal{A}_{\varphi_i}$; (iii) the implementation cost of τ_i as $\texttt{cost}_{\tau_i}$; (iv) the accumulated distance of τ_Θ to its original task specification φ_i as $\texttt{dist}_{\varphi_i}$, defined similarly as in (2.10). As an extension, Algorithm 3 could be applied under different α and β to derive several balanced accepting run candidates, of which the unique ones are saved. Then Algorithm 15 gives feedback about their implementation cost and their distances to individual specifications.

Lemma 4.1. *Assume τ_Θ and $\texttt{dist}_{\varphi_i}$ are the derived from Algorithm 15. Then $\texttt{dist}_{\varphi_i} = 0$ implies that $\tau_\Theta|_{\Pi_i}$* ***satisfies*** *φ_i.*

Proof. The proof is omitted as it is similar to that of Lemma 3.1. ■

Example 4.1. An example of a two-agent system is shown in Figures 4.2 and 4.3. Agent 1 needs to visit π_1 and π_2 infinitely often, but never be at π_1 with agent 2 at the same time. Agent 2 needs to visit π_1 and stay there. Six different motion plans are obtained by Algorithms 3 and 15 under different α and β, as in Figure 4.4. The same color indicates that the same balanced accepting run is found. Here we list two motion plan candidates:

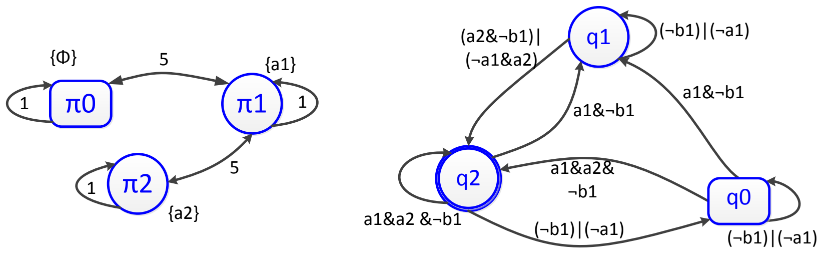


Figure 4.2: Agent 1's model and task: its FTS $\mathcal{T}_1$ (left) and NBA $\mathcal{A}_{\varphi_1}$ associated with $\varphi_1 = (\Box\Diamond a_1) \wedge (\Box\Diamond a_2) \wedge (\Box\neg(a_1 \wedge b_1))$ (right).

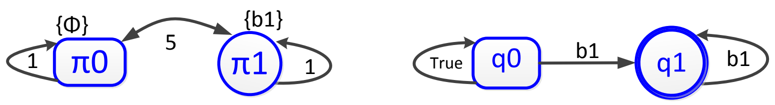


Figure 4.3: Agent 2's model and task: its FTS $\mathcal{T}_2$ (left) and NBA $\mathcal{A}_{\varphi_2}$ associated with $\varphi_2 = \Diamond\Box\, b_1$ (right).

(i) agent 1: $\pi_0\pi_1[\pi_2]^\omega$, agent 2: $\pi_0\pi_0[\pi_1]^\omega$ (which has $\texttt{dist}_{\varphi_1}$ 2, $\texttt{dist}_{\varphi_2}$ 0, $\texttt{cost}_{\mathcal{T}_1}$ 12, $\texttt{cost}_{\mathcal{T}_2}$ 8); (ii) agent 1: $\pi_0\pi_1[\pi_2\pi_1\pi_2]^\omega$, agent 2: $\pi_0\pi_0[\pi_1\pi_0\pi_1]^\omega$ (which has $\texttt{dist}_{\varphi_1}$ 0, $\texttt{dist}_{\varphi_2}$ 1, $\texttt{cost}_{\mathcal{T}_1}$ 20, $\texttt{cost}_{\mathcal{T}_2}$ 16). ▲

The above approach is applied to any other clusters within the multi-agent system. Particularly, (i) all agents need to confirm their dependency relation, i.e., which cluster they belong to; (ii) within each cluster an agreement on α and β should be achieved; (iii) Algorithm 3 is called to derive the balanced accepting run $R_{\text{bal},\Theta}$; (iv) each agent computes its individual plan by $R_{\text{bal},\Theta}|_{\Pi_i}$; (v) all agents within one cluster implements their motion plans in a synchronized way (Kloetzer and Belta, 2010).

Remark 4.2. This multi-agent framework can be applied to the single-agent case where the specification has the "conjunction" form $\varphi = \varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_N$. Then the sub-specification φ_i can be modeled as the individual specification of an "imaginary" agent which has identical movements as the "real" agent; β could represent different priorities among these sub-tasks. ▲

Let $|\mathcal{T}_i|$ and $|\mathcal{A}_{\varphi_i}|$ denote the size of agent i's FTS and the NBA. The size of $\mathcal{A}_{r,\Theta}$ by Definition 4.5 for one cluster with M members is $|\mathcal{A}_{r,\Theta}| =$

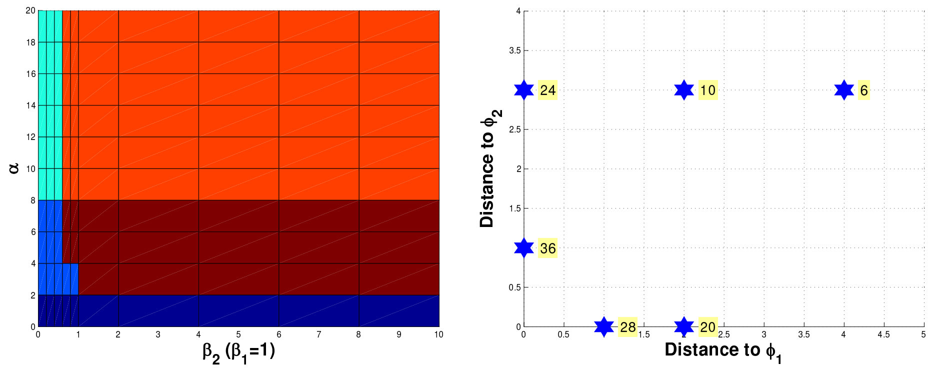


Figure 4.4: Left: the balanced accepting runs generated under different α and β ($\gamma = 5$). Right: the balanced plan candidates located by their distance to φ_1 and distance to φ_2 (y-axis), and labeled by the implementation cost.

$M \cdot \prod_{i=1}^{M} |\mathcal{T}_i| \cdot |\mathcal{A}_{\varphi_i}|$. Algorithm 3 runs in $\mathcal{O}(|\mathcal{A}_{r,\Theta}| \log |\mathcal{A}_{r,\Theta}| \cdot (|Q_0'| + |\mathcal{F}'|))$. Algorithm 15 have the complexity linear to the length of $R_{\text{bal},\Theta}$.

Case Study

Consider a team of four unicycle robots that satisfy: $\dot{x}_i = v_i \cos\theta_i$, $\dot{y}_i = v_i \sin\theta_i$, $\dot{\theta}_i = \omega_i$, where $\mathbf{p}_i = (x_i,\, y_i)^T \in \mathbb{R}^2$ is agent i's position; $\theta_i \in [0,\, 2\pi]$ is the orientation; and $v_i,\, \omega_i \in \mathbb{R}$ are the transition and rotation velocities, $i = 1,\, 2,\, 3,\, 4$. The workspace is shown in Figure 4.5, which consists of 26 polygonal regions. The continuous controller that drives the robots from an region to any geometrically adjacent region is based on Lindemann et al. (2006) by constructing vector fields over each triangular cell for each face. The controller design is omitted for brevity.

