Article

The International Journal of
Robotics Research
1–18
© The Author(s) 2014
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364914546174
ijr.sagepub.com

Multi-agent plan reconfiguration under local LTL specifications

Meng Guo and Dimos V Dimarogonas

Abstract
We propose a cooperative motion and task planning scheme for multi-agent systems where the agents have independently assigned local tasks, specified as linear temporal logic formulas. These tasks contain hard and soft sub-specifications. A least-violating initial plan is synthesized first for the potentially infeasible task and the partially-known workspace. This discrete plan is then implemented by the potential-field-based navigation controllers. While the system runs, each agent updates its knowledge about the workspace via its sensing capability and shares this knowledge with its neighbouring agents. Based on the knowledge update, each agent 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 is distributed as only local interactions are assumed. The overall framework is demonstrated by a case study and an experiment.

Keywords
Autonomous agents, distributed robot systems, networked system, cognitive robotics, agent-based systems

1. Introduction

Temporal-logic-based motion planning has gained significant attention in recent years, as it provides a fully automated correct-by-design controller synthesis approach for autonomous robots. Temporal logics such as linear temporal logic (LTL) and computation tree logic (CTL) provide formal high-level languages that can describe planning objectives more complex than the well-studied point-to-point navigation (LaValle, 2006; Karaman and Frazzoli, 2011). The task specification is given as a temporal logic formula with respect to the discretized abstraction of the robot motion modelled as a finite transition system (Fainekos et al., 2006; Belta et al., 2007; Bhatia et al., 2010; Kloetzer and Belta, 2010; Kress-Gazit et al., 2011; Wongpiromsarn et al., 2012). Then a high-level discrete plan is found by off-the-shelf model-checking algorithms (Baier and Katoen, 2008) given the finite transition system and the task specification. This discrete plan is then implemented through the corresponding low-level hybrid controller.

The LTL methodology has also been applied for multi-agent systems. Most of the existing work focuses on how to decompose a global specification to bisimilar local ones in a top-down approach, which can be then implemented by individual agents in a synchronized (Karaman and Frazzoli, 2008; Chen et al., 2012) and partially-synchronized (Ding et al., 2011; Kloetzer et al., 2011; Ulusoy et al., 2013) manner. Here we instead assume that local task specifications are assigned independently to each agent and there is no pre-specified global task, such as in Filippidis et al. (2012); Guo and Dimarogonas (2013); Tumova and Dimarogonas (2014); Guo and Dimarogonas (2014). Since we assume that the workspace is only partially-known to each agent initially, the task specification might be infeasible given this initial workspace model and the current motion plan might need to be revised whenever the workspace is updated.

Our previous work (Guo and Dimarogonas, 2013) addresses the first aspect by synthesizing a static motion plan off-line that fulfils a potentially infeasible task as much as possible; Guo and Dimarogonas (2014) considers hard and soft sub-specifications, where the hard specification should be always fulfilled while the soft part can be relaxed. We further improve this technique by combining the discrete plan revision with low-level hybrid controller

KTH Centre for Autonomous Systems and ACCESS Linnaeus Center, EES, KTH Royal Institute of Technology, Stockholm, Sweden

Corresponding author:
Meng Guo, KTH Centre for Autonomous Systems and ACCESS Linnaeus Center, EES, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden.
Email: mengg@kth.se

updates, to ensure successful execution of the plan. Similar strategies to synthesize the least-violating plan under safety rules are proposed by Tumova et al. (2013), where the level of satisfiability is measured differently from this work. Moreover, instead of synthesizing a static plan once and off-line, we put here emphasis on the real-time plan reconfiguration and implementation strategy to guarantee safety for all time and improve satisfiability gradually. The problem of revising infeasible task specification is also studied by Kim et al. (2012), where the relaxed specification is the closest to the original one and can be fulfilled. Since there is often more than one plan satisfying the relaxed specification that may fulfil the original task to different extents, we aim at synthesizing directly the plan that balances between its implementation cost and the satisfiability with respect to the original task, which at the same time is adapted along with the agent's knowledge update about the workspace. Raman and Kress-Gazit (2011) proposes a framework to analyse potentially unsynthesizable parts of the task specifications using LTLMoP toolkit. Regarding the second aspect related to real-time plan revising, related approaches can be found for single-agent systems, e.g., in our earlier work (Guo et al., 2013), in Livingston et al. (2012) by local "patching", in Maly et al. (2013) by a multi-layered synergistic framework; for multi-agent systems in Guo and Dimarogonas (2014). In this work, new algorithms for the on-the-fly construction of product automaton are proposed to reduce the computational complexity of both updating the system model and revising the motion plan. Johnson and Kress-Gazit (2011) analyses the correctness of high-level robot behaviours under sensor uncertainties with a probabilistic approach, while robustness of this controller synthesis approach for system with model uncertainties is investigated by Wolff et al. (2012) and Topcu et al. (2012).

Potential-field-based navigation techniques such as the navigation function proposed by Koditschek and Rimon (1990) provide an easy-to-implement and provably correct point-to-point navigation algorithm, which has been successfully applied in both single- (Loizou and Jadbabaie, 2006) and multi-agent (Dimarogonas and Kyriakopoulos, 2007) motion planning under different geometric constraints. Formal high-level languages such as LTL and CTL described earlier allow us to describe more complex planning objectives. Attempts to combine the strengths of both frameworks have appeared in the work of Loizou and Kyriakopoulos (2004), Fainekos et al. (2006) and Filippidis et al. (2012). Loizou and Kyriakopoulos (2004) provide a centralized approach to switch among various control subobjectives such as formation and navigation. Our earlier work (Filippidis et al., 2012) proposes a framework for decentralized verification from local LTL specifications and under connectivity constraints. However, in both references above the workspace is assumed to be fully-known and no real-time reconfiguration scheme is considered.

The proposed cooperative motion and task planning scheme for multi-agent systems has the following attributes: (i) locally assigned tasks and a partially-known workspace are considered; (ii) new regions of interest are added to the system model in real-time after knowledge update and incorporated in the planning algorithm afterwards; (iii) it is distributed since only local interaction between neighbouring agents are assumed; (iv) while the hard specification that preserves safety is always guaranteed by the reconfiguration algorithm, the satisfaction of the soft specification is improved gradually by triggering the optimal plan synthesis algorithm in an event-based fashion (Heemels et al., 2012); (v) the communication payload is significantly reduced compared with a fully synchronized solution, due to the subscriber–publisher scheme that will be described in the sequel; both static and dynamic communication topologies can be handled; (vi) computational complexity is reduced by the on-the-fly construction and revision of the system model; (vii) it can potentially be applied to many existing partition and motion planning techniques, such as probabilistic roadmap method and rapidly-exploring random trees (Bhatia et al., 2010; Karaman and Frazzoli, 2011; Bhatia et al., 2011).

The rest of the paper is organized as follows. Section 2 briefly introduces essential preliminaries. The description of agent dynamics and task specifications are given in Section 3. In Section 4, we present how to synthesize an initial optimal plan. Section 5 focuses on how the knowledge is transferred among the agents. Real-time algorithms to update the system model and the agent plan are provided in Section 6. Section 7 summarizes the overall structure of the framework. Numerical examples and experiments are presented in Section 8. Conclusions and directions for future research are summarized in Section 9.

2. Preliminary

2.1. Potential-field-based navigation

The workspace we consider is bounded by a large sphere in 2-D space $\mathcal{W} = \{q \in \mathbb{R}^2 |\, \|q - q_0\| \leq r_0\}$, where $q_0 \in \mathbb{R}^2$ and $r_0 > 0$ are the centre and radius. Within $\mathcal{W}$ there exists N smaller spheres around the points of interest, which are described by $\pi_i = \mathcal{B}_{r_i}(q_i) = \{q \in \mathbb{R}^2 |\, \|q - q_i\| \leq r_i\}$, where $q_i \in \mathbb{R}^2$ is the point of interest and $r_i > 0$ is the radius of π_i. Note that $\{r_i\}$ represent the margins we want to attain with respect to the points of interest, rather than necessarily some physical quantities. Denote by $\Pi = \{\pi_1, \ldots, \pi_N\}$ and $\pi_{\text{free}} = \mathcal{W} \backslash \Pi = \{q \in \mathbb{R}^2 |\, q \in \mathcal{W},\, q \notin \Pi\}$ as the free space in $\mathcal{W}$.

Definition 1. *The partition of* Π *within* $\mathcal{W}$ *is valid if and only if: (i)* $\pi_i \subset \mathcal{W}$, *for all* $\pi_i \in \Pi$; *(ii)* $\pi_i \cap \pi_j = \emptyset$, *for all* $\pi_i, \pi_j \in \Pi$ *and* $\pi_i \neq \pi_j$.

Given a valid set of sphere regions Π within $\mathcal{W}$, the potential function proposed by Koditschek and Rimon (1990) provides a straightforward algorithm to construct a path from almost any initial position in the free space (except a set of measure zero) to any goal position in the free space, while avoiding all obstacles represented by sphere regions in Π. In detail, denote by q_s, $q_d \in \pi_{\text{free}}$ the initial and goal positions, where $q_s \neq q_d$. The associated navigation function is given by

$$\Phi(q) \triangleq \frac{\gamma}{(\gamma^k + \beta)^{\frac{1}{k}}} \tag{1}$$

where $k > 0$ is a design parameter, $q \in \mathbb{R}^2$ and $\Phi \in [0,1]$. Here $\gamma \triangleq \|q - q_d\|^2$ represents the attractive potential from the goal q_d and $\beta \triangleq \prod_{j=0}^{N} \beta_j$ is the repulsive potential from the the workspace boundary and sphere obstacles, where $\beta_0 \triangleq r_0^2 - \|q - q_0\|^2$ and $\beta_j \triangleq \|q - q_j\|^2 - r_j^2$, $j = 1,\ldots, N$. It has been proved by Koditschek and Rimon (1990) that $\Phi(q)$ reaches its minimal value 0 only at the goal point and maximal value 1 at the boundaries of $\mathcal{W}$ and Π.

In addition to its provable mathematical correctness, another strength of (1) is that it provides a straightforward motion planning algorithm. By following the negated gradient $-\nabla_q \Phi$, it is guaranteed that for sufficiently large k, (i) $\gamma \to 0$, namely $q \to q_d$ when $t \to \infty$; (ii) $\beta_j > 0$ holds for all $t \geq 0$, $j = 0, 1,\ldots, N$. This navigation algorithm plays an important role in the hybrid controller synthesis scheme later.

2.2. LTL

The basic ingredients of an LTL formula are a set of atomic propositions AP and several Boolean and temporal operators, which are formed according to the following syntax (Baier and Katoen, 2008): $\varphi ::= \texttt{True} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \,\texttt{U}\, \varphi_2$, where $a \in AP$ and $\bigcirc$ (*next*), $\texttt{U}$ (*until*). For brevity, we omit the derivations of other operators such as $\square$ (*always*), $\diamond$ (*eventually*), $\Rightarrow$ (*implication*) and the semantics of LTL formulas. We refer the readers to Chapter 5 of Baier and Katoen (2008). There is a union of infinite words that satisfy φ: $\texttt{Words}(\varphi) = \{\sigma \in (2^{AP})^{\omega} | \sigma \models \varphi\}$, where $\models \subseteq (2^{AP})^{\omega} \times \varphi$ is the satisfaction relation, see Definition 5.6 of Baier and Katoen (2008).

Furthermore there exists a non-deterministic Büchi automaton (NBA) $\mathcal{A}_\varphi$ over 2^{AP} corresponding to φ (Clarke et al., 1999; Baier and Katoen, 2008). It is defined as $\mathcal{A}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ where Q is a finite set of states; $Q_0 \subseteq Q$ is the set of initial states, 2^{AP} is the set of input alphabets; δ: $Q \times 2^{AP} \to 2^Q$ is a transition relation and $\mathcal{F} \subseteq Q$ is a set of accepting states. Denote by $\chi(q_m, q_n) = \{l \in 2^{AP} | q_n \in \delta\,(q_m, l)\}$ the set of all input alphabets that enable the transition from q_m to q_n in δ. An infinite run R of a NBA is an infinite sequence of states, which starts from an initial state and follows the transition relation. An infinite run 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 Baier and Katoen (2008). Denote by $\mathcal{L}_\omega(\mathcal{A}_\varphi)$ the accepted language of $\mathcal{A}_\varphi$, which is the set of infinite words over 2^{AP} that result in an accepting run in $\mathcal{A}_\varphi$. There always exists $\mathcal{A}_\varphi$ such that $\texttt{Words}(\varphi) = \mathcal{L}_\omega(\mathcal{A}_\varphi)$, see Theorem 5.41 of Baier and Katoen (2008). There are fast translation algorithms (Gastin and Oddoux, 2001) from an LTL formula to NBA. This process can be done in time and space $2^{\mathcal{O}(|\varphi|)}$.

3. Agent description

3.1. Dynamics and abstraction

We consider a team of autonomous agents with unique identities (IDs) $k \in \mathcal{K} = \{1, 2,\ldots, K\}$. They all satisfy the single-integrator dynamics:

$$\dot{q}_k = u_k \tag{2}$$

where $q_k(t)$, $u_k(t) \in \mathbb{R}^2$ are the position and control signal of agent k at time t. For all agents, the allowed sphere workspace is the same and known *a priori*, denoted by $\mathcal{W} = \{q \in \mathbb{R}^2 | \|q - q_0\| \leq r_0\}$, where $q_0 \in \mathbb{R}$ and $r_0 > 0$ are the centre and radius. Note that the agents are modelled as point masses without volume, meaning that inter-agent collisions are not considered.

Each agent has different initial knowledge of the workspace $\mathcal{W}$ and this knowledge is updated via the sensing and communication scheme described later. In particular, the set of sphere regions known to agent k at time t is denoted by $\Pi_k^t = \{\pi_{k,i}, i = 1, \ldots, N_k\}$, where $\pi_{k,i} = \mathcal{B}_{r_{k,i}}(q_{k,i})$; $q_{k,i}$ and $r_{k,i}$ are the centre and radius; $N_k \in \mathbb{N}^+$. The corresponding free space is given by $\pi_{k,\text{free}}^t = \mathcal{W} \backslash \Pi_k^t = \{q \in \mathbb{R}^2 | q \in \mathcal{W}, q \notin \Pi_k^t\}$. There is a set of atomic propositions indicating the properties of interest to agent k, denoted by $AP_k = \{a_1, a_2, \ldots, a_{M_k}\}$, which does not change with time. The properties satisfied by $\pi_{k,i} \in \Pi_k^t$ known to agent k at time t is denoted by the labeling function $L_k^t : \Pi_k^t \to 2^{AP_k}$. Note that AP_k might be different among the agents due to heterogeneity.