Task Specifications

Robots 2, 3 and 4 are confined in rooms 2, 3 and 4 as shown in Figure 4.5. Each room has six regions, some of which are obstacle-occupied (in gray). They repetitively carry different goods from the storage region to the unloading region within each room, while avoiding obstacles. After picking up goods at the storage region, they have to drop the goods at unloading region before they return to the storage region. The storage, unloading and obstacle-occupied regions are labeled by $a_{i,s}$, $a_{i,u}$ and $a_{i,o}$ respectively for

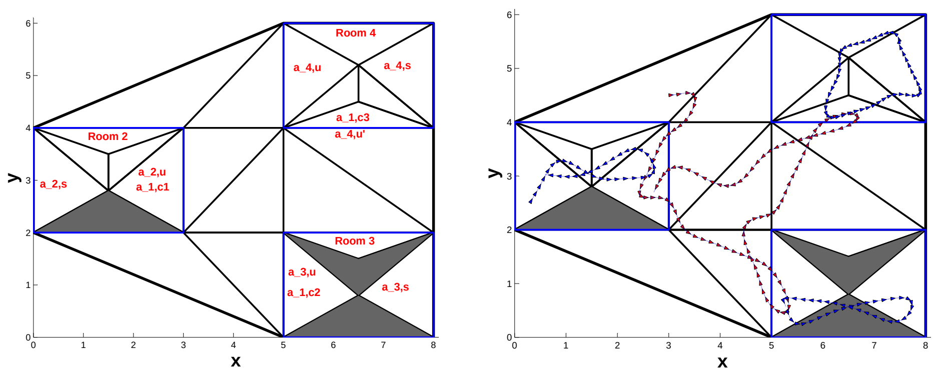


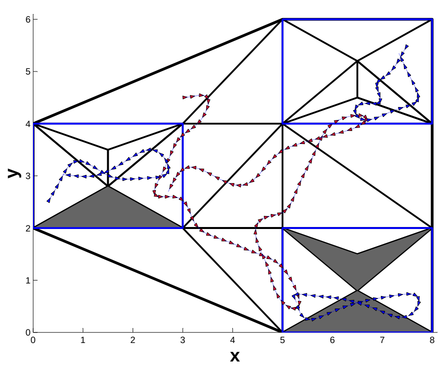
Figure 4.5: Left: the workspace model, where blue boxes indicate the confined rooms for robots 2, 3 and 4; Right: both φ_1 and φ_2 are fulfilled (corresponds to P1), and robot 3 chooses the plan that violates φ_3 the least.

agent $i = 2, 3, 4$. Robot 1 has to collect these goods at the regions labeled by $a_{1,c1}$, $a_{1,c2}$ and $a_{1,c3}$ repetitively. In addition, robot 4 needs to meet robot 1 at region labeled by $a_{4,u'}$. The obstacle-occupied regions for agent 1 are labeled by $a_{1,o}$. These tasks are specified as LTL formulas by:

robot 1: $\varphi_1 = \Box\Diamond(a_{1,c1}) \wedge \Box\Diamond(a_{1,c2}) \wedge \Box\Diamond(a_{1,c3} \wedge a_{4,u'}) \wedge \Box(\neg a_{1,o})$

robot i: $\varphi_i = \Box\Diamond a_{i,s} \wedge \Box\Diamond a_{i,u} \wedge \Box(a_{i,s} \Rightarrow \bigcirc(\neg a_{i,s} \,\mathsf{U}\, a_{i,u})) \wedge \Box(\neg a_{i,o})$, $i = 2, 3, 4,$, where $a_{i,s}$ and $a_{i,u}$ stand for "robot i is at its storage region and unloading region, respectively", $i = 2, 3, 4$; $a_{1,c1}$, $a_{1,c2}$, $a_{1,c3}$ indicate "robot 1 is at collecting regions $c1$, $c2$ and $c3$ respectively"; $a_{i,o}$ indicated "robot i is at obstacle-occupied regions", $i = 1, 2, 3, 4$; $a_{4,u'}$ indicates "the robot 4 is at the unloading region that robot 1 knows". These regions of interest are labeled by the atomic propositions that are true in Fig. 4.5.

Dependency and potential conflicts: by Definition 4.1, robots 1 and 4 are dependent while robots 2 and 3 run independently. There is a misunderstanding between robots 1 and 4 about the location of robot 4's unloading region, namely, $a_{4,u'}$ and $a_{4,u}$ indicate two different regions, as shown in Room 4 of Figure 4.5. But this does not necessarily mean that φ_1 and φ_4 are mutually infeasible. Moreover, φ_3 is infeasible for agent 3 because of the obstacles in room 3. Each robot can transit between any two geometrically adjacent regions within their confined workspace, of which the costs are uniformly 5. They could also stay at any region with the cost 1.



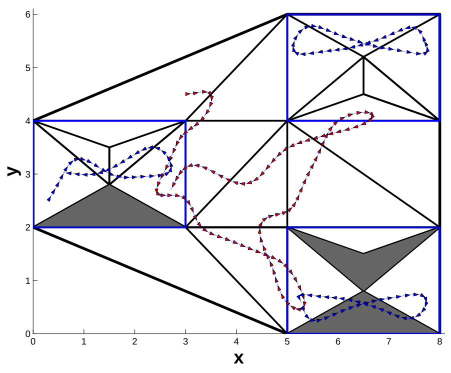


Figure 4.6: Left: robots 1 and 4 meet at $a_{1,u'}$ and φ_1 is fulfilled but not φ_4 (corresponds to P2). Right: robots 1 and 4 do not meet while φ_4 is fulfilled but not φ_1 (corresponds to P3).

$\mathcal{T}_1$ has 13 states while $\mathcal{T}_i$ has 6 states; $\mathcal{A}_{\varphi_1}$ has 4 states and $\mathcal{A}_{\varphi_i}$ has 5 states by Gastin and Oddoux (2001), $i = 2, 3, 4$.

Simulation Results

Algorithm 15 is applied to the cluster formed by robots 1 and 4. The composed transition system $\mathcal{T}_\Theta$ has 78 states. The relaxed product automaton $\mathcal{A}_{r,\Theta}$ consists of 3120 states and 1364 edges, which has three weighting parameters α, β_1 and β_2. By choosing $\alpha = 0, 20, 100$; $\beta_1 = 1$; $\beta_2 = 0, 0.5, 1, 10$, six unique motion plan candidates are found. Here we choose three of them: (P1) $\alpha = 100$, $\beta_1, \beta_2 = 1$. Robot 4 travels more distance from its unloading region to meet robot 1 at the collecting region ($\texttt{dist}_{\varphi_1}$ 0, $\texttt{dist}_{\varphi_4}$ 0, $\texttt{cost}_{\mathcal{T}_1}$ 140, $\texttt{cost}_{\mathcal{T}_4}$ 48); (P2) $\alpha = 100$, $\beta_1 = 1$, $\beta_2 = 0$. Robots 1 and 4 meet at robot 1's collecting region ($\texttt{dist}_{\varphi_1}$ 0, $\texttt{dist}_{\varphi_4}$ 8, $\texttt{cost}_{\mathcal{T}_1}$ 140, $\texttt{cost}_{\mathcal{T}_4}$ 21); (P3) $\alpha = 30$, $\beta_1, \beta_2 = 1$. Robots 1 and 4 do not meet ($\texttt{dist}_{\varphi_1}$ 2, $\texttt{dist}_{\varphi_4}$ 0, $\texttt{cost}_{\mathcal{T}_1}$ 126, $\texttt{cost}_{\mathcal{T}_4}$ 20). On the other hand, Algorithm 8 is applied for robot 3 to find the motion plan that violates φ_3 the least. We choose the motion plan under $\alpha = 2$, of which the implementation cost is 30 and the distance to φ_3 is 3. In particular, Figures 4.5 and 4.6 present the final motion of the composed system when the above motion plans are implemented by the lower-level hybrid controllers.

4.2 Independent Local Tasks

If the individual task only relies on local propositions, the team of robots are independent, i.e., their plans can be synthesized and executed independently from each other. However since the agents are usually located at various locations within the workspace, providing them access to the up-to-date knowledge about the actual workspace. Those local knowledge possessed by individual agent, if shared among the team, might benefit each member such that they can make better and more informed plans.

Partially-known Workspace

We consider a team of autonomous agents with unique identities (IDs) $i \in \mathcal{N} = \{1, 2 \cdots, N\}$, which coexist within the common but partially-known workspace $\mathcal{W}$. Agent i's possible motion within $\mathcal{W}$ is abstracted as a weighted finite transition system (wFTS):

$$\mathcal{T}_i^t = (\Pi_i,\, \longrightarrow_i^t,\, \Pi_{i,0},\, AP_i,\, L_i^t,\, W_i^t), \tag{4.6}$$

the components of which are defined in a similar way as (3.16). The superscript $t \geq 0$ indicates that the workspace is partially known and might be updated after the system starts. Each agent i 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 for agent i as (3.12).

Initial Synthesis

At $t = 0$, for each agent $i \in \mathcal{N}$, the initial motion and task plan can be obtained as the safe plan by following the framework proposed in Section 3.2, regarding the initial transition system $\mathcal{T}_i^0$ and φ_i. We assume the on-the-fly construction based on Algorithms 4 and 10 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 agent i at time t; R_i^t, τ_i^t as the derived balanced accepting run and the associated plan.

Cooperative Knowledge Transfer

In Section 3.3, we describe how a single agent could update its transition system through its sensing ability to observe the actual workspace and its communication ability to inquire and retrieve knowledge from external sources. Belonging to the same multi-agent system, the other agents could be the external source. In other words, they could share and transfer their knowledge about the workspace collectively in real-time. In this section, we explain in detail how to design the knowledge transfer protocol and how to combine it with the real-time motion planning and revising scheme from Section 3.3.

Communication Network

The communication network represents how the information flows among the agents. Each agent i has a set of neighboring agents, denoted by $\mathcal{N}_i \subseteq \mathcal{N}$. Agent i can send messages directly to any agent 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 agent has a communication range, denoted by $C_i \geq 0$. Agent i can only send messages to agent 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 agents 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 agent i at time t.

Communication Protocol

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

$$\mathbf{Request}_{j,i}^t = (j,\, \varphi_j|_{AP_j},\, i), \tag{4.7}$$

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

The sensing update of agent i is denoted by $\mathbf{Sense}_i^t$, the structure of which is given in (3.17). Then the *publishing* phase of each agent follows an event-driven approach: whenever $(\pi,\, S,\, S_\neg) \in \mathbf{Sense}_i^t$ is obtained, it

Algorithm 16: Transfer Knowledge to other agents, `TranKnow( )`

```
  Input: $\mathbf{Sense}_i^t$, $\mathbf{Request}_{j,i}^{t_0}$
  Output: $\mathbf{Reply}_{i,j}^t$
1 forall $(\pi, S, S_\neg) \in \mathbf{Sense}_i^t$ do
2     forall $\mathbf{Request}_{j,i}^{t_0}$ received at $t_0 < t$ do
3         $\mathbf{Request}_{j,i}^t = (j, \varphi_j|_{AP_j}, i)$
4         if $j \in \mathcal{N}_i^t$ then
5             $S' = S \cap (\varphi_j|_{AP_j})$
6             $S'_\neg = S_\neg \cap (\varphi_j|_{AP_j})$
7             if $S' \neq \emptyset$ or $S'_\neg \neq \emptyset$ then
8                 add $(\pi, S', S'_\neg)$ to $\mathbf{Reply}_{i,j}^t$
9 return $\mathbf{Reply}_{i,j}^t$
```

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 agent j regarding some propositions in in $\varphi_j|_{AP_j}$, then agent i checks if $j \in \mathcal{N}_i^t$. If so, it publishes a reply message to agent j that

$$\mathbf{Reply}_{i,j}^t = \{(\pi, S', S'_\neg)\}, \tag{4.8}$$

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 agent's task φ, every reply message should contain useful knowledge. The above procedure is summarized in Algorithm 16. 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.

Real-time Verification and Reconfiguration

Upon receiving $\mathbf{Sense}_i^t$ and $\mathbf{Reply}_{j,i}^t$ from $j \in \mathcal{N}_i^t$, agent i could update its transition system $\mathcal{T}_i$ by Algorithm 11. Regarding its current motion and task plan τ_i^t, the validity and safety of τ_i^t needs to be verified by Algorithm 12. Moreover, in case τ_i^t is falsified, i.e., either invalid or unsafe, Algorithm 14 is called to revise τ_i^t such that it becomes valid and safe, where $\widetilde{\Pi}_i^t$, $\aleph_i^t$, Ξ_i^t