Instead of finding a collision-free path in the free space from an initial point to a goal point as discussed in Section 2.1, we are interested in constructing the feedback controller that navigates agent k from any point within one sphere region $\pi_{k,s} \in \Pi_k^t$ to one point within another region $\pi_{k,g} \in \Pi_k^t$ where $\pi_{k,g} \neq \pi_{k,s}$, while avoiding the rest of the sphere regions in Π_k^t. Let $\pi_{k,g} = \mathcal{B}_{r_{k,g}}(q_{k,g})$ and $\pi_{k,s} = \mathcal{B}_{r_{k,s}}(q_{k,s})$. This can be done in two steps: (i) construct the potential function $\Phi(q)$ using (1), where $q_{k,g}$ is the goal position and $\{\pi_{k,j} \in \Pi_k^t, j \neq s, g\}$ are regions to avoid; specifically, $\gamma_g = \|q - q_{k,g}\|^2$ and $\beta_{sg} \triangleq \prod_{j=0, j\neq s,g}^{N_k} \beta_j$, where $\beta_0 \triangleq r_0^2 - \|q - q_0\|^2$, and $\beta_j \triangleq \|q - q_{k,j}\|^2 - r_{k,j}^2$; (ii) by following the negated gradient of $\Phi(q)$ with respect to q, it is guaranteed that from any point within $\pi_{k,s}$, there is a collision-free path to some point within $\pi_{k,g}$. In particular, we denote by $U_k^t(\pi_{k,s}, \pi_{k,g})$ the navigation controller from $\pi_{k,s}$ to $\pi_{k,g}$:

$$U_k^t(\pi_{k,s}, \pi_{k,g}) = u_k(t) = -\lambda_k \nabla_q \frac{\gamma_g}{(\gamma_g^k + \beta_{sg})^{\frac{1}{k}}} \tag{3}$$

where $\lambda_k > 0$ is the controller gain. Another particular case of (3) is that when $\pi_{k,s}$ is set to $\emptyset$ as used in Section 4.3, it means that this agent starts from one point in the free space and all regions in Π_k^t except $\pi_{k,g}$ are treated as regions to avoid. Note that the asymptotic stability of the above controller guarantees the convergence to some point within $\pi_{k,g}$ in finite time (Loizou and Jadbabaie, 2006). It can be verified that Π_k^t remains valid after removing $\pi_{k,g}$ and $\pi_{k,s}$ if Π_k^t is valid within $\mathcal{W}$.

With the above ingredients, agent k's possible motion within $\mathcal{W}$ at time t is abstracted as a weighted finite transition system (wFTS):

$$\mathcal{T}_k^t = (\Pi_k^t,\ \rightarrow_k^t,\ \Pi_{k,0}^t,\ AP_k,\ L_k^t,\ W_k^t) \tag{4}$$

where: (i) $\Pi_k^t = \{\pi_{k,1}, \ldots, \pi_{k,N_k}\}$ is the finite set of sphere regions defined earlier, where $\pi_{k,n} = \mathcal{B}_{r_{k,n}}(q_{k,n})$, for all $n = 1,\ldots, N_k$; (ii) $\rightarrow_k^t \subseteq \Pi_k^t \times \Pi_k^t$ is the transition relation; (iii) $\Pi_{k,0}^t \in \Pi_k^t$ is the initial region the robot may start from; (iv) $AP_k = \{a_1, a_2, \ldots, a_{M_k}\}$ is the set of atomic propositions introduced earlier; (v) $L_k^t : \Pi_k^t \rightarrow 2^{AP_k}$ is the labeling function; (vi) $W_k^t : \rightarrow_k^t \rightarrow \mathbb{R}^+$ represents the implementation (energy/time) cost (Guo et al., 2013), which is approximated by the straight-line distance between regions $\|q_{k,s} - q_{k,g}\|$ for $(\pi_{k,s}, \pi_{k,g}) \in \rightarrow_k^t$.

Note that Loizou and Kyriakopoulos (2003) also provides the NF-based control strategy for non-holonomic vehicles. The rest of the framework still applies by replacing (3) with the one proposed in Loizou and Kyriakopoulos (2003). Compared with other cell decomposition schemes such triangles (Belta et al., 2007), polygons (Belta et al., 2005) and hexagons (Oikonomopoulos et al., 2009), this sphere-area-based approach typically reduces the size of the resulting abstraction since it only includes regions of interest, rather than a complete partition of the workspace.

We assume that $\mathcal{T}_k^t$ does not have a terminal state (Baier and Katoen, 2008). Then an infinite path of $\mathcal{T}_k^t$ is an infinite sequence of states $\tau_k^t = \pi_{k,0}\pi_{k,1}\pi_{k,2}\ldots$ such that $(\pi_{k,i}, \pi_{k,i+1}) \in \rightarrow_k^t$ for all $i > 0$. Its trace is defined as the sequence of atomic propositions that are true at the states along the path, i.e. $\texttt{trace}(\tau_k^t) = L_k^t(\pi_{k,0})\, L_k^t(\pi_{k,1}) L_k^t(\pi_{k,2})\ldots$, which is an infinite word over 2^{AP_k}.

3.2. Local task specification

Agent k's task specification φ_k is an LTL formula over AP_k. We assume that φ_k remains unchanged for all agents once the system starts. Denote by $\varphi_k|_{AP_k}$ as the set of atomic propositions appearing in φ_k. In particular, we consider the task specification φ_k with the following structure:

$$\varphi_k = \varphi_k^{\text{soft}} \wedge \varphi_k^{\text{hard}} \tag{5}$$

where φ_k^{soft} and φ_k^{hard} are "soft" and "hard" sub-formulas. Here φ_k^{hard} could include safety constraints such as collision-avoidance: "avoid all obstacles" or energy-supply guarantee: "visit the charging station infinitely often". Introducing soft and hard specifications is due to the observation that the partially-known workspace might render parts of the specification infeasible initially and, thus, yielding the need for them to be relaxed, while the safety-critical parts should not be relaxed during the process.

Definition 2. *Given an infinite path $\tau_k^t = \pi_{k,0}\pi_{k,1}\pi_{k,2}\ldots$ of $\mathcal{T}_k^t$ and the task specification φ_k, τ_k is called: (i)* ***valid*** *if $(\pi_{k,i}, \pi_{k,i+1}) \in \rightarrow_k^t$, for $i = 0,1,2\ldots$; (ii)* ***safe*** *if $\texttt{trace}(\tau_k^t) \models \varphi_k^{\text{hard}}$; (iii)* ***satisfying*** *if $\texttt{trace}(\tau_k^t) \models \varphi_k$.*

4. Initial task and motion plan synthesis

In this section, we discuss how each agent synthesizes its initial task and motion plan when the system starts, based on their initial knowledge and task specification.

4.1. Safety-ensured product automaton

Denote by $\mathcal{A}_k^{\text{hard}} = (Q_1,\ 2^{AP_k},\ \delta_1,\ Q_{1,0},\ \mathcal{F}_1)$ and $\mathcal{A}_k^{\text{soft}} = (Q_2,\ 2^{AP_k},\ \delta_2,\ Q_{2,0},\ \mathcal{F}_2)$ as the NBA associated with φ_k^{hard} and φ_k^{soft}, respectively. Analogously the functions $\chi_1()$ of $\mathcal{A}_k^{\text{hard}}$ and $\chi_2()$ of $\mathcal{A}_k^{\text{soft}}$ are defined as in Section 2.2. Now we propose a way to construct the intersection of $\mathcal{A}_k^{\text{hard}}$ and $\mathcal{A}_k^{\text{soft}}$ in a way that it is safety-ensured and relaxed.

Definition 3 (Relaxed automata intersection)**.** *The relaxed intersection of $\mathcal{A}_k^{\text{hard}}$ and $\mathcal{A}_k^{\text{soft}}$ is defined by:*

$$\widetilde{\mathcal{A}}_{\varphi_k} = (Q,\ 2^{AP_k},\ \delta,\ Q_0,\ \mathcal{F}) \tag{6}$$

where $Q = Q_1 \times Q_2 \times \{1,2\}$; $Q_0 = Q_{1,0} \times Q_{2,0} \times \{1\}$; $\mathcal{F} = \mathcal{F}_1 \times Q_2 \times \{1\}$; δ: $Q \times 2^{AP_k} \rightarrow 2^Q$, with $\langle \check{q}_1, \check{q}_2, \check{c}\rangle \in \delta(\langle q_1, q_2, c\rangle, l)$ when the following three conditions hold: (i) $l \in \chi_1(q_1, \check{q}_1)$; (ii) $\chi_2(q_2, \check{q}_2) \neq \emptyset$; (iii) $q_1 \notin \mathcal{F}_1$ and $\check{c} = c = 1$; or $q_2 \notin \mathcal{F}_2$ and $\check{c} = c = 2$; or $q_1 \in \mathcal{F}_1$, $c = 1$ and $\check{c} = 2$; or $q_2 \in \mathcal{F}_2$, $c = 2$ and $\check{c} = 1$.

Note that in Definition 3 we relax the requirement that there should exist a common input alphabet that enables the transitions from q_i to $\check{q}_i$ for both $i = 1,2$, compared with the standard definition of Büchi automata intersection (see Chapter 4.3 of Baier and Katoen (2008)). An accepting run R of $\widetilde{\mathcal{A}}_{\varphi_k}$ should intersect with the accepting set $\mathcal{F}$ infinitely often. The last component $c \in \{1,2\}$ 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 1 below. Denote by $R|_{Q_1}$ the projection of R onto the states of $\mathcal{A}_k^{\text{hard}}$.

Theorem 1. *Given an accepting run R of $\widetilde{\mathcal{A}}_{\varphi_k}$, $R|_{Q_1}$ is an accepting run of $\mathcal{A}_k^{\text{hard}}$. Moreover, $\mathcal{L}_\omega(\widetilde{\mathcal{A}}_{\varphi_k}) \subseteq \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$.*

Proof. By the definition of an accepting run, 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 $R|_{Q_1}$. On the other hand, since $l \in \chi_1(q_1, \check{q}_1)$ is ensured by the definition of δ, all transitions along $R|_{Q_1}$ are valid for $\mathcal{A}_k^{\text{hard}}$. As a result, $R|_{Q_1}$ is an accepting run of $\mathcal{A}_k^{\text{hard}}$. For the second part, given any infinite word σ with $\sigma \in \mathcal{L}_\omega(\widetilde{\mathcal{A}}_{\varphi_k})$, σ results in an accepting run of $\widetilde{\mathcal{A}}_{\varphi_k}$, denoted by R_σ. It has been proved that $R_\sigma|_{Q_1}$ is also an accepting run of $\mathcal{A}_k^{\text{hard}}$, which implies that $\sigma \in \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$. Thus, for any $\sigma \in \mathcal{L}_\omega(\widetilde{\mathcal{A}}_{\varphi_k})$, $\sigma \in \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$ holds, namely $\mathcal{L}_\omega(\widetilde{\mathcal{A}}_{\varphi_k}) \subseteq \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$. □

Since we need to guarantee that φ_k^{hard} is fulfilled completely and φ_k^{soft} is fulfilled as much as possible, the standard model-checking-based planning algorithm (Baier and Katoen, 2008; Fainekos et al., 2009) may fail to provide a solution. We rely on our earlier work (Guo and Dimarogonas, 2013, 2014) to construct the weighted product automaton in order to handle both feasible and potentially infeasible LTL specifications.

Definition 4 (Weighted product automaton)**.** *The weighted product Büchi automaton $\widetilde{\mathcal{A}}_{p,k}^t = \mathcal{T}_k^t \times \widetilde{\mathcal{A}}_{\varphi_k} = (Q', \delta', Q_{0'}, \mathcal{F}', W_p)$ is defined as follows*:

- $Q' = \Pi_k^t \times Q$, $q' = \langle \pi, q\rangle \in Q'$, *for all* $\pi \in \Pi_k^t$ *and for all* $q \in Q$;
- δ': $Q' \to 2^{Q'}$. $\langle \pi_j, q_n\rangle \in \delta'(\langle \pi_i, q_m\rangle)$ *if and only if* $(\pi_i, \pi_j) \in \to_k^t$ and $q_n \in \delta(q_m, L_k^t(\pi_i))$;
- $Q_0' = \Pi_{0,t}^t \times Q_0$ *is the set of initial states*;
- $\mathcal{F}' = \Pi_k^t \times \mathcal{F}$ *is the set of accepting states*;
- $W_p : \delta' \to \mathbb{R}^+$ *is the weight function to be defined.*

Recall that $AP_k = \{a_1, \ldots, a_{M_k}\}$. To define the weight function W_p, we first introduce the evaluation function Eval: $2^{AP_k} \to \{0,1\}^{M_k}$:

$$\texttt{Eval}(l) = \nu \Leftrightarrow [\,\nu_i\,] = \begin{cases} 1 & \text{if } a_i \in l \\ 0 & \text{if } a_i \notin l \end{cases} \tag{7}$$

where $i = 1,2,\ldots, M_k$, $l \in 2^{AP_k}$ and $\nu \in \{0,1\}^{M_k}$. Then a metric $(2^{AP_k}, \rho)$ is defined as

$$\rho(l, l') = \|\nu - \nu'\|_1 = \sum_{i=1}^{M_k} |\,\nu_i - \nu_i'\,| \tag{8}$$

where $\nu = \texttt{Eval}(l)$, $\nu' = \texttt{Eval}(l')$ and $l, l' \in 2^{AP_k}$; $\|\cdot\|_1$ is the ℓ_1 norm. Here $\rho(\cdot)$ also measures the Hamming distance (Forney, 1966) between two Boolean strings l and l'. Then we define the distance between an element $l \in 2^{AP_k}$ to a set $\chi \subseteq 2^{AP_k} (\chi \neq \emptyset)$ (Boyd and Vandenberghe, 2009):

$$\texttt{Dist}(l, \chi) = \begin{cases} 0 & \text{if } \ l \in \chi \\ \min_{l' \in \chi} \rho(l, l') & \text{otherwise} \end{cases} \tag{9}$$

Note that $\texttt{Dist}(l, \chi)$ is not defined for $\chi = \emptyset$. For $\langle \pi_j, q_n\rangle \in \delta'(\langle \pi_i, q_m\rangle)$, its weight is computed in the following way:

$$\begin{aligned} &W_p(\langle \pi_i, q_m\rangle, \langle \pi_j, q_n\rangle) \\ &= W_k^t(\pi_i, \pi_j) + \alpha \cdot \texttt{Dist}(L_k^t(\pi_i), \chi_2(q_2, \check{q}_2)) \end{aligned} \tag{10}$$

where $q_m = \langle q_1, q_2, c\rangle$ and $q_n = \langle \check{q}_1, \check{q}_2, \check{c}\rangle$; $\alpha \geq 0$ is a design parameter; $\chi_2(q_2, \check{q}_2) = \{l \in 2^{AP_k} | (q_2, l, \check{q}_2) \in \delta_2\}$ consists of all input alphabets that enable the transition from q_2 to $\check{q}_2$ in $\mathcal{A}_k^{\text{soft}}$. Recall that $\chi_2(q_2, \check{q}_2) \neq \emptyset$ by condition (ii) in Definition 3. Moreover, it holds that $\texttt{Dist}\,(L_k^t(\pi_i), \chi_1(q_1, \check{q}_1)) = 0$, for all $\langle \pi_j, q_n\rangle \in \delta'(\langle \pi_i, q_m\rangle)$, $q_m = \langle q_1, q_2, c\rangle$ and $q_n = \langle \check{q}_1, \check{q}_2, \check{c}\rangle$ from the definition of δ'. The weight function (10) consists of two parts: $W_k^t(\pi_i, \pi_j)$ measures the implementation cost of the transition from π_i to π_j and $\texttt{Dist}\,(L_k^t(\pi_i), \chi_2(q_2, \check{q}_2))$ measures how much this transition violates the constraints imposed by $\mathcal{A}_k^{\text{soft}}$. The design parameter α reflects the relative penalty on violating the soft specification, which should be chosen relatively large under partially-known workspace.