Algorithm 17: Cooperative on-line planning for each agent $i \in \mathcal{N}$

```
Input: $\mathcal{T}_i^0$, $\tilde{\mathcal{A}}_{\varphi_i}$, $\tilde{\mathcal{A}}_{r,i}^0$, $x_i(t)$
Output: $R_i^t$, $\tau_i^t$, $\Upsilon_i^t$, $T_i^t$
1  $R_i^0 = \texttt{OptRun}(\tilde{\mathcal{A}}_{r,i}^0)$;                                   // Algorithm 3
2  $q'_{i,\text{cur}} = q'_{i,\text{next}} = R^0_{i,[\text{pre},1]}$, $\pi_{i,\text{next}} = q'_{i,\text{next}}|_{\Pi_i}$, $R_{i,\text{past}} = [\,]$, $\tau_{i,\text{past}} = [\,]$
3  while True do
4      send $\textbf{Request}_{i,g}^t$
5      check $\textbf{Reply}_{h,i}^t$, $\textbf{Request}_{j,i}^{t_0}$ and $\textbf{Sense}_i^t$
6      $\textbf{Reply}_{i,j}^t = \texttt{TranKnow}(\textbf{Sense}_i^t, \textbf{Request}_{j,i}^{t_0})$;   // Algorithm 16
7      send $\textbf{Reply}_{i,j}^t$
8      $(\mathcal{T}_i^{t^+}, \widetilde{\Pi}_i^t, \widehat{\Pi}_i^t) = \texttt{UpdaT}(\mathcal{T}_i^t, \textbf{Sense}_i^t, \textbf{Reply}_{h,i}^t)$ ; // Algorithm 11
9      $\tilde{\mathcal{A}}_{r,i}^{t^+} = \texttt{AdjProd}(\mathcal{T}_i^{t^+}, \widehat{\Pi}_i^t, \tilde{\mathcal{A}}_{\varphi_i})$;               // Algorithm 4
10     $(\aleph_i^t,\ \Xi_i^t) = \texttt{ValidRun}(\tilde{\mathcal{A}}_{r,i}^{t^+}, R_i^t, \widetilde{\Pi}_i^t, E_\neg)$ ;        // Algorithm 12
11     $\Upsilon_i^t = \Upsilon_i^t + |\widehat{\Pi}_i^t|$
12     $Q'_{i,\tau_\text{past}} = \texttt{CorProd}(\tilde{\mathcal{A}}_{r,i}^{t^+}, \tau_{i,\text{past}})$;                   // Algorithm 13
13     if $\Upsilon_i^t \geq N_i^\text{call}$ or $t - T_i^t \geq T_i^\text{call}$ then
14         $R_i^{t^+} = \texttt{OptRun}(\tilde{\mathcal{A}}_{r,i}^{t^+}, Q'_{i,\tau_\text{past}})$;                     // Algorithm 3
15         $\Upsilon_i^t = 0$, $T_i^t = t$, $q'_{i,\text{next}} = R^{t^+}_{i,[\text{pre},2]}$
16     else if $\aleph_i^t \neq \emptyset$ or $\Xi_i^t \neq \emptyset$ then
17         $R_i^{t^+} = \texttt{Revise}(\tilde{\mathcal{A}}_{r,i}^{t^+}, R_i^t, \aleph_i^t, \Xi_i^t, Q'_{i,\tau_\text{past}})$;      // Algorithm 14
18         $q'_{i,\text{next}} = R^{t^+}_{i,[seg,k]}$
19     if $x(t) \in q'_{i,\text{next}}|_{\Pi_i}$ confirmed then
20         $q'_{i,\text{cur}} = q'_{i,\text{next}}$
21         $\tau_{i,\text{past}} = \tau_{i,\text{past}} + \pi_{i,\text{next}},\ R_{i,\text{past}} = R_{i,\text{past}} + q'_{i,\text{next}}$
22         $q'_{i,\text{next}} = R^{t^+}_{i,[seg,k]} = \texttt{NextGoal}(q'_{i,\text{cur}}, R_i^{t^+})$;     // Algorithm 5
23     $\pi_{i,\text{cur}} = q'_{i,\text{cur}}|_{\Pi_i},\ \pi_{i,\text{next}} = q'_{i,\text{next}}|_{\Pi_i}$
24     $u_i = \mathcal{U}(x_i(t), \pi_{i,\text{cur}}, \pi_{i,\text{next}})$
```

stand for the set of regions in $\mathcal{T}_i^t$ with modified labeling functions at time t, the set of unsafe transitions and invalid transitions in R_i^t.
Algorithm 14 provides a way to locally revise the invalid or unsafe plan, which guarantees validity and safety by Theorem 3.3. However it does

not maintain the cost optimality of the safe accepting run compared with Algorithm 3. 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 Chan and Yang (2009) and Misra and Oommen (2005). 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 (Heemels et al., 2012) to ensure the optimality check. Denote by Υ_i^t the accumulated number of number of changes of $\mathcal{T}_i^t$ 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 is called. Let the thresholds N_i^{call}, $T_i^{\text{call}} \geq 0$ be chosen freely by each agent. Then whenever at least one of the following conditions holds: (1) $\Upsilon_i^t \geq N_i^{\text{call}}$; (2) $t - T_i \geq T_i^{\text{call}}$, Algorithm 3 is called with respect the latest $\mathcal{T}_i^t$ to derive the safe plan, but using $Q'_{\tau_{\text{past}}}$ as the set of initial states. Afterwards, $\Upsilon_i^{t^+}$ is reset to zero and T_i to the current time.

Overall Structure

The overall architecture is summarized in Algorithm 17. When the system starts, each agent synthesizes its own initial motion and task plan. It sends requests to neighboring agents. 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 automatically whenever: (i) the accepting run R_i^t is updated by either the revision Algorithm 14 and the optimal search Algorithm 3; (ii) the current goal region is confirmed to be reached.

Theorem 4.1. *For each agent i at any time $t \geq 0$, its motion plan τ_i^t derived by Algorithm 17 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^tcorresponds to the safe accepting run.*

Proof. The first part follows directly from Theorems 3.4 and 3.5. Moreover τ_i^t is the safe plan for $\tilde{\mathcal{A}}_{r,i}^{t^+}$ given $Q'_{i,\tau_{\text{past}}}$ whenever Algorithm 3 is called. Due to the triggering condition, it is called at least once within any time period with length T_i^{call}, which completes the proof. ∎

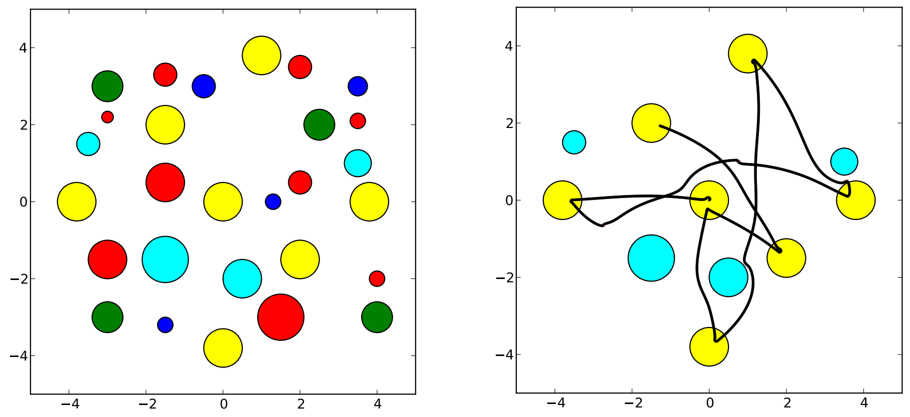


Figure 4.7: Left: the actual workspace as described in Section 4.2. Right: the final motion plan for aerial vehicles, which corresponds to the final optimal plan in Fig. 4.9. It satisfies both $\varphi^{\text{hard}}_{\text{Ar\_i}}$ and $\varphi^{\text{soft}}_{\text{Ar\_i}}$, since all base stations (in yellow) are surveiled and non-fly areas (in cyan) are avoided.

Case Study

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 and Agent Description

As shown Fig. 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`",$\cdots$,"`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.

Five aerial vehicles (denoted by Ar_1,$\cdots$, Ar_5) start randomly from one of the base stations. For aerial vehicles, "`b1`",$\cdots$, "`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

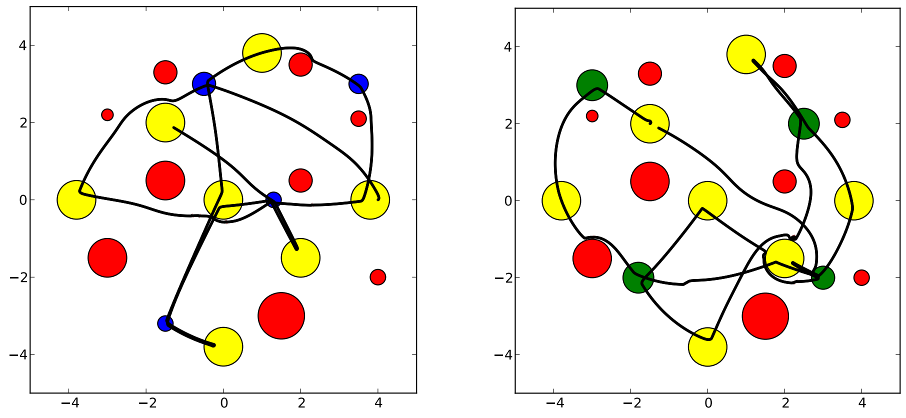


Figure 4.8: The final motion plan for agents Gw_i (left) and for agents Gf_i (right). It is shown that water (in blue) or food (in green) are fetched before any base station (in yellow) and all base stations are supplied infinitely often, while obstacle (in red) avoidance is always ensured.

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

$$\begin{aligned} \varphi^{\text{hard}}_{\mathsf{Ar\_i}} &= (\Box\neg\,\texttt{nfly})\,\wedge\,(\Box\Diamond(\varphi_{\texttt{one}})), \\ \varphi^{\text{soft}}_{\mathsf{Ar\_i}} &= (\Box(\Diamond\texttt{b1}\wedge\Diamond\texttt{b2}\wedge\cdots\wedge\Diamond\texttt{b7})), \end{aligned} \tag{4.9}$$

where $\varphi_{\texttt{one}} \triangleq \texttt{b1}\vee\texttt{b2}\vee\cdots\vee\texttt{b7}$ and $i = 1, \cdots, 5$. The NBA associated with $\varphi^{\text{hard}}_{\mathsf{Ar\_i}}$ has 2 states and 4 edges by Gastin and Oddoux (2001), while the one with $\varphi^{\text{soft}}_{\mathsf{Ar\_i}}$ has 8 states and 43 edges.

The other ten agents are ground vehicles: five of them (denoted by Gf_1,$\cdots$, Gf_5) collect food and the rest (denoted by Gw_1, $\cdots$, Gw_5) for water, to supply the base stations. For ground vehicles, "`b1`", $\cdots$, "`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 $\mathsf{Gw\_i}$ are given by

$$\begin{aligned}\varphi_{\mathsf{Gw\_i}}^{\text{hard}} &= (\Box\Diamond\neg\texttt{obs}) \wedge \varphi_{\texttt{order}}\\ \varphi_{\mathsf{Gw\_i}}^{\text{soft}} &= (\Box(\Diamond\texttt{b1} \wedge \Diamond\texttt{b2} \wedge \cdots \wedge \Diamond\texttt{b7})),\end{aligned} \tag{4.10}$$

where $\varphi_{\texttt{order}} \triangleq (\Box\Diamond\,\texttt{water}) \wedge (\Box(\texttt{water} \Rightarrow \bigcirc(\neg\texttt{water}\,\mathsf{U}\,(\varphi_{\texttt{one}})))$ $\wedge\,(\Box((\varphi_{\texttt{one}}) \Rightarrow \bigcirc(\neg(\varphi_{\texttt{one}})\,\mathsf{U}\,\texttt{water})),$ which says that water must be fetched and supplied to at least one base station infinitely often. $\varphi_{\mathsf{Gw\_i}}^{\text{soft}}$ is the same as for $\mathsf{Ar\_i}$, requiring that all base stations be supplied infinitely often. The NBA associated with $\varphi_{\mathsf{Gw\_i}}^{\text{hard}}$ have 10 states and 30 edges by Gastin and Oddoux (2001), while the one for $\varphi_{\mathsf{Gw\_i}}^{\text{soft}}$ has 8 states and 43 edges. The soft and hard specifications $\varphi_{\mathsf{Gf\_i}}^{\text{hard}}$ and $\varphi_{\mathsf{Gf\_i}}^{\text{soft}}$ for $\mathsf{Gf\_i}$ can be defined in a similar way by replacing proposition "`water`" with "`food`".

Clearly, for each agent 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`",$\cdots$, "`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 agent has a communication radius (set to $5m$ for all agents).

Simulation Results

Initially each agent 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 agents. Fig. 4.7 and 4.8 show the trajectories corresponding to the suffix part of the optimal run, for the three groups $\mathsf{Ar\_i}$, $\mathsf{Gf\_i}$, $\mathsf{Gw\_i}$. It can be seen that both soft and hard specifications are fulfilled by all agents.

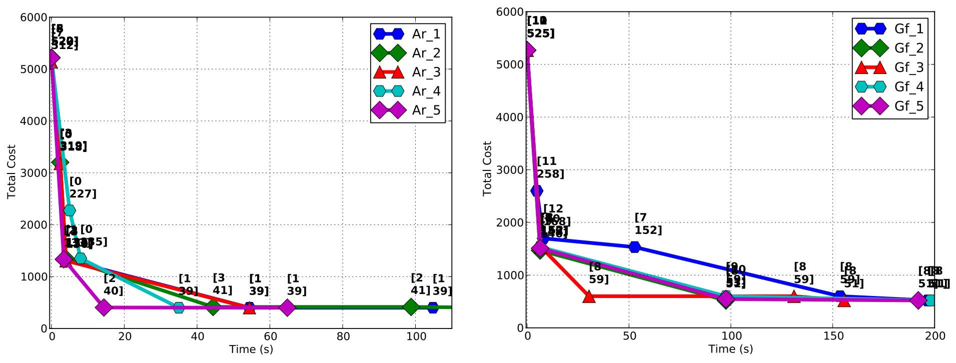


Figure 4.9: The evolution of the plan total cost of agents Ar_i (left) and Gf_i (right). For Ar_1, the initial plan has a total cost 5220 ([5, 521]), when it only knows two base stations. Its final plan has a total cost 391 ([1, 39]), as shown in Fig. 4.7. For Gf_1, the initial plan has a total cost 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 Fig. 4.8.

Fig. 4.9 and 4.10 shows how the optimal plans of Ar_i, Gw_i and Gf_i evolve with time. In particular, under the proposed scheme, the total cost of the optimal plan for each agent 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. Fig. 4.10 illustrates the number of messages received by each agent under the dynamic communication topology. Compared with the synchronized solution that requires information exchange at each time step Ding et al. (2011a), Karaman and Frazzoli (2008), the messages are only sent and received following the subscriber-publisher scheme.