Theorem 2. *Assume that R_k^t is an accepting run of $\widetilde{\mathcal{A}}_{p,k}^t$. Its projection on Π_k^t, $\tau_k = R_k^t|_{\Pi_k^t}$, is both **valid** and **safe** for $\mathcal{T}_k^t$ and φ_k by Definition 2.*

Proof. The fact that τ_k^t is valid can be verified from the definition of δ'. This is because every transition in δ', when projected onto Π_k^t, is a valid transition within $\to_k$, meaning that τ_k^t is always implementable by $\mathcal{T}_k^t$. Second, since R_k^t is an accepting run of $\widetilde{\mathcal{A}}_{p,k}^t$, then $\texttt{trace}(\tau_k^t) \in \mathcal{L}_\omega(\widetilde{\mathcal{A}}_{\varphi_k})$, which implies $\texttt{trace}(\tau_k^t) \in \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$by Theorem 1. Since $\texttt{Words}(\varphi_k^{\text{hard}}) = \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$, $\texttt{trace}(\tau_k^t) \in \texttt{Words}(\varphi_k^{\text{hard}})$, which indicates that τ_k^t is also safe by Definition 2.

Given the desired value of α in (10), $\widetilde{\mathcal{A}}_{p,k}^t$ can be fully constructed by Definition 4, as proposed in Guo et al (2013) and Smith et al. (2011). However since the size of $\widetilde{\mathcal{A}}_{p,k}^t$ is particularly very large for practical problems, especially given that the sizes of both $\mathcal{A}_k^{\text{hard}}$ and $\mathcal{A}_k^{\text{soft}}$ are exponential to the length of formulas φ_k^{soft} and φ_k^{hard}. Furthermore, as discussed in Section 5, $\mathcal{T}_k^t$ and $\widetilde{\mathcal{A}}_{p,k}^t$ need to be updated frequently given the multi-agent knowledge transfer scheme and partially-known workspace.

Thus, we propose to construct $\widetilde{\mathcal{A}}_{p,k}^t$ on-the-fly along with the plan synthesis and revising algorithms proposed in Algorithms 3 and 8 later. In other words, the states and transition relations of $\widetilde{\mathcal{A}}_{p,k}^t$ are built "on demand". When the planning algorithm visits any state $q_s{}' \in Q'$ and calls Algorithm 1 for the adjacency relation of $q_s{}'$, Algorithm 1 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 $q_s{}'$ need to be re-constructed. If $q_s{}'$ is marked 'unvisited', lines 5–12 construct the transitions by Definition 4 where q'_s's successor q'_g is added to q' as states to be visited (line 10) by the planning algorithms introduced later. If q'_s is marked 'visited', it means that its adjacency relation has been constructed thus can be returned directly (lines 2–4). Algorithm 2 checks whether $q'_g \in \delta'(q'_s)$, given a pair of states (q'_s, q'_g). If so, q'_g is added to $\delta'(q'_s)$ and its weight is calculated by Definition 4 (lines 4–5). The markers "visited" and "unvisited" play an important role in the proposed framework: (i) the adjacency relation of states that

Algorithm 1. Adjacency relation of $\tilde{\mathcal{A}}^t_{p,k}$, `Adj()`.

```
Input: product state $q'_s$, $\tilde{\mathcal{A}}^t_{p,k}$
Output: successors $\delta'(q'_s)$ and associated weights
Update: $\tilde{\mathcal{A}}^t_{p,k}$
1  $q'_s = \langle \pi_i, q_m \rangle$
2  if $q'_s$ is marked 'visited' then
3      Return $\delta'(q'_s)$ and associated weights
4  else if $q'_s$ is marked 'unvisited' then
5      forall the $(\pi_i, \pi_j) \in \longrightarrow^t_k$ do
6          forall the $q_n \in \delta(q_m, l), l \in 2^{AP_k}$ do
7              $q'_g = \langle \pi_j, q_n \rangle$
8              if $q'_g \notin Q'$ then
9                  add $q'_g$ to $Q'$
10             if CheckTran($(q'_s, q'_g), \tilde{\mathcal{A}}^t_{p,k}$) is True then
11                 add $q'_g$ to $\delta'(q'_s)$.
12     mark $q'_s$ as 'visited'
13     Return $\delta'(q'_s)$ and associated weights
```

Algorithm 2. Check potential transition (q'_s, q'_g) of $\tilde{\mathcal{A}}^t_{p,k}$, `CheckTran()`.

```
Input: pair of states $(q'_s, q'_g)$, $\tilde{\mathcal{A}}^t_{p,k}$
Output: Boolean value to indicate if $q'_g \in \delta'(q'_s)$
Update: $\tilde{\mathcal{A}}^t_{p,k}$
1  $q'_s = \langle \pi_i, q_m \rangle$, $q'_g = \langle \pi_j, q_n \rangle$
2  $q_m = \langle q_1, q_2, c \rangle$, $q_n = \langle \check{q}_1, \check{q}_2, \check{c} \rangle$
3  if $L^t_k(\pi_i) \in \chi_1(q_1, \check{q}_1)$ then
4      add $q'_g$ to $\delta'(q'_s)$
5      compute its weight $W_p(q'_s, q'_g)$ by (10)
6      return True
7  return False
```

have been visited before can be returned directly, meaning that the results from previous planning iterations can be reused later; (ii) it provides an efficient way to update $\widetilde{\mathcal{A}}^t_{p,k}$ in case of updates in $\mathcal{T}^t_k$ as described in Algorithm 6 that follows.

4.2. Discrete plan synthesis

In order to measure the implementation cost of different accepting runs of $\widetilde{\mathcal{A}}_{p,k}$ and how much they violate the soft specification, we consider the accepting runs with the following *prefix–suffix* structure:

$$R^t_k = q'_0 q'_1 \cdots [q'_f q'_{f+1} \cdots\cdots q'_n]^\omega, \tag{11}$$

where $q'_0 \in Q'_0$ and $q'_f \in \mathcal{F}'$. The prefix part $(q'_0 q'_1 \ldots q'_f)$ of R^t_k from an initial state q'_0 to one accepting state q'_f is executed only once. The suffix part $(q'_f q'_{f+1} \ldots q'_n)$ of R^t_k from q'_f back to itself is repeated infinitely. An accepting run with the prefix–suffix structure has a finite representation as (11) and the total cost of R^t_k is given by

$$\begin{aligned}\texttt{Cost}(R^t_k, \widetilde{\mathcal{A}}^t_{p,k}) &= \sum_{i=0}^{f-1} W_p(q'_i, q'_{i+1}) + \gamma \sum_{i=f}^{n-1} W_p(q'_i, q'_{i+1}) \\ &= \texttt{cost}_{\tau^t_k} + \alpha \cdot \texttt{dist}_{\varphi^{\text{soft}}_k}\end{aligned} \tag{12}$$

where $\gamma \geq 0$; the second equality is derived by inserting (10);

$$\texttt{cost}_{\tau^t_k} = \sum_{i=0}^{f-1} W^t_k(\pi_i, \pi_{i+1}) + \gamma \sum_{i=f}^{n-1} W^t_k(\pi_i, \pi_{i+1})$$

is the accumulated implementation cost of the motion plan $\tau^t_k = R^t_k|_{\Pi^t_k}$;

$$\begin{aligned}\texttt{dist}_{\varphi^{\text{soft}}_k} &= \sum_{i=0}^{f-1} \texttt{Dist}(L^t_k(\pi_i), \chi_2(q'_i|_{Q_2}, q'_{i+1}|_{Q_2})) \\ &\quad + \gamma \sum_{i=f}^{n-1} \texttt{Dist}(L^t_k(\pi_i), \chi_2(q'_i|_{Q_2}, q'_{i+1}|_{Q_2}))\end{aligned}$$

is the accumulated distance of τ^t_k with respect to the soft constraints imposed by $\mathcal{A}_{\varphi^{\text{soft}}_k}$, where $q'_i|_{Q_2}$ and $q'_{i+1}|_{Q_2}$ are the projection of q'_i and q'_{i+1} onto the states Q_2 of $\mathcal{A}^{\text{soft}}_k$. The design parameter γ represents the relative weighting on the cost of transient response (the prefix) and steady response (the suffix) to the task specification (Ulusoy et al., 2013).

Remark 1. *Compared with the cost function proposed in* Smith et al. *(2011) which is designed for fully-known workspaces and feasible task specifications*, (12) *incorporates both the implementation cost and satisfiability to the task specification, between which the balance is tunable by changing 'α'. This is an important aspect for the multi-agent reconfiguration scheme under partially-known workspaces and potentially infeasible tasks, where each agent could have different 'α' based on its knowledge about the workspace.*

The prefix–suffix structure is more of a way to formulate the total cost of an accepting run, rather than a conservative assumption. If an accepting run exists, by its definition at least one accepting state should appear in it infinitely often. Among all the finite number of cycles starting for this accepting state and back to itself there is one with the minimal cost. Thus, an accepting run of the form (11) can be built using this minimal cycle as the periodic suffix.

Definition 5 (Optimal plan)**.** *The accepting run that minimizes* $\texttt{Cost}(R^t_k, \widetilde{\mathcal{A}}^t_{p,k})$ *is called the optimal accepting run and denoted by* $R^t_{\text{opt},k}$*. Its projection onto* Π^t_k, $\tau^t_{\text{opt},k} = R^t_{\text{opt},k}|_{\Pi^t_k}$*, is called the optimal motion plan of* $\widetilde{\mathcal{A}}^t_{p,k}$.

Algorithm 3 takes as input arguments the adjacency relation $\texttt{Adj}(\widetilde{\mathcal{A}}^t_{p,k})$ from Algorithm 1, the set of initial states q'_0 and the set of accepting states $\mathcal{F}'$. It utilizes Dijkstra's algorithm (LaValle, 2006) for computing the

Algorithm 3. Optimal plan synthesis `OptPlan()`.

```
Input: $\tilde{\mathcal{A}}^t_{p,k}$
Output: the optimal run $R^t_{\mathrm{opt},k}$ and associated plan $\tau^t_{\mathrm{opt},k}$
Update: $\tilde{\mathcal{A}}^t_{p,k}$
1 Initialize $Q'_0$, $\mathcal{F}'$ of $\tilde{\mathcal{A}}^t_{p,k}$
2 For each initial state $q'_0 \in Q'_0$, call DijksTargets(Adj($\tilde{\mathcal{A}}^t_{p,k}$), $q'_0$, $\mathcal{F}'$)
3 For each accepting state $q'_f \in \mathcal{F}'$, call DijksCycle(Adj($\tilde{\mathcal{A}}^t_{p,k}$), $q'_f$)
4 Find the pair of $(q'_{0,*}, q'_{f,*})$ that minimizes the summed cost by (12)
5 Optimal accepting run $R^t_{\mathrm{opt},k}$: prefix, the shortest path from $q'_{0,*}$ to $q'_{f,*}$; suffix, the shortest cycle from $q'_{f,*}$ and back to itself
6 $\tau^t_{\mathrm{opt},k} = R^t_{\mathrm{opt},k}|_{\Pi^t_k}$
```

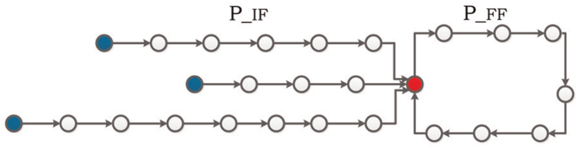

Fig. 1. For every pair of initial state $q'_0 \in q'_0$ (in blue) and accepting state $q'_f \in \mathcal{F}'$ (in red), the shortest path from q'_0 to q'_f and the shortest cycle containing q'_f are computed.

shortest path from a single source node to a set of target nodes within a weighted graph. In particular, function DijksTargets(Adj($\widetilde{\mathcal{A}}^t_{p,k}$), *source*, *targets*) computes shortest paths in $\widetilde{\mathcal{A}}^t_{p,k}$ from "*source*" state to every target state belonging to the set '*targets*'. Function DijksCycle (Adj($\widetilde{\mathcal{A}}^t_{p,k}$), *source*) is used to compute shortest cycle from the "*source*" state back to itself. As shown in Figure 1, 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 DijksTargets($\cdot$) is called while the shortest cycle containing q'_f is obtained from line 2 of Algorithm 3 where DijksCycle($\cdot$) is called. Finally, the pair $(q'_{0,*}, q'_{f,*})$ that minimizes the total cost defined by (12) is chosen. Then the optimal accepting run is determined by setting its prefix as the shortest path from $q'_{0,*}$ to $q'_{f,*}$ and its suffix as shortest cycle containing $q'_{f,*}$ (line 4).