Experiments

To further testify the proposed framework, three nexus ground vehicles are deployed in the Smart Mobility Lab, Royal Institute of Technology KTH, as shown in Fig. 4.11. Similar task specifications are designed as in Section 4.1. One vehicle $\mathcal{R}_1$ has the surveillance task over four base stations and avoids all obstacles, as specified by (4.9). One vehicle $\mathcal{R}_2$ needs to visit the blue

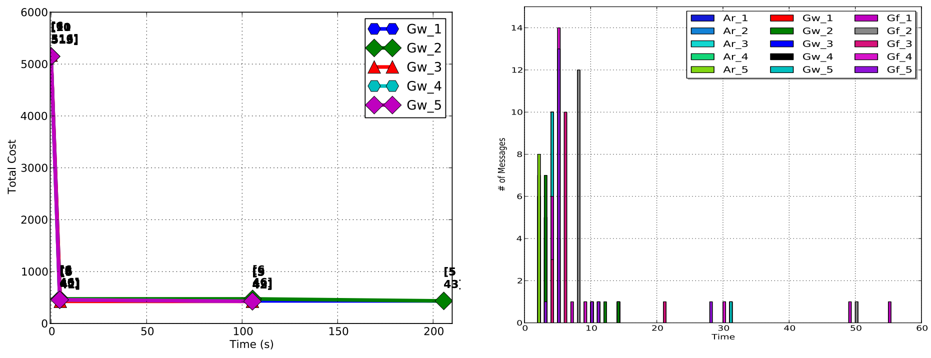
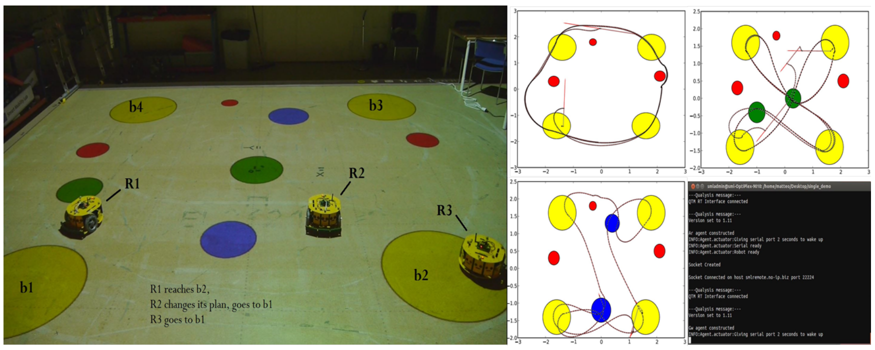


Figure 4.10: Left: the evolution of the plan total cost of agents Gw_i. For agent Gw_1, the initial plan has a total cost around 5147 ([11, 513]), when it only knows the location of one base station and one water resource. Its final plan has a total cost 456 ([6, 45]), as shown in Fig. 4.8. Right: the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

regions and all base stations infinitely often in an interleaved order while avoiding all obstacles, as specified by (4.10). One vehicle $\mathcal{R}_3$ needs to visit the green regions and all base stations in a similar manner. The real-time feedback of vehicles' position and orientation is obtained from the indoor motion capture system "Qualisys". Obstacles are detected by the on-board sonar sensors. Wireless communication between the control PC and vehicles are handled by APC220 Radio Communication Modules.

In detail, the lab workspace is a $6 \times 6m^2$ and modelled by the circle $\mathcal{B}_3([0, 0])$. Four base stations are located at the center and four corners; two blue regions, two green regions, three obstacles regions are scattered in the workspace (as shown in Fig. 4.11). Initially $\mathcal{R}_1$ 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. The experiment results are recorded in real-time (Guo and Dimarogonas, 2014b). Fig 4.11 shows the workspace and the recorded panels in the experiment.


b4
b3
R1
R2
R3
b1
b2
R1 reaches b2,
R2 changes its plan, goes to b1
R3 goes to b1
smladmin@sml-OptiPlex-9010: /home/matteo/Desktop/single_demo

```
---Qualysis message:---
QTM RT Interface connected

---Qualysis message:---
Version set to 1.11

Ar agent constructed
INFO:Agent.actuator:Giving serial port 2 seconds to wake up
INFO:Agent.actuator:Serial ready
INFO:Agent.actuator:Robot ready

Socket Created

Socket Connected on host smlremote.no-ip.biz port 22224

---Qualysis message:---
QTM RT Interface connected

---Qualysis message:---
Version set to 1.11

Gw agent constructed
INFO:Agent.actuator:Giving serial port 2 seconds to wake up
```

Figure 4.11: Left: the workspace used in the experiment, with three Nexus vehicles. Right: the recorded panels, including video streams, vehicles' trajectories and the log window.

4.3 ROS Implementation

There are two main model-checking-based motion planning software packages: Linear Temporal Logic Motion Planner (LTLMoP) by Finucane et al. (2010) and LTL Robust Multi-Robot Planner (LROMP) by Ulusoy et al. (2012). However since both are simulation-oriented, it is not straightforward to use them to control physical robots that interact with actual workspace and communicate with external sources.

Robot operating system (ROS) is currently one of the most popular operating systems for a large variety of robotic platforms. "ROS is a flexible framework for writing robot software. 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" (ROSWiki, 2013). It keeps a minimum and simple interfaces between different functional packages, which is beneficial for the purpose of reuse, modification and integration of packages designed for different robotic applications.

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, our planning algorithm can be running to revise, update and improve the plan while the robot is moving to the next goal region, in contrary to the blocking scenario where the robot can only move after it has finished the planning phase.

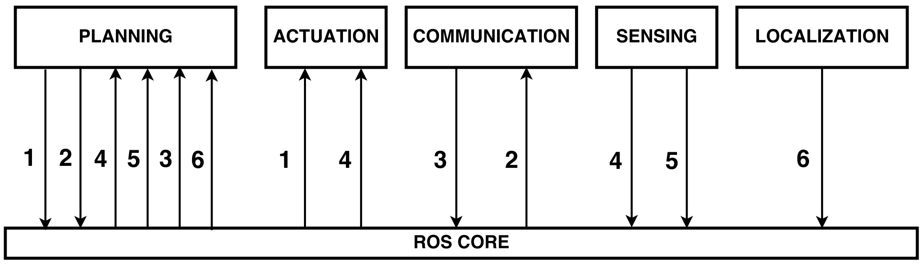


Figure 4.12: Architecture for our ROS-based implementation. Name of the topics: 1. next goal region or next action; 2. request to external source; 3. reply from communication; 4. observation from sensing; 5. confirmation for motion or action; 6. robot's position.

Package Architecture

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 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 our 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 Fig. 4.12. 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

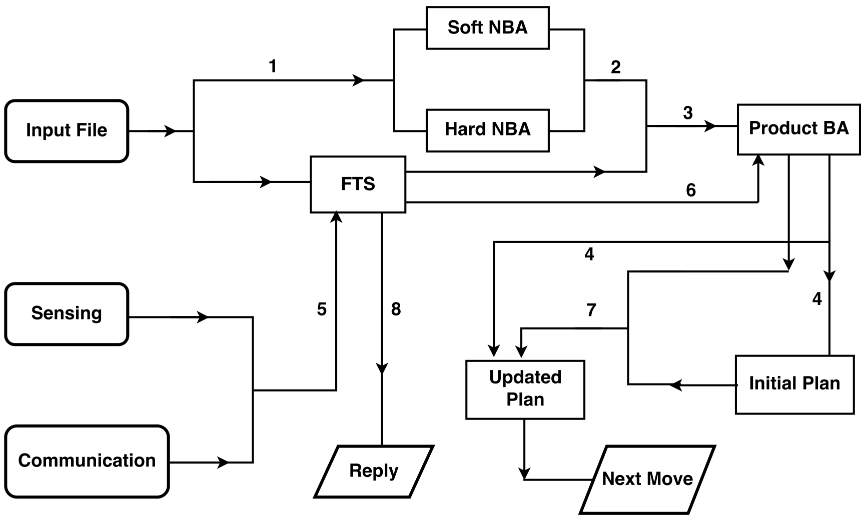


Figure 4.13: Data flow within the ROS node for planning. Input data is in rounded rectangular; internal variables are in rectangular; output data is in parallelogram. Texts on the arrows are algorithms: 1. LTL formulas to Büchi automaton and encodings; 2. relaxed automaton intersection by Definition 3.3; 3. relaxed product automaton by Algorithms 4 and 10; 4. plan synthesis by Algorithm 3; 5. update transition system by Algorithm 11; 6. product automaton update by Algorithm 4; 7. validate and revise plan by Algorithm 14; 8. reply to requests by Algorithm 16.

hardware and are outputs of the same program.

ROS Node for Planning

The main contribution lies in the ROS node package for the motion and task planning algorithms presented in Chapters 2 and 3. 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 (ROSWiki, 2013).

The structure of the ROS node for planning is shown in Fig. 4.13. In initialization step, the initial finite transition system is encoded by the NetworkX graph structure (Hagberg et al., 2008) and 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 formulas into Büchi automaton by running the executable file from Gastin and Oddoux (2001). 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 Lex parser (Horrigan, 2013). It allows for: (i) fast encoding and evaluation of propositional formulas when constructing the product automaton; (ii) efficient integration with the distance evaluation in Algorithms 7 and 10. Both the full construction by Algorithm 1 and on-the-fly construction by Algorithm 4 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 (Likhachev et al., 2008; Van Den Berg et al., 2006). Normally anytime planning algorithms find a preliminary, possibly highly suboptimal solution very quickly within a bounded time window. While this preliminary plan is being executed, the planner continues to improve the preliminary plan until the optimal one is found. They work particularly well for static and fully-known workspaces. However if the workspace is dynamic, the robot has to re-plan frequently during the execution. Then the anytime planner may loss the anytime capability as it needs to generate new (highly-suboptimal) plans from scratch frequently. Instead, the local revising algorithm by Algorithm 14 and the event-based optimality check are more suitable.

4.4 Discussion

In this chapter, we extend the single-agent reconfiguration scheme under infeasible tasks to the multi-agent system with locally-assigned tasks. A decentralized solution is proposed to synthesize the individual motion plans that satisfy the mutual specification as much as possible. Secondly, we propose a cooperative planning and reconfiguration framework based on local knowledge transfer and share among the agents. At last, we present the software implementation of the motion and task planning algorithms proposed in this thesis.

Chapter 5

Conclusions and Future Work

THis chapter concludes the thesis by summarizing the main results presented in Chapters 2-4, as well as some future research topics.

5.1 Summary

Model-checking-based Motion and Task Planning

In Chapter 2 we present the complete framework to synthesize a hybrid control strategy that navigates an autonomous robot such that a high-level task specification is fulfilled. First of all, a detailed description is given about constructing the finite abstraction of the robot's motion within the workspace as a finite transition system, which takes into account both the workspace geometrical structure and the robot dynamics. A generic control task can be specified formally with respect to this transition system using Linear-time Temporal Logic formulas. A fully-automated scheme is proposed to synthesize (i) the discrete motion and task plan that satisfies the task; (ii) the hybrid controller that executes this discrete plan. It is guaranteed that the robot's final trajectory within the workspace fulfills the given task. Detailed algorithms are provided for implementation purposes.

Reconfiguration and Real-time Adaptation

Chapter 3 is devoted to improve the reconfigurability and real-time adaptability of the nominal framework presented in Chapter 2. We first tackle the scenario when the given task is infeasible for the initially-known workspace.

Instead of returning a failure, we propose an approach to synthesize a balanced plan that fulfills the task as much as possible. Then we extend this approach to the case where the task specification contains hard and soft parts. It is shown that the synthesized plan fulfills the hard specification for safety and the soft specification as much as possible for performance. The plan synthesized off-line may not be executable since the initial workspace model might not be consistent with the actual workspace. Thus we propose an on-line and real-time planing framework that takes into account unexpected changes in the transition system, evaluates the current plan regarding validity and safety and revising the plan if needed. In the end, due to the observation that most tasks of practical interest consists of not only sequence of motions but also actions to be performed, we present a systematic way to model the robot's full functionality including both motion and actions, such that any task specification in terms of desired motions and actions can be treated within the same framework.

Multi-agent Systems with Local Tasks

In Chapter 4, we consider a team of interconnected robots with different, independently-assigned, even conflicting local tasks. Firstly, we address the cooperative planning problem where one robot's task might be dependent on some of the other robots. We abstract the dependency cluster within the network where all robots belonging to one cluster are dependent directly or indirectly by a dependency chain. Their motion and task plans are synthesized together such that the mutual specification is satisfied as much as possible by their jointed motions. Secondly, even if they do not have dependent tasks, one robot may have knowledge about the workspace that is of interest to other robots. Thus we propose a distributed knowledge transfer and share framework for multi-agent systems. While the system runs, each robot updates its knowledge about the workspace via its sensing capability and shares this knowledge with its neighboring robots. Based on the knowledge update, each robot verifies and revises its motion plan in real time. It is ensured that the hard specification is always fulfilled for safety and the satisfaction for the soft specification is improved gradually. The design only assumes local interactions and applies to both static or dynamic communication topologies. Last but not least, as an important part of the contribution, the software implementation of the proposed motion and action planning framework is presented with detailed descriptions.

5.2 Future Work

Our future work will be focused on the following aspects:

(1) automated abstraction. As mentioned in Section 2.1, the process of constructing the finite transition system is not fully automated as it relies on analyzing both the workspace structure and the robot's dynamical properties. Part of our future work is to bridge this gap for certain types of workspace structure and robot dynamics.

(2) optimality. As mentioned in Chapter 2, the projection of the obtained optimal accepting run of the product automaton is not necessarily the optimal plan. The structure of Büchi automaton plays an important role within this framework. Different translation softwares and different formula expressions of the same task may result in distinctive results. A future research direction would be to either implement the tight Büchi automaton construction from Clarke et al. (1994) or search for the optimal plan given any generic Büchi automaton. However the trade-off between computational complexity and optimality remains.

(3) complexity. The on-the-fly construction is only one of the existing techniques in model-checking-based verification algorithms to tackle the state explosion problem. There are other methods like logic representation of the finite transition system (McMillan, 1993); bounded model-checking (Biere et al., 2003). Nevertheless, how to utilize those techniques for our concerns on real-time reconfiguration or adaptation and cost optimality remains challenging.

(4) robustness and fault tolerance. The failure occurred during the robot's motion from one region to another can be incorporated in its transition system update as shown in Section 3.3. However since it is not uncommon to see robot being unsuccessful in accomplishing certain actions, the planner should also adapt itself to these action failures. This is closely related to the motion and action planning presented in Section 3.4. How to model the action failure within the same framework and how to adjust the motion and action plan accordingly are our on-going research.

(5) flexibility. The solution proposed in Section 4.1 for synthesizing the joined motion is distributed with respect to dependency clusters, meaning that the synchronized coordination for agents belonging to one cluster is required. This imposes certain communication load and less flexibility for the overall system. A fully-distributed coordination scheme with reduced communication and improved flexibility is part of our future work.

(6) continuous constraints. Beside the dependency relation introduced by cooperative tasks, the motion of networked agents may also be subjected to constraints from their continuous states, e.g., pairwise distance and relative velocity. Future work involves to incorporate these continuous constraints into our discrete planning and motion control framework.