As a result, Algorithm 3 relies on Algorithm 1 to explore the adjacency relation of $\widetilde{\mathcal{A}}^t_{p,k}$, which at the same time is constructed on-the-fly by adding states and transitions as discussed in Section 4.1. By Theorem 2, $\tau^t_{\mathrm{opt},k} = R^t_{\mathrm{opt},k}|_{\Pi^t_k}$ is always valid and safe no matter how the values of α and γ are chosen. Moreover, if the accumulated distance $\mathtt{dist}_{\varphi^{\mathrm{soft}}_k} = 0$ in (12), it implies that $\tau^t_{\mathrm{opt},k}$ satisfies φ_k fully, i.e. both $\varphi^{\mathrm{soft}}_k$ and $\varphi^{\mathrm{hard}}_k$. Algorithm 3 summarizes the optimal plan synthesis algorithm.

Remark 2. Algorithm 3 *can be applied directly when* $\varphi^{\mathrm{soft}}_k$ *is feasible without any modification. This is due to the fact that when 'α' is large enough, i.e. the penalty on violating* $\mathcal{A}^{\mathrm{soft}}_k$ *is severe, Algorithm 3 will select the accepting run that satisfies* $\varphi^{\mathrm{soft}}_k$.

4.3. Hybrid controller synthesis

Without loss of generality, let the optimal plan from Algorithm 3 be $\tau^t_{\mathrm{opt},k} = \pi_{k,0}\,\pi_{k,1}\cdots\pi_{k,i}\cdots$, where $\pi_{k,i}$ is the *i*th state along $\tau^t_{\mathrm{opt},k}$. The corresponding hybrid controller that implements the plan is constructed as follows: for each transition $(\pi_{k,i}, \pi_{k,i+1}) \in \tau^t_{\mathrm{opt},k}$, the corresponding controller $U^t_k(\pi_{k,i}, \pi_{k,i+1})$ is activated until agent *k* is out of region $\pi_{k,i}$ and within the free space $\pi^t_{k,\mathrm{free}}$. The purpose is to avoid that agent *k* enters region $\pi_{k,i}$ again. After that, the controller is switched to $U^t_k(\emptyset, \pi_{k,i+1})$ to navigate agent *k* to region $\pi_{k,i+1}$ without entering other regions in Π^t_k. To avoid discontinuities between controller switching, we introduce a fading function as proposed in Loizou and Kyriakopoulos (2004), which ensures a continuous transition from $U^t_k(\emptyset, \pi_{k,i})$ to $U^t_k(\pi_{k,i}, \pi_{k,i+1})$ and from $U^t_k(\pi_{k,i}, \pi_{k,i+1})$ to $U^t_k(\emptyset, \pi_{k,i+1})$. Assume that agent *k* enters region $\pi_{k,i}$ at time T_1, leaves region $\pi_{k,i}$ at time $T_2 > T_1$, and enters region $\pi_{k,i+1}$ at time $T_3 > T_2$. We construct the switch function:

$$s(x) = \frac{1}{2}(sat(2x-1)+1) \tag{13}$$

where $sat(\cdot)$ is the standard saturation function: $sat(x) = x$, if $|x| \leq 1$; $sat(x) = x/|x|$ if $|x| > 1$. Then for $t \in [T_1, T_2)$, the control signal $u_k(t)$ is given by

$$u_k(t) = (1 - s(\sigma))\,U^t_k(\emptyset, \pi_{k,i}) + s(\sigma)\,U^t_k(\pi_{k,i}, \pi_{k,i+1}) \tag{14}$$

where $\sigma = \frac{t-T_1}{\nu_1}$; $\nu_1 > 0$ is a design parameter indicating the time period of this switching process. For $t \in [T_2, T_3)$, the control signal $u_k(t)$ is given by

$$u_k(t) = (1 - s(\sigma))\,U^t_k(\pi_{k,i}, \pi_{k,i+1}) + s(\sigma)\,U^t_k(\emptyset, \pi_{k,i+1}) \tag{15}$$

where $\sigma = \frac{t-T_2}{\nu_2}$; $\nu_2 > 0$ is a design parameter indicating the time period of this switching process. This procedure is repeated for all $i = 0,1,2\ldots$, where the suffix of $\tau^t_{\text{opt},k}$ is repeated infinitely often.

5. Knowledge update and transfer

The agents have both the sensing ability to discover the workspace and the communication functionality to share knowledge with their neighbouring agents. In this section, we will discuss how the sensing ability is modelled, the structure of the communication network and the communication protocol among the agents.

Denote by $\mathbf{Sense}^t_k$ as the set of sensing information obtained at time $t \geq 0$ by agent k. Note that this information might be gathered when an agent reaches a region or during the transition from one region to another. Moreover, this information should only reflect the changes of the workspace model perceived by agent k. It has the following format:

$$\mathbf{Sense}^t_k = \{([q, r], S, S_\neg), E, E_\neg\} \tag{16}$$

where $[q, r]$ is the dimension of an perceived sphere region; $S \subseteq AP_k$ is the set of propositions satisfied by this region; $S_\neg \subseteq AP_k$ is the set of propositions not satisfied by this region; $(\pi_i, \pi_j) \in E$ if (π_i, π_j) needs to be added to $\rightarrow^t_k$; $(\pi_i, \pi_j) \in E_\neg$ if (π_i, π_j) needs to be removed from $\rightarrow^t_k$. $\mathbf{Sense}^t_k$ reflects the actual workspace observed by agent k at time t.

This sensing functionality can be modelled by assigning a sensing radius $h_k \geq 0$, such that all points of interest that lie with the sphere $\{q \in \mathbb{R}^2 | \|q - q_k(t)\| \leq h_k\}$ are visible, where $q_k(t) \in \mathbb{R}^2$ is agent k's position at time t.

5.1. Knowledge transfer

In this section, we explain how the knowledge about the workspace is shared among the agents.

5.1.1. Communication network. The communication network represents how the information flows among the agents. Each agent k has a set of neighbouring agents, denoted by $\mathcal{K}_k \subseteq \mathcal{K}$. Agent k can send messages directly to any agent belonging to $\mathcal{K}_k$. We take into account two different ways to model the communication network: (i) global communication with a fixed topology; (ii) limited communication with a dynamic topology. In the first case, $\mathcal{K}_k$ is pre-defined and fixed after the system starts. In the second case, each agent has a communication range, denoted by $c_k \geq 0$. Agent k can only send messages to agent g if their relative distance is less than c_k, i.e. $\|q_g(t) - q_k(t)\| \leq c_k$ where $q_g(t)$, $q_k(t) \in \mathbb{R}^2$ are the positions of agents g and k at time t. Then $\mathcal{K}^t_k = \{g \in \mathcal{K} | \|q_g(t) - q_k(t)\| \leq c_k\}$ is the time-varying neighbouring set of agent k at time t, namely k can only send messages to $g \in \mathcal{K}^t_k$. Different communication radius would result in a directed communication topology.

Algorithm 4. Transfer knowledge, `TranKnow()`.

```
   Input: $\mathbf{Sense}^t_k$, $\mathbf{Request}^{t_0}_{g,k}$
   Output: $\mathbf{Reply}^t_{k,g}$
1  forall the $([q, r], S, S_\neg) \in \mathbf{Sense}^t_k$ do
2      forall the $\mathbf{Request}^{t_0}_{g,k}$ received at $t_0 < t$ do
3          if $g \in \mathcal{K}^t_k$ then
4              $S' = S \cap (\varphi_g|_{AP_g})$
5              $S'_\neg = S_\neg \cap (\varphi_g|_{AP_g})$
6              if $S' \neq \emptyset$ or $S'_\neg \neq \emptyset$ then
7                  add $([q, r], S', S'_\neg)$ to $\mathbf{Reply}^t_{k,g}$
```

5.1.2. Communication protocol. Agent k is interested in all of the propositions appearing in φ_k, namely $\varphi_k|_{AP_k}$. We propose a *subscriber–publisher* communication mechanism to reduce the communication load for each agent. Whenever agent g communicates with agent $k \in \mathcal{K}^t_g$ for the *first* time at time t, it follows the *subscribing* procedure: agent g sends a request message to agent k that

$$\mathbf{Request}^t_{g,k} = \varphi_g|_{AP_g} \tag{17}$$

which informs agent k the set of workspace properties agent g is interested in. Each agent has a subscriber list, containing the request messages it has received. Note that each agent sends a request to any of its neighbouring agents only once, meaning that each agent also needs to keep track of the agents which it has subscribed to.

Afterwards, the *publishing* phase of each agent follows an event-driven approach: whenever an agent k has obtained a sensing update $\mathbf{Sense}^t_k$, it checks its subscriber list whether the content might be of interest to any of the subscribers regarding some propositions. If it is of interest to agent g regarding some proposition in $\varphi_g|_{AP_g}$, then agent k checks if $g \in \mathcal{K}^t_k$. If so, it publishes a reply message to agent g that

$$\mathbf{Reply}^t_{k,g} = \{([q, r], S', S'_\neg)\} \tag{18}$$

where $S' = S \cap (\varphi_g|_{AP_g})$ and $S'_\neg = S_\neg \cap (\varphi_g|_{AP_g})$; $[q, r]$ contains the dimension of the sphere region that satisfies S' but not $S'_\neg$ according to agent k's latest sensing update. The above procedure is summarized in Algorithm 4. Note that since $'$ and $S'_\neg$ only contain propositions that are relevant to agent g's task φ_g, every reply message from agent k to g contains useful knowledge for agent g. This subscriber–publisher scheme can be easily implemented by multicast or unicast wireless protocols.

5.2. Knowledge update

At time t, agent k might obtain new knowledge from $\mathbf{Sense}^t_k$ and $\mathbf{Reply}^t_{g,k}$ as described before, based on which it needs to update its own system model. In particular, the size and property of certain regions need to be changed and new regions might be added. The update includes two aspects: the finite transition system $\mathcal{T}^t_k$ and the underlying navigation controllers.

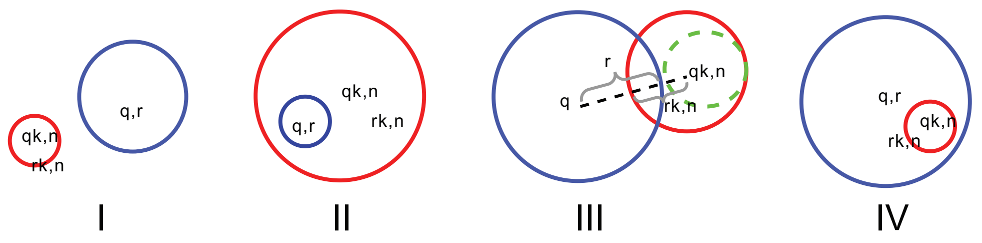

Fig. 2. Different relative positions between the newly-perceived sphere region (in blue) $\pi_{\text{new}} = \mathcal{B}_r(q)$ and an existing region (in red) $\pi_{k,n} \in \Pi_k^{t^-}$.

5.2.1. Transition system update. Denote by $\mathcal{T}_k^{t^-}$ and $\mathcal{T}_k^{t^+}$ as the transition system before and after the update at time t. Recall that $([q, r], S, S_\neg) \in \textbf{Sense}_k^t$ or $\textbf{Reply}_{g,k}^t$ indicates that a sphere region satisfying S and not satisfying $S_\neg$ is detected. Denote by $\pi_{\text{new}} = \mathcal{B}_r(q)$ as the newly-perceived region. Since agent k's initial knowledge of the workspace might not be correct in terms of centre, radius and property of the regions, these parameters need to be updated accordingly. For $\pi_{k,n} \in \Pi_k^{t^-}$, let $\mathcal{B}_{r_{k,n}^{t^-}}(q_{k,n}^{t^-})$, $\mathcal{B}_{r_{k,n}^{t^+}}(q_{k,n}^{t^+})$, $L_k^{t^-}(\pi_{k,n})$ and $L_k^{t^+}(\pi_{k,n})$ be the size and property of $\pi_{k,n}$ before and after the update. Depending on the relative position between π_{new} and any $\pi_{k,n}$, there are three different cases:

(i) $\pi_{\text{new}} \cap \pi_{k,n} = \emptyset$, for all $\pi_{k,n} \in \Pi_k^{t^-}$; namely π_{new} does not intersect with any of the existing region in $\Pi_k^{t^-}$ (case I in Figure 2); then π_{new} is added to $\Pi_k^{t^-}$ and it can be verified that $\Pi_k^{t^+}$ is valid;

(ii) $\pi_{\text{new}} \subseteq \pi_{k,n}$ or $\pi_{k,n} \subset \pi_{\text{new}}$; it means that the size and properties of $\pi_{k,n}$ should be changed, namely $\pi_{k,n} = \mathcal{B}_r(q)$ and $L_k^{t^+}(\pi_{k,n}) = L_k^{t^-}(\pi_{k,n}) \cap S \setminus S_\neg$ (cases II and IV in Figure 2);

(iii) $\pi_{\text{new}} \cap \pi_{k,n} \neq \emptyset$, $\pi_{\text{new}} \not\subset \pi_{k,n}$ and $\pi_{k,n} \not\subset \pi_{\text{new}}$; then π_{new} is added to $\Pi_k^{t^-}$ as $\pi_{k,|N_k+1|}$ and $L_k^{t^+}(\pi_{k,|N_k+1|}) = S$, while the center and radius $q_{k,n}^{t^+}, r_{k,n}^{t^+}$ of $\pi_{k,n}$ is modified by

$$\left[\frac{r_{k,n}^{t^-} + r}{d} q_{k,n}^{t^-} - \frac{r_{k,n}^{t^-} + r - d}{d} q, \ \frac{r_{k,n}^{t^-} + d - r}{2} - \varepsilon\right]$$

where $d = \|q - q_{k,n}^{t^-}\|$ and $\epsilon > 0$ is a tuning parameter indicating the minimal distance between two neighbouring regions (case III in Figure 2); thus $\pi_{k,n} = \mathcal{B}_{q_{k,n}^{t^+}}(r_{k,n}^{t^+})$ and $L_k^{t^+}(\pi_{k,n}) = L_k^{t^-}(\pi_{k,n})$; if π_{new} intersects with more than one regions in $\Pi_k^{t^-}$, the above procedure needs to be applied to each one of them.

Regarding $E, E_\neg \in \textbf{Sense}_k^t$, new transitions are added to $\mathcal{T}_k^{t^-}$ based on E while transitions in $E_\neg$ are removed from $\mathcal{T}_k^{t^+}$. Algorithm 5 describes how to construct $\mathcal{T}_k^{t^+}$ from $\mathcal{T}_k^{t^-}$ based on $\textbf{Sense}_k^t$ and $\textbf{Reply}_{g,k}^t$; $\tilde{\Pi}_k^t \subseteq \Pi$ is used to store the set of regions within $\Pi_k^{t^-}$ of which the labelling function is changed during the update, which serves as an input argument to Algorithm 6 later. Note that if both $\textbf{Sense}_k^t$ and $\textbf{Reply}_{g,k}^t$ are empty, $\mathcal{T}_k^{t^+}$ remains the same as $\mathcal{T}_k^{t^-}$.

Algorithm 5. Update knowledge, `UpdKnow( )`.

```
   Input: $\mathcal{T}_k^{t^-}$, $\textbf{Sense}_k^t$, $\textbf{Reply}_{g,k}^t$.
   Output: $\mathcal{T}_k^{t^+}$, $\tilde{\Pi}_k^t$
1  forall the $([q, r], S, S_\neg) \in \textbf{Sense}_k^t$ or $\textbf{Reply}_{g,k}^t$ do
2      if Case I then
3          add $\pi_{\text{new}}$ to $\Pi_k^{t^-}$
4          $\pi_{\text{new}} = \mathcal{B}_r(q)$ and $L_k^{t^+}(\pi_{\text{new}}) = S$
5      if Cases II or IV then
6          $\pi_{k,n} = \mathcal{B}_r(q)$ and
           $L_k^{t^+}(\pi_{k,n}) = L_k^{t^-}(\pi_{k,n}) \cap S \setminus S_\neg$
7          if $L_k^{t^+}(\pi_{k,n}) \neq L_k^{t^-}(\pi_{k,n})$ then
8              add $\pi_{k,n}$ to $\tilde{\Pi}_k^t$
9      if Case III then
10         add $\pi_{\text{new}}$ to $\Pi_k^{t^-}$
11         $\pi_{\text{new}} = \mathcal{B}_r(q)$ and $L_k^{t^+}(\pi_{\text{new}}) = S$
12         $\pi_{k,n} = \mathcal{B}_{q_{k,n}^{t^+}}(r_{k,n}^{t^+})$ by (19),
           $L_k^{t^+}(\pi_{k,n}) = L_k^{t^-}(\pi_{k,n})$
13 remove $(\pi_i, \pi_j)$ from $\longrightarrow_k^{t^-}$, for all $(\pi_i, \pi_j) \in E_\neg^t$
14 add $(\pi_i, \pi_j)$ to $\longrightarrow_k^{t^-}$, for all $(\pi_i, \pi_j) \in E^t$
```

Algorithm 6. Update product automaton `UpdProd( )`.

```
   Input: $\tilde{\mathcal{A}}_{p,k}^t$, $\tilde{\Pi}_k^t$, $E^t$, $E_\neg^t$
   Update: $\tilde{\mathcal{A}}_{p,k}^t$
1  forall the $\pi_i \in \tilde{\Pi}_k^t$ and $(\pi_i, \pi_j) \in (E^t \cup E_\neg^t)$ do
2      forall the $q_m \in Q$ do
3          if $\langle \pi_i, q_m \rangle \in Q'$ then
4              mark $q_s' = \langle \pi_i, q_m \rangle$ as "unvisited"
```

5.2.2. Hybrid control scheme update. Assume that $\mathcal{T}_k^{t^-}$ is updated to $\mathcal{T}_k^{t^+}$ by Algorithm 5 at $t = T'$. Let agent k's current motion plan be $\tau_{\text{opt},k}$, which is obtained as the output of Algorithms 3 or 8. If agent k belongs to one region at $t = T'$, e.g. $q_k(T') \in \pi_{k,i}$, where $\pi_{k,i}$ is the ith state in $\tau_{\text{opt},k}$, the control signal of agent k for $t > T'$ is given by

$$u_k^*(t) = (1 - s(\sigma))\, u_k(T') + s(\sigma)\, U_k^{t^+}(\pi_{k,i}, \pi_{k,i+1}), \qquad (20)$$

where $u_k(T')$ is the control signal given by (14); $\sigma = \frac{t-T'}{\nu_3}$; the switching function $s(\sigma)$ is defined in (13); $\nu_3 > 0$ is a design parameter indicating the time period of this switching process; $U_k^{t^+}(\pi_{k,i}, \pi_{k,i+1})$ is given by (3) with respect to $\mathcal{T}_k^{t^+}$.

If agent k is during the transition from $\pi_{k,i}$ to $\pi_{k,i+1}$ at time $t = T'$, where $\pi_{k,i}$, $\pi_{k,i+1}$ are the i th and $(i+1)$ th state in $\tau_{\text{opt},k}$, the control signal for $t > T'$ is given by

$$u_k^*(t) = (1 - s(\sigma))\, u_k(T') + s(\sigma)\, \mathrm{U}_k^{t^+}(\emptyset, \pi_{k,i+1}), \qquad (21)$$

where $u_k(T')$ is the control signal given by (15); $\sigma = \frac{t-T'}{\nu_4}$; the switching function $s(\sigma)$ is defined in (13); $\nu_4 > 0$ is a design parameter indicating the time period of this switching process; $\mathrm{U}_k^{t^+}(\emptyset, \pi_{k,i+1})$ is given by (3) with respect to $\mathcal{T}_k^{t^+}$.

Given a sufficiently small transition time ν_1, ν_2, ν_3, $\nu_4 > 0$, the asymptotic stability of controllers $\mathrm{U}_k^{t^+}(\pi_{k,i}, \pi_{k,i+1})$ and $\mathrm{U}_k^{t^+}(\emptyset, \pi_{k,i+1})$ guarantees that the goal region $\pi_{k,i+1}$ can be reached in finite time.

6. Real-time reconfiguration

Since $\mathcal{T}_k^t$ might be updated as described in Section 5, the optimal accepting run and its corresponding optimal motion plan derived by Algorithm 3 in Section 4 need to be evaluated regarding their validity, safety and optimality.

6.1. Product automaton update

Denote by $\widetilde{\mathcal{A}}_{p,k}^{t^-}$ and $\widetilde{\mathcal{A}}_{p,k}^{t^+}$ as the product automaton corresponding to $\mathcal{T}_k^{t^-}$ and $\mathcal{T}_k^{t^+}$, respectively. Recall that $\tilde{\Pi}_k^t$ is the set of regions whose labelling function has changed, while E^t and $E_\neg^t$ are two sets of transitions in $\mathcal{T}_k^{t^-}$ that have been added and removed by Algorithm 5. To update $\widetilde{\mathcal{A}}_{p,k}^t$, a brute-force approach would be either to reconstruct the complete $\widetilde{\mathcal{A}}_{p,k}^t$ from scratch by Definition 4 using $\widetilde{\mathcal{A}}_{\varphi_k}$ and $\mathcal{T}_k^{t^+}$, or to re-evaluate all transitions within $\widetilde{\mathcal{A}}_{p,k}^{t^-}$ that are relevant to the latest changes in $\mathcal{T}_k^{t^+}$ as proposed in Guo and Dimarogonas (2014). However, both methods have the complexity proportional to the number of transitions within $\mathcal{A}_{\varphi_k}$ and more importantly most of the updated transitions of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$ might not be used by Algorithms 3 and 8 later. It would be computationally inefficient to reconstruct $\widetilde{\mathcal{A}}_{\varphi_k}$ using either of the above ways, in particular given the existence of the multi-agent knowledge transfer scheme and partially-known workspace.

Thus, we propose to incorporate the update information including $\tilde{\Pi}_k^t$, E^t and $E_\neg^t$ into the adjacency relation function of $\widetilde{\mathcal{A}}_{p,k}^t$ in Algorithm 1. As summarized in Algorithm 6, for any state $\pi_i \in \tilde{\Pi}_k^t$ whose labelling function has been changed, and all added or removed edges $(\pi_i, \pi_j) \in (E^t \cup E_\neg^t)$, the corresponding states $q'_s = \langle \pi_i, q_m \rangle \in q'$ are marked as "unvisited", indicating that all of the transitions originated from q'_s have to be re-constructed. In this way, the update information will only be used when q'_s is revisited by the planning Algorithms 3 and 8 and then $\delta'(q'_s)$ is re-constructed.

Algorithm 7. Validate the current run `ValidRun()`.

```
Input: $\tilde{\mathcal{A}}_{p,k}^{t^+}$, $R_k^{t^-}$, $\tilde{\Pi}_k^t$, $E_\neg^t$
Output: $\Xi_k^t$, $\aleph_k^t$
Update: $\tilde{\mathcal{A}}_{p,k}^{t^+}$
1 forall the $(q'_s, q'_{s+1}) \in$ Edge($R_k^{t^-}$) do
2     $q'_s = \langle \pi_i, q_m \rangle$, $q'_{s+1} = \langle \pi_j, q_n \rangle$
3     if $(\pi_i, \pi_j) \in E_\neg^t$ then
4         add $(q'_s, q'_{s+1})$ to $\Xi_k^t$
5     else if $\pi_i \in \tilde{\Pi}_k^t$ then
6         if CheckTran($(q'_s, q'_{s+1})$, $\tilde{\mathcal{A}}_{p,k}^{t^+}$) is False
          then
7             add $(q'_s, q'_{s+1})$ to $\aleph_k^t$
```

6.2. Validity and safety

Given the updated product automaton $\widetilde{\mathcal{A}}_{p,k}^{t^+}$, an accepting run $R_k^{t^-}$ of $\widetilde{\mathcal{A}}_{p,k}^{t^-}$ and the corresponding plan $\tau_k^{t^-}$, two natural questions arise. (1) Is $\tau_k^{t^-}$ still valid or safe? (2) If not, how can we modify $\tau_k^{t^-}$ such that it remains valid and safe for $\mathcal{T}_k^{t^+}$ and φ_k?

Before answering the first question, we need to define invalid and unsafe transitions within the current accepting run $R_k^{t^-}$. Denote by $\texttt{Edge}(R_k^{t^-})$ the set of all transitions appearing in $R_k^{t^-}$, which is finite due to the prefix–suffix structure by (11).

Definition 6. *Given $\mathcal{T}_k^{t^+}$ from Algorithm 5, a transition $(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) \in \texttt{Edge}(R_k^{t^-})$ is called: (i) invalid if $(\pi_i, \pi_j) \notin \to_k^{t^+}$; (ii) unsafe if $L_k^{t^+}(\pi_i) \notin \chi_1(q_m|_{Q_1}, q_n|_{Q_1})$, where $q_m|_{Q_1}$ and $q_n|_{Q_1}$ are the projections of q_m, q_n onto Q_1.*

The notation Ξ_k^t and $\aleph_k^t$ in Algorithm 7 are used to store the sets of invalid and unsafe transitions in $R_k^{t^-}$. Algorithm 7 iterates through each transition $(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle)$ within $\texttt{Edge}(R_k^{t^-})$ and checks whether (π_i, π_j) has been removed from $\mathcal{T}_k^t$ (line 3) or if the changed label $L_k^{t^+}(\pi_i)$ would make this transition unsafe (line 6), where function `CheckTran()` is called.

Theorem 3. *Assume that $R_k^{t^-}$ is an accepting run of $\widetilde{\mathcal{A}}_{p,k}^{t^-}$. Here Ξ_k^t, $\aleph_k^t$ are obtained from Algorithm 7. Then: (i) $\tau_k^{t^-}$ remains **valid** if and only if $\Xi_k^t = \emptyset$; (ii) $\tau_k^{t^-}$ remains **safe** if $\aleph_k^t = \emptyset$.*

Proof. Since $R_k^{t^-}$ is an accepting run of $\widetilde{\mathcal{A}}_{p,k}^{t^-}$, $\tau_k^{t^-}$ is both valid and safe for $\mathcal{T}_k^{t^-}$ by Theorem 2. If $\Xi_k^t = \emptyset$ and $\aleph_k^t = \emptyset$, $\texttt{Edge}(R_k^{t^-})$ does not contain any invalid or unsafe transitions by Definition 6. Thus $R_k^{t^-}$ is still an accepting run of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$. 'If' part of (i): by Theorem 2, $\tau_k^{t^-}$ is valid since $R_k^{t^-}$ remains an accepting run of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$. 'Only if' part of (i): if $\Xi_k^t \cap \texttt{Edge}(R_k^{t^-}) \neq \emptyset$, $R_k^{t^-}|_\Pi$ contains at least one

Algorithm 8. Revise the current plan, `Revise()`.

```
    Input: $\Xi_k^t$, $\aleph_k^t$, $R_k^{t^-}$, $\tilde{\mathcal{A}}_{p,k}^{t^+}$
    Output: $R_k^{t^+}$
    Update: $\tilde{\mathcal{A}}_{p,k}^{t^+}$
 1  forall the $(q'_s, q'_{s+1}) \in (\Xi_k^t \cup \aleph_k^t)$ do
 2      if $(q'_s, q'_{s+1}) \in R_{\text{pre}}^{t^-}$ then
 3          if $q'_s$ is first($R_{\text{pre}}^{t^-}$) then
 4              $q'_{s-1} = q'_s$
 5          $bridge$ = DijksTarget(Adj($\tilde{\mathcal{A}}_{p,k}^{t^+}$), $q'_{s-1}$, tail($q'_s$, $R_{\text{pre}}^{t^-}$))
 6          if $bridge \neq \emptyset$ then
 7              $R_{\text{pre}}^{t^+}$ = [head($q'_{s-1}$, $R_{\text{pre}}^{t^-}$) $bridge$ tail(last($bridge$), $R_{\text{pre}}^{t^-}$)]
 8          else
 9              $R_k^{t^+}$ = OptPlan($\tilde{\mathcal{A}}_{p,k}^{t^+}$)                    // Algorithm 3
10              break
11      if $(q'_s, q'_{s+1}) \in R_{\text{suf}}^{t^-}$ then
12          if $q'_s$ is last($R_{\text{pre}}^{t^-}$) then
13              tail($q'_s$, $R_{\text{suf}}^{t^-}$) = first($R_{\text{suf}}^{t^-}$)
14          repeat lines 3–8 but replace $R_{\text{pre}}^{t^-}$ by $R_{\text{suf}}^{t^-}$
15      $R_k^{t^-} = [R_{\text{pre}}^{t^+}, R_{\text{suf}}^{t^+}]$
```

invalid transition, thus not valid by Definition 2. 'If' part of (ii): by Theorem 2, $\tau_k^{t^-}$ is safe since $R_k^{t^-}$ remains an accepting run of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$. □

One possible solution for the second question could be to recall Algorithm 3 with respect to $\widetilde{\mathcal{A}}_{p,k}^{t^+}$, which generates a new optimal accepting run of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$. But this method requires a complete new search of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$, which is computationally inefficient (the worst-case time complexity being $\mathcal{O}(|\widetilde{\mathcal{A}}_{p,k}^t| \cdot \log |\widetilde{\mathcal{A}}_{p,k}^t| \cdot (|Q'_0| + |\mathcal{F}'|)))$ in the case of large $\mathcal{T}_k^t$ and complex φ_k, in the case of real-time applications and especially when $\widetilde{\mathcal{A}}_{p,k}^t$ has to be updated frequently.