Bibliography

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(2s): 95 (2013).

S. B. Akers. Binary decision diagrams. *Computers, IEEE Transactions on*, 100(6): 509–516 (1978).

R. Aldebaran. Nao. `http://www.aldebaran-robotics.com/en/` (2014).

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*, 343–352. ACM (2013).

C. Baier, J.-P. Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge (2008).

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).

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*, 2689–2696. IEEE (2010).

A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58: 117–148 (2003).

R. Botsman and R. Rogers. *What's mine is yours: The rise of collaborative consumption.* HarperCollins (2010).

S. P. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge university press (2004).

E. P. Chan and Y. Yang. Shortest path tree computation in dynamic graphs. *Computers, IEEE Transactions on*, 58(4): 541–557 (2009).

E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at ltl model checking. In *Computer Aided Verification*, 415–427. Springer (1994).

E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* MIT press (1999).

J. Cortés. Distributed algorithms for reaching consensus on general functions. *Automatica*, 44(3): 726–737 (2008).

Creattica. Smart house. `http://creattica.com/infographics/smart-house/46656` (2014).

DARPA. Darpa robotics challenge. `http://www.fbo.gov/utils/view?id=74d674ab011d5954c7a46b9c21597f30` (2012).

T. L. Dean and M. P. Wellman. *Planning and control.* Morgan Kaufmann Publishers Inc. (1991).

D. V. Dimarogonas and K. J. Kyriakopoulos. Decentralized motion control of multiple agents with double integrator dynamics. In *16th IFAC World Congress, to appear* (2005).

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).

X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta. Automatic deployment of robotic teams. *Robotics & Automation Magazine*, 18(3): 75–86 (2011a).

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*, 532–538. IEEE (2011b).

J. C. Doyle, B. A. Francis, and A. Tannenbaum. *Feedback control theory*, volume 1. Macmillan Publishing Company New York (1992).

E. H. Durfee. Distributed problem solving and planning. In *Multi-agent systems and applications*, 118–149. Springer (2006).

G. E. Fainekos. Revising temporal logic specifications for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 40–45. IEEE (2011).

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).

G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42): 4262–4291 (2009).

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).

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*, 6235–6240. IEEE (2012).

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*, 1988–1993. IEEE (2010).

P. Gastin and D. Oddoux. Fast ltl to büchi automata translation. In *Computer Aided Verification*, 53–65. Springer (2001).

M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16) (1998).

M. Ghallab, D. Nau, and P. Traverso. *Automated planning: theory & practice*. Access Online via Elsevier (2004).

Gostai. Gostai by jazz security. `http://www.gostai.com/security/` (2013).

M. Guo and M. Colledanchise. Ltl planning with nao. `http://www.youtube.com/watch?v=-TLVGPN8Lxo` (2014).

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*. IEEE (2013).

M. Guo and D. V. Dimarogonas. Distributed plan reconfiguration via knowledge transfer in multi-agent systems under local ltl specifications. In *IEEE International Conference on Robotics and Automation, Hongkong, China* (2014a).

M. Guo and D. V. Dimarogonas. Multi-agent cooperative motion and task planning. `https://www.dropbox.com/s/wzzy2hvv6z9sh4y/Demo.avi` (2014b).

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*, 240–245. IEEE (2013a).

M. Guo, K. H. Johansson, and D. V. Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *IEEE International Conference on Robotics and Automation, Karlsruhe, Germany* (2013b).

A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL) (2008).

D. Hardawar. Driving app waze builds its own siri for hands-free voice control (2012).

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*, 3270–3285. IEEE (2012).

D. Horrigan. Lex parser. `https://github.com/pyrocms/lex` (2013).

S. Karaman and E. Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *American Control Conference, 2008*, 2003–2009. IEEE (2008).

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7): 846–894 (2011).

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).

H. K. Khalil. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River (2002).

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*, 265–271. IEEE (2012).

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).

D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4): 412–442 (1990).

J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *The Journal of Machine Learning Research*, 7: 1789–1828 (2006).

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).

S. M. LaValle. Rapidly-exploring random trees a new tool for path planning. *Technical Report 98-11* (1998).

S. M. LaValle. *Planning algorithms.* Cambridge university press (2006).

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).

M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14): 1613–1643 (2008).

S. R. Lindemann, I. I. Hussein, and S. M. LaValle. Real time feedback control for nonholonomic mobile robots with obstacles. In *Decision and Control, 2006 45th IEEE Conference on*, 2406–2411. IEEE (2006).

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*, 5163–5170. IEEE (2012).

S. G. Loizou and A. Jadbabaie. Density functions for navigation-function-based systems. *Automatic Control, IEEE Transactions on*, 53(2): 612–617 (2008).

S. G. Loizou and K. J. Kyriakopoulos. Closed loop navigation for multiple holonomic vehicles. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, 2861–2866. IEEE (2002).

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).

K. L. McMillan. *Symbolic model checking*. Springer (1993).

S. Misra and B. J. Oommen. Dynamic algorithms for the shortest path routing problem: learning automata-based solutions. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35(6): 1179–1192 (2005).

R. Olfati-Saber and J. S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, 6698–6703. IEEE (2005).

E. P. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, 324–332. Morgan Kaufmann Publishers Inc. (1989).

V. Raman and H. Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using ltlmop. In *Computer Aided Verification*, 663–668. Springer (2011).

W. Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multi-agent coordination. In *American Control Conference, 2005. Proceedings of the 2005*, 1859–1864. IEEE (2005).

P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *arXiv preprint arXiv:1105.5444* (2011).

Rethink. Rethink robotics. `http://www.rethinkrobotics.com/` (2013).

N. Robotics. Professional service robots: continued increase. `http://www.worldrobotics.org/index.php?id=home&news_id=262` (2012).

Romo. Phone-powered robots for fun and games. `http://romotive.com/meet-romo` (2013).

Roomba. Roomba vacuum cleaner. `http://www.irobot.com/global/sv/roomba_range.aspx` (2013).

ROSWiki. Ros wikipedia. `http://wiki.ros.org/ROS/Introduction` (2013).

H. Saïdi and N. Shankar. Abstract and model check while you prove. In *Computer Aided Verification*, 443–454. Springer (1999).

V. Schuppan and A. Biere. *Shortest counterexamples for symbolic model checking of LTL with past.* Springer (2005).

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).

S. Srinivas, R. Kermani, K. Kim, Y. Kobayashi, and G. Fainekos. *A Graphical Language for LTL Motion and Mission Planning.* Ph.D. thesis, Master's thesis, Arizona State University (2013).

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).

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* (2013a).

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*, 1–10. ACM (2013b).

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*, 4693–4698. IEEE (2012).

J. Van Den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Robotics and Automation, 2006. Proceedings 2006 IEEE International Conference on*, 2366–2371. IEEE (2006).

F. Van Harmelen, V. Lifschitz, and B. Porter. *Handbook of knowledge representation*. Elsevier (2008).

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).

E. M. Wolff, U. Topcu, and R. M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *Decision and Control,IEEE 51st Conference on*, 3372–3379. IEEE (2012).

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*, 101–110. ACM (2010).

B. Yordanov and C. Belta. Formal analysis of discrete-time piecewise affine systems. *Automatic Control, IEEE Transactions on*, 55(12): 2834–2840 (2010).

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).