Instead we are interested in revising $R_k^{t^-}$ locally such that it fulfils the accepting condition of $\widetilde{\mathcal{A}}_{p,k}^{t^+}$, but not necessarily optimal. We propose Algorithm 8 that serves this purpose, which is similar to Algorithm 5 of Guo et al. (2013). Denote by $R_{\text{pre}}^{t^-}$ and $R_{\text{suf}}^{t^-}$ the prefix and suffix part of $R_k^{t^-}$; `tail`$(q'_s, R_{\text{pre}}^{t^-})$ is the segment of $R_{\text{pre}}^{t^-}$ after q'_s, not including q'_s; `head`$(q'_{s-1}, R_{\text{pre}}^{t^-})$ is the segment of $R_{\text{pre}}^{t^-}$ before q'_{s-1} (not including q'_{s-1}). Furthermore, `last`(*bridge*) is the last state on the path segment "*bridge*" and first($R_{\text{pre}}^{t^-}$) is the first state of $R_{\text{pre}}^{t^-}$. Note that if q'_s is the first element of $R_{\text{pre}}^{t^-}$, q'_{s-1} is set to q'_s (lines 3–4) and `head`$(q'_{s-1}, R_{\text{pre}}^{t^-}) = \emptyset$. Algorithm 8 essentially takes in the invalid or unsafe transition (q'_s, q'_{s+1}) in $R_k^{t^-}$ from Algorithm 7 (either in $R_{\text{pre}}^{t^-}$ or $R_{\text{suf}}^{t^-}$) and locally finds a "bridging" segment that make up this transition by breadth-first search. Function `DijksTarget(Adj(`$\widetilde{\mathcal{A}}_{p,k}^{t^+}$`)`, *source*, *targets*) is defined similarly as function `DijksTargets(·)` in Algorithm 3, which returns the shortest path from the single "*source*" state to *any* target state belonging to the set "*targets*", where `Adj(`$\widetilde{\mathcal{A}}_{p,k}^{t^+}$`)` is the adjacency function from Algorithm 1. Thus, it returns the shortest path once one of the targets is reached. Last but not least, if any of the above calls to function `DijksTarget(·)` returns an empty "*bridge*", it means that the current accepting state is not reachable from q'_{s-1}. Then Algorithm 3 is called to perform an optimal search over $\widetilde{\mathcal{A}}_{p,k}^{t^+}$. Note that if q'_s is the last element of $R_{\text{suf}}^{t^-}$, `tail`$(q'_s, R_{\text{suf}}^{t^-})$ returns the first element of $R_{\text{suf}}^{t^-}$, i.e. the accepting state, which guarantees the cyclic structure of the suffix (lines 12–13). It is worth mentioning that the accepting run $R_k^{t^-}$ is revised iteratively and the condition $(q'_s, q'_{s+1}) \in R_{\text{pre}}^{t^-}$ or $R_{\text{suf}}^{t^-}$ is also checked iteratively (lines 2 and 9).

6.3. Optimality

Algorithm 8 offers a way to locally revise the invalid or unsafe plan, which however does not maintain the cost optimality as Algorithm 3. The general problem of computing and maintaining the shortest paths in a graph where the edges are inserted/deleted and edge weights are increased/decreased is referred to as the *dynamic shortest path problem* (DSPP) of Chan and Yang (2009) and Misra and Oommen (2005). As also pointed out in both papers, it is inefficient to re-compute the shortest path "from scratch" using the well-known static solution such as Dijkstra each time a topology change occurs in the graph. Moreover, the DSPP algorithms proposed there require much more complicated mechanism and sometimes not more efficient than calculating everything from scratch, especially when multiple changes occur simultaneously.

Algorithm 9. Knowledge-transfer-based motion planning scheme for each agent $k \in \mathcal{K}$.

```
    Input: $\mathcal{T}_k^0$, $\tilde{\mathcal{A}}_{\varphi_k}$
    Output: $R_k^t$, $\tau_k^t$, $\Upsilon_k^t$, $T_k$
 1  if $t = 0$ then
 2      $R_{\text{opt},k}^0$ = OptPlan($\tilde{\mathcal{A}}_{p,k}^0$)                      // Algorithm 3
 3  send Request$_{k,g}^t$
 4  check Reply$_{h,k}^t$ and Request$_{g,k}^{t_0}$
 5  read Sense$_k^t$ from sensors
 6  send Reply$_{k,g}^t$ = TranKnow(Sense$_k^t$, Request$_{g,k}^{t_0}$)          // Algorithm 4
 7  [$\mathcal{T}_k^{t^+}$, $\tilde{\Pi}_k^t$] = UpdKnow($\mathcal{T}_k^t$, Sense$_k^t$, Reply$_{g,k}^t$)       // Algorithm 5
 8  $\tilde{\mathcal{A}}_{p,k}^{t^+}$ = UpdProd($\tilde{\mathcal{A}}_{p,k}^{t^-}$, $\tilde{\Pi}_k^t$, $E$, $E_\neg$)                // Algorithm 6
 9  [$\aleph_k^t$, $\Xi_k^t$] = ValidRun($\tilde{\mathcal{A}}_{p,k}^{t^+}$, $R_k^t$, $\tilde{\Pi}_k^t$, $E_\neg$)           // Algorithm 7
10  $\Upsilon_k^{t^+} = \Upsilon_k^t + |\tilde{\Pi}_k^t| + |E| + |E_\neg|$
11  if $\Upsilon_k \geq N_k^{\text{call}}$ or $t - T_k \geq T_k^{\text{call}}$ then
12      $R_k^{t^+}$ = OptPlan($\tilde{\mathcal{A}}_{p,k}^{t^+}$)                            // Algorithm 3
13      $\Upsilon_k^{t^+} = 0, T_k = t$
14  else
15      $R_k^{t^+}$ = Revise($\tilde{\mathcal{A}}_{p,k}^{t^+}$, $R_k^t$, $\aleph_k^t$, $\Xi_k^t$)                 // Algorithm 8
16  $R_k^t = R_k^{t^+}$, $\tau_k^t = R_k^t|_{\Pi_k^{t^+}}$, $\Upsilon_k^t = \Upsilon_k^{t^+}$, $\mathcal{T}_k^t = \mathcal{T}_k^{t^+}$
```

Thus, in this paper, we propose the following event-based criterion (Heemels et al., 2012) to ensure the optimality check. Denote by Υ_k^t the accumulated number of number of changes in $\widetilde{\mathcal{A}}_{p,k}^t$ at time t. Here $\Upsilon_k^{t^+} = \Upsilon_k^{t^-} + |\tilde{\Pi}_k^t| + |E| + |E_\neg|$ and $\Upsilon_k^0 = 0$. Denote by T_k the last time instant when the optimal planner Algorithm 3 is called. Let the thresholds $N_k^{\text{call}}, T_k^{\text{call}} \geq 0$ be chosen freely by each agent. Then whenever at least one of the following conditions holds: (1) $\Upsilon_k^t \geq N_k^{\text{call}}$; (2) $t - T_k \geq T_k^{\text{call}}$, Algorithm 3 is called with respect the latest $\widetilde{\mathcal{A}}_{p,k}^t$ to derive the optimal motion plan $\tau_{\text{opt},k}^t$, but using agent k's current position as the initial state in $\mathcal{T}_k^t$. Then $\Upsilon_k^{t^+}$ is reset to zero and T_k is set to the current time.

7. Overall architecture

The overall architecture is summarized in Algorithm 9. When the system starts, each agent synthesizes its own initial motion and task plan using Algorithm 3. It sends requests to neighbouring agents that it has not requested before. 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 and product automaton. At last, it decides whether the revising algorithm or the optimal planner should be called based on the triggering condition.

Theorem 4. *For each agent k at any time $t \geq 0$, its motion plan τ_k^t derived by Algorithm 9 is always valid and safe for $\mathcal{T}_k^t$ and φ_k. Moreover, for any $t' \geq 0$, there exists time $t \in [t', t' + T_k^{call}]$ such that τ_k^t is the optimal motion plan.*

Proof. Whenever Algorithm 3 is called with respect to the latest $\widetilde{\mathcal{A}}_{p,k}^t$, it generates an optimal accepting run $R_{\text{opt},k}^t$. When Algorithm 3 is not called, Algorithm 8 revises the potentially falsified R_k^t such that R_k^t is an accepting run for $\widetilde{\mathcal{A}}_{p,k}^t$. In both cases, it is guaranteed that an accepting run for $\widetilde{\mathcal{A}}_{p,k}^t$ will be found if there exists one. As a result, R_k^t is always an accepting run for $\widetilde{\mathcal{A}}_{p,k}^t$, thus the corresponding motion plan τ_k^t is always valid and safe. Moreover, τ_k^t is the optimal motion plan for $\mathcal{T}_k^t$ and φ_k whenever Algorithm 3 is called. Due to the triggering condition (line 10 of Algorithm 9), Algorithm 3 is called at least once within any time period with length T_k^{call}, which completes the proof. □

The low-level implementation of the discrete plan has the important aspect of safe region-to-region navigation. As pointed out earlier, it could potentially be combined with many existing partition and motion planning techniques, such as vector-field-based (Desai et al., 1998), navigation function for linear (Belta et al., 2005) or non-holonomic dynamic models (Lindemann et al., 2006).

The correctness of the proposed solutions follows from Theorem 4 and the correctness of Dijkstra's shortest path algorithm. Let $|\widetilde{\mathcal{A}}_{p,k}^t|$ be the size of $\widetilde{\mathcal{A}}_{p,k}^t$ from Definition 4. Algorithm 3 runs in $\mathcal{O}(|\widetilde{\mathcal{A}}_{p,k}^t| \cdot \log |\widetilde{\mathcal{A}}_{p,k}^t| \cdot (|Q_0'| + |\mathcal{F}'|))$. Algorithms 5 and 6 has the complexity linear to the size of **Sense**$_k^t$. Algorithm 8 has the complexity linear to the length of $R_k^{t^-}$.

8. Case study

In the following case study, we apply the framework to a team of 15 autonomous robots: five of them are aerial vehicles that repetitively surveil over base stations; the rest are

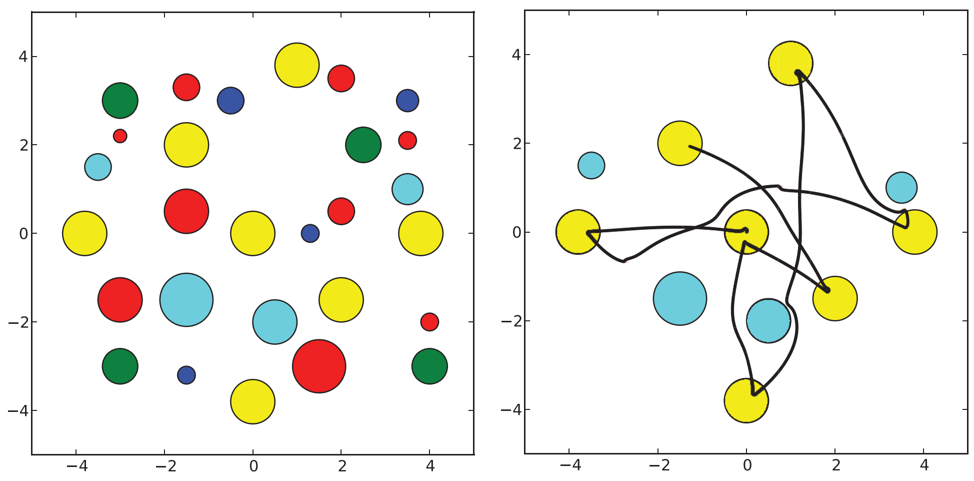

Fig. 3. Left: The actual workspace model as described in Section 8.1. Right: The final motion plan for aerial vehicles, which corresponds to the final optimal plan in Figure 5. It satisfies both $\varphi^{\text{hard}}_{\texttt{Ar\_i}}$ and $\varphi^{\text{soft}}_{\texttt{Ar\_i}}$, since all base stations are surveiled and non-fly areas are avoided.

ground vehicles that collect food and water resources to supply the base stations. All algorithms and modules are implemented in Python 2.7. All simulations are carried out on a desktop computer (3.06 GHz Duo CPU and 8 GB of RAM).

8.1. Workspace and agent description

As shown Figure 3, the workspace we consider is a $10 \times 10\text{m}^2$ square which is approximated by the circle $\mathcal{B}_5([0, 0])$. There are 7 base stations with size $1 \times 1\text{m}^2$ (in yellow, denoted by ‘`b1`’,…,‘`b7`’). In addition, there are numerous no-fly zones (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, Ar_2,…, Ar_5`) start randomly from one of the base stations. For aerial vehicles, ‘`b1`’,…, ‘`b7`’, ‘`nfly`’, ‘`water`’, ‘`food`’ are their known propositions, which does not include ‘`obs`’. It means that aerial vehicles cannot detect obstacles on the ground. Initially, they know the location and size of some base stations and some no-fly zones, but not the water and food resources. They have an average speed 0.1 m/s and a sensing radius of 2 m 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

$$\begin{aligned}\varphi^{\text{hard}}_{\texttt{Ar\_i}} &= (\Box\neg\,\texttt{nfly}) \wedge (\Box\Diamond(\varphi_{\text{one}})),\\ \varphi^{\text{soft}}_{\texttt{Ar\_i}} &= (\Box(\Diamond\texttt{b1} \wedge \Diamond\texttt{b2} \wedge \ldots \wedge \Diamond\texttt{b7}))\end{aligned} \tag{22}$$

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

The other 10 robots are ground vehicles: five of them (denoted by `Gf_1, Gf_2,…, Gf_5`) collect food and the rest (denoted by `Gw_1, Gw_2, …, Gw_5`) for water, to supply the base stations. For ground vehicles, ‘`b1`’, …, ‘`b7`’, ‘`obs`’, ‘`water`’, ‘`food`’ are known propositions, which does not include ‘`nfly`’. It means that ground vehicles cannot recognize ‘`nfly`’ zones. Initially, they start randomly from one base station and only knows the location and size of one water (or food) resource, but not the obstacles and other base stations. They have an average speed 0.05 m/s and a sensing radius of 1 m 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 specifications for `Gw_i` are given by

$$\begin{aligned}\varphi^{\text{hard}}_{\texttt{Gw\_i}} &= (\Box\Diamond\neg\texttt{obs}) \wedge \varphi_{\text{order}}\\ \varphi^{\text{soft}}_{\texttt{Gw\_i}} &= (\Box(\Diamond\texttt{b1} \wedge \Diamond\texttt{b2} \wedge \ldots \wedge \Diamond\texttt{b7}))\end{aligned} \tag{23}$$

where $\varphi_{\texttt{order}} \triangleq (\quad \Box\Diamond\texttt{water}) \wedge (\quad \Box(\texttt{water} \Rightarrow \bigcirc\, (\neg\, \texttt{water}\ \texttt{U}(\varphi_{\texttt{one}}))) \wedge (\ \Box((\varphi_{\texttt{one}}) \Rightarrow \bigcirc\, (\neg(\varphi_{\texttt{one}})\ \texttt{U}\ \texttt{water})),$ which says that water must be fetched and supplied to at least one base station infinitely often. Here $\varphi^{\text{soft}}_{\texttt{Gw\_i}}$ is the same as for `Ar_i`, requiring that all base stations be supplied infinitely often. The NBA associated with $\varphi^{\text{hard}}_{\texttt{Gw\_i}}$ have 10 states and 30 edges by Gastin and Oddoux (2001), while the one for $\varphi^{\text{soft}}_{\texttt{Gw\_i}}$ has 8 states and 43 edges. The soft and hard specifications $\varphi^{\text{hard}}_{\texttt{Gf\_i}}$ and $\varphi^{\text{soft}}_{\texttt{Gf\_i}}$ for `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 and size of all base stations. However, since ‘`b1`’, ‘`b2`’…, ‘`b7`’ belong to all vehicles, meaning that any relevant information can be shared within the

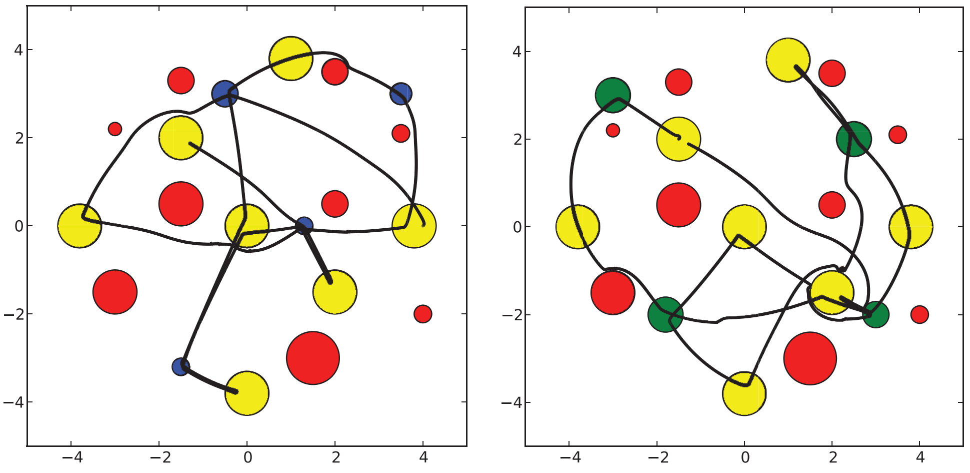

Fig. 4. The final motion plan for agents Gw_i (left) and for agents Gf_i (right). It can be seen that water or food are fetched before any base station and all base stations are supplied infinitely often, while at the same time obstacle avoidance is ensured.

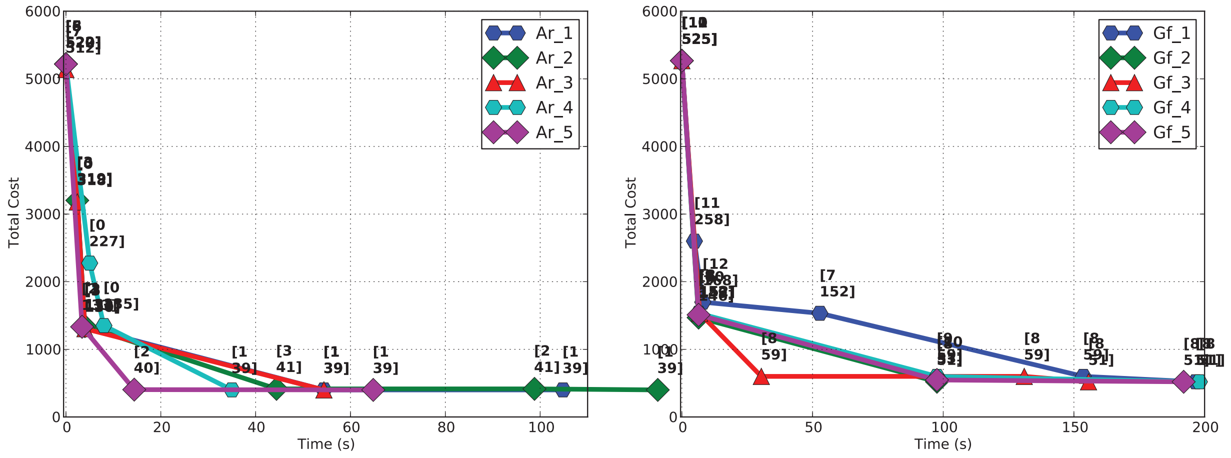

Fig. 5. Each optimal plan is labelled by the cost of its prefix and suffix. Left: For Ar_i, the initial plan has a total cost around 5220, with prefix and suffix costs being [5, 521], when it only knows the location of two base stations. Its final plan has a total cost 391, with prefix and suffix costs being [1, 39] (as shown in Figure 3). Right: For Gf_i, The initial plan has a total cost around 5268, with prefix and suffix costs being [12, 526], when it only knows the location of one base station and one food resource. Its final plan has a total cost 610, with prefix and suffix costs being [3, 60] (as shown in Figure 4).

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 topology, a dynamic topology where each agent has a communication radius (set to 5 m for all agents) as introduced in Section 5.1.

8.2. 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_k^{call} and T_k^{call} to be (3,60s) for aerial vehicles and (3,100s) for ground vehicles. The system was simulated for 600*s* and it took 200*s* for the workspace to be fully discovered by all agents. Figures 3 and 4 show the trajectories corresponding to the suffix part of the optimal run, for the three groups Ar_i, Gf_i, Gw_i. It can be seen that both soft and hard specifications are fulfilled by all agents.

Figures 5 and 6 shows how the optimal plans of Ar_i, Gw_1 and Gf_i evolve with time. In particular, under the proposed scheme, the total cost of the optimal plans for each agent decreases gradually until its knowledge of the workspace is complete. It can be seen that each time the optimal planner could incorporates the update knowledge of the workspace into the optimal plan, which satisfies the soft task specification more. Figure 6 illustrates the number of messages received by each agent under the dynamic communication topology (communication radius 5 m).

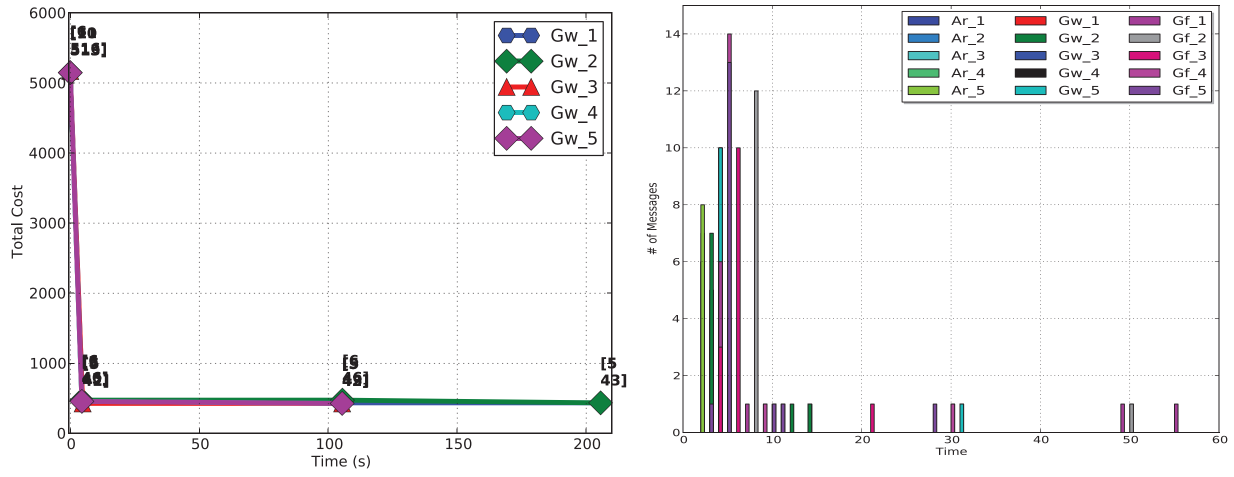

Fig. 6. Left: The total cost of the optimal plan of agents `Gw_i` evolves with time. The initial plan has a total cost around 5147, with prefix and suffix costs being [11, 513], when it only knows the location of one base station and one water resource. Its final plan has a total cost 456, with prefix and suffix costs being [6, 45] (as shown in Figure 4). Right: It shows the number of messages (including sensing and reply messages) received by each agent under the dynamic communication topology.

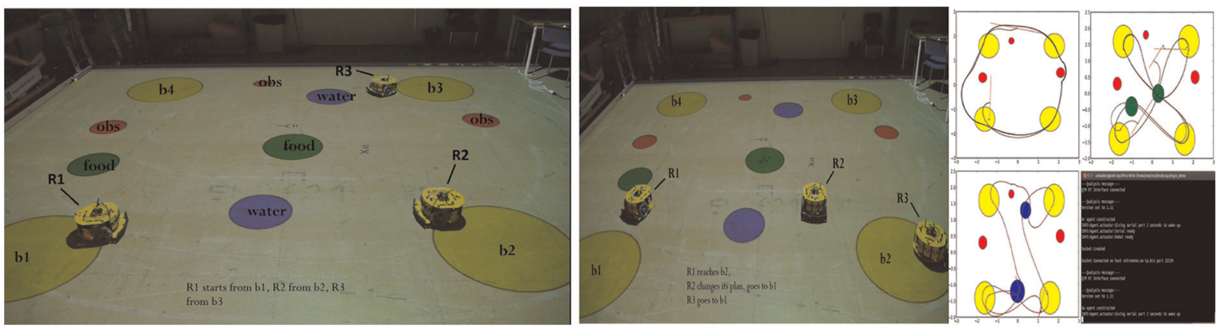
b4
obs
R3
water
b3
obs
obs
food
food
R1
R2
water
b1
b2
R1 starts from b1, R2 from b2, R3 from b3
R1 reaches b2,
R2 changes its plan, goes to b1
R3 goes to b1

Fig. 7. Left: Workspace used in the experiment, with three Nexus vehicles. Right: Recorded panels including video streams, vehicles' trajectories and the log window.

Compared with the synchronized solution that requires information exchange at each time step (Ding et al., 2011; Karaman and Frazzoli, 2008), the messages are only sent and received following the subscriber-publisher scheme introduced in Section 5.1.

8.3. 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 Figure 7. Similar task specifications are designed as in Section 8. One vehicle $\mathcal{R}_1$ has the surveillance task over four base stations and avoids all obstacles, as specified by (22). One vehicle $\mathcal{R}_2$ needs to visit the blue regions and all base stations infinitely often in an interleaved order while avoiding all obstacles, as specified by (23). 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 $6 \times 6\text{m}^2$ and modelled by the circle $\mathcal{B}_3([0,\,0])$. Four base stations are located at four corners; two blue regions, two green regions, three obstacles regions are scattered in the workspace (as shown in Figure 7). The inter-agent collision avoidance is tackled locally by a distributed reactive navigation controller. Initially $\mathcal{R}_1$ knows the location of two 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. Figure 7 shows the main panels recorded during the experiment, including the video stream, each vehicle's trajectory and its local knowledge, and the log window displaying the messages exchanged among the vehicles and the updates of each vehicle's plan. Figure 8 shows four snapshots during the experiments when one of the agents receives knowledge updates and adapts its motion and task

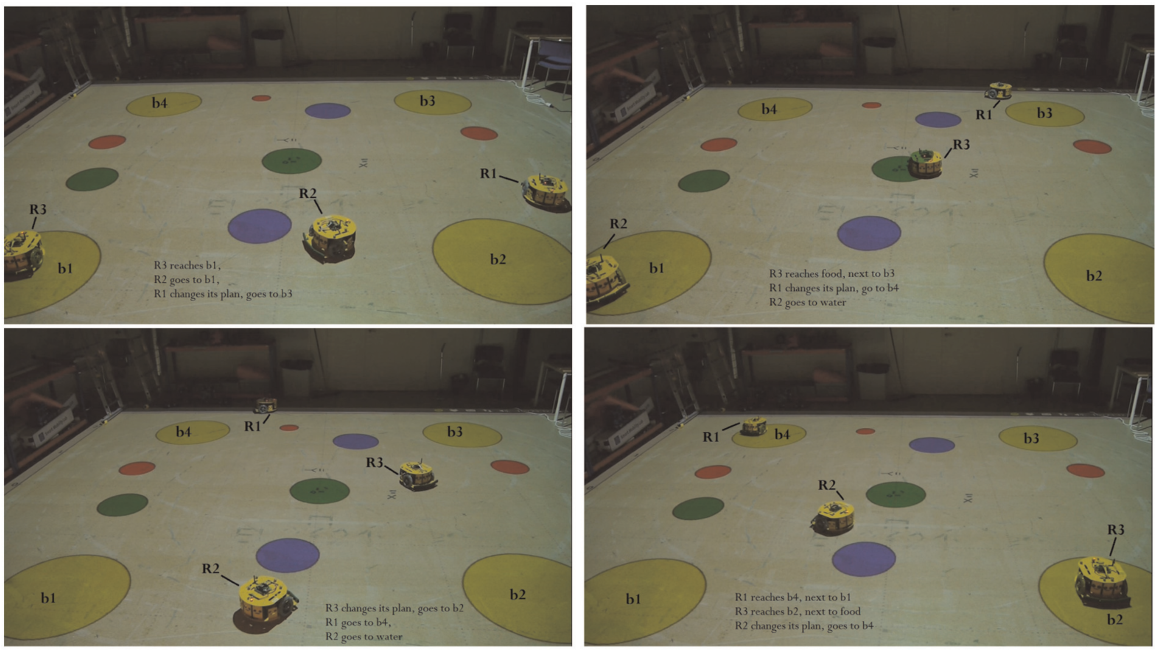
b4
b3
R1
R2
R3
b1
b2
R3 reaches b1,
R2 goes to b1,
R1 changes its plan, goes to b3
R3 reaches food, next to b3
R1 changes its plan, go to b4
R2 goes to water
R3 changes its plan, goes to b2
R1 goes to b4,
R2 goes to water
R1 reaches b4, next to b1
R3 reaches b2, next to food
R2 changes its plan, goes to b4

Fig. 8. Upper-left: Agent $\mathcal{R}_1$ receives knowledge about b_3 from $\mathcal{R}_2$ and changes its plan by going to b_3 instead of going back to b_2. Upper-right: Agent $\mathcal{R}_1$ receives knowledge about b_4 from $\mathcal{R}_3$ and changes its plan by going to b_4 instead of going to b_1. Bottom-left: Agent $\mathcal{R}_3$ receives knowledge about b_2 from $\mathcal{R}_2$ and changes its plan by going to b_2 instead of going back to b_3. Bottom-right: Agent $\mathcal{R}_2$ receives knowledge about b_4 from $\mathcal{R}_3$ and changes its plan by going to b_4 instead of going to b_1.

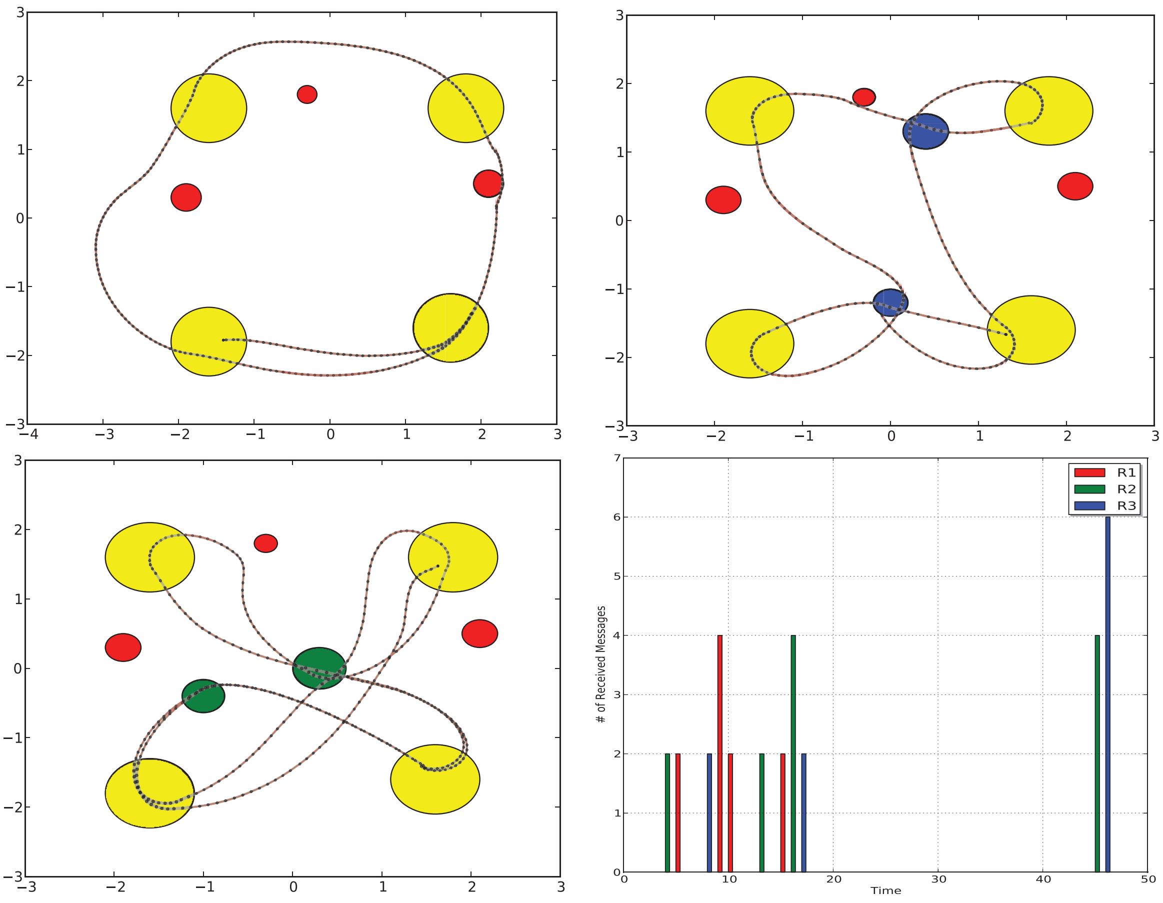

Fig. 9. Upper-left: The suffix part of agent $\mathcal{R}_1$'s trajectory satisfying (22). Upper-right: The suffix part of agent $\mathcal{R}_2$'s trajectory satisfying (23). Bottom-left: The suffix part of agent $\mathcal{R}_3$'s trajectory satisfying similar tasks for `Gf_i`. Bottom-right: Number of messages received by each agent during the process.

plan. After all agents have the same knowledge about the workspace which reflects the actual workspace, each agent's task plan would remain static and coincide with the optimal plan given the complete knowledge, as shown in Figure 9.

9. Conclusion and discussion

We have proposed a knowledge transfer scheme for cooperative motion and task planning of multi-agent systems under local LTL specifications. The workspace is assumed to be partially known. The specifications consist of hard and soft constraints, which might be infeasible initially. Algorithms are provided for the agents to update and exchange their knowledge about the workspace, based on which they revise and improve their plans. Future work could include the consideration of dependent task specifications.

Acknowledgements

We thank Matteo Vanin at KTH Royal Institute of Technology for his help on the implementation. The authors are grateful for the constructive comments by the anonymous reviewers.

Funding

This work was supported by the Swedish Research Council (VR) and EU STREP RECONFIG: FP7-ICT-2011-9-600825.

References

Baier C and Katoen J-P (2008) *Principles of model checking*. Cambridge, MA: The MIT Press.

Belta C, Bicchi A, Egerstedt M, Frazzoli E, Klavins E and Pappas GJ (2007) Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine* 14: 61–71.

Belta C, Isler V and Pappas GJ (2005) Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics* 21(5): 864–874.

Bhatia A, Kavraki LE and Vardi MY (2010) Sampling-based motion planning with temporal goals. In: *IEEE international conference on robotics and automation (ICRA)*, Anchorage, AK.

Bhatia A, Maly MR, Kavraki LE and Vardi MY (2011) Motion planning with complex goals. *IEEE Robotics and Automation Magazine* 18(3): 55–64.

Boyd S and Vandenberghe L (2009) *Convex Optimization*. Cambridge: Cambridge University Press.

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

Chen Y, Xu CD, Stefanescu A and Belta C (2012) Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics* 28(1): 158–171.

Clarke EM, Grumberg O and Peled DA (1999) *Model Checking*. Cambridge, MA: The MIT Press.

Desai J, Ostrowski J and Kumar V (1998) Controlling formations of multiple mobile robots. In: *IEEE international conference on robotics and automation (ICRA)*, pp. 2864–2869.

Dimarogonas DV and Kyriakopoulos KJ (2007) Decentralized navigation functions for multiple robotic agents with limited sensing capabilities. *Journal of Intelligent and Robotic Systems* 48(3): 411–433.

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

Fainekos GE, Girard A, Kress-Gazit H and Pappas GJ (2009) Temporal logic motion planning for dynamic mobile robots. *Automatica* 45(2): 343–352.

Fainekos GE, Loizou SG and Pappas GJ (2006) Translating temporal logic to controller specifications. In: *IEEE conference on decision and control (CDC)*, San Diego, CA.

Filippidis IF, Dimarogonas DV and Kyriakopoulos KJ (2012) Decentralized multi-agent control from local LTL specifications. In: *IEEE conference on decision and control (CDC)*, Maui, HI.

Forney G Jr (1966) Generalized minimum distance decoding. *IEEE Transactions on Information Theory* 12(2): 125–131.

Gastin P and Oddoux D (2001) Fast LTL to Büchi automaton translation. In: *International conference on computer aided verification (CAV'01)*, Paris, France.

Guo M, Johansson KH and Dimarogonas DV (2013) Revising motion planning under linear temporal logic specifications in partially known workspaces. In: *IEEE international conference on robotics and automation (ICRA)*, Karlsruhe, Germany.

Guo M and Dimarogonas DV (2013) Reconfiguration in motion planning of single- and multi-agent systems under infeasible local LTL specifications. In: *IEEE conference on decision and control (CDC)*, Florence, Italy.

Guo M and Dimarogonas DV (2014) Distributed plan reconfiguration via knowledge transfer in multi-agent systems under local LTL specifications. In: *IEEE international conference on robotics and automation (ICRA)*, Hongkong, China.

Heemels WPMH, Johansson KH and Tabuada P (2012) An introduction to event-triggered and self-triggered control. In: *IEEE conference on decision and control (CDC)*, Maui, HI.

Johnson B and Kress-Gazit H (2011) Probabilistic analysis of correctness of high-Level robot behavior with sensor error. In: *Proceedings of robotics: science and systems*, Los Angeles, CA.

Karaman S and Frazzoli E (2008) Vehicle routing with linear temporal logic specifications: applications to multi-UAV mission planning. In: *Navigation, and control conference in AIAA Guidance*, Minnesota, MN.

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

Kim K, Fainekos GE and Sankaranarayanan S (2012) On the revision problem of specification automaton. In: *IEEE international conference on robotics and automation (ICRA)*, pp. 5171–5176.

Kloetzer M and Belta C (2010) Automatic deployment of distributed teams of robots from temporal logic specifications. *IEEE Transactions on Robotics* 26(1): 48–61.

Kloetzer M, Ding XC and Belta C (2011) Multi-robot deployment from LTL specifications with reduced communication. In: *IEEE conference on decision and control (CDC)*, Florida, FL.

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

Kress-Gazit H, Wongpiromsarn T and Topcu U (2011) Correct, reactive robot control from abstraction and temporal logic specifications. *IEEE Robotics and Automation Magazine* 18(3): 65–74.

LaValle SM (2006) *Planning Algorithms*. Cambridge: Cambridge University Press.

Lindemann SR, Hussein II and LaValle SM (2006) Real time feedback control for nonholonomic mobile robots with obstacles. In: *IEEE conference on decision and control (CDC)*, San Diego, CA.

Livingston SC, Murray RM and Burdick JW (2012) Backtracking temporal logic synthesis for uncertain environments. In: *IEEE international conference on robotics and automation (ICRA)*, Minnesota, MN.

Loizou SG and Jadbabaie A (2006) Density functions for navigation function based systems. In: *IEEE conference on decision and control (CDC)*, San Diego, CA.

Loizou SG and Kyriakopoulos KJ (2003) Closed loop navigation for multiple non-holonomic vehicles. In: *IEEE international conference on robotics and automation*, vol. 3, pp. 4240–4245.

Loizou SG and Kyriakopoulos KJ (2004) Automatic synthesis of multi-agent motion tasks based on LTL specifications. In: *IEEE conference on decision and control (CDC)*, Paradise Island, Bahamas.

Maly MR, Lahijanian M, Kavraki LE, Kress-Gazit H and Vardi MY (2013) Iterative temporal motion planning for hybrid systems in partially unknown environments. In: *Proceedings of the 16th international conference on hybrid systems: computation and control*. New York: ACM Press, pp. 353–362.

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

Oikonomopoulos AS, Loizou SG and Kyriakopoulos KJ (2009) Coordination of multiple non-holonomic agents with input constraints. In: *IEEE international conference on robotics and automation (ICRA)*, Kobe, Japan.

Raman V and Kress-Gazit H (2011) Analyzing unsynthesizable specifications for high-Level robot behavior. In: *Proceedings of computer aided verification*, Snowbird, UT.

Smith SL, Tumova J, Belta C and Rus D (2011) Optimal path planning for surveillance with temporal logic constraints. *The International Journal of Robotics Research* 30(14): 1695–1708.

Topcu U, Ozay N, Liu J and Murray RM (2012) On synthesizing robust discrete controllers under modeling uncertainty. In: *Hybrid system: computation and control*, Beijing, China, pp. 85–94.

Tumova J and Dimarogonas DV (2014) A receding horizon approach to multi-agent planning from local LTL specifications. In: *American control conference (ACC)*, Portland, OR.

Tumova J, Hall GC, Karaman S, Frazzoli E and Rus D (2013) Least-violating control strategy synthesis with safety rules. In: *Proceedings of the 16th international conference on hybrid systems: computation and control*. New York: ACM Press.

Ulusoy A, Smith SL, Ding XC, Belta C and Rus D (2013) Optimality and robustness in multi-Robot path planning with temporal logic constraints. *The International Journal of Robotics Research* 32(8): 889–911.

Wolff E, Topcu U and Murray RM (2012) Robust control of uncertain markov decision processes with temporal logic specifications. In: *IEEE conference on decision and control (CDC)*, Maui, HI.

Wongpiromsarn T, Topcu U and Murray R (2012) Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control* 57(11): 2817–2830.